Déduction automatique appliquée à l'analyse et la vérification de systèmes infinis

Laurent Vigneron

Nancy-Université
UFR Mathématiques Informatique

École Doctorale IAEM Lorraine

Département de Formation Doctorale en informatique

Déduction automatique appliquée à l'analyse et la vérification de systèmes infinis

Habilitation

présentée et soutenue publiquement le 14/11/2011

pour l'obtention d'une

Habilitation à Diriger des Recherches de l'Université Nancy 2

(spécialité informatique)

par

Laurent Vigneron

Composition du jury

Président :	Laurent Fribourg	(CNRS, LSV - ENS Cachan)
Rapporteurs :	Bernhard Gramlich	(TU Wien)
	Christopher Lynch	(Clarkson University)
	Ralf Treinen	(Université Paris Diderot)
Examinateurs :	Paliath Narendran	(University at Albany)
	Michaël Rusinowitch	(INRIA Nancy Grand-Est)
	Peter Ryan	(University of Luxembourg)
	Jeanine Souquières	(Université Nancy 2)

Loria Laboratoire lorrain de recherche en informatique et ses applications

Laboratoire Lorrain
de Recherche en Informatique
et ses Applications
UMR 7503

Mis en page avec la classe thloria.

Remerciements

Je tiens tout d'abord à remercier les membres du jury :
Laurent Fribourg,
qui a accepté de présider ce jury ;
Bernhard Gramlich, Christopher Lynch et Ralf Treinen,
qui ont accepté la lourde tâche d'évaluer ce manuscrit dans des délais très serrés,
au risque de se fâcher avec leur famille en raison de week-ends (trop) studieux ;
Paliath Narendran et Peter Ryan,
qui m'ont fait l'amitié d'accepter de participer à ce jury ;
Jeanine Souquières,
qui a été un moteur infatigable pour me pousser à enfin passer cette HDR ;
et bien sûr Michaël Rusinowitch,
toujours présent pour m'aider à ne pas perdre de vue la recherche,
malgré mes lourdes charges d'enseignement et d'administration.

Je tiens également à remercier
Yannick Chevalier, Judson Santos Santiago et Najah Chridi,
« mes » premiers Doctorants,
pour le plaisir que j'ai eu à travailler avec eux
et à les aider à tracer leur chemin dans le monde de la recherche.

Je suis également profondément reconnaissant à tous mes co-auteurs,
mais aussi aux membres des équipes Eureca, Protheo et Cassis dont j'ai fait parti,
car un travail d'équipe est toujours très enrichissant et en général source des plus belles idées.

Enfin, je remercie mes collègues
de l'UFR Mathématiques et Informatique de l'Université Nancy 2,
pour la confiance qu'ils m'accordent chaque jour dans la gestion de formations,
mais aussi pour l'excellente ambiance dans laquelle nous travaillons.

Et merci à mes joyeux camarades
qui, bien qu'étant habilités avant moi, ont continué à accepter ma compagnie... ☺

Dédicace spéciale
à tous ceux qui m'ont aidé à m'épanouir dans ce que certains appellent mon « second métier »,
et que je qualifierais plutôt de troisième poumon,
et en particulier à la famille d'Amezule (Gadget, Fiola, Orient, Rivage, Unaken) et à Joli Môme,
et bien sûr à Aurélie et Christine.

À Papa.

Table des matières

Introduction

Chaque jour, nous utilisons des outils pour communiquer, pour travailler, parfois sans en connaître les auteurs, et souvent sans avis sur leur qualité. Nous avons tous dû faire face à des bugs, nous faisant perdre du temps et monter le taux d'adrénaline. En tant qu'informaticiens, nous savons que dans tout logiciel de taille importante des bugs sont pratiquement inévitables. Cependant, tant qu'il s'agit de problèmes détectables par les utilisateurs, le mal est moindre. Par exemple, lorsque dans une gare, un poste d'aiguillage entièrement informatisé « tombe en panne », des centaines d'usagers sont gênés, et les conséquences financières pour la société de chemin de fer ne sont pas anodines. Mais le cas est beaucoup plus grave si le problème n'est pas détecté et qu'il engendre un accident.

La vérification de logiciels est donc un besoin incontestable, déjà au cœur des préoccupations de la recherche scientifique depuis bien longtemps, et appliquée à plus ou moins grande échelle dans l'industrie. Mais les verrous scientifiques restent nombreux, et bien sûr un grand nombre de cas sont intraitables.

La situation est encore plus complexe lorsqu'il ne s'agit pas de trouver des failles dans un logiciel statique, mais de considérer un environnement totalement dynamique pouvant librement interagir avec le logiciel à vérifier. C'est le cas des communications devant transiter sur un réseau ouvert comme Internet.

Ce mémoire retrace les activités de recherche auxquelles je me suis consacré depuis... vingt ans, et plus particulièrement celles abordées depuis ma nomination comme Maître de Conférences à l'Université Nancy 2 en 1997.

Les paragraphes ci-dessous décrivent les thématiques étudiées, en commençant par la conception de techniques de déduction automatique, domaine par lequel j'ai mis un pied dans le monde de la recherche. Ces travaux, assez théoriques, m'ont ouvert les yeux sur les nombreux apports que la recherche fondamentale peut offrir aux utilisateurs finaux, directs ou indirects, de logiciels. Depuis je n'ai eu de cesse de trouver des applications aux techniques développées, même si certaines applications restent très scientifiques. J'ai ainsi participé à la conception de techniques efficaces de réécriture, indispensables pour améliorer l'efficacité de démonstrateurs ; j'ai également participé à l'analyse de propriétés de logiques non classiques utilisées en fouille de données. Et depuis plus de dix ans, j'ai participé à la conception de méthodes d'analyse de protocoles de communication, apportant ainsi un petit grain de sable dans la machine insécuritaire de la communication sur les réseaux.

Ces applications ont toujours été accompagnées de développement d'outils, car c'est le meilleur moyen de valider l'intérêt d'une méthode et d'en convaincre les utilisateurs potentiels.

Ce mémoire ne se veut pas être d'une extrême rigueur scientifique. De nombreux articles apportent théorèmes et démonstrations, et le lecteur avisé pourra s'y référer. L'objectif ici est en fait de placer les résultats obtenus dans leur contexte et d'en montrer l'intérêt scientifique et

pratique.

Déduction automatique

Le thème de la déduction automatique est né historiquement de la tentative de mécanisation des mathématiques, et donc d'automatiser la production de théorèmes. Les premiers travaux issus de la découverte par Alan Robinson du principe de la résolution sont à la base des concepts et des outils de la programmation logique. Les années 1980 ont consacré l'automatisation du raisonnement équationnel, permettant la manipulation mécanique des spécifications algébriques.

Dans la continuité de cette évolution, je m'intéresse à la recherche de stratégies correctes, complètes et efficaces de déduction automatique en logique du premier ordre. Il s'agit de concevoir des méthodes qui permettent de trouver si une propriété est vrai ou fausse dans un environnement donné. Dans le cas général, ce problème est extrêmement difficile, puisqu'il ne peut être résolu en un temps fini. Cela n'empêche pas cependant de définir des méthodes qui s'avèrent efficaces dans un grand nombre de cas. Ainsi, j'ai conçu (et implanté dans le logiciel daTac) des règles de déduction qui permettent de dissocier de la description d'un environnement une partie de ses propriétés, et de les traiter à part, diminuant ainsi la difficulté du travail principal. J'ai également montré comment le traitement de ces propriétés spécifiques peut être efficacement effectué, grâce à une représentation sous forme de contraintes (Chapitre 1).

Je me suis aussi intéressé à l'une des briques de base de la déduction automatique, garante de son efficacité : la réécriture. Ce principe consiste à transformer des informations pour les simplifier, grâce aux connaissances déjà acquises. En collaboration avec les Professeurs IV Ramakrishnan et Leo Bachmair de l'Université de Stony Brook (NY, USA), puis Ashish Tiwari de l'Université de Stanford (CA, USA), nous avons mis au point une technique qui, à partir d'un ensemble d'équations représentant des propriétés, permet de construire l'ensemble des propriétés déductibles des propriétés initiales, groupées par classes d'équivalence. Cette technique est de loin la plus efficace jamais construite, et son mode de réalisation permet très facilement de modéliser les techniques précédemment définies, et d'expliquer leurs avantages et inconvénients. J'ai implanté ce résultat dans un petit logiciel appelé AbstractCC (Chapitre 2).

Dans un tout autre contexte, en collaboration avec le Professeur Anita Wasilewska de l'Université de Stony Brook (NY, USA), j'ai appliqué les résultats obtenus en déduction automatique, à des problèmes de classification d'informations imprécises, incertaines ou incomplètes, rencontrés en fouille de données. Nous avons ainsi pu construire un modèle de logiques non classiques, et démontrer automatiquement de nombreuses propriétés de ce modèle, utilisant même une représentation graphique pour en montrer le fonctionnement (Chapitre 3).

Je reste toujours très impliqué dans le domaine de la déduction automatique, avec en particulier l'organisation à Nancy des deux premières éditions de l'école internationale sur la réécriture (ISR) en 2006 et 2007, l'organisation d'un workshop sur l'unification (UNIF) au Japon en 2005, et l'organisation d'un colloque sur la démonstration en logique du premier ordre (FTP) en Espagne en 2003.

Protocoles de sécurité

La sécurisation des échanges d'informations sur un réseau (comme Internet) est un problème très important de nos jours. Ainsi, des protocoles sont chargés d'authentifier les acteurs et de protéger les informations échangées. Beaucoup s'avèrent partiellement défaillants. Certains permettent à un intrus, par exemple, de récupérer des informations confidentielles, de se faire

passer pour quelqu'un d'autre, ou de perturber un échange d'informations. Dans ce domaine, je participe au développement de méthodes et d'outils permettant de formaliser des protocoles et de simuler leur exécution pour essayer de trouver leurs failles ou de démontrer leur correction.

L'ensemble de ces travaux se base sur des méthodes que nous utilisons depuis très longtemps dans l'équipe : réécriture, unification, résolution de contraintes, théories équationnelles. Ce sont elles qui permettent de définir des techniques de vérification entièrement automatiques.

Notre première approche a été de définir un formalisme pour décrire les échanges de messages composant un protocole, et d'utiliser le logiciel daTac pour analyser des propriétés de confidentialité et d'authentification (plateformes Casrul et AVISS).

Vérification dans un cadre non borné

La recherche d'attaques sur des protocoles est vite indécidable ; il suffit de combiner plusieurs exécutions du protocole et de ne pas limiter la taille des messages échangés. Et même en fixant des bornes, la recherche d'attaques reste très complexe. Pour la thèse de Yannick Chevalier, nous avons défini une stratégie paresseuse pour l'intrus qui permet d'obtenir une méthode efficace de recherche d'attaques. Nous avons montré qu'il était possible de décider du secret pour un nombre non borné d'exécutions, en décomposant l'ensemble des participants au protocole en deux parties : ceux qui jouent une session officielle (limités), et ceux qui sont manipulés par l'intrus (non limités). D'autres résultats ont suivi sur la décidabilité et la complexité de divers modèles de l'intrus, par exemple pour considérer les propriétés d'opérateurs utilisés dans les algorithme de chiffrement des messages, tels que le ou exclusif et l'exponentielle.

Réalisation du logiciel AVISPA

J'ai activement participé à la réalisation du logiciel AVISPA, qui permet via une interface Web très simple, de spécifier et d'analyser des protocoles cryptographiques. Ce logiciel représente l'implantation des travaux réalisés lors du projet européen AVISPA. Le langage de spécification de protocoles défini dans ce projet est très expressif. Les spécification obtenues peuvent ainsi être vues comme des programmes à base de règles. La sémantique n'en est que plus claire, et bien que les transitions mêlent à la fois non déterminisme et informations temporelles, le logiciel reste facilement accessible à tout utilisateur non spécialiste.
Ce logiciel est le point de départ de nombreuses collaborations, comme avec France Télécom pour des protocoles de porte-monnaie électronique ou de vote électronique.
Il est très largement utilisé par des industriels, des enseignants et étudiants, et des chercheurs.

Vérification de propriétés évoluées

Le thème de la vérification de protocoles de sécurité est très vaste. Ainsi, pour la thèse de Judson Santos Santiago, nous nous sommes intéressés à la vérification de propriétés difficiles à formaliser et à vérifier, telles que la non répudiation et l'anonymat.

Pour la thèse de Najah Chridi, nous nous sommes intéressés à l'analyse de protocoles de groupe. Ils permettent à un ensemble de personnes de communiquer en toute sécurité sur un réseau avec ou sans fil, via des ordinateurs, téléphones ou PDA. La première difficulté est de gérer la taille non prédéfinie du groupe : nous modélisons le rôle de chaque participant et fixons le nombre de rôles effectifs en identifiant des comportements similaires. La deuxième difficulté est la modélisation des propriétés à vérifier : elles portent généralement sur la clef de chiffrement commune au groupe, devant évoluer avec la composition du groupe ; nous avons ainsi mis au point une gestion hiérarchisée des groupes et de leurs clefs ; nous avons également défini une

modélisation des caractéristiques et des propriétés de sécurité de ces protocoles afin d'identifier clairement les propriétés incriminées lors d'une attaque.

L'ensemble de ces travaux est décrit dans deux chapitres. Le premier se concentre sur les langages de spécification de protocoles, leur sémantique et les outils développés (Chapitre 4). Le second chapitre (Chapitre 5) décrit les principaux résultats théoriques obtenus pour l'analyse de protocoles (la plupart ayant été implantés dans des logiciels), mais aussi les différentes propriétés de sécurité étudiées ainsi que quelques études de cas.

Chapitre 1

Préliminaires pour être daTac

L'objectif de ce chapitre est de présenter mes travaux sur la déduction automatique, thème par lequel j'ai abordé la recherche, et qui forme le socle scientifique sur lequel repose la très grande majorité des travaux effectués depuis. J'y introduit les mécanismes de déduction utilisés, comme la résolution et la paramodulation, mais aussi les principales stratégies indispensables pour restreindre la taille de l'espace de recherche et éviter les redondances. Ces stratégies sont très variées :

- sélection d'éléments dans les clauses avec lesquels effectuer des déductions ;
- simplification de clauses par réécriture et élimination de clauses redondantes ;
- ajout de contraintes pour mieux contrôler les termes introduits à la suite d'unifications.

Ces techniques de déduction et ces stratégies ont été définies pour le raisonnement modulo une théorie équationnelle, dans la lignée des travaux initiés par Plotkin dans [21]. Les principaux résultats que j'ai obtenus sont brièvement mentionnés. Le logiciel daTac appliquant ces résultats aux théories commutatives et associatives-commutatives est également brièvement présenté.

L'article référence de ce chapitre est celui présenté à la conférence internationale CSL (Computer Science Logic) en 1995 [30]. Il fait la synthèse de l'ensemble de ces travaux, en mettant l'accent sur une stratégie de sélection appelée *positive*.

1.1 Notions de base

Définissons les principales notions utilisées dans ce chapitre, mais aussi les chapitres suivants, basées sur les notations standard et les définitions sur la réécriture et l'unification données dans [8, 11].

Soit un ensemble $\mathcal{F}$ d'opérateurs fonctionnels (f, g, ...), un ensemble $\mathcal{P}$ d'opérateur s de prédicats (p, q, ...), et un ensemble $\mathcal{X}$ de variables (x, y, ...). Les constantes dans $\mathcal{F}$ seront notées a, b, c, ...

Un *terme* est une formule composée d'éléments de $\mathcal{F}$ et $\mathcal{X}$. Un terme est dit *clos* s'il ne contient pas de variables. Les termes sont notés s, t, ... Un *atome* (A, B, ...) est une formule $p(t_1, \ldots, t_n)$ où $p \in \mathcal{P}$, et les t_i sont des termes. Une *équation* est un atome dont le prédicat est l'opérateur est $\approx$. Dans la suite, l'opérateur équationnel étant commutatif, nous ne distinguerons pas les atomes $(s \approx t)$ et $(t \approx s)$. Un *littéral* (L) est un atome positif (A) ou négatif ($\neg A$). Une *clause* (C, D, ...) est une disjonction de littéraux : $L_1 \vee \ldots \vee L_n$, dans laquelle toutes les variables sont considérées comme universellement quantifiées.

Un terme s est un *sous-terme* d'un terme t, noté $t[s]$, si s se trouve à une position interne de t. Considérer un terme $t[s']$ signifie remplacer s par s' dans $t[s]$. L'ensemble des variables d'un

terme t est noté $\mathcal{V}ar(t)$. Une *substitution* (σ) est une fonction de variables dans des termes. $\sigma(t)$ représente le terme issu du remplacement des variables de t par les termes associés tel que défini dans la substitution σ. $\sigma(L_1 \vee \ldots \vee L_n)$ (resp. $\sigma(\neg A)$) est équivalent à $\sigma(L_1) \vee \ldots \vee \sigma(L_n)$ (resp. $\neg\sigma(A)$). Une substitution σ *unifie* les termes t_1 et t_2 si les termes $\sigma(t_1)$ et $\sigma(t_2)$ sont syntaxiquement égaux (noté $\sigma(t_1) = \sigma(t_2)$). Une substitution σ est un *filtre* du terme t_1 dans le terme t_2 si $\sigma(t_1) = t_2$.

Étant donné un ensemble d'équations E, appelé théorie équationnelle, une substitution σ est un *E-unificateur* des termes t_1 et t_2 si $\sigma(t_1) =_E \sigma(t_2)$, c'est-à-dire si $\sigma(t_1)$ et $\sigma(t_2)$ sont équivalents dans la théorie E. Une substitution σ est un *E-filtre* du terme t_1 dans le terme t_2 si $\sigma(t_1) =_E t_2$.

Une règle de déduction dont les prémisses sont les clauses C_1 et C_2, et dont la conclusion est la clause C est notée :

$$\frac{C_1 \qquad C_2}{C}$$

1.2 Techniques de déduction

Dans cette section, nous présentons les techniques de déduction développées pour le raisonnement en logique du premier ordre, en présence d'équations. Ces techniques sont basées sur les règles de résolution et de paramodulation.

1.2.1 Déduction en logique du premier ordre

Une étape essentielle dans la domaine de la déduction automatique a été franchie en 1965 lorsque J.A. Robinson a défini la règle de *résolution* [23]. Le principe de cette règle est d'identifier deux littéraux opposés dans deux clauses, et de déduire une nouvelle clause composée de la disjonction de tous les autres littéraux de ces deux clauses.

Définition 1 (Résolution) $\dfrac{A_1 \vee D_1 \qquad \neg A_2 \vee D_2}{\sigma(D_1 \vee D_2)}$
où σ unifie A_1 et A_2, c'est-à-dire $\sigma(A_1) = \sigma(A_2)$.

Une interprétation moins formelle de cette règle est la suivante : la première clause signifie "`si non` A_1`, alors` D_1"; la seconde clause signifie "`si` A_2`, alors` D_2". Si A_1 et A_2 sont identiques, il est alors naturel de conclure "D_1 `ou` D_2".

La règle de résolution est généralement définie avec une règle appelée *factorisation*. Son principe est d'éliminer les occurrences multiples d'un littéral dans une même clause. Ainsi, appliquer la règle de factorisation consiste à engendrer des clauses où les occurrences multiples d'un même littéral ont été supprimées.

Définition 2 (Factorisation) $\dfrac{L_1 \vee L_2 \vee \ldots \vee L_n \vee D_1}{\sigma(L_1 \vee D_1)}$
où σ unifie L_1, L_2, ..., et L_n.

1.2.2 Déduction en présence d'équations

Lorsque le prédicat d'égalité ($\approx$) est utilisé, il faut alors considérer toutes ses propriétés :

$$\begin{array}{rl}
\textit{Réflexivité :} & (x \approx x) \\
\textit{Symétrie :} & \neg(x \approx y) \vee (y \approx x) \\
\textit{Transitivité :} & \neg(x \approx y) \vee \neg(y \approx z) \vee (x \approx z) \\
\mathcal{F}\textit{-Réflexivité :} & \forall f \in \mathcal{F}, \forall i,\ \neg(x_i \approx y_i) \vee (f(\ldots, x_i, \ldots) \approx f(\ldots, y_i, \ldots)) \\
\mathcal{P}\textit{-Réflexivité :} & \forall p \in \mathcal{P}, \forall i,\ \neg(x_i \approx y_i) \vee \neg p(\ldots, x_i, \ldots) \vee p(\ldots, y_i, \ldots)
\end{array}$$

Utiliser ces propriétés sous forme de clauses mène à un système de déduction terriblement inefficace.

La règle de *paramodulation* a été introduite par G.A. Robinson et Wos [22] pour remplacer ces propriétés du prédicat d'égalité. Cette règle applique la notion de remplacement d'un terme par un terme égal.

Définition 3 (Paramodulation) $\dfrac{(t_1 \approx t_2) \vee D_1 \qquad L_2[t'_1] \vee D_2}{\sigma(L_2[t_2] \vee D_1 \vee D_2)}$

où σ unifie t_1 et t'_1.

Le sous-terme t'_1 de L_2 est identifié avec le membre gauche de l'équation $(t_1 \approx t_2)$ (grâce à la substitution σ), puis remplacé par le membre droit de cette équation.

Cependant, malgré cette règle de paramodulation, l'axiome de réflexivité doit être conservé explicitement. Une règle supplémentaire a été définie pour simuler le rôle de cette propriété de réflexivité de l'égalité. Cette règle est nommée *réflexion* (or *trivialisation*). Elle permet d'éliminer une équation négative dont les deux membres peuvent être unifiés.

Définition 4 (Réflexion) $\dfrac{(t_1 \not\approx t_2) \vee D_1}{\sigma(D_1)}$

où σ unifie t_1 et t_2.

1.2.3 Complétude

L'intérêt de concevoir un système de déduction n'est avéré que lorsque sa complétude est démontrée.

Théorème 1 *Les règles de résolution, factorisation, paramodulation et réflexion forment un système de déduction réfutationnellement complet.*

Un système de déduction est dit *réfutationnellement complet* si il dérivera toujours une contradiction à partir d'un ensemble de clauses inconsistant. Cela signifie, étant donné un ensemble S de clauses et une clause C, si C est une conséquence logique de S, le système de déduction dérivera toujours une contradiction à partir de S et de la négation de C. La contradiction est représentée par une clause sans littéraux, appelée *clause vide*.

1.3 Stratégies de déduction

Dans cette section, nous présentons plusieurs stratégies dont le but est de guider les étapes de déduction. Les stratégies ordonnées permettent de sélectionner les littéraux dans les clauses. La stratégie de superposition décrit une variante de la règle de paramodulation. Les stratégies

de simplification éliminent ou transforment les clauses redondantes. Les stratégies contraintes conservent une trace de tous les choix effectués.
Toutes ces stratégies peuvent être combinées sans altérer le résultat de complétude présenté dans la section précédente.

1.3.1 Stratégies ordonnées

Pour obtenir un système de déduction efficace, il est essentiel de définir des stratégies sélectionnant les étapes de déduction et les littéraux sur lesquels les appliquer. Dans ce but, nous pouvons définir une mesure de complexité sur les termes et l'utiliser pour comparer plusieurs termes. Une telle mesure est appelée un *ordre*. Nous présentons ci-dessous deux stratégies de sélection de littéraux.

Stratégie ordonnée. La stratégie ordonnée a été définie par Hsiang et Rusinowitch [9]. Elle repose sur un *ordre de simplification total* $>$ sur les termes, c'est-à-dire un ordre satisfaisant les propriétés suivantes :

- Bonne fondation, aussi appelée Noethérianité (il n'existe pas de suite infinie décroissante) ;
- Monotonicité (si $s > t$, alors $u[s] > u[t]$) ;
- Stabilité par sous-terme ($t[s] > s$) ;
- Stabilité par instanciation (si $s > t$, alors $\sigma(s) > \sigma(t)$) ;
- Totalité sur les termes clos ($\forall s, t$ clos, soit $s > t$, soit $t > s$, soit $s = t$).

La définition de cette stratégie consiste en deux lois :

1. N'appliquer des étapes de déduction qu'entre les littéraux maximaux de clauses ;
2. Pour une étape de paramodulation, le terme remplacé (membre gauche de l'équation) doit être plus grand que le terme remplaçant (membre droit de l'équation).

L'objectif de cette stratégie est donc de garantir que toute clause déduite est moins complexe, au sens de l'ordre, que les clauses utilisées pour l'engendrer. La seconde loi a pour but d'utiliser les équations comme des règles de réécriture.

Stratégie positive. La stratégie positive est une variante de la stratégie ordonnée. Son objectif est le suivant : pour dériver la clause vide, c'est-à-dire trouver une contradiction, des littéraux doivent être éliminés de clauses. Il n'existe que deux moyens d'éliminer des littéraux : la résolution et la réflexion. Ces deux règles éliminent au moins un littéral négatif. Donc, si on évite d'engendrer de nombreuses clauses avec des littéraux négatifs, rechercher une contradiction devrait être plus facile.
Le principe de la stratégie positive est précisément d'éviter d'engendrer des clauses avec des littéraux négatifs : une étape de déduction ne peut utiliser une clause avec des littéraux négatifs que si au moins l'un d'entre eux est impliqué dans la déduction. La définition de cette stratégie peut être formulée ainsi :

1. Pour utiliser un littéral positif dans une étape de déduction, il doit appartenir à une clause positive, c'est-à-dire une clauses sans aucun littéral négatif.

Les conditions de maximalité des littéraux impliqués dans les étapes de déduction sont similaires à celles de la stratégie ordonnée, à l'exception des littéraux négatifs : un littéral négatif devra être maximal par rapport aux autres littéraux négatifs de sa clauses ; il n'y a pas de condition d'ordre par rapport aux littéraux positifs de sa clause.

1.3.2 Stratégie de superposition

La stratégie de superposition est basée sur le définition de la règle de *superposition*, une variante de la règle de paramodulation. Cette règle de superposition applique des remplacements à l'aide d'équations, comme la règle de paramodulation. Cependant lorsqu'un remplacement est appliqué dans une équation, avec la stratégie de superposition il doit être effectué dans le plus grand membre de l'équation.

Pour préserver le résultat de complétude, il faut définir une règle supplémentaire, la *factorisation équationnelle*. Cette règle transforme des clauses afin que le membre gauche de l'équation maximale, c'est-à-dire son membre maximal, n'apparaît pas dans d'autres équations.

Définition 5 (Factorisation équationnelle) $\dfrac{(t_1 \approx t_2) \vee (t'_1 \approx t_3) \vee D_1}{\sigma((t'_1 \approx t_3) \vee \neg(t_2 \approx t_3) \vee D_1)}$

où σ unifie t_1 et t'_1, l'équation $(t_1 \approx t_2)$ est maximale dans la clause, et $\sigma(t_1)$ est pus grand que $\sigma(t_2)$ et $\sigma(t_3)$.

1.3.3 Stratégies de simplification

Être capable de simplifier un ensemble de clauses est une tâche essentielle pour l'efficacité. La simplification consiste à remplacer une clause par une clause moins complexe au sens de l'ordre, ou même à supprimer des clauses inutiles.

Transformer une clauses en une plus simple peut s'effectuer par réécriture. La règle correspondant à cette technique est appelée *démodulation* (ou *simplification*). Par exemple, étant donnée une équation $(t_1 \approx t_2)$ où t_1 est plus grand que t_2, toute clause de la forme $C[\sigma(t_1)]$ peut être remplacée par la clause $C[\sigma(t_2)]$.
Une clause peut être supprimée si elle contient une équation positive $(t \approx t)$, ou si le même atome apparaît à la fois en positif et en négatif. Une autre technique de suppression est la *subsomption*, qui peut être schématisée ainsi : si une clause C_1 appartient à un ensemble de clauses S, alors toute clause de la forme $\sigma(C_1) \vee D$ peut être ôtée de S.

Noter que dans toutes ces règles de simplification et de suppression, si une substitution doit être appliquée à des variables, ce sera toujours dans la clause utilisée pour la simplification. La substitution n'instancie jamais les variables de la clause simplifiée. Le calcul d'une telle substitution s'appelle un problème de *filtrage*.

1.3.4 Stratégies contraintes

Des contraintes peuvent être ajoutées aux clauses pour conserver une trace de tous les choix effectués. Par exemple, la condition de maximalité d'un littéral dans une clause peut être impossible à décider en raison des variables. Une étape de déduction peut cependant être appliquée avec ce littéral, le test de maximalité se résumant à vérifier qu'aucun autre littéral de la clause ne lui est supérieur. La stratégie contrainte va alors consister à ajouter le test de maximalité non résolu à la clause déduite. Ainsi, lors cette dernière clause sera utilisée par la suite dans des déductions, nous pourrons alors vérifier que cette condition de maximalité qui a permis de l'engendrer n'est pas enfreinte. Cela permet donc de garantir la cohérence des choix tout au long du processus de déduction.

Des contraintes portant sur les problèmes d'unification rencontrés peuvent également être utilisées pour éviter des déductions inutiles. Ci-après, nous présentons deux stratégies dans ce sens : la *stratégie basique* et la *stratégie contrainte*.

Stratégie basique. Le principe de la stratégie basique énoncée par Bachmair, Ganzinger, Lynch et Snyder dans [6], extension du narrowing basique de Hullot [10], est d'interdire toute paramodulation dans des termes *étrangers*. Un terme étranger dans une clause est un terme introduit par une substitution dans une étape précédente de déduction. Par exemple, considérant l'étape de paramodulation

$$\frac{(f(a,y) \approx y) \qquad p(f(x,c)) \vee q(x,b)}{p(\boxed{c}) \vee q(\boxed{a}, b)}$$

(avec l'unificateur $\{x \mapsto a,\ y \mapsto c\}$), $f(x,c)$ est remplacé par c, en supposant que $p(f(a,c)) > q(a,b)$. Les sous-termes a et c de la clause déduite sont des termes étrangers, car introduits par la substitution. Ils sont donc bloqués (encadrés) pour montrer qu'une étape de paramodulation ne doit pas s'appliquer directement sur eux.
Avec une notation contrainte, la clause déduite est :

$$p(y) \vee q(x,b)\ [\![p(f(a,c)) >^? q(a,b)\ \wedge\ x = a\ \wedge\ y = c]\!]$$

Cette représentation avec des contraintes montre bien l'isolement des termes étrangers, car ils n'apparaissent plus dans le corps de la clause.

Stratégie contrainte. La stratégie contrainte décrite par Kirchner, Kirchner et Rusinowitch dans [13], et reprise par Nieuwenhuis et Rubio dans [16], diffère de la stratégie basique par le fait qu'elle ne résout pas le problème d'unification. Il est ajouté comme contrainte à la clause déduite ; ainsi seul un test de satisfaisabilité est nécessaire.
Chaque clause se voit donc associer une conjonction de contraintes. Et quand une nouvelle clause est déduite, elle hérite des contraintes de ses clauses parentes, auxquelles s'ajoutent les nouvelles contraintes introduites par l'étape de déduction l'ayant engendrée. L'exemple décrit pour la stratégie basique devient alors :

$$\frac{(f(a,y) \approx y)\ [\![T_1]\!] \qquad p(f(x,c)) \vee q(x,b)\ [\![T_2]\!]}{p(y) \vee q(x,b)\ [\![T_1 \wedge T_2 \wedge T_3]\!]}$$

où T_3 contient la contrainte d'unification $f(x,c) =^? f(a,y)$, et la contrainte d'ordre $p(f(x,c)) >^? q(x,b)$ pour représenter une condition non résolue de la stratégie ordonnée.

1.4 Déduction modulo une théorie équationnelle

Pour appliquer des déductions dans une théorie équationnelle, la première étape est de décomposer l'ensemble initial de clauses en deux parties : l'ensemble E des équations représentant la théorie à considérer, et l'ensemble S de toutes les autres clauses. Notre but est alors d'appliquer des déductions entre les clauses de S, et de considérer E comme une théorie implicite. Ainsi, engendrer une propriété P des clauses de S signifie que P est une conséquence logique de S et E.

L'intérêt de travailler modulo une théorie E est d'éliminer l'utilisation explicite d'axiomes très coûteux. Par exemple, la propriété de commutativité d'un opérateur binaire f, $(f(x_1,x_2) \approx f(x_2,x_1))$, est non déterministe, car l'équation ne peut être orientée comme une règle de réécriture lors des étapes de paramodulation.
D'autres propriétés engendrent des dérivation infinies. Un exemple très connu est l'associativité d'un opérateur binaire g, $(g(g(x_1,x_2),x_3) \approx g(x_1,g(x_2,x_3)))$. Étant donnée une équation

$(g(a,b) \approx c)$, il est possible de dériver une nouvelle équation $(g(a,g(b,z_1)) \approx g(c,z_1))$ par paramodulation dans le sous-terme $g(x_1,x_2)$ de l'axiome d'associativité, avec la substitution $\{x_1 \mapsto a, x_2 \mapsto b\}$ et en renommant x_3 en z_1. Avec cette nouvelle équation, il est possible d'appliquer une paramodulation dans le même sous-terme $g(x_1,x_2)$ de l'axiome d'associativité, engendrant ainsi l'équation $(g(a,g(g(b,z_1),z_2) \approx g(g(c,z_1),z_2)$. De cette manière, nous pouvons dériver une infinité de nouvelles équations, ajoutant une nouvelle variable z_i à chaque étape.

1.4.1 Travaux de Plotkin

Les premiers résultats significatifs dans le domaine de la déduction modulo une théorie équationnelle sont dûs à Plotkin [21]. Il a défini un système de déduction basé sur les règles de résolution et de paramodulation. Les détails de ses travaux sont décrits ci-dessous car ils sont à l'origine de mes travaux. Les règles données ci-après sont cependant simplifiées, les originales intégrant en plus la notion de factorisation.

La première règle définie par Plotkin est la **E-résolution** : à partir de deux clauses $A_1 \vee D_1$ et $\neg A_2 \vee D_2$, nous dérivons la clause $\sigma(D_1 \vee D_2)$; σ est une substitution unifiant A_1 et A_2 dans la théorie E ; cela signifie que l'égalité de $\sigma(A_1)$ et $\sigma(A_2)$ est une conséquence logique de E.
Plotkin définit également un règle de **E-trivialisation**, dans laquelle l'unification dans E est utilisée au lieu de l'unification classique (dans la théorie vide).

La règle de E-paramodulation définie par Plotkin est la plus intéressante.

Définition 6 (E-Paramodulation) $\dfrac{(t_1 \approx t_2) \vee D_1 \qquad [\neg]p(\ldots,t,\ldots) \vee D_2}{\sigma([\neg]p(\ldots,s[x/t_2],\ldots) \vee D_1 \vee D_2)}$
où σ est un E-unificateur de t et $s[x/t_1]$, et $< x,s >$ est une paire spéciale.

Expliquons les notions utilisées dans cette définition. Une étape de paramodulation est appliquée à un littéral positif ou négatif ($[\neg]$). $s[x/u]$ représente les terme s dans lequel la variable x est remplacée par le terme u. Une *paire spéciale* est une paire variable-terme définie ainsi :

- $< x,x >$ est une paire spéciale ;
- $< x,s >$ est spéciale si s est un terme $f(x_1,\ldots,x_{i-1},s',x_{i+1},\ldots,x_n)$, $< x,s' >$ est une paire spéciale, les variables x_j sont toutes distinctes et nouvelles (donc aucune variable x_j n'est égale à x ou n'apparaît dans s').

Dans cette règle, nous remarquons que le remplacement n'est pas appliqué dans un sous-terme, mais directement au sommet du terme t, argument de l'opérateur de prédicat p. La position dans t où le remplacement est effectivement réalisé est indiquée par la variable x dans le terme s. Cela revient à simuler un remplacement à l'aide de l'équation $(s[x/t_1] \approx s[x/t_2])$ au lieu de $(t_1 \approx t_2)$. Une telle équation est appelée une *extension* de l'équation initiale.

Pour résumer, les techniques utilisées par Plotkin sont :

1. Unification dans la théorie E ;
2. Extensions implicites.

Mais de nombreuses étapes pour appliquer des déductions sont non déterministes :

- Aucune sélection des littéraux dans les clauses ;
- Aucune orientation des équations quand utilisées pour effectuer des remplacements ;
- Aucune aide pour construire les paires spéciales.

1.4.2 Résultats de complétude

Démontrer la complétude réfutationnelle d'un système de déduction est une tâche très difficile. Nous présentons rapidement dans cette section les principaux résultats obtenus concernant la déduction dans une théorie équationnelle.

En premier lieu, Plotkin a démontré que son système de déduction est complet pour n'importe quelle théorie équationnelle E. Ce résultat est très impressionnant. Cependant, en raison de la définition très générale de la règle de paramodulation, et aux nombreux non déterminismes, le système de Plotkin ne peut être efficace.

Des années plus tard, de nouveaux systèmes de déduction ont été définis pour des classes de théories plus spécifiques. Les premières théories étudiées ont été les théories associatives-commutatives, car elles sont présentes partout en pratique. Pour ces théories, nous avons obtenu les premières preuves de complétude pour des mécanismes de déduction basés sur la paramodulation [26, 27], mais d'autres systèmes fondés sur les mêmes bases ont été démontrés complets par Paul [18], et Bachmair et Ganzinger [5].

En 1992, Wertz [31] a démontrée la complétude réfutationnelle de sont système de déduction pour toute théorie E satisfaisant la propriété suivante :

$$\forall s, t \text{ clos, si } s =_E t[s], \text{ alors } \forall s' \text{ clos, } s' =_E t[s'] \tag{1.1}$$

La technique utilisée par Wertz pour démontrer ce résultat de complétude est basée sur la construction de modèle de Bachmair et Ganzinger [4]. Paul a aussi démontré la complétude dans la même classe de théories pour un système de déduction similaire [19], mais en utilisant une technique de preuve différente, basée sur les arbres sémantiques transfinis de Hsiang et Rusinowitch [9].
Nous avons généralisé le système de déduction développé pour les théories associatives-commutatives dans [27] à une théorie équationnelle E, et nous avons démontré sa complétude réfutationnelle pour toute théorie régulière [29, 30] :

$$E \text{ est } \textit{régulière} \text{ ssi } \forall (t_1 \approx t_2) \in E,\ \mathcal{V}ar(t_1) = \mathcal{V}ar(t_2) \tag{1.2}$$

Notre technique de preuve est également basée sur les arbres sémantiques transfinis de Hsiang et Rusinowitch, mais diffère significativement de celle de Paul, car l'arbre sémantique construit est totalement différent. Nous avons également démontré que notre système de déduction est complet pour les théories satisfaisant la propriété (1.1), car certaines théories non régulières satisfont (1.1), alors que de nombreuses théories régulière ne satisfont pas (1.1).

Exemple 1 ((1.1) $\not\Rightarrow$ (1.2)) *Soit E la théorie $\{f(x, y) \approx x\}$, où f est le seul opérateur fonctionnel, et a est la seule constante. E n'est pas régulière, cependant E satisfait la propriété (1.1). En effet, tout terme clos est E-égal à a.*

Exemple 2 ((1.2) $\not\Rightarrow$ (1.1)) *Soit E la théorie $\{f(x, 0) \approx x\}$. E est régulière. Le terme clos $f(0, 0)$ est E-égal à 0. Cependant, s'il existe une constante a, $f(0, a)$ n'est pas E-égal à a. E ne satisfait donc pas la propriété (1.1).*

1.4.3 Stratégies compatibles

Dans [29], j'ai démontré que les stratégies ordonnées, de simplification et de superposition décrites dans la Section 1.3 peuvent être utilisées sans perte de complétude pour la déduction modulo une théorie équationnelle E. Dans les paragraphes suivants, je précise les principales adaptations de ces stratégies à la déduction modulo E.

Stratégies ordonnées. Les stratégies ordonnées et positives peuvent être appliquées à condition que l'ordre $>$ utilisé pour comparer les termes soit *compatible* avec la théorie E, c'est-à-dire :

$$\forall s, t, s', t' \text{ si } s > t,\ s' =_E s \text{ et } t' =_E t,\ \text{alors } s' > t'$$

Peu d'ordres satisfaisant cette propriété ont été définis. Il en existe pour les théories associatives-commutatives [14, 25] et les théories associatives [24].
Remarque: Il est avéré qu'il existe une relation entre la définition de tels ordres et les résultats de complétude que nous avons obtenus : étant donnée une théorie E, s'il existe un ordre total E-compatible, alors E est une théorie régulière [2].

Stratégies de simplification. Les stratégies pour éliminer ou transformer des clauses redondantes reste valides. Pour les appliquer, il faut utiliser un algorithme de E- filtrage, et des étapes de simplification peuvent nécessiter l'utilisation d'extensions d'équations (cf. paragraphes suivants).

Stratégies contraintes. Les stratégie basique et contrainte ont été démontrée valides pour la déduction dans des théories associatives-commutatives [28, 15] et dans des théories associatives [24]. Les avantages théoriques de la stratégie contrainte sont beaucoup plus significatifs que dans la théorie vide :

- Une seule clause est déduite à chaque étape, au lieu d'autant de clauses que de solutions principales au problème de E-unification, certains problèmes pouvant en avoir de nombreuses. Par exemple, pour le problème d'unification $x * x * x * x =^?_{AC} y_1 * y_2 * y_3 * y_4$ où l'opérateur $*$ est associatif-commutatif (AC), au lieu d'engendrer une clause par solution principale (c'est-à-dire 34.359.607.481 clauses), nous n'en engendrons qu'une seule. Cette stratégie permet aussi de travailler dans des théories dont certains problèmes d'unification ont une infinité de solutions principales, comme les théories associatives.
- Il suffit de vérifier la satisfaisabilité des problèmes de E-unification. Par exemple, le gain est exponentiel dans le cas des théories associatives-commutatives.

Stratégies d'extension. Dans la règle de paramodulation de Plotkin, nous avons montré l'utilisation implicite d'extensions d'équations : à partir d'une équation $(t_1 \approx t_2)$, une équation $(s[t_1] \approx s[t_2])$ est simulée. Ces extensions ont été étudiées par Jouannaud et Kirchner [12], et par Bachmair et Dershowitz [3]. Détaillons un exemple simple pour expliquer pourquoi des extensions doivent être considérées.

Exemple 3 *Soit E la propriété d'associativité d'un opérateur f :*

$$\{f(f(x_1, x_2), x_3) \approx f(x_1, f(x_2, x_3))\}$$

Soit S l'ensemble composé des trois clauses suivantes :

$$\begin{cases} (f(a, b) \approx c) & (1) \\ (f(a, f(b, d)) \approx e) & (2) \\ \neg(f(c, d) \approx e) & (3) \end{cases}$$

S est incohérent avec la théorie E, mais en utilisant la règle de paramodulation classique et de la E-unification, il n'est pas possible d'appliquer une seule déduction. Cependant, l'incohérence

peut être montrée ainsi :

$$\begin{array}{ccc} f(\underbrace{f(a,b)}_{\downarrow(1)},d) & =_E & \underbrace{f(a,f(b,d))}_{\downarrow(2)} \\ f(\overbrace{c}^{},d) & \approx & \overbrace{e}^{} \end{array}$$

À partir de l'axiome d'associativité et des équations (1) *et* (2)*, nous pouvons dériver* $(f(c,d) \approx e)$*, contredisant la clause* (3)*.*

La solution pour trouver cette contradiction par nos règles de déduction est de considérer l'extension $(f(f(a,b),z) \approx f(c,z))$ *de la première équation. Une étape de paramodulation de cette extension dans l"équation* (2) *engendre immédiatement* $(f(c,d) \approx e)$*.*

Les extensions peuvent être considérées de manière explicite ou implicite. Explicitement, il faut définir une règle de déduction engendrant les extensions possibles d'une équation donnée. Cette technique a été utilisée par Peterson et Stickel [20], Bachmair et Ganzinger [5], Wertz [31] et Paul [19]. Son principal défaut est la nécessité d'un contrôle strict de l'application de cette règle et des extensions engendrées, afin d'éviter des étapes de déduction inutiles.
Nous avons utilisé la technique de Plotkin, c'est-à-dire les extensions implicites, en les codant dans la règle de paramodulation [27, 30]. Ceci est réalisé en pré-calculant les contextes possibles simplement à partir des équations contenues dans la théories E. Un contexte est le squelette ajouté autour de chaque membre de l'équation à étendre. Ainsi dans l'exemple précédent, $f(\cdot,z)$ est le contexte utilisé pour étendre $(f(a,b) \approx c)$. Ce calcul de contextes est combiné avec un algorithme détectant ceux qui sont redondants [30].

1.5 Systèmes de déduction

1.5.1 Le système daTac

Le système daTac [1], signifiant *Déduction Automatique dans des Théories Associatives et Commutatives*, a été développé au LORIA. Ce logiciel est écrit en OCaml (≈22000 lignes), un langage de la famille ML développé à l'INRIA [1]. Il peut être exécuté sur toute plate-forme. Son interface graphique a été réalisée en Tcl/Tk [17].

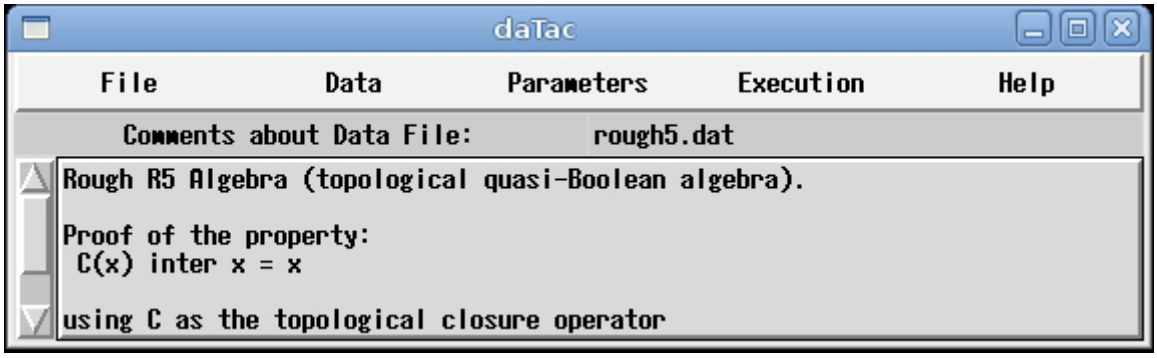


FIG. 1.1 – Le système daTac.

daTac est conçu pour la déduction automatique en logique du premier ordre, et dans des théories E composées d'axiomes de commutativité et d'associativité-commutativité. Les techniques

[1] daTac est disponible sur la page Web `http://www.loria.fr/equipes/cassis/softwares/daTac/`.

de déduction implantées sont détaillées dans [27, 28, 29]. De nombreuses stratégies peuvent être combinées :

- Pour sélectionner les littéraux : stratégies ordonnée et positive.
- Pour le prédicat d'égalité : stratégies de paramodulation et de superposition.
- Pour éliminer les clauses redondantes : subsomption, simplification par réécriture, ...
- Pour les problèmes de E-unification : stratégies basique et contrainte.

daTac propose de démontrer des propriétés, soit par réfutation, soit par déduction en avant (complétion).
La technique de réfutation est une technique d'existence de preuve : la négation de la propriété à démontrer doit être ajoutée, et l'outil cherche une contradiction. Alors, si une preuve existe, le démonstrateur devrait annoncer : « oui, c'est un théorème », c'est-à-dire le démonstrateur fonctionne comme un vérificateur d'existence de preuve. Bien sûr, le système est semi-décidable, comme tout démonstrateur classique en logique des prédicats.
Dans la déduction en avant, le démonstrateur agit comme un système de déduction, c'est-à-dire le résultat produit est un ensemble de propriétés accompagnées de leur preuve formelle. Cette seconde technique a principalement été utilisées pour étudier des algèbres approximantes (cf. Chapitre 3).

Lorsqu'une exécution est terminée, l'utilisateur peut demander un grand nombre d'informations supplémentaires. En particulier, le démonstrateur peut fournir le détail des étapes ayant mené à une propriété dérivée, ou à la contradiction recherchée. Diverses statistiques sont également disponibles, comme le nombre d'étapes de déduction, le nombre d'étapes de simplification, le nombre de clauses supprimées, etc.

1.5.2 Systèmes récents

Cette section liste les principaux outils de déduction qui mettent en œuvre les techniques de paramodulation ou de superposition décrites dans ce chapitre. Les principales stratégies liées aux travaux décrits dans ce chapitre sont énumérées. Cette liste d'outils n'est pas exhaustive, elle correspond aux outils de déduction ayant pris part à la compétition CASC[2] lors de la dernière édition de la conférence CADE en 2011 [7].

a - E (`http://www4.informatik.tu-muenchen.de/~schulz/E/`)

Démonstrateur équationnel pour la logique du premier ordre, basé sur la superposition et la réécriture. Plusieurs stratégies de sélection de littéraux sont proposées, mais toutes dans la philosophie de la stratégie ordonnée positive.

b - iProver-Eq (`http://www.cs.man.ac.uk/~sticksec/iprover-eq/`)

Démonstrateur basé sur un solveur de clauses closes, mais utilisant la superposition unitaire pour raffiner le modèle construit.

c - Otter (`http://www.cs.unm.edu/~mccune/otter/`)

Démonstrateur en logique du premier ordre avec égalité. Il est basé sur les règles de résolution et de paramodulation. Son extension, appelée EQP (`http://www.cs.unm.edu/~mccune/eqp/`), applique des déductions modulo des théories associatives-commutatives, et propose la stratégie basique. C'est EQP qui a démontré la conjecture de Robbins en 1996.

[2] Cf. `http://www.cs.miami.edu/~tptp/CASC/23/` pour plus de détails sur cette compétition.

d - SPASS (`http://www.spass-prover.org/`)

Démonstrateur pour la logique du premier ordre avec égalité, étendant la technique de superposition avec des sortes. Il intègre également les règles de résolution (et hyper résolution), factorisation et paramodulation, ainsi qu'une règle de splitting. La stratégie ordonnée classique est également proposée.

e - Vampire (`http://vprover.org/`)

Démonstrateur pour la logique du premier ordre classique. Il implante les règles de résolution et de superposition, ainsi qu'une règle de splitting sans backtracking.

En conclusion, de très nombreux démonstrateurs utilisent les techniques de déduction de paramodulation ou de superposition, ainsi que les stratégies de sélection ordonnées et ses dérivées, mais très rares sont ceux qui arrivent à isoler une théorie équationnelle et à appliquer des déductions modulo cette théorie. Le logiciel daTac reste unique, et même s'il est très ancien maintenant, il pourrait toujours être utilisé comme un noyau auquel greffer des stratégies appropriées à une problématique précise, comme nous l'avons fait dans Casrul pour l'analyse de protocoles de communication.

Chapitre 2

Normalisation efficace de termes

Ce chapitre décrit des travaux sur l'une des briques de base de la déduction, la réécriture. Elle sert à simplifier les informations déduites, d'une part pour vérifier qu'elles sont utiles par rapport aux informations déjà connues, mais aussi pour les normaliser, c'est-à-dire les « affecter » à une classe d'informations équivalentes pour le système considéré.

L'objectif de mes travaux dans ce domaine a été de limiter les actions redondantes lorsque des termes sont simplifiés par réécriture. Cette étape, qui correspond à une normalisation par rapport aux équations présentes dans l'état courant du système, est extrêmement coûteuse, principalement en raison de la répétition de transformations identiques : les mêmes termes et sous-termes sont très souvent engendrés lors de déductions, et à chaque fois il faut répéter les mêmes transformations pour les normaliser. Dans ce chapitre, je présente donc deux méthodes d'optimisation de la normalisation : par tabulation, et par clôture de congruence.

L'article référence de ce chapitre est celui publié dans le journal international JAR (Journal of Automated Reasoning) en 2003 [35]. Il présente la synthèse des travaux que j'ai effectués dans ce domaine, c'est-à-dire la procédure de clôture par congruence abstraite et son extension aux théories associatives et commutatives, mais aussi montre comment les anciens algorithmes de clôture par congruence peuvent être décrits comme une stratégie de combinaison des règles d'inférence de notre procédure.

2.1 Introduction

Le calcul de clôture par congruence est une préoccupation très ancienne, qui a pour but de tester la consistance d'un ensemble d'équations et de diséquations, ou d'étudier le problème du mot (équivalence de deux termes dans une théorie équationnelle), et ceci dans un cadre avec ou sans variables.

L'expression « clôture par congruence » désigne en général une structure de données représentant les relations de congruence induites par un ensemble d'équations closes.

Son calcul consiste donc, à partir d'un ensemble d'équations, de construire les classes d'équivalence des termes représentés. En général, ce calcul met en œuvre des techniques très lourdes et relativement compliquées. Parmi ces techniques, nous pouvons citer :

- *le DAG* (graphe acyclique dirigé), pour représenter l'univers des termes,
- *la structure de données union-find*, pour maintenir les classes d'équivalence,
- *la table de signatures*, pour stocker une signature par sommet du graphe,

– *la table d'utilisation* (ou liste des prédécesseurs), pour représenter la propriété de sous-terme.

Une signature est représentée par un terme $f(c_1, \ldots, c_n)$ où f est un opérateur, et $c_1, \ldots, c_n$ sont des noms de classes d'équivalence.

Les procédures traditionnelles de calcul de clôture par congruence sont celles de Nelson-Oppen [55] (utilisant un DAG en entrée, une table d'utilisation et une structure union-find), Downey-Sethi-Tarjan [43] (utilisant un DAG en entrée, une table d'utilisation et une table de signatures) et Shostak [63] (procédure dynamique basée sur une table d'utilisation).

Comme le montre Shostak, le problème de clôture par congruence est en fait un problème de combinaison de théories en logique du premier ordre avec égalité, chaque théorie consistant en exactement un opérateur de fonction ou une constante.

Avec Leo Bachmair et I.V. Ramakrishnan je me suis intéressé à l'extension de ces travaux pour considérer des théories implicites. Nous nous sommes focalisés sur l'exemple classique des théories associatives-commutatives. Des travaux préliminaires [33] ont été effectués dans la théorie vide et dans les théories AC, mais en se basant sur le principe de tabulation des transformations, principe à la mode à l'époque à Stony Brook (NY) en raison de la conception d'un Prolog avec cette stratégie, XSB. Ces travaux sont décrits en Section 2.2.

Suite à l'article de Kapur [49] étudiant le problème de projection de l'algorithme de clôture par congruence de Shostak [63] dans le cadre de la complétion close vers des règles de réécriture, Bachmair et Tiwari [34] ont repris nos travaux et défini une clôture par congruence abstraite, représentée par un système de réécriture clos convergent. Je décris brièvement en Section 2.3 cette méthode, puis présente en Section 2.4 l'extension réalisée avec ces auteurs aux théories associative-commutatives, ainsi que la procédure pour reconstruire un système de réécriture convergent sur la signature initiale (Section 2.5).

2.2 Normalisation efficace par tabulation

Étant donnés un ensemble d'équations S et un ensemble d'expressions buts G, l'objectif est de simplifier les expressions buts à l'aide de S.
À partir d'un ordre sur les termes, la première étape consiste à transformer l'ensemble d'équations S en un ensemble de règles de réécriture R. Ensuite, la simplification des expressions de G revient à normaliser par réécriture chaque expression par rapport à R.

Un procédé de normalisation comporte deux opérations coûteuses :
– chercher une règle de R réduisant un terme de G ;
– répéter des étapes de réécriture déjà réalisées.

Pour l'étape de simplification, il existe deux familles de méthodes : oublieuses (*oblivious*) et non oublieuses.

Une méthode oublieuse ne mémorise pas l'historique de ses calculs. Parmi les nombreuses méthodes de cette famille, citons la réduction directe (*straight reduction*) [57, 46, 47], la méthode parallèle externe (*parallel-outermost*) [57], la méthode gauche externe (*leftmost-outermost*), ainsi que les méthodes de Huet-Levy [48] et Sekar-Ramakrishnan [61].
D'autre part, des stratégies ont également été définies sur les règles, comme par exemple les réseaux discriminants (*discrimination nets*) [40, 32].

Une méthode non oublieuse stocke l'historique des calculs effectués. Cependant mémoriser des paires (terme, forme normale) peut s'avérer très inefficace en temps et en espace.

En 1980, Chew [38, 39] a défini une méthode combinant la réduction directe et l'algorithme de clôture par congruence. Cet algorithme a ensuite été amélioré par Downey, Sethi et Tarjan [43], Nelson et Oppen[56] et Kozen [51].

Le but de la méthode de Chew est de rechercher les conséquences d'un ensemble fini d'équation closes. Les avantages de cette méthode sont d'une part une structure de données très compacte pour l'historique, et d'autre part le fait qu'une règle de réécriture n'est jamais appliquée deux fois.
Cependant, sa principale limite est de ne pouvoir être appliquée qu'à des systèmes linéaires à gauche, sans entrelacements et consistants.

Plus tard, Verma et Ramakrishnan [66] ont démontré que la méthode de Chew est aussi valide pour les systèmes nœthériens sans entrelacements. Verma [65] a même démontré que la méthode est valide pour tout système consistent et convergent (confluent et nœthérien), en transformant le système de réécriture en un système sans entrelacements.

Dans la suite de cette section, je présente une procédure calculant une forme normale d'un terme donné, pour un ensemble de règles de réécriture. Pour cette procédure, aucune propriété particulière n'est requise pour le système de réécriture. Nous décrivons une représentation précise permettant une implantation rapide.

Nous montrons également comment adapter cette procédure pour tenir compte de la présence d'opérateurs associatifs-commutatifs.

Ces travaux ont été présentés au workshop CCL en 1996 [33], mais n'ont jamais été réellement publiés.[3]

2.2.1 Normalisation avec tabulation

Un **état de la procédure** est décrit par un quadruplet (G^i, T^i, R^i, H^i), où
- G^i est l'ensemble des termes buts partiellement réduits ;
- T^i est l'ensemble de tous les termes rencontrés ;
- R^i est l'ensemble de toutes les instances utilisées de règles de R ;
- H^i est l'historique de toutes les réductions appliquées : ensemble d'ensembles H^i_j de termes ; chaque ensemble de termes représente en fait une classe d'équivalence pour R^i, et contient un unique terme marqué, considéré comme le représentant de la classe.

Propriété 1 *L'historique satisfait la propriété suivante :*

$$\forall j, \exists t_j \in H^i_j, \forall s \in H^i_j, s \rightarrow^*_{R_i} t_j$$

Le terme t_j est dit marqué dans H^i_j.

L'**état initial** est défini par :
- G^0 est égal à G, l'ensemble des termes buts ;
- T^0 est l'ensemble des termes et sous-termes de G ;
- R^0 est vide ;
- H^0 est l'ensemble des singletons $\{t\}$ pour tout terme t de T^0.

Une **transition** est un pas

$$(G^i, T^i, R^i, H^i) \longrightarrow (G^{i+1}, T^{i+1}, R^{i+1}, H^{i+1})$$

résultant des actions suivantes :

[3] L'idée présentée est cependant réapparue dans les travaux de Kapur [49] et Bachmair-Tiwari [34], puis a été la base des travaux réalisés avec ces deux derniers auteurs.

1. Sélectionner un terme ou un sous-terme but t dans G^i.
2. Réduire ce terme par une instance $l\sigma \to r\sigma$ d'une règle de réécriture $l \to r$ de R, c'est-à-dire trouver un filtre σ du membre gauche l de la règle vers t.
3. Ajouter cette règle $l\sigma \to r\sigma$ à R^i pour définir R^{i+1}.
4. Calculer la forme normale pour R^{i+1} de l'instance du membre droit $r\sigma$ de la règle, notée $r\sigma{\downarrow_{R^{i+1}}}$.
5. Vérifier si le terme obtenu $r\sigma{\downarrow_{R^{i+1}}}$ existe déjà dans la base de données.
6. Stocker la réduction dans l'historique, c'est-à-dire ajouter la réduction $t \to r\sigma{\downarrow_{R^{i+1}}}$, et mémoriser que la nouvelle forme normale de tous termes se réécrivant en t est $r\sigma{\downarrow_{R^{i+1}}}$.
7. Appliquer cette réduction dans tous les termes en forme normale pour R^i : pour tout terme $f(\ldots, t, \ldots)$ irréductible par R^i,
   (a) Calculer la forme normale s pour R^{i+1} de $f(\ldots, r\sigma{\downarrow_{R^{i+1}}}, \ldots)$ (cf. cas 4).
   (b) Vérifier si s existe déjà dans la base de données (cf. cas 5).
   (c) Stocker la réduction $f(\ldots, t, \ldots) \to s$ dans l'historique (cf. cas 6).
   (d) Appliquer cette réduction à tous les termes en forme normale pour R^i (cf. cas 7).

La première normalisation concerne le membre droit $r\sigma$ de la règle de réécriture choisie (point 4 de la procédure). Lorsque $r\sigma{\downarrow_{R^{i+1}}}$ a été calculé, ce terme devient le terme marqué dans $H^{i+1}[r\sigma]$. Pour clore l'initialisation du processus de normalisation, tous les termes de $H^i[l\sigma]$ doivent être transférés dans $H^{i+1}[r\sigma]$ et T^{i+1} au fur et à mesure de leur traitement.

Ensuite, la normalisation des termes marqués de H^i (point 7 de la procédure) consiste à répéter la normalisation du terme marqué de chaque ensemble de H^i, jusqu'à ce que H^i soit vide.

Une telle transition doit avoir pour conséquence la propagation de l'étape de réécriture dans tous les autres termes considérés. Cependant, nous n'avons pas besoin de calculer la forme normale de chaque terme de T^i. Le principe de l'historique est de mémoriser qu'un groupe de termes se réduit en un unique terme marqué. C'est la signification des ensembles de termes de H^i. Dans chaque ensemble, il y a au plus un terme irréductible pour R^i. En considérant R^{i+1}, nous n'avons donc qu'à calculer la forme normale d'un seul terme par ensemble de l'historique, le terme marqué.

La **fin de la procédure** est détectée à l'étape n si aucun terme but ou sous-terme de G^n ne peut être réduit par une instance d'une règle de R. Alors, la forme normale d'un terme t de G est le terme marqué de l'ensemble de H^n auquel appartient t.

La propagation de pas de réécriture réalisés lors de transitions s'effectue par un calcul de forme normale pour R^{i+1} à l'aide de H^i.

Remarques

La procédure décrite dans cette section soulève les remarques suivantes :
- Si une règle $l\sigma \to r\sigma$ a été utilisée pour réduire un terme but, aucune règle dont le membre gauche contient $l\sigma$ comme sous-terme ne sera utilisée par la suite.
- La **réécriture avec priorité** est impossible. Par exemple, supposons que les règles $f(a) \to b$ et $a \to c$ appartiennent à R, la première étant prioritaire à la seconde. Soit t un terme but qui se réécrit en plusieurs étapes en $t'[f(a)]$. Supposons que $a \to c$ a été utilisée dans certaines de ces réductions. Alors, le terme $t'[f(a)]$ est normalisé en $t'[f(c)]$, alors qu'il devrait être réduit par la règle de priorité supérieure $f(a) \to b$ en $t'[b]$.

- La procédure décrite dans cette section **termine** si le système de réécriture est terminant. Cette propriété implique aussi que chaque terme a au plus une forme normale.
- Si un terme a plusieurs formes normales, notre procédure n'en calcule qu'une. Ainsi, même si deux termes ont une forme normale commune, ils peuvent ne pas apparaître dans le même ensemble de l'historique à la fin du processus.
- Propriété de **correction** : *deux termes du même ensemble de l'historique ont une forme normale commune*. Un terme t est ajouté dans un ensemble si, soit l'ancien terme irréductible de cet ensemble (terme marqué) se réduit en t, soit le terme t se réduit en ce terme marqué.

2.2.2 Structure de données

Le point clé de cette normalisation avec tabulation est la représentation des informations manipulées, et donc le choix de la structure de données qui va permettre d'implanter efficacement la notion d'historique décrite précédemment.

Pour cela, nous nous sommes tournés vers la notion de signature définie par Chew [38, 39], basée sur le nommage des termes.

Définition 7 *Soit N une fonction qui associe un nom à un terme. Cette fonction est définie par :*

1. *Le domaine de N est T^i.*
2. *Soient t_1 et t_2 deux termes de T^i. $N(t_1)$ est égal à $N(t_2)$ si et seulement si $H^i[t_1]$ est égal à $H^i[t_2]$.*
3. *Soient t_1 et t_2 deux termes de T^i tels que $N(t_1) = N(t_2)$. Il existe un terme t_3 dans T^i tel que $t_1 \rightarrow^*_{R^i} t_3$, $t_2 \rightarrow^*_{R^i} t_3$ et $N(t_3) = N(t_1)$.*

Le troisième point de cette définition est impliqué par le deuxième, par définition de H^i : dans chaque ensemble S de H^i, il existe un terme t tel que tout terme t' de S est réductible par R^i en t.

Définition 8 *À l'aide de la fonction N, nous définissons la représentation d'un terme $f(t_1, \ldots, t_m)$, appelée signature de ce terme, par : $(f, N(t_1), \ldots, N(t_m))$.*

Cette définition requiert que T^i contienne tous les sous-termes des termes rencontrés jusque là lors de la procédure de normalisation.

Ces signatures sont utilisées dans l'historique : H^i est un ensemble d'ensembles de signatures. Par définition de la fonction N^i, tous les éléments d'un même ensemble ont le même nom. Notons H^i_k l'ensemble de H^i dont les éléments ont pour nom k.
Les termes représentés par un ensemble de H^i, ou par une signature, sont définis par :

- Un terme t est *représenté par l'ensemble* H^i_k de H^i si et seulement si il existe une signature dans H^i_k représentant t.
- Un terme $f(t_1, \ldots, t_k)$ est *représenté par une signature* $(f, n_1, \ldots, n_k)$ si et seulement si t_k est représenté par l'ensemble $H^i_{n_j}$ pour $j = 1, \ldots, k$.

Une signature est *réductible pour* R^i si tous les termes représentés par cette signature sont réductibles pour R^i. Une signature qui n'est pas réductible pour R^i est dite *irréductible pour* R^i.

La *forme normale pour* R^i *d'un ensemble* H^i_j, dénotée $H^i_j{\downarrow_{R^i}}$, est le terme défini par

$$f(H^i_{n_1}{\downarrow_{R^i}}, \ldots, H^i_{n_k}{\downarrow_{R^i}})$$

où $(f, n_1, \ldots, n_k)$ est la signature irréductible pour R^i de H^i_j.

Dans la description de notre procédure de normalisation (Section 2.2.1), nous avons mentionné qu'un et un seul terme par ensemble de l'historique est marqué. Ces termes marqués sont les uniques termes considérés pour les réductions. Transposons cela aux signatures : les *signatures marquées* sont les signatures irréductibles pour R^i, et les termes considérés pour les réductions sont les formes normales pour R^i des ensembles H^i_j de H^i.

Les deux exemples suivants montrent l'intérêt de représenter les termes par des signatures.

Exemple 4 *Soit R l'ensemble $\{a \to b\}$ et soit G^0 l'ensemble $\{f(a,a)\}$.*
Par application de la procédure de normalisation, le terme $f(a,a)$ est normalisé par la règle de réécriture $a \to b$ en normalisant d'abord tous les arguments de l'opérateur f. En conséquence, la règle est appliquée deux fois, pour obtenir le terme $f(b,b)$ irréductible pour R.
Avec les signatures, T^0 est $\{f(a,a), a\}$, et la réduction s'applique comme suit.

$$H^0 \begin{cases} H^0_1 = \{\ (f,2,2)\ \} \\ H^0_2 = \{\ (a)\ \} \end{cases} \qquad H^1 \begin{cases} H^1_1 = \{\ (f,2,2)\ \} \\ H^1_2 = \{\ \cancel{(a)},\ (b)\ \} \end{cases}$$

La forme normale de $f(a,a)$ est le terme irréductible pour R^1 de l'ensemble H^1_1, c'est-à-dire $f(b,b)$. Cette normalisation a été réalisée en une seule réduction.

Exemple 5 *Soit R l'ensemble $\{f(a) \to a\}$ et soit G^0 l'ensemble $\{f(f(a))\}$. La procédure de normalisation s'exécute ainsi :*

$$H^0 \begin{cases} H^0_1 = \{\ (f,2)\ \} \\ H^0_2 = \{\ (f,3)\ \} \\ H^0_3 = \{\ (a)\ \} \end{cases} \qquad H^1 \begin{cases} H^1_3 = \{\ \cancel{(f,3)},\ (a)\ \} \end{cases}$$

La forme normale de $f(f(a))$ est a, et il est même exprimé dans H^1 que a est la forme normale de $f^+(a)$.

Ces exemples sont très simples, mais la puissance de la représentation des termes par des signatures est encore plus évidente si on imagine que la règle de réécriture $a \to b$ (ou $f(a) \to a$) représente une séquence de 1000 étapes de réduction.

2.2.3 Actions sur la structure de données

Reprenons la procédure décrite en Section 2.2.1 et adaptons-la à la notion de signature.

1. Sélectionner la signature s irréductible pour R^i (c'est-à-dire marquée) d'un ensemble H^i_k représentant un terme ou sous-terme but.
2. Réduire cette signature s avec une instance $l\sigma \to r\sigma$ d'une règle de réécriture $l \to r$ de R : trouver le filtre du membre gauche l de la règle dans le terme représenté par s.
3. Ajouter cette règle $l\sigma \to r\sigma$ dans R^i pour définir R^{i+1}.
4. Construire la signature s' représentant la forme normale pour R^{i+1} du membre droit $r\sigma$ de la règle.
5. Vérifier si cette signature s' existe déjà dans la base de données.
6. Mémoriser que la signature s se réduit en la signature s' ; la signature irréductible pour R^{i+1} de l'ensemble H^{i+1}_k devient s'.
7. Appliquer cette réduction à toutes les autres signatures irréductible pour R^i : pour toute signature marquée sk de la forme $(f, \ldots, k, \ldots)$,

(a) Calculer la forme normale sk' de $(f, \ldots, k, \ldots)$ pour R^{i+1} ;

(b) Vérifier si la signature sk' existe déjà dans la base de données (cf. cas 5),

(c) Mémoriser que la signature $(f, \ldots, k, \ldots)$ se réduit en sk' (cf. cas 6),

(d) Appliquer cette réduction à toutes les signatures irréductibles pour R^i (cf. cas 7).

2.2.4 Propriétés de cette représentation

Proposition 1 *Il existe exactement une signature irréductible pour R^i par ensemble de H^i.*

Preuve : Les ensembles initiaux de H^0 contiennent une seule signature. Ces signatures sont R^0-irréductibles car R^0 est vide. Il faut alors étudier les cas 5, 6 et 7b, 7c de la procédure décrite dans la section précédente. Lorsqu'une signature s_j, R^i-irréductible, est réduite à l'étape $i+1$, soit sa forme normale pour R^{i+1} n'appartient pas à la base de données et est ajoutée dans H^i_j, soit sa forme normale pour R^{i+1} est déjà dans un ensemble $H^i_{j'}$ de la base de données et alors les ensembles H^i_j et $H^i_{j'}$ sont combinés.

Montrons d'abord qu'il ne peut y avoir plus d'une signature irréductible pour R^i dans un ensemble de H^i. Ce pourrait être le cas lorsque deux ensembles H^i_j et $H^i_{j'}$ sont combinés. Cependant, une telle union ne se produit que lorsqu'une signature s_j de H^i_j, R^i-irréductible, est R^{i+1}-réductible en une signature de $H^i_{j'}$, mais alors également réductible en la signature $s_{j'}$, R^{i+1}-irréductible, de $H^i_{j'}$. Ainsi, toute signature de H^i_j est réductible en $s_{j'}$, et $s_{j'}$ est l'unique signature des deux ensembles qui soit irréductible pour R^{i+1}.

La dernière étape de cette preuve consiste à montrer qu'il existe au moins une signature irréductible pour R^i dans chaque ensemble de H^i. Lorsque la signature s_j R^i-irréductible d'un ensemble H^i_j est réduite à l'étape $i+1$, la nouvelle signature R^{i+1}-irréductible de H^i_j est la forme normale pour R^{i+1} de s_j. L'ensemble de règles de réécriture R terminant, cette forme normale existe. □

De cette existence d'une unique signature irréductible par ensemble de H^i, nous pouvons déduire la conséquence suivante sur les termes.

Corollaire 1 *Pour tout ensemble H^i_j de H^i, il existe un terme t_j représenté par H^i_j, tel que pour tout terme t'_j représenté par H^i_j, $t'_j \rightarrow^*_{R^i} t_j$.*

Preuve : Par la Proposition 1, il y a exactement une signature s_j, R^i-irréductible, par ensemble H^i_j. Soit t_j le terme représenté par s_j et construit uniquement à partir de signatures irréductible pour R^i. Ce terme t_j est R^i-irréductible. □

2.2.5 Normalisation module AC

Nous nous sommes intéressés à l'adaptation de la méthode décrite précédemment pour considérer les propriétés d'associativité et de commutativité d'opérateurs.

Nous avons ainsi étendu la définition de signature en signature AC $(f, args)$, avec aplatissement de l'opérateur AC f, et représentation des arguments de cet opérateur par un multi-ensemble $args$. Et nous avons décrit deux méthodes de propagation des réductions dans une telle signature :

– soit par application au sommet des opérateurs AC, c'est-à-dire au terme composé de tous les éléments de son multi-ensemble d'arguments ; cette solution est facilement réalisable en effectuant des réécritures contextuelles (à l'aide de règles $f(l, x) \rightarrow f(r, x)$ étendant la règle $l \rightarrow r$ de R, si f est l'opérateur AC au sommet de l) ;

– soit par application de réductions dans un « sous-terme », c'est-à-dire un terme formé par l'opérateur AC f et d'un sous-ensemble de $args$.

La seconde méthode est bien sûr très lourde car les tentatives de réduction seront souvent répétées, et le choix des sous-ensembles d'arguments peut s'avérer coûteux, et nécessiterait une structure de données spécifique pour être plus efficace.

2.2.6 Avantages et inconvénients de cette méthode

Une des plus importantes applications de la réécriture de termes avec tabulation est la complétion d'un ensemble d'équations. Dans la procédure de Knuth-Bendix, le processus de normalisation d'un terme est exécuté très souvent, et répète de nombreuses fois les mêmes réductions.

Avec la tabulation, nous n'appliquons que les réductions qui ne l'ont pas encore été. Quand nous obtenons un terme qui a déjà été normalisé auparavant, nous récupérons immédiatement sa forme normale.

Un autre avantage de notre technique est son incrémentalité : nous pouvons ajouter autant de nouvelles règles de réécriture que désiré.

Cependant, il reste quelques limites à notre approche : la présence de variables dans les termes buts est assez problématique ; et si le système de réécriture est non terminant, son utilisation nécessite de prendre des précautions.

2.3 Clôture par congruence abstraite

2.3.1 Introduction

Kapur [49] a étudié le problème de projeter l'algorithme de clôture par congruence de Shostak [63] dans le cadre de la complétion close vers des règles de réécriture. Avec Leo Bachmair, puis Ashish Tiwari, de l'Université de Stony Brook (NY), nous nous sommes intéressés à la formalisation, non pas d'un seul, mais de plusieurs algorithmes de clôture par congruence, afin de pouvoir mieux les comparer et les analyser.

Nous avons suggéré qu'abstraitement, la clôture par congruence peut être définie comme un système clos convergent, sans restreindre d'aucune façon l'applicabilité de la clôture par congruence. Nous donnons des bornes fortes sur la longueur des dérivations utilisées pour construire une clôture par congruence abstraite. Ceci fait ressortir naturellement une relation entre les longueurs des dérivations et les ordres sur les termes utilisés dans la dérivation. La description abstraite à base de règles des aspects logiques des nombreux algorithmes de clôture par congruence publiés permet une meilleur compréhension de ces méthodes. Elle explique le comportement observé dans les implantations et permet également d'identifier les faiblesses de certains algorithmes spécifiques.

Nous avons également montré comment utiliser une signature étendue comme formalisme pour modéliser et raisonner sur des structures de données telles que les DAGs, qui sont basés sur l'idée de partage de structure. Ce point de vue est également applicable à d'autres algorithmes.

Les différentes comparaisons avec les procédures de Nelson et Oppen [55], Downey, Sethi et Tarjan [43] et Shostak [63] ont été réalisées à partir de l'implantation, AbstractCC [4], que j'ai réalisée (en C[5]) de notre procédure abstraite.

[4] AbstractCC est disponible à l'adresse `http://www.csl.sri.com/~tiwari/abstractCC.tar.gz`

[5] J'avais réalisé une première implantation en XSB, un Prolog développé à Stony Brook, dans le but d'utiliser les facilités de ce langage pour tabuler les réductions effectuées, mais notre procédure évitant justement la répétition

2.3.2 Construction efficace de systèmes clos convergents

Des algorithmes de clôture par congruence basés sur des graphes ont également été utilisés pour construire un ensemble convergent de règles de réécriture closes en un temps polynomial par Snyder [64]. Plaisted et. al. [59] ont donné une méthode directe, pas basée sur l'utilisation de clôture par congruence, pour compléter un système de réécriture clos en temps polynomial. Notre travail correspond donc au chaînon manquant entre ces deux travaux, montrant que la clôture par congruence n'est rien d'autre qu'une complétion close.

Snyder [64] utilise une implantation particulière de la clôture par congruence, ce qui l'oblige à faire deux passes de calcul pour obtenir le résultat escompté. Travaillant avec une clôture par congruence abstraite, nous sommes totalement indépendants du choix de l'implantation. Chaque étape de l'algorithme de Snyder peut être décrite par des étapes de notre construction de clôture par congruence, et le résultat final de l'algorithme de Snyder correspond à une clôture par congruence abstraite. En conclusion, l'approche de Snyder consiste à résoudre le problème en abandonnant les techniques de réécriture pour reconstruire un problème sur les graphes de termes, alors que nous restons en réécriture en considérant des extensions.

Plaisted et Sattler-Klein [59] ont montré que les systèmes de réécriture clos peuvent être complétés en un nombre polynomial d'étapes de réécriture, en utilisant une structure de données appropriée pour les termes et en traitant les règles de manière bien spécifique. Nous décrivons la construction de systèmes clos convergents à l'aide de clôture par congruence utilisée comme complétion avec des extensions, puis par une phase de traduction arrière. Plaisted et Sattler-Klein ont montré une complexité quadratique de leur procédure de complétion.

2.4 Clôture par congruence modulo AC

Nous nous sommes donc intéressés au problème de construction d'une clôture par congruence pour un ensemble d'équations closes sur une signature contenant des opérateurs fonctionnels binaires associatifs et commutatifs. Les méthodes traditionnelles basées sur des DAGs sont loin d'être facilement adaptables à ce problème, même si traiter uniquement la commutativité ne nécessite que de très simples modifications (cf. [43] page 767).

Soient Σ une signature avec α pour fonction d'arité, et E un ensemble d'équations closes définies sur Σ. Soit Σ_{AC} le sous-ensemble de Σ, contenant tous les opérateurs associatifs-commutatifs. Notons P les identités

$$f(x_1, \ldots, x_k, s, y_1, \ldots, y_l, t, z_1, \ldots, z_m) \approx f(x_1, \ldots, x_k, t, y_1, \ldots, y_l, s, z_1, \ldots, z_m)$$

où $f \in \Sigma_{AC}$, $k, l, m \geq 0$, et $k + l + m + 2 \in \alpha(f)$; et notons F l'ensemble des identités

$$f(x_1, \ldots, x_m, f(y_1, \ldots, y_r), z_1, \ldots, z_n) \approx f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n)$$

où $f \in \Sigma_{AC}$ et $\{m + n + 1, m + n + r, r\} \subset \alpha(f)$. La congruence induite par toutes les instances closes de P est appelée *congruence de permutation*. L'aplatissement fait référence à la normalisation d'un terme par rapport à l'ensemble F (considéré comme un système de réécriture). L'ensemble $AC = F \cup P$ définit une théorie AC. Les symboles de Σ_{AC} sont appelés opérateurs associatifs-commutatifs.[6] Pour tout opérateur $f \in \Sigma - \Sigma_{AC}$, $\alpha(f)$ est un ensemble singleton, et $\alpha(f) = \{2, 3, 4, \ldots\}$ pour tout $f \in \Sigma_{AC}$.

de réductions, cela s'est avéré inutile.

[6] Les équations $F \cup P$ définissent une extension conservatrice de la théorie d'associativité et commutativité aux termes d'arité variable. Pour un opérateur d'arité binaire fixe, les équations $f(x, y) \approx f(y, x)$ and $f(f(x, y), z) \approx f(x, f(y, z))$ définissent une théorie AC.

Commençons par décrire la forme des termes et équations qui seront utilisés dans la description du calcul de clôture par congruence. Des définitions similaires sont introduites dans [44, 45, 49, 62].

Définition 9 *Soient Σ une signature, dont un sous-ensemble Σ_{AC} contient les opérateurs associatifs-commutatifs, et K un ensemble de constantes disjoint de Σ. Une D-équation (par rapport à Σ et K) est une équation de la forme*

$$f(c_1, \ldots, c_k) \approx c$$

où $f \in \Sigma$ est un opérateur d'arité k et $c_1, \ldots, c_k, c$ sont des constantes de l'ensemble K. Une D-équation orientée, $f(c_1, \ldots, c_k) \to c$, est une D-règle.

Une C-équation (par rapport à K) est une équation $c \approx d$, où c et d sont des constantes de K. Une C-équation orientée est une C-règle.

Des équations, qui lorsqu'elles sont complètement aplaties sont de la forme $f(c_1, \ldots, c_k) \approx f(d_1, \ldots, d_l)$, où $f \in \Sigma_{AC}$, et $c_1, \cdots, c_k, d_1, \cdots, d_l \in K$, sont appelées A-équations. Une A-équation orientée est appelée A-règle.

Par $AC\backslash R$, nous notons le système de réécriture composé de toutes les règles $u \to v$ telles que $u \leftrightarrow^*_{AC} u'\sigma$ et $v = v'\sigma$, pour une règle $u' \to v'$ de R et une substitution σ. $AC\backslash R$ est confluent modulo AC si pour tous termes s, t tels que $s \leftrightarrow^*_{R \cup AC} t$, il existe des termes w et w' tels que $s \to^*_{AC\backslash R} w \leftrightarrow^*_{AC} w' \leftarrow^*_{AC\backslash R} t$. Il s'agit d'une *confluence close* si cette condition est vraie pour tous termes clos s et t.

Une partie de la condition pour la confluence modulo AC peut être satisfaite par l'inclusion d'ex*tensions* de règles [58]. Étant donné un opérateur AC f et une règle de réécriture $\rho : f(c_1, c_2) \to c$, nous considérons son extension $\rho^e : f(f(c_1, c_2), x) \to f(c, x)$. Étant donné un ensemble de règles de réécriture R, nous notons R^e l'ensemble R plus les extensions des règles de R. Ces extensions doivent être utilisées pour réécrire les termes et calculer les paires critiques lorsque des opérateurs AC sont présents. La propriété clé des règles étendues est que si un terme t est réductible par $AC\backslash R^e$ et $t \leftrightarrow^*_{AC} t'$, alors t' est aussi réductible par $AC\backslash R^e$.

Définition 10 *Soit R un ensemble de D-règles, C-règles et A-règles (par rapport à Σ et K). Une constante c de K représente un terme t dans $\mathcal{T}(\Sigma \cup K)$ (via le système de réécriture R) si $t \leftrightarrow^*_{AC\backslash R^e} c$. Un terme t est dit représenté par R s'il est représenté par une constante via R.*

Définition 11 *Soit Σ une signature et K un ensemble de constantes disjointes de Σ. Un système de réécriture clos $R = A \cup D \cup C$ est une clôture par congruence associative-commutative (par rapport à Σ et K) si*

(i) D est un ensemble de D-règles, C est un ensemble de C-règles, A est un ensemble de A-règles, et chaque constante $c \in K$ représente au moins un terme $t \in \mathcal{T}(\Sigma)$ via R, et

(ii) $AC\backslash R^e$ est convergent clos modulo AC sur $\mathcal{T}(\Sigma \cup K)$.

En plus, si E est un ensemble d'équations closes définies sur $\mathcal{T}(\Sigma \cup K)$ tel que

(iii) Si s et t sont des termes sur $\mathcal{T}(\Sigma)$, alors $s \leftrightarrow^*_{AC \cup E} t$ si et seulement si $s \to^*_{AC\backslash R^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC\backslash R^e} t$,

alors R est appelé une clôture par congruence associative-commutative pour E.

2.4.1 Construction d'une clôture par congruence associative-commutative

Soit U un ensemble de symboles à partir duquel les nouveaux noms (constantes) sont choisis. Nous avons besoin d'un ordre de réduction (partiel) AC-compatible qui oriente les D-règles dans le bon sens, et oriente toutes les C- et A-équations. L'ordre $\succ$ basé sur une précédence défini dans [60] convient très bien, pourvu que la précédence sur les opérateurs respecte la condition suivante : $f \succ_{\Sigma\cup U} c$, si $f \in \Sigma$ et $c \in U$. D'autres ordres plus simples conviendraient également, mais nous utiliserons cet ordre car dans notre cas il signifie simplement que les D-règles sont orientées de la gauche vers la droite, et que l'orientation d'une A-règle est donnée en comparant ainsi les termes entièrement aplatis : $f(c_1, \ldots, c_i) \succ f(c'_1, \ldots, c'_j)$ ssi soit $i > j$, soit $i = j$ et $\{c_1, \ldots, c_i\} \succ^{mult} \{c'_1, \ldots, c'_j\}$, c'est-à-dire si les deux termes ont le même nombre d'arguments, il faut comparer les multi-ensembles de constantes en utilisant une extension multi-ensemble $\succ^{mult}$ de la précédence $\succ_{\Sigma\cup U}$ (cf. [41]).

Nous présentons ci-dessous une méthode générale pour construire des clôtures par congruence associatives-commutatives. Nous la décrivons à l'aide de règles de transition qui agissent sur un triplet (K, E, R), où K est un ensemble de nouvelles constantes introduites (la signature originale Σ est fixe) ; E est un ensemble d'équations closes (définies sur $\Sigma \cup K$) à traiter ; et R est un ensemble de C-, D- et A-règles. Une triplet représente un état dans le procédé de calcul d'une clôture. L'état initial est $(\emptyset, E, \emptyset)$, où E est l'ensemble donné d'équations closes.

Les nouvelles constantes sont introduites par la transition suivante.

Extension: $$\frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \to c\})}$$

si $t \to c$ est une D-règle, $c \in U - K$, et t se trouve dans une équation de E qui n'est ni une A-équation ni une D-équation.

Dès qu'une D-règle a été introduite par Extension, elle peut être utilisée pour simplifier des équations.

Simplification: $$\frac{(K, E[s], R)}{(K, E[t], R)}$$

où s se trouve dans une équation de E, et $s \to_{AC\setminus R^e} t$.

Il est évident de constater que toute équation de E peut être transformée en une D-, C- ou A-équation par application de Extension et Simplification.[7]

Les équations sont déplacées du deuxième au troisième composant de l'état par Orientation. Toutes les règles ajoutées dans ce troisième composant sont soit des C-règles, soit des D-règles, soit des A-règles.

Orientation: $$\frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R \cup \{s \to t\})}$$

si $s \succ t$, et $s \to t$ est soit une D-règle, soit une C-règle, soit une A-règle.

La règle Elimination nous permet de supprimer les équations triviales.

Elimimation: $$\frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R)}$$

si $s \leftrightarrow^*_{AC} t$.

[7] Nous n'avons pas besoin d'une règle explicite pour aplatir les termes, car la Définition 9 permet que des termes non aplatis soient dans des A-règles.

Exemple 6 (Exemple montrant le problème posé par AC) *Soit Σ composé des symboles a, b, c, f et g (f est AC), et soit E_0 un ensemble de trois équations $f(a,c) \approx a$, $f(c, g(f(b,c))) \approx b$ et $g(f(b,c)) \approx f(b,c)$. Par application de Extension et Orientation, nous pouvons obtenir une représentation des équations de E_0 à l'aide uniquement de D-règles et de C-règles :*

$$\begin{aligned} R_1 \;=\; & \{a \to c_1,\ b \to c_2,\ c \to c_3,\ f(c_2,c_3) \to c_4, \\ & g(c_4) \to c_5,\ f(c_1,c_3) \to c_1,\ f(c_3,c_5) \to c_2,\ c_5 \to c_4\} \end{aligned}$$

Cependant, le système de réécriture R_1 ci-dessus n'est pas une clôture par congruence de E_0, car ce n'est pas un système de réécriture clos convergent. Mais il est possible de transformer R_1 en un système de réécriture convenable, en utilisant des règles décrites ci-après. Ces règles forment un procédé ressemblant fortement à de la complétion (modulo AC). Le système ainsi obtenu

$$\begin{aligned} R' \;=\; & \{a \to c_1,\ b \to c_2,\ c \to c_3,\ fc_2c_3 \to c_4,\ fc_3c_4 \to c_2,\ fc_1c_3 \to c_1, \\ & fc_2c_2 \to fc_4c_4,\ fc_1c_2 \to fc_1c_4,\ gc_4 \to c_4\} \end{aligned}$$

représente E_0 d'une manière plus compacte. Cependant, tout essai de remplacer les A-règles par deux D-règles (en introduisant une nouvelle constante) mènerait à des dérivations infinies.

La règle suivante considère les superpositions entre extensions de A-règles.

$$\textbf{ACSup}\text{erposition:}\quad \frac{(K,E,R)}{(K,E\cup\{f(s,x\sigma)\approx f(t,y\sigma)\},R)}$$

si $f \in \Sigma_{AC}$, il existe dans R deux D- ou A-règles qui aplaties sont de la forme $f(c_1,\ldots,c_k) \to s$ et $f(d_1,\ldots,d_l) \to t$, les ensembles $C = \{c_1,\ldots,c_k\}$ et $D = \{d_1,\ldots,d_l\}$ ne sont pas disjoints, $C \not\subseteq D$, $D \not\subseteq C$, et la substitution σ est la substitution close de l'ensemble des unificateurs AC les plus généraux de $f(c_1,\ldots,c_k,x)$ et $f(d_1,\ldots,d_l,y)$.[8]

Le cas où l'un des multi-ensembles de constantes est inclus dans l'autre est considéré par la règle ACEffondrement.

$$\textbf{ACEff}\text{ondrement:}\quad \frac{(K,E,R\cup\{t\to s\})}{(K,E\cup\{t'\approx s\},R)}$$

si pour une règle $u \to v \in R$, $t \to_{AC\setminus\{u\to v\}^e} t'$, et si $t \leftrightarrow^*_{AC} u$ alors $s \succ v$.

Noter que les extensions AC des règles ne sont pas ajoutées explicitement dans R. Par conséquence, toute règle de R est soit une C-règle, soit une D-règle, soit une A-règle, mais *pas* son extension. Nous travaillons implicitement avec les extensions dans la règle ACSuperposition.

D'autres règles sont nécessaires pour effectuer des simplifications dans les membres gauches et droits des règles de réécriture. L'utilisation de C-règles pour simplifier les membres gauches des règles est considéré par ACEffondrement. La simplification des membres droits est subsumée par la règle Composition suivante.

$$\textbf{Com}\text{position:}\quad \frac{(K,E,R\cup\{t\to s\})}{(K,E,R\cup\{t\to s'\})}$$

si $s \to_{AC\setminus R^e} s'$.

[8] Dans notre cas, l'ensemble des unificateurs AC les plus généraux contient exactement 2 substitutions, une seule étant close.

Exemple 7 *Soit* $E_0 = \{f(a,c) \approx a,\ f(c,g(f(b,c))) \approx b,\ g(f(b,c)) \approx f(b,c)\}$. *Nous décrivons ci-dessous les étapes intermédiaires d'une dérivation (les exposants dans la dernière colonne indiquent le nombre d'applications de la règle concernée). Nous supposons que* f *est AC et* $c_i \succ c_j$ *si* $i < j$.

i	*Constantes* K_i	*Équations* E_i	*Règles* R_i	*Transitions*
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_1, c_3\}$	$\{fcgfbc \approx b,$ $gfbc \approx fbc\}$	$\{a \to c_1, c \to c_3,$ $fc_1c_3 \to c_1\}$	$\mathbf{Ext}^2 \cdot \mathbf{Sim} \cdot$ $\mathbf{Ori}$
2	$K_1 \cup \{c_2, c_4\}$	$\{fcgfbc \approx b\}$	$R_1 \cup \{b \to c_2,$ $fc_2c_3 \to c_4, gc_4 \to c_4\}$	$\mathbf{Sim}^2 \cdot \mathbf{Ext}^2 \cdot$ $\mathbf{Sim} \cdot \mathbf{Ori}$
3	K_2	$\emptyset$	$R_2 \cup \{fc_3c_4 \to c_2\}$	$\mathbf{Sim}^6 \cdot \mathbf{Ori}$
4	K_2	$\emptyset$	$R_3 \cup \{fc_1c_2 \to fc_1c_4\}$	$\mathbf{ACSup} \cdot \mathbf{Ori}$
5	K_2	$\emptyset$	$R_4 \cup \{fc_2c_2 \to fc_4c_4\}$	$\mathbf{ACSup} \cdot \mathbf{Ori}$

La dérivation déplace les équations, une par une, du second composant de l'état vers le troisième par Simplification, Extension et Orientation. Il est assez facile de vérifier que R_5 *est une clôture par congruence de* E_0. *D'autres ACSuperpositions sont applicables, mais les équations engendrées sont ensuite supprimées. Noter que la dernière condition de la règle Extension interdit de casser une A-règle en deux D-règles, ce qui est crucial pour la terminaison.*

2.4.2 Terminaison et correction

Définition 12 *Notons* $\vdash$ *la relation de transition d'un pas sur les états définis par les règles de transition décrites auparavant. Une dérivation est une séquence d'états*

$$(K_0, E_0, R_0) \vdash (K_1, E_1, R_1) \vdash \cdots$$

Une dérivation est dite équitable si toute règle de transition qui est continuellement applicable est effectivement appliquée à un certain moment. L'ensemble R_∞ *des règles persistantes est défini par* $\cup_i \cap_{j>i} R_j$ *; et similairement,* $K_\infty = \cup_i \cap_{j>i} K_j$.

Nous montrons que toute dérivation équitable n'engendre qu'un nombre fini de règles de réécriture persistantes (dans la troisième composante) en utilisant le lemme de Dickson [37]. Les multi-ensembles sur K_∞ peuvent être comparés en utilisant la relation d'inclusion multi-ensemble. Si K_∞ est fini, cette relation définit un ordre partiel de Dickson. Les démonstrations de tous les résultats énoncés ci-dessous sont détaillées dans [35].

Lemme 1 *Soit* E *un ensemble fini d'équations closes. L'ensemble des règles persistances* R_∞ *dans toute dérivation équitable d'état initial* $(\emptyset, E, \emptyset)$ *est fini.*

Lemme 2 *Supposons* $(K, E, R) \vdash (K', E', R')$. *Alors, pour tous termes* $s, t \in \mathcal{T}(\Sigma)$, *on a* $s \leftrightarrow^*_{AC \cup E' \cup R'} t$ *ssi* $s \leftrightarrow^*_{AC \cup E \cup R} t$. *De plus, pour tous termes* $s_0, s_k \in \mathcal{T}(\Sigma \cup K)$, *si* π *est une preuve close* $s_0 \leftrightarrow s_1 \leftrightarrow \cdots \leftrightarrow s_k$ *dans* $AC \cup E \cup R$, *alors il existe une preuve* π' $s_0 = s'_0 \leftrightarrow s'_1 \leftrightarrow \cdots \leftrightarrow s'_l = s_k$ *dans* $AC \cup E' \cup R'$ *telle que* $\pi \succeq_{\mathcal{P}} \pi'$.

Noter que dans toute dérivation, les extensions des règles de réécriture ne sont jamais ajoutées explicitement, et donc ne sont jamais supprimées non plus. Dès que nous avons convergé vers R_∞, nous introduisons les extensions pour réduire les pics dans les preuves.

Lemme 3 *Si R_∞ est un ensemble de règles persistantes d'une dérivation équitable d'état initial $(\emptyset, E, \emptyset)$, alors R^e_∞ est un système de réécriture clos convergent (modulo AC). De plus, $E_\infty = \emptyset$.*

À l'aide des Lemmes 2 et 3, on peut facilement démontrer le théorème suivant.

Théorème 2 *Soit R_∞ un ensemble de règles persistantes d'une dérivation équitable d'état initial $(\emptyset, E, \emptyset)$. Alors, l'ensemble R^e_∞ est une clôture par congruence associative-commutative pour E.*

Comme R_∞ est fini, il existe un k tel que $R_\infty \subseteq R_k$. Ainsi l'ensemble des règles persistantes peut être obtenu en utilisant seulement des dérivations finies.

2.4.3 Améliorations

L'ensemble de règles de transition pour calculer une clôture par congruence AC peut être amélioré en ajoutant des simplifications et des optimisations. En premier, nous pouvons aplatir les termes de E.

$$\textbf{Apl}\text{atissement:} \qquad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E \cup \{u \approx t\}, R)}$$

où $s \to_F u$.
Cette règle permet donc de construire des clôtures par congruence entièrement aplaties.

Une autre amélioration peut être apportée si l'on observe que certaines extensions de règles ne sont pas nécessaires. Par exemple, il est inutile de considérer les extensions des D-règles créées par la règle Extension pour nommer un sous-terme strict de E. Ce fait peut être facilement intégré à l'aide de contraintes.

Enfin, le choix de l'ordre entre constantes de K peut avoir une forte influence sur l'efficacité de la procédure de calcul. Cet ordre peut être choisi en cours d'exécution. Par exemple, on pourrait toujours choisir cet ordre pour minimiser les applications de ACEffondrement et Composition : pour orienter $c \approx d$, on peut compter le nombre d'occurrences de c et d dans l'ensemble des D- et A-règles (dans le composant R de l'état), et la constante qui a le moins d'occurrences est faite plus grande.

2.5 Construction d'un système de réécriture clos convergent

Nous avons présenté des règles de transition pour construire une présentation convergente sur une signature étendue pour un ensemble d'équations closes. Nous discutons ci-dessous le problème d'obtenir un système de réécriture clos convergent (modulo AC) pour une théorie close donnée sur la signature originale. Nous nous focalisons donc sur la transformation d'un système convergent sur une signature étendue en un système convergent sur la signature originale.

L'idée principale de cette transformation est l'élimination de constantes dans la présentation R : (i) si une constante c de K n'est pas redondante, on choisit un terme $t \in \mathcal{T}(\Sigma)$ qui est représenté par c, et on remplace toutes les occurrences de c par t dans R ; (ii) si une constante c de K est redondante (il existe une C-règle $c \to d$ dans R), alors toutes les occurrences de c sont remplacées par d dans R.

Cette méthode permet de construire un système clos convergent lorsqu'il n'y a pas d'opérateurs AC. Cependant, elle n'est pas suffisante en présence d'opérateurs AC ; en général, le système obtenu ne termine pas. Mais avec une notion adéquate de réécriture AC, les règles sont

vues comme étant convergentes pour cette nouvelle définition. Ceci est utile pour deux raisons : (i) la nouvelle notion de réécriture AC semble plus pratique, dans le sens où elle nécessite moins de travail que la réduction standard $AC\backslash R^e$; et, (ii) elle aide à clarifier l'avantage offert par l'utilisation de signatures étendues lors du traitement d'un ensemble d'équations closes définies sur une signature contenant des symboles associatifs et commutatifs.

2.6 Conclusion

Le fait de pouvoir construire une clôture par congruence AC implique que le problème du mot pour des théories AC finiment représentables est décidable (cf. [54], [52] et [42]).

Nous arrivons à ce résultat sans supposer l'existence d'un ordre de simplification AC total sur les termes clos. L'existence d'un tel ordre avait été établie dans [54], mais nécessitait une démonstration non triviale.

Comme nous construisons un système de réécriture convergent, le problème consistant à déterminer si deux théories AC finiment représentables sont équivalentes est aussi décidable. Et comme les semi-groupes commutatifs sont des cas particuliers de théories AC, où la signature consiste en un seul symbole AC et un ensemble fini de constantes, nos résultats s'appliquent à ce cas spécial [53, 50].

L'idée d'utiliser une abstraction pour transformer un ensemble d'équations avec plusieurs symboles AC en un ensemble d'équations dans lequel chaque équation contient exactement un symbole AC a été mentionnée dans [42]. Toutes les équations contenant le même symbole AC sont isolées, et complétées en un système de réécriture canonique (modulo AC) en utilisant la méthode présentée dans [36]. Cependant, notre méthode pour combiner des théories AC closes avec d'autres théories closes est différente. Dans [42], la théorie close (non AC) est traitée par une complétion close (et utilise un ordre récursif sur les chemins pendant cette complétion). De notre côté, nous utilisons la clôture par congruence. L'intérêt de notre approche peut aussi être observé de par la simplicité de la preuve de correction, et par les résultats obtenus pour transformer un système convergent sur une signature étendue en un système convergent sur la signature initiale.

La méthode pour compléter un semi-groupe commutatif finiment représenté a été décrite sous de nombreuses formes dans la littérature (cf. [36] par exemple). Elle correspond principalement en une spécialisation de l'algorithme de Buchberger pour les idéaux de polynômes, au cas des idéaux de binômes (c'est-à-dire lorsque l'idéal est défini par des polynômes composés d'exactement deux monômes avec coefficients $+1$ et -1).

Il faut noter cependant qu'il y a une différence subtile entre notre méthode et les nombreux autres algorithmes pour décider du problème du mot pour les semi-groupes commutatifs : par exemple, travailler avec des extensions de règles est différent de travailler avec des règles sur les classes d'équivalence (modulo AC) de termes ; ainsi, avec notre méthode, nous pouvons appliquer certaines optimisations très utiles.

L'idée principale derrière notre construction de clôture de congruence associative-commutative est que nous ne considérons que certaines instances des axiomes AC non clos. Si nous sommes intéressés par une $\mathcal{E}$-algèbre décrite par E (où $\mathcal{E}$ est composé uniquement d'axiomes AC pour des symboles de fonction par exemple, et E est un ensemble d'équations closes), alors comme $\mathcal{E}$ contient des axiomes non clos, il faut alors connaître quelles instanciations de ces axiomes doivent être considérées. Pour le cas où $\mathcal{E}$ est un ensemble d'axiomes AC, nous montrons que nous avons besoin de considérer les instances closes dans lesquelles chaque variable est remplacée par des sous-termes de E. Cette observation peut être généralisée et le problème est alors de

savoir quelles restrictions sur les instances des axiomes de $\mathcal{E}$ permettent de décider du problème du mot dans les $\mathcal{E}$-algèbres. Ainsi, Evans [44, 45] donne une caractérisation en terme d'encastrement de $\mathcal{E}$-algèbres partielles. Semi-groupes commutatifs à part, cette méthode fonctionne pour les treillis, les groupoïdes, les quasi-groupes, les boucles, etc.

Chapitre 3

Analyse de quelques approximations...

Ce chapitre décrit l'application de la déduction automatique à l'analyse d'algèbres très spécifiques, appelées algèbres approximantes (*rough sets*). Il s'agit d'explorer l'espace des propriétés de ces algèbres, proches des algèbres modales, d'émettre des hypothèses et de les vérifier automatiquement.

Les travaux décrits ci-après montrent que la déduction automatique peut être mise à profit de diverses manières :

- engendrer des propriétés : le côté déduction est parfois oublié au profit du côté démonstration, et pourtant déduire des milliers de propriétés à partir d'une spécification peut s'avérer très utile comme je le montre dans les Sections 3.3.1 et 3.3.2.
- comparer des algèbres : disposant de très nombreuses propriétés pour plusieurs algèbres, la déduction automatique peut alors être utilisée pour comparer ces propriétés, et donc comparer ces algèbres, comme décrit en Section 3.3.3.
- démontrer des propriétés : l'étude d'algèbres, et en particulier des milliers de propriétés engendrées, permet d'émettre des hypothèses sur certaines propriétés ; la déduction automatique peut alors permettre de valider ou d'invalider ces hypothèses (cf. Section 3.4).

L'article référence de ce chapitre est celui publié comme chapitre du livre Rough Sets in Knowledge Discovery en 1998 [82]. Il contient la description des tous premiers travaux réalisés, mais a été l'élément déclencheur de l'intérêt de la communauté rough sets et fuzzy sets pour nos travaux, et m'a permis de rencontrer et de présenter mes travaux à d'illustres chercheurs, comme Eva Orlowska et Andrew Skowron.

3.1 Introduction

Ce chapitre présente une application réalisée en collaboration avec Anita Wasilewska de l'Université de Stony Brook (NY). Le terme « application » ne désigne pas ici une application industrielle, mais l'application des techniques de déductions définies durant ma thèse à l'étude des propriétés logiques dans un domaine très à la mode dans les années 1990, les ensembles approximants (*rough sets*). Notre projet a consisté à étudier des algèbres liées à la théorie des ensembles approximants. Partant de descriptions simples de ces algèbres, sous forme d'ensembles d'axiomes, nous avons utilisé le démonstrateur daTac pour découvrir de nouvelles propriétés pour chaque algèbre, pour comparer ces algèbres, puis pour démontrer des propriétés de congruence liées à ces algèbres.

Ce chapitre montre comment des techniques générales de déduction peuvent servir à étudier des propriétés mathématiques de différentes manières. Mais je montre aussi que le fossé entre le raisonnement d'un humain et celui d'un démonstrateur est gigantesque, et nécessite parfois d'être comblé par des techniques visuelles.

L'égalité dans ces théories d'ensembles approximants a déjà été étudiée mais en suivant des méthodes très différentes : une étude logique effectuée par Banerjee [67], et une étude algébrique effectuée par Wasilewska [79].

3.2 Théorie des ensembles approximants

La théorie des ensembles approximants fournit un cadre méthodologique pour étudier les problèmes de classification en présence d'informations imprécises ou incomplètes. Cette théorie a été introduite par Pawlak [72, 73]. Elle fournit un modèle complémentaire à la théorie fuzzy [74] ainsi qu'à la théorie de l'évidence [76] pour traiter des informations imprécises, bruitées ou incomplètes. C'est aussi un point de départ pour l'étude en apprentissage automatique, en découverte de connaissances et en fouille de données [83]. Elle a également été appliquée dans de nombreux domaines tels que le diagnostic médical, la récupération d'informations, l'acquisition d'algorithmes de contrôle, et l'analyse de marchés.

Nous avons établi deux nouvelles connexions pour la théorie des ensembles approximants. Nous l'avons liée avec la démonstration automatique et avec la logique algébrique. Plus précisément, nous avons d'abord utilisé la relation connue depuis longtemps [71] entre les ensembles approximants et la logique modale S5, et donc avec des algèbres booléennes topologiques. Nous avons aussi utilisé leur relation avec les *algèbres approximantes*, établie dans [68, 79]. Enfin, nous avons utilisé le démonstrateur daTac pour découvrir de nouveaux théorèmes dans deux algèbres modales et deux algèbres approximantes, mais aussi pour examiner les liens entre ces algèbres. Les algèbres approximantes considérées sont des versions quasi-booléennes d'algèbres de clôture introduite dans [69]. La structure de ces algèbres est relativement complexe, et leurs propriétés sont souvent beaucoup moins intuitives que pour des algèbres modales standard.

Mais avant de détailler ces différentes études, nous donnons ci-dessous quelques définitions de base sur les ensembles approximants et les algèbres modales et approximantes.

3.2.1 Définitions

Les ensembles approximants fournissent l'une des premières méthodologies non statistiques pour l'analyse de données, la découverte de connaissances et la fouille de données [83]. L'un des principaux avantages de cette approche est l'existence d'un modèle théorique bien solide [73] et de nombreux travaux fondamentaux anciens. Voici les principales définitions décrivant cela.

Espace approximant. [73] Soit U un ensemble non vide appelé univers, et soit R une relation d'équivalence sur U. Le triplet $(U, \emptyset, R)$ est appelé un *espace approximant*.

Approximations basse et haute. Soit $(U, \emptyset, R)$ et $A \subset U$. Notons $[u]$ une classe d'équivalence de R. Les ensembles

$$\begin{aligned} IA &= \bigcup\{[u] \in A/R : \ [u] \subset A\}, \\ CA &= \bigcup\{[u] \in A/R : \ [u] \cap A \neq \emptyset\} \end{aligned}$$

sont appelés *approximations basse* et *haute* de A, respectivement. Nous utilisons ici une notation topologique pour ces approximations en raison de leur interprétation topologique effectuée plus tard. Ces approximations sont illustrées dans la Figure 3.1, où chaque case représente une classe d'équivalence.

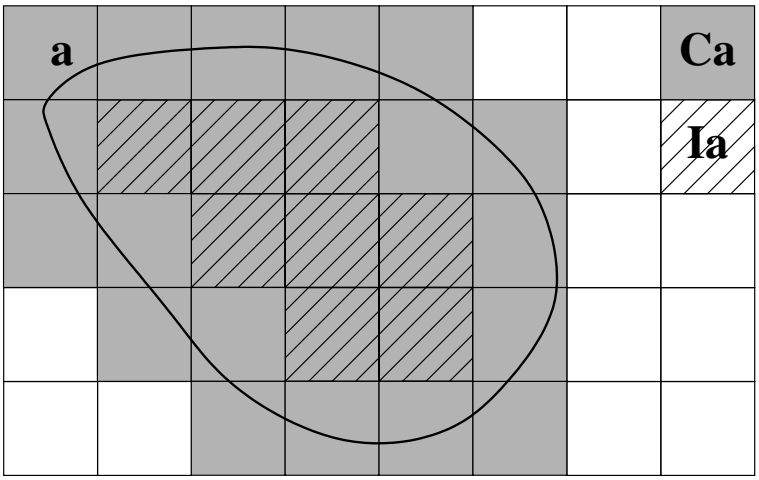


FIG. 3.1 – Approximation d'un ensemble a.

Égalité approximante. [70] Étant donné un espace approximant $(U, \emptyset, R)$ et deux ensembles $A, B \subset U$, ces ensembles A et B sont dits *approximativement égaux*, noté $A \sim_R B$, si est seulement si $IA = IB$ et $CA = CB$.

Algèbre booléenne. Une algèbre abstraite $(A, 1, \cap, \cup, \neg)$ avec élément neutre 1 est appelée *algèbre booléenne* s'il s'agit d'un treillis distributif et chaque élément $a \in A$ possède un complément $\neg\, a \in A$.

Algèbre quasi-booléenne. [75] Une algèbre abstraite $(A, 1, \cap, \cup, \sim)$ est appelée *algèbre quasi-booléenne* si $(A, 1, \cap, \cup)$ est un treillis distributif d'élément neutre 1, et pour tous $a, b \in A$, $\sim(a \cup b) = \sim a \cap \sim b$ et $\sim \sim a = a$.

Noter que le complément dans une algèbre quasi-booléenne est un complément approximatif, d'où sa notation $\sim$, par opposition au complément $\neg$ d'une algèbre booléenne.

Algèbre booléenne topologique. [75] Une algèbre booléenne topologique désigne une algèbre abstraite $(A, 1, \cap, \cup, \neg, I)$ où $(A, 1, \cap, \cup, \neg)$ est une algèbre booléenne, et les propriétés suivantes sont satisfaites : $I(a \cap b) = Ia \cap Ib$, $Ia \cap a = Ia$, $IIa = Ia$, et $I1 = 1$, pour tous $a, b \in A$.

L'élément Ia est appelé *intérieur de* a. L'élément $\neg\, I \neg\, a$ est appelé *clôture de* a et est noté Ca. Ainsi les opérations I et C sont telles que $Ca = \neg\, I \neg\, a$ et $Ia = \neg\, C \neg\, a$. L'élément a est dit *ouvert* (resp. *clos*) si $a = Ia$ (resp. $a = Ca$).

Algèbre modale S4. Toute algèbre booléenne topologique est appelée ici algèbre modale S4, ou plus simplement *algèbre S4*.

Algèbre modale S5. Une algèbre S4 est appelée *algèbre S5* si en plus tout élément ouvert est clos, et tout élément clos est ouvert, c'est-à-dire lorsque nous ajoutons l'un des axiomes suivants : $CIa = Ia$ ou $ICa = Ca$.

Nous considérons ici deux classes d'algèbres approximantes : les algèbres R4 et R5. Elles correspondent aux algèbres modales S4 et S5, et sont des cas particuliers d'algèbres approximantes topologiques définies dans [79]. Il ne s'agit pas de pures inventions mathématiques car, par exemple, l'algèbre approximante de formules de la logique modale S5 comme développée et étudiée dans [68] est un exemple d'algèbre R5. L'algèbre R4 est le pendant quasi-booléen de l'algèbre booléenne topologique, c'est-à-dire de l'algèbre S4. L'algèbre R5 est la version quasi-booléenne de l'algèbre booléenne topologique S5. Les définitions formelles des algèbres R4 et R5 sont les suivantes.

Algèbres R4 et R5. [81] Une algèbre abstraite $(A, 1, \cap, \cup, \sim, I, C)$ est appelée *algèbre R4* si c'est un treillis distributif d'élément neutre 1 et pour tous $a, b \in A$ les propriétés suivantes sont satisfaites : $\sim\sim a = a$, $\sim(a \cup b) = \sim a \cap \sim b$, $I(a \cap b) = Ia \cap Ib$, $Ia \cap a = Ia$, $IIa = a$, $I1 = 1$, $Ca = \sim I \sim a$.
L'algèbre obtenue d'une algèbre R4, par ajout de l'axiome $CIa = Ia$ (ou $ICa = Ca$) est appelée *algèbre R5*.

Une interprétation naturelle sur la théorie des ensembles des propriétés des algèbres booléennes topologiques a été établie par le théorème de représentation de Stone [77]. Par exemple, $a \cap Ia = Ia$ signifie que tout ensemble A contient son intérieur IA. Le théorème de représentation fournit une motivation intuitive pour de nouvelles propriétés et est une source utile de contre-exemples.

Le cas des algèbres R4 et R5 est plus compliqué et beaucoup moins intuitif. Alors que les opérations $\cup$ et $\cap$ sont représentées comme l'union et l'intersection sur la théorie des ensembles, l'opération $\sim$ ne peut pas être représentée comme le complément dans la théorie des ensembles. L'interprétation dans la théorie des ensembles du complément approximant dépend d'une fonction $g : A \longrightarrow A$ telle que pour tout $a \in A$, $g(g(a)) = a$, appelée *involution*. Un théorème de représentation pour ces algèbres R4 et R5 est donné dans [79], montrant à quel point l'aspect intuitif des propriétés des algèbres booléennes topologiques n'est plus de mise ici. Par exemple, l'interprétation de la clôture d'un ensemble A, CA, est : $U - g(I(U - g(A)))$, pour toute involution g. Il est donc très difficile de trouver des propriétés de ces algèbres approximantes sans l'aide d'un outil automatique.

3.2.2 Axiomatisation

Nous décrivons dans cette section l'axiomatisation complète des quatre algèbres étudiées. L'*algèbre R4*, $(A, 1, \cup, \cap, \sim, I, C)$, est définie par : $(A, 1, \cup, \cap)$ est un treillis distributif d'élément neutre 1,

$$\begin{array}{rcl@{\qquad}rcl} x_1 \cap x_2 & \approx & x_2 \cap x_1 & (x_1 \cap x_2) \cap x_3 & \approx & x_1 \cap (x_2 \cap x_3) \\ x_1 \cup x_2 & \approx & x_2 \cup x_1 & (x_1 \cup x_2) \cup x_3 & \approx & x_1 \cup (x_2 \cup x_3) \\ (x_1 \cap x_2) \cup x_2 & \approx & x_2 & x_1 \cap (x_2 \cup x_3) & \approx & (x_1 \cap x_2) \cup (x_1 \cap x_3) \\ x_1 \cap (x_1 \cup x_2) & \approx & x_1 & (x_1 \cup x_2) \cap (x_1 \cup x_3) & \approx & x_1 \cup (x_2 \cap x_3) \\ x_1 \cup 1 & \approx & 1 & x_1 \cap 1 & \approx & x_1 \end{array}$$

l'opérateur $\sim$ est un quasi-complément,

$$\sim\sim x_1 \approx x_1 \qquad \sim(x_1 \cup x_2) \approx \sim x_1 \cap \sim x_2$$

et les opérateurs d'intérieur I et de clôture C sont définis par :

$$\begin{array}{rcl} I(x_1 \cap x_2) & \approx & I(x_1) \cap I(x_2) \\ I(I(x_1)) & \approx & x_1 \\ C(x_1) & \approx & \sim I(\sim x_1) \end{array} \qquad \begin{array}{rcl} I(x_1) \cap x_1 & \approx & I(x_1) \\ I(1) & \approx & 1 \end{array}$$

L'*algèbre R5* est une algèbre R4 avec la propriété additionnelle suivante :

$$C(I(x_1)) \approx I(x_1)$$

Cette propriété est parfois appelée *propriété clopen*, car elle signifie qu'un ensemble clos est ouvert, et qu'un ensemble ouvert est clos.

Les *algèbres modales S4 et S5* sont définies comme les algèbre approximantes R4 et R5 respectivement, à l'exception du complément (noté $\neg$ dans les définitions) qui est entièrement défini par l'ajout des propriétés suivantes :

$$\sim x_1 \cup x_1 \approx 1 \qquad \sim x_1 \cap x_1 \approx \sim 1$$

3.3 Analyse des algèbres modales et approximantes

Nous décrivons ci-après les premiers travaux effectués sur les différences algèbres présentées précédemment. Il s'agit de compléter les propriétés des algèbres quasi-booléennes, puis d'engendrer un grand nombre de propriétés pour toutes les algèbres considérées, et enfin de comparer ces ensembles de propriétés pour mieux comprendre les algèbres et leurs différences.

3.3.1 Algèbres quasi-booléennes

Le premier travail effectué avec le démonstrateur daTac a consisté à étudier les algèbres quasi-booléennes, sans ajouter les axiomes concernant les opérateurs d'intérieur et de clôture. Dans ces algèbres, les opérateurs $\cup$ et $\cap$ sont associatifs et commutatifs : ces propriétés seront déclarées dans la spécification initiale, et les axiomes correspondant n'auront pas besoin d'être donnés. Ainsi, seules 8 équations seront précisées dans la spécification.

Une complétion de ces équations a été réalisée par daTac, ce qui a abouti à un ensemble de 11 équations persistantes.

$$\begin{array}{rcl} (x_1 \cap x_2) \cup x_2 & \approx & x_2 \\ x_1 \cup 1 & \approx & 1 \\ \sim\sim x_1 & \approx & x_1 \\ x_1 \cup x & \approx & x_1 \\ x_1 \cap \sim 1 & \approx & \sim 1 \\ \sim(x_1 \cap x_2) & \approx & \sim x_1 \cup \sim x_2 \end{array} \qquad \begin{array}{rcl} x_1 \cap (x_2 \cup x_3) & \approx & (x_1 \cap x_2) \cup (x_1 \cap x_3) \\ x_1 \cap 1 & \approx & x_1 \\ \sim(x_1 \cup x_2) & \approx & \sim x_1 \cap \sim x_2 \\ x_1 \cap x_1 & \approx & x_1 \\ \sim 1 \cup x_1 & \approx & x_1 \end{array}$$

Ces équations représentent un système canonique (modulo AC) de toutes les propriétés des algèbres quasi-booléennes. Cela signifie que, transformées en règles de réécriture par simple orientation de la gauche vers la droite, elles réduisent une équation en une tautologie si et seulement si cette équation est une propriété des algèbres quasi-booléennes.

Il est possible de compléter cet ensemble d'équations par l'introduction d'un nouveau symbole, 0, représentant le complément de 1, ~ 1. Deux équations viennent alors s'ajouter à cet ensemble : $\sim 1 \approx 0$ et $\sim 0 \approx 1$.

Le même travail de complétion demandé sur les équations représentant une algèbre booléenne, c'est-à-dire par simple ajout de l'équation $\sim x_1 \cup x_1 \approx 1$, ne termine pas car il y a une infinité de propriétés irréductibles, comme par exemple :

$$\begin{aligned}
(x_1 \cap \sim x_2) \cup x_2 &\approx x_1 \cup x_2 \\
(x_1 \cap \sim x_2) \cup (x_1 \cap \sim x_3) \cup (x_2 \cap x_3) &\approx x_1 \cup (x_2 \cap x_3) \\
(x_1 \cap \sim x_2) \cup (x_1 \cap \sim x_3) \cup (x_1 \cap \sim x_4) \cup (x_2 \cap x_3 \cap x_4) &\approx x_1 \cup (x_2 \cap x_3 \cap x_4) \\
&\vdots \\
(x_1 \cap \sim x_2) \cup (x_1 \cap \sim x_3) \cup (x_1 \cap \sim x_4) \cup \ldots \cup (x_2 \cap x_3 \cap x_4 \cap \ldots) &\approx x_1 \cup (x_2 \cap x_3 \cap x_4 \cap \ldots)
\end{aligned}$$

Sémantiquement, ces propriétés sont toutes subsumées par la première, car elles représentent les formes irréductibles des équations :

$$(x_1 \cap \sim (x_2 \cap x_3 \cap x_4 \cap \ldots)) \cup (x_2 \cap x_3 \cap x_4 \cap \ldots) \approx x_1 \cup (x_2 \cap x_3 \cap x_4 \cap \ldots)$$

Cependant il est impossible de les éliminer, car leur présentation distribuée est indispensable.

3.3.2 Recherche de nouvelles propriétés

La première phase de notre étude des quatre algèbres considérées a consisté à engendrer un maximum de nouvelles propriétés pour chacune, pour avoir une idée de la forme de ces propriétés et de leur nombre. Pour cela, nous avons utilisé daTac non pas comme un démonstrateur, mais comme un moteur de déduction.

Mais auparavant nous avons voulu vérifier que l'espace de recherche était bien gigantesque, comme nous nous en doutions. Pour cela, nous avons lancé daTac sur une étude en largeur d'abord de l'algèbre R4. Cette option de l'outil correspond à ce qui est parfois appelé *complétion linéaire* : recherche de toutes les propriétés de profondeur i avant de chercher celles de profondeur $i+1$. À partir des 16 équations initiales, 158 nouvelles ont été déduites mais seulement 26 équations ont été conservées au final ; de ces 26, 2241 ont été déduites, ce qui a donné 144 équations conservées ; avec ces 144 équations, daTac ne s'en est jamais sorti : entre le nombre énorme de déductions possibles et les problème d'unification et de filtrage AC de plus en plus compliqués, il s'est mis rapidement à piétiner et à saturer la mémoire... (et pourtant sur un ordinateur de puissance raisonnable... à l'époque).

Après ce petit revers, nous avons donc décidé de restreindre l'espace de recherche en utilisant quelques paramètres du logiciel : ne pas calculer toutes les solutions des problèmes d'unification AC (surtout lorsque ceux-ci sont très compliqués) ; borner la taille des équations engendrées ; borner le nombre de variables différentes dans une équation. Placer ces bornes fait bien sûr perdre la complétude des déductions effectuées, mais comme nous savions que le nombre de propriétés est infini et comme nous ne cherchions pas à démontrer des propriétés précises, cela n'a eu pour conséquence que de nous priver de propriétés complexes.

Nous avons donc fait tourner daTac pendant plusieurs heures sur chaque spécification d'algèbre. Les résultats ont été présentés dans [78], et sont décrits dans la Figure 3.2 après avoir refait récemment toutes les exécutions.

Cette table décrit, à partir d'un ensemble initial d'équations, combien ont été engendrées, et combien ont été gardées à la fin. Les autres statistiques sont plus informatives, mais montrent que les techniques de simplification et d'élimination (ainsi que les bornes posées) ont été très efficaces.

Essayer d'étudier ces milliers de propriétés n'a plus grand sens. Pourtant les premières exécutions n'ayant donné « que » quelques centaines de propriétés, elles nous ont été très utiles pour

	$R4$	$R5$	$S4$	$S5$
Équations initiales	18	19	20	21
Équations engendrées	1 851 101	3 639 555	990 784	20 365 712
Équations finales	28 466	5 440	15 674	5 522
Nbre de déductions	346 213	210 809	249 612	246 222
Nbre de simplifications	1 466 064	1 281 971	2 203 321	2 567 921
Nbre de suppressions	1 822 635	3 634 115	975 110	20 360 190
Temps en secondes	111 964	10 638	53 949	27 345
Bornes (taille/nb var.)	30/4	30/4	30/4	30/4

FIG. 3.2 – Étude des algèbres approximantes et modales.

trouver des schémas de propriétés, comme par exemple :

$$\begin{aligned} &I(C(x_1)\cup\ldots\cup C(x_n)\cup I(y_1)\cup\ldots\cup I(y_m)) \\ &\qquad\qquad\approx\quad C(x_1)\cup\ldots\cup C(x_n)\cup I(y_1)\cup\ldots\cup I(y_m) \\ &C(C(x_1)\cap\ldots\cap C(x_n)\cap I(y_1)\cap\ldots\cap I(y_m)) \\ &\qquad\qquad\approx\quad C(x_1)\cap\ldots\cap C(x_n)\cap I(y_1)\cap\ldots\cap I(y_m) \end{aligned}$$

Et Anita Wasilewska était enchantée de lire toutes ces propriétés d'algèbres qu'elle avait définies elle-même...

3.3.3 Comparaison des algèbres modales et approximantes

Nous avons vu que l'étude des quatre algèbres a permis d'engendrer de très nombreuses propriétés pour chacune. Mais ces algèbres étant très proches de par leur définition, nous avons donc décidé de les comparer en comparant les ensembles de propriétés engendrés. Pour cela, nous avons mis au point une procédure permettant de comparer deux ensembles de clauses toujours à l'aide de daTac. Cette procédure, décrite ci-dessous, est basée sur des techniques de subsomption et de simplification, mais aussi sur l'étude des inférences ayant engendré les propriétés.

Procédure de comparaison de deux algèbres.

Soient A et B deux algèbres représentées chacune par un ensemble de clauses en logique du premier ordre. Pour comparer ces algèbres, nous appliquons la technique suivante, à l'aide du démonstrateur daTac :

1. Premièrement, pour chaque algèbre, nous déduisons un grand nombre de propriétés. Soient A_1 et B_1 les ensembles obtenus pour A et B respectivement.
2. Étant donné A_1, nous essayons de charger chaque clause de B_1. Certaines de ces clauses sont immédiatement supprimées, car subsumées par des clauses de A_1. D'autres clauses subsument des clauses de A_1. Soient A_2 et B_2 les ensembles de clauses persistantes de A_1 et B_1 respectivement.
3. Nous appliquons la même opération avec B_1 pour ensemble initial, en chargeant chaque clause de A_1. Soient A_3 et B_3 les ensembles de clauses persistantes de A_1 et B_1 respectivement.
4. Le premier résultat important est l'*intersection entre A_1 et B_1*. En effet, l'ensemble des propriétés représentant $A_1 \cap B_1$ est défini par : $A_2 \backslash A_3$, équivalent à $B_3 \backslash B_2$. Cette méthode

pour obtenir l'intersection de deux ensembles peut sembler compliquée, mais l'unique autre moyen serait une extraction manuelle. Or, étant donné les milliers de clauses concernées, le renommage des variables et les permutations sous les opérateurs associatifs et commutatifs, deux clauses équivalentes peuvent être difficiles à identifier.

5. Étant donnés A_2 et B_2, nous supprimons un maximum de clauses de B_2 par étapes de simplification. Soit B_4 l'ensemble des clauses persistantes de B_2.
   Étant donnés A_3 et B_3, nous supprimons un maximum de clauses de A_3 par étapes de simplification. Soit A_4 l'ensemble des clauses persistantes de A_3.
6. Maintenant, nous avons toutes les informations nécessaires pour trier les propriétés A_1 de A : les clauses de $A_2 \backslash A_3$ sont communes avec B_1 ; les clauses de $A_3 \backslash A_4$ sont des *tautologies* pour B_1 ; les clauses restantes, A_4, sont *candidates pour être des propriétés pures de A*.
7. On peut aussi trier les propriétés B_1 de B : les clauses de $B_3 \backslash B_2$ sont communes avec A_1 ; les clauses de $B_2 \backslash B_4$ sont des *tautologies* pour A_1. Finalement, les clauses de B_4 sont *candidates pour être de pures propriétés de B*.
8. On peut étudier plus précisément les candidates pour être purement A (resp. B). daTac ne fait pas qu'engendrer des clauses, il permet aussi de retrouver facilement leur preuve détaillée. Donc on peut vérifier la preuve de chaque clause de A_4 (resp. B_4). Les propriétés dont la preuve n'utilise que des clauses communes à A_1 et B_1 ne sont pas purement dans A (resp. B). Les autres sont dites être de *fortes candidates pour être pures dans A (resp. B)*.
   À noter que si $A \subseteq B$, soit A_4 est vide, soit nous pouvons vérifier que chaque clause de A_4 est démontrée en utilisant uniquement des clauses communes à A_1 et B_1.
9. Enfin, pour démontrer qu'une propriété P est pure dans une théorie A (resp. B), il suffit d'arriver à démontrer les propriétés de A (resp. B) à partir de $B \cup \{P\}$ (resp. $A \cup \{P\}$).

Cette procédure a été appliquée pour comparer les quatre algèbres étudiées. Cela a permis d'isoler un grand nombre de propriétés « pures », et c'est l'étude de ces propriétés-là qui nous a été très utile pour bien comprendre ces différentes algèbres. Les résultats de ces comparaisons

A vs B	$A \cap B$	*triviales A*	*futures B*	*pures A*
R4 vs R5	301	367	16	–
R5 vs R4	301	0	63	108
R4 vs S4	130	0	554	–
S4 vs R4	130	51	0	682
R5 vs S5	126	0	346	–
S5 vs R5	126	46	0	504
S4 vs S5	469	383	11	–
S5 vs S4	469	11	151	45

FIG. 3.3 – Comparaison des algèbres approximantes et modales.

sont décrits dans la Figure 3.3. Nous avons donc comparé les algèbres deux à deux. Une ligne de ce tableau s'interprète ainsi : en comparant *A vs B*, la colonne $A \cap B$ représente le nombre de propriétés communes à A et B ; la colonne *triviales A* montre le nombre de clauses de A qui sont subsumées par B ou simplifiées en une tautologie par les propriétés de B ; la colonne *futures B* donne le nombre de clauses de A qui ont été engendrées uniquement par des clauses de $A \cap B$ (cela signifie que ces propriétés auraient été engendrées par B si nous n'avions pas

arbitrairement arrêté l'exécution ; la dernière colonne contient le nombre de propriétés pures dans A, c'est-à-dire de propriétés dans A qui ne sont pas des propriétés dans B ; il n'y a pas de valeur dans cette colonne si $A \subseteq B$.

3.4 Ensembles approximants généralisés

La notion d'égalité approximante est habituellement restreinte aux ensembles approximants. Nous avons étendu cette notion aux espaces topologiques approximants et aux ensembles approximants généralisés (espaces topologiques ayant la propriété clopen, $C(I(x)) \approx I(x)$), et en conséquence aux algèbres topologiques et booléennes approximantes, respectivement.

Étant donnée une algèbre topologique (approximante) $\mathcal{A}$, nous avons répondu aux questions suivantes : l'égalité topologique (approximante) est-elle une congruence par rapport aux opérations de $\mathcal{A}$? Peut-on définir une algèbre $\mathcal{B}$ similaire à $\mathcal{A}$ telle que l'égalité topologique soit une congruence par rapport à toutes ses opérations ?

Nous avons montré que l'égalité topologique est une congruence uniquement pour l'opération de complément des algèbres topologiques (S4, par exemple).

L'égalité approximante est une congruence pour les opérations de complément, d'intérieur et de clôture pour les algèbres topologiques approximantes (S5, par exemple)

Enfin, il est possible de définir, à partir d'une algèbre topologique approximante, une algèbre similaire pour laquelle l'égalité approximante est une congruence pour toutes ses opérations. Nous avons également montré que l'algèbre quotientée obtenue est une algèbre quasi-booléenne approximante (R5).

Mais pour obtenir ces résultats, nous avons d'abord étudié les travaux effectués sur ces différentes algèbres, et en particulier ceux de Banerjee [68, 67]. Ceux-ci étant basés sur un raisonnement dans la logique modale S5, et les esquisses de preuves suivant un raisonnement sémantique sur cette logique, il nous a été impossible de les vérifier, et nous en sommes même arrivés à douter des résultats annoncés.

Afin d'essayer de comprendre tout cela, et de nous faire une meilleure idée sur la correction des résultats, nous avons mis au point une méthode d'analyse graphique, appelée diagrammes approximants. Il s'agit de diagrammes à la Ven, permettant de représenter très simplement les ensembles et les classes d'équivalence sur les ensembles. Nous avons donc défini graphiquement des contre-exemples de propriétés non satisfaites par ces algèbres, puis avons essayé de comprendre les travaux de Banerjee grâce à ces diagrammes. Enfin, une fois convaincus de leur correction, nous avons réussi à démontrer ces différents résultats entièrement automatiquement à l'aide de daTac.

Les Figures 3.4 et 3.5 montrent que l'égalité approximante n'est pas une congruence pour les opérations d'union et d'intersection standard.

En observant ces diagrammes, nous avons décidé d'essayer de définir graphiquement de nouveaux opérateurs d'union et d'intersection. C'est ainsi que nous avons obtenu les diagrammes des Figures 3.6 et 3.7 qui peuvent être algébriquement retranscrits en :

$$a \sqcup b = a \cup Ib \cup (b \cap \neg I(a \cup b))$$
$$a \sqcap b = (a \cap b) \cup (a \cap Cb \cap \neg C(a \cap b))$$

Ces nouveaux opérateurs assez surprenants consistent simplement à appliquer quelques transfor-

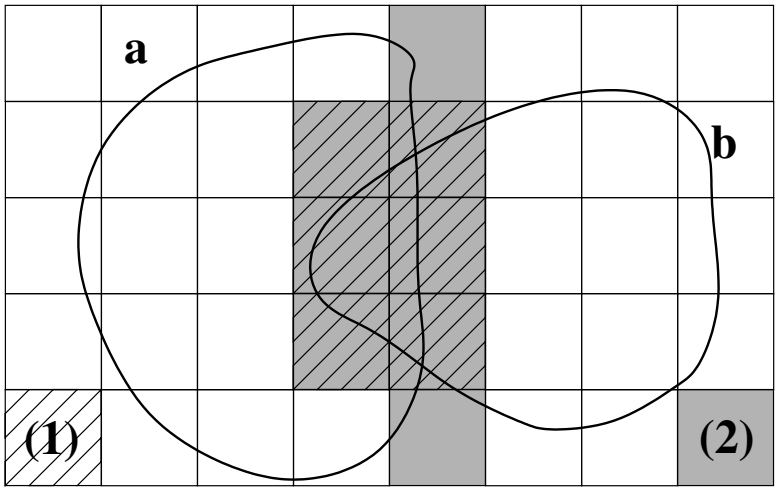


FIG. 3.4 – **(1)** : $C(a \cap b)$ – **(2)** : $Ca \cap Cb$

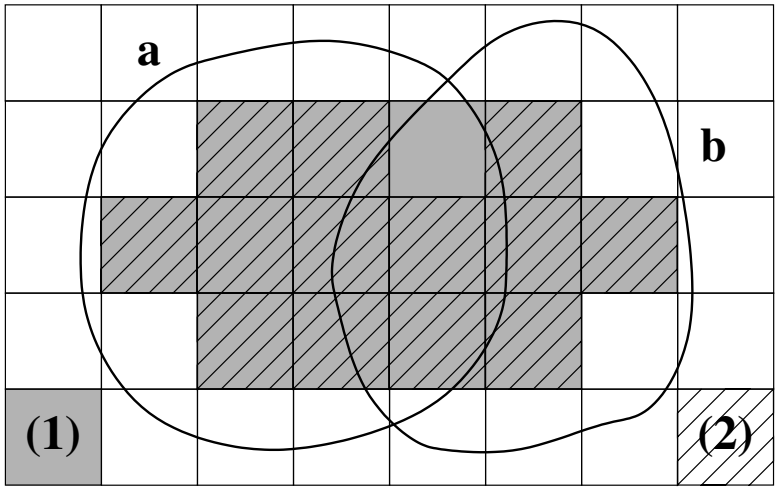


FIG. 3.5 – **(1)** : $I(a \cup b)$ – **(2)** : $Ia \cup Ib$

mations sur les opérateurs classiques : ajouter un morceau pour l'intersection ; ôter un morceau pour l'union.

Nous nous sommes alors rendus compte que ces opérateurs sont équivalents à ceux définis par Banerjee.

Nous avons aussi essayé d'autres définitions préservant la commutativité (ce qui n'est pas le cas de nos deux opérateurs comme cela peut facilement se voir sur les diagrammes), comme par exemple :

$$a \sqcup b = (a \cap b) \cup Ia \cup Ib \cup ((a \cup b) \cap \neg I(a \cup b))$$
$$a \sqcap b = (a \cap b) \cup (a \cap Cb \cap \neg C(a \cap b)) \cup (b \cap Ca \cap \neg C(a \cap b))$$

Mais cette variante, bien que « plus jolie » ne satisfait pas des propriétés importantes, comme illustré dans la diagramme de la Figure 3.8 où les ensembles a et b ont une intersection vide.

Nous avons donc étudié l'algèbre construite avec les nouveaux opérateurs d'union et d'intersection, quotientée par la congruence formée par l'égalité approximante, $\mathcal{B}_r = (A/\approx_r, \sqcap, \sqcup, \neg, I, C)$, et nous avons démontré les propriétés suivantes :

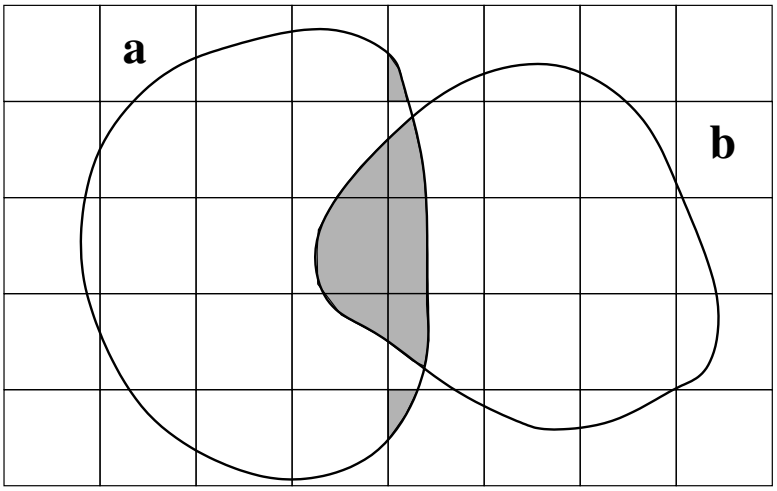


FIG. 3.6 – Nouvelle intersection ($\sqcap$)

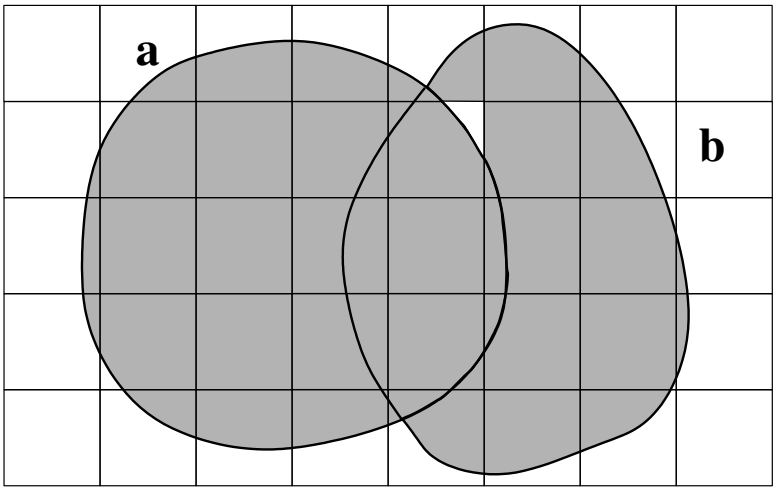


FIG. 3.7 – Nouvelle union ($\sqcup$)

- $(A/\approx_r, \sqcup, \sqcap, 0, 1)$ est un treillis distributif avec 0 et 1.
- Pour tous $a, b \in A/\approx_r$, $\neg(a \sqcup b) = (\neg a \sqcap \neg b)$,
- Pour tout $a \in A/\approx_r$, $\neg\neg a = a$.
- L'algèbre congruente $\mathcal{B}_r$ n'est pas une algèbre booléenne, car il existe un élément $a \in A/\approx_r$ tel que $\neg a \sqcap a \neq 0$ et $\neg a \sqcup a \neq 1$ (cf. Figure 3.9).
- Pour tous $a, b \in A/\approx_r$, $I(a \sqcap b) = Ia \sqcap Ib$, $I(a \sqcup b) = Ia \sqcup Ib$, $Ia \leq a$, $IIa = Ia$, $I1 = 1$, et $CIa = Ia$, où $Ca = \neg I \neg a$ et $\leq$ est un ordre de treillis.

Enfin, nous avons mis au point une procédure de décision permettant de tester si une formule est une propriété de l'algèbre topologique approximante S5 [80]. Cette procédure, construite en suivant le fonctionnement des diagrammes approximants, a été implantée par un étudiant d'Anita Wasilewska, Max Lifantsev, et donne d'excellents résultats lorsque le nombre de variables différentes reste raisonnable (inférieur ou égal à 5). Lorsqu'une formule n'est pas une propriété de l'algèbre, des contre-exemples sont listés.

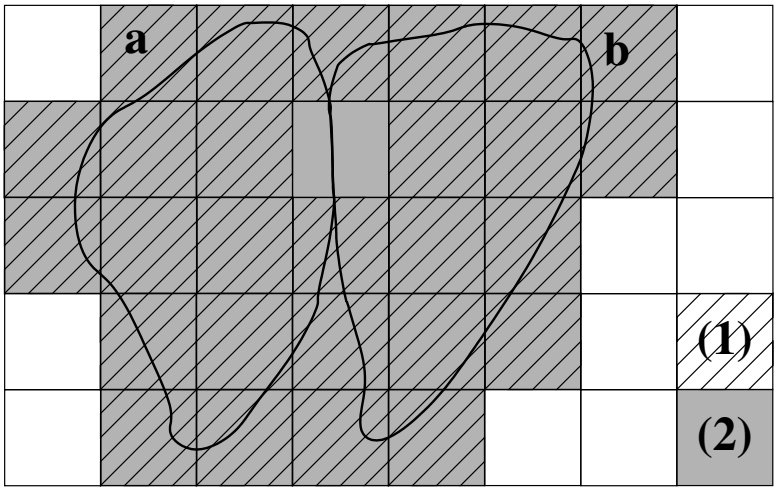


FIG. 3.8 – **(1)** : $C(a \sqcup b)$, **(2)** : $Ca \sqcup Cb$

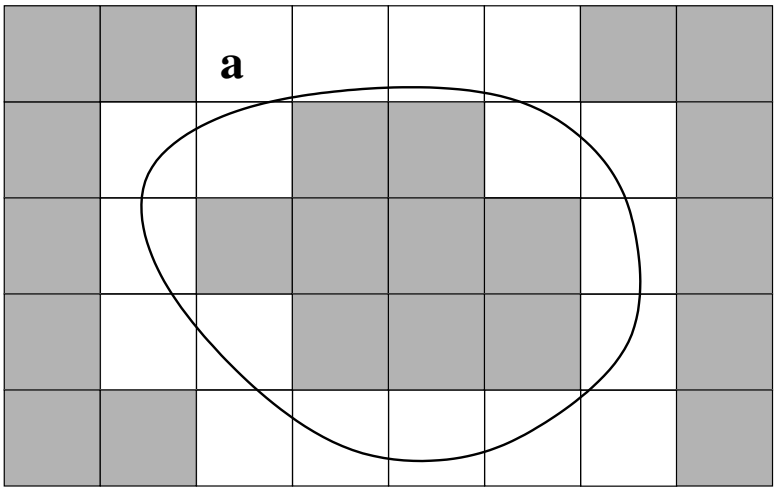


FIG. 3.9 – $I(a \sqcup \neg a)$ $(\neq I1 = 1)$

3.5 Conclusion

Nous avons présenté dans ce chapitre un projet qui a été réalisé en collaboration avec Anita Wasilewska. Initié suite à des travaux purement théoriques sur différentes algèbres topologiques liées aux ensembles approximants, il a permis à la fois de bien comprendre ces travaux, d'obtenir des preuves complètes et compréhensibles des résultats énoncés, mais il a surtout permis de montrer l'intérêt d'utiliser un outil automatique de démonstration pour engendrer ou démontrer de nouvelles propriétés. Enfin, une seconde approche complémentaire basée sur les diagrammes approximants a permis de mettre au point une procédure de décision faisant ressortir des contre-exemples lorsque des formules ne sont pas satisfaites.

Chapitre 4

Protocoles de sécurité : langages, sémantiques et outils

La cryptographie est utilisée depuis la nuit des temps pour protéger des informations lors de leur échange. Cependant seule, elle ne peut être suffisante, et un protocole d'échange doit être mis au point pour guider les actions de chaque parti, normalisant ainsi la communication.

La définition de protocoles de sécurité est une tâche difficile, car il s'agit de décrire les messages devant être échangés entre divers partis pour satisfaire une certaine propriété de sécurité (confidentialité d'une information, ou authentification des partis, par exemples). La cryptographie n'est que l'un des outils mis à la disposition des concepteurs de protocoles.

La vérification des protocoles de sécurité, c'est-à-dire la démonstration qu'ils garantissent bien les propriétés pour lesquelles ils ont été conçus, est également une tâche très difficile.

Dans ce chapitre, nous n'abordons que le premier point : la définition de protocoles. Le chapitre suivant sera consacré à la vérification des protocoles.

Nous décrivons donc dans ce chapitre l'évolution des langages de spécification de protocoles, partant de spécifications simplistes utilisées dans les articles scientifiques, pour aboutir à des spécifications orientées programmation requises par les industriels. Nous illustrons ces deux principales vues par les langages sur lesquels nous avons travaillé (dans l'équipe CASSIS du LORIA), pour les logiciels Casrul/AVISS et AVISPA, et décrivons leur formalisation pour pouvoir à la fois simuler automatiquement les protocoles, mais aussi les analyser (toujours automatiquement).

L'article référence de ce chapitre est celui publié à la conférence internationale LPAR (Logic for Programming and Automated Reasoning) en 2000 [110]. Il représente la première avancée importante dans la mise en œuvre de méthodes et outils basés sur la déduction automatique, pour spécifier et analyser des protocoles de sécurité. Ce chapitre décrit également les nombreux travaux sur la spécification et la sémantique de protocoles, et leur implantation, qui ont suivi cet article référence.

4.1 Les protocoles vus par les chercheurs : spécifications simplistes

Dans la littérature scientifique, les protocoles sont décrits dans un langage très simpliste, appelé Alice-Bob. En fait, il s'agit plus d'une notation que d'un langage normalisé, car il en existe de très nombreuses variantes. L'objectif de cette notation est d'apporter une description

très concise des protocoles[9], tout en préservant leur lisibilité.

Nous décrivons dans cette section les principes de cette notation, nous permettant en même temps de présenter les différents mécanismes applicables pour composer des protocoles de sécurité.

4.1.1 Notation Alice-Bob

Dans cette notation, l'accent est mis sur les informations échangées entre les participants du protocole, et une spécification se résume souvent à une séquence d'échanges décrits comme suit :

$$\text{Alice} \rightarrow \text{Bob} : \text{Message}$$

exprimant que l'agent Alice envoie Message à l'agent Bob. Noter que très souvent les noms des agents sont abrégés à leur initiale dans le protocole.

Le contenu des messages fait bien sûr appel à des primitives cryptographiques classiques dont nous détaillons ci-après les principales.

Concaténation. Un message est souvent composé de plusieurs éléments mis bout à bout. Cette concaténation est en général traduite par :

$$\text{A} \rightarrow \text{B} : \text{Element}_1,\text{Element}_2,\ldots,\text{Element}_n$$

la virgule étant parfois remplacée par un point.

La cryptographie fait appel à des algorithmes de chiffrement des messages par des clefs. Il existe deux grandes catégories de chiffrements : symétrique et asymétrique, leur principale différence étant sur la distinction entre les algorithmes de chiffrement et de déchiffrement.

Chiffrement symétrique. Le chiffrement symétrique d'un message consiste à appliquer un algorithme de chiffrement utilisant une clef, et à utiliser cette même clef avec le même algorithme (ou un algorithme facilement dérivable du premier) pour le déchiffrement. C'est ainsi que par abus de langage la clef est dite symétrique. En général, les spécifications simplistes ne font pas état des algorithmes utilisés, et décrivent le chiffrement et le déchiffrement de manière identique, par la même clef.

$$\text{A} \rightarrow \text{B} : \{M\}_K$$

Le message envoyé par Alice est le résultat du chiffrement de M par la clef symétrique K. Pour déchiffrer ce message, Bob devra appliquer le même chiffrement avec la même clef sur le message reçu, car par définition : $\{\{M\}_K\}_K = M$.
Noter que la clef utilisée pour un chiffrement symétrique peut être une clef composée, c'est-à-dire construite comme un message à part entière. Par exemple, le triplet K, A, B peut représenter une clef.

Chiffrement asymétrique. Le chiffrement asymétrique fait appel à deux algorithmes différents : l'un sert à chiffrer, l'autre à déchiffrer. La clef utilisée pour chiffrer est une clef publique, c'est-à-dire pouvant être connue de tout agent. La clef utilisée pour déchiffrer est une clef privée, c'est-à-dire connue d'un seul agent (ou parfois aussi d'un serveur qui a créé ces clefs).

$$\text{A} \rightarrow \text{B} : \{M\}_K$$

Le message envoyé par Alice est le résultat du chiffrement de M par la clef publique de Bob K. Pour déchiffrer ce message, Bob devra appliquer un algorithme de déchiffrement avec la clef privée associée à la clef publique K, notée K^{-1} : $\{\{M\}_K\}_{K^{-1}} = M$.

[9] Il est bien connu que la place coûte très cher dans les articles scientifiques... ;-)

Signature. La signature d'un message correspond au chiffrement de celui-ci par une clef privée, ce qui le rend potentiellement déchiffrable par tout agent à l'aide de la clef publique associée.

$$\mathrm{A} \rightarrow \mathrm{B} : \{M\}_{K^{-1}}$$

Le message envoyé par Alice est le résultat du chiffrement de M par sa clef privée K^{-1}. Pour déchiffrer ce message, Bob devra appliquer un algorithme de déchiffrement avec la clef publique K de Alice : $\{\{M\}_{K^{-1}}\}_K = M$.

Dans les descriptions ci-dessus des différents modes de chiffrement et déchiffrement, nous avons utilisé le même nom de clef K afin de montrer l'ambiguïté de cette notation : comment distinguer un chiffrement symétrique d'un chiffrement asymétrique ? comment savoir si une clef K est une clef symétrique ou asymétrique ?

La distinction entre les modes de chiffrement se fait parfois par l'ajout d'une lettre précisant s'il s'agit d'un chiffrement symétrique, $\{M\}^s_K$, ou asymétrique, $\{M\}^p_K$ (utilisant donc une clef publique ou privée).

Mais cette distinction se fait souvent sur une simple identification de la clef : un chiffrement par une clef symétrique sera donc un chiffrement symétrique, et un chiffrement par une clef publique ou privée sera un chiffrement asymétrique[10]. L'essentiel est donc l'identification de la clef. Pour cela, les noms des agents sont souvent précisés dans la clef. Ainsi, la clef Ka représentera la clef publique de l'agent A, et donc Ka^{-1} sa clef privée correspondante. La clef Kab représentera une clef partagée par les agents A et B, donc une clef symétrique.

Fonction de hachage. En plus du chiffrement, la cryptographie fait souvent appel à des fonctions de hachage, c'est-à-dire des fonctions non inversible. Ainsi,

$$\mathrm{A} \rightarrow \mathrm{B} : H(M)$$

décrit l'envoi par Alice à Bob du résultat de l'application de la fonction H au message M, Bob étant dans l'impossibilité de retrouver la valeur de M, même s'il connaît la fonction H. Ces fonctions sont en général utilisées pour vérifier l'intégrité d'un message : si Bob a reçu M et $H(M)$, et connaît la fonction H, il pourra calculer $H(M)$ et vérifier que cela correspond bien à ce qu'il a reçu.

Nonce et timestamp. La qualité d'un protocole repose sur le chiffrement mis en œuvre, mais aussi sur sa fraîcheur : éviter qu'une exécution du protocole soit utilisée plusieurs fois, même en partie. Pour cela, des nombres aléatoires peuvent être utilisés dans les messages ; ces nombres, calculés au moment de l'exécution du protocole, ne sont donc (a priori) utilisés qu'une seule fois, d'où leur nom, *nonce* (***n**umber used only **once***). Ainsi, Na désignera un nonce créé par l'agent Alice. Une variante consiste à utiliser des *timestamps*, des marqueurs de date et heure pouvant avoir une validité limitée dans le temps.

Ces différentes notions sont mises en œuvre pour décrire les protocoles cryptographiques et bien sûr sont combinées entre elles. Les messages peuvent bien sûr être composés d'autres informations élémentaires, comme les noms des agents, du texte,... Des fonctionnalités plus avancées peuvent également être utilisées, comme des tables de clefs publiques, ou des opérateurs arithmétiques tels que l'exponentielle et le ou exclusif (souvent utilisés dans les algorithmes de chiffrement).

[10] Noter cependant l'ambiguïté suivante : a priori, une clef publique ou privée pourrait très bien être utilisée pour effectuer un chiffrement symétrique.

Exemple de spécification

Afin d'illustrer les notations qui viennent d'être décrites, voici un exemple de spécification de protocole. Il s'agit d'un protocole défini par Denning et Sacco [104].

$$\begin{aligned} &\mathrm{A} \rightarrow \mathrm{S} : A, B \\ &\mathrm{S} \rightarrow \mathrm{A} : \{B, Kab, T, \{A, Kab, T\}_{Kbs}\}_{Kas} \\ &\mathrm{A} \rightarrow \mathrm{B} : \{A, Kab, T\}_{Kbs} \end{aligned}$$

Ce protocole fait intervenir trois agents : Alice, Bob et un Serveur, chacun étant représenté par son initiale. Le premier message correspond à une demande d'Alice au Serveur pour l'obtention d'une clef symétrique à partager entre elle et Bob. Dans sa réponse, le serveur fournit le clef demandée (Kab) et utilise les clefs symétriques Kas (partagée avec Alice) et Kbs (partagée avec Bob), ainsi qu'un timestamp T. Le dernier message est la transmission par Alice à Bob d'une partie du message reçu du Serveur, permettant à Bob de récupérer la nouvelle clef Kab. Une fois cette clef en possession d'Alice et Bob, ces deux agents peuvent engager une conversation dont le contenu sera protégé par chiffrement à l'aide de cette clef.

Des dizaines de protocoles ont ainsi été spécifiés, et le sont encore dans de nombreux articles scientifiques. Un rapport de Clark et Jacob datant de 1997 [98] recense un grand nombre de protocoles de ce genre. Dans ce rapport, la seule variante de notation est le chiffrement $\{M\}_K$ noté $E(K : M)$.

Jusqu'à présent, nous n'avons pas mentionné la spécification de propriétés de sécurité à vérifier par les protocoles. Il s'avère que dans la plupart des articles scientifiques, ces propriétés sont exprimées en langage naturel. Nous allons cependant voir dans la section suivante que ces propriétés doivent effectivement être spécifiées, tout comme d'autres informations supplémentaires, dès qu'il est question d'utiliser un outil d'analyse du protocole.

4.1.2 Spécification étendue pour les outils d'analyse

Comme nous venons de le voir, la notation Alice-Bob est une notation appropriée à la spécification rapide de protocoles relativement simples. Mais la spécification de protocoles ne peut se résumer à décrire les messages échangés. L'analyse d'un protocole demande une description précise, sans ambiguïté. Or, cette notation repose souvent sur des conventions de nommage des variables, pas toujours facile pour un outil...

Dans cette section, nous allons donc voir quelles informations peuvent être ajoutées dans une spécification, tout d'abord des informations relatives à l'exécution du protocole lui-même, puis relatives à l'analyse de propriétés de ce protocole. Nous illustrerons cela avec la spécification suivante du protocole EKE [89] pour notre logiciel Casrul [109].

```
Protocol EKE;
Identifiers
  A, B   : user;
  Na, Nb : number;
  Ka     : public_key;
  P,R    : symmetric_key;
Knowledge
  A: B,P;
  B: P;
Messages
```

```
  1.  A -> B:  {Ka}P
  2.  B -> A:  {{R}Ka}P
  3.  A -> B:  {Na}R
  4.  B -> A:  {Na,Nb}R
  5.  A -> B:  {Nb}R
Session_instances
[ A:a; B:b; P:p ];
Intruder divert, impersonate;
Intruder_knowledge a;
Goal correspondence_between A B;
```

a - Spécification du protocole lui-même

Le cœur de la description d'un protocole est la liste des messages échangés entre les agents. La notation Alice-Bob est utilisée pour le logiciel Casrul, avec deux variantes : le chiffrement est noté $\{M\}K$, et chaque message est précédé d'une étiquette indiquant son rang d'exécution.

Afin d'éliminer une grosse partie des ambiguïtés d'une spécification d'échanges de messages, une section de **déclaration des variables** utilisées est ajoutée. Elle permet ainsi de déclarer les variables intervenant dans les messages échangés, comme les noms d'agents, les nonces (*number*), les clefs publiques, les clefs symétriques, les fonctions de hachage,... Ces déclarations vont permettre de déduire pour chaque chiffrement apparaissant dans un message s'il est symétrique ou asymétrique.

L'autre information indispensable à la bonne compréhension d'un protocole est l'ensemble de **connaissances initiales** de chaque participant. Ainsi, dans notre exemple, l'agent A a pour connaissances initiales (en plus de son propre nom, connaissance implicite) le nom de l'agent B et la clef symétrique P. L'agent B, lui, ne connaît que la clef symétrique P.
Certaines informations apparaissant dans le protocole ne sont connues initialement d'aucun agent. Cela signifie que ces informations seront certainement engendrées par un agent au cours de l'exécution, et certainement apprises par d'autres agents. C'est le cas de Na, Nb, Ka et R dans notre exemple.

Ces déclarations supplémentaires, en plus de limiter l'ambiguïté de certaines expressions, offrent une plus grande liberté de nommage des variables aux utilisateurs.

b - Informations supplémentaires pour l'analyse

Le protocole spécifié, il reste à exprimer les propriétés à analyser, et le contexte dans lequel cette analyse doit avoir lieu.

La première chose est de spécifier le **scénario** sur lequel portera l'analyse. Cela s'exprime par la liste des sessions du protocole qu'il faudra exécuter en parallèle. Une session correspond à l'instanciation des variables initiales du protocole. Ainsi, dans notre exemple, l'instanciation

```
[ A:a; B:b; P:p ];
```

correspond à la session jouée par deux agents a et b, dans les rôles respectifs de A et B, partageant la clef symétrique p. Plusieurs instances peuvent bien sûr être spécifiées, certaines pouvant faire apparaître le nom de l'intrus I, agent malhonnête par définition.
Dans certains logiciels, comme AVISS [85] le successeur de Casrul, il est possible de spécifier une liste d'instances de rôles, une session du protocole étant alors la conséquence de l'exécution de plusieurs rôles en parallèle.

Pour analyser un protocole, il est indispensable de décrire le **comportement de l'intrus**, agent ayant pour but de perturber le protocole et de trouver des failles de sécurité. Ainsi, l'intrus peut avoir les comportements suivants :

`eaves_dropping` : lire le contenu des messages échangés ;
`divert` : détourner des messages de leur destination ;
`impersonate` : envoyer des messages sous l'identité d'un autre agent.

Il s'avère que la majeure partie des analyses s'effectue avec un intrus au maximum de sa puissance, c'est-à-dire pouvant appliquer les trois comportements précédents. Le modèle d'un tel intrus est celui décrit par Dolev et Yao [105], et est ainsi souvent appelé *intrus de Dolev-Yao*.
Une dernière information importante sur l'intrus doit être spécifiée : la liste de ses **connaissances initiales**. Comme l'intrus va agir sur les instances du protocole décrites dans le scénario, ses connaissances sont des constantes de ces instances ainsi que d'éventuelles connaissances personnelles.

Enfin, dernière partie de la spécification, et non la moindre, la liste des **propriétés** que le protocole est censé satisfaire. Une propriété proposée dans tout outil est le secret, c'est-à-dire la confidentialité d'une information, qui ne doit donc pas parvenir dans les connaissances de l'intrus.
Une autre propriété tout aussi importante est la correspondance entre deux agents A et B. Cela signifie que lorsqu'un agent A a terminé de jouer une session avec B, alors B a effectivement commencé une session avec A. Cette propriété est déclinée dans certains logiciels sous forme d'une authentification entre deux agents sur une information précise.

4.2 Une première formalisation

L'utilisation d'un langage simpliste pour spécifier des protocoles a des inconvénients. Cela provoque des ambiguïtés qui peuvent engendrer des erreurs lors de leur implantation, en raison d'une mauvaise interprétation de la spécification. Par exemple, à la réception d'un message $\{M\}_K$, l'agent est-il capable de reconnaître ce message parce qu'il connaît M et K ? ou bien parce qu'il a déjà reçu auparavant ce cipher (sans connaître K, ni peut-être M) ? ou bien l'agent va-t-il apprendre M parce qu'il est capable de déchiffrer le cipher reçu ? ou enfin l'agent ne peut-il rien reconnaître ?
Remarque : si K est une clef publique, l'agent est capable de déchiffrer ce cipher s'il connaît la clef privée correspondante ; or, cette clef privée n'est pas visible dans la spécification du protocole.

Il est donc indispensable de définir une sémantique opérationnelle très précise. Nous avons effectué ce travail avec comme premier objectif de vérifier qu'un protocole est bien exécutable, c'est-à-dire que ses participants sont bien capables de composer les messages qu'ils doivent envoyer. Cette sémantique est basée sur une gestion complète de l'évolution des connaissances des participants, permettant aussi de vérifier les éléments connus dans les messages reçus.

Mais la sémantique définie permet également de décrire sans ambiguïté le comportement de l'intrus, en plus de celui du protocole. Ci-dessous, nous ne détaillons pas totalement ce travail [110], mais décrivons les principales étapes.

4.2.1 Opérateurs de gestion des connaissances

La première tâche à réaliser est la définition des opérateurs qui vont permettre de gérer les connaissances de chaque participant du protocole. Ces opérateurs sont :

- **infer** : à partir d'un ensemble de connaissances, calculer l'ensemble des connaissances supplémentaires pouvant être découvertes, par exemple par déchiffrement ou décomposition ;
- **known** : recenser l'ensemble des connaissances d'un agent à une étape précise du protocole ; cette notion est importante, car il ne s'agit pas de permettre à un agent d'utiliser une information lors d'une étape, alors qu'il ne peut l'apprendre que plus tard ;
- **compose** : décrire comment, à partir de ses connaissances actuelles, un agent compose un message qu'il doit émettre ;
- **expect** : décrire, lors de la réception d'un message, ce qu'un agent peut vérifier (informations déjà connues) et ce qu'il apprend.

Ces différents opérateurs se basent sur la spécification fournie, c'est-à-dire sur les déclarations de connaissances initiales des agents, et sur les messages échangés. Ils permettent donc au final de décrire, pour chaque agent, quelles sont ses connaissances initiales (déclarées dans le protocole), acquises (reçues dans des messages) et engendrées (créées lors de la composition de messages), et ceci à chaque étape du protocole.
Noter que les messages non décomposables reçus font partie des connaissances acquises, car ils pourront néanmoins être utilisés par la suite pour composer un message, ou bien être décomposés lorsque l'information nécessaire pour cela sera acquise.

Nous décrivons ci-dessous les opérateurs **compose** et **expect**.

a - Composition d'un message émis

La composition d'un message M par un agent U à l'étape i du protocole, notée $\mathit{compose}(U, M, i)$, est définie par :

$$\begin{aligned}
\mathit{compose}(U, M, i) &= t \quad \text{si } M \text{ connu par } U \text{ et nommé } t \\
\mathit{compose}(U, \langle M_1, M_2\rangle, i) &= \langle \mathit{compose}(U, M_1, i), \mathit{compose}(U, M_2, i)\rangle \\
\mathit{compose}(U, \{M\}_{SK}, i) &= \{\mathit{compose}(U, M, i)\}_{\mathit{compose}(U,SK,i)} \\
\mathit{compose}(U, \{M\}_{K}, i) &= \{\mathit{compose}(U, M, i)\}_{\mathit{compose}(U,K^{-1},i)} \\
\mathit{compose}(U, \{M\}_{K^{-1}}, i) &= \{\mathit{compose}(U, M, i)\}_{\mathit{compose}(U,K,i)} \\
\mathit{compose}(U, f(M), i) &= \mathit{compose}(U, f, i)(\mathit{compose}(U, M, i)) \\
\mathit{compose}(U, M, i) &= \mathit{fresh}(M) \quad \text{si } M \text{ est frais} \\
\mathit{compose}(U, M, i) &= \texttt{Echec} \quad \text{sinon}
\end{aligned}$$

Noter qu'une information fraîche est une information devant être engendrée spécifiquement pour composer ce message (exemples : un nonce, un clef, une identité,...). Enfin, si un élément du message ne peut être composé, cela provoque l'échec de la procédure.

b - Préparation d'un message attendu

Le squelette d'un message M attendu par un agent U à l'étape i du protocole, noté $\mathit{expect}(U, M, i)$, est composé ainsi :

$$\begin{aligned}
\mathit{expect}(U, M, i) &= \mathit{compose}(U, M, i) \quad \text{si pas } \texttt{Echec} \\
\mathit{expect}(U, \langle M_1, M_2\rangle, i) &= \langle \mathit{expect}(U, M_1, i), \mathit{expect}(U, M_2, i)\rangle \\
\mathit{expect}(U, \{M\}_{SK}, i) &= \{\mathit{expect}(U, M, i)\}_{\mathit{compose}(U,SK,i)} \\
\mathit{expect}(U, \{M\}_{K}, i) &= \{\mathit{expect}(U, M, i)\}_{\mathit{compose}(U,K^{-1},i)^{-1}} \\
\mathit{expect}(U, \{M\}_{K^{-1}}, i) &= \{\mathit{expect}(U, M, i)\}_{\mathit{compose}(U,K,i)^{-1}} \\
\mathit{expect}(U, M, i) &= x^U_M \quad \text{sinon}
\end{aligned}$$

L'objectif est d'identifier un maximum d'informations reconnaissables dans le message attendu. Cependant, si un élément est réellement inconnu, il sera donc appris et on lui donne un nom (x_M^U).

4.2.2 Sémantique opérationnelle

La sémantique partiellement décrite dans cette section et entièrement définie dans le papier présenté à LPAR en 2000 [110], et est maintenant une référence maintes fois citée. Cette sémantique montre comment les agents peuvent vérifier les morceaux de messages qu'ils connaissent déjà, comment ils peuvent utiliser de nouvelles connaissances pour décrypter d'anciens messages, et comment ils composent un nouveau message à partir de toutes ces informations.

a - Sémantique des échanges de messages

La sémantique opérationnelle que nous avons définie consiste à représenter le protocole et ses échanges de messages sous forme de règles de narrowing de la forme

$$\text{message reçu} \Rightarrow \text{message émis}$$

exprimant ainsi qu'une étape du protocole consiste toujours à décrire qu'un agent attend un message, et lorsqu'il le reçoit, émet une réponse.

Ces règles de narrowing sont bien sûr assez complexes car elles décrivent l'état des connaissances de l'agent au moment de l'attente, le squelette du message attendu, puis comment le message réponse est composé ainsi que l'évolution des connaissances de l'agent. Ainsi, pour une séquence de messages

$\mathrm{T} \rightarrow \mathrm{U} : M_i$
$\mathrm{U} \rightarrow \mathrm{V} : M_{i+1}$

la règle de narrowing concernant l'agent U sera de la forme :

$$\mathit{known}(U, i), \mathit{expect}(U, M_i, i) \Rightarrow \mathit{compose}(U, M_{i+1}, i+1), \mathit{known}(U, i+1)$$

Bien sûr, pour un protocole de n messages, M_0 et M_{n+1} seront vides.

Ces règles permettent de vérifier si un protocole est exécutable, c'est-à-dire si chaque participant est capable de composer les messages qu'il est censé envoyer, à l'aide de ses connaissances au moment de l'envoyer. Ainsi, un protocole est exécutable si le calcul de ces règles n'engendre pas d'échec. D'autre part, ces règles peuvent permettre de détecter des erreurs de spécification, par exemple en constatant qu'une information est considérée comme fraîche dans une transition alors que le concepteur du protocole la supposait connue.

Exemple 8 *Calculons ces transitions (restreintes aux opérations expect et compose) pour tester l'exécutabilité du protocole EKE. Les connaissances des agents seront nommées A_M pour l'information M selon l'agent A. Une information M apprise par un agent A lors de la réception d'un message sera nommée x_M^A comme prévu dans la définition de l'opérateur expect.*
Selon la spécification du protocole, les connaissances initiales de A sont A_A, A_B et A_P, et celles de B sont B_B et B_P.

- *Le premier agent actif est A, qui transmet le premier message à l'agent B : $\{Ka\}_P$. L'opérateur compose appliqué à ce message permet de construire la transition suivante :*

  $$\Rightarrow \{\mathit{fresh}(Ka)\}_{A_P}$$

car Ka n'étant pas dans les connaissances initiales de A, il est considéré comme créé spécialement ; le résultat de fresh(Ka) sera stocké sous le nom A_{Ka}. La clef P, elle, fait partie des connaissances initiales de A.

- *L'agent B reçoit donc ce premier message, lui permettant d'apprendre la valeur de Ka, puis répond à A en lui envoyant le message $\{\{R\}_{Ka}\}_P$.*

  $$\{x^B_{Ka}\}_{B_P} \Rightarrow \{\{fresh(R)\}_{x^B_{Ka}}\}_{B_P}$$

  La valeur apprise de Ka est utilisée dans la réponse, et la clef symétrique R est engendrée pour la réponse (et sera mémorisée sous le nom B_R).
- *L'agent A apprend la valeur de R dans le message reçu, et l'utilise pour chiffrer un nonce spécialement engendré dans la réponse.*

  $$\{\{x^A_R\}_{A_{Ka}}\}_{A_P} \Rightarrow \{fresh(Na)\}_{x^A_R}$$
- *L'agent B apprend la valeur de Na dans le message reçu, et pour la réponse engendre à son tour un nonce Nb et utilise ces deux nonces dans la réponse à A.*

  $$\{x^B_{Na}\}_{B_R} \Rightarrow \{x^B_{Na}, fresh(Nb)\}_{B_R}$$
- *L'agent A prend connaissance du nonce Nb, et peut transmettre le dernier message à B.*

  $$\{A_{Na}, x^A_{Nb}\}_{A_R} \Rightarrow \{x^A_{Nb}\}_{A_R}$$
- *Enfin, l'agent B vérifie le dernier message.*

  $$\{B_{Nb}\}_{B_R} \Rightarrow$$

La construction des transitions de ce protocole n'a pas échoué, donc le protocole est exécutable. D'autre part, quatre informations sont fraîchement engendrées au cours de l'exécution : Ka, R, Na et Nb.

b - Sémantique du comportement de l'intrus

Le comportement de l'intrus peut également être exprimé sous forme de règles de narrowing. Ces règles vont caractériser les actions suivantes :

- L'ajout à ses connaissances des messages qui lui sont destinés.
- S'il a la capacité d'écouter le réseau, l'ajout à ses connaissances de tout message transmis.
- S'il a la capacité d'intercepter des messages, l'ajout à ses connaissances de tout message transmis et la suppression de celui-ci du réseau.
- La décomposition de ses connaissances (qui correspond à l'opérateur **infer**).
- La composition de messages à partir de ses connaissances.
- L'envoi d'un message en son propre nom.
- L'envoi d'un message sous l'identité d'un autre agent, s'il en a la capacité.

Toutes ces actions sont décrire totalement indépendamment du protocole considéré, ce qui garantit la généricité du comportement de l'intrus.

Bien sûr, pour plus d'efficacité, la composition et l'envoi de messages par l'intrus peuvent être « guidés » par le protocole, en rajoutant des règles contrôlant qu'un message composé correspond au squelette d'un message attendu par un agent.

c - Recherche d'attaque

L'analyse de propriétés de sécurité s'exprime également indépendamment du protocole considéré. L'objectif étant de rechercher des attaques, cela se traduit par exemple,

– pour un problème de confidentialité, par l'ajout d'un prédicat déclarant l'information comme secrète, au moment où elle est engendrée dans le protocole, et l'ajout d'une clause spécifiant qu'une information déclarée secrète ne peut être dans les connaissances de l'intrus ;
– pour un problème de correspondance entre deux agents, par l'ajout d'une clause spécifiant que si un agent A a terminé une session (officiellement avec un agent B), alors B a bien commencé une session avec A.

4.2.3 Casrul, un outil pour chercheurs

Dans la lignée des travaux de Meadows [115] qui est la première à avoir utilisé du narrowing pour analyser des protocoles cryptographiques, nous (avec Florent Jacquemard et Michaël Rusinowitch) avons défini une représentation clausale des protocoles, afin de les analyser avec daTac. Ces « clauses » peuvent en fait être vues comme des règle de réécriture (ou de transformation) décrivant l'action à effectuer (en général l'envoi d'un nouveau message) à la réception d'un message attendu, correspondant exactement à la sémantique opérationnelle décrite dans la section précédente.

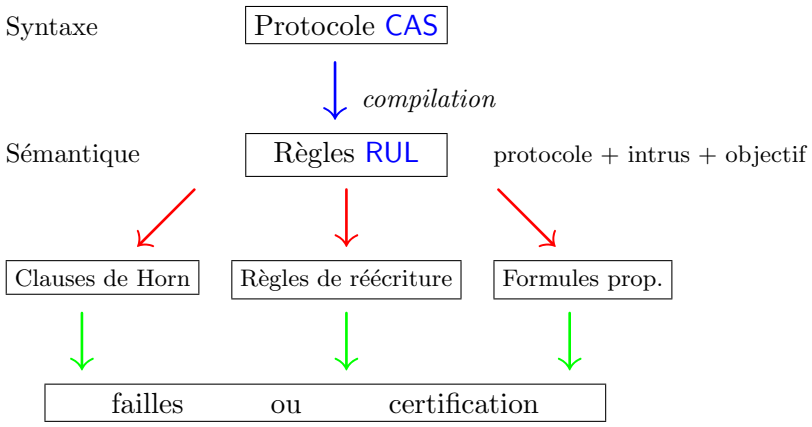


FIG. 4.1 – Principes du logiciel Casrul

Pour avoir un environnement homogène et générique, nous avons défini un langage de description de protocoles très simple, basé sur la notation Alice-Bob comme la plupart des outils existant alors. Nous avons également implanté un traducteur de ce langage vers des clauses fournies à daTac. Il a représenté la tâche la plus difficile, car il implante la transformation d'une spécification très simpliste en un ensemble de clauses suivant la sémantique opérationnelle partiellement décrite dans la section précédente. Les clauses engendrées représentent donc

– les échanges de messages, incluant l'évolution des connaissances des agents concernés ;
– l'état initial de chaque agent, pour chaque session jouée du protocole ;
– les interventions de l'intrus (lecture d'un message, interception d'un message, envoi d'un message en son nom ou sous une autre identité), selon ses capacités décrites dans la spécification ;
– la composition d'un message par l'intrus à partir de ses connaissances ;
– les propriétés vérifiées.

Le traducteur fait également office de compilateur, pour détecter toute erreur de spécification, syntaxique ou sémantique ; par exemple, il vérifie que le protocole est bien exécutable.

Le logiciel daTac a été utilisé car il permet de manière totalement transparente de représenter

le non-déterminisme du choix d'une transition à partir d'une liste de messages, représentée grâce à un opérateur associatif et commutatif. D'autre part, lors de la découverte d'une attaque, une trace des déductions effectuées pour l'atteindre est affichée, ce qui permet de comprendre sans trop de difficulté le scénario joué.
Le logiciel obtenu, appelé Casrul [11], permet d'analyser les protocoles de manière entièrement automatique, avec un modèle d'intrus prédéfini dans la spécification initiale, mais qui reste indépendant du protocole.
Casrul a subi de nombreuses évolutions (décrites dans le chapitre suivant), en particulier sous le nom AVISS en raison du projet européen éponyme. Les résultats obtenus sur de nombreux protocoles sont décrits dans la Table 4.1, montrant le type d'attaque trouvée et le temps (en secondes).

Le premier défaut de Casrul est sa lenteur. En effet, la traduction d'une spécification en un ensemble de clauses est extrêmement rapide, mais daTac est un outil de déduction très générique, pas du tout optimisé pour ce type d'analyse, qui met en œuvre des mécanismes d'unification, de filtrage et de subsomption pouvant s'avérer très compliqués.

Son second défaut concerne la spécification des protocoles. Le langage basé sur la notation Alice-Bob est un avantage au niveau lisibilité, mais il est relativement ambigu, car ne montrant pas l'évolution des connaissances des participants. D'autre part, il est très loin des problèmes d'implantation que peuvent rencontrer les concepteurs de protocoles industriels.

La section suivante revient sur ces problèmes et décrit nos travaux pour tenter de les résoudre.

4.3 Les protocoles vus par les industriels : spécifications programmatoires

Les travaux sur l'analyse de protocoles cryptographiques auxquels j'ai participé ont d'abord concerné l'automatisation de l'analyse par narrowing, avec pour implantation le logiciel Casrul. Mais ils ont rapidement été poursuivis par des collaborations dans le cadre de projets européens. Le premier projet, AVISS, avec les universités de Fribourg (Allemagne) et Gênes (Italie), s'est centré sur notre outil Casrul, étendant un peu le langage de spécification et greffant d'autres outils d'analyse (en plus de daTac). D'énormes avancées ont également été réalisées dans les méthodes d'analyse (cf. Chapitre 5). Le logiciel résultat, AVISS [85], a été assez peu diffusé.

Un second projet européen, AVISPA, a suivi avec les mêmes partenaires plus Siemens Munich. L'arrivée d'un industriel, membre de l'IETF, a révolutionné les objectifs en mettant en avant les besoins des concepteurs de protocoles. Cette section décrit ce bouleversement.

4.3.1 Limites des spécifications à la Alice-Bob

Les spécifications de protocoles utilisant un langage simple comme la notation Alice-Bob sont très faciles à écrire et globalement faciles à comprendre. Leur taille très réduite est également pratique, par exemple pour les intégrer dans un article. Cependant, elles masquent d'énormes défauts.

Composition et réception des messages : La spécification d'un message se résume à donner son contenu précis, ainsi que le nom de l'agent émetteur et celui du récepteur. Par exemple, le

[11] Le nom aurait dû être CASROL, clin d'œil aux divers logiciels similaires contenant souvent les lettres CAS (*Common Abstract Syntax*), mais nous n'avons pas osé conserver ce nom, bien que son logo soit effectivement une casserole, de peur d'avoir à le traîner longtemps derrière nous. . . ;)

TAB. 4.1 – Performances de Casrul dans le cadre du projet AVISS

Protocole	**Attaque**	**Casrul**
ISO symmetric key 1-pass unilateral authentication	Replay	1.98
ISO symmetric key 2-pass mutual authentication	Replay	3.86
Andrew Secure RPC Protocol	Type flaw	4.26
	Replay	32.74
ISO CCF 1-pass unilateral authentication	Replay	2.23
ISO CCF 2-pass mutual authentication	Replay	4.55
Needham-Schroeder Conventional Key	Replay STS	63.43
Denning-Sacco (symmetric)	Type flaw	15.98
Otway-Rees	Type flaw	10.71
Yahalom	Type flaw	44.08
Woo-Lam Π_1	Type flaw	0.81
Woo-Lam Π_2	Type flaw	0.80
Woo-Lam Π_3	Type flaw	0.82
Woo-Lam Π	Parallel-session	1074.95
Woo-Lam Mutual Authentication	Parallel-session	245.56
Needham-Schroeder Signature protocol	Man-in-middle	53.88
**Neuman Stubblebine initial part*	Type flaw	6.19
**Neuman Stubblebine repeated part*	Replay STS	3.54
Neuman Stubblebine (complete)	Type flaw	46.78
Kehne Langendorfer Schoenwalder (repeated part)	Parallel-session	199.43
Kao Chow Repeated Authentication, 1	Replay STS	76.82
Kao Chow Repeated Authentication, 2	Replay STS	45.25
Kao Chow Repeated Authentication, 3	Replay STS	50.09
ISO public key 1-pass unilateral authentication	Replay	4.23
ISO public key 2-pass mutual authentication	Replay	11.06
**Needham-Schroeder Public Key*	Man-in-middle	12.91
Needham-Schroeder Public Key with key server	Man-in-middle	TO
**Needham-Schroeder with Lowe's fix*	Type flaw	31.12
SPLICE/AS Authentication Protocol	Replay	352.42
Hwang and Chen's modified SPLICE	Man-in-middle	13.10
Denning Sacco Key Distribution with Public Key	Man-in-middle	936.90
Shamir Rivest Adelman Three Pass Protocol	Type flaw	0.70
Encrypted Key Exchange	Parallel-session	240.77
Davis Swick Private Key Certificates, protocol 1	Type flaw	106.15
	Replay	TO
Davis Swick Private Key Certificates, protocol 2	Type flaw	348.49
	Replay	TO
Davis Swick Private Key Certificates, protocol 3	Replay	2.68
Davis Swick Private Key Certificates, protocol 4	Replay	35.97

Légende : TO : Time Out
Replay STS : Replay attack based on a Short-Term Secret

message suivant est extrait du protocole de Denning-Sacco :

$$S \rightarrow A : \{B, Kab, T, \{A, Kab, T\}_{Kbs}\}_{Kas}$$

Cependant, cette écriture masque deux actions très différentes : l'agent S doit composer ce message, et l'agent A doit le recevoir.
Pour la composition, rien n'indique comment S va la réaliser : connaît-il toutes les informations atomiques du message ? un bloc chiffré n'est-il connu que dans sa globalité ? des informations doivent-elles être engendrées (un nonce par exemple) ?
Pour la réception, rien n'indique ce qui peut être vérifié : A peut-il déchiffrer l'ensemble du message ? si un morceau ne peut être déchiffré, peut-il le recomposer pour vérifier qu'il s'agit de la même chose ? quelles informations connaît-il déjà et peuvent donc être vérifiées ?

Gestion des connaissances des agents : En plus de la composition et de la réception des messages, une autre partie de la gestion des connaissances des agents est masquée par ce type de spécification : l'apprentissage d'une information (une clef par exemple) à la réception d'un message peut permettre de décomposer d'anciens messages et d'en apprendre le contenu.

Analyse des propriétés : Les propriétés à analyser sont en général énoncées à la fin de la spécification, liées à une information utilisée dans les messages. Cependant, leur déclaration ne permet pas de montrer à (partir de) quelles étapes du protocole elles doivent être vérifiées.

Implantation du protocole : D'une manière générale, les spécifications à la Alice-Bob n'apportent aucune information sur la manière d'implanter le protocole. Non seulement elles laissent libre cours à des erreurs lors de la spécification, mais elles ouvrent la porte à des erreurs d'interprétation pour réaliser l'implantation.

Simplicité des protocoles spécifiables : Le dernier défaut et non le moindre est la limite du langage : seuls des protocoles simples proposant des échanges ping-pong de messages entre agents sont spécifiables ; il est impossible de définir un protocole conditionnel ou non déterministe.

Ce type de langage de spécification n'est donc pas adapté pour considérer des protocoles complexes comme ceux développés par l'IETF comme nous allons le voir dans la section suivante.

4.3.2 La nouvelle génération de protocoles de sécurité

L'IETF (Internet Engineering Task Force)[12] fournit un forum à des groupes de travail pour coordonner le développement technique de nouveaux protocoles. Sa fonction la plus importante est le développement et la sélection de standards parmi les suites de protocoles de l'Internet.

L'histoire de la cryptographie et de la cryptanalyse a montré qu'une discussion ouverte et une analyse des algorithmes permet de mettre en évidence des faiblesses non identifiées par les auteurs originaux, et mène donc à de meilleurs algorithmes, plus sûrs. C'est donc l'objectif de l'IETF qui soutient et encourage la discussion ouverte, mais prudente, des vulnérabilités du matériel et du logiciel dans tous les domaines appropriés, fournissant aux constructeurs de produits vulnérables des informations importantes sur les failles découvertes, afin qu'ils aient l'opportunité de les corriger.

[12] `http://www.ietf.org/`

Les méthodes formelles commencent à faire leur chemin à l'IETF pour diverses raisons : le nombre de protocoles conçus chaque année est très élevé, rendant impossible une analyse purement « manuelle » de chacun ; le niveau d'exigence sur leur sécurité est croissant ; des preuves formelles des propriétés des protocoles sont un atout. En conséquence, cela impose de nouveaux challenges aux communautés de spécification formelle et de vérification. Écrire une spécification formelle requiert l'utilisation d'un langage formel suffisamment évolué pour considérer des protocoles à grande échelle, dont les messages sont souvent basés sur une structure riche. Et la vérification de la sécurité des protocoles demande d'abord de définir explicitement les propriétés requises (ce qui n'est pas toujours le cas dans les documents produits par l'IETF), puis d'appliquer des méthodes formelles d'analyse.

Pour montrer la diversité des protocoles et des propriétés considérés par l'IETF, je liste ci-dessous les catégories identifiées par Siemens, grâce à divers travaux de l'IETF, dans un livrable pour le projet AVISPA [88].

a - Exemples de protocoles

Voici une liste de protocoles classés par mécanismes d'authentification pour Internet :
- Systèmes de mot de passe à usage unique : S/Key, OTP, SecureID
- Systèmes de challenge-réponse : APOP, ACAP, HTTP Digest, AKA, CRAM-MD5, Kerberos, SIM
- Systèmes de preuve de mots de passe sans connaissance : EKE, A-EKE, SPEKE, SRP
- Systèmes serveurs de certificats : SSL/TLS, IPsec
- Systèmes à clef publique mutuelle : SSL/TLS, IPsec IKE, S/MIME
- Systèmes d'authentification générique : GSS-API, SASL, EAP
- Systèmes serveurs d'authentification : Kerberos, RADIUS, DIAMETER

b - Exemples de propriétés

Voici une liste de propriétés considérées par les protocoles cités précédemment :
- Authentification (unicaste) : vérifier une identité prétendue par une entité système ; l'authentification est en général unilatérale ; une authentification mutuelle est une authentification dans les deux sens ; des types d'authentification plus précis peuvent être étudiés : *authentification d'une entité*, *authentification d'un message*, *protection contre le re-jeu*.
- Authentification multicaste : authentification requise pour des groupes avec une unique source et un grand nombre de récepteurs potentiels ; des propriétés plus précises sont : *authentification de la destination implicite*, *authentification de la source*.
- Autorisation (par un tiers sûr) : un tiers sûr (TTP) présente un agent à un autre en se portant garant.
- Propriétés d'accord de clef :
  - *authentification de clef* : garantie que seuls les agents prévus ont accès à une clef confidentielle ;
  - *confirmation de clef* : preuve de possession d'une clef confidentielle ;
  - *secret futur parfait* : si une clef à long terme est compromise, cela ne compromet pas les clefs de sessions passées ;
  - *dérivation d'une clef fraîche* : gestion dynamique de clef pour engendrer une clef de session fraîche ;
  - *négociation de capacités sécurisées* : garantie de l'origine des capacités et paramètres négociés.

- Confidentialité (secret) : garantie qu'une information échangée n'est pas disponible à des individus non autorisés.
- Anonymat : propriété rarement assurée par des protocoles, mais certains masquent les identités : *protection de l'identité contre le détournement*, *protection de l'identité contre un pair*.
- Résistance (limitée) au déni de service (DoS) : propriété très difficile à vérifier, donc souvent définie pour des objectifs précis : *DoS sur l'allocation mémoire*, *DoS sur la puissance de calcul*, *attaque par bombardement sur des tiers*.
- Invariance de l'émetteur : pas de modification de la source de la communication tout au long des échanges.
- Non répudiation : prévention contre le déni d'une action (*preuve d'origine*, *preuve de réception*).
- Propriété de sécurité temporelle : utilisation d'opérateurs logiques temporels (*toujours* et *de par le passé*) pour décrire diverses propriétés.

D'autres propriétés sont parfois considérées comme relevant de la sécurité par l'IETF :

- Séparation cryptographique de clefs
- Négociation d'une suite de cipher
- Résistance à une attaque par dictionnaire
- Support pour reconnexion rapide
- Indépendance de sessions
- Résistance à une attaque par homme au milieu
- Vivacité d'un serveur

Les deux dernières propriétés sont liées à des propriétés d'authentification.

4.3.3 Langages de spécification haut/bas-niveau

Les protocoles cités dans la section précédente font parfois intervenir des notions complexes, impossibles à exprimer avec une simple notation à la Alice-Bob. Il est donc nécessaire d'utiliser un langage de spécification plus expressif, mais aussi plus proche des problématiques d'implantation. En effet, les spécifications réalisées ont plusieurs objectifs :

- une description d'un protocole rédigeable par des industriels ;
- un guide pour l'implantation ;
- une description formelle connectable à des outils de vérification.

Les bonnes propriétés des langages de spécification sont :

- simplicité, compréhension ;
- flexibilité ;
- non ambiguïté ;
- modularité ;
- expressivité : contrôle de flux, connaissances, primitives cryptographiques, types de messages, propriétés algébriques.

Au regard de ces critères, un langage proche d'un langage de programmation serait le plus adapté. Cependant, un tel langage, s'il est pratique pour les concepteurs, n'est pas adapté à l'analyse formelle. Pour combler ce grand écart rencontré lors du projet AVISPA, il a été décidé d'utiliser deux langages de spécifications : l'un haut-niveau, proche d'un langage de programmation ; l'autre, bas-niveau, accessible aux outils d'analyse formelle ; ces deux langages devant être liés par un traducteur automatique d'une spécification haut-niveau en une spécification bas-niveau.

a - Langage haut-niveau

Le langage haut-niveau de spécification doit donc offrir les services suivants pour pouvoir représenter des protocoles complexes.

- Il doit être modulaire. Cette modularité s'exprime par la représentation des échanges de messages par rôle, chaque rôle correspondant à un participant du protocole et exprimé sous forme d'une processus indépendant. Cela permet une gestion claire de l'évolution des connaissances de chaque participant.
  La modularité doit aussi intervenir au niveau de la description des scénarios à étudier, pour définir par exemple quels rôles composer pour représenter une session, quels rôles sont communs à toutes les sessions, et bien sûr combien de sessions doivent être considérées et comment les instancier.
- Il doit proposer des bases cryptographiques variées : clefs symétriques (non atomiques), clefs publiques/privées, fonctions de hachage, nonces.
- Il doit permettre un typage total des informations manipulées : types atomiques et types composés.
- Il doit permettre l'utilisation d'opérateurs algébriques comme la concaténation de messages, le ou-exclusif et l'exponentiation.
- Il doit permettre de décrire les niveaux de protection des messages envoyés et reçus ; pour cela, des canaux de communication doivent être utilisés, chacun ayant un niveau de protection déclaré.
- Il doit proposer des structures de contrôle, comme des conditions et des itérations.
- Il doit décrire explicitement l'évolution des connaissances des participants et leur utilisation pour composer les messages transmis, ainsi que le moment où créer une information fraîche (comme un nonce par exemple).
- Enfin, il doit proposer une grande variété de propriétés de sécurité analysables.

Ce type de langage doit aussi avoir une *sémantique explicite*. Par exemple, utiliser un fragment de la logique temporelle des actions (TLA) de Lamport [111] permet de faire la différence entre vérifier une information dans un message reçu et apprendre une information.

Exemple : le langage HLPSL du projet AVISPA. Le langage HLPSL (*High Level Protocol Specification Language*) [86, 97] défini lors du projet AVISPA respecte ces critères. Chaque rôle est décrit comme un sous-programme paramétré, possédant ses propres variables locales, les actions étant décrites sous forme de transitions non-déterministes gardées. Des rôles spécifiques décrivent la composition des rôles agents ainsi que leur instanciation. Les propriétés sont exprimées soit sous forme de macro pour les plus répandues (secret et authentification faible ou forte), soit sous forme de formules temporelles.
La sémantique de ce langage est décrite dans la thèse de Luca Compagna [100] et dans [97].

Exemple : le langage du projet PROUVE. Le langage défini pour le projet PROUVE [91] est basé sur le HLPSL, mais avec deux principales modifications : les actions dans les rôles sont décrites par une séquence d'instructions (le non déterminisme ne porte pas sur toutes les transitions, mais peut être représenté par une instruction spécifique) ; le scénario à analyser est également décrit par une séquence d'instructions, avec la possibilité d'exprimer l'exécution en parallèle d'instructions.

b - Langage bas-niveau

Le langage bas-niveau doit permettre de représenter le protocole sous une forme suffisamment formelle pour être exploitée sans difficulté par des outils d'analyse. Ce langage ne doit pas détruire les apports du langage haut-niveau, mais se focaliser sur les besoins des outils d'analyse, à savoir :

- l'état initial de chaque instance de rôle ;
- la liste des transitions, c'est-à-dire des changements d'états de chaque rôle ;
- les propriétés à vérifier, exprimées soit sous forme de formules logiques devant toujours être satisfaites, soit sous forme d'états à atteindre pour trouver une attaque.

Exemple : le langage IF du projet AVISPA. Le langage IF (*Intermediate Format)* [87] reprend exactement les points cités ci-dessus, chaque transition étant exprimée sous forme d'une clause. Les propriétés à vérifier sont exprimées à la fois sous forme de formules logiques en TLA, et sous forme d'états à atteindre pour obtenir une attaque, permettant à tout outil d'analyse d'y trouver son bonheur.

4.3.4 AVISPA, un outil pour les industriels

L'outil AVISPA est le logiciel issu du projet Européen AVISPA (*Automated Validation of Internet Security Protocols and Applications*), collaboration entre l'Université de Gênes, l'ETH Zurich, Siemens Munich et l'INRIA Lorraine. Faisant suite au projet AVISS rassemblant uniquement des centres de recherche, il a été motivé principalement grâce au nouveau partenaire industriel, Siemens, qui nous a servi d'intermédiaire avec l'IETF pour établir les besoins pour spécifier et analyser de nombreux protocoles d'Internet en cours d'utilisation ou de développement.

L'architecture de l'outil, schématisée dans la Figure 4.2, met à disposition des concepteurs de protocoles un langage de haut niveau, appelé HLPSL, brièvement décrit dans la section précédente. Les spécifications ressemblent à de vrais programmes, chaque rôle du protocole étant représenté par un sous-programme indépendant.

Un traducteur entièrement automatique (que j'ai développé), appelé HLPSL2IF, permet de transformer toute spécification HLPSL en une spécification bas-niveau dans le langage IF (également décrit brièvement dans la section précédente).

Ensuite, quatre outils d'analyse sont proposés :

- OFMC [134] : outil utilisant des techniques symboliques pour explorer l'espace de recherche, développé par à l'ETH Zurich ;
- AtSe [173] : outil appliquant de la résolution de contraintes combinée avec des heuristiques de simplification et des techniques d'élimination de redondance, développé dans l'équipe CASSIS du LORIA, implantant les méthodes d'analyses que nous avons définies ;
- SATMC [130] : outil développé à l'Université de Gênes qui construit une formule propositionnelle codant toutes les attaques possibles (de taille bornée) et transmet cela à un solveur SAT ;
- TA4SP [135] : outil qui approxime les connaissances de l'intrus à l'aide de langages d'arbres réguliers et de réécriture, pour produire des sous- et sur-approximations, développé dans l'équipe CASSIS à Besançon.

Noter que les trois premiers outils cherchent des attaques, alors que le dernier essaie de démontrer la validation des propriétés, tâche beaucoup plus difficile et ne pouvant être appliquée que pour des propriétés relativement restreintes.

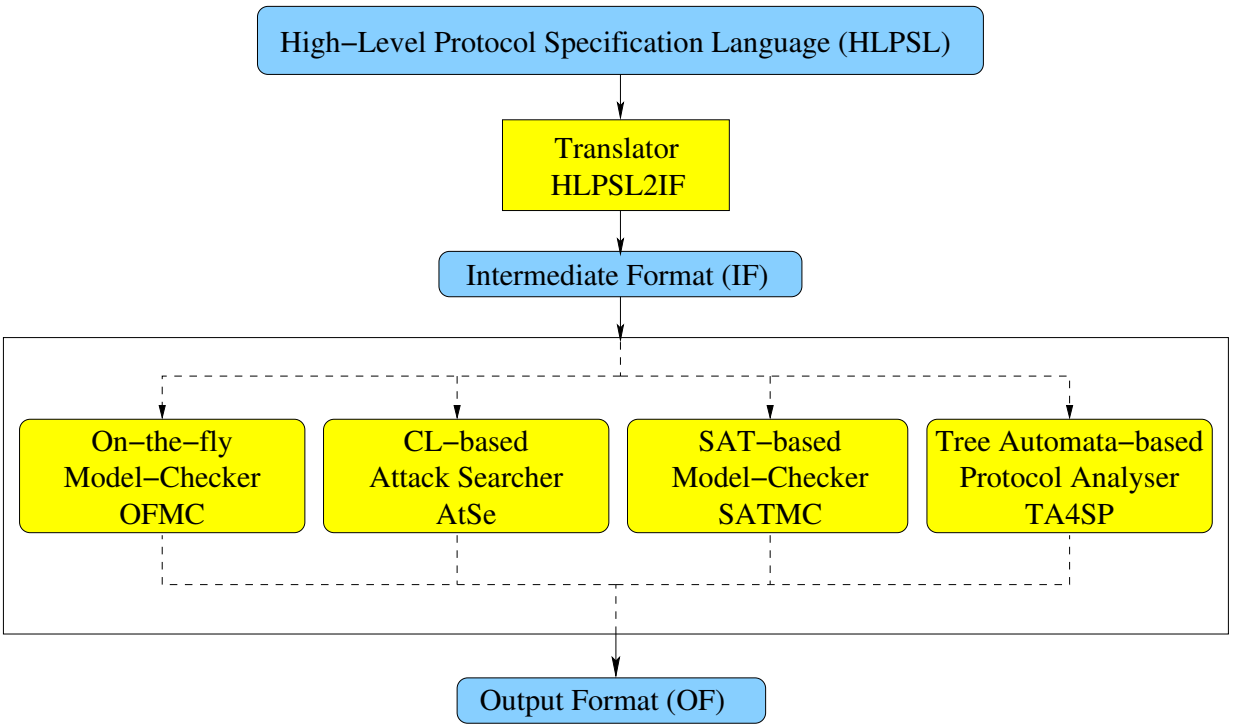


FIG. 4.2 – Architecture du logiciel AVISPA

Ces quatre outils produisent un résultat dans un format de sortie commun, pour le confort de l'utilisateur.

Une interface graphique (cf. Figure 4.3) est également disponible selon deux modes : un mode basique simplifiant les actions de l'utilisateur, et lançant les quatre outils en parallèle, le résultat de chacun étant ensuite visible ; et un mode expert permettant de choisir l'outil d'analyse et de régler quelques paramètres d'analyse.
Lorsqu'une attaque est trouvée, un diagramme de séquence permet de visualiser les messages échangés qui ont mené à cette attaque.

L'outil AVISPA est librement disponible à l'adresse `http://www.avispa-project.org/`. Et même si pour l'instant il n'évolue plus sous forme de nouvelles distributions officielles, il reste très téléchargé et la mailing-list des utilisateurs est active. Ses utilisateurs sont des industriels et des chercheurs développant de nouveaux protocoles, mais aussi de nombreuses universités dans le cadre de cours sur la sécurité des communications.

Dans le cadre du projet AVISPA, un grand nombre de protocoles a été spécifié (une cinquantaine, pour plus de 230 problèmes) et analysé [84, 126], permettant de trouver de nombreuses attaques, dont plusieurs inédites.

4.4 Autres outils d'analyse de protocoles

Le but de cette section est de présenter très brièvement quelques outils (ou techniques) d'analyse de protocoles, en commençant par ceux existant lorsque j'ai commencé à travailler sur ce domaine, puis en terminant par les quelques outils actuels, librement diffusés.

Les logiciels, au développement desquels j'ai participé (Casrul, AVISS, AVISPA), ne sont pas rappelés dans cette section, ayant été décrits plus tôt dans ce chapitre.

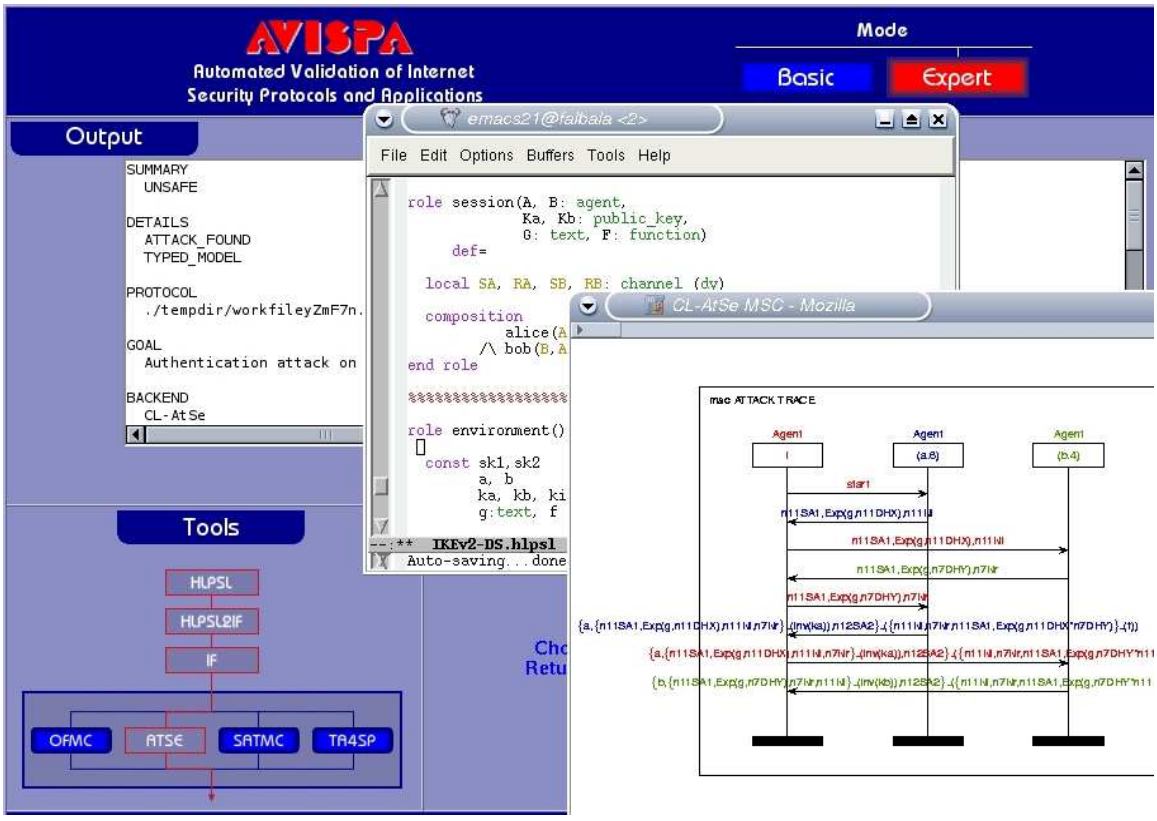


FIG. 4.3 – Interface graphique du logiciel AVISPA

4.4.1 Premiers outils

Le but de cette première section est de présenter brièvement l'état de l'art logiciel au moment où nous nous sommes intéressés à ce domaine, et d'y situer les apports des logiciels que nous avons développés.

Jusqu'à la fin du XXe siècle, les outils basés sur des méthodes formelles pour l'analyse de protocoles de sécurité étaient principalement basés sur soit de la vérification interactive, soit du model-checking automatique mais à nombre fini d'états : Coq [92], STA [93], AAPA [94], BAN-like logics [96, 125], TAPS [99], CAPSL [103], LOTOS [112], Casper et FDR [113, 106, 122], CVS [107], NRL [116, 117], Murphi [119], Isabelle/HOL [120], FDR2 [123], Spass [127].
Les outils interactifs demandent un investissement considérable en temps par des utilisateurs experts et ne fournissent aucun support pour la détection d'erreur lorsque le protocole est faillible. Les outils automatiques en général requièrent de fortes hypothèses qui bornent l'information analysée, pour qu'un système à nombre infini d'états soit approximé par un système à nombre fini d'états. De plus, le niveau de support d'automatisation d'alors passait difficilement à l'échelle et était insuffisant pour valider des protocoles réalistes. D'autre part, le paramétrage de la spécification (comme construire les approximations appropriées) et des outils demandait souvent une expertise importante.

Par exemple, les deux premiers outils entièrement automatiques ont été Casper et CAPSL. Casper [113] est un compilateur transformant une spécification de protocole en algèbre de processus (CSP). L'approche est orientée pour la vérification à nombre fini d'états par model-checking

avec FDR [122].
CAPSL [118] est un langage de spécification pour des protocoles d'authentification semblable à l'entrée de Casper. Un compilateur [102] traduit la spécification en un formalisme intermédiaire CIL qui peut ensuite être converti en l'entrée d'outils de vérification automatique comme Maude [114], PVS [121] et NRL [95].

Le logiciel Casrul [109] que nous avons développé a marqué le début de la transition entre tous ces outils semi-automatiques ou à analyse bornée et des outils d'analyse formelle de protocoles réels. Entièrement automatique, basé sur une sémantique rigoureuse, les analyses se font en toute généralité, découvrant une attaque s'il en existe une. Et comme pour CAPSL, l'extension de Casrul en AVISS [85] a permis de connecter plusieurs outils d'analyse (SATMC et OFMC, versions préliminaires de celles développées par la suite pour le logiciel AVISPA) et d'étudier un grand nombre de protocoles décrits dans la bibliothèque de Clark et Jacob [98].

Comparé à Casper, le logiciel AVISPA a permis de considérer des modèles à nombre d'états infinis moins restrictifs, utilisant des techniques d'abstractions et de model-checking symbolique pour considérer cette notion d'infini.
D'autres outils plus récents existent, comme Athena [124] qui propose de combiner des techniques de model-checking et de démonstration avec le modèle d'espace de Strand. L'objectif est alors de réduire l'espace de recherche et de démontrer automatiquement la correction de protocoles pour un nombre arbitraire de sessions concurrentes. Athena a ainsi été utilisé pour analyser avec succès des protocoles de taille réduite, mais son passage à l'échelle reste discutable. Par exemple, comme pour Casper/FDR2, cette approche n'est pas capable de considérer des attaque par typage. De même, il ne permet pas de considérer des protocoles à clefs non atomiques, ce qui est indispensable dans certains protocoles de e-commerce par exemple.

4.4.2 La génération actuelle d'outils

En plus de l'outil AVISPA décrit dans une section précédente, il n'existe principalement que trois outils librement diffusés et dont l'utilisation n'est pas totalement confidentielle : Maude-NPA, ProVerif et Scyther. Ces outils sont brièvement décrits ci-dessous.

a - Maude-NPA (`http://maude.cs.uiuc.edu/tools/Maude-NPA/`)

Ce logiciel correspond à l'incorporation de l'outil NRL Protocol Analyzer (développé par C. Meadows) dans Maude, un environnement de programmation en logique de réécriture (développé au SRI). Ce mariage a permis d'intégrer des fonctionnalités essentielles dans l'outil d'analyse de protocoles : l'intégration de la gestion de théories équationnelles (unification, filtrage), de techniques efficaces de réécriture, etc.

Le logiciel obtenu [108] est un model-checker fonctionnant en recherche en arrière, appliquant des déductions modulo des théories équationnelles spécifiques (ou-exclusif, exponentiation de Diffie-Hellman, neutralisation des chiffrements/déchiffrements), et effectuant des analyses dans un modèle à sessions non bornées. En conséquence, les analyses sont effectuées sur un modèle exact, garantissant que si une attaque existe, elle sera trouvée. Cependant, la terminaison du processus n'est bien sûr pas garantie, malgré un très grand nombre de techniques limitant l'espace de recherche.

b - ProVerif (`http://www.proverif.ens.fr/`)

Ce logiciel, développé par B. Blanchet [90], effectue des analyses de protocoles pour un nombre non borné de sessions. Les spécifications initiales sont fournies dans un langage de

calcul de processus, extension du pi-calcul. Elles peuvent contenir des propriétés équationnelles représentant les propriétés d'opérateurs algébriques comme le ou-exclusif ou l'exponentiation de Diffie-Hellman. Le protocole est automatiquement traduit en clauses de Horn par abstractions, avec l'inconvénient de pouvoir faire perdre une partie (très limitée) de la sémantique initiale.

Les abstractions utilisées pour ne pas limiter le nombre de sessions ont l'inconvénient de pouvoir provoquer de fausses attaques, même pour un protocole correct. Et la terminaison de l'analyse n'est pas garantie.

c - Scyther (`http://people.inf.ethz.ch/cremersc/scyther/`)

Ce logiciel [101] permet de spécifier les protocoles dans une syntaxe basée sur les rôles, mais sans expliciter la sémantique de composition ou de réception des messages : les messages ont une syntaxe à la Alice-Bob.

L'analyse est effectuée sur des schémas de traces, dans un environnement borné de sessions. Mais si aucune attaque n'est trouvée, et qu'aucun schéma de trace persiste, alors le comportement est similaire à une analyse avec un nombre non borné de sessions, et donc la propriété étudiée est validée.

En conclusion, même si les outils cités précédemment s'avèrent efficaces, AVISPA reste l'un des outils les plus largement diffusés et utilisés, accessible à un grand nombre, industriels, chercheurs ou étudiants, grâce à un langage de spécification puissant mais très compréhensible et limitant les erreurs (grâce à sa sémantique explicite). Cet effort de diffusion que nous avons réalisé est indispensable pour montrer tout l'intérêt des techniques de vérification que nous avons développées (cf. chapitre suivant), mais aussi et surtout pour participer à la « banalisation » de l'utilisation d'outils formels pour vérifier les protocoles conçus.

Chapitre 5

Protocoles de sécurité : propriétés, méthodes d'analyse et études de cas

Les protocoles de sécurité sont conçus pour établir la confiance lors de transactions électroniques. Ils sont représentés par des échanges de messages entre les participants de la transaction, ces messages utilisant diverses primitives cryptographiques pour assurer l'intégrité des données échangées, l'anonymat des participants ou la confidentialité des informations échangées, par exemples.

Or, il s'est avéré que de nombreux protocoles se sont révélés faillibles, sans même s'intéresser à l'aspect cryptologie, c'est-à-dire sans essayer de « casser » une clef. Les problèmes viennent en général de la combinaison de plusieurs sessions par un agent malhonnête, qui peut également se faire passer pour quelqu'un d'autre lors de certains envois de messages. Ces attaques ne nécessitent donc pas de gros moyens de calcul, et peuvent avoir des conséquences économiques très graves.

L'analyse de protocoles de sécurité est une tâche très difficile, car il faut faire face à de nombreuses dimensions infinies : nombre de sessions pouvant être lancées en parallèle, taille des messages pouvant être construits, nombre de participants possibles. Et il faut combiner cela avec d'autres difficultés : gestion des connaissances des participants, gestion du comportement d'un intrus, gestion des propriétés algébriques des primitives cryptographiques et des structures de données utilisées pour composer les messages.

Nous nous sommes donc intéressés à ce problème car, au contraire des outils de model checking habituellement utilisés, les techniques de déduction que nous avons développées dans le passé permettent de considérer les différentes dimensions de ce problème sans le restreindre. En particulier, l'utilisation d'unification permet de facilement traiter la notion d'information fraîche, et disposer de techniques de déduction modulo des propriétés d'associativité-commutativité permet à la fois de considérer le non-déterminisme lors de l'application d'une transition, mais aussi de traiter des propriétés d'opérateurs comme le ou-exclusif.

Notre objectif a donc été, partant d'une spécification standard d'un protocole, de définir une méthode permettant une analyse entièrement automatique cherchant des failles ou bien démontrant des propriétés.

Le chapitre précédent ayant décrit les langages de spécification et les logiciels sur lesquels j'ai travaillé, dans ce chapitre nous allons nous intéresser aux principaux résultats théoriques obtenus concernant l'analyse de protocoles. Ces résultats portent sur la conception de stratégies permettant de restreindre l'espace de recherche d'une attaque, la conception de méthodes de recherche d'attaque modulo des propriétés algébriques, la conception de méthodes d'analyse pour

un nombre non borné de sessions d'un protocole, l'analyse de propriétés complexes (comme la non répudiation), l'analyse de protocoles de groupes et la définition formelle de leurs propriétés. Ce chapitre mentionne également quelques études de cas menées en partenariat avec divers industriels ou chercheurs.

L'article référence de ce chapitre est celui publié dans le journal international ASE (Automated Software Engineering) en 2004 [146]. Il représente une avancée importante dans la vérification de protocoles de sécurité en évitant de borner avec excès l'espace de recherche. De nombreux autres résultats importants sont décrits dans ce chapitre, mais c'est celui-ci qui pour moi fait preuve de la plus grande originalité.

5.1 Stratégie paresseuse de l'intrus

Les premiers résultats obtenus pour l'analyse de protocoles cryptographiques ont consisté à appliquer des techniques générales de déduction pour rechercher une attaque. Ces travaux ont été brièvement décrits dans le chapitre précédent et correspondent à la publication fondatrice [155] de cet axe de recherche dans l'équipe CASSIS. Les expérimentations effectuées avec le logiciel Casrul, utilisant le logiciel daTac, ont montré les limites de cette technique trop générale.

Les travaux avec Yannick Chevalier, pour sa thèse [141], ont essentiellement porté sur l'amélioration et l'extension des techniques définies dans Casrul, avec en parallèle la réalisation du logiciel AVISS dans le cadre du projet européen du même nom[13].

Le premier résultat important obtenu a été la définition d'une stratégie paresseuse pour l'intrus [144, 143]. Auparavant, des règles étaient engendrées pour décrire les différents messages à composer par l'intrus lorsqu'il veut se faire passer pour l'un des participants officiels. Ces règles limitaient à un nombre fini les possibilités, mais cependant offraient une combinatoire encore très importante et empêchaient l'intrus de faire ce qu'il voulait (ce qui est le principe d'un intrus standard, comme décrit par Dolev et Yao). Ainsi, le nouveau modèle consiste à représenter symboliquement les messages attendus par les participants, comme décrit dans la Section 4.2.1, les parties non vérifiables étant remplacées par des variables. La stratégie paresseuse consiste à tester si l'intrus peut composer un tel squelette de message. Si c'est le cas, des contraintes sont alors conservées pour mémoriser que l'intrus devra aussi instancier les variables (désignant les parties non vérifiables du message) à partir de ses connaissances actuelles. La propagation de ces contraintes est importante, car les étapes suivantes du protocole peuvent imposer le contenu de ces variables (ou une partie de celui-ci).

Cette stratégie dynamique est correcte et complète, et termine, pour un nombre borné de sessions du protocole. Les variables ne sont pas typées, ce qui permet de détecter des failles de typage et de considérer des messages de taille non bornée. Les expérimentations ont confirmé le gain considérable avec cette stratégie, car pour un message attendu, au lieu d'engendrer tous les messages constructibles par l'intrus, un seul message est engendré, contenant les contraintes de composition à respecter.

Une stratégie paresseuse avait déjà été proposée par Roscoe dans [164]. Cependant, elle consistait à chercher les messages pouvant être composés avant l'exécution du protocole, et à les préparer statiquement à l'avance, et pour borner leur nombre total à utiliser des informations de typage. Il s'agissait donc d'une restriction forte du modèle de Dolev-Yao.

Notre méthode est plus proche de celle décrite par Basin dans [133], qui se basait également sur un modèle paresseux dynamique, mais par contre uniquement concentré sur le typage paresseux

[13] AVISS : Automated Verification of Infinite State Systems (FET-Open Assessment Project IST-2000-26410)

des données, géré grâce au langage Haskell.

De petites améliorations ont accompagné ce principal résultat, comme la possibilité d'utiliser des fonctions de hachage, des clefs composées pour le chiffrement symétrique, l'ajout de l'opérateur ou-exclusif, et l'ajout de la propriété de secret à court terme, tout ceci dans l'objectif de pouvoir spécifier et analyser un plus grand nombre de protocoles.

Ces ajout ont demandé l'adaptation du comportement de l'intrus, et plus particulièrement pour le ou-exclusif qui nécessite de considérer ses propriétés algébriques. Mais le logiciel daTac, proposant d'utiliser des opérateurs associatifs-commutatifs ou simplement commutatif a grandement facilité la tâche.
En ce qui concerne le secret à court terme, cela consiste simplement à vérifier qu'une information confidentielle le reste bien durant l'exécution du protocole qui l'a engendrée. Dès la fin de celle-ci, cette information est rendue publique, sous la forme de sa transmission à l'intrus.

5.2 Nombre non borné de sessions

Les méthodes classiques pour analyser un protocole consistent à prédéfinir un scénario, correspondant à une composition de sessions (ou de rôles) du protocole à étudier, et à n'effectuer l'analyse que sur ce scénario. Cette limite est très contraignante, car en conséquence la non découverte d'une attaque n'est qu'un résultat de correction partielle. Avec Yannick Chevalier, nous avons défini un modèle permettant l'utilisation d'un nombre non borné d'exécutions d'un protocole [145, 146]. Ce modèle consiste à utiliser une session normale, et à permettre à l'intrus d'utiliser autant de sessions qu'il le souhaite en parallèle.

Plus précisément, nous distinguons deux types d'instances de rôles :
- les instances régulières, représentant les sessions officielles sur lesquelles les propriétés seront analysées ; il n'existe qu'un nombre borné de ces instances, mais les messages sont de taille non bornée ;
- les instances « marionnettes », représentant des sessions officieuses qui n'ont pour but que d'être exploitées un nombre non borné de fois par l'intrus pour perturber les sessions officielles ; la taille des message est ici bornée, en particulier parce que toute information fraîche reste identique pour toutes les instances d'un même rôle (elle n'est pas réengendrée).

Nous avons montré que cette méthode termine, est correcte et complète, avec un petit inconvénient : des fausses attaques peuvent être trouvées (en raison de l'abstraction des nonces).

Ce résultat a été étendu, grâce à un modèle alternatif, pour montrer qu'on peut combiner une analyse avec un nombre non borné de sessions et une taille bornée des messages, et une analyse avec un nombre borné de sessions et une taille non bornée des messages [142].

Notre méthode permet aussi de détecter des attaques par confusion de type, car les informations composant les messages peuvent ne pas être typées. Elle permet également de détecter plusieurs attaques s'il n'y en a pas qu'une seule.

Ces travaux ont été implantés dans l'extension du logiciel Casrul suite au projet AVISS et ont été appliqués sur de nombreux protocoles de la bibliothèque de Clark et Jacob, permettant de retrouver de très nombreuses attaques [129], mais aussi de nouvelles attaques, comme par exemple sur le protocole à clef symétrique de Denning et Sacco [104]. Les spécifications de protocoles permettent ainsi de travailler à un niveau plus fin : au lieu de simplement spécifier des instances de sessions du protocole, on peut spécifier des instances de rôles joués officiellement, mais aussi des instances de rôles pouvant être utilisées un nombre non borné de fois par l'intrus pour perturber les rôles officiels.

5.3 Protocoles de non-répudiation

Les méthodes d'analyse de protocoles définies jusqu'à présent permettent de considérer une large gamme de protocoles, pour des propriétés relativement classiques. Afin d'augmenter ce spectre, dans le cadre de la thèse de Judson Santos Santiago [168], nous nous sommes intéressés aux protocoles de non-répudiation, c'est-à-dire aux protocoles dont le but est de fournir à un agent des preuves de sa participation ou de la participation d'un partenaire à une communication. Le principal intérêt de ces protocoles est qu'ils considèrent des propriétés très originales, leur analyse semblant donc moins standard.

5.3.1 Propriétés de non-répudiation

Les propriétés abordées par la non-répudiation sont en fait des services requis par un grand nombre d'applications de sécurité (pour le e-commerce par exemple). Tous ces services sont définis pour un message envoyé par un agent émetteur A à un agent récepteur B, éventuellement en présence d'un tiers de confiance TTP servant d'agent de livraison.

Non-répudiation d'origine : Le service de non-répudiation d'origine fournit à l'agent récepteur B un ensemble de preuves qui garantissent que l'agent émetteur A a bien envoyé le message. L'évidence d'origine est engendrée par A et détenue par B. Cette propriété protège le récepteur contre un émetteur malhonnête.

Non-répudiation de réception : Le service de non-répudiation de réception fournit à l'agent émetteur A un ensemble de preuves qui garantissent que l'agent récepteur B a bien reçu le message. L'évidence de réception est engendrée par B et détenue par A. Cette propriété protège l'émetteur contre un récepteur malhonnête.

Non-répudiation de soumission : Le service de non-répudiation de soumission fournit à l'agent émetteur A un ensemble de preuves qui garantissent que le message a été soumis à l'agent récepteur B. Ce service ne s'applique qu'en présence d'un tiers de confiance. L'évidence de soumission est engendrée par TTP et détenue par A. Cette propriété protège l'émetteur contre un récepteur malhonnête.

Non-répudiation de livraison : Le service de non-répudiation de livraison fournit à l'agent émetteur A un ensemble de preuves qui garantissent que l'agent récepteur B a bien reçu le message. Ce service ne s'applique qu'en présence d'un tiers de confiance. L'évidence de livraison est engendrée par TTP et détenue par A. Cette propriété protège l'émetteur contre un récepteur malhonnête.

Équité : Un service d'équité (aussi appelé *équité forte*) pour un protocole de non-répudiation fournit des preuves que, à la fin de l'exécution du protocole, soit l'émetteur a l'évidence de réception et le récepteur a l'évidence d'émission, soit aucun des deux n'a d'information exploitable. Cette propriété protège les deux agents A et B.

Timeliness : Un service de timeliness pour un protocole de non-répudiation garantit que, quoiqu'il se passe durant l'exécution du protocole, tous les participants peuvent atteindre, en un temps fini, un état qui préserve l'équité.

Cette liste de propriétés n'est pas exhaustive, car il existe par exemple différentes formes d'équité, mais elle permet de couvrir les besoins de la plupart des protocoles de non-répudiation.

5.3.2 La non-répudiation comme authentification

La première difficulté consiste à formaliser ces propriétés. La non-répudiation étant souvent considérée comme une forme d'authentification [165], nous avons décidé de montrer les liens effectifs, mais aussi les différences, entre ces deux propriétés.

Dans [166], nous avons montré comment exprimer sous forme d'une combinaison de propriétés d'authentifications chacune des propriétés de non-répudiation, et l'avons illustré sur le protocole équitable de Zhou et Gollmann [174].

La première observation a été de constater qu'il fallait utiliser des authentifications faibles, c'est-à-dire permettant d'utiliser un même témoin pour plusieurs demandes d'authentifications. Sans cela, une attaque par rejeu est immédiatement réalisable, en particulier pour le protocole considéré.

Cependant, les limites de cette représentation des propriétés de non-répudiation ont vite pris les devants. La première est la difficulté de considérer un agent malhonnête tout en préservant l'analyse des propriétés. En effet, si un agent malhonnête participe à une authentification, il peut facilement la faire échouer soit en engendrant de fausses requêtes, soit en ne fournissant pas les bons témoins aux requêtes reçues. Dans chacun des cas, il se pénalise et il ne serait pas anormal de considérer qu'il s'agit de fausses attaques, mais si ce genre d'attaque est possible, cela perturbe les outils d'analyse qui risquent de s'interrompre à leur découverte. Aussi, des outils comme AVISPA vont volontairement omettre d'analyser des propriétés impliquant l'intrus. La différence entre un intrus et un agent malhonnête n'est donc pas faite par les outils courants, ce qui empêche l'analyse complète de la non-répudiation représentée par des authentifications. De « petits trucs » de spécification sont souvent possibles (comme donner à l'intrus la clef privée d'un agent devant être malhonnête), mais cela ne permet pas de s'assurer de la complétude de l'analyse.

Un second problème se pose avec l'utilisation d'authentifications : certains protocoles de non-répudiation sont non-déterministes, et la définition des authentifications est donc difficile, voire impossible, à placer correctement. C'est le cas par exemple des protocoles optimistes, qui sont en fait la combinaison de plusieurs sous-protocoles.

5.3.3 La non-répudiation basée sur les connaissances

Pour permettre l'analyse de propriétés de non-répudiation dans leur intégralité, nous avons défini une seconde méthode basée sur la gestion des connaissances des agents participant au protocole [167]. Il s'agit en fait d'un système d'annotation des spécifications, combiné à une logique Booléenne classique décrivant des propriétés sur ces annotations. Les annotations désignent simplement qu'en un certain point du protocole, un agent connaît (ou peut déduire) une information.

Nous avons montré sur le protocole optimiste de Cederquist, Corin et Dashti [140] comment placer ces annotations dans le protocole, et comment s'expriment les diverses propriétés de non-répudiation à étudier. L'analyse a d'ailleurs permis de trouver deux attaques sur ce protocole, l'une entre agents honnêtes lorsqu'un message d'abandon met trop de temps à atteindre son destinataire, l'autre en présence d'un agent malhonnête.

Avec Francis Klay (France Telecom R&D), nous avons repris cette méthode et l'avons formellement décrite [157], montrant qu'elle pouvait être appliquée dans un environnement très

standard, ne nécessitant pas de développements complexes, mais juste un système de déduction de connaissances, présent dans la plupart des outils d'analyse de protocoles. Cette méthode s'applique très bien aux propriétés des protocoles de non-répudiation, mais peut aussi être appliquée aux protocoles de signature de contrat [169]. L'implantation dans AVISPA a été très rapide et s'est montrée immédiatement efficace avec les exemples traités.

J'ai appliqué cette méthode pour analyser un protocole conçu avec Jing Liu (Université de Sun Yat-Sen). Ce protocole de non-répudiation peut être utilisé sur des canaux de communication non résilients et avec une connexion limitée, grâce à une carte à puce du côté du récepteur [158].

5.4 Protocoles de groupes

Avec le succès obtenu dans la vérification de protocoles relativement classiques [129, 128], de nombreux travaux se sont orientés vers de nouveaux types de protocoles, beaucoup plus complexes. L'une de ces classes de protocoles concerne les protocoles de groupe [171, 159, 172, 170]. Un protocole de groupe est un protocole ou une suite de sous-protocoles pour l'établissement d'une clef secrète entre un nombre non borné d'agents. Ceci peut prendre la forme d'un établissement simple de clefs, exécuté entre les agents, ou d'une série de requêtes pour joindre ou quitter le groupe avec les mises-à-jour associées de la clef. Pour assurer la communication au sein d'un groupe, les membres ont généralement recours à cette clef secrète pour sécuriser leurs communications. L'opération la plus cruciale et la plus délicate dans ce type de protocoles est alors la gestion des clefs. Des protocoles dédiés à cette opération ont été et continuent d'être le sujet de nombreuses études de recherche [132, 131, 156, 139].

La vérification de protocoles de groupe se confronte à plusieurs problèmes. En effet, la sécurité des communications au sein de groupes n'est pas nécessairement une extension d'une communication sécurisée entre deux parties [171]. Elle est beaucoup plus compliquée. Une fois la communication commencée, le groupe peut changer de structure en ajoutant ou en supprimant un ou plusieurs membres. Les services de sécurité sont alors plus élaborés. En outre, les protocoles de groupe mettent en cause un nombre non borné de participants. Cependant, la plupart des approches automatisées de vérification de protocoles nécessitent un modèle concret. La taille du groupe doit alors être fixée à l'avance. Or, cette contrainte restreint considérablement le nombre d'attaques détectables. Ensuite, vu que les besoins en sécurité sont généralement liés aux protocoles, il est très difficile de décrire les propriétés que le protocole doit satisfaire.

Ainsi, toujours dans le but d'élargir le spectre de protocoles pour lesquels nous proposons des techniques de vérification, pour la thèse de Najah Chridi [148], nous nous sommes donc intéressés aux protocoles paramétrés, et plus particulièrement aux protocoles de groupes. Suite à une première étude réalisée lors de son stage d'ingénieur de l'ENSI (Tunisie), de nombreuses difficultés se sont révélées : formaliser les propriétés étudiées, spécifier les protocoles, faire intervenir la notion de hiérarchie de groupe, et bien sûr définir des méthodes d'analyse.

5.4.1 Formalisation des propriétés

La spécification des propriétés des protocoles de groupe est critique : toute erreur (de compréhension ou de formalisation de ces propriétés) peut engendrer de fausses failles de sécurité, ou provoquer la non détection d'une vraie faille. Des efforts ont été effectués pour une spécification des propriétés indépendamment des applications visées, à l'aide d'abstractions par exemple [153]. Des langages formels ont également servi à définir des propriétés comme l'authentification [160] :

un message est accepté par un agent si et seulement si, il l'a effectivement demandé à un second agent, et ce dernier le lui a bien envoyé.

La définition et la spécification des propriétés de sécurité est donc un passage obligé très délicat, avant même d'entamer la vérification. La difficulté est d'autant plus grande lorsqu'il faut considérer des propriétés non standards, comme celles liées à l'aspect dynamique des protocoles de groupe.

Nous avons donc proposé un modèle [147, 149, 150] pour les protocoles de groupe et plus généralement pour les protocoles contribuants (protocoles dans lesquels chaque participant contribue à la construction d'une clef, par exemple). Ce modèle permet de décrire un protocole contribuant, d'étudier ses caractéristiques et ses propriétés de sécurité, et donc d'identifier les différents types d'attaques possibles.

Nous avons ainsi défini un ensemble de caractéristiques, représentant des éléments de bonne construction du protocole, formellement décrites en fonctions des participants, de leurs connaissances et de leurs contributions (appelées services) :

- Unicité des identificateurs des agents : les identités des agents doivent être différentes.
- Visibilité des connaissances privées : les connaissances dites « privées » des agents doivent être réellement confidentielles, elles ne peuvent pas être partagées ; de plus, elles ne peuvent pas être diffusées en clair.
- Visibilité des connaissances partagées : les connaissances partagées entre plusieurs agents sont en fait des connaissances confidentielles vis-à-vis des autres agents ; chaque agent sait avec qui il partage des connaissances ; et comme pour les connaissances privées, les connaissances partagées ne doivent pas être transmises en clair.
- Distinction des services utiles : les ensembles des services nécessaires à la construction de la clef de groupe pour deux agents doivent être différenciés par au moins un élément.
- Indépendance des services utiles : les ensembles de services nécessaires à la construction de la clef de groupe pour deux agents ne doivent pas être liés par une relation d'inclusion.
- Correspondance des services : les services nécessaires à un agent pour construire la clef doivent provenir de contributions d'autres agents.
- Déduction de la même clef de groupe : pour un agent, la clef de groupe est engendrée en appliquant un algorithme prédéfini sur ses connaissances privées et partagées, ainsi que sur les services nécessaires de cet agent ; tous les membres du groupe doivent déduire la même clef, l'application de l'algorithme doit donc donner le même résultat pour tous les membres.
- Services minimaux : toutes les contributions doivent être utilisées pour la génération de la clef de groupe ; un agent ne peut pas se limiter à un sous-ensemble de ses services utiles pour déduire la clef.

Nous avons également identifié et formalisé un ensemble de propriétés de sécurité. Certaines sont indépendantes du temps :

- Authentification implicite de la clef : à la fin de la session du protocole, chaque participant est assuré qu'aucun élément externe ne peut acquérir sa vue de la clef de groupe.
- Secret de la clef : seuls les membres du groupe peuvent engendrer la clef partagée.
- Confirmation de la clef : pour qu'un agent confirme avoir engendré la clef, il doit confirmer à chaque émetteur d'un service utile à la génération de la clef qu'il a reçu sa contribution.
- Intégrité de la clef : permet de vérifier que tous les agents contribuent à la clef de groupe et que toute personne extérieure au groupe ne doit pas participer à la génération de la clef partagée.

D'autres propriétés de sécurité ont un lien fort avec la notion de temps :

- Secret futur : cette propriété garantit qu'un adversaire passif qui connaît un certain nombre

d'anciennes clefs de groupe ne peut pas découvrir une clef de groupe plus récente.
- Protection contre une attaque par service connu : un protocole est vulnérable à ce type d'attaque si la connaissance d'un service d'une session passée permet d'avoir la vue d'un agent pour la clef d'une session future.
- Secret passé : cette dernière propriété garantit qu'un intrus passif connaissant un sous-ensemble de clefs de groupe ordonnées ne peut pas découvrir une clef de groupe antérieure aux clefs connues.

Nous avons appliqué ces définitions sur divers protocoles de groupe et avons pu ainsi identifier les différents types d'attaques détectées sur chacun (cf. Figure 5.1).

Protocole	**Référence protocole**	**Référence attaque**	**Caractéristique(s) non vérifiée(s)**	**Propriété(s) non vérifiée(s)**
A-GDH.2	[162]	[163]	– déduction de la même clef de groupe.	– authentification implicite, – secret de la clef, – confirmation de la clef, – intégrité.
SA-GDH.2	[132]	[163]	– déduction de la même clef de groupe.	– authentification implicite, – secret de la clef, – secret futur (attaque par service connu).
Asokan-Ginzboorg	[131]	[170]	– déduction de la même clef de groupe.	– authentification implicite.
Bresson-Chevassut-Essiari-Pointcheval	[139]	[161]	– correspondance de services.	– secret futur (attaque par service connu et attaque par clef connue).

FIG. 5.1 – Synthèse des protocoles étudiés

5.4.2 Protocoles de groupes hiérarchiques

En collaboration avec l'équipe MADYNES du LORIA, nous nous sommes intéressés aux protocoles de groupes hiérarchiques [136, 137], qui permettent de représenter au sein d'un groupe la notion de classe ou de niveau, liée au rôle des membres. Ainsi, des protocoles de gestion de clef de groupe ont été orientés vers ces architectures hiérarchiques, comme [154]. Mais de nouvelles difficultés sont soulevées par ces protocoles : les propriétés de sécurité à vérifier sont en général liées à la notion de classe. Par exemple, une information ne doit être connue que par un sous-groupe, et rester secrète pour tout le reste du groupe. Or, ce type de protocole n'avait jamais été spécifié et vérifié.

C'est pour cette raison que nous avons étudié un protocole conçu dans le contexte du projet RNRT SAFECAST. Ce protocole est une instance du protocole Hi-KD [154], prévu pour fonctionner sur des réseaux PMR (*Professional Mobile Radio*), permettant aux groupes d'utilisateurs équipés de matériel mobile de sécuriser leur voix, leurs données ou leurs communications multimédia.

Un groupe est subdivisé en plusieurs classes hiérarchiques, avec différents niveaux. Les membres d'une classe utilisent une clef partagée pour assurer la sécurité des communications entre eux. Dans chaque classe, un agent est identifié comme le chef. Les membres d'une classe ont accès aux communications sécurisées des classes inférieures, mais pas à celles des classes supérieures. Dans ce protocole, la confidentialité des clefs est assurée par huit sous-protocoles :

- Génération et distribution de la clef principale et des clefs des classes inférieures.
- Renouvellement périodique de la clef principale.
- Ajout d'un membre au groupe.
- Réintégration d'un ancien membre du groupe.
- Exclusion d'un membre du groupe.
- Promotion d'un membre du groupe.
- Dégradation d'un membre du groupe.
- Rassemblement de groupes.

Pour analyser ce protocole, il a d'abord fallu définir précisément le contenu des messages de chaque sous-protocole, ainsi que les connaissances initiales des participants.

Nous avons ensuite spécifié ce protocole en HLPSL. La possibilité d'utiliser l'exponentiation a été un avantage, mais une limite importante a alors été trouvée : impossible de réaliser un broadcast, c'est-à-dire l'envoi d'un message à un grand nombre d'agents. Nous avons alors dû nous restreindre à des instances particulières du protocole, par exemple en limitant le nombre de classes à 3, et en limitant le nombre d'agents par classe (mais les agents d'une même classe ayant un comportement similaire, à l'exception du chef de la classe, cette borne n'est pas gênante). La notion de rôle du langage de spécification permet cependant de limiter fortement les modifications à effectuer pour faire varier ces bornes arbitraires.

L'analyse des divers sous-protocoles avec AVISPA, et plus particulièrement l'outil AtSe [173], a permis de trouver une attaque sur le protocole de promotion d'un membre, ce qui a permis de le corriger.

Nous avons également analysé le protocole BALADE [138], conçu pour des réseaux ad-hoc et subdivisant un groupe en clusters non hiérarchisés, et n'avons pas trouvé d'attaque.

5.4.3 Analyse de protocoles de groupes

L'analyse de protocoles de groupes hiérarchiques décrite dans la section précédente a essentiellement consisté en un travail de spécification, les propriétés analysées étant classiques : confidentialité et authentification.

Cependant, en complément de la formalisation des propriétés, mentionnée dans la Section 5.4.1, nous avons défini une stratégie de détection d'attaques pour des propriétés de sécurité bien spécifiques aux protocoles de groupes. Notre approche [151, 152] est basée à la fois sur le modèle orienté services décrit dans [149], et sur la résolution de contraintes. Cette dernière technique a fait ses preuves dans le passé pour des propriétés d'atteignabilité, et s'est révélée une très bonne base pour l'implantation. Le modèle orienté services est un modèle pour les protocoles d'accord de clef de groupe qui requiert un service de la part de chaque participant, une contribution à la clef de groupe. Il permet de spécifier des propriétés de sécurité comme des ensembles de contraintes.

5.5 Conclusion

Ce chapitre, bien que le plus court en nombre de pages, résume brièvement le travail de nombreuses années, et en particulier relate les résultats des trois doctorants que j'ai encadrés : Yannick Chevalier, Judson Santos Santiago et Najah Chridi.

Ces activités de recherche ont été très variées. C'est avec Yannick que nous avons obtenus les avancées les plus importantes, ayant permis de définir des méthodes efficaces d'analyse, totalement indépendantes des protocoles considérés. Hélas, une partie de ces travaux, implantés dans AVISS, n'a pas été intégrée dans AVISPA, principalement parce que le logiciel daTac a été remplacé par le logiciel dédié AtSe, ce qui était bien compréhensible pour des problèmes de temps d'exécution.
Avec Judson, nous avons étudié les propriétés de non-répudiation et d'équité, et réussi à montrer qu'elles pouvaient être considérées par les techniques d'analyse implantées dans AVISPA.
Avec Najah, nous avons étudié une catégorie très différente de protocoles : les protocoles de groupe. Au-delà de l'analyse de quelques exemples sur des propriétés classiques, nous avons dû formaliser les nombreuses propriétés très originales de ces types de protocoles, et définir de nouvelles stratégies de détection d'attaques sur ces propriétés.

Il ne faut pas oublier cependant que l'ensemble de ces résultats repose sur les travaux initiaux réalisés avec Florent Jacquemard et Michaël Rusinowitch [155] sur l'utilisation de techniques de déduction automatique pour l'analyse de protocoles de sécurité, décrits dans le chapitre précédent.

Conclusion et perspectives

L'équipe CASSIS du LORIA a fondé ses travaux de recherche sur des méthodes formelles et des techniques de preuves automatiques de propriétés. Depuis sa création en 2002, à laquelle j'ai participé avec Michaël Rusinowitch, nous nous sommes toujours attachés à utiliser ce socle fondamental pour réaliser des applications effectives. J'ai toujours eu cet objectif, comme le montre le nombre de logiciels réalisés (parfois en partenariat avec d'autres centres de recherche et industriels, comme pour AVISS et AVISPA). La recherche fondamentale est indispensable, et sa vraie valeur n'est dévoilée que lorsque les résultats sont mis à la disposition d'autres, au travers de logiciels accessibles à un grand nombre.

Le succès le plus visible de cette valorisation est le logiciel AVISPA, utilisé par plusieurs centaines d'entreprises et universités, ce qui est énorme pour un logiciel assez « pointu ». Il m'a permis de travailler avec des sociétés comme Siemens, SAP, IBM, France Télécom pour ne citer que des entreprises avec lesquelles je suis toujours en contact. Il me permet également, encore à ce jour, de travailler à distance avec de nombreux utilisateurs, pour concevoir de nouveaux protocoles, étendre le spectre des protocoles spécifiables, définir de nouvelles propriétés de sécurité, et développer de nouvelles techniques de vérification.

Mes perspectives d'activités de recherche sont donc toujours dans cette « philosophie » : concevoir des méthodes de déduction et d'analyse automatiques et développer des logiciels les mettant en œuvre et utilisables par un grand nombre. Je décris ci-dessous quelques sujets de recherche que je souhaiterais aborder à court ou moyen terme, certains, plus détaillés, pourraient représenter de réels sujets de thèse.

1 Protocoles de sécurité

Plus de dix années d'activité de recherche sur l'analyse de protocoles de sécurité sont loin d'avoir épuisé le domaine. Nous avons déjà réalisé un très grand pas : passer de l'analyse de protocoles « jouets » à celle de protocoles industriels. Mais je décris ci-après quelques pistes sur lesquelles il reste beaucoup à faire.

1.1 Combinaison de protocoles

L'analyse de protocoles donne d'excellents résultats pour des propriétés classiques comme la confidentialité et l'authentification. Cependant, la mise en place d'une communication nécessite en général de faire appel à plusieurs protocoles. Les problèmes de sécurité posés lors de la combinaison de ces protocoles n'ont jusqu'à présent jamais été étudiés. Pourtant, même s'ils interviennent parfois à des niveaux différents de l'infrastructure, il est indispensable de vérifier que cette combinaison n'introduit pas des failles, même si chacun des protocoles est sécurisé. Il s'agit là d'une problématique classique en déduction automatique, par exemple pour la combinaison de procédures de décisions, et un parallèle pourrait très certainement être fait.

1.2 Nombre de participants non connu

De nombreux protocoles de sécurité sont définis pour des applications très particulières, qui s'exécutent dans un environnement assez différent d'une communication classique sur un réseau avec ou sans fil, parfois même hétérogène. Par exemple, un protocole de vote électronique demande la participation d'agents particuliers : autorité électorale, scrutateur, électeur. Chacun de ces agents a un fonctionnement qui correspond à un sous-protocole, sans pour autant être totalement indépendant des autres. Ainsi, le nombre de participants effectifs n'est pas connu a priori, car on ne connaît que le nombre maximum d'électeurs possibles. Le scrutateur doit pouvoir vérifier le bon déroulement des votes, sans pour autant savoir qui a voté et comment.

La première difficulté pour ces protocoles particuliers est donc de réussir à les spécifier correctement, comme c'est également le cas pour des protocoles de sécurité de groupe, dont la taille est variable. Il faut donc un langage de spécification suffisamment expressif pour cela, mais avec une sémantique non ambiguë.
Il faut ensuite vérifier l'exécutabilité du protocole et gérer l'évolution des connaissances des participants, mais aussi de tout agent externe (malveillant en général).

Les propriétés à vérifier correspondent donc à des constatations de comportements, et non à des actions précises. Les méthodes d'analyse de ces propriétés doivent donc se focaliser sur les similarités de comportements des participants au protocole, d'une part pour permettre d'« oublier » leur nombre (non connu), mais aussi pour étudier si de légers changements de comportement peuvent ne pas être détectés.

Nous avons cité les protocoles de vote et les protocoles de groupe comme exemples, mais il en existe de nombreux autres, comme les protocoles d'enchères ou les protocoles faisant intervenir un broadcast.

1.3 Canaux de communication variés

Les principales méthodes d'analyse de protocoles reposent sur des canaux de communication ouverts. Or, il s'avère que pour de nombreux protocoles évolués, des canaux variés sont utilisés, offrant différents niveaux de protection. Par exemple, un canal peut être totalement protégé, ou bien peut être écouté mais sans détournement de messages, ou bien un canal peut être résilient, c'est-à-dire garantir l'ordre d'arrivée des messages. Or, chaque type de canal correspond à un modèle d'agent malveillant différent. L'analyse d'un protocole doit bien sûr s'effectuer en respectant ces modèles, leur remplacement par des canaux ouverts rendant souvent l'analyse inutile en engendrant de fausses attaques.

Il n'est pas forcément nécessaire de définir une méthode d'analyse pour chaque type de canal, car l'analyse peut « simplement » se traduire par une gestion adaptée des actions de l'intrus et de l'évolution de ses connaissances. Cependant, l'intrus ne peut plus disposer de n'importe quel agent pour initier des discussions afin de récupérer des informations exploitables dans d'autres sessions. Par exemple, un tiers de confiance avec lequel on ne peut communiquer que via un canal sécurisé, ne pourra peut-être pas être contacté par l'intrus sous une fausse identité.

Ainsi, le nombre d'actions de l'intrus peut croître de par la diversité des canaux utilisables, mais il peut aussi diminuer parce qu'une partie des échanges de messages ne lui est pas accessible car via des canaux protégés. Une dimension complexe peut également s'ajouter, le temps. Pour un canal résilient, l'intrus doit envoyer son message au bon moment, pour qu'il s'intercale au bon endroit entre les messages émis par les autres agents.

1.4 Vérification de l'implantation des protocoles

Les algorithmes cryptographiques utilisés dans l'implantation de protocoles influent sur la sécurité du résultat, car il en existe un grand nombre, offrant des niveaux de sécurité très variés. Une partie de leurs mécanismes sont parfois décrits dans les protocoles, sous forme d'opérateurs arithmétiques (ou exclusif, exponentielle,...), et des travaux théoriques ont été effectués pour en tenir compte dans les méthodes d'analyse. Cependant, l'implantation des protocoles demande de descendre à un niveau encore plus technique, pour tenir compte réellement des algorithmes mis en œuvre, mais aussi des types (au sens programmatoire du terme) des informations manipulées.

En effet, certains outils d'analyse détectent des attaques de types, lorsqu'une information (ou un ensemble d'informations) est confondue avec une autre information. En général, ces attaques ne sont pas réalistes, car demandent une confusion entre deux informations dont les types informatiques sont de tailles totalement différentes. L'implantation réelle d'un protocole demande de fixer la représentation de chaque information, et donc de fixer la taille nécessaire pour la mémoriser. Il n'est donc pas possible de confondre par exemple une clef de 512 bits avec le nom d'un agent de 64 bits.

L'analyse de protocoles à un tel niveau de détail est donc plus délicate, mais serait très appropriée pour les industriels.

1.5 Propriétés de non-répudiation, encore et toujours...

Les travaux sur les protocoles de non-répudiation avec Francis Klay, ont soulevé des problèmes qui ouvrent de nombreuses voies à explorer.

a - Non-répudiation vs. authentification

Nous avons montré que les propriétés de non-répudiation ne pouvaient être représentées à l'aide de simples authentifications sur les évidences. Cependant, une réflexion plus poussée peut remettre en cause cette affirmation. Les évidences sont des termes qu'un agent doit apprendre pour sa propre protection. Les attaques sur les protocoles de non-répudiation proviennent souvent de la combinaison d'évidences de plusieurs sessions du protocole. Comment contrôler la provenance de ces évidences ?

Une solution pourrait consister à construire un chaînage d'authentifications au sein d'une session normale du protocole. Ces authentifications permettraient de remonter de l'évidence obtenue à chaque terme élémentaire la composant ou ayant permis de la composer. Cela revient donc à construire un arbre représentant toutes les authentifications à effectuer. Cependant, comme il faut étudier tous les moyens possibles pour un agent de recevoir une certaine information utile pour l'évidence, l'arbre n'est pas unique, et cela peut s'exprimer par des disjonctions d'authentifications.

Ainsi, chaque rôle du protocole aurait son arbre décrivant le cheminement des informations utiles pour les évidences. Les arbres de plusieurs rôles sont liés par les échanges de messages.

La mise en place de toutes ces authentifications intermédiaires peut donc se faire automatiquement, à partir des évidences et d'une session normale du protocole. L'analyse va consister à rechercher un flux complet d'authentifications valides.

Bien sûr, pour que cette méthode par authentifications soit utile, il faut arriver à démontrer qu'elle implique le calcul de non-répudiation, voire qu'il y a équivalence entre les deux.

b - Vérification et utilisation des évidences

Lorsqu'un protocole de non-répudiation est conçu, son auteur précise la liste des évidences nécessaires pour garantir les différentes propriétés de non-répudiation. Cependant, ces évidences sont-elles bien formées ? Comment le vérifier ?

La méthode décrite dans la section précédente construit des chaînages d'authentifications en partant des évidences, et jusqu'aux termes élémentaires les composant. Elle permet donc de vérifier quels agents interviennent dans la composition des évidences. Il est donc facile de vérifier si l'agent duquel se protège le possesseur des évidences intervient dans leur conception. Et comme toutes les possibilités de composition des évidences sont considérées, s'il en existe une qui n'utilise pas l'agent adverse, alors cela signifie que les évidences ne sont pas correctes et ne garantissent pas la participation de l'agent adverse au protocole.

Une autre tâche liée aux protocoles de non-répudiation n'est jamais considérée dans les analyses : le rôle du juge, lorsqu'on lui présente des évidences en cas de conflit. Ce juge va devoir analyser les évidences produites, en général avec l'aide de divers agents pour les déchiffrer, et conclure si elles sont valables ou pas, et donc si l'agent les possédant a bien une preuve de la participation au protocole de l'agent adverse. Ce rôle, toujours ignoré, est important, d'une part pour avoir connaissance de sa complexité, des agents devant interagir avec lui (en général des tiers de confiance), mais aussi vérifier qu'aucun agent malhonnête ne peut agir à ce moment-là.

c - Importance de la propriété de timeliness

La propriété de timeliness garantit que chaque participant du protocole pourra toujours atteindre un état final, et normalement en préservant l'équité. Cette propriété pourrait paraître anecdotique car pouvant être traitée avec des timeouts d'abandon. Cependant, elle est très importante, car son absence peut permettre de bâtir des dénis de services : lors d'une attente prolongée de réponse, les agents laissent des canaux de communication ouverts, et il est donc facile d'émettre de grandes quantités de messages (par exemple le message d'ouverture d'une session) pour saturer ces canaux.

Mais dans des contextes où des attaques par déni de service ne sont pas possibles (par exemple lors d'un paiement par carte bancaire chez un commerçant), l'utilisation d'un protocole qui ne respecte pas la timeliness peut être préférable, car ils sont plus simples et en général déterministes.

Il serait donc très intéressant d'étudier les cas pour lesquels la timeliness est importante, et d'identifier comment concevoir un protocole qui la garantit. Bien sûr, il faudrait aussi concevoir des méthodes permettant de vérifier que cette propriété est bien vérifiée.

Par exemple, pour étudier l'équité d'un protocole, la timeliness joue un rôle essentiel car c'est elle qui garantit que la vérification peut être faite uniquement en fin de session.

2 Composition de services Web

La composition de services Web est un domaine « à la mode » qui est en plein essor, en lien avec les nouvelles méthodes de développement de logiciels comme SOA. Cet engouement soulève également de nombreux problèmes pour un bon fonctionnement de ces logiciels. Si les services Web permettent de simplifier l'interopérabilité entre applications, et semblent apporter une plus grande souplesse, il faut cependant vérifier la compatibilité des services impliqués. Cette compatibilité intervient au niveau du travail attendu du service, mais aussi au niveau de la sécurité mise en œuvre.

2.1 Compatibilité de la composition

Lorsqu'une composition abstraite est fixée, il s'agit de déterminer les bonnes instances de services permettant par exemple de satisfaire la requête d'un client. Ce type de problème peut être traité par une approche à base de contraintes dynamiques, construites et propagées durant le déroulement de la composition. Elles permettent d'aider à sélectionner les services les mieux adaptés, mais aussi à modifier l'instanciation si un service s'avère défectueux.
L'objectif est donc d'arriver à gérer des contraintes à différents niveaux (contraintes de la composition abstraite, contraintes décrivant les propriétés des services, requêtes de l'utilisateur), et tout cela ne peut se faire avec un simple solveur. Il faut définir des mécanismes d'analyse et de déduction en plus de la résolution.

2.2 Compatibilité de la sécurité

L'un des autres problèmes induits par la combinaison de services est la sécurité. Chaque service Web possède une politique de sécurité, et la combinaison doit en tenir compte. Ainsi, vérifier une politique de sécurité peut être vu comme un processus de raisonnement par rapport à des connaissances indiquées par un client potentiel. C'est ainsi par exemple qu'on peut modéliser du contrôle d'accès.
Il est donc nécessaire de combiner des solveurs et des outils de vérification pour construire des compositions satisfaisant des politiques de sécurité.

3 Analyse de systèmes infinis

Les travaux présentés tout au long de ce rapport concernent l'étude de systèmes infinis, par des techniques de déduction automatique.

Les résultats présentés sur la normalisation efficace de termes ont eu pour objectif de n'appliquer une réécriture qu'une seule fois, grâce à une représentation extrêmement compacte des termes. Or, il s'avère que des systèmes de réécriture non terminants peuvent être représentés ainsi, la procédure de complétion pouvant s'arrêter. Il serait très intéressant de vérifier pour quels types de systèmes non terminants cette représentation est adaptée, et d'étudier comment les systèmes obtenus peuvent être utilisés, pour le problème du mot par exemple.

Nous avons également vu avec les travaux sur les algèbres approximantes que la déduction automatique peut être utilisée, non pas pour tenter d'atteindre un but fixé, mais dans un objectif purement prospectif. L'exploration partielle d'un espace infini de propriétés peut permettre d'avoir une meilleure connaissance de cet espace, et de poser des hypothèses dont la véracité peut ensuite être vérifiée. Ces travaux, initiés suite à une rencontre fortuite devant une imprimante dans un local borgne de l'Université de Stony Brook, m'ont ouvert les yeux sur le grand nombre de chercheurs et professionnels pour lesquels des outils de déduction automatique seraient très utiles, et je reste depuis attentif aux activités des personnes rencontrées, que ce soit dans le cadre d'activité de recherche ou d'enseignement.

Bibliographie

Les références bibliographiques utilisées précédemment dans de document sont regroupées dans ce chapitre par sections, chacune correspondant au thème d'un chapitre.

1 Bibliographie sur la déduction automatique

[1] OCaml. INRIA. `http://caml.inria.fr/ocaml/`.

[2] F. Baader. Combination of Compatible Reduction Orderings that are Total on Ground Terms. In V. Pratt, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science, LICS'97*, pages 2–13, Los Alamitos, CA, 1997.

[3] L. Bachmair and N. Dershowitz. Completion for Rewriting Modulo a Congruence. *Theoretical Computer Science*, 67(2-3) :173–202, October 1989.

[4] L. Bachmair and H. Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer-Verlag, July 1990.

[5] L. Bachmair and H. Ganzinger. Associative-Commutative Superposition. Technical report MPI-I-93-267, Max Planck Institut für Informatik, Saarbrücken, 1993.

[6] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2) :172–192, 1995.

[7] N. Bjorner and V. Sofronie-Stokkermans, editors. *23rd International Conference on Automated Deduction*, volume 6803 of *Lecture Notes in Computer Science*, Wroclaw, Poland, 2011. Springer.

[8] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland), 1990.

[9] J. Hsiang and M. Rusinowitch. Proving Refutational Completeness of Theorem Proving Strategies : The Transfinite Semantic Tree Method. *Journal of the ACM*, 38(3) :559–587, July 1991.

[10] J.-M. Hullot. *Compilation de Formes Canoniques dans les Théories équationelles*. Thèse de Doctorat de Troisième Cycle, Université de Paris Sud, Orsay (France), 1980.

[11] J.-P. Jouannaud and C. Kirchner. Solving Equations in Abstract Algebras : a Rule-based Survey of Unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge (MA, USA), 1991.

[12] J.-P. Jouannaud and H. Kirchner. Completion of a Set of Rules Modulo a Set of Equations. *SIAM Journal of Computing*, 15(4) :1155–1194, 1986. Preliminary version in Proceedings

11th ACM Symposium on Principles of Programming Languages, Salt Lake City (USA), 1984.

[13] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with Symbolic Constraints. *Revue d'Intelligence Artificielle*, 4(3) :9–52, 1990. Special issue on Automatic Deduction.

[14] P. Narendran and M. Rusinowitch. Any Ground Associative-Commutative Theory Has a Finite Canonical System. In R. V. Book, editor, *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*. Springer-Verlag, 1991.

[15] R. Nieuwenhuis and A. Rubio. AC-Superposition with Constraints : no AC-unifiers Needed. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 545–559. Springer-Verlag, June 1994.

[16] R. Nieuwenhuis and A. Rubio. Paramodulation with Built-in AC-Theories and Symbolic Constraints. *Journal of Symbolic Computation*, 23(1) :1–21, 1997.

[17] J. K. Ousterhout. *Tcl and the Tk Toolkit*, volume ISBN 0.201.63337.X. Addison-Wesley, 1994.

[18] E. Paul. A General Refutational Completeness Result for an Inference Procedure Based on Associative-Commutative Unification. *Journal of Symbolic Computation*, 14(6) :577–618, 1992.

[19] E. Paul. E-Semantic Tree. Unpublished paper (E-mail : etienne.paul@issy.cnet.fr), 70 pages, 1994.

[20] G. E. Peterson and M. E. Stickel. Complete Sets of Reductions for Some Equational Theories. *Journal of the ACM*, 28 :233–264, 1981.

[21] G. Plotkin. Building-in Equational Theories. *Machine Intelligence*, 7 :73–90, 1972.

[22] G. A. Robinson and L. T. Wos. Paramodulation and First-order Theorem Proving. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, volume 4, pages 135–150. Edinburgh University Press, 1969.

[23] J. A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12 :23–41, 1965.

[24] A. Rubio. *Automated Deduction with Constrained Clauses*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, April 1994.

[25] A. Rubio and R. Nieuwenhuis. A Precedence-Based Total AC-Compatible Ordering. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 1993.

[26] M. Rusinowitch and L. Vigneron. Automated Deduction with Associative Commutative Operators. In Ph. Jorrand and J. Kelemen, editors, *Fundamental of Artificial Intelligence Research*, volume 535 of *Lecture Notes in Computer Science*, pages 185–199. Springer-Verlag, 1991.

[27] M. Rusinowitch and L. Vigneron. Automated Deduction with Associative-Commutative Operators. *Applicable Algebra in Engineering, Communication and Computation*, 6(1) :23–56, January 1995. Also available as INRIA Research Report 1896, or CRIN Research Report 93-R-252.

[28] L. Vigneron. Associative-Commutative Deduction with Constraints. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 530–544. Springer-Verlag, June 1994.

[29] L. Vigneron. *Déduction automatique avec contraintes symboliques dans les théories équationnelles*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, November 1994.

[30] L. Vigneron. Positive Deduction modulo Regular Theories. In H. K. Büning, editor, *Annual Conference of the European Association for Computer Science Logic, Paderborn (Germany)*, volume 1092 of *Lecture Notes in Computer Science*, pages 468–485. Springer-Verlag, September 1995. Selected paper.

[31] U. Wertz. First-Order Theorem Proving Modulo Equations. Technical Report MPI-I-92-216, Max Planck Institut für Informatik, April 1992.

2 Bibliographie sur la normalisation efficace

[32] L. Bachmair, T. Chen, and I. V. Ramakrishnan. Associative-Commutative Discrimination Nets. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT '93*, volume 668 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 1993.

[33] L. Bachmair, I. V. Ramakrishnan, and L. Vigneron. Efficient Term Rewriting Techniques. In M. Rodriguez-Artalejo, editor, *Proceedings of the CCL'96 Workshop*, San Lorenzo, Spain, September 1996.

[34] L. Bachmair and A. Tiwari. Abstract Congruence Closure and Specializations. In D. McAllester, editor, *Conference on Automated Deduction, CADE*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Pittsburgh, PA, June 2000. Springer-Verlag.

[35] L. Bachmair, A. Tiwari, and L. Vigneron. Abstract Congruence Closure. *Journal of Automated Reasoning*, 31(2) :129–168, 2003.

[36] A. M. Ballantyne and D. S. Lankford. New Decision Algorithms for Finitely Presented Commutative Semigroups. *Comp. and Maths. with Appls.*, 7 :159–165, 1981.

[37] T. Becker and V. Weispfenning. *Gröbner Bases : a Computational Approach to Commutative Algebra*. Springer, Berlin, 1993.

[38] L. P. Chew. An Improved Algorithm for Computing with Equations. In *21st Annual Symposium on the Foundations of Computer Science*, pages 108–117, 1980.

[39] L. P. Chew. *Normal Forms in Term Rewriting Systems*. PhD thesis, Faculty of Purdue University, 1981.

[40] J. Christian. Flatterms, Discrimination Nets, and Fast Term Rewriting. *Journal of Automated Reasoning*, 10(1) :95–113, February 1993.

[41] N. Dershowitz and Z. Manna. Proving Termination with Multiset Orderings. *Communications of the ACM*, 22(8) :465–476, 1979.

[42] E. Domenjoud and F. Klay. Shallow AC Theories. In *Proceedings of the 2nd CCL Workshop*, La Escala, Spain, September 1993.

[43] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the Common Subexpressions Problem. *Journal of the ACM*, 27(4) :758–771, 1980.

[44] T. Evans. The Word Problem for Abstract Algebras. *Journal of London Mathematical Society*, 26 :64–71, 1951.

[45] T. Evans. Word Problems. *Bulletin of the American Mathematical Society*, 84(5) :789–802, 1978.

[46] C. M. Hoffmann and M. J. O'Donnell. Programming with Equations. *ACM Transactions on Programming Languages and Systems*, 4(1) :83–112, January 1982.

[47] C. M. Hoffmann and M. J. O'Donnell. Implementation of an Interpreter for Abstract Equations. *ACM*, 4(0-89791-125-3/84/001/0111) :111–121, 1983.

[48] G. Huet and J.-J. Lévy. Computations in Orthogonal Rewriting Systems (I and II). In J.-L. Lassez and G. Plotkin, editors, *Computational Logic*, pages 395–414 (I), 415–443 (II). The MIT Press, 1991.

[49] D. Kapur. Shostak's Congruence Closure as Completion. In . Comon, editor, *Rewriting Techniques and Applications, RTA*, volume 1103 of *Lecture Notes in Computer Science*, pages 23–37, Sitges, Spain, July 1997. Springer-Verlag.

[50] U. Koppenhagen and E. W. Mayr. An Optimal Algorithm for Constructing the Reduced Gröbner Basis of Binomial Ideals. In Y. D. Lakshman, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 55–62, 1996.

[51] D. Kozen. Complexity of Finitely Presented Algebras. In *9th Annual ACM Symposium on Theory of Computing*, pages 164–177, 1977.

[52] C. Marche. On Ground AC-completion. In R. V. Book, editor, *4th International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 1991.

[53] E. W. Mayr and A. R. Meyer. The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. *Advances in Mathematics*, 46 :305–329, 1982.

[54] P. Narendran and M. Rusinowitch. Any Ground Associative-Commutative Theory Has a Finite Canonical System. In R. V. Book, editor, *4th International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 1991.

[55] G. Nelson and D. Oppen. Fast Decision Procedures Based on Congruence Closure. *Journal of the Association for Computing Machinery*, 27(2) :356–364, April 1980.

[56] G. Nelson and D. C. Oppen. Fast Decision Procedures based on Congruence Closure. *Journal of the ACM*, 27(2) :356–364, 1980.

[57] M. J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer, Berlin, West Germany, 1977.

[58] G. E. Peterson and M. E. Stickel. Complete Sets of Reductions for Some Equational Theories. *Journal of the ACM*, 28(2) :233–264, April 1981.

[59] D. Plaisted and A. Sattler-Klein. Proof Lengths for Equational Completion. *Information and Computation*, 125 :154–170, 1996.

[60] A. Rubio and R. Nieuwenhuis. A Precedence-based Total AC-compatible Ordering. In C. Kirchner, editor, *Proceedings of the 5 Intl. Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 1993.

[61] R. C. Sekar and I. V. Ramakrishnan. Programming in Equational Logic : Beyond Strong Sequentiality. *Information and Computation*, 104(1) :78–109, May 1993.

[62] D. J. Sherman and N. Magnier. Factotum : Automatic and Systematic Sharing Support for Systems Analyzers. In *Proc. TACAS*, volume 1384 of *Lecture Notes in Computer Science*, Lisbon, Portugal, 1998. Springer.

[63] R. E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1) :1–12, 1984.

[64] W. Snyder. A Fast Algorithm for Generating Reduced Ground Rewriting Systems from a Set of Ground Equations. *Journal of Symbolic Computation*, 15(7), 1993.

[65] R. M. Verma. A Theory of Using History for Equational Systems with Applications. *Journal of the ACM*, 42(5) :984–1020, September 1995.

[66] R. M. Verma and I. V. Ramakrishnan. Nonoblivious Normalization Algorithms for Nonlinear Systems. In M. S. Paterson, editor, *Proceedings 17th ICALP Conference*, volume 443 of *Lecture Notes in Computer Science*, pages 370–385, Coventry (England), 1990. Springer.

3 Bibliographie sur les ensembles approximants

[67] M. Banerjee. *A Cathegorical Approach to the Algebra and Logic of the Indiscernible*. PhD thesis, Mathematics Department, University of Calcutta, India, 1994.

[68] M. Banerjee and M. K. Chakraborty. Rough Algebra. *Bulletin of Polish Academia of Science (Mathematics)*, 41(4) :293–297, 1993.

[69] J. C. C. McKinsey and A. Tarski. The Algebra of Topology. *Annales of Mathematics*, 45 :141–191, 1944.

[70] M. Novotny and Z. Pawlak. On Rough Equalities. *Bulletin of Polish Academia of Science (Mathematics)*, 33 :99–104, 1985.

[71] E. Orlowska and Z. Pawlak. Expressive Power of Knowledge Representation Systems. *International Journal of Man-Machine Studies*, 20 :485–500, 1984.

[72] Z. Pawlak. Rough Sets. *International Journal of Computation and Information Sciences*, 11 :341–356, 1982.

[73] Z. Pawlak. *Rough Sets*. Theory and Decision Library. Kluwer Academic Publishers, 1991.

[74] Z. Pawlak. Rough Sets – A New Approach to Vagueness. In L. Zadeh and J. Kacpryzk, editors, *Fuzzy Logic for the Management of Uncertainty*, pages 105–118. John Wiley & Sons, Inc., 1992.

[75] H. Rasiowa. *An Algebraic Approach to Non-Classical Logics*, volume 78 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1974.

[76] A. Skowron and J. W. Grzymala-Busse. From Rough Set Theory to Evidence Theory. In M. Fedrizzi and J. Kacprzyk, editors, *Advances in the Dempster Shafer Theory of Evidence*, pages 193–236. John Wiley & Sons, Inc., 1994.

[77] M. H. Stone. Boolean algebras and their relation to topology. *Proceedings of National Academy of Sciences*, 20 :197–202, 1934.

[78] L. Vigneron and A. Wasilewska. Modal and Rough Algebras. In P. Borne, G. Dauphin-Tanguy, C. Sueur, and S. El Khattabi, editors, *Proceedings of the CESA'96 International Multiconference (Computational Engineering in Systems Applications), Symposium on Modelling, Analysis and Simulation*, pages 1107–1112, Lille (France), July 1996. IMACS/IEEE.

[79] A. Wasilewska. Topological Rough Algebras. In T. Y. Lin and N. Cercone, editors, *Rough Sets and Data Mining : Analysis of Imprecise Data*, pages 411–425. Kluwer Academic Publishers, 1996.

[80] A. Wasilewska, M. Lifantsev, and L. Vigneron. A Decision Procedure for Proofs Visualisation. In E. Orlowska, editor, *Session of the Polish Association for Logic and Philosophy of Science dedicated to the memory of Andrzej Mostowski and Helena Rasiowa*, Warsaw (Poland), November 1999.

[81] A. Wasilewska and L. Vigneron. Rough Equality Algebras. In *Proceedings of the Workshop on Rough Set Theory (RST'95), at the 2nd Joint Conference on Information Sciences, Wrightsville Beach (North Carolina)*, September 1995.

[82] A. Wasilewska and L. Vigneron. Rough Algebras and Automated Deduction. In L. Polkowski and A. Skowron, editors, *Rough Sets in Knowledge Discovery 1*, pages 261–275. Springer-Verlag, July 1998.

[83] W. Ziarko. Rough Sets, Fuzzy Sets and Knowledge Discovery. In *Proceedings of the International Workshop on Rough Sets and Knowledge Discovery (RSKD'93)*, pages 12–15, Alberta (Canada), October 1993. Springer-Verlag.

4 Bibliographie sur la spécification de protocoles de sécurité

[84] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P Hankes Drielsma, P.-C. Heám, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[85] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In E. Brinksma and K. Guldstrand Larsen, editors, *Computer-Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–354. Springer, 2002.

[86] AVISPA. Deliverable 2.1 : The High-Level Protocol Specification Language. Available at `http://www.avispa-project.org/publications.html`, 2003.

[87] AVISPA. Deliverable 2.3 : The Intermediate Format. Available at `http://www.avispa-project.org/publications.html`, 2003.

[88] AVISPA. Deliverable 6.1 : List of selected problems. Available at `http://www.avispa-project.org`, 2003.

[89] S. M. Bellovin and M. Merritt. Encrypted Key Exchange : Password-Based Protocols Secure Against Dictionary Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84, May 1992.

[90] B. Blanchet. Automatic Verification of Correspondences for Security Protocols. *Journal of Computer Security*, 17(4) :363–434, July 2009.

[91] Y. Boichut, N. Kosmatov, and L. Vigneron. Validation of Prouve Protocols using the Automatic Tool TA4SP. In *Proceedings of 3rd Taiwanese-French Conference on Information Technology (TFIT)*, pages 467–480, Nancy, France, March 2006.

[92] D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Society Press, 1997.

[93] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proceedings of the 28th International Conference on Automata, Language and Programming, ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 667–681. Springer, 2001.

[94] S. Brackin. Evaluating and Improving Protocol Analysis by Automatic Proof. In *Proceedings of 11th IEEE Computer Security Foundations Workshop, CSFW*, pages 138–152. IEEE Computer Society Press, 1998.

[95] S. Brackin, C. Meadows, and J. Millen. CAPSL Interface for the NRL Protocol Analyzer. In *Proceedings of ASSET'99, IEEE Symposium on Application-Specific Systems and Software Engineering Technology*. IEEE Computer Society Press, 1999.

[96] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1) :18–36, 1990.

[97] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Automated Software Engineering. Proceedings of the Workshop on Specification and Automated Processing of Security Requirements, SAPS'04*, pages 193–205, Austria, September 2004. Austrian Computer Society.

[98] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature : Version 1.0, 17 Nov. 1997. URL `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

[99] E. Cohen. TAPS : A First-Order Verifier for Cryptographic Protocols. In *Proceedings of the 13th Computer Security Foundations Workshop, CSFW*, pages 144–158. IEEE Computer Society Press, 2000.

[100] L. Compagna. *SAT-based Model-Checking of Security Protocols*. Phd thesis, Università degli Studi di Genova, Italy, and University of Edinburgh, Scotland, 2005.

[101] C. J. F. Cremers. The Scyther Tool : Verification, Falsification, and Analysis of Security Protocols. In *Proceedings of 20th International Conference on Computer Aided Verification, CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418, Princeton, USA, 2008. Springer.

[102] G. Denker and J. Millen. CAPSL Intermediate Language. In N. Heintze and E. Clarke, editors, *Proceedings of Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999. URL for CAPSL and CIL : `http://www.csl.sri.com/~millen/capsl/`.

[103] G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000. Available at `http://www.csl.sri.com/~millen/capsl/`.

[104] D. Denning and G. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 8(24), 1981.

[105] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[106] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[107] A. Durante, R. Focardi, and R. Gorrieri. CVS : A Compiler for the Analysis of Cryptographic Protocols. In *Proceedings of 12th Computer Security Foundations Workshop, CSFW*, pages 203–212. IEEE Computer Society Press, Mordano, Italy, 1999.

[108] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA : Cryptographic Protocol Analysis Modulo Equational Properties. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2009.

[109] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Narrowing Cryptographic Protocols. Technical Report 99-R-303, LORIA, Vandoeuvre les Nancy, December 1999. URL : `http://www.loria.fr/equipes/cassis/softwares/casrul/`.

[110] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160, St Gilles (Réunion, France), November 2000. Springer.

[111] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3) :872–923, May 1994.

[112] G. Leduc and F. Germeau. Verification of Security Protocols using LOTOS – Method and Application. *Computer Communications, special issue on Formal Description Techniques in Practice*, 23(12) :1089–1103, 2000.

[113] G. Lowe. Casper : a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1) :53–84, 1998.

[114] The Maude System. URL : `http://maude.csl.sri.com`.

[115] C. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*, 1(1) :5–36, 1992.

[116] C. Meadows. The NRL Protocol Analyzer : An Overview. *Journal of Logic Programming*, 26(2) :113–131, 1996.

[117] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of the DARPA Information and Survivability Conference and Exposition, DISCEX*, pages 237–250. IEEE Computer Society Press, January 2000.

[118] J. Millen. CAPSL : Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.

[119] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Mur-phi. In *Proceedings of Symposium on Security and Privacy*, pages 141–151, Oakland, CA, 1997. IEEE Computer Society Press.

[120] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1-2) :85–128, 1998.

[121] The PVS Specification and Verification System. URL : `http://pvs.csl.sri.com`.

[122] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proceedings of 8th IEEE Computer Security Foundations Workshop, CSFW*, pages 98–107, Kenmare, Ireland, 1995. IEEE Computer Society Press.

[123] A. W. Roscoe and M. Goldsmith. The Perfect "Spy" for Model-Checking Cryptoprotocols. In *Proceeding of DIMACS Workshop on Design and Formal Verification of Crypto Protocols*, 1997.

[124] D. Song, S. Berezin, and A. Perrig. Athena : a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9 :47–74, 2001.

[125] P. F. Syverson and I. Cervesato. The Logic of Authentication Protocols. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 63–136. Springer, 2001.

[126] L. Vigneron. A Tool helping to Design Cryptographic Protocols (tutorial). In 4th Conference on Security and Network Architectures (SAR'05), Batz sur Mer, France, June 2005.

[127] C. Weidenbach. Towards an Automatic Analysis of Security Protocols. In H. Ganzinger, editor, *Proceedings of 16th International Conference on Automated Deduction, CADE*, number 1632 in Lecture Notes in Computer Science, pages 378–382, Trento, Italy, 1999. Springer.

5 Bibliographie sur l'analyse de protocoles de sécurité

[128] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P Hankes Drielsma, P.-C. Heám, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

[129] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In E. Brinksma and K. Guldstrand Larsen, editors, *Computer-Aided Verification, CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–354. Springer, 2002.

[130] A. Armando and L. Compagna. SATMC : a SAT-based Model Checker for Security Protocols. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence, JELIA*, volume 3229 of *Lecture Notes in Artificial Intelligence*, pages 730–733, Lisbon, Portugal, 2004. Springer.

[131] N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. *Computer Communications*, 23(17), 2000.

[132] G. Ateniese, M. Steiner, and G. Tsudik. New Multiparty Authentication Services and Key Agreement Protocols. *IEEE Journal on Selected Areas in Communications*, 18(4) :628–639, 2000.

[133] D. Basin. Lazy Infinite-State Analysis of Security Protocols. In R. Baumgart, editor, *Secure Networking — CQRE (Secure)'99*, volume 1740 of *Lecture Notes in Computer Science*, Dusseldorf, Germany, 1999. Springer.

[134] D. Basin, S. Mödersheim, and L. Viganò. OFMC : A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3) :181–208, 2005.

[135] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic verification of security protocols using approximations. Technical Report RR-5727, INRIA, 2005.

[136] M. S. Bouassida, N. Chridi, I. Chrisment, O. Festor, and L. Vigneron. Automatic Verification of a Key Management Architecture for Hierarchical Group Protocols. In F. Cuppens and H. Debar, editors, *Proceedings of 5th Conference on Security and Network Architectures (SAR)*, pages 381–397, Seignosse, France, June 2006.

[137] M. S. Bouassida, N. Chridi, I. Chrisment, O. Festor, and L. Vigneron. Automated Verification of a Key Management Architecture for Hierarchical Group Protocols. *Annals of Telecommunications*, 62(11-12) :1365–1387, 2007.

[138] M. S. Bouassida, I. Chrisment, and O. Festor. BALADE : Diffusion multicast sécurisée d'un flux multimédia multi-sources séquentielles dans un environnement ad-hoc. In R. Castanet, editor, *CFIP*, pages 531–546, Bordeaux, France, 2005. Hermès Lavoisier,.

[139] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval. Mutual authentication and group key agreement for low-power mobile devices. *Journal of Computer Communications*, 27(17) :1730–1737, 7 2004. Special Issue on Security and Performance in Wireless and Mobile Networks. Elsevier Science.

[140] J. Cederquist, R. Corin, and M. T. Dashti. On the Quest for Impartiality : Design and Analysis of a Fair Non-repudiation Protocol. In S. Qing, W. Mao, Lopez ; J., and G. Wang, editors, *Information and Communications Security, 7th International Conference, ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 27–39, Beijing (China), 2005. Springer.

[141] Y. Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques*. Phd thesis, Université Henri Poincaré, Nancy, December 2003.

[142] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. In M. Baaz and J. A. Makowsky, editors, *Computer Science Logic (CSL 03) and 8th Kurt Gödel Colloquium (8th KCG)*, volume 2803 of *Lecture Notes in Computer Science*, pages 128–141, Vienna, Austria, August 2003. Springer.

[143] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols (short paper). In *Proceedings of ASE-2001 : The 16th IEEE Conference on Automated Software Engineering*, pages 373–376, San Diego (CA), November 2001. IEEE CS Press.

[144] Y. Chevalier and L. Vigneron. Towards Efficient Automated Verification of Security Protocols. In *Proceedings of the Verification Workshop (VERIFY'01) (in connection with IJCAR'01), Università degli studi di Siena, TR DII 08/01*, pages 19–33, 2001.

[145] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. Guldstrand Larsen, editors, *14th International Conference on Computer Aided Verification, CAV'2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 324–337, Copenhagen (Denmark), July 2002. Springer.

[146] Y. Chevalier and L. Vigneron. Strategy for Verifying Security Protocols with Unbounded Message Size. *Journal of Automated Software Engineering*, 11(2) :141–166, April 2004.

[147] N. Chridi. Vérification de protocoles de groupes. Rapport de DEA, LORIA – Université Henri Poincaré, Nancy, 2005.

[148] N. Chridi. *Contributions à la vérification automatique de protocoles de groupes*. Thèse de doctorat, Université Henri Poincaré - Nancy 1, 09 2009.

[149] N. Chridi and L. Vigneron. Modélisation des propriétés de sécurité de protocoles de groupe. In *Actes du 1er Colloque sur les Risques et la Sécurité d'Internet et des Systèmes, CRiSIS*, pages 119–132, Bourges, France, October 2005. Élu meilleur papier de la conférence.

[150] N. Chridi and L. Vigneron. Sécurité des communications de groupe. *Revue de l'Électricité et de l'Électronique*, 6/7 :51–60, juin/juillet 2006.

[151] N. Chridi and L. Vigneron. Strategy for Flaws Detection based on a Services-driven Model for Group Protocols. In Benjamin Blanc, Arnaud Gotlieb, and Claude Michel, editors, *Proceedings of the 1st Workshop on Constraints in Software Testing, Verification and Analysis, CSTVA*, pages 88–99, Nantes, France, September 2006.

[152] N. Chridi and L. Vigneron. *Strategy for Flaws Detection based on a Services-driven Model for Group Protocols*, chapter 24, pages 361–370. Future and Trends in Constraint Programming. ISTE, April 2007.

[153] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Authenticity and Provability - A Formal Framework. In *Infrastructure Security Conference, InfraSec*, volume 2437, pages 227–245, 2002.

[154] H. Hassan, A. Bouabdallah, H. Bettahar, and Y. Challal. HI-KD : Hash-based Hierarchical Key Distribution for Group Communication. In *IEEE Infocom*, 2005.

[155] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160, St Gilles (Réunion, France), November 2000. Springer.

[156] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. In *ACM Transactions on Information and System Security (TISSEC)*, volume 7, pages 60–96. ACM Press New York, February 2004.

[157] F. Klay and L. Vigneron. Automatic Methods for Analyzing Non-repudiation Protocols with an Active Intruder. In P. Degano, J. D. Guttman, and F. Martinelli, editors, *Formal Aspects in Security and Trust, 5th International Workshop, FAST 2008, Malaga, Spain, October 9-10, 2008, Revised Selected Papers*, volume 5491 of *Lecture Notes in Computer Science*, pages 192–209. Springer, 2009.

[158] J. Liu and L. Vigneron. Design and Verification of a Non-Repudiation Protocol Based on Receiver-Side Smart Card. *IET Information Security*, 4(1) :15–29, March 2010.

[159] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of the DARPA Information and Survivability Conference and Exposition, DISCEX*, pages 237–250. IEEE Computer Society Press, January 2000.

[160] C. Meadows and P. Syverson. Formalizing gdoi group key management requirements in npatrl. In *CCS '01 : Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 235–244. ACM Press, 2001.

[161] J. Nam, S. Kim, and D. Won. A weakness in the bresson-chevassut-essiari-pointcheval's group key agreement scheme for low-power mobile devices. *IEEE Communication Letters*, 9(5) :429–431, 2005.

[162] O. Pereira and J.-J. Quisquater. Security Analysis of the Cliques Protocols Suites : First Results. In *IFIP TC11 16th Annual Working Conference on Information Security, SEC*, pages 151–166, Paris, France, 2001. Kluwer.

[163] O. Pereira and J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4) :555–580, 2003.

[164] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[165] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols.* Addison Wesley, 2000.

[166] J. Santiago and L. Vigneron. Study for Automatically Analysing Non-repudiation. In *Actes du 1er Colloque sur les Risques et la Sécurité d'Internet et des Systèmes, CRiSIS*, pages 157–171, Bourges, France, October 2005.

[167] J. Santiago and L. Vigneron. Optimistic Non-repudiation Protocol Analysis. In D. Sauveron et al., editor, *Proceedings of the Workshop in Information Security Theory and Practices (WISTP'2007), Smart Cards, Mobile and Ubiquitous Computing Systems*, volume 4462 of *Lecture Notes in Computer Science*, pages 90–101, Heraklion (Greece), May 2007. Springer.

[168] J. Santos Santiago. *Spécification et analyse de protocoles complexes dans AVISPA*. Thèse de doctorat, Université Nancy 2, novembre 2006.

[169] V. Shmatikov and J. C. Mitchell. Analysis of Abuse-Free Contract Signing. In Y. Frankel, editor, *Proceedings of the 4th International Conference on Financial Cryptography, FC*, volume 1962 of *Lecture Notes in Computer Science*, pages 174–191, Anguilla (British West Indies), 2000. Springer.

[170] G. Steel, A. Bundy, and M. Maidl. Attacking a protocol for group key agreement by refuting incorrect inductive conjectures. In D. Basin and M. Rusinowitch, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 3097 in LNAI, pages 137–151, Cork, Ireland, July 2004. Springer.

[171] M. Steiner, G. Tsudik, and M. Waidner. Cliques : A new approach to group key agreement, 1998.

[172] M. Taghdiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *FORTE*, pages 240–256, 2003.

[173] M. Turuani. The CL-Atse Protocol Analyser. In F. Pfenning, editor, *Proceedings of 17th International Conference on Rewriting Techniques and Applications, RTA*, Lecture Notes in Computer Science, Seattle (WA), August 2006. Springer.

[174] J. Zhou and D. Gollmann. Towards Verification of Non-repudiation Protocols. In *Proceedings of International Refinement Workshop and Formal Methods Pacific*, pages 370–380, Canberra, Australia, 1998.

Annexe

Cette annexe contient un article référence par chapitre.

Article de référence du Chapitre 1

L. Vigneron.
Positive Deduction modulo Regular Theories.
In H.K. Büning, editor,
Proceedings of the Annual Conference of the European Association for Computer Science Logic, Paderborn (Germany),
volume 1092 of Lecture Notes in Computer Science, pages 468-485,
September 1995.
Springer-Verlag.
Selected paper.

Positive Deduction modulo Regular Theories *

Laurent Vigneron

CRIN-CNRS & INRIA Lorraine
B.P.239, 54506 Vandœuvre-lès-Nancy Cedex, France
E-mail: Vigneron@Loria.Fr

Abstract. We propose a *new technique* for dealing with an equational theory $\mathcal{E}$ in the clausal framework. This consists of the definition of two inference rules called *contextual superposition* and *extended superposition*, and of an *algorithm* for computing the only needed applications of these last inference rules only by examining the axioms of $\mathcal{E}$. We prove the refutational completeness of this technique for a class of theories $\mathcal{E}$ that include *all the regular theories*, i.e. any theory whose axioms preserve variables. This generalizes the results of Wertz [31] and Paul [17] who could not prove the refutational completeness of their superposition-based systems for any regular theory.
We also combine a *collection of strategies* that decrease the number of possible deductions, without loss of completeness: the superposition strategy, the positive ordering strategy, and a simplification strategy.
These results have been implemented in a theorem prover called `DATAC`, for the case of commutative, and associative and commutative theories. It is an interesting tool for comparing the efficiency of strategies, and practical results will follow.

1 Introduction

The paramodulation rule permits one to deal with the equality predicate without explicitly describing its properties of reflexivity, symmetry, transitivity and functional reflexivity. It is also based on a notion of replacement. Over time, several refinements have been added to this rule. Brand [6] has shown that only the reflexivity axiom $x \simeq x$ is needed. Peterson [18] has shown that paramodulations into variables are useless. Hsiang and Rusinowitch [10] have introduced ordering restrictions to the application of these rules, and have proved the completeness of the following *ordering strategy*: *each inference step has to be applied between maximal (w.r.t. an ordering) literals in clauses, and in each paramodulation step, a term cannot be replaced by a bigger one.*
Other refinements, such as the superposition strategy which applies replacements only in biggest sides of equations, and clausal simplifications which delete redundant clauses, have followed [4].

* This work was done during a fellowship at the University of Stony Brook (NY, USA), funded by the *Institut National de Recherche en Informatique et en Automatique* (France).

The complete Hsiang-Rusinowitch strategy and others are unfortunately often inefficient in the presence of clauses such as the associativity property of an operator f, $f(f(x,y),z) \simeq f(x,f(y,z))$, which produces the divergence of derivations by successive superpositions into the subterm $f(x,y)$. Other properties, such as the commutativity of an operator, induce problems with the superposition rule because they cannot be oriented.

The most established solution was proposed by Plotkin [20]. He proposed to define an equational theory $\mathcal{E}$, by extracting the above properties from the set of clauses, and to define a *unification algorithm modulo* $\mathcal{E}$. This result has been the basis of much work: Lankford and Ballantyne [14] for the particular case of associative and commutative theories, and Peterson and Stickel [19] for completion. When applying ordering strategies to theorem proving in equational theories, we need to add various additional techniques. The techniques usually proposed in equational deduction are:

1. either to add an inference rule applying replacements into axioms of $\mathcal{E}$, and therefore generating *extensions* of these axioms,
2. or to associate to each equation the set of its possible extensions, which may be used later by the superposition rule.

These extensions have been studied by Jouannaud and Kirchner [13], and Bachmair and Dershowitz [1]. Both techniques have been used by Wertz [31]; the second has been used by Paul [17] too.

We propose in this paper a *new technique* for dealing with these extensions in the clausal framework, by defining two inference rules called *contextual superposition* and *extended superposition*. We also define an *algorithm* for computing the only needed applications of these last inference rules, only by examining the axioms of $\mathcal{E}$ and generalizing the $\mathcal{E}$-redundant context notions of Jouannaud and Kirchner [13] defined for equational completion. Our inference system is defined by combining the superposition strategy and the positive ordering strategy; it is also compatible with the simplification strategy.

The positive strategy was initially proposed by Robinson [21] and has been transformed many times later. Our definition of this strategy, whose first version was presented in [25], is a much more attractive one. The usual condition is to apply superposition steps from a positive clause. Here, we mention that whenever we want to use a positive literal, it has to belong to a positive clause. A similar strategy has also been independently defined by Paul in [17].

Our positive strategy uses a particular case of the superposition calculus with selection, defined by Bachmair and Ganzinger in [3], for selecting negative literals. But in addition we define a new kind of selection on positive literals: a positive literal can be used in a deduction if it belongs to a positive clause and if it is maximal in this clause (for a given ordering).

We prove the refutational completeness of our inference system for all the equational theories $\mathcal{E}$ allowed by Wertz [31] and Paul [17], but in addition we prove it *for any regular theory* $\mathcal{E}$, i.e. any theory whose axioms preserve variables.

Moreover, our algorithm for detecting $\mathcal{E}$-redundant contexts permits a significant decreasing of the number of possible deductions.

These results have been implemented in a theorem prover called DATAC, for commutative, and associative and commutative theories. It is an interesting tool for comparing the efficiency of strategies, and practical results will follow.

The layout of this paper is the following: after introducing the basic notions in Sect. 2, we describe our inference rules in Sect. 3. Section 4 presents a procedure to compute useful contexts for extended equations. The proof of refutational completeness of the inference system is sketched in Sect. 5, but it is detailed in [30] (see also [26, 29]). Section 6 presents an example of trace with our system DATAC.

2 Notations and Definitions

Let us define some basic notions, based on the standard notations and definitions for term rewriting and unification given in [8, 12].

Let $\mathcal{E}$ be an equational theory, i.e. a set of equations. The congruence generated by this set $\mathcal{E}$ is called *$\mathcal{E}$-equality* and written $=_{\mathcal{E}}$. A *substitution* is a function replacing some variables by terms. A substitution σ is said to *$\mathcal{E}$-unify* two terms s and t if $s\sigma$ and $t\sigma$ are $\mathcal{E}$-equal, and if σ is a most general $\mathcal{E}$-unifier of s and t (see [12]). In this case, σ is a solution of the $\mathcal{E}$-unification problem $s =^{?}_{\mathcal{E}} t$.

An *atom* is an equality $l \simeq r$. A *clause* is denoted $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$, where $A_1, \ldots, A_n, B_1, \ldots, B_m$ are atoms; this means A_1 *and*... *and* A_n *implies* B_1 *or*... *or* B_m. A *literal* is an atom appearing in a clause. A *literal is positive* (resp. *negative*) if it appears on the right-hand side (resp. left-hand side) of $\rightarrow$. A *clause is positive* if it contains only positive literals, i.e. if the left-hand side of $\rightarrow$ is empty.

To express subterms and substitutions, we use *positions*. Envision a term represented as a tree; a position in a term is a node of this tree. The subterm at position p of a term t is written $t|_p$. A position is a sequence of integers: $f(t_1, \ldots, t_n)|_{i \cdot p} = t_i|_p$; the empty sequence (*empty position*) is denoted ϵ ($t|_\epsilon = t$). A position p in a term t is a *non-variable position* if $t|_p$ is not a variable. The set of all non-variable positions of a term t is denoted $\mathcal{FP}os(t)$. The term $s[t]_p$ represents the term s whose subterm at position p is t.

To decrease the number of possible deductions, we use an ordering strategy. So, we assume we are given a *simplification ordering* $>$, defined on terms and atoms. For the sake of completeness, it has to be *total* on ground $\mathcal{E}$-congruence classes and *$\mathcal{E}$-compatible*, i.e.

$$\forall s, t \text{ ground terms, if } s > t \text{ and } s \neq_{\mathcal{E}} t, \text{ then } \forall s' =_{\mathcal{E}} s,\ \forall t' =_{\mathcal{E}} t,\ s' > t'$$

So, in our inference rules, we will use this ordering to orient equations and to check the maximality of an equation w.r.t. other equations. However terms may be incomparable; we will write that a term is maximal w.r.t. another term, if it

is not smaller than or equal to this second term.
Given an equality $l \simeq r$, we will assume l is maximal w.r.t. r.

3 Inference Rules

We describe in this section a set of inference rules for applying deductions modulo an equational theory $\mathcal{E}$. These rules are based on the *superposition strategy*, a variant of the paramodulation strategy; it applies replacements only in maximal sides of equations. This superposition strategy is combined with a positive ordering strategy to prune the search space. This strategy is described in the next definition, and needs a total simplification ordering for comparing terms.

Definition 1 ((Positive Ordering Strategy)).

- *If an inference rule uses a positive literal in a clause, this clause has to be positive. In addition, the positive literal used has to be maximal in the clause.*
- *If an inference rule uses a negative literal in a clause, this literal has to be maximal w.r.t. the other negative literals of the clause.*

The first inference rule is the Equational Factoring. Its purpose is to derive clauses in which two positive equations do not have $\mathcal{E}$-equal left-hand sides. This inference rule is essential for the completeness of the superposition strategy. Note that it is applied only on positive clauses.

Definition 2 ((Equational Factoring)). $\dfrac{\rightarrow l_1 \simeq r_1, l_2 \simeq r_2, R}{r_1\sigma \simeq r_2\sigma \rightarrow l_2\sigma \simeq r_2\sigma, R\sigma}$

where σ $\mathcal{E}$-unifies l_1 and l_2, $l_1\sigma \simeq r_1\sigma$ is maximal w.r.t. $l_2\sigma \simeq r_2\sigma$ and each equation of $R\sigma$. Moreover, $l_1\sigma$ has to be maximal w.r.t. $r_1\sigma$.

The next rule stands for avoiding the addition of the reflexivity axiom $x \simeq x$ of the equality predicate. It is the only rule which can derive the empty clause, symbolizing an incoherence in the initial set of clauses, since it is the only rule which deletes a literal.

Definition 3 ((Reflexion)). $\dfrac{l \simeq r, L \rightarrow R}{L\sigma \rightarrow R\sigma}$

where σ $\mathcal{E}$-unifies l and r, and $l\sigma \simeq r\sigma$ is maximal w.r.t. each equation of $L\sigma$.

The superposition rule applies the replacement of a term by an equal one, from a positive clause. It is decomposed into a *Left* and a *Right Superposition* rule, respectively defined by

$$\frac{\rightarrow l_1 \simeq r_1, R_1 \qquad l_2 \simeq r_2, L_2 \rightarrow R_2}{l_2[r_1]_{p_2}\sigma \simeq r_2\sigma, L_2\sigma \rightarrow R_1\sigma, R_2\sigma} \quad \text{and} \quad \frac{\rightarrow l_1 \simeq r_1, R_1 \qquad \rightarrow l_2 \simeq r_2, R_2}{\rightarrow l_2[r_1]_{p_2}\sigma \simeq r_2\sigma, R_1\sigma, R_2\sigma}$$

where σ is a $\mathcal{E}$-unifier of l_1 and the subterm at position p_2 of l_2. But, even with these two inference rules, the procedure of deduction is not complete, as shown in the next example.

Example 1. Let $\mathcal{E} = \{\ f(f(x_1, x_2), x_3) \simeq f(x_1, f(x_2, x_3))\ \}$. The following clauses

$$(1) \quad \rightarrow f(a, b) \simeq c \qquad (2) \quad \rightarrow f(a, f(b, d)) \simeq e \qquad (3) \quad f(c, d) \simeq e \rightarrow$$

form an incoherent set with $\mathcal{E}$, since: $\mathcal{E}$ permits a modification of the parentheses in the left-hand side of the second clause, to obtain $\rightarrow f(f(a, b), d) \simeq e$, and replacing $f(a, b)$ by c in this term (thanks to (1)), we deduce $\rightarrow f(c, d) \simeq e$ which contradicts (3).
However, there is no possible inference step between the three initial clauses. $\Diamond$

We solve this problem by applying superpositions from *extended equations*, i.e. from equations $e[l_1]_p \simeq e[r_1]_p$, where e is a term and p a non-variable position in e. Such a pair (e, p) is called a *context*. In the previous example, a contradiction can be derived using the context $(f(f(x_1, x_2), x_3), 1)$, producing the extended equation $f(f(a, b), x_3) \simeq f(c, x_3)$ from $f(a, b) \simeq c$.

By the Critical Pairs Lemma of Jouannaud and Kirchner [13], we know that contexts can be computed. We define in Sect. 4 a procedure to compute all the possible contexts for a given equational theory $\mathcal{E}$. Given a term l, $\mathcal{C}ont(l)$ is the set of all contexts (e, p) for $\mathcal{E}$ such that $e|_p$ and l are $\mathcal{E}$-unifiable. These contexts are used in three new inference rules.

The first two rules simulate replacements from an equation or an extended equation. Indeed, we assume that the context (l, ϵ) belongs to $\mathcal{C}ont(l)$. Left and right superposition rules are therefore particular cases of the next inference rules.

Definition 4 ((Left Contextual Superposition)).

$$\frac{\rightarrow l_1 \simeq r_1, R_1 \qquad l_2 \simeq r_2, L_2 \rightarrow R_2}{l_2[e_1[r_1]_{p_1}]_{p_2}\sigma \simeq r_2\sigma, L_2\sigma \rightarrow R_1\sigma, R_2\sigma}$$

where p_2 is a non-variable position in l_2, (e_1, p_1) is a context[1] *in $\mathcal{C}ont(l_1)$, σ $\mathcal{E}$-unifies $l_2|_{p_2}$ and $e_1[l_1]_{p_1}$, $l_1\sigma \simeq r_1\sigma$ is maximal for $>$ in its clause, and $l_2\sigma \simeq r_2\sigma$ is maximal w.r.t. each atom of $L_2\sigma$. Moreover, $l_1\sigma$ has to be maximal w.r.t. $r_1\sigma$, and $l_2\sigma$ has to be maximal w.r.t. $r_2\sigma$. The replacing term in the deduced clause is the extension of the right-hand side, $e_1[r_1]_{p_1}$.*

Definition 5 ((Right Contextual Superposition)).

$$\frac{\rightarrow l_1 \simeq r_1, R_1 \qquad \rightarrow l_2 \simeq r_2, R_2}{\rightarrow l_2[e_1[r_1]_{p_1}]_{p_2}\sigma \simeq r_2\sigma, R_1\sigma, R_2\sigma}$$

where the only difference with Left Contextual Superposition is that $l_2\sigma \simeq r_2\sigma$ is maximal in its clause and maximal w.r.t. $l_1\sigma \simeq r_1\sigma$.

The next inference rule simulates a superposition between two extended equations, at the top of their maximum side.

Definition 6 ((Extended Superposition)).

$$\frac{\rightarrow l_1 \simeq r_1, R_1 \qquad \rightarrow l_2 \simeq r_2, R_2}{\rightarrow e_1[r_1]_{p_1}\sigma \simeq e_2[r_2]_{p_2}\sigma, R_1\sigma, R_2\sigma}$$

[1] (e_1, p_1) may be an empty context, i.e. $p_1 = \epsilon$.

where, given a non-empty context (e_1, p_1) *in* $\mathcal{C}ont(l_1)$ *and a non-empty context* (e_2, p_2) *in* $\mathcal{C}ont(l_2)$, σ $\mathcal{E}$*-unifies* $e_1[l_1]_{p_1}$ *and* $e_2[l_2]_{p_2}$. *Each equation has to be maximal in its clause, and their left-hand side has to be maximal w.r.t. their right-hand side.*

3.1 About the Superposition Strategy

The principle of the superposition strategy is *to apply replacements only into maximal sides of equations*, and has been extensively used for term rewriting and completion. But, the completeness of inference systems representing this strategy has been a longstanding open problem. For completeness, either some deductions using non-maximal literals [24], or some replacements into minimal sides of equations [3], were needed. Bachmair and Ganzinger [2] have proved the completeness in the empty theory of the entire superposition strategy by adding two *Equational Factoring* rules (one for negative and one for positive literals). Defining a particular ordering for comparing negative and positive literals, Nieuwenhuis and Rubio [16] have proved that the rule on negative literals is useless.

In our inference rules, we never need to compare such literals: we always compare literals of the same sign. So for us, *specifying a special ordering on literals is useless*.

3.2 Other Predicate Symbols

Our five inference rules have been defined for deduction in first-order logic with a unique predicate, the equality predicate. This restriction has been decided only for simplifying notations, but it is easy to adapt the inference rules to the presence of other predicate symbols. And we have to add a Factoring rule (applied only on positive clauses) and a Resolution rule (applied with a positive clause) for dealing with the non-equational literals (see [26]). The new system of deduction remains complete if the equality symbol is minimal in precedence.

Now that we have defined all the inference rules, let us show how to compute extended equations with contexts.

4 Extended Equations

An extension of an equation $l \simeq r$ is an equation $e[l]_p \simeq e[r]_p$, also called an *extended equation* of $l \simeq r$, where e is a term, p a non variable position in this term. The subterm at position p in e is $\mathcal{E}$-unifiable with l, the maximum side of the equation. The couple (e, p) is called the *context* of this extension.

The set of all possible contexts for a theory $\mathcal{E}$, written $\mathcal{C}_\mathcal{E}$, is defined by $\bigcup_{k\geq 0} \mathcal{C}ont_k$, where the sets $\mathcal{C}ont_k$ are inductively defined by:

$$\mathcal{C}ont_0 = \{ (e,p) \mid \exists e \simeq e' \text{ or } e' \simeq e \in \mathcal{E},\ p \in \mathcal{FP}os(e) \text{ and } p \neq \epsilon \}$$

$$\mathcal{C}ont_{k+1} = \{ (e_1[e_2]_{p_1}, p_1{\cdot}p_2) \mid (e_1,p_1) \in \mathcal{C}ont_0,\ (e_2,p_2) \in \mathcal{C}ont_k, \text{ and } e_1|_{p_1} \text{ and } e_2 \text{ are } \mathcal{E}\text{-unifiable} \}$$

Then, given an equation $l \simeq r$ where l is maximal w.r.t. r, the set of all possible contexts which can extend $l \simeq r$ is defined by:

$$\mathcal{C}ont(l) \;=\; \{\ (e,p) \in \mathcal{C}_{\mathcal{E}} \;\mid\; e|_p \text{ and } l \text{ are } \mathcal{E}\text{-unifiable }\} \;\cup\; \{\ (l,\epsilon)\ \}$$

We have added (l, ϵ) for avoiding the definition of special inference rules, applying superpositions without context.

Let $l \to r$ be a ground rewrite rule. Let $\mathcal{C}_l$ be the set of all ground instances of contexts of $\mathcal{C}ont(l)$. We define the relation $\longrightarrow_{\mathcal{C}_l,\mathcal{E}}$ by:

$$t_1 \longrightarrow_{\mathcal{C}_l,\mathcal{E}} t_2 \;\text{ if }\; \exists (e_l,p_l) \in \mathcal{C}_l,\ \exists q \in \mathcal{FP}os(t_1),\ t_1|_q =_{\mathcal{E}} e_l,\ t_2 = t_1[e_l[r]_{p_l}]_q$$

This relation $\longrightarrow_{\mathcal{C}_l,\mathcal{E}}$ satisfies a property called $\mathcal{E}$-*closure* if: whenever a term t is reducible into a term t_1 by the relation $\longrightarrow_{\mathcal{C}_l,\mathcal{E}}$, then for each term t_2, $\mathcal{E}$-equal to t, t_2 and t_1 are reducible by $\longrightarrow_{\mathcal{C}_l,\mathcal{E}}$ into two $\mathcal{E}$-equal terms.
A set of contexts $\mathcal{C}$ is said to be $\mathcal{E}$-*covering* if, for any ground term l, the relation $\longrightarrow_{\mathcal{C}_l,\mathcal{E}}$ satisfies the property of $\mathcal{E}$-closure, where $\mathcal{C}_l = \mathcal{C} \cap \mathcal{C}ont(l)$.

Proposition 1. *Let $\mathcal{E}$ be an equational theory. The set of contexts $\mathcal{C}_{\mathcal{E}}$ is $\mathcal{E}$-covering.*

This Proposition means that the role of the equations of $\mathcal{E}$ is entirely simulated by superpositions with contexts of $\mathcal{C}_{\mathcal{E}}$. Its proof is similar to the proof of the Critical Pairs Lemma of Jouannaud and Kirchner [13] (see [29]), and consists of simple case analyses.

However, the definition of $\mathcal{C}_{\mathcal{E}}$ is very general, and for efficiency we combine it with a procedure deleting redundant contexts. Before describing this procedure, let us introduce some definitions.

Definition 7. *A context (e_1, p_1) is* **redundant at a position p** *w.r.t. a set of contexts $\mathcal{C}$, if p is a non-variable position in e_1 and $p_1 = p{\cdot}q$ (where $q \neq \epsilon$), and if there is a context (e_2, p_2) in $\mathcal{C}$ and a substitution σ such that:*

1. *$e_2[\cdot]_{p_2}\sigma =_{\mathcal{E}} (e_1|_p)[\cdot]_q$ for guaranteeing the equivalence of terms $e_2\sigma$ and $e_1|_p$,*
2. *$(e_2|_{p_2})\sigma =_{\mathcal{E}} e_1|_{p_1}$ for guaranteeing the equivalence of subterms where replacements will apply.*

where the symbol $\cdot$ is a new constant. (e_1, p_1) is said to be redundant at p w.r.t. $\mathcal{C}$, by $(e_2\sigma, p_2)$. If p is ϵ, the context (e_1, p_1) is said to be top-redundant.

Definition 8. *Let (e_1, p_1) be a context. Let e'_1 be a ground term and σ a ground substitution such that e'_1 is $\mathcal{E}$-equal to $e_1\sigma$. The representation e'_1 of the context (e_1, p_1) is said to be* **$\mathcal{E}$-redundant at a position p** *w.r.t. a set of contexts $\mathcal{C}$, if there is a term e_2 $\mathcal{E}$-equal to $e'_1|_p$ and a non-variable position p_2 in e_2, s.t. :*

1. *(e_2, p_2) is top-redundant w.r.t. $\mathcal{C}$, by a context (e_3, p_3),*
2. *$(e_1\sigma, p_1)$ is top-redundant by $(e'_1[e_3]_p, p{\cdot}p_3)$.*

Note that the position p may be the empty position.

A context (e_1, p_1) is **$\mathcal{E}$-redundant** w.r.t. a set of contexts $\mathcal{C}$ if,

1. either (e_1, p_1) is top-redundant w.r.t. $\mathcal{C}$,
2. or, for each term e'_1, $\mathcal{E}$-equal to a ground instance $e_1\sigma$ of e_1,
   (a) either there is a non-variable position p'_1 in e'_1 such that (e'_1, p'_1) is top-redundant by (e_1, p_1),
   (b) or the representation e'_1 of the context (e_1, p_1) is $\mathcal{E}$-redundant at a position p' w.r.t. $\mathcal{C}$.

Fig. 1. Redundancy criteria of a context in $\mathcal{E}$.

Proposition 2. *Let $\mathcal{E}$ be an equational theory. The set of contexts $\mathcal{C}_\mathcal{E}$, constructed with the $\mathcal{E}$-redundancy criteria described in Fig. 1, is $\mathcal{E}$-covering.*

Proof. Uselessness of redundant contexts is easily derived from the algorithm described in Fig. 1 as follows:

1. *If there is a context (e_2, p_2) in $\mathcal{C}$ such that any replacement with the context (e_1, p_1) is an instance of a replacement with this context (e_2, p_2), then to use (e_2, p_2) instead of (e_1, p_1) generates the same result, or a more general one.*
2. *Let us study the terms in which the context (e_1, p_1) could be applied. A first remark is there is no need to use this context with terms that are instances of e_1; the replacement can be applied directly at the position p_1. We can generalize this remark: (e_1, p_1) is useless if all terms in which it could be applied can be treated without context or with another context of $\mathcal{C}$. But to test this for each term $\mathcal{E}$-equal to e_1 is not sufficient, because a term $\mathcal{E}$-equal to an instance of e_1 may not be an instance of a term $\mathcal{E}$-equal to e_1. So, the context (e_1, p_1) is $\mathcal{E}$-redundant if, for each term e'_1 $\mathcal{E}$-equal to a ground instance $e_1\sigma$ of e_1,*
   (a) either a term $\mathcal{E}$-equal to $e_1|_{p_1}$ appears at a position p'_1 of e'_1, i.e. the replacement can be directly done at this position; in addition, we have to check that the result is identical to the one obtained with the context (e_1, p_1),
   (b) or a context of $\mathcal{C}$ can be applied at a position p' of e'_1, producing the same result as applying the context (e_1, p_1) to the top of e'_1.

In practice, to check the second point does not consist of studying all the ground instances of e_1, but of enumerating the different forms that can have these instances. And we can note that if the context (e_1, p_1) is redundant at a non-empty position p by a context (e_2, p_2), then all its representations $e'_1 =_\mathcal{E} e_1\sigma$ such that

$$\exists p' \in \mathcal{FP}os(e'_1),\ e'_1|_{p'} =_\mathcal{E} (e_1|_p)\sigma \text{ and } e'_1[\cdot]_{p'} =_\mathcal{E} e_1[\cdot]_p\sigma$$

are $\mathcal{E}$-redundant at the position p' by the context (e_2, p_2). □

A simple algorithm for constructing the contexts with the redundancy criteria of Fig. 1 is, for each context newly created, to verify it is not $\mathcal{E}$-redundant

w.r.t. the set $\mathcal{C}$ of the contexts already constructed; then, we delete from $\mathcal{C}$ the contexts $\mathcal{E}$-redundant by the addition of this new context. Moreover, it would be interesting, when applying an inference rule involving a context, to check the non-redundancy of the instance of this context used, and even to check the non-redundancy of the representation of its term in the clause where the replacement is going to apply.

Let us give two examples of the construction of contexts.

Example 2 ((Associativity and Commutativity)). If $\mathcal{E}$ represents properties of associativity and commutativity of an operator f,

$$\mathcal{E} \;=\; \{\; f(f(x_1,x_2),x_3) \simeq f(x_1,f(x_2,x_3)),\;\; f(x_1,x_2) \simeq f(x_2,x_1) \;\}$$

$\mathcal{C}ont_0$ contains two contexts, $(f(f(x_1,x_2),x_3),1)$ and $(f(x_1,f(x_2,x_3)),2)$, but the second one is top-redundant by the first one. $(f(f(f(x_1,x_2),x_3),x_4),1{\cdot}1)$, the unique context of $\mathcal{C}ont_1$, is top-redundant by $(f(f(x_1,x_2),x_3),1)$ too.
Hence, $\mathcal{C}_{AC} = \{\; (f(f(x_1,x_2),x_3),1) \;\}$, which means that the only possible extension of an equation $l \simeq r$ is $f(l,x_3) \simeq f(r,x_3)$. $\Diamond$

Example 3 ((Associativity)). If $\mathcal{E}$ contains the property of associativity of f,

$$\mathcal{E} \;=\; \{\; f(f(x_1,x_2),x_3) \simeq f(x_1,f(x_2,x_3)) \;\}$$

the non-redundant contexts are

$$\mathcal{C}ont_0 \;=\; \{\; (f(f(x_1,x_2),x_3),1),\;\; (f(x_1,f(x_2,x_3)),2) \;\}$$
$$\mathcal{C}ont_1 \;=\; \{\; (f(f(x_1,f(x_2,x_3)),x_4),1{\cdot}2) \;\}$$

So, there are three useful extensions of an equation $l \simeq r$ where f is the top-symbol of l: to add a new variable, either on the right, $f(l,x_3) \simeq f(r,x_3)$, or on the left, $f(x_1,l) \simeq f(x_1,r)$, or on both sides, $f(f(x_1,l),x_4) \simeq f(f(x_1,r),x_4)$. $\Diamond$

4.1 Refining the Construction of Contexts

In the construction of the sets of contexts $\mathcal{C}ont_k$, we have used the notion of $\mathcal{E}$-unifiability. For instance, for building a context $(e_1[e_2[e_3]_{p_2}]_{p_1}, p_1{\cdot}p_2{\cdot}p_3)$, we have assumed that $e_2|_{p_2}$ and e_3 are $\mathcal{E}$-unifiable, and that $e_1|_{p_1}$ and $e_2[e_3]_{p_2}$ are $\mathcal{E}$-unifiable. But, in this last test, we have lost the information that $e_2|_{p_2}$ and e_3 have to be $\mathcal{E}$-unifiable. There may be no substitution satisfying both conditions, and therefore the context may be useless.

A simple way for avoiding such cases, is to add a third element to each context: *the conjunction of the $\mathcal{E}$-unification constraints* encountered to construct it. In the previous example, the context would be:

$$(e_1[e_2[e_3]_{p_2}]_{p_1},\; p_1{\cdot}p_2{\cdot}p_3,\; \{e_2|_{p_2} =^?_{\mathcal{E}} e_3 \;\wedge\; e_1|_{p_1} =^?_{\mathcal{E}} e_2[e_3]_{p_2}\})$$

Hence, a context is created only if its unification problems admit at least one solution. As a second consequence, for applying an inference rule using a context,

we solve the specific unification problem of this rule, but in conjunction with the unification problems of the context.

With this additional parameter, less contexts are constructed, less inference rules are applicable and their unification problems have less solutions.

However, even with this optimization, there is an infinite number of contexts for a lot of theories, as shown in the next example. This can be dealt with using the algorithm building contexts with incrementality.

Example 4 ((Distributivity)). Let $\mathcal{E} = \{ f(x_1, g(x_2, x_3)) \simeq g(f(x_1, x_2), f(x_1, x_3)) \}$. $\mathcal{C}ont_0$ contains the three contexts

$$(f(x_1, g(x_2, x_3)), 2),\ (g(f(x_1, x_2), f(x_1, x_3)), 1),\ (g(f(x_1, x_2), f(x_1, x_3)), 2)$$

The context $(f(x_1, f(x_2, g(x_3, x_4))), 2{\cdot}2)$ belongs to $\mathcal{C}ont_1$, and so on... We can build an infinite sequence of contexts of the form:

$$(f(x_1, f(x_2, \ldots f(x_n, g(x_{n+1}, x_{n+2})))), 2^n)$$

These contexts are all useful: they permit to recover the subterm $g(x_{n+1}, x_{n+2})$, where the replacement occurs, from the representation:

$$g(f(x_1, f(x_2, \ldots f(x_n, x_{n+1}))), f(x_1, f(x_2, \ldots f(x_n, x_{n+2})))) \qquad \Diamond$$

5 Refutational Completeness

A set of clauses S is said to be *$\mathcal{E}$-incoherent* if there is no model such that $S \cup \mathcal{E}$ is valid in this model. Let us define two properties of a theory $\mathcal{E}$:

- **(P1)** *Regularity.* For any equation $e_1 \simeq e_2$ in $\mathcal{E}$, each variable of e_1 is a variable of e_2, and vice-versa.
- **(P2)** For any ground term s that is $\mathcal{E}$-equal to one of its strict subterms $s|_p$ $(p \neq \epsilon)$, for any ground term t, $s[t]_p$ has to be $\mathcal{E}$-equal to t.

Let *INF* be the set of the five inference rules described in Sect. 3. Let us state the theorem of refutational completeness of *INF*.

Theorem 1 ((Completeness)). *Let $\mathcal{E}$ be an equational theory satisfying at least one of the properties (P1) and (P2), and let S be a set of clauses. If S is $\mathcal{E}$-incoherent, INF will always derive a contradiction from S.*

This theorem states that our inference system is refutationally complete if $\mathcal{E}$ satisfies (P1) or (P2). This result is an important improvement of previous works of Wertz [31] and Paul [17], since they proved the completeness of their systems only if $\mathcal{E}$ satisfies (P2). Moreover, our inference system limits the number of possible deductions much more than Wertz' and Paul's systems, thanks to the positive ordering strategy and the notion of $\mathcal{E}$-redundant contexts.

Note that many theories satisfy (P1) but not (P2). For instance, if $\mathcal{E}$ is $\{f(x, 0) \simeq x\}$, $\mathcal{E}$ is regular but: given the ground term $f(0, 0)$, it is $\mathcal{E}$-equal to 0; however, for a constant a, $f(0, a)$ is not $\mathcal{E}$-equal to a. (P2) is not satisfied.

There are also some particular theories $\mathcal{E}$ that satisfy (P2) but not (P1). For instance[2], if $\mathcal{E}$ is $\{f(x, y) \simeq x\}$, f is the only functional symbol and a is

[2] This example has been suggested to me by Wayne Snyder.

the only constant, the term $f(a,a)$ is $\mathcal{E}$-equal to a, and for any ground term t, $f(a,t) =_{\mathcal{E}} t$. Indeed, since f and a are the only symbols, any ground term (a, $f(a,a)$, $f(f(a,a),a)$,...) is $\mathcal{E}$-equal to a.

We prove the Theorem of Completeness by the transfinite semantic tree method of Hsiang and Rusinowitch [10], extended to deduction modulo an equational theory in [26, 29]. Let us give a sketch of this proof, as it is rather similar to the proof for the particular case of associative and commutative theories [26] (see [29, 30] for the detailed proofs).

Proof. Let $\mathcal{E}$ be a theory satisfying (P1) or (P2). Let S be an $\mathcal{E}$-incoherent set of clauses. Let us describe the main steps of the proof of refutational completeness. It is realized in the ground case, because each deduction step with ground instances clauses can be lifted to the general case.

Given a total $\mathcal{E}$-compatible ordering on ground atoms, we sort them by increasing order, and we construct the transfinite semantic tree $\mathcal{T}$ (in the empty theory). An interpretation *is a node of this tree.*
As S is $\mathcal{E}$-incoherent, each branch of the semantic tree $\mathcal{T}$ has a node that falsifies either a ground instance of a clause of S, or a trivial equation $t \simeq t'$ where $t =_{\mathcal{E}} t'$. Such nodes are called failure nodes. *The maximal subtree of $\mathcal{T}$ which does not contain a failure node is called the* maximal consistent tree, *and written $MCT(S)$.*
Our inference system INF is refutationally complete if it is always able to derive a contradiction (the empty clause) from S. Let $INF^*(S)$ be the set of all clauses deduced by INF from S, in any number of steps. For proving that $INF^*(S)$ contains the empty clause, we show that the maximal consistent tree for $INF^*(S)$, $MCT(INF^*(S))$, is reduced to an empty tree.

The first step is to choose a branch in $MCT(INF^*(S))$ that is consistent with the theory $\mathcal{E}$. This is done by adding new special failure nodes: distant failure nodes *and* quasi-failure nodes.

- *Let K be a failure node at the level of an atom $u \simeq w$ s.t. $u > w$, w is reducible and $u \simeq w$ is falsified by K. If there is an irreducible atom $u \simeq v$, smaller than $u \simeq w$ and s.t. K satisfies $w \simeq v$ (therefore K falsifies $u \simeq v$), the restriction of K to the level of $u \simeq v$ is a* distant failure node.
  This distant failure node permits to avoid a branch where there is a failure node falsifying an equation in which only the smallest side is reducible (condition of the superposition strategy).
- *Let K be an interpretation defined on atoms $A_1, \ldots, A_n$. Let A_{n+1} be an irreducible equation $u_1 \simeq v$ s.t. $u_1 > v$. K has two extensions: L, satisfying $u_1 \simeq v$, and R, falsifying $u_1 \simeq v$. R is a* quasi-failure node *if there is a term u_2, $\mathcal{E}$-equal to u_1, s.t. $u_2 \simeq v$ is reducible by an equation $l \simeq r$ into $u_2[r] \simeq v$, and K satisfies $u_2[r] \simeq v$.*
  This quasi-failure node avoids to have $u_1 \simeq v$ satisfied and $u_2 \simeq v$ falsified in the same branch; this would be inconsistent with $\mathcal{E}$.

In the proof of consistency with $\mathcal{E}$ of this branch, we encounter a major problem; we have to prove that the following case cannot happen in the chosen branch: two $\mathcal{E}$-equal atoms $u_1 \simeq v$ and $u_2 \simeq v$ are interpreted differently, $u_1 \simeq v$ is reducible in u_1 by $l_1 \simeq r_1$, and $u_2 \simeq v$ is reducible in u_2 by $l_2 \simeq r_2$. For the case of associative and commutative theories [26], we show that the branch falsifies a ground instance of a clause of $INF^*(S)$, produced by an extended superposition between $l_1 \simeq r_1$ and $l_2 \simeq r_2$. But, for a general theory $\mathcal{E}$, it is not so easy. The terms $u_1[l_1]$ and l_1 may be $\mathcal{E}$-equal, and in such a situation, we have to prove that $u_1[r_1] \simeq r_1$ is valid in the chosen branch.
Wertz [31] and Paul [17] have decided to only consider theories $\mathcal{E}$ such that, whenever $u_1[l_1]$ and l_1 are $\mathcal{E}$-equal, $u_1[r_1]$ and r_1 are $\mathcal{E}$-equal too (Property (P2)).
In addition, studying the transformation of $u_1[l_1]$ into l_1 by $\mathcal{E}$-equality steps, we prove that $u_1[r_1] \simeq r_1$ is always valid if the theory $\mathcal{E}$ is regular (Property (P1)).

The last step of the proof is to show that the branch is empty. This implies that $MCT(INF^*(S))$ is empty, and also that the empty clause belongs to $INF^*(S)$. A study of the leaves of this branch, i.e. of failure nodes, distant failure nodes and/or quasi-failure nodes cutting it, shows that this branch falsifies a clause of $INF^*(S)$, deduced from clauses falsified by the leaves.
The final solution is that the branch is empty, and therefore the empty clause belongs to $INF^*(S)$.

The compatibility with the positive strategy is a consequence of the following property: if a (distant) failure node along the chosen branch, occuring at the level of an atom A_i, falsifies A_i, then it falsifies a positive clause of $INF^*(S)$. The proof of this property is done by induction on the failure and distant failure nodes, as in [23] for the deduction in the empty theory. □

Our inference system *INF* is compatible with the *simplification strategy*, if the derivations are fair, i.e. do not infinitely forget a possible deduction. This strategy has for purpose the deletion of redundant clauses. Let us give some examples of simplification rules:

- *Simplification* (also called *Demodulation*): it consists of applying a term rewriting step, using a procedure of matching modulo $\mathcal{E}$.
- *Clausal Simplification*: if there is a clause $\rightarrow A$ (resp. $A \rightarrow$), then each clause of the form $A', L \rightarrow R$ (resp. $L \rightarrow A', R$), where A' is $\mathcal{E}$-equal to an instance of A, is replaced by $L \rightarrow R$.
- *Trivial Reflexion*: a clause of the form $l \simeq r, L \rightarrow R$, where l is $\mathcal{E}$-equal to r, is replaced by $L \rightarrow R$.
- *Tautology Deletion*: each clause of the form $L \rightarrow l \simeq r, R$ where $l =_{\mathcal{E}} r$, or $A, L \rightarrow A', R$ where $A =_{\mathcal{E}} A'$, is deleted.

INF is also compatible with the subsumption: if a clause C_1 subsumes a clause C_2 thanks to a substitution σ, i.e. each literal of $C_1\sigma$ is $\mathcal{E}$-equal to a literal of C_2, the clause C_2 is deleted.

6 Implementation

The inference system described in this paper is implemented in the system `DATAC` for the case where $\mathcal{E}$ represents properties of commutativity, or associativity and commutativity (AC), of operators.

`DATAC` is a theorem prover written in CAML Light (18000 lines), a functional language of the ML family; it has a graphical interface written in Tcl/Tk, X11 Toolkit based on the language Tcl. It runs on SUN, HP and IBM PC workstations.
It uses an AC-unification algorithm based on the algorithm of Stickel [27] and the technique for solving Diophantine equations of Fortenbacher [9]. The algorithm for AC-matching is inspired by the algorithm of Hullot [11]. The ordering for comparing terms is the APO of Bachmair and Plaisted [5] with the improvements of Delor and Puel [7].

Let us detail an example of execution in modular lattices, where $\cdot$ denotes the function *meet*, $+$ the function *join*, 1 the greatest element and 0 the least element. The predicate symbol $Comp$ denotes the complementarity of two elements ($Comp$ is commutative). The equational theory $\mathcal{E}$ is the following:

$$\mathcal{E} \;=\; \left\{ \begin{array}{c} (x_1 + x_2) + x_3 \;\simeq\; x_1 + (x_2 + x_3) \\ x_1 + x_2 \;\simeq\; x_2 + x_1 \\ (x_1 \cdot x_2) \cdot x_3 \;\simeq\; x_1 \cdot (x_2 \cdot x_3) \\ x_1 \cdot x_2 \;\simeq\; x_2 \cdot x_1 \end{array} \right\}$$

There are only two useful contexts for this theory $\mathcal{E}$ (cf. Example 2):

$$\mathcal{C}_{\mathcal{E}} \;=\; \{\; ((x_1 + x_2) + x_3, 1),\; ((x_1 \cdot x_2) \cdot x_3, 1) \;\}$$

The initial set of clauses is:

$$\begin{array}{lll}
\textit{(1)} \;\to\; x_1 \cdot x_1 \simeq x_1 & & \textit{(2)} \;\to\; x_1 + x_1 \simeq x_1 \\
\textit{(3)} \;\to\; x_1 \cdot (x_1 + x_2) \simeq x_1 & & \textit{(4)} \;\to\; x_1 + (x_1 \cdot x_2) \simeq x_1 \\
\textit{(5)} \;\to\; x_1 \cdot 0 \simeq 0 & & \textit{(6)} \;\to\; x_1 + 0 \simeq x_1 \\
\textit{(7)} \;\to\; x_1 \cdot 1 \simeq x_1 & & \textit{(8)} \;\to\; x_1 + 1 \simeq 1 \\
\textit{(9)} \; x_1 \cdot x_2 \simeq x_1 \;\to\; x_2 \cdot (x_1 + x_3) \simeq x_1 + (x_3 \cdot x_2) & & \\
\textit{(10)} \; Comp(x_1, x_2) \;\to\; x_1 \cdot x_2 \simeq 0 & & \textit{(11)} \; Comp(x_1, x_2) \;\to\; x_1 + x_2 \simeq 1 \\
\textit{(12)} \; x_1 + x_2 \simeq 1,\; x_1 \cdot x_2 \simeq 0 \;\to\; Comp(x_1, x_2) & &
\end{array}$$

The property we want to prove is:

> *For all elements a and b, let c_1 be the complement of $a \cdot b$ and let c_2 be the complement of $a + b$; then $c_2 + (c_1 \cdot b)$ is the complement of a.*

For this purpose, we add three new clauses that represent the negation of this property (A, B, C_1 and C_2 are new constants):

$$\begin{array}{lll}
\textit{(13)} \;\to\; Comp(C_1, A \cdot B) & & \textit{(14)} \;\to\; Comp(C_2, A + B) \\
\textit{(15)} \; Comp(A, C_2 + (C_1 \cdot B)) \;\to & &
\end{array}$$

The theorem prover DATAC is run with these 15 initial clauses, and with the precedence ordering $\cdot > + > B > A > C_1 > C_2 > 1 > 0$ on functional operators, and $Comp > \simeq$ on predicate operators. Deductions are applied thanks to the inference rules defined in Sect. 3, combined with a resolution rule (for dealing with the predicate $Comp$). These deduction rules combine the positive ordering strategy with the superposition strategy. When a contextual superposition uses an empty context, we simply call it a superposition.
Note that we are going to use a flattened representation under AC operators, i.e. a term $C_1 \cdot (A \cdot B)$ will be written $C_1 \cdot A \cdot B$.
DATAC automatically derives a contradiction, the empty clause written $\square$, in the following way:

Resolution between 10 and 13
(16) $\rightarrow \; A \cdot B \cdot C_1 \simeq 0$

Resolution between 11 and 13
(17) $\rightarrow \; (A \cdot B) + C_1 \simeq 1$

Resolution between 10 and 14
(18) $\rightarrow \; (A + B) \cdot C_2 \simeq 0$

Resolution between 11 and 14
(19) $\rightarrow \; A + B + C_2 \simeq 1$

Left Contextual Superposition from 1 into 9
(32) $x_1 \cdot x_2 \simeq x_1 \cdot x_2 \; \rightarrow \; x_1 \cdot ((x_1 \cdot x_2) + x_3) \simeq (x_1 \cdot x_2) + (x_3 \cdot x_1)$

Trivial Reflexion in 32
(32) $\rightarrow \; x_1 \cdot ((x_1 \cdot x_2) + x_3) \simeq (x_1 \cdot x_2) + (x_3 \cdot x_1)$

Left Contextual Superposition from 3 into 9
(63) $x_1 \cdot x_3 \simeq x_1 \cdot x_3 \; \rightarrow \; (x_1 + x_2) \cdot ((x_1 \cdot x_3) + x_4) \simeq (x_1 \cdot x_3) + (x_4 \cdot (x_1 + x_2))$

Trivial Reflexion in 63
(63) $\rightarrow \; (x_1 + x_2) \cdot ((x_1 \cdot x_3) + x_4) \simeq (x_1 \cdot x_3) + (x_4 \cdot (x_1 + x_2))$

Right Superposition from 17 into 32
(131) $\rightarrow \; (B \cdot A) + (C_1 \cdot B) \simeq B \cdot 1$

Simplification from 7 into 131
(131) $\rightarrow \; (B \cdot A) + (C_1 \cdot B) \simeq B$

Extended Superposition between 3 and 63
(197) $\rightarrow \; ((x_2 \cdot x_3) + (x_4 \cdot (x_2 + x_1))) \cdot x_1 \simeq x_1 \cdot ((x_2 \cdot x_3) + x_4)$

Extended Superposition between 4 and 131
(267) $\rightarrow \; A + (C_1 \cdot B) \simeq B + A$

Right Superposition from 18 into 197
(397) $\rightarrow \; A \cdot ((B \cdot x_1) + C_2) \simeq ((B \cdot x_1) + 0) \cdot A$

Simplification from 6 into 397
(397) $\rightarrow \; A \cdot ((B \cdot x_1) + C_2) \simeq B \cdot x_1 \cdot A$

Left Superposition from 397 into 12
(1214) $(B \cdot x_1) + C_2 + A \simeq 1, \;\; B \cdot x_1 \cdot A \simeq 0 \; \rightarrow \; Comp((B \cdot x_1) + C_2, A)$

`Left Superposition from 16 into 1214`
(2541) $(B \cdot C_1) + C_2 + A \simeq 1,\ 0 \simeq 0 \ \rightarrow\ Comp((B \cdot C_1) + C_2, A)$

`Trivial Reflexion in 2541`
(2541) $(B \cdot C_1) + C_2 + A \simeq 1 \ \rightarrow\ Comp((B \cdot C_1) + C_2, A)$

`Clausal Simplification in 2541 thanks to 15`
(2541) $(B \cdot C_1) + C_2 + A \simeq 1 \ \rightarrow$

`Simplification from 267 into 2541`
(2541) $B + A + C_2 \simeq 1 \ \rightarrow$

`Clausal Simplification in 2541 thanks to 19`
(2541) $\square$

The following table compares our positive ordering strategy with the classical ordering strategy [10], which requires only that deductions have to apply between maximal literals of clauses. For this comparison, we applied two linear completion steps on the 12 initial clauses of the previous example. A step of linear completion consists of applying all possible deductions between the initial clauses, but none with one of the deduced clauses. The second step for the ordering strategy was stopped because of a lack of memory while solving a tricky AC-unification problem.
The last column of this table presents statistics for the example traced above. For this example, we have used a simplified version of the AC-unification algorithm that permits not to compute all the minimal solutions and not to solve tricky problems. A consequence is the loss of the completeness of the strategy, but the main advantage is that we avoid problems of memory size.

Linear Completion	First step		Second step		Example
	Ordering	Positive	Ordering	Positive	
Initial Clauses	12	12	53	19	15
Generated Clauses	111	63	>3336	240	2526
Final Clauses	53	19	>1508	46	258
Resolutions	0	0	0	0	4
Superpositions	14	12	>554	59	783
Cont. Superpositions	20	14	>74	24	125
Ext. Superpositions	9	6	>67	12	748
Deductions	43	32	>695	95	1660
Simplifications	132	109	>4410	413	5086
Deletions	151	133	$\gg$1881	324	3407

These statistics give an idea of the advantage of the positive strategy, but the proportions cannot be generalized. Indeed, the positive strategy may be less powerful if some initial clauses have several negative literals. In addition, if the positive strategy reduces the width of the search space, it increases the depth of the proofs (depth 5 for previous example, while depth 4 with the ordering strategy).

7 Conclusion

In this paper, we have defined an inference system for automated deduction modulo equational theories. This system combines the superposition strategy with a positive ordering strategy to prune the search space. Moreover, we have described a procedure for computing contexts, from the theory $\mathcal{E}$ only, i.e. without the use of the initial set of clauses.
Our system has been proved refutationally complete for a large class of equational theories, including all the regular theories. This and our algorithm for constructing non-redundant contexts are important improvements of previous results of Wertz [31] and Paul [17]. One of our further works is to implement this algorithm and to study theories where there is an infinity of non-redundant contexts.

Our technique of deduction modulo some equations has shown its interest in our theorem prover `DATAC`, for the case of associative and commutative theories. However, for testing it on other theories, we need to study orderings for comparing terms and unification algorithms, since there are very few in the literature. This lack of orderings may be solved by term rewriting techniques as in [7, 22]. Unification algorithms may be solved by term rewriting techniques too, for dealing with parts of these theories such as in [15].
However, it seems that one of the most interesting ways for dealing with these problems of $\mathcal{E}$-unification is to use symbolic constraints, as in [28].

Acknowlegments: I would like to thank Prof. Anita Wasilewska of Stony Brook for the numerous discussions we had on the history of the bases of this paper.

I would like to dedicate this paper to the memory of my colleague Valentin Antimirov of INRIA Lorraine (France), with whom I had very interesting discussions while preparing a first version of this paper.

References

1. L. Bachmair and N. Dershowitz. Completion for Rewriting Modulo a Congruence. *Theoretical Computer Science*, 67(2-3):173–202, October 1989.
2. L. Bachmair and H. Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer-Verlag, July 1990.
3. L. Bachmair and H. Ganzinger. Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):1–31, 1994.
4. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.
5. L. Bachmair and D. Plaisted. Associative Path Orderings. In *Proceedings 1st Conference on Rewriting Techniques and Applications, Dijon (France)*, volume 202 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

6. D. Brand. Proving Theorems with the Modification Method. *SIAM Journal of Computing*, 4:412–430, 1975.
7. C. Delor and L. Puel. Extension of the Associative Path Ordering to a Chain of Associative Symbols. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1993.
8. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland), 1990.
9. A. Fortenbacher. *Effizientes Rechnen in AC-Gleichungstheorien*. PhD thesis, Universität Karlsruhe (Germany), February 1989.
10. J. Hsiang and M. Rusinowitch. Proving Refutational Completeness of Theorem Proving Strategies: The Transfinite Semantic Tree Method. *Journal of the ACM*, 38(3):559–587, July 1991.
11. J.-M. Hullot. *Compilation de Formes Canoniques dans les Théories équationelles*. Thèse de Doctorat de Troisième Cycle, Université de Paris Sud, Orsay (France), 1980.
12. J.-P. Jouannaud and C. Kirchner. Solving Equations in Abstract Algebras: a Rule-based Survey of Unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge (MA, USA), 1991.
13. J.-P. Jouannaud and H. Kirchner. Completion of a Set of Rules Modulo a Set of Equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City (USA), 1984.
14. D. S. Lankford and A. Ballantyne. Decision Procedures for Simple Equational Theories with Associative Commutative Axioms: Complete Sets of Associative Commutative Reductions. Technical report, Univ. of Texas at Austin, Dept. of Mathematics and Computer Science, 1977.
15. C. Marché. *Réécriture modulo une théorie présentée par un système convergent et décidabilité du problème du mot dans certaines classes de théories équationnelles*. Thèse de Doctorat d'Université, Université de Paris-Sud, Orsay (France), October 1993.
16. R. Nieuwenhuis and A. Rubio. Basic Superposition is Complete. In B. Krieg-Brückner, editor, *Proceedings of ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Springer-Verlag, 1992.
17. E. Paul (E-mail: etienne.paul@issy.cnet.fr). E-Semantic Tree. Unpublished paper, 70 pages, 1994.
18. G. E. Peterson. A Technique for Establishing Completeness Results in Theorem Proving with Equality. *SIAM Journal of Computing*, 12(1):82–100, 1983.
19. G. E. Peterson and M. E. Stickel. Complete Sets of Reductions for Some Equational Theories. *Journal of the ACM*, 28:233–264, 1981.
20. G. Plotkin. Building-in Equational Theories. *Machine Intelligence*, 7:73–90, 1972.
21. J. A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12:23–41, 1965.
22. A. Rubio and R. Nieuwenhuis. A Precedence-Based Total AC-Compatible Ordering. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 374–388. Springer-Verlag, 1993.
23. M. Rusinowitch. *Démonstration automatique — Techniques de réécriture*. InterEditions, 1989.

24. M. Rusinowitch. Theorem-proving with Resolution and Superposition. *Journal of Symbolic Computation*, 11:21–49, 1991.
25. M. Rusinowitch and L. Vigneron. Associative Commutative Deduction. In E. Domenjoud and Claude Kirchner, editors, *Proceedings of the 1st CCL Workshop, Le Val d'Ajol (France)*, October 1992.
26. M. Rusinowitch and L. Vigneron. Automated Deduction with Associative-Commutative Operators. *Applicable Algebra in Engineering, Communication and Computing*, 6(1):23–56, January 1995.
27. M. E. Stickel. A Unification Algorithm for Associative-Commutative Functions. *Journal of the ACM*, 28:423–434, 1981.
28. L. Vigneron. Associative-Commutative Deduction with Constraints. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 530–544. Springer-Verlag, June 1994.
29. L. Vigneron. *Automated Deduction with Symbolic Constraints in Equational Theories*. PhD Thesis, Université Henri Poincaré - Nancy 1, November 1994. Available as Research Report *CRIN 94-T-266* (in French).
30. L. Vigneron. Theorem Proving modulo Regular Theories. Technical report 95-1, Department of Computer Science, SUNY at Stony Brook, Stony Brook, January 1995.
31. U. Wertz. First-Order Theorem Proving Modulo Equations. Technical Report MPI-I-92-216, Max Planck Institut für Informatik, April 1992.

Article de référence du Chapitre 2

L. Bachmair, A. Tiwari and L. Vigneron.
Abstract Congruence Closure.
Journal of Automated Reasoning, 31(2) : 129-168,
January 2003.
Kluwer Academic Publishers.


Journal of Automated Reasoning **31:** 129–168, 2003.

Abstract Congruence Closure ⋆

LEO BACHMAIR,[1] ASHISH TIWARI,[2] and LAURENT VIGNERON[3]
[1]*Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, U.S.A. e-mail: leo@cs.sunysb.edu*
[2]*SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, U.S.A. e-mail: tiwari@csl.sri.com*
[3]*LORIA – Université Nancy 2, Campus Scientifique, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France. e-mail: vigneron@loria.fr*

(Received: 18 April 2001)

Abstract. We describe the concept of an *abstract congruence closure* and provide equational inference rules for its construction. The length of any maximal derivation using these inference rules for constructing an abstract congruence closure is at most quadratic in the input size. The framework is used to describe the logical aspects of some well-known algorithms for congruence closure. It is also used to obtain an efficient implementation of congruence closure. We present experimental results that illustrate the relative differences in performance of the different algorithms. The notion is extended to handle associative and commutative function symbols, thus providing the concept of an *associative-commutative congruence closure*. Congruence closure (modulo associativity and commutativity) can be used to construct ground convergent rewrite systems corresponding to a set of ground equations (containing AC symbols).

Key words: term rewriting, congruence closure, associative-commutative theories.

1. Introduction

Term-rewriting systems provide a simple and very general mechanism for computing with equations. The Knuth–Bendix completion method and its extensions to equational term-rewriting systems can be used on a variety of problems. However, completion-based methods usually yield semi-decision procedures; and in the few cases where they provide decision procedures, the time complexity is considerably worse than that of certain other efficient algorithms for solving the same problem. On the other hand, the specialized decision algorithms for particular problems are not very useful when considered for integration with general-purpose theorem-proving systems. Moreover, the logical aspects inherent in the problem and the algorithm seem to get lost in descriptions of specific algorithms.

We are interested in developing *efficient* procedures for a large class of decidable problems using standard and general techniques from theorem proving so as to bridge the gap alluded to above. We first consider equational theories induced by systems of ground equations. Efficient algorithms for computing congruence clo-

⋆ The research described in this paper was supported in part by the National Science Foundation under grant CCR-9902031. Some of the results described in this paper also appeared in [5, 4].

sure can be used to decide whether a ground equation is an equational consequence of a set of ground equations. All algorithms for congruence closure computation rely on the use of certain data structures, in the process obscuring any inherent logical aspects.

In general, a system of ground equations can be completed into a convergent ground term-rewriting system by using a total termination ordering. However, this process can in the worst case take exponential time unless the rules are processed using a certain strategy [25]. Even under the specific strategy, the resulting completion procedure is quadratic, and the $O(n \log(n))$ efficiency of congruence closure algorithms is not attained. There are also known techniques [29] to construct ground convergent systems that use graph-based congruence closure algorithms.

We attempt to capture the essence of some of the efficient congruence closure algorithms using standard techniques from term rewriting. We do so by introducing symbols and extending the signature to abstractly represent sharing that is inherent in the use of term-directed acyclic graph data structures. We thus define a notion of abstract congruence closure and provide transition rules that can be used to construct such abstract congruence closures. A whole class of congruence closure algorithms can be obtained by choosing suitable strategies (and implementations) for the abstract transition rules. The complexity of any such congruence closure algorithm is directly related to the length of derivation (using these transition rules) required to compute an abstract congruence closure with the chosen strategy. We give bounds on the length of arbitrary maximal derivations, and we show its relationship with the choice of ordering used for completion.

We describe some of the specific well-known congruence closure algorithms in the framework of abstract congruence closure, and we show that the abstract framework suitably captures the sources of efficiency in some of these algorithms. The description separates the logical aspects inherent in these algorithms from implementation details.

The concept of an abstract congruence closure is useful in more than one way. Many other algorithms, like those for syntactic unification and rigid E-unification, that rely either on congruence closure computation or on the use of term directed acyclic graph (dag) representation for efficiency also admit simpler and more abstract descriptions using an abstract congruence closure [6, 5].

Furthermore, if certain function symbols in the signature are assumed to be associative and commutative, we can introduce standard techniques from rewriting modulo an equational theory to handle it. Thus, we obtain a notion of congruence closure modulo associativity and commutativity. As an additional application, we consider the problem of constructing ground convergent systems (in the original signature) for a set of ground equations. We show how to eliminate the new constants introduced earlier to transform all equations back to the original signature while preserving some of the nice properties of the system over the extended signature, thus generalizing the results in [29].

PRELIMINARIES

Let Σ be a set, called a *signature*, with an associated *arity function* $\alpha\colon \Sigma \to 2^{\mathbb{N}}$, and let $\mathcal{V}$ be a disjoint (denumerable) set. We define $\mathcal{T}(\Sigma, \mathcal{V})$ as the smallest set containing $\mathcal{V}$ and such that $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ whenever $f \in \Sigma, n \in \alpha(f)$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$. The elements of the sets Σ, $\mathcal{V}$, and $\mathcal{T}(\Sigma, \mathcal{V})$ are respectively called *function symbols*, *variables*, and *terms* (over Σ and $\mathcal{V}$). Elements c in Σ for which $\alpha(c) = \{0\}$ are called *constants*. By $\mathcal{T}(\Sigma)$ we denote the set $\mathcal{T}(\Sigma, \emptyset)$ of all variable-free, or *ground*, terms. The symbols $s, t, u, \ldots$ are used to denote terms; $f, g, \ldots$, function symbols; and $x, y, z, \ldots$, variables. We write $t[s]$ to indicate that a term t contains s as a subterm and (ambiguously) denote by $t[u]$ the result of replacing a particular occurrence of s by u.

An *equation* is a pair of terms, written $s \approx t$. The *replacement relation* $\to_{E^g}$ induced by a set of equations E is defined by $u \to_{E^g} v$ if, and only if, $u = u[l]$ contains l as a subterm and $v = u[r]$ is obtained by replacing l by r in u, where $l \approx r$ is in E. The *rewrite relation* $\to_E$ induced by a set of equations E is defined by $u \to_E v$ if, and only if, $u = u[l\sigma]$, $v = u[r\sigma]$, $l \approx r$ is in E, and σ is some substitution.

If $\to$ is a binary relation, then $\leftarrow$ denotes its inverse, $\leftrightarrow$ its symmetric closure, $\to^+$ its transitive closure, and $\to^*$ its reflexive-transitive closure. Thus, $\leftrightarrow^*_{E^g}$ denotes the *congruence relation*⋆ induced by E. We shall mostly be interested in sets E of ground equations whence the distinction between rewrite relation and replacement relation disappears. The *equational theory* of E is defined as the relation $\leftrightarrow^*_E$. Equations are often called *rewrite rules*, and a set E a *rewrite system*, if one is interested particularly in the rewrite relation $\to^*_E$ rather than the equational theory $\leftrightarrow^*_E$.

A term t is *irreducible*, or in *normal form*, with respect to a rewrite system R if there is no term u such that $t \to_R u$. We write $s \to^!_R t$ to indicate that t is an R-normal form of s.

A rewrite system R is said to be (ground) *confluent* if for every pair s, s' of (ground) terms, if there exists a (ground) term t such that $s \leftarrow^*_R t \to^*_R s'$, then there exists a (ground) term t' such that $s \to^*_R t' \leftarrow^*_R s'$. Thus, if R is (ground) confluent, then every (ground) term t has at most one normal form. A rewrite system R is *terminating* if there exists no infinite reduction sequence $s_0 \to_R s_1 \to_R s_2 \cdots$ of terms. Clearly, if R is terminating, then every term t has at least one normal form. Rewrite systems that are (ground) confluent and terminating are called (ground) *convergent*.

A rewrite system R is *left reduced* if every left-hand side term (of any rule in R) is irreducible by all other rules in R. A rewrite system R is *right reduced* if every right-hand side term (of any rule in R) is in R-normal form. A rewrite system that is both left reduced and right reduced is said to be *fully reduced*.

⋆ A congruence relation is a reflexive, symmetric, and transitive relation on terms that is also a replacement relation.

2. Abstract Congruence Closure

We first describe the form of terms and equations that will be used in the description of an abstract congruence closure. Definitions that introduce similar concepts also appear in [16–18, 27].

DEFINITION 1. Let Σ be a signature and K be a set of constants disjoint from Σ. A *D-rule* (with respect to Σ and K) is a rewrite rule of the form

$$f(c_1, \ldots, c_k) \to c,$$

where $f \in \Sigma$ is a k-ary function symbol and $c_1, \ldots, c_k, c$ are constants in set K.

A *C-rule* (with respect to K) is a rule $c \to d$, where c and d are constants in K.

For example, if $\Sigma_0 = \{a, b, f\}$ and $E_0 = \{a \approx b, fffa \approx fb\}$,* then

$$D_0 = \{a \to c_0, b \to c_1, fc_0 \to c_2, fc_2 \to c_3, fc_1 \to c_4\}$$

is a set of D-rules over Σ_0 and $K_0 = \{c_0, c_1, c_2, c_3, c_4\}$. Using these D-rules, we can simplify the original equations in E_0. For example, the term $fffa$ can be rewritten to c_3 as $fffa \to_{D_0} ffc_0 \to_{D_0} fc_2 \to_{D_0} c_3$. Original equations in E_0 can thus be simplified by using D_0 to give $C_0 = \{c_0 \approx c_1, c_3 \approx c_4\}$. The set $D_0 \cup C_0$ may be viewed as an alternative representation of E_0 over an extended signature. The equational theory presented by $D_0 \cup C_0$ is a conservative extension of the theory E_0. This reformulation of the equations E_0 in terms of an extended signature is (implicitly) present in all congruence closure algorithms; see Section 3.

The constants in the set K can be thought of as names for equivalence classes of terms. A D-rule $f(c_1, \ldots, c_k) \to c_0$ indicates that a term with top function symbol f and arguments belonging to the equivalence classes $c_1, \ldots, c_k$ itself belongs to the equivalence class c_0. In this sense, a set of D-rules can be thought of as defining a bottom-up tree automaton [10]. Other interpretations for the constants in K are possible too, especially in the context of term directed acyclic graph representation; see Section 3 for details.

A constant c in K is said to *represent* a term t in $\mathcal{T}(\Sigma \cup K)$ (via the rewrite system R) if $t \leftrightarrow_R^* c$. A term t is *represented* by R if it is represented by some constant in K via R. For example, the constant c_3 represents the term $fffa$ via D_0.

DEFINITION 2 (Abstract congruence closure). Let Σ be a signature and K be a set of constants disjoint from Σ. A ground rewrite system $R = D \cup C$ of D-rules and C-rules (with respect to Σ and K) is said to be an (*abstract*) *congruence closure* if

(i) each constant $c \in K$ represents some term $t \in \mathcal{T}(\Sigma)$ via R, and
(ii) R is ground convergent.

* When writing a term, we remove parentheses wherever possible for clarity.

If E is a set of ground equations over $\mathcal{T}(\Sigma \cup K)$ and in addition R is such that

(iii) for all terms s and t in $\mathcal{T}(\Sigma)$, $s \leftrightarrow_E^* t$ if, and only if, $s \rightarrow_R^* \circ \leftarrow_R^* t$,

then R will be called an (abstract) congruence closure *for* E.

Condition (i) essentially states that K contains no superfluous constants; condition (ii) ensures that equivalent terms have the same representative (which usually also implies that congruence of terms can be tested efficiently); and condition (iii) implies that R is a conservative extension of the equational theory induced by E over $\mathcal{T}(\Sigma)$.

The rewrite system $R_0 = D_0 \cup \{c_0 \rightarrow c_1, c_3 \rightarrow c_4\}$ above is not a congruence closure for E_0, as it is not ground convergent. But we can transform R_0 into a suitable rewrite system, using a completion-like process described in more detail below, to obtain a congruence closure

$$\begin{aligned} R_1 \;=\; \{&a \rightarrow c_1, b \rightarrow c_1, fc_1 \rightarrow c_4, fc_4 \rightarrow c_4,\\ &c_0 \rightarrow c_1, c_2 \rightarrow c_4, c_3 \rightarrow c_4\}. \end{aligned}$$

2.1. CONSTRUCTION OF ABSTRACT CONGRUENCE CLOSURES

We next present a general method for construction of an abstract congruence closure. Our description is fairly abstract, in terms of transition rules that manipulate triples (K, E, R), where K is the set of constants that extend the original fixed signature Σ, E is the set of ground equations (over $\Sigma \cup K$) yet to be processed, and R is the set of C-rules and D-rules that have been derived so far. Triples represent *states* in the process of constructing a congruence closure. Construction starts from an *initial state* $(\emptyset, E, \emptyset)$, where E is a given set of ground equations.

The transition rules can be derived from those for standard completion as described in [3], with some differences so that (i) application of the transition rules is guaranteed to terminate and (ii) a convergent system is constructed over an *extended* signature. The transition rules do *not* require a total reduction ordering* on terms in $\mathcal{T}(\Sigma)$, but simply an ordering on $\mathcal{T}(\Sigma \cup U)$ (that is, terms in $\mathcal{T}(\Sigma)$ need not be comparable in this ordering), where U is an infinite set disjoint from Σ from which new constants $K \subset U$ are chosen. In particular, we assume $\succ_U$ is any ordering on the set U and define $\succ$ as follows: $c \succ d$ if $c \succ_U d$ and $t \succ c$ if $t \rightarrow c$ is a D-rule. For simplicity, we take U to be the set $\{c_0, c_1, c_2, \ldots\}$ and assume that $c_i \succ_U c_j$ if, and only if, $i < j$.

A key transition rule introduces new constants as names for subterms.

Extension: $$\frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \rightarrow c\})}$$

* By an *ordering* we mean any irreflexive and transitive relation on terms. A *reduction ordering* is an ordering that is also a well-founded replacement relation. An ordering $\succ$ is *total* if for any two distinct elements s and t, either $s \succ t$ or $t \succ s$.

where $t \to c$ is a D-rule, t is a term occurring in (some equation in) E, and $c \in U - K$.

The following three rules are versions of the corresponding rules for standard completion specialized to the ground case.

$$\textbf{Sim}\text{plification:} \quad \frac{(K, E[t], R \cup \{t \to c\})}{(K, E[c], R \cup \{t \to c\})}$$

where t occurs in some equation in E. (It is fairly easy to see that by repeated application of extension and simplification, any equation in E can be reduced to an equation that can be oriented by the ordering $\succ$.)

$$\textbf{Ori}\text{entation:} \quad \frac{(K \cup \{c\}, E \cup \{t \approx c\}, R)}{(K \cup \{c\}, E, R \cup \{t \to c\})}$$

if $t \succ c$.

Trivial equations may be deleted.

$$\textbf{Del}\text{etion:} \quad \frac{(K, E \cup \{t \approx t\}, R)}{(K, E, R)}$$

In the case of completion of ground equations, deduction steps can all be replaced by suitable simplification steps, in particular by collapse. To guarantee termination, however, we formulate collapse by two different specialized transition rules. The usual side condition in the collapse rule, which refers to the *encompassment ordering*, can be considerably simplified in our case.

$$\textbf{Ded}\text{uction:} \quad \frac{(K, E, R \cup \{t \to c, t \to d\})}{(K, E \cup \{c \approx d\}, R \cup \{t \to d\})}$$

$$\textbf{Col}\text{lapse:} \quad \frac{(K, E, R \cup \{s[c] \to c', c \to d\})}{(K, E, R \cup \{s[d] \to c', c \to d\})}$$

if c is a proper subterm of s.

As in standard completion the simplification of right-hand sides of rules in R by other rules is optional and not necessary for correctness. Right-hand sides of rules in R are always constants.

$$\textbf{Com}\text{position:} \quad \frac{(K, E, R \cup \{t \to c, c \to d\})}{(K, E, R \cup \{t \to d, c \to d\})}$$

Various known congruence closure algorithms can be abstractly described by using different strategies over the above rules. All the above transition rules with the exception of the composition rule constitute the *mandatory* set of transition rules.

EXAMPLE 1. Consider the set of equations $E_0 = \{a \approx b, fffa \approx fb\}$. An abstract congruence closure for E_0 can be derived from the initial state $(K_0, E_0, R_0) = (\emptyset, E_0, \emptyset)$ as follows:

i	Constants K_i	Equations E_i	Rules R_i	Transition
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_0\}$	$\{c_0 \approx b, fffa \approx fb\}$	$\{a \to c_0\}$	**Ext**
2	$\{c_0\}$	$\{fffa \approx fb\}$	$\{a \to c_0, b \to c_0\}$	**Ori**
3	$\{c_0\}$	$\{ffc_0 \approx fc_0\}$	$\{a \to c_0, b \to c_0\}$	**Sim** (twice)
4	$\{c_0, c_1\}$	$\{fc_1 \approx fc_0\}$	$R_3 \cup \{fc_0 \to c_1\}$	**Ext**
5	$\{c_0, c_1\}$	$\{fc_1 \approx c_1\}$	$R_3 \cup \{fc_0 \to c_1\}$	**Sim**
6	K_5	$\{\}$	$R_5 \cup \{fc_1 \to c_1\}$	**Ori**

The rewrite system R_6 is an abstract congruence closure for E_0.

2.2. CORRECTNESS

We use the symbol $\vdash$ to denote the one-step transformation relation on states induced by the above transformation rules. A *derivation* is a sequence of states $(K_0, E_0, R_0) \vdash (K_1, E_1, R_1) \vdash \cdots$.

THEOREM 1 (Soundness). *If $(K, E, R) \vdash (K', E', R')$, then, for all terms s and t in $\mathcal{T}(\Sigma \cup K)$, we have $s \leftrightarrow^*_{E' \cup R'} t$ if, and only if, $s \leftrightarrow^*_{E \cup R} t$.*

Proof. For simplification, orientation, deletion, and composition, the claim follows from correctness result for the standard completion transition rules [3]. The claim is also easily verified for the specialized collapse and deduction rules.

Now, suppose $(K', E', R' = R \cup \{u \to c\})$ is obtained from (K, E, R) by using extension. For $s, t \in \mathcal{T}(\Sigma \cup K)$, if $s \leftrightarrow^*_{E \cup R} t$, then clearly $s \leftrightarrow^*_{E' \cup R'} t$. Conversely, if $s \leftrightarrow^*_{E' \cup R'} t$, then $s\sigma \leftrightarrow^*_{E'\sigma \cup R'\sigma} t\sigma$, where σ is (homomorphic extension of) the mapping $c \mapsto u$. But $s\sigma = s$ and $t\sigma = t$ as $c \notin K$. Furthermore, $E'\sigma = E$, and $R'\sigma = R \cup \{u \to u\}$. Therefore, $s = s\sigma \leftrightarrow^*_{E \cup R} t\sigma = t$. □

LEMMA 1. *Let K_0 be a finite set of constants (disjoint from Σ), E_0 a finite set of equations (over $\Sigma \cup K$), and R_0 a finite set of D-rules and C-rules such that for every C-rule $c \to d$ in R_0 we have $c \succ_U d$. Then each derivation starting from the state (K_0, E_0, R_0) is finite. Furthermore, if $(K_0, E_0, R_0) \vdash^* (K_m, E_m, R_m)$, then the rewrite system R_m is terminating.*

Proof. We first define the measure of a state (K, E, R) to be the number of occurrences of symbols from Σ in E. Two states are compared by comparing their measures using the usual "greater-than" ordering on natural numbers. It can be easily verified that each transformation rule either reduces this measure or leaves it unchanged. Specifically, extension always reduces this measure.

Now, consider a derivation starting from the state (K_0, E_0, R_0). Any such derivation can be written as

$$(K_0, E_0, R_0) \vdash^* (K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots,$$

where the derivation $(K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots$ contains no applications of extension, and hence the set $K_n = K_{n+1} = \cdots$ is finite. Therefore, the ordering $\succ_{K_n}$ (defined as the restriction of the ordering $\succ_U$ on K_n) is well founded.

Next we prove that the derivation $(K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots$ is finite. Assign a weight $w(c)$ to each symbol c in K_n so that $w(c) > w(d)$ if, and only if, $c \succ_{K_n} d$; and set $w(f) = \max\{w(c) : c \in K_n\} + 1$, for each $f \in \Sigma$. Let $\gg$ be the Knuth–Bendix ordering using these weights. Define a secondary measure of a state (K, E, R) as the set $\{\{\{s, t\}\} : s \approx t \in E\} \cup \{\{\{s\}, \{t\}\} : s \to t \in R\}$. Two states are compared by comparing their measures using a twofold multiset extension* of the ordering $\gg$ on terms. It is straightforward to see that application of any transition rule (except extension) to a state reduces the secondary measure of the state. Moreover, every rule in R_j is reducing in the reduction ordering $\gg$, and hence each rewrite system R_j is terminating. □

The following lemma says that extension introduces no superfluous constants.

LEMMA 2. *Suppose that* $(K, E, R) \vdash (K', E', R')$ *and that for every* $c \in K$, *there exists a term* $s \in \mathcal{T}(\Sigma)$ *such that* $c \leftrightarrow^*_{E \cup R} s$. *Then, for every* $d \in K'$, *there exists a term* $t \in \mathcal{T}(\Sigma)$ *such that* $d \leftrightarrow^*_{E' \cup R'} t$.

Proof. If $d \in K'$ also belongs to the set K, then the claim is easily proved by using Theorem 1. Otherwise let $d \in K' - K$. The only nontrivial case is when (K', E', R') is obtained by using extension.

Let $f(c_1, \ldots, c_k) \to d$ be the rule introduced by extension. Since $c_1, \ldots, c_k \in K$, there exist terms $s_1, \ldots, s_k \in \mathcal{T}(\Sigma)$ such that $s_i \leftrightarrow^*_{E \cup R} c_i$; and hence, from Theorem 1, $s_i \leftrightarrow^*_{E' \cup R'} c_i$. The term $f(s_1, \ldots, s_k)$ is the required term t. □

We call a state (K, E, R) *final* if no mandatory transition rule is applicable to this state. It follows from Lemma 1 that final states can be finitely derived. The third component of a final state is always an abstract congruence closure.

THEOREM 2. *Let* Σ *be a signature and* K_1 *a finite set of constants disjoint from* Σ. *Let* E_1 *be a finite set of equations over* $\Sigma \cup K_1$ *and* R_1 *be a finite set of D-rules and C-rules such that every* $c \in K_1$ *represents some term* $t \in \mathcal{T}(\Sigma)$ *via* $E_1 \cup R_1$, *and* $c \succ_U d$ *for every C-rule* $c \to d$ *in* R_1. *If* (K_n, E_n, R_n) *is a final state such that* $(K_1, E_1, R_1) \vdash^* (K_n, E_n, R_n)$, *then* $E_n = \emptyset$, *and* R_n *is an abstract congruence closure for* $E_1 \cup R_1$ (*over* Σ *and* K_n).

* A *multiset* over a set S is a mapping M from S to the natural numbers. Any ordering $\succ$ on a set S can be extended to an ordering $\succ^m$ on multisets over S as follows: $M \succ^m N$ iff $M \neq N$ and whenever $N(x) > M(x)$, then $M(y) > N(y)$, for some $y \succ x$. The multiset ordering $\succ^m$ (on finite multisets) is well founded if the ordering $\succ$ is well founded [13].

Proof. Since the sets K_1, E_1, and R_1 are finite and the state (K_n, E_n, R_n) is obtained from (K_1, E_1, R_1) by using a finite derivation, it follows that K_n, E_n, and R_n are all finite sets. If $E_n \neq \emptyset$, then either extension or orientation will be applicable. Since (K_n, E_n, R_n) is a final state, $E_n = \emptyset$.

To show that R_n is an abstract congruence closure for $E_1 \cup R_1$, we need to prove the three conditions in Definition 2.

(1) Lemma 2 implies that every $c \in K_n$ represents some term $t \in \mathcal{T}(\Sigma)$ via R_n.
(2) Using Lemma 1, we know that R_n is terminating. Furthermore, since (K_n, E_n, R_n) is a final state, R_n is left reduced. By the critical pair lemma [1], therefore, R_n is confluent and hence convergent.
(3) Theorem 1 establishes that if $s \leftrightarrow^*_{E_1 \cup R_1} t$ for some $s, t \in \mathcal{T}(\Sigma)$, then $s \leftrightarrow^*_{E_n \cup R_n} t$. Since $E_n = \emptyset$ and R_n is convergent, $s \rightarrow^*_{R_n} \circ \leftarrow^*_{R_n} t$. □

2.3. PROPERTIES

To summarize, we have presented an abstract notion of congruence closure and given a method to construct such an abstract congruence closure for a given set of ground equations. The only parameters required by the procedure are a denumerable set U of constants (disjoint from Σ) and an ordering (irreflexive and transitive relation) on this set. It might appear that the abstract congruence closure one obtains depends on the ordering $\succ_U$ used. In this section, we first show that we can construct an abstract congruence closure that is independent of the ordering on constants.

In the process of construction of an abstract congruence closure, we may deduce an equality between two constants in K, and we require an ordering $\succ_U$ to deal with such equations. Since constants are essentially "names" for equivalence classes, it is redundant to have two different names for the same equivalence class. Hence, one such constant and the corresponding ordering dependence can be eliminated.

DEFINITION 3. Any constant $c \in K$ that occurs as a left-hand side of a C-rule in R is called *redundant* in R.

Redundant constants in R can be eliminated after composition and collapse steps with C-rules in R have been applied exhaustively.

$$\textbf{Comp}\text{ression:} \quad \frac{(K \cup \{c, d\}, E, R \cup \{c \rightarrow d\})}{(K \cup \{d\}, E\langle c \mapsto d\rangle, R\langle c \mapsto d\rangle)}$$

if c occurs only once as a left-hand side term, the notation $\langle c \mapsto d\rangle$ denotes the homomorphic extension of the mapping σ defined as $\sigma(c) = d$ and $\sigma(x) = x$ for $x \neq c$, and $E\langle c \mapsto d\rangle$ denotes the set of equations obtained by applying the mapping $\langle c \mapsto d\rangle$ to each term in the set E.

Correctness of the new enhanced set of transition rules for construction of congruence closure can be established in the same way as before.

THEOREM 3. *Let Σ be a signature and E be a finite set of equations over Σ. Then, there exists an abstract congruence closure D for E (over Σ and some K) consisting only of D-rules.*

Proof. Let $(\emptyset, E, \emptyset) \vdash^* (K_n, E_n, R_n)$ such that none of the mandatory transition rules nor compression is applicable to the state (K_n, E_n, R_n).

We observe that the following version of soundness (Theorem 1) is still true: If $(K_i, E_i, R_i) \vdash (K_j, E_j, R_j)$, then, for all terms s and t in $\mathcal{T}(\Sigma \cup (K_i \cap K_j))$, $s \leftrightarrow^*_{E_j \cup R_j} t$ iff $s \leftrightarrow^*_{E_i \cup R_i} t$. Additionally, Lemma 1 and Lemma 2 continue to hold with the new set of transition rules, and the proofs remain essentially unchanged. Thus, we can use Theorem 2 in this new setting to conclude that R_n is an abstract congruence closure. Since compression is not applicable to the final state, there can be no C-rules in R_n. □

Graph-based congruence closure algorithms can be described by using D-rules; see Section 3. However, we can define a *generalized D-rule* (with respect to Σ and K) as any rule of the form $t \to c$ where $c \in K$ and $t \in \mathcal{T}(\Sigma, K) - K$, as done in [5]. The transition rules for construction of congruence closure can be suitably generalized with minimal changes. The new definition of D-rules allows for preserving as much of the original term structure as possible.

Choosing an Ordering $\succ_U$ on the Fly. As remarked earlier, the set of transition rules presented in Section 2.1⋆ for construction of abstract congruence closure is parameterized by a denumerable set U of constants and an ordering $\succ_U$ on this set. Since elements of U serve only as names, we can choose U to be any countable set of symbols. An ordering $\succ_U$ need not be specified a priori but can be defined on the fly as the derivation proceeds. We need to maintain irreflexivity whenever the ordering relation is extended. Observe that we need an ordering only when there is a C-equation to orient.

If we exhaustively apply simplification before trying to orient a C-equation, *any* orientation of the fully simplified C-equation can be used. Given a derivation $(K_0, E_0, D_0 \cup C_0) \vdash \cdots \vdash (K_i, E_i, D_i \cup C_i)$ using this strategy, we construct a sequence of relations $\succ_0, \succ_1, \ldots$, where each $\succ_j$ is defined by $c \succ_j d$ if $c \to d \in \bigcup_{k \le j} C_k$. We claim that each $\succ_j$ defines an ordering. To see this, note that $\succ_0$ defines a trivial ordering (in which no two elements in U are comparable). Moreover, whenever the relation $\succ_j$ is extended by $c \succ d$, the constants c and d are incomparable in the transitive closure of the existing relation $\succ_j$, and hence irreflexivity of the ordering defined by $\succ_{j+1}$ is established.

Bounding the Maximal Derivation Length. The above observation establishes that there exist derivations for congruence closure construction in which we do not spend any time in comparing elements. However, we shall shortly show that the length of derivations crucially depends on the chosen ordering. This reveals

⋆ We exclude compression for rest of the discussion.

a tradeoff between the effort spent in choosing an ordering and the lengths of derivations obtained when using that ordering.

DEFINITION 4. An ordering $\succ$ on the set U is *feasible* for a state (K, E, R) if there exists an unfailing* maximal derivation starting from the state (K, E, R) that uses the ordering $\succ$.

The *depth* or *height* of an ordering $\succ$ is the length of the longest chain. More specifically, if the longest chain for ordering $\succ$ is $c_0 \succ c_1 \succ \cdots \succ c_\delta$, then the depth of $\succ$ is δ.

Congruence closure computation using specialized data structures is known to be more efficient than naive standard completion. We next show, by proving a bound on the length of *any* maximal derivation, that our description captures the cause of this efficiency.

LEMMA 3. *Any maximal derivation starting from the state* $(K_0 = \emptyset, E_0, R_0 = \emptyset)$ *is of length* $O((2k + l)\delta + n)$*, where* k *is the number of applications of extension,* l *is the difference between the number of occurrences of* 0*-arity symbols in* E_0 *and number of distinct* 0*-arity symbols in* E_0*,* δ *is the depth of ordering* $\succ_U$ *used to construct the derivation, and* n *is the number of* Σ*-symbols in* E_0.

Proof. To simplify the argument, we first split simplification and deduction rules as follows (ignoring the K-component):

$$\textbf{Sim1}: \quad \frac{(E[f(\ldots)], R \cup \{f(\ldots) \to c\})}{(E[c], R \cup \{f(\ldots) \to c\})} \qquad \textbf{Sim2}: \quad \frac{(E[c], R \cup \{c \to d\})}{(E[d], R \cup \{c \to d\})}$$

$$\textbf{Ded1}: \quad \frac{(E, R \cup \{f(\ldots) \to c, f(\ldots) \to d\})}{(E \cup \{c \approx d\}, R \cup \{f(\ldots) \to d\})}$$

$$\textbf{Ded2}: \quad \frac{(E, R \cup \{c \to d, c \to d'\})}{(E \cup \{d \approx d'\}, R \cup \{c \to d\})}$$

Next, we bound the number of applications of individual rules in *any* derivation as follows:

(i) A derivation step using sim2, ded2, collapse, or composition corresponds to *rewriting* some constant. Since the length of a rewriting sequence $c_1 \to c_2 \to \cdots$ is bounded by δ and $2k + l$ is an upper bound on the number of occurrences of constants (from K_∞) in $E_i \cup R_i$ (for any i), the number of applications of sim2, ded2, collapse, and composition is $O((2k + l)\delta)$.

(ii) The number of deletion steps is at most $|E_0| + k$ because each transition rule, with the exception of extension and deletion, preserves the cardinality of $E_i \cup R_i$ and extension increases this number by one while deletion decreases it by one.

* By *unfailing* we mean that the set of unoriented equations in the final state is empty.

(iii) The number of sim1 and ded1 steps is at most n because each such step reduces the number of Σ-symbols (in $E \cup R$).
(iv) The number of Extension steps is k.
(v) Application of Orientation at most doubles the length of any derivation.

Thus, the total length of any derivation is $O((2k + l)\delta + n)$. □

The number k of extension steps used in any maximal derivation is $O(n)$ because the total number of Σ-symbols in the second component of the state is non-increasing in any derivation and an application of extension reduces this number by one.

LEMMA 4. *A starting state $(K_0 = \emptyset, E_0, R_0 = \emptyset)$ can be transformed into a state (K_m, E_m, R_m) in $O(n)$ derivation steps, where n is the total number of symbols in the finite set E_0 of ground equations such that*

(i) *the set E_m consists of only C-equations and R_m consists of only D-rules, and*
(ii) *the total number of symbols in $E_m \cup R_m$ is $O(n)$.*

Proof. We construct the desired derivation by an exhaustive application of extension and simplification rules. Clearly, the set E_m contains only C-equations and R_m contains only D-rules. The length of this derivation is $O(n)$ because every application of extension and simplification reduces the total number of Σ-symbols in E_i by at least one. Moreover, the total number of symbols in $E_m \cup R_m$ is $O(n)$ because every application of extension and simplification increases the total number of symbols by a constant. □

Informally speaking, therefore, since l is clearly $O(n)$, Lemma 3 gives us an upper bound of $O(n\delta)$ on the length of maximal derivations. Any total (linear) order on the set K_∞ of constants is feasible but has depth equal to the cardinality of K_∞, which is $O(n)$. This gives a quadratic bound on the length of a derivation. However, we can also show that there exist feasible orderings with smaller depth.

LEMMA 5. *Let (K_m, E_m, R_m) be a state such that E_m consists of only C-equations and R_m consists of only D-rules. Then, there exists a feasible ordering $\succ_U$ for this state with depth $O(\log(n))$, where n is the number of constants in K_m.*

Proof. We shall exhibit an unfailing derivation that constructs the required ordering on the fly as discussed before; that is, during the derivation, we ensure that whenever we apply orientation as $(K_i, E_i \cup \{c \approx d\}, D_i \cup C_i) \vdash (K_i, E_i, D_i \cup C_i \cup \{c \to d\})$, the constants c and d are in C_i-normal form. Additionally, we also impose the requirement that the cardinality of the set $\{c' \in K_m : c' \leftrightarrow^*_{C_i} c\}$ is less than or equal to the cardinality of $\{c' \in K_m : c' \leftrightarrow^*_{C_i} d\}$.

As argued before, the relation thus built defines an ordering. Suppose $(K_\infty, E_\infty, D_\infty \cup C_\infty)$ is the final state of this unfailing derivation. If $c_1 \succ c_2 \succ \cdots \succ c_j$ is a maximal descending chain, then the cardinality of the set $\{c' \in K_m : c' \leftrightarrow^*_{C_\infty} c_j\}$ is at least 2^{j-1}. But, since the cardinality of K_m is $O(n)$, therefore, $j = O(\log(n))$. □

Combining these three lemmas leads to the following result.

THEOREM 4. *There exists a maximal derivation of length* $O(n \log(n))$ *with starting state* $(\emptyset, E_0, \emptyset)$*, where n is the total number of symbols in the finite set* E_0 *of ground equations.*

Proof. We construct the derivation in two stages. In the first stage we use the derivation constructed in the proof of Lemma 4 to obtain an intermediate state (K_m, E_m, R_m) from the starting state $(K_0 = \emptyset, E_0, R_0 = \emptyset)$. In the second stage, we start with this intermediate state and carry out the derivation in the proof of Lemma 5 to reach a final state. The claim then follows from Lemma 4 and Lemma 3. □

Theorem 4 establishes the possibility of obtaining short maximal derivations by using (simple strategies on) the abstract transition rules. However, to get an efficient, say $O(n \log(n))$, algorithm for computing a congruence closure, we need to show that the ordering on constants can be efficiently computed and that each individual step in the derivation can be applied in (amortized) constant time. The first of these is easily achieved by extending the state triple (K, E, R) by an additional component that is a function, `counter`, that maps each constant in K to a natural number. More precisely, `counter`(c) stores the cardinality of the set

$$[c]_C \stackrel{\text{def}}{=} \{c' \in K : c' \leftrightarrow_C^* c\},$$

where C is the set of C-equations in R. Thus, `counter`(c) is the number of constants in the current equivalence class of c (see proof of Lemma 5). The function `counter` can easily be updated when a C-equation, say $c \approx d$, is oriented into, say, $c \to d$, by setting `counter`(d) = `counter`(c) + `counter`(d).

Second, efficient application of each transition step requires specialized data structures and/or efficient indexing mechanisms. Some such details have been described in the literature, and we discuss these in the next section.

We observe here that in the special case when each congruence class modulo E_0 is finite, feasible orderings with constant depth (in fact, depth 1) can be constructed efficiently on the fly. During orientation, only those C-equations are oriented that contain constants whose congruence class $[c]_{C_i}$ (w.r.t. the set C_i of C-equations in the present state) is known to *not* change in subsequent states. For example, if c is one such constant and $[c]_{C_i} = \{c, c_1, \ldots, c_k\}$, then we orient so that we add rules $\{c_i \to c : i = 1, \ldots, k\}$ to the third component. That such C-equations always exist and can be efficiently identified is a simple consequence of the finiteness assumption; see [30, 15] for details. Thus, we obtain a linear bound on the length of (certain) maximal derivations for construction of congruence closure in this special case.

3. Congruence Closure Strategies

The literature abounds with various implementations of congruence closure algorithms. The general framework of abstract congruence closure can be used to

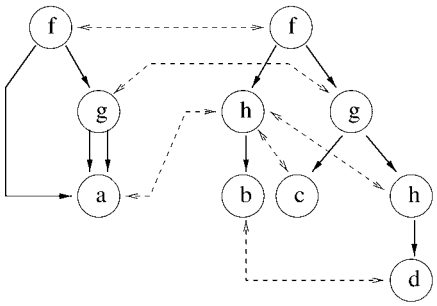
uniformly describe the logical characteristics of such algorithms and provides a context for interpreting differences in their performance. We next describe the algorithms proposed by Downey, Sethi, and Tarjan [15], Nelson and Oppen [23], and Shostak [28] in this way. That is, we provide a description of these algorithms (the description does not capture certain implementation details) using abstract congruence closure transition rules.

Directed acyclic graphs are a common data structure used to implement algorithms that work with terms. In fact, many congruence closure algorithms assume that the input is an equivalence relation on vertices of a given dag, and the desired output, the congruence closure of this equivalence, is again represented by an equivalence on the same dag.

A set of C-rules and D-rules may be interpreted as an abstraction of a dag representation. The constants in K (or U) represent nodes in a dag. The D-rules specify edges, and the C-rules represent a binary relation on the nodes. More precisely, a D-rule $f(c_1, \ldots, c_k) \to c$ specifies that the node c is labeled by the symbol f and has pointers to the nodes $c_1, \ldots, c_k$. Conversely, any dag and an associated binary relation on its nodes can be represented by using D-rules and C-rules. Figure 1 illustrates the representation of a set of terms (and a binary relation on them) using dags and using D-rules and C-rules. The solid lines represent *subterm* edges, and the dashed lines represent a binary relation on the vertices. We have a D-rule corresponding to each vertex, and a C-rule for each dashed edge. (We note here that generalized D-rules (with respect to Σ and K) as defined in Section 2.3 correspond to storing *contexts*, rather than just symbols from Σ, in each node of the term dag. We do not pursue this optimization in this paper.)

Traditional congruence closure algorithms employ data structures that are suitably abstracted in our presentation as follows:

(i) To obtain a representation via D-rules and C-equations for the *input dag* corresponding to equation set E_0, we start from the state $(\emptyset, E_0, \emptyset)$ and repeatedly



D-rules representing the term dag:

$$\begin{array}{rcl@{\qquad}rcl} a & \to & c_1 & gc_1c_1 & \to & c_2 \\ b & \to & c_4 & fc_1c_2 & \to & c_3 \\ c & \to & c_6 & gc_6c_8 & \to & c_9 \\ d & \to & c_7 & hc_7 & \to & c_8 \\ hc_4 & \to & c_5 & fc_5c_9 & \to & c_{10} \end{array}$$

C-rules representing the relation on vertices:

$$\begin{array}{lll} c_1 \approx c_5 & c_2 \approx c_9 & c_3 \approx c_{10} \\ c_4 \approx c_7 & c_6 \approx c_5 & c_5 \approx c_8 \end{array}$$

Figure 1. A term dag and a relation on its vertices.

apply a single extension step followed by an exhaustive application of simplification (represented using the expression $(\mathbf{Ext} \cdot \mathbf{Sim}^*)^*$). In the resulting state (K_1, E_1, D_1), the set D_1 represents the input dag, and the set E_1 contains only C-equations representing the input equivalence on nodes of this dag. Note that because of eager simplification, we obtain a representation of a dag with maximum possible sharing. For example, if $E_0 = \{a \approx b, fffa \approx fb\}$, then $K_1 = \{c_0, c_1, c_2, c_3, c_4\}$, $E_1 = \{c_0 \approx c_1, c_3 \approx c_4\}$, and $R_1 = \{a \to c_0, b \to c_1, fc_0 \to c_2, fc_2 \to c_3, fc_1 \to c_4\}$.

(ii) The signature of a term $f(t_1, \ldots, t_k)$ is defined as $f(c_1, \ldots, c_k)$, where c_i is the name of the equivalence class containing term t_i. A *signature table* (indexed by vertices of the input dag) stores a signature for some or all vertices. A signature table specifies a set of fully left reduced D-rules.

(iii) The *use table* (also called predecessor list) is a mapping from the constant c to the set of all nodes whose signature contains c. In our presentation this translates to a method of indexing the set of D-rules.

(iv) A *union-find* data structure is used to maintain equivalence classes on the set of nodes of the input dag. In the abstract representation, C-rules describe equivalence relations on constants in K. Operations on the union-find structure exhibit as transitions on C-rules. For instance, application of composition specifies path-compression on the union-find structure.

We note that D-rules serve a twofold purpose: they represent both a term dag and a signature table.

3.1. SHOSTAK'S ALGORITHM

We show that Shostak's congruence closure procedure is a specific strategy over the general transition rules for abstract congruence closure.

Shostak's congruence closure is *dynamic* in that equations are processed one at a time. The strategy underlying Shostak's procedure can be described by the following regular expression:

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^?)^* \cdot (\mathbf{Del} \cup \mathbf{Ori}) \cdot (\mathbf{Col} \cdot \mathbf{Ded}^*)^*)^*$$

This expression should be interpreted as follows. Given a (start) state (K, E, R), (i) Pick an equation $s \approx t$ from the set E. (ii) Reduce the terms s and t to constants, say c and d, respectively, by repeatedly applying simplification and extension, always eagerly applying simplification before any possible extension. (iii) If c and d are identical, then apply deletion (and continue with (i)) and, if not, create a C-rule, say $c \to d$, using orientation. (iv) Replace c by d using collapse, and follow it by exhaustive application of deduction. Repeat this until there are no more possible collapse steps. Finally, apply steps (i) through (iv) repeatedly. Shostak's procedure halts if no unoriented equations remain.

Shostak's procedure uses indexing based on the idea of the *use*() list. This *use*()-based indexing helps in identifying all possible collapse applications.

It is fairly easy to observe that a maximal derivation starting from state $(\emptyset, E_0, \emptyset)$ and using the above strategy ends in a *final* state. Hence, Theorem 2 establishes that the third component of Shostak's halting state is convergent and an abstract congruence closure (for E_0).

EXAMPLE 2. We use the set E_0 from Example 1 to illustrate Shostak's method, showing the essential intermediate steps in the derivation.

i	Cnsts K_i	Equations E_i	Rules R_i	Transition
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_0, c_1\}$	$\{fffa \approx fb\}$	$\{a \to c_0, b \to c_1, c_0 \to c_1\}$	**Ext** · **Ext**·
			$c_0 \to c_1\}$	**Ori**
2	$\{c_0, c_1\}$	$\{ffc_1 \approx fb\}$	$\{a \to c_0, b \to c_1, c_0 \to c_1\}$	**Sim** · **Sim**
3	$\{c_0, \ldots, c_3\}$	$\{c_3 \approx fb\}$	$R_2 \cup \{fc_1 \to c_2, fc_2 \to c_3\}$	**Ext** · **Ext**
4	$\{c_0, \ldots, c_3\}$	$\{c_3 \approx c_2\}$	R_3	**Sim** · **Sim**
5	$\{c_0, \ldots, c_3\}$	$\emptyset$	$R_4 \cup \{c_3 \to c_2\}$	**Ori**

3.2. DOWNEY, SETHI, AND TARJAN'S ALGORITHM

The Downey–Sethi–Tarjan algorithm assumes that the input is a dag and an equivalence relation on its vertices. Thus, the starting state is a triple given by $(K_1, \emptyset, D_1 \cup C_1)$, where D_1 represents the input dag and C_1 the given equivalence. The underlying strategy of this algorithm can be described as

$$((\mathbf{Col} \cdot (\mathbf{Ded} \cup \{\epsilon\}))^* \cdot (\mathbf{Sim}^* \cdot (\mathbf{Del} \cup \mathbf{Ori}))^*)^*,$$

where ϵ is the null transition rule. This strategy is implemented by repeating the following steps: (i) Repeatedly apply the collapse rule and any resulting deduction steps until no more collapse steps are possible. (ii) If no collapse steps are possible, repeatedly select a C-equation, fully simplify it, and then either delete or orient it.

In the Downey, Sethi, and Tarjan procedure an equation $c \approx d$ is oriented to $c \to d$ if the equivalence class c contains fewer terms (in the set of all subterms in the input set of equations) than does the equivalence class d. This point is crucial in ensuring the $O(n \log(n))$ time complexity for this algorithm; cf. Theorem 4.

If $(K_n, E_n, D_n \cup C_n)$ is the last state in a derivation from $(K_1, \emptyset, D_1 \cup C_1)$ using the above strategy, then $(K_n, E_n, D_n \cup C_n)$ is a final state, and hence the set $D_n \cup C_n$ is convergent and an abstract congruence closure. The rewrite system D_n represents the information contained in the signature table, and C_n represents information in the union-find structure. The set C_n is usually considered the output of the Downey, Sethi, and Tarjan procedure.

EXAMPLE 3. We illustrate the Downey–Sethi–Tarjan algorithm by using the same set of equations E_0 as above. The start state is $(K_1, \emptyset, D_1 \cup C_1)$, where

$K = \{c_0, \ldots, c_4\}$, $D_1 = \{a \to c_0, b \to c_1, fc_0 \to c_2, fc_2 \to c_3, fc_1 \to c_4\}$, and, $C_1 = \{c_0 \to c_1, c_3 \to c_4\}$.

i	Consts K_i	Eqns E_i	Rules R_i	Transition
1	K_1	$\emptyset$	$D_1 \cup C_1$	
2	K_1	$\emptyset$	$\{a \to c_0, b \to c_1, fc_1 \to c_2,$ $fc_2 \to c_3, fc_1 \to c_4\} \cup C_1$	**Col**
3	K_1	$\{c_2 \approx c_4\}$	$R_2 - \{fc_1 \to c_2\}$	**Ded**
4	K_1	$\emptyset$	$R_3 \cup \{c_4 \to c_2\}$	**Ori**

Note that $c_4 \approx c_2$ was oriented in a way that no further collapses were needed thereafter.

3.3. NELSON AND OPPEN'S ALGORITHM

The Nelson–Oppen procedure is not exactly a completion procedure, and it does *not* generate a congruence closure in our sense. The initial state of the Nelson–Oppen procedure is given by the tuple (K_1, E_1, D_1), where D_1 is the input dag, and E_1 represents an equivalence on vertices of this dag. The sets K_1 and D_1 remain unchanged in the Nelson–Oppen procedure. In particular, the inference rule used for deduction is different from the conventional deduction rule:*

$$\textbf{NODed}\text{uction:} \quad \frac{(K, E, D \cup C)}{(K, E \cup \{c \approx d\}, D \cup C)}$$

if there exist two D-rules $f(c_1, \ldots, c_k) \to c$, and, $f(d_1, \ldots, d_k) \to d$ in the set D; and, $c_i \to_C^! \circ \leftarrow_C^! d_i$, for $i = 1, \ldots, k$.

The Nelson–Oppen procedure can now be (at a certain abstract level) represented as

$$\textbf{NO} = (\textbf{Sim}^* \cdot (\textbf{Ori} \cup \textbf{Del}) \cdot \textbf{NODed}^*)^*,$$

which is applied in the following sense: (i) select a C-equation $c \approx d$ from the E-component; (ii) simplify the terms c and d using simplification steps until the terms can't be simplified any more; (iii) either delete or orient the simplified C-equation; (iv) apply the NODeduction rule until there are no more nonredundant applications of this rule; and (v) if the E-component is empty, then stop; otherwise continue with step (i).

Assume that, using the Nelson–Oppen strategy, we get a derivation $(K_1, E_1, D_1) \vdash_{\textbf{NO}}^* (K_n, E_n, D_n \cup C_n)$. One consequence of using a nonstandard deduction rule, NODeduction, is that the resulting set $D_n \cup C_n = D_1 \cup C_n$ need not necessarily be convergent, although the rewrite relation D_n/C_n [12] is convergent.

* This rule performs deduction modulo C-equations; that is, we compute critical pairs between D-rules modulo the congruence induced by C-equations. Hence, the Nelson–Oppen procedure can be described as an *extended completion* [12] (or completion modulo C-equations) method over an extended signature.

EXAMPLE 4. Using the same set E_0 as equations, we illustrate the Nelson–Oppen procedure. The initial state is given by (K_1, E_1, D_1), where $K_1 = \{c_0, c_1, c_2, c_3, c_4\}$; $E_1 = \{c_0 \approx c_1, c_3 \approx c_4\}$; and, $D_1 = \{a \to c_0, b \to c_1, fc_0 \to c_2, fc_2 \to c_3, fc_1 \to c_4\}$.

i	Constants K_i	Equations E_i	Rules R_i	Transition
1	K_1	E_1	D_1	
2	K_1	$\{c_3 \approx c_4\}$	$D_1 \cup \{c_0 \to c_1\}$	**Ori**
3	K_1	$\{c_2 \approx c_4, c_3 \approx c_4\}$	R_2	**NODed**
4	K_1	$\{c_3 \approx c_4\}$	$R_2 \cup \{c_2 \to c_4\}$	**Ori**
5	K_1	$\emptyset$	$R_4 \cup \{c_3 \to c_4\}$	**Ori**

Consider deciding the equality $fa \approx ffb$. Even though $fa \leftrightarrow^*_{E_0} ffb$, the terms fa and ffb have distinct normal forms with respect to R_5. But terms in the original term universe have identical normal forms.

4. Experimental Results

We have implemented several congruence closure algorithms, including those proposed by Nelson and Oppen (NO) [23], Downey, Sethi, and Tarjan (DST) [15], and Shostak (SHO) [28], and two algorithms based on completion – one with an indexing mechanism (IND) and the other without (COM). Implementation of the first three procedures is based on the representation of terms by directed acyclic graphs and the representation of equivalence classes by a union-find data structure. Union-find data structure uses path compression, and the same code (with only minor variations) is used in all three implementations.

NO is an implementation of the pseudocode given on page 358 (with some details on page 359) of [23]. In particular, the predecessor lists are kept sorted and duplicates are removed whenever two predecessor lists are merged. Furthermore, the double loop described in step 4 of the algorithm is implemented as an optimized linear search (with a "sorting" overhead) as suggested in [23]. We tested other minor variants, too. The variant in which splicing the predecessor list was done in constant time (allowing for duplicates in the process), and step 4 was implemented as a nested loop gave the best running times on our examples, which we report here.

The DST implementation corresponds exactly to the pseudocode on page 761 of [15]. In particular, the *signature table* is implemented as a hash table, equivalence classes are represented in union-find, and the sets *pending* and *combine* are implemented as singly linked lists of pointers to graph nodes and to graph edges, respectively.

Implementation SHO of Shostak's algorithm is based on the specialization to the pure theory of equality of the combination method described on page 8 of [28]. The main data structures in the implementation are the union-find, *use* lists, and

sig, which stores a signature for each vertex. The manipulation of these data structures, especially the *use* lists, and the sequence of calls to *merge* are exactly as described in [28]. This algorithm (with only a slight difference in the order of calls to subroutine merge) is also described in [11, 18].

The completion procedure COM uses the following strategy:

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^*)^* \cdot (\mathbf{Del} \cup \mathbf{Ori}) \cdot ((\mathbf{Com}^* \cdot \mathbf{Col}^*) \cdot \mathbf{Ded} \cdot (\mathbf{Del} \cup \mathbf{Ori}))^*)^*.$$

More specifically, we process one equation at a time, fully simplify it, and if necessary use extension to generate a C-equation. The C-equation is oriented, and composition and collapse are applied exhaustively, followed by a deduction step. The generated C-equation is similarly handled. When no more C-equations can be produced, we process the next equation. In short, this strategy is based on eager elimination of redundant constants.

The indexed variant IND uses a slightly different strategy:

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^*)^* \cdot ((\mathbf{Del} \cup \mathbf{Ori}) \cdot (\mathbf{Col}^* \cdot \mathbf{Com}^? \cdot \mathbf{Ded}^?)^* \cdot \mathbf{Sim}^*)^*)^*.$$

As before, using $\mathbf{Sim}^* \cdot \mathbf{Ext}^*$ we convert one equation to a C-equation. This equation is oriented; and, individually on every D-rule, we perform all simplifications using this C-rule, namely, collapse and composition, followed by any deduction step $(\mathbf{Col}^* \cdot \mathbf{Com}^? \cdot \mathbf{Ded}^?)$. Subsequently, simplification of equations using the oriented C-rule is done. All the C-equations are processed this way before we take up the next equation to process. Indexing refers to the use of suitable data structures to efficiently identify which D-rules contain specified constants, thus making the process of identifying collapse, composition, and superposition efficient.

In all our implementations, input is read from a file containing equations in a specified syntax. It is parsed and represented internally as a list of tree node pairs (representing terms with no sharing). There is a preprocessing step in the NO and DST algorithms to convert this representation into a dag and to initialize the other required data structures. In DST we construct a dag in which all vertices have outdegree at most two. The other three algorithms interleave construction of a dag with deduction steps. The published descriptions of DST and NO do not address construction of a dag. Our implementation maintains in a hash table the list of terms that have been represented in the dag and creates a new node for each term not yet represented.

The input set of equations E can be classified based on (i) the size of the input and the number of equations, (ii) the number of equivalence classes on terms and subterms of E, and (iii) the average number of occurrences of a constant in the set of D- and C-rules, which roughly corresponds to average size of *use* lists in most of the implementations. The first set of examples is relatively simple and developed by hand to highlight strengths and weaknesses of the various algorithms. Example 11 contains five equations that induce a single equivalence class.* Example 12 is the

* The equation set is $\{f^2(a) \approx a,\ f^{10}a \approx f^{15}b,\ b \approx f^5 b,\ a \approx f^3 a,\ f^5 b \approx b\}$.

Table I. Total running time (in milliseconds) for Examples 11–14. *Eqns* refers to the number of equations, *Vert* to the number of vertices in the initial dag, and *Class* to the number of equivalence classes induced on the dag.

	Eqns	Vert	Class	DST	NO	SHO	COM	IND
Ex.11	5	27	1	1.286	1.640	0.281	0.606	0.409
Ex.12	20	27	1	2.912	2.772	0.794	1.858	0.901
Ex.13	12	20	6	1.255	0.733	0.515	0.325	0.323
Ex.14	34	105	2	10.556	22.488	7.275	12.077	4.416

same as 11 except that it contains five copies of all the equations. Example 13 requires slightly larger *use* lists.** Example 14 consists of equations that are oriented in the "wrong" way.‡

In a first set of experiments, we assume that the input is a set of equations presented as pairs of trees (representing terms). Thus, the total running time given includes time spent on preprocessing and construction of the dag (for NO and DST). In Table I the times shown are the averages of several runs on a Sun Ultra workstation under similar load conditions. The time was computed by using the *gettimeofday* system call.

Table II contains similar comparisons for considerably larger examples consisting of randomly generated equations over a specified signature. The equations are obtained by fixing a signature and a bound on the depth of terms and randomly picking $2n$ terms from the set of all bounded depth terms in the given signature. We generate n equations by pairing the $2n$ terms thus obtained. The choice of signatures and depth bound was governed by the need to randomly generate interesting instances (i.e., where there are a fair number of deductions). The columns Σ_i denote the number of function symbols of arity i in the signature and d denotes the maximum term depth. The total running time includes the preprocessing time.‡‡

In Table III we show the time for computing a congruence closure assuming terms are already represented by a dag. In other words, we do not include the time it takes to create a dag. Note that we include no comparison with Shostak's method, as the dynamic construction of a dag from given term equations is inherent in this procedure. However, a comparison with a suitable strategy (in which all extension steps are applied before any deduction steps) of IND is possible. We denote by IND* indexed completion based on a strategy that first constructs a dag. The examples are the same as in Table II.

Several observations can be drawn from these results. First, the Nelson–Oppen procedure NO is competitive only when deduction steps are few and the number of

** The equation set is $\{g(a,a,b) \approx f(a,b), gabb \approx fba, gaab \approx gbaa, gbab \approx gabb, gbba \approx gbab, gaaa \approx faa, a \approx c, c \approx d, d \approx e, b \approx c1, c1 \approx d1, d1 \approx e1\}$.

‡ The set is $\{g(f^i(a), h^{10}(b)) \approx g(a,b), i = \{1,\ldots,25\}, h^{47}(b) \approx b, b \approx h^{29}(b), h(b) \approx c0, c0 \approx c1, c1 \approx c2, c2 \approx c3, c3 \approx c4, c4 \approx a, a \approx f(a)\}$.

‡‡ Times for COM are not included because indexing is indispensable for larger examples.

Table II. Total running time (in seconds) for randomly generated sets of equations.

	Eqns	Vert	$\Sigma_0, \Sigma_1, \Sigma_2$, d	Class	DST	NO	SHO	IND
Ex.21	10000	17604	2, 0, 2, 3	7472	11.1	3.19	10.2	13.0
Ex.22	5000	4163	2, 1, 1, 3	3	2.28	306	3.09	0.77
Ex.23	5000	7869	3, 0, 1, 3	2745	2.44	1.36	3.52	3.99
Ex.24	6000	8885	3, 0, 1, 3	9	3.55	1152	52.4	7.07
Ex.25	7000	9818	3, 0, 1, 3	1	4.63	1682	47.8	5.47
Ex.26	5000	645	4, 2, 0, 23	77	1.22	1.58	0.37	0.36
Ex.27	5000	1438	10, 2, 0, 23	290	1.45	3.67	0.39	0.37

Table III. Running time (in seconds) when input is in a dag form.

	DST	NO	IND*		DST	NO	IND*
Ex.21	0.919	0.296	0.076	Ex.25	0.958	1614.961	9.770
Ex.22	0.309	319.112	1.971	Ex.26	0.026	0.781	0.060
Ex.23	0.241	0.166	0.030	Ex.27	0.048	2.470	0.176
Ex.24	0.776	1117.239	7.301				

equivalence classes is large. In logical terms, this is because it uses a nonstandard deduction rule (see [5]), which may force the procedure to unnecessarily repeat the same deduction steps many times over a single execution. Not surprising, straightforward completion without indexing is also inefficient when many deduction steps are necessary. Indexing is of course a standard technique employed in all practical implementations of completion.

The running time of the DST procedure critically depends on the size of the hash table that contains the signatures of all vertices. If the hash table size is large, enough potential deductions can be detected in (almost) constant time. If the hash table size is reduced, to say 100, then the running time increases by a factor of up to 50. A hash table with 1,000 entries was sufficient for our examples (which contained fewer than 10,000 vertices). Larger tables did not improve the running times substantially.

Indexed Completion, DST, and Shostak's method are roughly comparable in performance, though Shostak's algorithm has some drawbacks. For instance, equations are always oriented from left to right. In contrast, Indexed Completion always orients equations in a way so as to minimize the number of applications of the collapse rule, an idea that is also implicit in Downey, Sethi, and Tarjan's algorithm. Example 12 illustrates this fact. More crucial, the manipulation of the *use* lists in Shostak's method is done in a convoluted manner, and hence redundant inferences may be made when searching for the correct nonredundant ones. As a consequence,

Shostak's algorithm performs poorly on instances where *use* lists are large and deduction steps are many, such as in Examples 13, 24 and 25.

We note that the indexing technique used in our implementation of completion is simple – with every constant c we associate a list of D-rules that contain c as a subterm. On the other hand, DST maintains at least two different ways of indexing the signatures, and hence it is more efficient when the examples are large and deduction steps numerous. On small examples, the overhead to maintain the data structures dominates. This also suggests that the use of more sophisticated indexing schemes for indexed completion might improve performance.

5. Associative-Commutative Congruence Closure

We next consider the problem of constructing a congruence closure for a set of ground equations over a signature consisting of binary function symbols that are associative and commutative. It is not obvious how the traditional dag-based algorithms can be modified to handle associativity and commutativity of certain function symbols, though commutativity alone is easily handled by simple modifications; see comments on page 767 of [15].

Let Σ be a signature with arity function α, and E a set of ground equations over Σ. Let Σ_{AC} be some subset of Σ, containing all the associative-commutative operators. We denote by P the identities

$$f(x_1, \ldots, x_k, s, y_1, \ldots, y_l, t, z_1, \ldots, z_m) \approx f(x_1, \ldots, x_k, t, y_1, \ldots, y_l, s, z_1, \ldots, z_m),$$

where $f \in \Sigma_{AC}$, $k, l, m \geq 0$, and $k + l + m + 2 \in \alpha(f)$, and by F the set of identities

$$f(x_1, \ldots, x_m, f(y_1, \ldots, y_r), z_1, \ldots, z_n) \approx f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n),$$

where $f \in \Sigma_{AC}$ and $\{m + n + 1, m + n + r, r\} \subset \alpha(f)$. The congruence induced by all ground instances of P is called a *permutation congruence*. Flattening refers to normalizing a term with respect to the set F (considered as a rewrite rule). The set $AC = F \cup P$ defines an AC-theory. The symbols in Σ_{AC} are called associative-commutative operators.* We require that $\alpha(f)$ be a singleton set for all $f \in \Sigma - \Sigma_{AC}$ and $\alpha(f) = \{2, 3, 4, \ldots\}$ for all $f \in \Sigma_{AC}$.

We note that apart from the D-rules and the C-rules, in the presence of AC-symbols we additionally need A-rules.

DEFINITION 5. Let Σ be a signature and K be a set of constants disjoint from Σ. Equations that when fully flattened are of the form $f(c_1, \ldots, c_k) \approx f(d_1, \ldots, d_l)$,

* The equations $F \cup P$ define a conservative extension of the theory of associativity and commutativity to varyadic terms. For a fixed arity binary function symbol, the equations $f(x, y) \approx f(y, x)$ and $f(f(x, y), z) \approx f(x, f(y, z))$ define an AC-theory.

where $f \in \Sigma_{AC}$ and $c_1, \ldots, c_k, d_1, \ldots, d_l \in K$, will be called *A-equations*. Directed A-equations are called *A-rules*.

We can now generalize all definitions made in Section 2 to the case when certain function symbols are known to be associative and commutative. By $AC\backslash R$ we denote the rewrite system consisting of all rules $u \to v$ such that $u \leftrightarrow^*_{AC} u'\sigma$ and $v = v'\sigma$, for some rule $u' \to v'$ in R and some substitution σ. We say that $AC\backslash R$ is confluent modulo AC if for all terms s, t such that $s \leftrightarrow^*_{R\cup AC} t$, there exist terms w and w' such that $s \to^*_{AC\backslash R} w \leftrightarrow^*_{AC} w' \leftarrow^*_{AC\backslash R} t$. We speak of *ground confluence* if this condition is true for all ground terms s and t. The other definitions are analogous.

Part of the condition for confluence modulo AC can be satisfied by the inclusion of so-called extensions of rules [24]. Given an AC-operator f and a rewrite rule $\rho\colon f(c_1, c_2) \to c$, we consider its extension $\rho^e\colon f(f(c_1, c_2), x) \to f(c, x)$. Given a set of rewrite rules R, by R^e we denote the set R plus extensions of rules in R. Extensions have to be used for rewriting terms and computing critical pairs when working with AC-symbols. The key property of extended rules is that whenever a term t is reducible by $AC\backslash R^e$ and $t \leftrightarrow^*_{AC} t'$, then t' is also reducible by $AC\backslash R^e$.

DEFINITION 6. Let R be a set of D-rules, C-rules, and A-rules (with respect to Σ and K). We say that a constant c in K *represents* a term t in $\mathcal{T}(\Sigma \cup K)$ (via the rewrite system R) if $t \leftrightarrow^*_{AC\backslash R^e} c$. A term t is also said to be *represented* by R if it is represented by some constant via R.

DEFINITION 7. Let Σ be a signature and K be a set of constants disjoint from Σ. A ground rewrite system $R = A \cup D \cup C$ is said to be an *associative-commutative congruence closure* (with respect to Σ and K) if

(i) D is a set of D-rules, C is a set of C-rules, A is a set of A-rules, and every constant $c \in K$ represents at least one term $t \in \mathcal{T}(\Sigma)$ via R, and
(ii) $AC\backslash R^e$ is ground convergent modulo AC over $\mathcal{T}(\Sigma \cup K)$.

In addition, if E is a set of ground equations over $\mathcal{T}(\Sigma \cup K)$ such that

(iii) if s and t are terms over $\mathcal{T}(\Sigma)$, then $s \leftrightarrow^*_{AC\cup E} t$ if, and only if, $s \to^*_{AC\backslash R^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC\backslash R^e} t$,

then R will be called an associative-commutative congruence closure *for* E.

When Σ_{AC} is empty, this definition specializes to that of an abstract congruence closure in Definition 2.

For example, let Σ consist of function symbols, a, b, c, f, and g (f is AC), and let E_0 be a set of three equations $f(a, c) \approx a$, $f(c, g(f(b, c))) \approx b$ and $g(f(b, c)) \approx f(b, c)$. Using extension and orientation, we can obtain a representation of the equations in E_0 using D-rules and C-rules as

$$R_1 = \{a \to c_1, b \to c_2, c \to c_3, f(c_2, c_3) \to c_4,$$
$$g(c_4) \to c_5, f(c_1, c_3) \to c_1, f(c_3, c_5) \to c_2, c_5 \to c_4\}.$$

However, the rewrite system R_1 above is not a congruence closure for E_0, since it is not a ground convergent rewrite system. But we can transform R_1 into a suitable rewrite system, using a completion-like (modulo AC) process described in more detail in the next section, to obtain a congruence closure (details are given in Example 5),

$$R' = \{a \to c_1, b \to c_2, c \to c_3, fc_2c_3 \to c_4, fc_3c_4 \to c_2, fc_1c_3 \to c_1, \\ fc_2c_2 \to fc_4c_4, fc_1c_2 \to fc_1c_4, gc_4 \to c_4\},$$

that provides a more compact representation of E_0. Attempts to replace every A-rule by two D-rules (introducing a new constant in the process) lead to non-terminating derivations.

5.1. CONSTRUCTION OF ASSOCIATIVE-COMMUTATIVE CONGRUENCE CLOSURE

Let U be a set of symbols from which new names (constants) are chosen. We need a (partial) AC-compatible reduction ordering that orients the D-rules in the right way and orients all the C- and A-equations. The precedence-based AC-compatible ordering $\succ$ of [26], with any precedence in which $f \succ_{\Sigma \cup U} c$, whenever $f \in \Sigma$ and $c \in U$, serves the purpose. Much simpler partial orderings would suffice, too, but for convenience we use the ordering in [26]. In our case, this simply means that orientation of D-rules is from left to right and that the orientation of an A-rule is given by comparing the fully flattened terms as follows: $f(c_1, \ldots, c_i) \succ f(c'_1, \ldots, c'_j)$ iff either $i > j$, or $i = j$ and $\{c_1, \ldots, c_i\} \succ^{mult} \{c'_1, \ldots, c'_j\}$. That is, if the two terms have the same number of arguments, we compare the multisets of constants using a multiset extension $\succ^{mult}$ of the precedence $\succ_{\Sigma \cup U}$; see [13].

We next present a general method for construction of associative-commutative congruence closures. Our description is fairly abstract, in terms of transition rules that operate on triples (K, E, R), where K is a set of new constants that are introduced (the original signature Σ is fixed); E is a set of ground equations (over $\Sigma \cup K$) yet to be processed; and R is a set of C-rules, D-rules and A-rules. Triples represent possible *states* in the process of constructing a closure. The initial state is $(\emptyset, E, \emptyset)$, where E is the input set of ground equations.

New constants are introduced by the following transition.

$$\textbf{Ext}\text{ension:} \quad \frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \to c\})}$$

if $t \to c$ is a D-rule, $c \in U - K$, and t occurs in some equation in E that is neither an A-equation nor a D-equation.

Once a D-rule has been introduced by extension, it can be used to simplify equations.

$$\textbf{Sim}\text{plification:} \quad \frac{(K, E[s], R)}{(K, E[t], R)}$$

where s occurs in some equation in E, and, $s \rightarrow_{AC \setminus R^e} t$.

It is fairly easy to see that any equation in E can be transformed to a D-, a C-, or an A-equation by suitable extension and simplification.*

Equations are moved from the second to the third component of the state by orientation. All rules added to the third component are C-rules, D-rules, or A-rules.

$$\textbf{Ori}\text{entation:} \quad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R \cup \{s \rightarrow t\})}$$

if $s \succ t$, and $s \rightarrow t$ is a D-rule, a C-rule, or an A-rule.

Deletion allows us to delete trivial equations.

$$\textbf{Del}\text{etion:} \quad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R)}$$

if $s \leftrightarrow^*_{AC} t$.

We consider overlaps between extensions of A-rules in ACSuperposition.

$$\textbf{ACSup}\text{erposition:} \quad \frac{(K, E, R)}{(K, E \cup \{f(s, x\sigma) \approx f(t, y\sigma)\}, R)}$$

if $f \in \Sigma_{AC}$, there exist D- or A-rules (fully flattened as) $f(c_1, \ldots, c_k) \rightarrow s$ and $f(d_1, \ldots, d_l) \rightarrow t$ in R, the sets $C = \{c_1, \ldots, c_k\}$ and $D = \{d_1, \ldots, d_l\}$ are not disjoint, $C \not\subseteq D$, $D \not\subseteq C$, and the substitution σ is the ground substitution in a minimal complete set of AC-unifiers for $f(c_1, \ldots, c_k, x)$ and $f(d_1, \ldots, d_l, y)$.**

In the special case when one multiset is contained in the other, we obtain the ACCollapse rule.

$$\textbf{ACCol}\text{lapse:} \quad \frac{(K, E, R \cup \{t \rightarrow s\})}{(K, E \cup \{t' \approx s\}, R)}$$

if for some $u \rightarrow v \in R$, $t \rightarrow_{AC \setminus \{u \rightarrow v\}^e} t'$, and if $t \leftrightarrow^*_{AC} u$, then $s \succ v$.

The Deduction inference rule in Section 2.1 (for non-AC terms) is subsumed by ACCollapse. Note that we do not explicitly add AC extensions of rules to the set R. Consequently, any rule in R is a C-rule, a D-rule, or an A-rule and *not* its extension. We implicitly work with extensions in ACSuperposition.

We need additional transition rules to perform simplifications on the left- and right-hand sides of other rules. The use of C-rules to simplify left-hand sides of rules is captured by ACCollapse. The simplification on the right-hand sides is subsumed by the following generalized composition rule.

$$\textbf{Com}\text{position:} \quad \frac{(K, E, R \cup \{t \rightarrow s\})}{(K, E, R \cup \{t \rightarrow s'\})}$$

if $s \rightarrow_{AC \setminus R^e} s'$.

* We do not need an explicit rule for flattening because Definition 5 allows for nonflattened terms to occur in A-rules.

** For the special case in hand, a minimal complete set of AC-unifiers contains exactly two substitutions, exactly one of which is ground.

EXAMPLE 5. Let $E_0 = \{f(a,c) \approx a, f(c,g(f(b,c))) \approx b, g(f(b,c)) \approx f(b,c)\}$. We show some intermediate states of a derivation below (superscripts in the last column indicate the number of applications of the respective rules). We assume that f is AC and $c_i \succ c_j$ if $i < j$.

i	Constants K_i	Equations E_i	Rules R_i	Transitions
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_1, c_3\}$	$\{fcgfbc \approx b,$ $gfbc \approx fbc\}$	$\{a \to c_1, c \to c_3,$ $fc_1c_3 \to c_1\}$	$\mathbf{Ext}^2 \cdot \mathbf{Sim} \cdot$ **Ori**
2	$K_1 \cup \{c_2, c_4\}$	$\{fcgfbc \approx b\}$	$R_1 \cup \{b \to c_2,$ $fc_2c_3 \to c_4, gc_4 \to c_4\}$	$\mathbf{Sim}^2 \cdot \mathbf{Ext}^2 \cdot$ **Sim** $\cdot$ **Ori**
3	K_2	$\emptyset$	$R_2 \cup \{fc_3c_4 \to c_2\}$	$\mathbf{Sim}^6 \cdot$ **Ori**
4	K_2	$\emptyset$	$R_3 \cup \{fc_1c_2 \to fc_1c_4\}$	**ACSup** $\cdot$ **Ori**
5	K_2	$\emptyset$	$R_4 \cup \{fc_2c_2 \to fc_4c_4\}$	**ACSup** $\cdot$ **Ori**

The derivation moves equations, one by one, from the second component of the state to the third component through simplification, extension, and orientation. It can be verified that the set R_5 is an AC congruence closure for E_0. There are more ACSuperpositions, but the resulting equations get deleted. Note that the side-condition in extension disallows breaking of an A-rule into two D-rules, which is crucial for termination.

5.2. TERMINATION AND CORRECTNESS

DEFINITION 8. We use the symbol $\vdash$ to denote the one-step transition relation on states induced by the above transition rules. *A derivation* is a sequence of states $(K_0, E_0, R_0) \vdash (K_1, E_1, R_1) \vdash \cdots$. A derivation is said to be *fair* if any transition rule that is continuously enabled is eventually applied. The set R_∞ of persisting rules is defined as $\bigcup_i \bigcap_{j>i} R_j$; and similarly, $K_\infty = \bigcup_i \bigcap_{j>i} K_j$.

We shall prove that any fair derivation will generate only finitely many persisting rewrite rules (in the third component) by using Dickson's lemma [8]. Multisets over K_∞ can be compared by using the multiset inclusion relation. If K_∞ is finite, this relation defines a Dickson partial order.

LEMMA 6. *Let E be a finite set of ground equations. The set of persisting rules R_∞ in any fair derivation starting from state $(\emptyset, E, \emptyset)$ is finite.*

Proof. We first claim that K_∞ is finite. To see this, note that new constants are created by extension. Using finitely many applications of extension, simplification, and orientation, we can move all rules from the initial second component E of the state tuple to the third component R. Fairness ensures that this situation will eventually happen. Thereafter, any equations added to E can be oriented by using orientation; hence we never apply extension subsequently (see the side condition of the extension rule). Let $K_\infty = \{c_1, \ldots, c_n\}$.

Next we claim that the set R_∞ is finite. Suppose R_∞ is an infinite set. Since non-Σ_{AC} symbols have fixed arities, R_∞ contains infinitely many rules with top symbol from Σ_{AC}. Since Σ_{AC} is finite, one AC-operator, say $f \in \Sigma_{AC}$, must occur infinitely often as the top symbol in the left-hand sides of R_∞. By Dickson's lemma, there exists an infinite chain of rules (written as fully flattened for simplicity), $f(c_{11}, \ldots, c_{1k_1}) \to s_0$, $f(c_{21}, \ldots, c_{2k_2}) \to s_1, \ldots$, such that $\{c_{11}, \ldots, c_{1k_1}\} \subseteq \{c_{21}, \ldots, c_{2k_2}\} \subseteq \cdots$, where $\{c_{i1}, \ldots, c_{ik_i}\}$ denotes a multiset and $\subseteq$ denotes multiset inclusion. But, this contradicts fairness (in application of ACCollapse). □

5.3. PROOF ORDERING

The correctness of the procedure will be established by using proof simplification techniques for associative-commutative completion, as described by Bachmair [1] and Bachmair and Dershowitz [2]. In fact, we can directly use the results and the proof measure from [2]. However, since all rules in R have a special form, we can choose a simpler proof ordering. One other difference is that we do not have explicit transition rules to create extensions of rules in the third component. Instead we use extensions of rules for simplification and computation of superpositions.

Let $s = s[u\sigma] \leftrightarrow s[v\sigma] = t$ be a proof step using the equation (rule) $u \approx v \in AC \cup E \cup R$. The complexity of this proof step is defined by

$$\begin{array}{llll} (\{s,t\}, \bot, \bot) & \text{if } u \approx v \in E & (\{s\}, \bot, t) & \text{if } u \approx v \in AC \\ (\{s\}, u, t) & \text{if } u \to v \in R & (\{t\}, v, s) & \text{if } v \to u \in R \end{array}$$

where $\bot$ is a new symbol. Tuples are compared lexicographically using the multiset extension of the reduction ordering $\succ$ on terms over $\Sigma \cup K_\infty$ in the first component, and the ordering $\succ$ in the second and third component. The constant $\bot$ is assumed to be minimum. The complexity of a proof is the multiset of complexities of its proof steps. The multiset extension of the ordering on tuples yields a proof ordering, denoted by the symbol $\succ_{\mathcal{P}}$. The ordering $\succ_{\mathcal{P}}$ on proofs is well founded because it is a lexicographic combination of well-founded orderings.

LEMMA 7. *Suppose* $(K, E, R) \vdash (K', E', R')$. *Then, for any two terms* $s, t \in \mathcal{T}(\Sigma)$, *it is the case that* $s \leftrightarrow^*_{AC\cup E'\cup R'} t$ *iff* $s \leftrightarrow^*_{AC\cup E\cup R} t$. *Further, for any* $s_0, s_k \in \mathcal{T}(\Sigma \cup K)$, *if* π *is a ground proof* $s_0 \leftrightarrow s_1 \leftrightarrow \cdots \leftrightarrow s_k$ *in* $AC \cup E \cup R$, *then there is a proof* π' $s_0 = s'_0 \leftrightarrow s'_1 \leftrightarrow \cdots \leftrightarrow s'_l = s_k$ *in* $AC \cup E' \cup R'$ *such that* $\pi \succeq_{\mathcal{P}} \pi'$.

Proof. The first part of the lemma, which states that the congruence on $\mathcal{T}(\Sigma)$ remains unchanged, is easily verified by exhaustively checking it for each transition rule. In fact, except for extension, all the other transition rules are standard rules for completion modulo a congruence, and hence the result follows. Consider the case when the state $(K' = K \cup \{c\}, E', R' = R \cup \{t \to c\})$ is obtained from the state (K, E, R) by extension. Now, if $s \leftrightarrow^*_{AC\cup E\cup R} t$, then clearly $s \leftrightarrow^*_{AC\cup E'\cup R'} t$. Conversely, if $s \leftrightarrow^*_{AC\cup E'\cup R'} t$, then we replace all occurrences of c in this proof by t to get a proof in $AC \cup E \cup R$.

For the second part, one needs to check that each equation in $(E-E')\cup(R-R')$ has a simpler proof in $E' \cup R' \cup AC$ for each transition rule application; see [2]. In detail, we have the following cases:

(i) Extension. The proof $s[t] \leftrightarrow_E u$ is replaced by a proof $s[t] \rightarrow_{R'} s[c] \leftrightarrow_{E'} u$, and the new proof is smaller as $\{s[t], u\} \succ^m \{s[t]\}$, and $\{s[t], u\} \succ^m \{s[c], u\}$.

(ii) Simplification. The proof $r[s] \leftrightarrow_E u$ is replaced by the new proof $r[s] \leftrightarrow^*_{AC} r' \rightarrow_{R'} r[t] \leftrightarrow_{E'} u$.* Now, $\{r[s], u\} \succ^m \{r''\}$ for every term r'' in the sequence of terms $r[s] \leftrightarrow^*_{AC} r'$, and $\{r[s], u\} \succ^m \{r[t], u\}$.

(iii) ACCollapse. The proof $t \rightarrow_R s$ is transformed to the smaller proof $t \leftrightarrow^*_{AC} t' \rightarrow_{\{u\rightarrow v\}} t'' \leftrightarrow_{E'} s$. This new proof is smaller because the rewrite step $t \rightarrow_R s$ is more complex than (a) all proof steps in $t \leftrightarrow^*_{AC} t'$ (in the second component); (b) the proof step $t' \rightarrow_{\{u\rightarrow v\}} t''$ in the second component if $t \not\leftrightarrow^*_{AC} u$, and in the third component if $t \leftrightarrow^*_{AC} u$ (see side condition in ACCollapse); and (c) the proof step $t'' \leftrightarrow_{E'} s$ (in the first component).

(iv) Orientation. In this case, $s \leftrightarrow_E t$ is more complex than the new proof $s \rightarrow_{R'} t$, and this follows from $\{s, t\} \succ^m \{s\}$.

(v) Deletion. We have $s \leftrightarrow_E t$ more complex than $s \leftrightarrow^*_{AC} t$ because $\{s, t\} \succ^m \{s'\}$ for every s' in $s \leftrightarrow^*_{AC} t$.

(vi) Composition. We have the proof $t \rightarrow_R s$ transformed to the smaller proof $t \rightarrow_R s' \leftarrow_{R'} s'' \leftrightarrow^*_{AC} s$. This new proof is smaller because the rewrite step $t \rightarrow_R s$ is more complex than (a) the rewrite step $t \rightarrow_{R'} s'$ in the third component, (b) all proof steps in $s'' \leftrightarrow^*_{AC} s$ in the first component, and (c) the rewrite step $s'' \rightarrow_{R'} s'$ in the first component.

The ACSuperposition transition rule does not delete any equation. This completes the proof of the lemma. □

Note that in any derivation, extensions of rules are not added explicitly, and hence they are never deleted either. Once we converge to R_∞, we introduce extensions to take care of cliffs in proofs.

LEMMA 8. *If R_∞ is a set of persisting rules of a fair derivation starting from the state $(\emptyset, E, \emptyset)$, then R^e_∞ is a ground convergent (modulo AC) rewrite system. Furthermore, $E_\infty = \emptyset$.*

Proof. Fairness implies that all critical pairs (modulo AC) between rules in R^e_∞ are contained in the set $\bigcup_i E_i$. Since a fair derivation is nonfailing, $E_\infty = \emptyset$. Since the proof ordering is well founded, for every proof in $E_i \cup R_i \cup AC$, there exists a minimal proof π in $E_\infty \cup R_\infty \cup AC$. We argue by contradiction that certain proof patterns cannot occur in the minimal proof π: specifically, there can be no peaks $s \leftarrow_{R^e_\infty} u \rightarrow_{AC\setminus R^e_\infty} t$, nonoverlap cliffs, or variable overlap cliffs.

(i) Peaks. A peak caused by a nonoverlap or a variable overlap $s \leftarrow_{R^e_\infty} u \rightarrow_{AC\setminus R^e_\infty} t$ can be transformed to a simpler proof $s \rightarrow^*_{AC\setminus R^e_\infty} v \leftarrow^*_{AC\setminus R^e_\infty} t$.

* Note that we used extended rules in specifying simplification, but for purposes of proof transformations, we consider only the original (nonextended) rules as being present in the third component.

The new proof is simpler because u is bigger than each term in the new proof. Next suppose that the above pattern is caused by a proper overlap. In this case, it is easy to see that $s \leftrightarrow^*_{AC} s' \leftrightarrow_{CP_{AC}(R^e_\infty)} t' \leftrightarrow^*_{AC} t$, where $CP_{AC}(R^e_\infty)$ denotes the set of all equations created by ACSuperposition and ACCollapse transition rules applied on the rules in R^e_∞. Since by fairness $CP_{AC}(R^e_\infty) \subseteq \bigcup_k E_k$, there is a proof $s \leftrightarrow^*_{AC} s' \leftrightarrow_{E_k} t' \leftrightarrow^*_{AC} t$ for some $k \geq 0$. This proof, which we name π, is strictly smaller than the original peak. Using Lemma 7, we may infer that there is a proof π' in $AC \cup R_\infty$ such that π' is strictly smaller than the original peak, a contradiction.

(ii) Cliffs. A nonoverlap cliff $w[v, s] \leftrightarrow_{AC} w[u, s] \rightarrow_{AC \setminus R^e_\infty} w[u, t]$ can be transformed to the following less complex proof: $w[v, s] \rightarrow_{AC \setminus R^e_\infty} w[v, t] \leftrightarrow_{AC} w[u, t]$. Clearly, $w[v, s] \succ w[v, t]$ and hence the proof $w[v, t] \leftrightarrow_{AC} w[u, t]$ is smaller than the proof $w[v, s] \leftrightarrow_{AC} w[u, s]$ (in the first component). The complexity of the proof $w[u, s] \rightarrow_{AC \setminus R^e_\infty} w[u, t]$ is identical to the complexity of the proof $w[v, s] \rightarrow_{AC \setminus R^e_\infty} w[v, t]$.

In the case of AC, a variable overlap cliff $s \leftrightarrow_{AC} u \rightarrow_{AC \setminus R^e_\infty} t$ can be eliminated in favor of the proof $s \rightarrow_{AC \setminus R^e_\infty} t' \leftrightarrow_{AC} t$. Note that the proof $u \rightarrow_{AC \setminus R^e_\infty} t$ and the proof $s \rightarrow_{AC \setminus R^e_\infty} t'$ are of the same complexity, and additionally the proof $s \leftrightarrow_{AC} u$ is larger than the proof $t' \leftrightarrow_{AC} t$ as all terms in the latter proof are smaller than u.

In summary, the proof π cannot contain peaks $s \leftarrow_{R^e_\infty} u \rightarrow_{AC \setminus R^e_\infty}$, or nonoverlap or variable overlap cliffs $s \leftrightarrow_{AC} u \rightarrow_{AC \setminus R^e_\infty} t$. The cliffs arising from proper overlaps can be replaced by extended rules, as $(R^e_\infty)^e = R^e_\infty$. The minimal proof π in $R_\infty \cup AC$ can, therefore, be only of the form $s \rightarrow^*_{AC \setminus R^e_\infty} s' \leftrightarrow^*_{AC} t' \leftarrow^*_{AC \setminus R^e_\infty} t$, which is a rewrite proof. □

Note that we did not define the proof complexities for the extended rules, since the rules are introduced only at the end. Hence, the argument given here is not identical to the one in [2], though it is similar. Using Lemmas 7 and 8, we can easily prove the following.

THEOREM 5. *Let R_∞ be the set of persisting rules of a fair derivation starting from state $(\emptyset, E, \emptyset)$. Then, the set R^e_∞ is an associative-commutative congruence closure for E.*

Proof. To show that R_∞ is an associative-commutative congruence closure for E_0, we need to prove the three conditions in Definition 7.

(1) The transition rules ensure that R_∞ consists of only D-rules, C-rules, and A-rules. We prove that every constant represents some term in $\mathcal{T}(\Sigma)$ by induction. Let c be any constant in K_∞. Since all constants are added by extension, let $f(c_1, \ldots, c_k) \rightarrow c$ be the rule introduced by extension when c was added. As induction hypothesis we can assume that all constants added before c represent a term in $\mathcal{T}(\Sigma)$ via R_∞. Therefore, there exist terms $s_1, \ldots, s_k \in \mathcal{T}(\Sigma)$ such that $s_i \leftrightarrow^*_{AC \setminus R^e_\infty} c_i$, and hence

$$f(s_1, \ldots, s_k) \leftrightarrow^*_{AC \setminus R^e_\infty} f(c_1, \ldots, c_k) \rightarrow_{\cup_i R_i} c.$$

Using Lemma 7, we get $f(s_1, \ldots, s_k) \leftrightarrow^*_{R^e_\infty \cup E_\infty \cup AC} c$. Lemma 8 shows that $E_\infty = \emptyset$, and $f(s_1, \ldots, s_k) \leftrightarrow^*_{AC \setminus R^e_\infty} c$.

(2) Lemma 8 shows that $AC \setminus R^e_\infty$ is ground convergent.

(3) Let $s, t \in \mathcal{T}(\Sigma)$. Using Lemma 7, we know $s \leftrightarrow^*_{E \cup AC} t$ if, and only if, $s \leftrightarrow^*_{E_\infty \cup R_\infty \cup AC} t$. Since $E_\infty = \emptyset$, Lemma 8 implies that $s \rightarrow^*_{AC \setminus R^e_\infty} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC \setminus R^e_\infty} t$. □

Since R_∞ is finite, there exists a k such that $R_\infty \subseteq R_k$. Thus, the set of persisting rules can be obtained by using only finite derivations.

5.4. OPTIMIZATIONS

The set of transition rules for computing an AC congruence closure can be further enhanced by additional simplifications and optimizations. First, we can flatten terms in E.

$$\textbf{Fla}\text{ttening:} \quad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E \cup \{u \approx t\}, R)}$$

where $s \rightarrow_F u$. Now, however, the correctness proof given above, Lemma 7 in particular, fails because the new proof $s \leftrightarrow_{AC} u \leftrightarrow_{E'} t$ of the deleted equation $s \approx t$ is larger than the old proof $s \leftrightarrow_{E'} t$. But we can still establish the correctness of the extended set of inference rules as follows. Assume that flattening does not delete the equation $s \approx t$ from E but only marks it. All subsequent derivation steps do not work on the marked equations. Once the derivation converges (ignoring the marked equations), we can delete the marked equations as any such equation, say $s \approx t$, would have a proof $s \leftrightarrow_{AC} u \leftrightarrow_{AC \cup R_\infty} t$, and hence also a desired rewrite proof (using the persisting set of rewrite rules).

As a consequence of the flattening rule, we can construct fully flattened AC congruence closures, that is, where each term in the congruence closure is fully flattened.

As a second optimization, the extension variable of a rewrite rule can be constrained to allow for fine-grained deletion of instances of rewrite rules. For example, after deducing the critical pair $fc_1c_2 \approx fc_2c_3$ that arises by overlapping the rules $fc_1c_2x \rightarrow fc_2x$ and $fc_1c_1y \rightarrow fc_3y$, we can delete the instance $fc_1c_1c_2 \rightarrow fc_3c_2$ of the latter rule as it has a smaller proof $fc_1c_1c_2 \rightarrow fc_1c_2 \approx fc_2c_3$ using the deduced equation. We can delete this instance by replacing the rule $fc_1c_1y \rightarrow fc_3y$ by the new rule $fc_1c_1y \rightarrow fc_3y$ `if` C, where C is the constraint that "y is not of the form $f(c_2, z)$." These new constraints can be carried to new equations generated in a deduction step.

Finally, we note that, as in the case of congruence closure discussed before, we can choose the ordering between two constants in K on the fly. As an optimization we could always choose it in a way so as to minimize the applications

of ACCollapse and composition later. In other words, when we need to choose the orientation for $c \approx d$, we can count the number of occurrences of c and d in the set of D- and A-rules (in the R-component of the state), and the constant with fewer occurrences is made larger.

5.5. PROPERTIES

The results in the preceding sections establish the decidability of the word problem for ground theories presented over a signature containing finitely many associative-commutative symbols. Note that we are implicitly decomposing the equations (over a signature consisting of several symbols) into equations over exactly one function symbol and a set of new constants. A set of equations over exactly one AC symbol and finitely many constants defines a finitely presented commutative semigroup.

The word problem for commutative semigroups is known to be complete for deterministic *EXP* space [9]. It is a simple observation that the word problem for commutative semigroups can be reduced to the ideal membership problem for binomial ideals. In fact, an optimal exponential space algorithm for generating the reduced Gröbner basis of binomial ideals was presented in [19], but that algorithm was not based on critical pair completion.

Thus, using the approach proposed in our paper, we can construct an AC congruence closure in time $O(n|\Sigma|T(n))$ and space $O(n^2 + S(n))$ using an algorithm for constructing Gröbner bases for binomial ideals that uses $O(T(n))$ time and $S(n)$ space. We have not worked out the time complexity of the critical pair completion-based algorithm (as presented in our paper) for constructing Gröbner bases for binomial ideals. That remains as future work.

6. Construction of Ground Convergent Rewrite Systems

We have presented transition rules for constructing a convergent presentation in an extended signature for a set of ground equations. We next discuss the problem of obtaining a ground convergent (AC) rewrite system for the given ground (AC-) theory *in the original signature*. Hence, now we focus our attention on the problem of transforming a convergent system over an extended signature to a convergent system in the original signature.

The basic idea of transforming back is elimination of constants from the presentation R as follows: (i) if a constant c is not redundant (Definition 3), then we pick a term $t \in \mathcal{T}(\Sigma)$ that is represented by c and replace all occurrences of c by t in R; (ii) if a constant c is redundant (and say $c \to d$ is a C-rule in which c occurs as the left-hand side term), then all occurrences of c can be replaced by d in R.

In the case when there are no AC-symbols in the signature, the above method generates a ground convergent system from any given abstract congruence closure. This gives an indirect way to construct ground convergent systems equivalent to

a given set of ground equations. However, we run into problems when we use the same method for translation in presence of AC-symbols. Typically, after translating back, the set of rules obtained is nonterminating modulo AC (see Example 6). But if we suitably define the notion of AC-rewriting, the rules are seen to be convergent in the new definition. This is useful in two ways: (i) the new notion of AC-rewriting seems to be more practical, in the sense that it involves strictly less work than a usual $AC\backslash R^e$ reduction; and (ii) it helps to clarify the advantage offered by the use of extended signatures when dealing with a set of ground equations over a signature containing associative and commutative symbols.

6.1. TRANSITION RULES

We describe the process of transforming a rewrite system over an extended signature $\Sigma \cup K$ to a rewrite system over the original signature Σ by transformation rules on states (K, R), where K is the set of constants to be eliminated and R is a set of rewrite rules over $\Sigma \cup K$ to be transformed.

Redundant constants can be easily eliminated by the compression rule.

$$\textbf{Comp}\text{ression:} \quad \frac{(K \cup \{c\}, R \cup \{c \to t\})}{(K, R\langle c \mapsto t\rangle)}$$

where $\langle c \mapsto t\rangle$ denotes the (homomorphic extension of the) mapping $c \mapsto t$, and $R\langle c \mapsto t\rangle$ denotes the application of this homomorphism to each term in the set R.

The basic idea for eliminating a constant c that is not redundant in R involves picking a representative term t (over the signature Σ) in the equivalence class of c and replacing c by t everywhere in R.

$$\textbf{Sel}\text{ection:} \quad \frac{(K \cup \{c\}, R \cup \{t \to c\})}{(K, R\langle c \mapsto t\rangle \cup R')}$$

if (i) c is not redundant in R, (ii) $t \in \mathcal{T}(\Sigma)$, and (iii) if $t \equiv f(t_1, \ldots, t_k)$ with $f \in \Sigma_{AC}$ then $R' = \{f(t_1, \ldots, t_k, X) \to f(f(t_1, \ldots, t_k), X)\}$; otherwise $R' = \emptyset$.

If $\Sigma_{AC} = \emptyset$, we note that R' will always be empty. We also require that terms *not* be flattened after the application of mapping $R\langle c \mapsto t\rangle$. The variable X is a special sequence variable that can be instantiated only by nonempty sequences. We shall formally define its role later.

EXAMPLE 6. Consider the problem of constructing a ground convergent system for the set E_0 from Example 5. A fully reduced congruence closure for E_0 is given by the set R_0

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl@{\qquad}rcl}
a & \to & c_1 & b & \to & c_2 & c & \to & c_3 & fc_2c_3 & \to & c_4 \\
fc_3c_4 & \to & c_2 & fc_1c_3 & \to & c_1 & fc_2c_2 & \to & fc_4c_4 & fc_1c_2 & \to & fc_1c_4 \\
gc_4 & \to & c_4 & & & & & & & & &
\end{array}$$

under the ordering $c_2 \succ c_4$ between constants. For the constants c_1, c_2, and c_3 we have no choice but to choose a, b, and c as representatives, respectively. Thus, after three applications of selection, we get

$$\begin{array}{lll} fcc_4 \to b & fac \to a & fbb \to fc_4c_4 \\ fbc \to c_4 & gc_4 \to c_4 & fab \to fac_4. \end{array}$$

Next we are forced to choose fbc as the representative for the class c_4. This gives us the transformed set R_1,

$$\begin{array}{lll} fc(fbc) \to b & fac \to a & fbb \to f(fbc)(fbc) \\ fbcX \to f(fbc)X & gfbc \to fbc & fab \to fa(fbc). \end{array}$$

The relation $\to_{AC\setminus R_1^e}$ is clearly nonterminating (with the variable X considered as a regular term variable).

6.2. REWRITING WITH SEQUENCE EXTENSIONS MODULO PERMUTATION CONGRUENCE

Let X denote a variable ranging over nonempty sequences of terms. A sequence substitution σ is a substitution that maps variables to the sequences. If σ is a sequence substitution that maps X to the sequence $\langle s'_1, \ldots, s'_m \rangle$, then $f(s_1, \ldots, s_k, X)\sigma$ is the term $f(s_1, \ldots, s_k, s'_1, \ldots, s'_m)$.

DEFINITION 9. Let ρ be a ground rule of the form $f(t_1, \ldots, t_k) \to g(s_1, \ldots, s_m)$ where $f \in \Sigma_{AC}$. We define the *sequence extension* ρ^s of ρ as $f(t_1, \ldots, t_k, X) \to f(s_1, \ldots, s_m, X)$ if $f = g$, and as $f(t_1, \ldots, t_k, X) \to f(g(s_1, \ldots, s_m), X)$ if $f \neq g$.

Now we are ready to define the notion of rewriting we use. Recall that P denotes the equations defining the permutation congruence, and that $AC = F \cup P$. Given a set R, we denote by R^s the set R plus sequence extensions of all ground rules in R.

DEFINITION 10. Let R be a set of rewrite rules. For ground terms $s, t \in \mathcal{T}(\Sigma)$, we say that $s \to_{P\setminus R^s} t$ if there exists a rule $l \to r \in R^s$ and a sequence substitution σ such that $s = C[l']$, $l' \leftrightarrow_P^* l\sigma$, $r' = r\sigma$, and $t = C[r']$.

Note that the difference with standard rewriting modulo AC is that instead of performing matching modulo AC, we do matching modulo P. For example, if ρ is $fac \to a$, then the term $f(f(a, b), c)$ is not reducible by $\to_{P\setminus\rho^s}$, although it is reducible by $\to_{AC\setminus\rho^e}$. The term $f(f(a, b), c, a)$ can be rewritten by $\to_{P\setminus\rho^s}$ to $f(f(a, b), a)$.

EXAMPLE 7. Following up on Example 6, we note that the relation $P\setminus R_1^s$ is convergent. For instance, a normalizing rewrite derivation for the term $fabc$ is

$$fabc \to_{P\setminus R_1^s} fa(fbc)c \to_{P\setminus R_1^s} fab \to_{P\setminus R_1^s} fa(fbc).$$

On closer inspection, we find that we are essentially doing a derivation in the original rewrite system R_0 (over the extended signature),

$$fc_1c_2c_3 \to_{P\setminus R_0^s} fc_1c_4c_3 \to_{P\setminus R_0^s} fc_1c_2 \to_{P\setminus R_0^s} fc_1c_4.$$

A $P\setminus R_0^s$ proof step can be projected onto a $P\setminus R_1^s$ proof step; see Lemma 9(a) and Lemma 10(a). This is at the core of the proof of correctness; see Theorem 6.

6.3. CORRECTNESS

We shall prove that compression and selection transform a fully flattened AC congruence closure over $\Sigma \cup K$ into a rewrite system R over Σ that is convergent modulo P and that defines the same equational theory over fully flattened terms over Σ. First note that any derivation starting from the state (K, R), where R is an AC congruence closure over Σ and K, is finite. This is because K is finite, and each application of compression and selection reduces the cardinality of K by one. Furthermore, in any intermediate state (K, R), R is always a rewrite system over $\Sigma \cup K$. Hence, in the final state (K_∞, R_∞), if $K_\infty = \emptyset$, then R_∞ is a rewrite system over Σ, the original signature. We shall show that K_∞ is actually empty and that the reduction relation $P\setminus R_\infty^s$ is terminating on $\mathcal{T}(\Sigma)$ and confluent on fully flattened terms in $\mathcal{T}(\Sigma)$.

In this section, we say that R is left reduced (modulo P) if every left-hand side of any rule in R is irreducible by $P\setminus\rho$ and $P\setminus\rho^s$ for every other rule ρ in R; we say that R is terminating (modulo P) if $P\setminus R^s$ is.

LEMMA 9. *Let* $(K_1, R_1 = R_0'\sigma)$ *be obtained from* $(K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{c \to u\})$ *using compression, where* $\sigma = \langle c \mapsto u\rangle$. *Assume that the rewrite system* R_0 *is left reduced and terminating. Then,*

(a) *For any two terms* $s, t \in \mathcal{T}(\Sigma \cup K_0)$, *if* $s \to_{P\setminus R_0^s} t$, *then* $s\sigma \to^{0,1}_{P\setminus R_1^s} t\sigma$.

(b) *For any two terms* $s, t \in \mathcal{T}(\Sigma \cup K_1)$, *if* $s \to_{P\setminus R_1^s} t$, *then* $s\theta \leftrightarrow^+_{P\setminus R_0^s} t\theta$, *where* $\theta = \langle u \mapsto c\rangle$.*

(c) R_1 *is left reduced and terminating.*

Proof. To prove (a), let s, t be two terms over $\Sigma \cup K_0$ such that $s = C[l_0']$, $l_0' \leftrightarrow_P^* l_0\sigma^s$, and $t = C[r_0\sigma^s]$, where $l_0 \to r_0$ is (a sequence extension of) some rule in R_0 and σ^s is a sequence substitution. Clearly, $(l_0\sigma^s)\sigma = (l_0\sigma)(\sigma^s\sigma) = l_1(\sigma^s\sigma)$, and similarly $(r_0\sigma^s)\sigma = (r_0\sigma)(\sigma^s\sigma) = r_1(\sigma^s\sigma)$, where either $l_1 = r_1$, or, $l_1 \to r_1$ is (a sequence extension of) some rule in R_1. In the first case $s\sigma \leftrightarrow_P^* t\sigma$, and in the second case $s\sigma \to_{P\setminus R_1^s} t\sigma$.

To prove (b), note that since R_0 is left reduced, a compression step has the same effect as a sequence of composition steps followed by deletion of a rule. Hence, if $s \to_{R_1^s} t$, then $s \leftrightarrow^+_{R_0^s} t$. Therefore, $s\theta \to^*_{\{c\to u\}} s \leftrightarrow^+_{R_0} t \leftarrow^*_{\{c\to u\}} t\theta$.

* Note that if θ is defined by $\langle fab \mapsto c_0\rangle$, then $fabc\theta = fabc$, but $f(fab)c\theta = fc_0c$.

To prove (c), note that termination is preserved by composition and deletion. Furthermore, the left-hand side terms do not change, and hence the system continues to remain left reduced. □

LEMMA 10. *Let* $(K_1, R_1 = R_0'\sigma \cup R')$ *be obtained from* $(K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{u \to c\})$ *using selection, where* $\sigma = \langle c \mapsto u \rangle$. *Assume that the rewrite system* R_0 *is left reduced and terminating. Then,*

(a) *For any two terms* $s, t \in \mathcal{T}(\Sigma \cup K_0)$, *if* $s \to_{P \setminus R_0^s} t$, *then* $s\sigma \to^{0,1}_{P \setminus R_1^s} t\sigma$.

(b) *For any two terms* $s, t \in \mathcal{T}(\Sigma \cup K_1)$, *if* $s \to_{P \setminus R_1^s} t$, *then* $s\theta \to^{+}_{P \setminus R_0^s} t\theta$, *where* $\theta = \langle u \mapsto c \rangle$.

(c) R_1 *is left reduced and terminating.*

Proof. The proof of (a) is identical to the proof of Lemma 9(a). Note that when $u = f(u_1, \ldots, u_k)$, where $f \in \Sigma_{AC}$, $s \leftrightarrow_P^* C[f(u_1, \ldots, u_k, X\sigma^s)]$, and $t = C[f(c, X\sigma^s)]$, the proof

$$\begin{array}{ll} s\sigma & \leftrightarrow_P^* \quad (C\sigma)[f(u_1, \ldots, u_k, X\sigma^s\sigma)] \\ & \to_{P \setminus R_1^s} \quad (C\sigma)[f(f(u_1, \ldots, u_k), X\sigma^s\sigma)] = t\sigma \end{array}$$

uses the rule in the set R'.

To prove (b), let s, t be two terms over $\Sigma \cup K_1$ such that $s \leftrightarrow_P^* C[l_1\sigma^s]$ and $t = C[r_1\sigma^s]$, where $l_1 \to r_1$ is (a sequence extension of) some rule in R_1. First consider the case when $l_1 = f(u_1, \ldots, u_k, X) \to f(f(u_1, \ldots, u_k), X) = r_1$ is the rule in R'. Since $X\sigma^s$ is nonempty,

$$s\theta \leftrightarrow_P^* (C\theta)[f(u_1, \ldots, u_k, X\sigma^s\theta)] \to_{P \setminus R_0^s} (C\theta)[f(c, X\sigma^s\theta)] = t\theta.$$

In the other case, assume $l_1 = l_0\sigma$ and $r_1 = r_0\sigma$, where $l_0 \to r_0$ is (an extension of) some rule different from $u \to c$ in R_0. Since R_0 is left reduced modulo P, $s\theta \leftrightarrow_P^* (C[(l_0\sigma)\sigma^s])\theta = (C\theta)[l_0(\sigma^s\theta)]$, and therefore we have

$$s\theta \leftrightarrow_P^* (C\theta)[l_0(\sigma^s\theta)] \to_{R_0} (C\theta)[r_0(\sigma^s\theta)] \to^*_{\{u \to c\}} (C[(r_0\sigma)\sigma^s])\theta = t\theta.$$

Since R_0 is terminating, it follows from (b) that R_1 is also terminating. Finally, to prove that $R_1 = R_0'\sigma \cup R'$ is left reduced, note that $R_0'\sigma$ is left reduced because R_0 is. Furthermore, Condition (i) in Selection and the fact that R_0 is left reduced together imply that $R_0'\sigma \cup R'$ is left reduced, too. □

The second step in the correctness argument involves showing that if $K_i \neq \emptyset$, then we can always apply either selection or compression to get to a new state.

LEMMA 11. *Let* (K_i, R_i) *be a state in the derivation starting from* (K_0, R_0), *where* $R_0 = D_0 \cup C_0 \cup A_0$ *is a left-reduced (modulo AC) associative-commutative congruence closure over the signature* $\Sigma \cup K_0$. *Assume that for every constant c in*

K_0, there exists a term t in $\mathcal{T}(\Sigma)$ such that[⋆] $t \to^*_{D_0/C_0} c$.[⋆⋆] *If $K_i \neq \emptyset$, then either selection or compression is applicable to the state (K_i, R_i).*

Proof. Since $K_i \neq \emptyset$, let c be some constant in K_i. By assumption c represents some term $t \in \mathcal{T}(\Sigma)$ such that $t \to^*_{D_0/C_0} c$.[‡] It follows from convergence of $AC \backslash R_0$ that

$$t \to^*_{D_0 \cup C_0} c' \leftarrow^*_{C_0} c.$$

Since R_0 is a left-reduced (modulo AC) congruence closure, therefore R_0 is left reduced and terminating modulo P, and hence Lemma 9 and Lemma 10 are applicable. As none of the constants in $K_0 - K_i$ occur in the terms t and c, using Lemma 9(a) and Lemma 10(a), we have

$$t \to^*_{P \backslash R_i^s} \circ \leftarrow^*_{P \backslash R_i^s} c,$$

where the right-hand side of each rule used in the above proof is either a constant or a term in $\mathcal{T}(\Sigma)$. If c is reducible by R_i, then c is a redundant constant that can be eliminated by compression. If there are no redundant constants, then the above proof is of the form $t \to^*_{P \backslash R_i^s} c$. If $l \to d \in R_i^s$ is the first rule used in the above proof that has a constant as a right-hand side, then we can choose l as the representative for d, and hence selection is applicable. □

THEOREM 6. *If (K_∞, R_∞) is the final state of a maximal derivation starting from state (K, R), where R is a left reduced fully flattened AC congruence closure such that for every constant c in K_0, there exists a term t in $\mathcal{T}(\Sigma)$ such that $t \to^*_{D_0/C_0} c$, then* (i) *$K_\infty = \emptyset$,* (ii) *$\to_{P \backslash R_\infty^s}$ is ground convergent on all fully flattened terms over Σ, and* (iii) *the equivalence over flattened $\mathcal{T}(\Sigma)$ terms defined by this relation is the same as the equational theory induced by $R \cup AC$ over flattened $\mathcal{T}(\Sigma)$ terms.*

Proof. Statement (i) is a consequence of Lemma 11. It follows from Lemma 9(c) and Lemma 10(c) that $\to_{P \backslash R_\infty^s}$ is terminating. Let s, t be fully flattened terms over $\mathcal{T}(\Sigma)$ such that $s \leftrightarrow^*_{P \cup R_\infty^s} t$. From Lemma 9(b) and Lemma 10(b), it follows that $s \leftrightarrow^*_{AC \cup R} t$. This, in turn, implies that $s \to^*_{AC \backslash R^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC \backslash R^e} t$, and hence, by projecting this proof onto fully flattened terms (normalize each term in the proof by F), we obtain a proof $s \to^*_{P \backslash R^s} \circ \leftrightarrow^*_P \circ \leftarrow^*_{P \backslash R^s} t$, as R is assumed to be fully flattened. From Lemma 9(a) and Lemma 10(a), this normal form proof can be projected onto a proof $s \to^*_{P \backslash R_\infty^s} \circ \leftrightarrow^*_P \circ \leftarrow^*_{P \backslash R_\infty^s} t$. This establishes claims (ii) and (iii). □

Note that in the special case when Σ_{AC} is empty, the notion of rewriting corresponds to the standard notion, and hence R_∞ is convergent in the standard sense by this theorem.

⋆ $\to_{D/C} = (\leftrightarrow^*_C \circ \to_D \circ \leftrightarrow^*_C)$.

⋆⋆ If $\Sigma_{AC} = \emptyset$, then this condition is satisfied by any abstract congruence closure.

‡ Note that if the nonextended form of an A-rule is a D-rule, it is included in the set D_0.

7. Conclusion

ABSTRACT CONGRUENCE CLOSURE

Kapur [18] considered the problem of casting Shostak's congruence closure [28] algorithm in the framework of ground completion on rewrite rules. Our work has been motivated by the goal of formalizing not just one but several congruence closure algorithms, so as to be able to better compare and analyze them.

We have suggested that, abstractly, congruence closure can be *defined* as a ground convergent system and that this definition does not restrict the applicability of congruence closure. We give strong bounds on the length of derivations used to construct an abstract congruence closure. This brings out a relationship between derivation lengths and term orderings used in the derivation. The rule-based abstract description of the logical aspects of the various published congruence closure algorithms leads to a better understanding of these methods. It explains the observed behavior of implementations and also allows one to identify weaknesses in specific algorithms.

The paper also illustrates the use of an extended signature as a formalism to model and subsequently reason about data structures like the term dags, which are based on the idea of structure sharing. This insight is more generally applicable to other algorithms as well [6].

EFFICIENT CONSTRUCTION OF GROUND CONVERGENT SYSTEMS

Graph-based congruence closure algorithms have also been used to construct a convergent set of ground rewrite rules in polynomial time by Snyder [29]. Plaisted et al. [25] gave a *direct* method, not based on using congruence closure, for completing a ground rewrite system in polynomial time. Hence our work completes the missing link, by showing that congruence closure is nothing but ground completion.

Snyder [29] uses a *particular implementation* of congruence closure because of which some postprocessing followed by a *second* run of congruence closure is required. We, on the other hand, work with abstract congruence closure and are free to choose any implementation. All the steps in the algorithm in [29] can be described by using our *construction of abstract congruence closure* steps, and the final output of Snyder's algorithm corresponds to an abstract congruence closure. The compression and selection rules for *translating back* in our work actually correspond to what Snyder calls *printing-out the reduced system,* and this is not included in the algorithm's time complexity of $O(n \log(n))$ as computed in [29]. Finally, the approach in [29] is to solve the problem "by abandoning rewriting techniques altogether and recasting the problem in graph theoretic terms." On the other hand, we stick to rewriting over extensions.

Plaisted and Sattler-Klein [25] show that ground term-rewriting systems can be completed in a polynomial number of rewriting steps by using an appropriate data

structure for terms and processing the rules in a certain way. Our work describes the construction of ground convergent systems using congruence closure as completion with *extensions*, followed by a translating back phase. Plaisted and Sattler-Klein prove a quadratic time complexity of their completion procedure.

AC CONGRUENCE CLOSURE

The fact that we can construct an AC congruence closure implies that the word problem for finitely presented ground AC-theories is decidable; see [20, 22], and [14]. We arrive at this result *without* assuming the existence of an AC-simplification ordering that is total on ground terms. The existence of such AC-simplification orderings was established in [22] but required a nontrivial proof.

Since we construct a convergent rewrite system, even the problem of determining whether two finitely presented ground AC-theories are equivalent is decidable. Since commutative semigroups are special kinds of AC-theories, where the signature consists of a single AC-symbol and a finite set of constants, these results carry over to this special case [21, 19].

Domenjoud and Klay present the idea of using variable abstraction to transform a set of equations over several AC-symbols into a set of equations in which each equation contains exactly one AC-symbol [14]. All equations containing the same AC-symbol are separated out and completed into a canonical rewriting system (modulo AC) by using the method proposed in [7]. However, the combination of ground AC-theories with other ground theories is done differently here. In [14], the ground theory (non-AC part) is handled by using ground completion (and a recursive path ordering during completion). We, on the other hand, use a congruence closure. The usefulness of our approach can also be seen from the simplicity of the correctness proof and the results we obtain for transforming a convergent system over an extended signature to one over the original signature.

The method for completing a finitely presented commutative semigroup (using what we call A-rules here) has been described in various forms in the literature, for example, [7].* It is essentially a specialization of Buchberger's algorithm for polynomial ideals to the case of binomial ideals (i.e., when the ideal is defined by polynomials consisting of exactly two monomials with coefficients $+1$ and -1).

The basic idea behind our construction of associative-commutative congruence closure is that we consider only certain ground instantiations of the nonground AC axioms. If we are interested in the $\mathcal{E}$-algebra presented by E (where $\mathcal{E}$ consists of only AC axioms for some function symbols in the signature Σ in our case, and E is a set of ground equations), then since $\mathcal{E}$ consists of nonground axioms, one needs to worry about what instantiations of these axioms to consider. For the

* Actually there is a subtle difference between the proposed method here and the various other algorithms for deciding the word problem for commutative semigroups, too. For example, working with rule extensions is not the same as working with rules on equivalence classes (under AC) of terms. Hence, in our method, we can apply certain optimizations as mentioned in Section 5.4.

case when $\mathcal{E}$ is a set of AC axioms, we show that we need to consider ground instances in which every variable is replaced by some subterm occurring in E. This observation can be generalized, and one can ask for what choices of $\mathcal{E}$ axioms does considering such restricted instantiations suffice to decide the word problem in $\mathcal{E}$-algebras. Evans [16, 17] gives a characterization in terms of *embeddability of partial $\mathcal{E}$-algebras*. Apart from commutative semigroups, this method works for lattices, groupoids, quasigroups, loops, and so forth.

Acknowledgments

We thank Deepak Kapur, Rakesh Verma, and the anonymous referees for their helpful comments.

References

1. Bachmair, L.: *Canonical Equational Proofs*, Birkhäuser, Boston, 1991.
2. Bachmair, L. and Dershowitz, N.: Completion for rewriting modulo a congruence, *Theoret. Comput. Sci.* **67**(2 & 3) (Oct. 1989), 173–201.
3. Bachmair, L. and Dershowitz, N.: Equational inference, canonical proofs, and proof orderings, *J. ACM* **41** (1994), 236–276.
4. Bachmair, L., Ramakrishnan, I., Tiwari, A. and Vigneron, L.: Congruence closure modulo Associativity-Commutativity, in H. Kirchner and C. Ringeissen (eds), *Frontiers of Combining Systems, Third International Workshop, FroCoS 2000*, Nancy, France, March 2000, Lecture Notes in Artificial Intelligence 1794, Springer, Berlin, 2000, pp. 245–259.
5. Bachmair, L. and Tiwari, A.: Abstract congruence closure and specializations, in D. McAllester (ed.), *Conference on Automated Deduction, CADE 2000*, Pittsburgh, PA, June 2000, Lecture Notes in Artificial Intelligence 1831, Springer, Berlin, 2000, pp. 64–78.
6. Bachmair, L. and Tiwari, A.: Congruence closure and syntactic unification, in C. Lynch and D. Narendran (eds), *14th International Workshop on Unification*, 2000.
7. Ballantyne, A. M. and Lankford, D. S.: New decision algorithms for finitely presented commutative semigroups, *Comp. Math. Appl.* **7** (1981), 159–165.
8. Becker, T. and Weispfenning, V.: *Gröbner Bases: A computational Approach to Commutative Algebra*, Springer-Verlag, Berlin, 1993.
9. Cardozo, E., Lipton, R. and Meyer, A.: Exponential space complete problems for petri nets and commutative semigroups, in *Proc. 8th Ann. ACM Symp on Theory of Computing*, 1976, pp. 50–54.
10. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.
11. Cyrluk, D., Lincoln, P. and Shankar, N.: On Shostak's decision procedure for combination of theories, in M. A. McRobbie and J. Slaney (eds), *Proceedings of the 13th Int. Conference on Automated Deduction*, Lecture Notes in Comput. Sci. 1104, Springer, Berlin, 1996, pp. 463–477.
12. Dershowitz, N. and Jouannaud, J. P.: Rewrite systems, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science (Vol. B: Formal Models and Semantics)*, North-Holland, Amsterdam, 1990.
13. Dershowitz, N. and Manna, Z.: Proving termination with multiset orderings, *Comm. ACM* **22**(8) (1979), 465–476.

14. Domenjoud, E. and Klay, F.: Shallow AC theories, in *Proceedings of the 2nd CCL Workshop*, La Escala, Spain, Sept. 1993.
15. Downey, P. J., Sethi, R. and Tarjan, R. E.: Variations on the common subexpressions problem, *J. ACM* **27**(4) (1980), 758–771.
16. Evans, T.: The word problem for abstract algebras, *J. London Math. Soc.* **26** (1951), 64–71.
17. Evans, T.: Word problems, *Bull. Amer. Math. Soc.* **84**(5) (1978), 789–802.
18. Kapur, D.: Shostak's congruence closure as completion, in H. Comon (ed.), *Rewriting Techniques and Applications, RTA 1997*, Sitges, Spain, July 1997, Lecture Notes in Comput. Sci. 1103, Springer, Berlin, pp. 23–37.
19. Koppenhagen, U. and Mayr, E. W.: An optimal algorithm for constructing the reduced Gröbner basis of binomial ideals, in Y. D. Lakshman (ed.), *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, 1996, pp. 55–62.
20. Marche, C.: On ground AC-completion, in R. V. Book (ed.), *4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Comput. Sci. 488, Springer, Berlin, 1991, pp. 411–422.
21. Mayr, E. W. and Meyer, A. R.: The complexity of the word problems for commutative semigroups and polynomial ideals, *Adv. in Math.* **46** (1982), 305–329.
22. Narendran, P. and Rusinowitch, M.: Any ground associative-commutative theory has a finite canonical system, in R. V. Book (ed.), *4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Comput. Sci. 488, Springer, Berlin, 1991, pp. 423–434.
23. Nelson, G. and Oppen, D.: Fast decision procedures based on congruence closure, *J. Assoc. Comput. Mach.* **27**(2) (Apr. 1980), 356–364.
24. Peterson, G. E. and Stickel, M. E.: Complete sets of reductions for some equational theories, *J. ACM* **28**(2) (Apr. 1981), 233–264.
25. Plaisted, D. and Sattler-Klein, A.: Proof lengths for equational completion, *Inform. and Comput.* **125** (1996), 154–170.
26. Rubio, A. and Nieuwenhuis, R.: A precedence-based total AC-compatible ordering, in C. Kirchner (ed.), *Proceedings of the 5 Intl. Conference on Rewriting Techniques and Applications*, Lecture Notes in Comput. Sci. 960, Springer, Berlin, 1993, pp. 374–388.
27. Sherman, D. J. and Magnier, N.: Factotum: Automatic and systematic sharing support for systems analyzers, in *Proc. TACAS*, Lecture Notes in Comput. Sci. 1384, 1998.
28. Shostak, R. E.: Deciding combinations of theories, *J. ACM* **31**(1) (1984), 1–12.
29. Snyder, W.: A fast algorithm for generating reduced ground rewriting systems from a set of ground equations, *J. Symbolic Comput.* **15**(7) (1993).
30. Tiwari, A.: Decision procedures in automated deduction, Ph.D. thesis, State University of New York at Stony Brook, New York, 2000.

Article de référence du Chapitre 3

A. Wasilewska and L. Vigneron.
Rough Algebras and Automated Deduction.
In L. Polkowski and A. Skowron, editors,
Rough Sets in Knowledge Discovery 1, pages 261-275.
Springer Verlag,
July 1998.

Rough Algebras and Automated Deduction

Anita Wasilewska[1] *and Laurent Vigneron*[2]

[1] Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, USA. E-mail: anita@cs.sunysb.edu
[2] Loria – Université Nancy 2, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France. E-mail: vigneron@loria.fr

Abstract: The notion of *rough equality* was introduced by Pawlak in [17]. It was extensively examined in [14], [4], [16], and the most recently in [2], [27], and [28]. The rough R5 and R4 algebras investigated here are particular cases of topological rough algebras introduced in [26]. We examine and discuss here some of their most interesting properties, their relationship with each other, and with the topological Boolean S4 and S5 algebras, which are algebraic models for modal logics S4 and S5, respectively. The presented properties were chosen out of over seven hundred theorems which were discovered and proved automatically by the theorem prover daTac (*Déduction Automatique dans des Théories Associatives et Commutatives*). This prover was developed at Loria, Nancy (France), by the second author.

1 Introduction

It is difficult to establish who was the first to use the algebraic methods. The investigations in logic of Boole himself led to the notion which we now call Boolean algebra, but one of the turning points in the algebraic study of logic was the introduction by Lindenbaum and in a slight different form by Tarski (in [24]) of the method of treating formulas, or equivalence classes of formulas as elements of an abstract algebra, called now the *Lindenbaum-Tarski algebra*.

In our work we use the algebraic logic techniques to link rough set theory with logic, abstract algebras and topology. In particular, we have shown in [26] that the notion of *rough equality* of sets leads, via logic and a Lindenbaum-Tarski like construction of an algebra of formulas, to a definition of new classes of algebras, called here *topological quasi-Boolean algebras* and *topological rough algebras*. These algebras are a non-classical (quasi-Boolean instead of Boolean) version of topological Boolean algebras. The topological Boolean algebras were introduced in [10], [11] under the name of closure algebras. They were first (algebraic) models for modal logics, as opposed to Kripke models invented some 20 years later [7].

This paper is a continuation of investigations of [2], [27], [28], and [26]. The organization of the paper is as follows.

In Section 2 we introduce some basic definitions and facts in order to make the paper self contained. We also give a short overview of the work by Banerjee and Chakraborty [2] and Wasilewska [26].
In Section 3 we introduce and investigate two of the topological Boolean algebras, named S4 and S5 because they are models for modal logics S4 and S5, respectively. The topological rough algebras considered here are called, accordingly, R4 and R5 algebras, where "R" stands for their rough equality origins.
In Section 4 we examine the properties and relationship between the R4 and R5 algebras and in Section 4 we discuss the relationship between the rough R4 and R5 algebras and their Boolean S4 and S5 counterparts.

All presented properties were discovered and proved automatically by a theorem prover daTac (*Déduction Automatique dans des Théories Associatives et Commutatives*). We give a short presentation of the deduction techniques implemented in daTac in Section 5.

2 Topological Boolean and Topological Rough Algebras

To make our paper self contained we first review in this section some basic definitions and facts.

Approximation space. Let U be a non-empty set called a universe, and let R be an equivalence relation on U. The triple $(U, \emptyset, R)$ is called an approximation space.

Lower, upper approximations. Let $(U, \emptyset, R)$ and $A \subset U$. We denote by $[u]$ an equivalence class of R. The sets

$$IA = \bigcup\{[u] \in A/R : \ [u] \subset A\},$$

$$CA = \bigcup\{[u] \in A/R : \ [u] \cap A \neq \emptyset\}$$

are called *lower* and *upper* approximations of A, respectively. We use here a topological notation for the lower and upper approximations because of their topological interpretation and future considerations.

Rough equality. Given an approximation space $(U, \emptyset, R)$ and any $A, B \subset U$. We say that the sets A and B are *roughly equal* and denote it by $A \sim_R B$ if and only if $IA = IB$ and $CA = CB$.

Boolean algebra. An abstract algebra $(A, 1, \cap, \cup, \neg)$ with unit element 1 is said to be a Boolean algebra if it is a distributive lattice and every element $a \in A$ has a complement $\neg a \in A$.

Orlowska has shown in [15] that propositional aspects of rough set theory are adequately captured by the modal system S5. In this case a Kripke model gives the approximation space $(A, \emptyset, R)$ in which the well formed formulas are interpreted as *rough sets*.

Following Orlowska result, Banerjee and Chakraborty introduced in [2] a new binary connective $\sim$ in S5, the intended interpretation of which is the notion of the *rough equality*. I.e, they added to the standard set $\{\cup, \cap, \rightarrow, \Leftrightarrow, \neg, \Box, \Diamond\}$ of propositional modal connectives a new binary connective $\sim$ defined in terms of

standard connectives as follows: for any formulas A, B (of the modal S5 language), we write $A \sim B$ for the formula $((\Box A \Leftrightarrow \Box B) \cap (\Diamond A \Leftrightarrow \Diamond B))$. In the next step they have used this new connective to define a construction similar to the construction of Lindenbaum-Tarski algebra on the set of all formulas of $S5$ with additional connective $\sim$. Before describing their construction leading to the definition of the rough algebra, we include below a description of a standard construction of a Lindenbaum-Tarski algebra for a given logic.

Lindenbaum-Tarski construction. Given a propositional logic with a set $\mathcal{F}$ of formulas. We define first two binary relations $\leq$ and $\approx$ in the algebra $\mathcal{F}$ of formulas of the given logic as follows. For any $A, B \in F$,

$A \leq B$ if and only if $\vdash (A \Rightarrow B)$, and
$A \approx B$ if and only if $\vdash (A \Rightarrow B)$ and $\vdash (B \Rightarrow A)$.

Then we use the set of axioms and rules of inference of the given logic to prove all facts listed below.

The relation $\leq$ is a quasi-ordering in F.
The relation $\approx$ is an equivalence relation in F. We denote the equivalence class containing a formula A by $[A]$.
The quasi-ordering $\leq$ on F induces an ordering relation on $F/\approx$ defined as follows: $[A] \leq [B]$ if and only if $A \leq B$.
The equivalence relation $\approx$ on F is a congruence with respect to all logical connectives.
The resulting algebra with universe $F/\approx$ is called a Lindenbaum-Tarski algebra.

Example 1. The Lindenbaum-Tarski algebra for *classical propositional logic* with the set of connectives $\{\cup, \cap, \Rightarrow, \neg\}$ is the following.

$$\mathcal{LT} = (F/\approx, \cup, \cap, \Rightarrow, \neg),$$

where the operations $\cup$, $\cap$, $\Rightarrow$ and $\neg$ are determined by the congruence relation $\approx$, i.e. $[A] \cup [B] = [(A \cup B)]$, $[A] \cap [B] = [(A \cap B)]$, $[A] \Rightarrow [B] = [(A \Rightarrow B)]$, $\neg[A] = [\neg A]$.

We prove, in this case (see [19]) that the Lindenbaum-Tarski algebra is a Boolean algebra with a unit element V. Moreover, for any formula A, $\vdash A$ if and only if $[A] = V$.

Example 2. The Lindenbaum-Tarski algebra for modal logic $S4$ or $S5$ with the set of connectives $\{\cup, \cap, \Rightarrow, \neg, \Box, \Diamond\}$ is the following.

$$\mathcal{LT} = (F/\approx, \cup, \cap, \Rightarrow, \neg, I, C),$$

where the operations $\cup$, $\cap$, $\Rightarrow$, $\neg$, I and C are determined by the congruence relation $\approx$, i.e. $[A] \cup [B] = [(A \cup B)]$, $[A] \cap [B] = [(A \cap B)]$, $[A] \Rightarrow [B] = [(A \Rightarrow B)]$, $\neg[A] = [\neg A]$, $IA = [\Box A]$, and $CA = [\Diamond A]$.

In the case of modal logic $S4$ the Lindenbaum-Tarski algebra (see [9], [10], [19]) is *a topological Boolean algebra* and in the case of $S5$ it is topological Boolean algebra such that every open element is closed and every closed element is open. Moreover, in both cases, for any formula A, $\vdash A$ if and only if $[A] = V$.

Banerjee, Chakraborty construction. We define a new binary relation $\approx$ on the set F of formulas of the modal S5 logic as follows. For any $A, B \in F$,

$A \approx B$ if and only if $A \sim B$, i.e.
$A \approx B$ if and only if $\vdash ((\Box A \Leftrightarrow \Box B) \cap (\Diamond A \Leftrightarrow \Diamond B))$.
We prove that the above relation $\approx$, corresponding to the notion of rough equality is *an equivalence relation* on the set F of formulas of $S5$.
We define a binary relation $\leq$ on $F/\approx$ as follows.
$[A] \leq [B]$ if and only if $\vdash ((\Box A \Rightarrow \Box B) \cap (\Diamond A \Rightarrow \Diamond B))$.
We prove that $\leq$ is an order relation on $F/\approx$ with the greatest element $1 = [A]$, for any formula A, such that $\vdash A$, and with the least element $0 = [B]$, such that $\vdash \neg B$.
We prove that $\approx$ *is a congruence relation* with respect to the logical connectives $\neg$, $\Box$, $\Diamond$, but *is not a congruence relation* with respect to $\Rightarrow$, $\cap$ and $\cup$.
We introduce two new operations $\sqcup$ and $\sqcap$ in $F/\approx$ as follows.
$[A] \sqcup [B] = [(A \cup B) \cap (A \cup \Box A \cup \Box B \cup \neg\Box(A \cup B))]$,
$[A] \sqcap [B] = [(A \cap B) \cup (A \cap \Diamond A \cap \Diamond B \cap \neg\Diamond(A \cap B))]$.
We call the resulting structure a *rough algebra (of formulas of logic S5)* or S5 rough algebras, for short.

The formal definition of the S5 rough algebra is hence the following.

S5 Rough algebra. An abstract algebra

$$\mathcal{R} = (F/\approx, \sqcup, \sqcap, \neg, I, C, 0, 1),$$

such that the operations $\sqcup$, $\sqcap$ are defined above and the operations $\neg$, I, C are induced, as in the Lindenbaum-Tarski algebra, by the relation $\approx$, i.e. $\neg[A] = [\neg A]$, $IA = [\Box A]$, and $CA = [\Diamond A]$, is called the S5 rough algebra.

In [1], many important properties of the S5 rough algebra were were proved. They were also reported in [2]. We cite here only those which are relevant to our future investigations.

P1 $(F/\approx, \leq, \sqcup, \sqcap, 0, 1)$ is a distributive lattice with 0 and 1.
P2 For any $[A], [B] \in F/\approx$, $\neg([A] \sqcup [B]) = (\neg[A] \sqcap \neg[B])$,
P3 For any $[A] \in F/\approx$, $\neg\neg[A] = [A]$.
P4 The rough algebra is not a Boolean algebra, i.e. there is a formula A of a modal logic $S5$, such that $\neg[A] \sqcap [A] \neq 0$ and $\neg[A] \sqcup [A] \neq 1$.
P5 For any $[A], [B] \in F/\approx$, $I([A] \sqcap [B]) = (I[A] \sqcap I[B])$, $I[A] \leq [A]$, $II[A] = I[A]$, $I1 = 1$, and $CI[A] = I[A]$, where $C[A] = \neg I\neg[A]$.

The above, and other properties of the rough algebra lead to some natural questions and observations.

By the property **P4**, the rough algebra's complement operation ($\neg$) is not a Boolean complement. Let's call it *a rough complement*. We can see that the rough complement is pretty close to the Boolean complement because the other de Morgan law $\neg([A] \sqcap [B]) = (\neg[A] \sqcap \neg[B])$ holds in the rough algebra, as well as the very Boolean laws $\neg 1 = 0$ and $\neg 1 = 0$. So what kind of a complement is the rough complement? The rough algebra *is not*, by **P4**, a Boolean algebra, so which kind of algebra is it? Has such an algebra been discovered and investigated before?

Observation. A complement operation with similar properties to the *rough complement* was introduced in 1935 by Moisil [12] and lead to a definition of a notion of *de Morgan Lattices*. De Morgan lattices are distributive lattices satisfying the conditions **P2** and **P3**. In 1957 Białynicki-Birula and Rasiowa have used the de Morgan lattices to introduce a notion of *a quasi-Boolean algebra*. They defined (in [3]) the quasi-Boolean algebras as de Morgan lattices with *unit* element 1. The above led, in [26] to the following definition and observation.

Definition 1 Topological quasi-Boolean algebra. An algebra $(A, \cap, \cup, \sim, 1, I)$ is called *a topological quasi-Boolean algebra* if $(A, \cup, \cap, \sim, 1)$ is a quasi-Boolean algebra and for any $a, b \in A$, $I(a \cap b) = Ia \cap Ib$, $Ia \cap a = Ia$, $IIa = Ia$, and $I1 = 1$.

The element Ia is called a *quasi-interior of a*. The element $\sim I \sim a$ is called *quasi-closure of a*. It allows us to define in A an unary operation C such that $Ca = \sim I \sim a$. We can hence represent the topological quasi-Boolean algebra as an algebra $(A, \cap, \cup, \sim, I, C, 0, 1)$ *similar* to the rough algebra $(F/\approx, \sqcup, \sqcap, \neg, I, C, 0, 1)$. From **P4** we immediately get the following.

Fact 2. *A rough algebra* $\mathcal{R} = (F/\approx, \sqcup, \sqcap, \neg, I, C, 0, 1)$ *is a* topological quasi-Boolean algebra.

Moreover, the property **P5** of the rough algebra tells us also that the operations I and C fulfill an additional property: for any $[A] \in F/\approx$, $CI[A] = I[A]$. This justifies the following definition.

Definition 3 Topological rough algebra. A topological quasi-Boolean algebra $(A, \cap, \cup, \sim, I, C, 0, 1)$ such that for any $a \in A$, $CIa = Ia$, is called a *topological rough algebra*.

3 R5 and R4 Algebras

The R5 and R4 algebras are particular cases of the topological rough algebras [26]. They are not purely mathematical invention. The S5 rough algebra developed and examined in [2] is an example the R5 algebra. The R4 algebra is a quasi-Boolean correspondent of the topological Boolean algebra.

The R5 algebra is a quasi-Boolean version of the topological Boolean algebras such that each open element is closed and each closed element is open.
We adopt here the following formal definition of R4 and R5 algebras.

Definition 4 R4 and R5 algebras. An abstract algebra $(A, 1, \cup, \cap, \sim, I, C)$ is called *a R4 algebra* if it is *a distributive lattice with unit element* 1 and additionally for all $a, b \in A$ the following equations are satisfied:

q1 $\sim\sim a = a$,
q2 $\sim(a \cup b) = \sim a \cap \sim b$,
t1 $I(a \cap b) = Ia \cap Ib$,
t3 $Ia \cap a = Ia$,
t4 $IIa = a$,
t5 $I1 = 1$,
t6 $Ca = \sim I \sim a$.

The algebra obtained from the R4 algebra by adding the following axiom:

CI $CIa = Ia$

is called a R5 algebra.

Axioms **q1**, **q2** say that R4 is a quasi-Boolean algebra, axioms **t1** – **t5** are the axioms of a topological space, **t6** defines the rough closure operation, and axiom **CI** says that every (roughly) open element is closed.

A natural set theoretical interpretation of the properties of the topological Boolean algebras is established by the representation theorem. For example, $a \cap Ia = Ia$ means that any set A contains its interior **I**A. The representation theorem provides an intuitive motivation for new properties and is an useful source of counter-examples.

The case of R4 and R5 algebras is more complicated and much less intuitive. While the operations $\cup$ and $\cap$ are represented as set theoretical union and intersection, the operation $\sim$ cannot be represented as a set theoretical complementation. The set theoretical interpretation of the rough complement depends on the mapping $g : A \longrightarrow A$ such that for all $a \in A$, $g(g(a)) = a$, called *involution*. The representation theorem for R4 or R5 algebras states that their properties have to hold in R4, R5 algebras (fields) of sets. For example, the set theoretical meaning of a R4 algebra property $a \cap \sim 1 = \sim 1$ is the following. Given any non empty set X, given any involution g on X, for any $A \subset X$, $A \cap (X - g(X)) = (X - g(X))$. This property is intuitively obvious, because any involution has to map the set X onto itself.

The set theoretical interpretation of the definition of the closure operation in R4 is the following: $CA = X - g(I(X - g(A)))$, for any involution g. One can see that it becomes less intuitive than the "normal" Boolean topological definition of closure as complement of the interior of the complement of the set. The situation becomes even more complex when we think about possible (or impossible) properties. For example, one of the simplest properties of R4 algebras proven by the prover (see Section 3.2) is $a \cap Ib \subseteq C(a \cap Ib) \cap b$. Its set theoretical R4 interpretation is that for any $A, B \subset X$ and for any involution g on X, $A \cap IB \subseteq (X - g(I(X - g(A \cap IB)))) \cap B$.

The above examples show that it is much more difficult to build an intuitive understanding of properties of R4 and R5 algebras, than it is in classical case

of topological Boolean algebras. It is not only difficult to prove new properties, it is also difficult to think how they should look like. We have hence used the theorem prover daTac as a tool to generate the R4, R5 algebras' properties (and their proofs). Moreover, we have also used it as a tool for a study of the relationship between both algebras. We strongly believe that such a study would be impossible without the use of the prover.

3.1 Automated Deduction of Properties

The properties of the R4 and R5 algebras presented here are chosen from over seven hundred which were discovered and proved automatically by the theorem prover daTac (*Déduction Automatique dans des Théories Associatives et Commutatives*) developed at Loria, Nancy (France). This software implements a new technique [21] (see Section 5 for a description of this technique) of automated deduction in first-order logic, in presence of associative and commutative operators. This technique combines an ordering strategy [5], a system of deduction rules based on resolution [20] and paramodulation [22] rules, and techniques for the deletion of redundant clauses.

daTac proposes either to prove properties by refutation, or by straightforward deduction from clauses. The refutation technique is a proof existence technique, i.e. we add the negation of a formula we want to prove to the set of initial formulas and we search for a contradiction. In this case, if the proof exists the prover would say: "yes, it is a theorem", i.e. the prover acts as a proof existence checker. Of course, the whole system is, as a classical predicate logic, semi-decidable. In the straightforward deduction, the prover acts as a deductive system, i.e. the end product is a set of properties with their formal proof. We have used here mainly the second technique.

Given the R4 algebra $(A, 1, \cup, \cap, \sim, I, C)$. As the first step we used the prover on a non-topological subset of its axioms, i.e. we used as its input only axioms **q1**, **q2** plus axioms for a distributive lattice. As the fact that the considered operators $\cup$ and $\cap$ are associative and commutative is embeded in the structure of the prover, we did not need to specify that portion of the distributive lattice axioms. The prover has immediately deduced the following properties:

$$\begin{aligned} a \cup a &= a, & a \cap a &= a, \\ a \cap \sim 1 &= \sim 1, & a \cup \sim 1 &= a, \\ \sim(a \cap b) &= \sim a \cup \sim b. \end{aligned}$$

We have added them to the set of the initial axioms of R4. We have also added the definition of the C operator, i.e. the following equation.

$$Ca = \sim I \sim a.$$

In the paper we use the above, extended version of the definition of the R4 algebra.

3.2 Properties Common to R4 and R5

The axioms of R4 are strictly included in the set of axioms for R5. Hence all properties we can prove in R4, we can prove in R5 and there are also *pure* R5 properties, i.e. the R5 properties which cannot be proved in R4. Of course in general setting the *pure* R5 properties are the set theoretical difference between all R5 properties and those which are *common* to R4 and R5. In general case there is a countably infinite number of all properties of the R4 and R5 algebras, so we never can find all pure R5 properties. In our case all sets of generated properties are finite and we present here a practical way of finding the *common* and *pure* properties. It is not straightforward because of the nature of the prover and we discuss the results in this section for the common properties and in Section 3.3 for the pure ones.

We have used the prover separately for R4 and R5 algebras. All executions have been arbitrary stopped after 5000 deduced clauses. When we have stopped the experiments, the prover had kept only 407 properties for R4 and 294 properties for R5, thanks to the techniques of simplification and deletion used (see Section 5). The answer to the question which properties from $407 + 294 = 701$ are common to both algebras is found in the following way: running a matching program for comparing the properties of R4 and R5, we have found 217 properties belonging to both sets. The 190 $(407 - 217)$ remaining properties in R4 have been found to be either deduced and simplified properties in R5, or properties to be deduced in R5 (not yet deduced because of our arbitrary stop of the execution).
In the 77 remaining properties in R5 $(294 - 217)$, we discovered that 27 would be derived in R4 later, since their proof uses only axioms of R4.

So, in the 701 properties, 651 are R4 (and also R5). We decomposed these properties into two categories. The first-one corresponds to intuitively obvious properties of topological spaces (or modal logics S4 and S5), the second category contains all other properties. The properties of the second category seem to be really not trivial even for topological spaces with normal set theoretical operations.

Remark. As $(A, \cap, \cup)$ is a lattice, we use symbol $\subseteq$ for the natural lattice ordering defined as follows.

$$a \subseteq b \text{ if and only if } a \cup b = b \text{ and } a \cap b = a.$$

The proved has derived immediately some intuitively obvious properties:

$$\begin{array}{rr} C1 = 1, & Ia \subseteq a, \\ C \sim 1 = \sim 1, & a \subseteq Ca, \\ I \sim 1 = \sim 1, & Ia \subseteq Ca, \\ C(a \cup b) = Ca \cup Cb, & Ia \cap b \subseteq a, \\ CCa = Ca, & a \cap b \subseteq Ca, \\ \sim Ca = I \sim a, & \sim Ia = C \sim a. \end{array}$$

We list below some, much less intuitive properties derived by the prover.

$$\begin{aligned} Ia \cap b \cap c &\subseteq C(Ia \cap b) \cap a, \\ a \cap Ib &\subseteq C(a \cap Ib) \cap b, \\ I(a \cup b) &\subseteq (I(Ca \cup b) \cap a) \cup (I(Ca \cup b) \cap b), \\ Ia \cap Ib \cap Ic &\subseteq a \cap b \cap C(a \cap b \cap c), \\ Ia \cap b &\subseteq I(a \cup c) \cap C(b \cap Ia), \\ a \cap Ib &\subseteq a \cap C(a \cap C(a \cap Ib)) \cap b, \\ b \cap Ia &\subseteq C(C(b \cap Ia) \cap a), \\ Ca \cap I(Ia \cup b) \cap c &\subseteq Ia \cup (Ca \cap b), \end{aligned}$$

$$(I(a \cup b) \cap Ic \cap C(c \cap a)) \cup (I(a \cup b) \cap Ic \cap C(c \cap b)) = Ic \cap I(a \cup b),$$
$$(I(a \cup c) \cap I(a \cup b) \cap C(c \cap b)) \cup (I(a \cup c) \cap I(a \cup b) \cap Ca) = I(a \cup b) \cap I(a \cup c).$$

3.3 Purely R5 Properties

After 5000 properties of R5 deduced, our prover has kept only 294 of them. We subtracted from them the common 217 properties with R4 and the 27 properties deduced by using only R4 axioms. The 50 properties left are *strong candidates* for being *purely* R5 properties, as their proofs used the additional R5 axiom $CIa = Ia$. This does not affirm yet that they are purely R5 properties, because we have not yet proved they do not have other R4 proof. However for some of them, we were able to prove that they are purely R5, using the following process: given a strong candidate P, we have shown that the specific R5 axiom $CIa = Ia$ is a consequence of R4 plus P. Here are some of these *purely R5 properties*:

$$\begin{aligned} ICa &= Ca, \\ a &\subseteq I(Ca \cup b), \\ I(Ca \cup Ib) &= Ca \cup Ib, \\ C(Ca \cap Ib) &= Ca \cap Ib. \end{aligned}$$

Observations. The prover has a tendency to generate larger and larger formulas. It hence tries to simplify them into smaller ones. But this is not always possible and the study of these complicated formulas is sometimes interesting. For example, we have found (by direct examination of the 50 R5 formulas) the following 2 variables property:

$$I(Ia \cup Ib) = Ia \cup Ib.$$

We have also found the 3 and 4 variables properties:

$$\begin{aligned} I(Ia \cup Ib \cup Ic) &= Ia \cup Ib \cup Ic, \\ I(Ia \cup Ib \cup Ic \cup Id) &= Ia \cup Ib \cup Ic \cup Id. \end{aligned}$$

They are in fact the 3 and 4 variables generalizations of the first 2 variables property. It is easy to see that they follow an obvious pattern listed below (where $m > 0$).

$$I(Ia_1 \cup \ldots \cup Ia_m) \;=\; Ia_1 \cup \ldots \cup Ia_m.$$

The proof by mathematical induction that this pattern is an R5 property is straightforward.
There are many other patterns. For example the following formulas

$$\begin{array}{cc} I(Ca \cup Cb) = Ca \cup Cb, & C(Ca \cap Cb) = Ca \cap Cb, \\ I(Ca \cup Ib) = Ca \cup Ib, & C(Ca \cap Ib) = Ca \cap Ib, \\ I(Ia \cup Ib) = Ia \cup Ib, & C(Ia \cap Ib) = Ia \cap Ib. \end{array}$$

together with their 3 and 4 variables generalizations can be described by the next two patterns ($n + m > 0$).

$$\begin{array}{c} I(Ca_1 \cup \ldots \cup Ca_n \cup Ib_1 \cup \ldots \cup Ib_m) = Ca_1 \cup \ldots \cup Ca_n \cup Ib_1 \cup \ldots \cup Ib_m, \\ C(Ca_1 \cap \ldots \cap Ca_n \cap Ib_1 \cap \ldots \cap Ib_m) = Ca_1 \cap \ldots \cap Ca_n \cap Ib_1 \cap \ldots \cap Ib_m. \end{array}$$

Below is a method of construction of a formal proof in R5 of the first of these two generalized properties. First we use the following derivation to show how it is possible to add a $m + 1^{\text{st}}$ Ia to the union of already obtained m I's. (In the first deduction, b_1 is chosen equal to $Ia_1 \cup Ia_2$.)

$$\frac{\underbrace{I(Ia_1 \cup Ia_2)} = Ia_1 \cup Ia_2 \qquad I(\underbrace{Ib_1} \cup Ib_2 \cup \ldots Ib_m) = Ib_1 \cup Ib_2 \cup \ldots Ib_m}{I(\overbrace{Ia_1 \cup Ia_2} \cup Ib_2 \cup \ldots Ib_m) = I(Ia_1 \cup Ia_2) \cup Ib_2 \cup \ldots Ib_m}$$

$$\frac{\underbrace{I(Ia_1 \cup Ia_2)} = Ia_1 \cup Ia_2 \qquad I(Ia_1 \cup Ia_2 \cup Ib_2 \cup \ldots Ib_m) = \underbrace{I(Ia_1 \cup Ia_2)} \cup Ib_2 \cup \ldots Ib_m}{I(Ia_1 \cup Ia_2 \cup Ib_2 \cup \ldots Ib_m) = \overbrace{Ia_1 \cup Ia_2} \cup Ib_2 \cup \ldots Ib_m}$$

Secondly, as the below derivation shows, we transform an Ia formula into a Ca using the property $ICa = Ca$ and a substitution of Ca for c_1.

$$\frac{\underbrace{ICa} = Ca \qquad I(Cb_1 \cup \ldots Cb_n \cup \underbrace{Ic_1} \cup Ic_2 \cup \ldots Ic_m) = Cb_1 \cup \ldots Cb_n \cup Ic_1 \cup Ic_2 \cup \ldots Ic_m}{I(Cb_1 \cup \ldots Cb_n \cup \overbrace{Ca} \cup Ic_2 \cup \ldots Ic_m) = Cb_1 \cup \ldots Cb_n \cup ICa \cup Ic_2 \cup \ldots Ic_m}$$

$$\frac{\underbrace{ICa} = Ca \qquad I(Cb_1 \cup \ldots Cb_n \cup Ca \cup Ic_2 \cup \ldots Ic_m) = Cb_1 \cup \ldots Cb_n \cup \underbrace{ICa} \cup Ic_2 \cup \ldots Ic_m}{I(Cb_1 \cup \ldots Cb_n \cup Ca \cup Ic_2 \cup \ldots Ic_m) = Cb_1 \cup \ldots Cb_n \cup \overbrace{Ca} \cup Ic_2 \cup \ldots Ic_m}$$

It is obvious from above that once we know how to add a I operator and how to transform it into a C, the general property mentioned earlier is R5.

4 Rough R4, R5 and Boolean S4, S5 Algebras

The rough R4 algebra is the quasi-Boolean version of the topological Boolean algebra. The topological Boolean algebras are algebraic models (see [10]) for the modal logic S4, where the interior I and closure C operations correspond to modal operators $\Box$ and $\Diamond$, respectively.

The topological Boolean algebras such that each open element is closed and each closed element is open form algebraic models for the modal logic S5. This justifies the following definition.

Definition 5 Boolean S4, S5 algebras. Any topological Boolean algebra $(A, 1, \cap, \cup, \neg, I)$ is called a Boolean S4 algebra.
An S4 algebra $(A, 1, \cap, \cup, \neg, I)$ such that for any $a \in A$, $CIa = Ia$, where $Ca = \neg I \neg a$, is called a Boolean S5 algebra.

It is obvious from the representation theorem for R5 algebras (see Section 2) that the principal Boolean property

$$a \cup \sim a = 1$$

does not hold neither in R5 nor in R4. It was proved in [18] that when we add the above property to the axioms of the quasi-Boolean algebra we obtain a Boolean algebra. The quasi complementation $\sim$ becomes in this case a classical set theoretical complementation.

This proves the following theorem.

Theorem 6. *A R4 (R5) algebra $(A, 1, \cup, \cap, \sim, I)$ with one of the following additional axioms (where 0 denotes ~ 1)*

$$a \cup \sim a = 1 \quad or \quad a \cap \sim a = 0$$

is called a S4 (S5) topological Boolean algebra.

We have added those two axioms to the set of axioms of R4 (R5, respectively) and let the prover run.

The prover has derived more than 300 properties for these topological S4, S5 Boolean algebras. Here are some we find interesting.

$$\begin{aligned} Ia \cup C \sim a = 1, \qquad & Ia \cap C \sim a = 0, \\ Ca \cup I \sim a = 1, \qquad & Ca \cap I \sim a = 0, \end{aligned}$$

$$\begin{aligned} (Ca \cap Cb) \cup (Ca \cap c) \cup (I \sim b \cap \sim c) \cup I \sim a &= 1, \\ (Ca \cap b) \cup Cc \cup (I \sim c \cap I \sim a) \cup (I \sim c \cap \sim b) &= 1, \\ (C(a \cap I \sim b) \cap Cb) \cup (C(a \cap I \sim b) \cap \sim a) \cup (a \cap I \sim b) &= C(a \cap I \sim b), \\ (Ia \cap C(a \cap C \sim a)) \cup (a \cap C \sim a) &= a \cap C(a \cap C \sim a), \\ (I \sim a \cap Ib) \cup (\sim a \cap b \cap Ca) \cup (\sim a \cap b \cap C \sim b) &= \sim a \cap b, \\ (Ca \cap \sim b) \cup (Ca \cap c) \cup (b \cap \sim c) \cup I \sim a &= 1, \\ (Ia \cap b) \cup Cc \cup (I \sim c \cap C \sim a) \cup (I \sim c \cap \sim b) &= 1. \end{aligned}$$

5 daTac — A Tool for Automated Deduction

The theorem prover daTac[3], for *Déduction Automatique dans des Théories Associatives et Commutatives*, has been developed at Loria, Nancy (France). This software is written in CAML Light (18000 lines), a language of the ML family. daTac can be used for theorem proving or for straightforward deduction. It manipulates formulas of first order logic with equality expressed in a clausal form $A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_m$. The deduction techniques implemented are detailed in [21]. We present here only a short overview of them.

5.1 Deduction Techniques

The prover is based on the first order logic with equality, hence the clauses use the equality predicate. But, we do not need to state all equality axioms. The symmetry, transitivity and functional reflexivity of the equality are simulated by a deduction rule called *Paramodulation* [22]. Its principle is to apply replacements in clauses. A paramodulation step in a positive literal is defined by

$$\frac{L_1 \Rightarrow l = r \vee R_1 \qquad L_2 \Rightarrow A[l'] \vee R_2}{(\ L_1 \wedge L_2 \Rightarrow A[r] \vee R_1 \vee R_2\ )\sigma}$$

where σ is a substitution (a mapping replacing variables by terms) unifying the term l and the subterm l' of A, i.e. $l\sigma$ is equal to $l'\sigma$. This last subterm is replaced by the right-hand side r of the equality $l = r$, and the substitution is applied on the whole deduced clause.
A similar paramodulation rule is defined for replacements in negative literals, i.e. literals on the left-hand side of the $\Rightarrow$ sign.

The reflexivity property of the equality predicate is simulated by a rule called *Reflexion*, defined by:

$$\frac{l = r \wedge L \Rightarrow R}{(\ L \Rightarrow R\ )\sigma}$$

where the substitution σ unifies the terms l and r.

The *Resolution* rule [20] permits to deal with the other predicate symbols than the equality:

$$\frac{A_1 \wedge L_1 \Rightarrow R_1 \qquad L_2 \Rightarrow A_2 \vee R_2}{(\ L_1 \wedge L_2 \Rightarrow R_1 \vee R_2\ )\sigma}$$

where σ unifies A_1 and A_2.

5.2 Strategies

In order to limit the number of possible deductions and to avoid useless deductions, the prover uses several strategies of deduction.

The first one is an *ordering strategy* [5]. It uses an ordering for comparing the terms and for orienting the equations. Hence, when a paramodulation step

[3] daTac Home Page (in the PROTHEO Group at the Loria):
`http://www.loria.fr/equipes/protheo/PROJECTS/DATAC/datac.html`

is applied from an equation $l = r$, it is checked that a term is never replaced by a bigger one, i.e. that the term r is not greater than the term l for this ordering. This technique is similar to the use of a rewriting rule $l \to r$.
The ordering is also used for selection of literals. For instance, it is possible to impose the condition that each deduction step has to use the maximal literal of a clause.

Another essential strategy is the *simplification*. We have defined some simplification rules whose purpose is to replace a clause by a simpler one, using term rewriting.
We have also defined some *deletion rules*. For instance, clauses which contain a positive equation $l = l$, or a same atom on the left-hand side and on the right-hand side of the implication sign $\Rightarrow$, are deleted. Another deletion technique is the *subsumption*, which can be schematized by: if a clause $L \Rightarrow R$ belongs to a set of clauses S, then any clause of the form $L\sigma \wedge L' \Rightarrow R\sigma \vee R'$ can be removed from S.

5.3 Deduction modulo *E*

The most important feature of daTac is the deduction modulo a set of equations E. The motivation for such deduction is the following.

The *commutativity property* of an operator f, $f(a, b) = f(b, a)$, cannot be oriented as a rewrite rule. This is a major problem when applying paramodulation steps.

Also, the *associativity property* of an operator f, $f(f(a, b), c) = f(a, f(b, c))$, has the disadvantage to provoke infinite sequences of paramodulation steps. For example, from an equation $f(d_1, d_2) = d_3$, we can derive

$$\begin{aligned} f(d_1, f(d_2, e_1)) &= f(d_3, e_1) \\ f(d_1, f(f(d_2, e_1), e_2)) &= f(f(d_3, e_1), e_2) \\ &\vdots \end{aligned}$$

Moreover, when these two properties (commutativity and associativity) are combined, it becomes very difficult to deduce useful clauses. For example, there are 1680 ways to write the term $f(a_1, f(a_2, f(a_3, f(a_4, a_5))))$, where f is associative and commutative. These 1680 terms are all semantically equivalent but none of them can be omitted, for the completeness of the deductions.

So, daTac is defined for being run modulo a set of equations E, composed of commutativity, and associativity and commutativity properties. These equations do not appear in the set of initial clauses. They are simulated by specific algorithms for equality checking, pattern matching and unification, in conjunction with some specially adapted paramodulation rules.

5.4 Advantages of daTac

daTac is an entirely automatic tool which, given a set of clauses, deduces new properties, consequences of these clauses. daTac is refutationally complete: if a

set of clauses is incoherent, it will find a contradiction. The only reason that may it fail in its search for a contradiction is the limit of the memory size of the computer.

The implemented techniques involve ordering and simplification strategies combined with a deduction system based on paramodulation and resolution, as mentioned earlier, but other important strategies are also available, as the superposition and basic strategies [25].

At the end of an execution, the user can ask for a lot of extra information. Especially the prover can present a proof of a derived property, or of the contradiction found. Many statistics are also available, such as the number of deduction steps, the number of simplification steps, and the number of deletions.

References

1. Banerjee, M.: A Cathegorical Approach to the Algebra and Logic of the indiscernible. Ph.D dissertation, Mathematics Department, University of Calcutta, India (1994).
2. Banerjee, M., Chakraborty, M. K.: Rough Algebra. Bull. Polish Acad. Sci. (Math.), 41(4):299–297 (1993).
3. Białynicki-Birula, A., Rasiowa, H.: On the representation of quasi-Boolean algebras. Bull. Ac. Pol. Sci. Cl. III, 5 (1957), pp. 259–261.
4. Bronikowski, Z.: Algebraic Structures of Rough Sets. In Ziarko, W., editor, Rough Sets, Fuzzy Sets and Knowledge Discovery, Workshops in Computing, pp. 242–247, Springer Verlag (1994).
5. Hsiang, J., Rusinowitch, M.: Proving Refutational Completeness of Theorem Proving Strategies: The Transfinite Semantic Tree Method. Journal of the ACM, 38(3):559–587 (1991).
6. Kelly, J.: General Topology. Van Nostrand (1955).
7. Kripke, S.: Semantics Analysis of Intuitionistic Logic. In Crossley, J.N., and Dummet, M.N.E., editors, Proc. of the Eight Logic Colloqium, Oxford 1963, pp. 92–130, North Holland Publishing C. (1965).
8. Kuratowski, C.: Topologie I, II. Warszawa (1958, 1961).
9. McKinsey, J. C. C.: A solution of the decision problem for the Lewis systems S.2 and S.4 with an application to topology. The Journal of Symbolic Logic 6, pp. 117–188 (1941).
10. McKinsey, J. C. C., Tarski, A.: The Algebra of Topology. Annales of Mathematics, 45:141–191 (1944).
11. McKinsey, J. C. C., Tarski, A.: On Closed Elements in Closure Algebras. Annales of Mathematics, 47:122–162 (1946).
12. Moisil, G. C.: Recherches sur l'algebre de la logique. Annales Sc. de l'Univerite de Jassy 22 (1935), pp. 1–117.
13. Nöbeling, G.: Grundlagen der analitishen Topologie. Berlin, Götingen, Heilderberg (1954).
14. Novotny, M., Pawlak, Z.: On Rough Equalities. Bull. Polish Acad. Sci. (Math.), 33(1-2):99–104 (1985).
15. Orłowska, E.: Semantics of vague concepts. In Dorn, G., Weingartner, P., editors, Foundations of Logic and Linguistics, Selected Contributions to the 7th International Congress of Logic, Methodology and Philosophy of Science, Saltzburg, Plenum Press, pp. 465–482 (1983).

16. Pagliani, P.: A Pure Logic-Algebraic Analysis of Rough Top and Rough Bottom Equalities. In Ziarko, W., editor, Rough Sets, Fuzzy Sets and Knowledge Discovery, Workshops in Computing, pp. 227–236, Springer Verlag (1994).
17. Pawlak, Z.: Rough Sets. Int. J. Comp. Inf. Sci., 11:341–356 (1982).
18. Rasiowa, H.: An Algebraic Approach to Non-Classical Logics. In Studies in Logic and the Foundations of Mathematics, vol. 78, North-Holland Publishing Company, Amsterdam, London - PWN, Warsaw (1974).
19. Rasiowa, H., Sikorski, R.: A proof of completeness theorem of Gódel. Fundamenta Mathematicae 37, pp. 193-200 (1950).
20. Robinson, J. A.: A Machine-oriented Logic Based on the Resolution Principle. Journal of the ACM, 12:23–41 (1965).
21. Rusinowitch, M., Vigneron, L.: Automated Deduction with Associative-Commutative Operators. Applicable Algebra in Engineering, Communication and Computing, 6(1):23–56 (1995).
22. Robinson, G. A., Wos, L. T.: Paramodulation and First-order Theorem Proving. In Meltzer, B., Mitchie, D., editors, Machine Intelligence, vol. 4, pp. 135–150, Edinburgh University Press (1969).
23. Stone, M. H.: Boolean Algebras and their Relation to Topology. In Proc. Nat. Ac. Sci., vol. 20, pp. 197–202 (1934).
24. Tarski, A.: Grundzüge des Syntemenkalküls. Ester Teil, Fundamenta Mathematicae 25, pp. 503–526 (1935).
25. Vigneron, L.: Associative-Commutative Deduction with Constraints. In Bundy, A., editor, Proceedings 12th International Conference on Automated Deduction, Nancy (France), vol. 814 of Lecture Notes in Artificial Intelligence, pp. 530–544, Springer-Verlag (1994).
26. Wasilewska, A.: Topological Rough Algebras. In Lin, T. Y., editor, Rough Sets and Database Mining, Kluwer (to appear).
27. Wasilewska, A., Banerjee, M.: Rough Sets and Topological Quasi-Boolean Algebras. In ACM CSC'95 23rd Annual Workshop on Rough Sets and Database Mining, Conference Proceedings, San Jose State University, San Jose, California, pp. 61–67 (1995).
28. Wasilewska, A., Banerjee, M.: Representation Theorem for Topological Rough Algebras. Bull. Polish Acad. Sci. (Math.) (to appear).

This article was processed using the LaTeX macro package with LMAMULT style

Article de référence du Chapitre 4

F. Jacquemard, M. Rusinowitch and L. Vigneron.
Compiling and Verifying Security Protocols.
In M. Parigot and A. Voronkov, editors,
Proceedings of 7th Conference on Logic for Programming and Automated Reasoning (LPAR 2000), St Gilles (Reunion, France),
volume 1955 of Lecture Notes in Computer Science, pages 131-160.
Springer-Verlag,
November 2000.

Compiling and Verifying Security Protocols

Florent Jacquemard[1], Michaël Rusinowitch[1], Laurent Vigneron[2]

[1] LORIA – INRIA Lorraine
Campus Scientifique, B.P. 239
54506 Vandœuvre-lès-Nancy Cedex, France
Florent.Jacquemard@loria.fr, Michael.Rusinowitch@loria.fr
http://www.loria.fr/~jacquema, http://www.loria.fr/~rusi
[2] LORIA – Université Nancy 2
Campus Scientifique, B.P. 239
54506 Vandœuvre-lès-Nancy Cedex, France
Laurent.Vigneron@loria.fr, http://www.loria.fr/~vigneron

Abstract. We propose a direct and fully automated translation from standard security protocol descriptions to rewrite rules. This compilation defines non-ambiguous operational semantics for protocols and intruder behavior: they are rewrite systems executed by applying a variant of ac-narrowing. The rewrite rules are processed by the theorem-prover daTac. Multiple instances of a protocol can be run simultaneously as well as a model of the intruder (among several possible). The existence of flaws in the protocol is revealed by the derivation of an inconsistency. Our implementation of the compiler CASRUL, together with the prover daTac, permitted us to derive security flaws in many classical cryptographic protocols.

Introduction

Many verification methods have been applied to the analysis of some particular cryptographic protocols [22, 5, 8, 24, 34]. Recently, tools have appeared [17, 13, 9] to automatise the tedious and error-prone process of translating protocol descriptions into low-level languages that can be handled by automated verification systems. In this research stream, we propose a concise algorithm for a direct and fully automated translation of any standard description of a protocol, into rewrite rules. For analysis purposes, the description may include security requirements and malicious agent (intruder) abilities. The asset of our compilation is that it defines non-ambiguous operational semantics for protocols (and intruders): they are rewrite rules executed on initial data by applying a variant of narrowing [15].

In a second part of our work, we have processed the obtained rewrite rules by the theorem-prover daTac [33] based on first order deduction modulo associativity and commutativity axioms (AC). Multiple instances of a protocol can be run simultaneously as well as a model of the intruder (among several possible). The existence of flaws in classical protocols (from [7]) has been revealed by the derivation of an inconsistency with our tool CASRUL.

In our semantics, the protocol is modelled by a set of transition rules applied on a multiset of objects representing a global state. The global state contains both sent messages and expected ones, as well as every piece of information collected by the intruder. Counters (incremented by narrowing) are used for dynamic generation of nonces (random numbers) and therefore ensure their freshness. The expected messages are automatically generated from the standard protocol description and describes concisely the actions to be taken by an agent when receiving a message. Hence, there is no need to specify manually these actions with special constructs in the protocol description. The verification that a received message corresponds to what was expected is performed by unification between a sent message and an expected one. When there is a unifier, then a transition rule can be fired: the next message in the protocol is composed and sent, and the next expected one is built too. The message to be sent is composed from the previously received ones by simple projections, decryption, encryption and pairing operations. This is made explicit with our formalism. The information available to an intruder is also floating in the messages pool, and used for constructing faked messages, by ac-narrowing too. The intruder-specific rewrite rules are built by the compiler according to abilities of the intruder (for diverting and sending messages) given with the protocol description.

It is possible to specify several systems (in the sense of [17]) running a protocol concurrently. Our compiler generates then a corresponding initial state. Finally, the existence of a security flaw can be detected by the reachability of a specific critical state. One critical state is defined for each security property given in the protocol description by mean of a pattern independent from the protocol.

We believe that a strong advantage of our method is that it is not ad-hoc: the translation is working without user interaction for a wide class of protocols and therefore does not run the risk to be biased towards the detection of a known flaw. To our knowledge, only two systems share this advantage, namely Casper [17] and CAPSL [21]. Therefore, we shall limit our comparison to these works.

Casper is a compiler from protocol specification to process algebra (CSP). The approach is oriented towards finite-state verification by model-checking with FDR [28]. We use almost the same syntax as Casper for protocols description. However, our verification techniques, based on theorem proving methods, will handle infinite states models. This permits to relax many of the strong assumptions for bounding information (to get a finite number of states) in model checking. Especially, our counters technique based on narrowing ensures directly that all randomly generated nonces are pairwise different. This guarantees the freshness of information over sessions. Our approach is based on analysing *infinite* traces by refutational theorem-proving and it captures automatically the traces corresponding to attacks. Note that a recent interesting work by D.Basin [4] proposes a lazy mechanism for the automated analysis of infinite traces.

CAPSL [21] is a specification language for authentication protocols in the flavour of Casper's input. There exists a compiler [9] from CAPSL to an in-

termediate formalism CIL which may be converted to an input for automated verification tools such as Maude, PVS, NRL [20]. The rewrite rules produced by our compilation is also an intermediate language, which has the advantage to be an idiom understood by many automatic deduction systems. In our case we have a single rule for every protocol message exchange, as opposite to CIL which has two rules. For this reason, we feel that our model is closer to Dolev and Yao original model of protocols [11] than other rewrite models are.

As a back-end system, the advantage of daTac over Maude is that ac-unification is built-in. In [8] it was necessary to program an ad-hoc narrowing algorithm in Maude in order to find flaws in protocols such as Needham-Schroeder Public Key.

We should also mention the works by C. Meadows [19] who was the first to apply narrowing to protocol analysis. Her narrowing rules were however restricted to symbolic encryption equations.

The paper is organised as follows. In Section 1, we describe the syntax for specifying a protocol $\mathcal{P}$ to be analysed and to give as input to the translator. Section 2 presents the algorithm implemented in the translator to produce, given $\mathcal{P}$, a set of rewrite rules $R(\mathcal{P})$. This set defines the actions performed by users following the protocol. The intruder won't follow the rules of the protocol, but will rather use various skill to abuse other users. His behaviour is defined by a rewrite system $\mathcal{I}$ given in Section 3. The execution of $\mathcal{P}$ in presence of an intruder may be simulated by applying narrowing with the rules of $R(\mathcal{P}) \cup \mathcal{I}$ on some initial term. Therefore, this defines an operational semantics for security protocols (Section 4). In Section 5, we show how flaws of $\mathcal{P}$ can be detected by pattern matching on execution traces, and Section 6 describes the deduction techniques underlying the theorem prover daTac and some experiments performed with this system. For additional informations the interested reader may refer to http://www.loria.fr/equipes/protheo/SOFTWARES/CASRUL/.

We assume that the reader is familiar with basic notions of cryptography and security protocols (public and symmetric key cryptography, hash functions) [30], and of term rewriting [10].

1 Input Syntax

We present in this section a precise syntax for the description of security protocols. It is very close to the syntax of CAPSL [21] or Casper [17] though it differs on some points – for instance, on those in Casper which concern CSP. The specification of a protocol $\mathcal{P}$ comes in seven parts (see Example 1, Figure 1). Three concern the protocol itself and the others describe an instance of the protocol (for a simulation).

1.1 Identifiers declarations

The identifiers used in the description of a protocol $\mathcal{P}$ have to be declared to belong to one of the following types: `user` (principal name), `public_key`,

`symmetric_key`, `table`, `function`, `number`. The type `number` is an abstraction for any kind of data (numeric, text or record ...) not belonging to one of the other types (`user`, `key` etc). An identifier T of type `table` is a one entry array, which associates public keys to users names ($T[D]$ is a public key of D). Therefore, public keys may be declared alone or by mean of an association table. An identifier F of type `function` is a one-way (hash) function. This means that one cannot retrieve X from the digest $F(X)$.

The unary postfix function symbol $\_^{-1}$ is used to represent the private key associated to some public key. For instance, in Figure 1, $T[D]^{-1}$ is the private key of D.

Among users, we shall distinguish an intruder I (it is not declared). It has been shown by G. Lowe [18] that it is equivalent to consider an arbitrary number of intruders which may communicate and one single intruder.

1.2 Messages

The core of the protocol description is a list of lines specifying the rules for sending messages,

$$(i.\ S_i \to R_i\ :\ M_i)_{1 \leq i \leq n}$$

For each $i \leq n$, the components i (step number), S_i, R_i (`users`, respectively sender and receiver of the message) and M_i (message) are ground terms over a signature $\mathbb{F}$ defined as follows. The declared `identifiers` as well as I are nullary function symbols of $\mathbb{F}$. The symbols of $\mathbb{F}$ with arity greater than 0 are $\_^{-1}$, $\_[\_]$ (for tables access), $\_(\_)$ (for one-way functions access), $\langle \_, \_ \rangle$ (pairing), $\{\_\}_\_$ (encryption). We assume that multiple arguments in $\langle \_, \ldots, \_ \rangle$ are right associated. We use the same notation for public key and symmetric key encryption (overloaded operator). Which function is really employed shall be determined unambiguously by the type of the key.

Example 1. Throughout the paper, we illustrate our method on two toy examples of protocols inspired by [36] and presented in Figure 1. These protocols describe messages exchanges in a home cable tv set made of a decoder D and a smartcard C. C is in charge of recording and checking subscription rights to channels of the user. In the first rule of the symmetric key version, the decoder D transmits his name together with an instruction *Ins* to the smartcard C. The instruction *Ins*, summarised in a `number`, may be of the form "(un)subscribe to channel n" or also "check subscription right for channel n". It is encrypted using a symmetric key K known by C and D. The smartcard C executes the instruction *Ins* and if everything is fine (*e.g.* the subscription rights are paid for channel n), he acknowledges to D, with a message containing C, D and the instruction *Ins* encrypted with K. In the public key version, the privates keys of D and C respectively are used for encryption instead of K.

protocol TV; # symmetric key
identifiers
C, D : user;
Ins : number;
K : symmetric_key;
messages
1. $D \to C : \langle D, \{Ins\}_K \rangle$
2. $C \to D : \langle C, D, \{Ins\}_K \rangle$
knowledge
$D : C, K$;
$C : K$;
session_instance :
$[D : tv, C : scard, K : key]$;
intruder : divert, impersonate;
intruder_knowledge : $scard$;
goal : correspondence_between $scard, tv$;

protocol TV; # public key
identifiers
C, D : user;
Ins : number;
T : table;
messages
1. $D \to C : \langle D, \{Ins\}_{T[D]^{-1}} \rangle$
2. $C \to D : \langle C, \{Ins\}_{T[C]^{-1}} \rangle$
knowledge
$D : C, T, T[D]^{-1}$;
$C : T, T[C]^{-1}$;
session_instance :
$[D : tv, C : scard, T : key]$;
intruder : eaves_dropping;
intruder_knowledge : key;
goal : secrecy_of Ins;

Fig. 1. Cable TV toy examples

1.3 Knowledge

At the beginning of a protocol execution, each principal needs some initial knowledge to compose his messages.

The field following `knowledge` associates to each `user` a list of terms of $\mathcal{T}(\mathbb{F})$ describing all the data (names, keys, function *etc*) he knows before the protocol starts. We assume that the own name of every user is always implicitly included in his initial knowledge. The intruder's name I may also figure here. In some cases indeed, the intruder's name is known by other (naïve) principals, who shall start to communicate with him because they ignore his bad intentions.

Example 2. In Example 1, D needs the name of the smartcard C to start communication. In the symmetric key version, both C and D know the shared key K. In the public key version, they both know the `table` T. It means that whenever D knows C's name, he can retrieve and use his public key $T[C]$, and conversely. Note that the `number` Ins is not declared in D's knowledge. This value may indeed vary from one protocol execution to one another, because it is created by D at the beginning of a protocol execution. The identifier Ins is therefore called a *fresh* number, or *nonce* (for oNly once), as opposite to persistent identifiers like C, D or K.

Definition 1. *Identifiers which occur in a* `knowledge` *declaration* $U : \ldots$ *(including the user name* U*) are called* persistent. *Other identifiers are called* fresh.

The subset of $\mathbb{F}$ of fresh identifiers is denoted $\mathbb{F}_{fresh}$. The identifier $ID \in \mathbb{F}_{fresh}$ is said to be *fresh in* M_i, if ID occurs in M_i and does not occur in any M_j for $j < i$. We denote $fresh(M_i)$ the list of identifiers fresh in M_i (occurring in this order). We assume that if there is a public key $K \in fresh(M_i)$ then K^{-1} also occurs in $fresh(M_i)$ (right after K). Fresh identifiers are indeed instantiated by a principal

in every protocol session, for use in this session only, and disappear at the end of the session. This is typically the case of nonces. Moreover we assume that the same fresh value cannot be created in two different executions of a protocol. Symmetric keys may either be persistent or fresh.

1.4 Session instances

This field proposes some possible values to be assigned to the persistent identifiers (*e.g.* tv for D in Figure 1) and thus describes the different systems (in the sense of Casper [17]) for running the protocol. The different sessions can take place concurrently or sequentially an arbitrary number of times.

Example 3. In Figure 1, the field `session_instance` contains only one trivial declaration, where one value is assigned to each identifier. This means that we want a simulation where only one system is running the protocol (*i.e.* the number of concurrent sessions is one, and the number of sequential sessions is unbounded).

1.5 Intruder

The `intruder` field describes which strategies the intruder can use, among passive `eaves_dropping`, `divert`, `impersonate`. These strategies are described in Section 3. A blank line here means that we want a simulation of the protocol without intruder.

1.6 Intruder knowledge

The `intruder_knowledge` is a set of values introduced in `session_instance`, but not a set of identifiers (like `knowledge` of others principals).

1.7 Goal

This is the kind of flaw we want to detect. There are two families of goals, `correspondence_between` and `secrecy_of` (see Sections 5.4 and 5.3). The secrecy is related to one identifier which must be given in the declaration, and the correspondence is related to two users.

2 Protocol rules

We shall give a formal description of the possible executions of a given protocol in the formalism of normalised ac-narrowing. More precisely, we give an algorithm which translates a protocol description $\mathcal{P}$ in the above syntax into a set of rewrite rules $R(\mathcal{P})$.

We assume given a protocol $\mathcal{P}$, described by all the fields defined in Section 1, such that

$$R_i = S_{i+1} \text{ for } i = 0 \dots n-1$$

This hypothesis is not restrictive since we can add empty messages. For instance, we can replace

$$\begin{array}{rcl} i.\ A \to B : M & \text{by} & i.\ A \to B : M \\ i+1.\ C \to D : M' & & i+1.\ B \to C : \emptyset \\ & & i+2.\ C \to D : M' \end{array}$$

For technical convenience, we let $R_0 = S_1$ and assume that S_0, M_0 are defined and are two arbitrary new constants of $\mathbb{F}$.

As in the model of Dolev and Yao [11] the translation algorithm associates to each step $S_i \to R_i : M_i$ a rewrite rule $l_i \to r_i$. An additional rule $l_{n+1} \to r_{n+1}$ is also created. The left member l_i describes the tests performed by R_{i-1} after receiving the message M_{i-1} – R_{i-1} compares M_{i-1} (by unification) with a *pattern* describing what was expected. The right member r_i describes how $S_i = R_{i-1}$ composes and send the next message M_i, and what is the pattern of the next message expected. This representation makes explicit most of the actions performed during protocol execution (recording information, checking and composing messages), which are generally hidden in protocol description. How to build the message from the pieces has to be carefully (unambiguously) specified. The expected pattern has also to be described precisely.

Example 4. In the symmetric key version of the protocol described in Figure 1, the cipher $\{Ins\}_K$ in last field of message 2 may be composed in two ways: either directly by projection on second field of message 1, or by decryption of this projection (on second field of message 1), and re-encryption of the value *Ins* obtained, with key K. The first (shortest) case is chosen in our procedure.
The pattern expected by C for message 1 is $\langle C, x_1, \{x_2\}_K \rangle$, because C does not know D's name in advance, nor the number *Ins*. The pattern expected by D for message 2 is $\langle C, D, \{Ins\}_K \rangle$, because D wants to check that C has sent the right *Ins*.

2.1 Normalised ac-narrowing

Our operational semantics for protocols are based on narrowing [15]. To be more precise, each step of an execution of the protocol $\mathcal{P}$ is simulated by a narrowing step using $R(\mathcal{P})$. We recall that narrowing unifies the left-hand side of a rewrite rule with a target term and replaces it with the corresponding right-hand side, unlike standard rewriting which relies on *matching* left-hand sides.

Let $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denote the set of terms constructed from a (finite) set $\mathcal{F}$ of function symbols and a (countable) set $\mathcal{X}$ of variables. The set of ground terms $\mathcal{T}(\mathcal{F}, \emptyset)$ is denoted $\mathcal{T}(\mathcal{F})$. In our notations, every variable starts by the letter x. We use $u[t]_p$ to denote a term that has t as a subterm at position p. We use $u[\cdot]$ to denote the context in which t occurs in the term $u[t]_p$. By $u|_p$, we denote the *subterm* of u rooted at *position* p. A rewrite *rule* over a set of terms is an ordered pair (l, r) of terms and is written $l \to r$. A *rewrite system* $\mathcal{S}$ is a finite set of such rules. The rewrite relation $\to_{\mathcal{S}}$ can be extended to rewrite over congruence

classes defined by a set of equations AC, rather than terms. These constitute ac-rewrite systems. In the following the set AC will be $\{x.(y.z) = (x.y).z, x.y = y.x\}$ where $\_.\_$ is a special binary function used for representing multisets of messages. The congruence relation generated by the AC axioms will be denoted $=_{ac}$. For instance $e.h.g =_{ac} g.e.h$. A term s *ac-rewrites by* S to another term t, denoted $s \to_{\mathcal{S}} t$, if $s|_p =_{ac} l\sigma$ and $t = s[r\sigma]_p$, for some rule $l \to r$ in $\mathcal{S}$, position p in s, and substitution σ. When s cannot be rewritten by S in any way we say it is a *normal form* for S. We note $s \downarrow_{\mathcal{S}} t$, or $t = s \downarrow_{\mathcal{S}}$ if there is a finite sequence of rewritings $s \to_{\mathcal{S}} s_1 \to_{\mathcal{S}} \ldots \to_{\mathcal{S}} t$ and t is a *normal form* for $\mathcal{S}$.

In the following we shall consider two rewrite systems $\mathcal{R}$ and $\mathcal{S}$. The role of the system $\mathcal{S}$ is to keep the messages normalised (by rewriting), while $\mathcal{R}$ is used for narrowing. A term s *ac-narrows by* $\mathcal{R}, \mathcal{S}$ to another term t, denoted $s \leadsto_{\mathcal{R},\mathcal{S}} t$, if $i)$ s is a normal form for $\mathcal{S}$, and $ii)$ $s|_p\sigma =_{ac} l\sigma$ and $t = (s[r]_p)\sigma \downarrow_{\mathcal{S}}$, for some rule $l \to r$ in $\mathcal{R}$, position p in s, and substitution σ.

Example 5. Assume $\mathcal{R} = \{a(x).c(x) \to c(x)\}$ and $\mathcal{S} = \{c(x).c(x) \to 0\}$. Then $a(0).b(0).c(x) \leadsto_{\mathcal{R},\mathcal{S}} b(0).c(0)$.

2.2 Messages algebra

We shall use for the rewrite systems $\mathcal{R}$ and $\mathcal{S}$ a sorted signature $\mathcal{F}$ containing (among other symbols) all the non-nullary symbols of $\mathbb{F}$ of Section 1, and a variable set $\mathcal{X}$ which contains one variable x_t for each term $t \in \mathcal{T}(\mathbb{F})$.

Sorts. The sorts for $\mathcal{F}$ are: `user`, `intruder`, `iuser` = `user` $\cup$ `intruder`, `public_key`, `private_key`, `symmetric_key`, `table`, `function`, `number`. Additional sorts are `text`, a super-sort of all the above sorts, and `int`, `message` and `list_of`.

Signature. All the constants occurring in a declaration `session_instance` are constant symbols of $\mathcal{F}$ (with the same sort as the identifier in the declaration). The symbol I is the only constant of sort `intruder` in $\mathcal{F}$. The pairing function $\langle \_, \_ \rangle$ (profile `text` $\times$ `text` $\to$ `text`) and encryption functions $\{\_\}_\_$ (`text` $\times$ `public_key` $\to$ `text` or `text` $\times$ `private_key` $\to$ `text` or `text` $\times$ `symmetric_key` $\to$ `text`) are the same as in $\mathbb{F}$ (see Section 1.2), as well as the unary function $\_^{-1}$ (`public_key` $\to$ `private_key` or `private_key` $\to$ `public_key`) for private keys (see Section 1.1), and as the table functions $\_[\_]$ (`table` $\times$ `iuser` $\to$ `public_key`). We use a unary function symbol $nonce(\_)$: `int` $\to$ `number` for the fresh numbers, see Section 2.4. We shall use similar unary functions $K(\_)$ (`int` $\to$ `public_key`) and $SK(\_)$ (`int` $\to$ `symmetric_key`) for respectively public and symmetric fresh keys.

At last, the constant 0 (sort `int`) and unary successor function $s(\_)$ (`int` $\to$ `int`) will be used for integer (time) encoding. Some other constants $1, \ldots, k$ and $\underline{0}$, $\underline{1}$ $\ldots$ and some alternative successor functions $s_1(\_)$, $\ldots$,$s_k(\_)$ are also used. The number k is fixed according to the protocol $\mathcal{P}$ (see page 10).

From now on, x_t, x_{pu}, x_p, x_s, x_{ps}, x_u, x_{f} are variables of respective sorts `table`, `public_key`, `public_key` $\cup$ `private_key`, `symmetric_key`, `public_key` $\cup$

`private_key` $\cup$ `symmetric_key`, `user`, and `function`. K, SK and KA will be arbitrary terms of $\mathcal{T}(\mathbb{F})$ of resp. sorts `public_key`$\cup$`private_key`, `symmetric_key` and `public_key` $\cup$ `private_key` $\cup$ `symmetric_key`.

Rewrite system for normalisation. In order to specify the actions performed by the principals, $\mathcal{F}$ contains some destructors. The decryption function applies to a text encrypted with some key, in order to extract its content. It is denoted the same way as the encryption function $\{\_\}\_$. Compound messages can be broken into parts using projections $\pi_1(\_), \pi_2(\_)$. Hence the relations it introduces in the message algebra are:

$$\left\{\{x\}_{x_s}\right\}_{x_s} \to x \tag{1}$$

$$\left\{\{x\}_{x_{pu}}\right\}_{x_{pu}^{-1}} \to x \tag{2}$$

$$\left\{\{x\}_{x_{pu}^{-1}}\right\}_{x_{pu}} \to x \tag{3}$$

$$x^{-1^{-1}} \to x \tag{4}$$

$$\pi_1(\langle x_1, x_2\rangle) \to x_1 \tag{5}$$

$$\pi_2(\langle x_1, x_2\rangle) \to x_2 \tag{6}$$

The rule (4) does not correspond to a real implementation of the generation of private key from public key. However, it is just a technical convenience. The terminating rewrite system (1) – (6) is called S_0. It can be easily shown that S_o is convergent [10], hence every message t admits a unique normal form $t\downarrow_{S_0}$ for S_0.

We assume from now on that the protocol $\mathcal{P}$ is *normalised*, in the following sense.

Definition 2. *A protocol $\mathcal{P}$ is called normalised if all the message terms in the field* `messages` *are in normal form w.r.t. $\mathcal{S}_0$.*

Note that this hypothesis is not restrictive since any protocol $\mathcal{P}$ is equivalent to the normalised protocol $\mathcal{P}\downarrow_{\mathcal{S}_0}$.

2.3 Operators on messages

We define in this section some functions to be called during the construction of the system $\mathcal{R}(\mathcal{P})$ in Section 2.4.

Knowledge decomposition. We denote by $know(U, i)$ the information that a user U has memorised at the end of the step $S_i \to R_i : M_i$ of the protocol $\mathcal{P}$. This information augments incrementally with i:

- if U is the receiver R_i, then he records the received message M_i as well as the sender's (official) name S_i,
- if U is the sender S_i, then he records the fresh elements (nonces ...) he has created for composing M_i (and may use latter),

– in any other case, the knowledge of U remains unchanged.

The set $know(U,i)$ contains labelled terms $V : t \in \mathcal{T}(\mathbb{F}) \times \mathcal{T}(\mathcal{F},\mathcal{X})$. The label t keeps track of the operations to derive V from the knowledge of U at the end of step i, using decryption and projection operators. This term t will be used later for composing new messages.

The informations are not only memorized but also decomposed with the function $CL^{(7-11)}()$ which is the closure of a set of terms using the following four rules:

$$\text{infer } M : \{t\}_{t'} \text{ from } \{M\}_{SK} : t \text{ and } SK : t' \tag{7}$$
$$\text{infer } M : \{t\}_{t'} \text{ from } \{M\}_{K} : t \text{ and } K^{-1} : t' \tag{8}$$
$$\text{infer } M : \{t\}_{t'} \text{ from } \{M\}_{K^{-1}} : t \text{ and } K : t' \tag{9}$$
$$\text{infer } M_1 : \pi_1(t) \tag{10}$$
$$\text{and } M_2 : \pi_2(t) \text{ from } \langle M_1, M_2\rangle : t \tag{11}$$

The function $know()$ is defined by:

$$\begin{aligned}
know(U,0) &= CL^{(7-11)}\big(\{T_1 : x_{T_1}, \ldots, T_k : x_{T_k}\}\big)\\
&\quad \text{where } \texttt{knowledge}\ U : T_1, \ldots, T_k \text{ is a statement of } \mathcal{P}.\\
know(U,i+1) &= know(U,i) \quad \text{if } U \neq S_{i+1} \text{ and } U \neq R_{i+1}\\
know(R_{i+1},i+1) &= CL^{(7-11)}(know(U,i) \cup \{M_{i+1} : x_{M_{i+1}}, S_{i+1} : x_{S_{i+1}}\})\\
know(S_{i+1},i+1) &= CL^{(7-11)}(know(U,i) \cup \{N_1 : x_{N_1}, \ldots, N_k : x_{N_k}\})\\
&\quad \text{where } N_1, \ldots, N_k = fresh(M_{i+1})
\end{aligned}$$

Example 6. In the symmetric-key version of the Cable TV example (Figure 1), we have $Ins : \{\pi_2(x_M)\}_K \in know(C,1)$ where M is the first message and x_M gets instantiated during the execution of a protocol instance.

Message composition. We define now an operator $compose(U,M,i)$ which returns a receipt of $\mathcal{T}(\mathcal{F},\mathcal{X})$ for the user U for building M from the knowledge gained at the end of step i (hence, U's knowledge at the begining of step $i+1$). In that way, we formalise the basic operations performed by a sender when he composes the pieces of the message M_{i+1}. In rule (16) below, we assume that M is the k^{th} nonce created in the message M_{i+1}.

$$compose(U,M,i) = t \quad \text{if } M : t \in know(U,i) \tag{12}$$
$$compose\big(U,\langle M_1,M_2\rangle,i\big) = \big\langle compose(U,M_1,i), compose(U,M_2,i)\big\rangle \tag{13}$$
$$compose\big(U,\{M\}_{KA},i\big) = \big\{compose(U,M,i)\big\}_{compose(U,KA,i)} \tag{14}$$
$$compose\big(U,T[A],i\big) = compose(U,T,i)[compose(U,A,i)] \tag{15}$$
$$compose\big(U,M,i\big) = nonce(s_k(x_{\text{time}})) \tag{16}$$
$$compose(U,M,i) = \textbf{Fail} \quad \text{in every other case} \tag{17}$$

The cases of the $compose()$ definition are tried in the given order. Other orders are possible, and more studies are necessary to evaluate their influence on the behaviour of our system.

The construction in case (16) is similar when M is a fresh public key or a fresh symmetric key, with respective terms $K(s_k(x_{\text{time}}))$, and $SK(s_k(x_{\text{time}}))$.

Expected patterns. The term of $\mathcal{T}(\mathcal{F},\mathcal{X})$ returned by the following variant of $compose(U,M,i)$ is a filter used to check received messages by pattern matching. More precisely, the function $expect(U,M,i)$ defined below is called right after the message M_{i+1} has been sent by U (hence with $U = S_{i+1} = R_i$).

$$expect(U,M,i) = t \quad \text{if } M : t \in know(U,i) \tag{18}$$

$$expect\big(U,\langle M_1,M_2\rangle,i\big) = \big\langle expect(U,M_1,i),\, expect(U,M_2,i)\big\rangle \tag{19}$$

$$expect\big(U,\{M\}_K,i\big) = \big\{expect(U,M,i)\big\}_{compose(U,K^{-1},i)^{-1}} \tag{20}$$

$$expect\big(U,\{M\}_{K^{-1}},i\big) = \big\{expect(U,M,i)\big\}_{compose(U,K,i)^{-1}} \tag{21}$$

$$expect\big(U,\{M\}_{SK},i\big) = \big\{expect(U,M,i)\big\}_{compose(U,SK,i)} \tag{22}$$

$$expect\big(U,T[A],i\big) = expect(U,T,i)[expect(U,A,i)] \tag{23}$$

$$expect(U,M,i) = x_{U,M,i} \quad \text{in every other case} \tag{24}$$

Note that unless $compose()$, the $expect()$ function cannot fail. If the call to $compose()$ fails in one of the cases (20)–(22), then the case (24) will be applied.

Example 7. The pattern expected by C for message 1 (Figure 1, symmetric key version) is $expect(C,\langle D,\{Ins\}_K\rangle,1) = \langle x_{C,D,1},\{x_{C,Ins,1}\}_{x_K}\rangle$ because C does not know D's name in advance, nor the number Ins, but he knows K.

2.4 Narrowing rules for standard messages exchanges

The global state associated to a step of a protocol instance will be defined as the set of messages $m_1.m_2.\ldots.m_n$ sent and not yet read, union the set of expected messages $w_1.\ldots.w_m$.

A sent message is denoted by $m(i,s',s,r,t,c)$ where i is the protocol step when it is sent, s' is the real sender, s is the official sender, r is the receiver, t is the body of the message and c is a session counter (incremented at the end of each session).

$$m : \texttt{step} \times \texttt{iuser} \times \texttt{iuser} \times \texttt{iuser} \times \texttt{text} \times \texttt{int} \rightarrow \texttt{message}$$

Note that s and s' may differ since messages can be impersonated (the receiver r never knows the identity of the real sender s').

A message expected by a principal is signalled by a term $w(i,s,r,t,\ell)$ with similar meaning for the fields i, s, r, t, and c, and where ℓ is a list containing r's knowledge just before step i.

$$w : \texttt{step} \times \texttt{iuser} \times \texttt{user} \times \texttt{text} \times \texttt{list\_of text} \times \texttt{int} \rightarrow \texttt{message}$$

Nonces and freshness. We describe now a mechanism for the construction of fresh terms, in particular of nonces. This is an important aspect of our method. Indeed, it ensures freshness of the randomly generated nonces or keys over several executions of a protocol. The idea is the following: nonces admit as argument a counter that is incremented at each transition (this argument is therefore the *age* of the nonce). Hence if two nonces are emitted at different steps in an execution trace, their counters do not match. We introduce another term in the global state for representing the counter, with the new unary head symbol h. Each rewrite rule $l \to r$ is extended to $h(s(x_{\text{time}})).l \to h(x_{\text{time}}).r$ in order to update the counter. Note that the variable x_{time} occurs in the argument of $nonce()$ in case (16) of the definition of $compose()$.

Rules. The rules set $R(\mathcal{P})$ generated by our algorithm contains (for $i = 0..n$):

$$
\begin{array}{l}
h(s(x_{\text{time}})). \\
w\big(i, x_{S_i}, x_{R_i}, x_{M_i}, \ell know(R_i, i), xc\big). \\
m(i, x_r, x_{S_i}, x_{R_i}, x_{M_i}, c) \to \\
\quad h(x_{\text{time}}). \\
\quad m\big(i+1, x_{R_i}, x_{R_i}, compose(R_i, R_{i+1}, i), compose(R_i, M_{i+1}, i), c\big). \\
\quad w\big(k_i, compose(R_i, S_{k_i}, i), x_{R_i}, expect(R_i, M_{k_i}, i'), \ell know(R_i, i'), c'\big)
\end{array}
$$

where k_i is the next step when R_i expects a message (see definition below), and $\ell know(R_i, i)$, $\ell know(R_i, i')$ are lists of variables described below.

If $i = 0$, the term $m(i, \dots)$ is missing in left member, and $c = xc$.
If $1 \leq i \leq n$, then $c = xc'$ (another variable).
If $i = n$, the term $m(i, \dots)$ is missing in right member.
In every case $(0 \leq i \leq n)$,
  if $k_i > i$ then $i' = i + 1$ and $c' = xc$,
  if $k_i \leq i$ then $i' = 0$ and $c' = s(xc)$.

Note that the calls of $compose()$ may return **Fail**. In this case, the construction of $R(\mathcal{P})$ stops with failure.

After receiving message i (of content x_{M_i}) from x_r (apparently from x_{S_i}), x_{R_i} checks whether he received what he was expecting (by unification of the two instances of x_{M_i}), and then composes and sends message $i + 1$. The term returned by $compose(R_i, M_{i+1}, i)$ contains some variables in the list $\ell know(R_i, i)$. As soon as he is sending the message $i + 1$, x_{R_i} gets into a state where he is waiting for new messages. This will be expressed by deleting the term $w(i, \dots)$ (previously expected message) and generating the term $w(k_i, \dots)$ in the right-hand side (next expected message). Hence sending and receiving messages is not synchronous (see e.g. [5]).

The function $\ell know(U, i)$ associates to a user U and a (step) number $i \in \{0..n\}$ a term corresponding to a list of variables, used to refer to the knowledge

of U. Below, $\ell :: a$ denotes the appending of the element a at the end of a list ℓ.

$$\begin{aligned}
\ell know(U,0) &= \langle x_U, x_{T_1}, \ldots, x_{T_n} \rangle \\
&\quad \text{where } \texttt{knowledge}\ U : T_1, \ldots, T_n \text{ is a statement of } \mathcal{P}, \\
\ell know(U,i+1) &= \ell know(U,i) \text{ if } U \neq R_i \\
&= \ell know(U,i) :: x_{M_i} :: x_{S_i} :: n_1 :: \ldots :: n_k \text{ if } U = R_i \\
&\quad \text{where } fresh(M_i) = N_1, \ldots, N_k \\
&\quad \text{and } n_i = x_{N_i} \text{ if } N_i \text{ is of sort } \texttt{nonce} \text{ or } \texttt{symmetric_key}, \\
&\quad \text{and } n_i = x_{N_i} :: x_{N_i^{-1}} \text{ if } N_i \text{ is of sort } \texttt{public_key},
\end{aligned}$$

The algorithm also uses the integer k_i which is the next session step when R_i expects a message. If R_i is not supposed to receive another message in the current session then either he is the session initiator S_1 and k_i is reinitialized to 0, otherwise k_i is the first step in the next session where he should receive a message (and then $k_i < i$). Formally, k_i is defined for $i = 0$ to n as follows:

$k_i = \min\{j \mid j > i \text{ and } R_j = R_i\}$ if this set is not empty;
otherwise $k_i = \min\{j \mid j \leq i \text{ and } R_j = R_i\}$ (recall that $R_0 = S_1$ by hypothesis);

Example 8. In both protocols presented in Figure 1, one has $R_0 = D$, $R_1 = C$, $R_2 = D$, and therefore: $k_0 = 2$, $k_1 = 1$, $k_2 = 0$.

Lemma 1. *k is a bijection from $\{0, \ldots, n\}$ to $\{0, \ldots, n\}$.*

Example 9. The translator generates the following $R(\mathcal{P})$ for the symmetric key version of the protocol of Figure 1. For sake of readability, in this example and the following ones, the fresh variables are denoted x_i (where i is an integer) instead of the form of the case (24) in the definition of *expect*().

$$\begin{aligned}
&h(s(x_{\text{time}})).w\big(0, x_{S_0}, x_D, x_{M_0}, \langle x_D, x_C, x_K \rangle, xc\big) \rightarrow \\
&\quad h(x_{\text{time}}).m\big(1, x_D, x_D, x_C, \langle x_D, \{nonce(s_1(x_{\text{time}}))\}_{x_K}, xc\big). \\
&\qquad w\big(2, x_C, x_D, \langle x_C, x_D, \{nonce(s_1(x_{\text{time}}))\}_{x_K} \rangle, \\
&\qquad\quad \langle x_D, x_C, x_K, x_{M_0}, x_{S_0}, nonce(s_1(x_{\text{time}})) \rangle, xc\big) \qquad (\text{tvs}_1) \\
&h(s(x_{\text{time}})).w\big(1, x_D, x_C, x_{M_1}, \langle x_C, x_K \rangle, xc\big). \\
&\qquad m\big(1, x_r, x_D, x_C, x_{M_1}, xc'\big) \rightarrow \\
&\quad h(x_{\text{time}}).m\big(2, x_C, x_C, \pi_1(x_{M_1}), \langle x_C, \pi_1(x_{M_1}), \pi_2(x_{M_1}) \rangle, xc'\big). \\
&\qquad w\big(1, x_D, x_C, \langle x_D, \{x_1\}_{x_K} \rangle, \langle x_C, x_K \rangle, s(xc)\big) \qquad (\text{tvs}_2) \\
&h(s(x_{\text{time}})).w\big(2, x_C, x_D, x_{M_2}, \langle x_D, x_C, x_K, x_{M_0}, x_{S_0}, x_{Ins} \rangle, xc\big). \\
&\qquad m\big(2, x_r, x_C, x_D, x_{M_2}, xc'\big) \rightarrow \\
&\quad h(x_{\text{time}}).w\big(0, x_{S_0}, x_D, x_{M_0}, \langle x_D, x_C, x_K \rangle, s(xc)\big) \qquad (\text{tvs}_3)
\end{aligned}$$

3 Intruder rules

The main difference between the behaviour of a honest principal and the intruder I is that the latter is not forced to follow the protocol, but can send messages

arbitrarily. Therefore, there will be no $w()$ terms for I. In order to build messages, the intruder stores some information in the global state with terms of the form $i()$, where i is a new unary function symbol. The rewriting rules corresponding to the various intruder's techniques are detailed below.
The intruder can record the information aimed at him, (25). If `divert` is selected in the field `intruder`, the message is removed from the current state (26), but not if `eaves_dropping` is selected (27).

$$m(x_i, x_u, x_u, I, x, xc) \to i(x).i(x_u) \tag{25}$$
$$m(x_i, x_u, x_u, x'_u, x, xc) \to i(x).i(x_u).i(x'_u) \tag{26}$$
$$m(x_i, x_u, x_u, x'_u, x, xc) \to m(x_i, x_u, x_u, x'_u, x, xc).i(x).i(x_u).i(x'_u) \tag{27}$$

After collecting information, I can decompose it into smaller $i()$ terms. Note that the information which is decomposed (*e.g.* $\langle x_1, x_2\rangle$) is not lost during the operation.

$$i\big(\langle x_1, x_2\rangle\big) \to i\big(\langle x_1, x_2\rangle\big).i(x_1).i(x_2) \tag{28}$$
$$i\big(\{x_1\}_{x_p}\big).i\big(x_p^{-1}\big) \to i\big(\{x_1\}_{x_p}\big).i\big(x_p^{-1}\big).i(x_1) \tag{29}$$
$$i\big(\{x_1\}_{x_s}\big).i\big(x_s\big) \to i\big(\{x_1\}_{x_s}\big).i\big(x_s\big).i(x_1) \tag{30}$$
$$i\big(\{x_1\}_{x_p^{-1}}\big).i\big(x_p\big) \to i\big(\{x_1\}_{x_p^{-1}}\big).i\big(x_p\big).i(x_1) \tag{31}$$

I is then able to reconstruct terms as he wishes.

$$i(x_1).i(x_2) \to i(x_1).i(x_2).i\big(\langle x_1, x_2\rangle\big) \tag{32}$$
$$i(x_1).i(x_{ps}) \to i(x_1).i(x_{ps}).i\big(\{x_1\}_{x_{ps}}\big) \tag{33}$$
$$i(x_{\mathrm{f}}).i(x) \to i(x_{\mathrm{f}}).i(x).i\big(x_{\mathrm{f}}(x)\big) \tag{34}$$
$$i(x_t).i(x_u) \to i(x_t).i(x_u).i\big(x_t[x_u]\big) \tag{35}$$

I can send arbitrary messages in his own name,

$$i(x).i(x_u) \to i(x).i(x_u).m(j, I, I, x_u, x, \underline{0}) \quad j \leq n \tag{36}$$

If moreover `impersonate` is selected, then I can fake others identity in sent messages.

$$i(x).i(x_u).i(x'_u) \to i(x).i(x_u).i(x'_u).m(j, I, x_u, x'_u, x, \underline{0}) \quad j \leq n \tag{37}$$

Note that the above intruder rules are independent from the protocol $\mathcal{P}$ in consideration. The rewrite system of the intruder (25)–(37) is denoted $\mathcal{I}$.

4 Operational semantics

4.1 Initial state

After the definition of rules of $R(\mathcal{P})$ and $\mathcal{I}$, the presentation of an operational "state/transition" semantics of protocol executions is completed here by the

definition of an initial state $t_{init}(\mathcal{P})$. This state is a term of the form $w(\dots)$ containing the patterns of the first messages expected by the principals, and their initial knowledge, for every session instance.
We add to the initial state term a set of initial knowledge for the intruder I. More precisely, we let $t_{init}(\mathcal{P}) := t_{init}(\mathcal{P}).i(v_1)\dots i(v_n)$ if the field `intruder_knowledge` : $v_1, \dots, v_n$; is declared in $\mathcal{P}$.

Example 10. The initial state for the protocol of Figure 1 (symmetric key version) is: $t_{init}(\mathcal{P}) := h(x_{\text{time}}).w(0, x_1, tv, x_2, \langle tv, scard, key\rangle, \underline{1})$
$.w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}).i(scard)$

4.2 Protocol executions

Definition 3. *Given a ground term t_0 and rewrite systems R, S the set of executions $EXEC(t_0, R, S)$ is the set of maximal derivations $t_0 \rightsquigarrow_{R,S} t_1 \rightsquigarrow_{R,S} \dots$*

Maximality is understood w.r.t. the prefix ordering on sequences. The *normal executions* of protocol $\mathcal{P}$ are the elements of the set

$$EXEC_n(\mathcal{P}) := EXEC\big(t_{\text{init}}(\mathcal{P}), R(\mathcal{P}), \mathcal{S}_0\big)$$

Executions in the presence of an intruder are the ones in

$$EXEC_i(\mathcal{P}) := EXEC\big(t_{\text{init}}(\mathcal{P}), R(\mathcal{P}) \cup \mathcal{I}, S_0\big)$$

4.3 Executability

The following Theorem 1 states that if the construction of $R(\mathcal{P})$ does not fail, then normal executions will not fail (the protocol can always run and restart without deadlock).

Theorem 1. *If $\mathcal{P}$ is normalised, the field* `session_instance` *of $\mathcal{P}$ contains only one declaration, and the construction of $R(\mathcal{P})$ does not fail on $\mathcal{P}$, then every derivation in $EXEC_n(\mathcal{P})$ is infinite.*

Theorem 1 is not true if the field `session_instance` of $\mathcal{P}$ contains at least two declarations, as explained in the next section. Concurrent executions may interfere and enter a deadlock state.

4.4 Approximations for intruder rules

Due to the intruder rules of Section 3 the search space is too large. In particular, the application of rules (32)–(33) is obviously non-terminating. In our experiences, we have used restricted intruder rules for message generation.

Intruder rules guided by expected messages. The first idea is to change rules (36)–(37) so that I sends a faked message $m(i, I, x_u, x'_u, x)$ only if there exists a term of the form $w(i, x_u, x'_u, x, x_\ell, xc)$ in the global state. More precisely, we replace (36), (37) in $\mathcal{I}$ by, respectively,

$$i(x).i(x_u).w(j, I, x_u, x, x_\ell, xc) \rightarrow$$
$$i(x).i(x_u).w(j, I, x_u, x, x_\ell, xc).m(j, I, I, x_u, x, \underline{0}) \quad \text{where } j \leq n \qquad (36')$$

$$i(x).i(x_u).i(x'_u).w(j, x_u, x'_u, x, x_\ell, xc) \rightarrow$$
$$i(x).i(x_u).i(x'_u).w(j, x_u, x'_u, x, x_\ell, xc).m(j, I, x_u, x'_u, x, \underline{0}) \text{ where } j \leq n \qquad (37')$$

The obtained rewrite system is called $\mathcal{I}_w$.

This approximation is complete: every attack in $EXEC_i(\mathcal{P})$ exists also in the trace generated by the modified system, indeed, the messages in a trace of $EXEC_i(\mathcal{P})$ and not in $EXEC(t_{\text{init}}(\mathcal{P}), R(\mathcal{P}) \cup \mathcal{I}_w, S_0)$ would be rejected by the receiver as non-expected or ill-formed messages. Similar observations are reported independantly in [32]. Therefore, there is no limitation for detecting attacks with this simplification (this strategy prunes only useless branches) but it is still inefficient.

Rules guided approximation. The above strategy is improved by deleting rules (32)–(35) and replacing each rules of (36'), (37') new rules (several for each protocol message), such that a sent message has the form $m(i, I, x_u, x'_u, t, \underline{0})$, where, roughly speaking, t follows the pattern M_i where missing parts are filled with some knowledge of I. Formally, we define a non-deterministic unary operator $* : \mathcal{T}(\mathbb{F}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$.

$$\langle M_1, M_2 \rangle^* = \langle M_1^*, M_2^* \rangle \qquad (38)$$
$$\{M\}_K^* = \{M^*\}_{K^*} \mid x_{\{M\}_K} \qquad (39)$$
$$F(M)^* = x_F(M^*) \mid x_{F(M)} \qquad (40)$$
$$T[A]^* = x_T[x_A] \mid x_{T[A]} \qquad (41)$$
$$ID^* = x_{ID} \quad \text{if } ID \text{ is a nullary function symbol of } \mathbb{F} \qquad (42)$$

Given $T \in \mathcal{T}(\mathbb{F})$ we denote $skel(T)$ the set of possible terms for T^*. Then, we replace (36'), (37') in $\mathcal{I}$ by,
for each $j \in 1..n$, for each $t \in skel(M_j)$, for each distinct identifier A of sort `user`, let $\{x_1, \ldots, x_m\} = Var(t) \cup \{x_A, x_{S_i}, x_{R_i}\}$ (no variable occurrence more than once in the sequence $x_1, \ldots, x_m$):

$$i(x_1). \ldots .i(x_m).w(i, x_{S_i}, x_{R_i}, x, x_\ell, xc) \rightarrow$$
$$i(x_1). \ldots .i(x_m).w(i, x_{S_i}, x_{R_i}, x, x_\ell, xc).m(i, I, I, x_A, t, \underline{0}) \qquad (36'')$$

and, if `impersonate` is selected in the field `intruder` of $\mathcal{P}$, by: for each $i \in 1..n$, for each $t \in skel(M_i)$, for each distinct identifiers A, B of sort `user`, let $\{x_1, \ldots, x_m\} = Var(t) \cup \{x_A, x_B, x_{S_i}, x_{R_i}\}$:

$$i(x_1). \ldots .i(x_m).w(i, x_{S_i}, x_{R_i}, x, x_\ell, xc) \rightarrow$$
$$i(x_1). \ldots .i(x_m).w(i, x_{S_i}, x_{R_i}, x, x_\ell, xc).m(i, I, x_A, x_B, t, \underline{0}) \qquad (37'')$$

Because of deletion of rules (32)-(35), one rule for public key decryption with tables needs to be added:

$$i(\{x_1\}_{x_t[x_u]^{-1}}).i(x_t).i(x_u) \rightarrow i(\{x_1\}_{x_t[x_u]^{-1}}).i(x_p).i(x_u).i(x_1) \qquad (43)$$

The obtained system depends on $\mathcal{P}$. Note that this approximation is not complete. However, it seems to give reasonable results in practice.

5 Flaws

In our state/transition model, a flaw will be detected when the protocol execution reaches some critical state. We define a critical state as a pattern $t_{\text{goal}}(\mathcal{P}) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, which is constructed automatically from the protocol $\mathcal{P}$. The existence of a flaw is reducible to the following reachability problem, where a can be either i or n:

$$\exists t_0, \ldots, t_{\text{goal}}(\mathcal{P})\sigma \in EXEC_a(\mathcal{P}) \text{ for some substitution } \sigma$$

5.1 Design flaws

It may happen that the protocol fails to reach its goals even without intruder, *i.e.* only in presence of honest agents following the protocol carefully. In particular, it may be the case that there is an interference between several concurrent runs of the same protocol: confusion between a message $m(i, \ldots)$ from the first run and another $m(i, \ldots)$ from the second one. An example of this situation is given in Appendix A. The critical state in this case is: (recall that xc and xc' correspond to session counters)

$$t_{\text{goal}}(\mathcal{P}) := w(i, x_s, x_r, x_m, x_l, xc).m(i, x_{s'}, x_s, x_r, x_m, xc').[xc \neq xc']$$

where $[xc \neq xc']$ is a constraint that can be checked either by extra rewrite rules or by an internal mechanism as in daTac.

5.2 Attacks, generalities

Following the classification of Woo and Lam [36], we consider two basic security properties for authentication protocols: secrecy and correspondence. *Secrecy* means that some secret information (*e.g.* a key) exchanged during the protocol is kept secret. *Correspondence* means that every principal was really involved in the protocol execution, *i.e.* that mutual authentication is ensured. The failure of one of these properties in presence of an intruder is called a flaw.

Example 11. The following scenario is a *correspondence attack* for the symmetric key version of the cable tv toy example in Figure 1:

$$\begin{array}{lll} 1. & D \rightarrow I(C) & : \langle D, \{Ins\}_K \rangle \\ 2. & I(C) \rightarrow D & : \langle C, D, \{Ins\}_K \rangle \end{array}$$

Following the traditional notation, the $I(C)$ in step 1 means that I did `divert` the first message of D to C. Note that this ability is selected in Figure 1. It may be performed in real world by interposing a computer between the decoder and the smartcard, with some serial interface and a smartcard reader. The sender $I(C)$ in the second message means that C did `impersonate` C for sending this message. Note that I is able to reconstruct the message of step 2 from the message he diverted at step 1, with a projection π_1 to obtain the name of D and projection π_2 to obtain the cipher $\{Ins\}_K$ and his initial knowledge (the name of the smartcard). Note that the smartcard C did not participate at all to this protocol execution. Such an attack may be used if the intruder wants to watch some channel x which is not registered in his smartcard. See [1] for the description of some real-world hacks on pay TV.

A *secrecy attack* can be performed on the public key version of the protocol in Figure 1. By listening to the message sent by the decoder at step 1, the intruder (with `eaves_dropping` ability) can decode the cipher $\{Ins\}_{T[D]^{-1}}$ since he knows the public key $T[D]$, and thus he will learn the secret instruction Ins. Note that there was no correspondence flaw in this scenario.

5.3 Secrecy attack

Definition 4. *We say that a principal U of $\mathcal{P}$* shares *a (secret) identifier N if there exists j and t such that $N : t \in know(U, j)$.*

In the construction of $R(\mathcal{P})$, we say that the term $t = compose(U, M, j)$ is *bound* to M.

Definition 5. *An execution $t_0, \ldots \in EXEC_i(\mathcal{P})$ satisfies the secrecy property iff for each j, t_j does not contain an instance of $i(t)$ as a subterm, where t is bound to a term N declared in a field* `goal : secrecy of` *N of $\mathcal{P}$.*

To define a critical state corresponding to a secrecy violation in our semantics, we add a binary function symbol $secret(\_,\_)$ to $\mathcal{F}$, which is used to store a term t (nonce or session key) that is bound to some data N declared as secret in $\mathcal{P}$, by `secrecy_of` N. If this term t appears as an argument of $i(\_)$, and I was not supposed to share t, then it means that its secrecy has been corrupted by the intruder I.

We must formalise the condition that "I was not supposed to share t". For this purpose, we add a second argument to $secret(\_,\_)$ which is a term of $\mathcal{T}(\{s, \underline{1}, \ldots, \underline{k}\})$, corresponding to the the value of a session counter, where k is the number of fields `session_instance :` ℓ in $\mathcal{P}$. Let $C = \{\underline{1}, \ldots, \underline{k}\}$. To each field `session_instance` in $\mathcal{P}$ is associated a unique constant in C by the protcedure described in Section 4.1. Let $\mathcal{J} \subseteq C$ be the set of session instances where I has not the role of a principal that shares N.

The critical state $t_{\text{goal}}(\mathcal{P})$ is any of the terms of the set:

$$\{i(x).secret(x, f(\underline{c}))\}_{\underline{c} \in \mathcal{J}}$$

The auxiliary unary function symbol $f(\_)$ scrapes off the $s(\dots)$ context in the values of session counters, using the following rewrite rule (added to $\mathcal{S}_0$):

$$f(s(x)) \to f(x) \tag{44}$$

The storage in $secret(\_,\_)$ is performed in the rewrite rule for constructing the message M_{i+1} where N appears for the first time. More precisely, there is a special construction in the rewrite rule for building M_{i+1}.The binding to the secret N is a side effect of the recursive call of the form $compose(U, N, i)$. The i^{th} rule constructed by our algorithm (page 12) will be in this case:

$$\left\{\begin{array}{l} h(s(x_{\text{time}})). \\ w(i, x_{S_i}, x_{R_i}, x_{M_i}, \ell know(R_i, i), xc). \\ m(i, x_r, x_{S_i}, x_{R_i}, x_{M_i}, xc') \end{array}\right\} \to \\ \left\{\begin{array}{l} h(x_{\text{time}}). \\ m(i+1, x_{R_i}, x_{R_i}, compose(R_i, R_{i+1}, i), compose(R_i, M_{i+1}, i), xc'). \\ w(k_i, compose(R_i, S_{k_i}, i), x_{R_i}, expect(R_i, M_{k_i}, i'), \ell'_i, c'). \\ secret(t, f(xc')) \end{array}\right\}$$

Example 12. The rules generated for the protocol of Figure 1, public key version, are:

$$h(s(x_{\text{time}})).w\big(0, x_{S_0}, x_D, x_{M_0}, \langle x_D, x_C, x_T, x_{T[D]^{-1}}\rangle, xc\big) \to \\ h(x_{\text{time}}).m\big(1, x_D, x_D, x_C, \langle x_D, \{nonce(s_1(x_{\text{time}}))\}_{x_{T[D]^{-1}}}\rangle, xc\big). \\ w\big(2, x_C, x_D, \langle x_C, x_D, \{nonce(s_1(x_{\text{time}}))\}_{x_1}\rangle, \\ \langle x_D, x_C, x_T, x_{T[D]^{-1}}, x_{M_0}, x_{S_0}, nonce(s_1(x_{\text{time}}))\rangle, xc\big) \\ secret(nonce(s_1(x_{\text{time}})), f(xc)) \tag{tvp$_1$}$$

$$h(s(x_{\text{time}})).w\big(1, x_D, x_C, x_{M_1}, \langle x_C, x_T, x_{T[C]^{-1}}\rangle, xc\big). \\ m\big(1, x_r, x_D, x_C, x_{M_1}, xc'\big) \to \\ h(x_{\text{time}}).m\big(2, x_C, x_D, \pi_1(x_{M_1}), \langle x_C, \pi_1(x_{M_1}), \pi_2(x_{M_1})\rangle, xc'\big). \\ w\big(1, x_1, x_{U_1}, \langle x_1, \{x_2\}_{x_K}\rangle, \langle x_C, x_T, x_{T[C]^{-1}}\rangle, s(xc)\big) \tag{tvp$_2$}$$

$$h(s(x_{\text{time}})).w\big(2, x_C, x_D, x_{M_2}, \langle x_D, x_C, x_T, x_{T[D]^{-1}}, x_{M_0}, x_{S_0}, x_{Ins}\rangle, xc\big). \\ m\big(2, x_r, x_C, x_D, x_{M_2}, xc'\big) \to \\ h(x_{\text{time}}).w\big(0, x_{S_0}, x_D, x_{M_0}, \langle x_D, x_C, x_T, x_{T[D]^{-1}}\rangle, s(xc)\big) \tag{tvp$_3$}$$

Note the term $secret(nonce(s_1(x_{\text{time}})), xc)$ in rule (tvp$_1$). As described in Example 11, it is easy to see that this protocol has a secrecy flaw. A subterm $secret(nonce(x), \underline{1}).i(nonce(x))$ is obtained in 4 steps, see appendix C.

5.4 Correspondence attack

The correspondence property between two users U and V means that when U terminates its part of a session c of the protocol (and starts next session $s(c)$), then V must have started his own part, and reciprocally. In Definition 6, we use the notation $\text{first}_S(U) = \min\{i \mid S_i = U\}$, assuming $\min(\emptyset) = 0$.

Definition 6. *An execution $t_0, \ldots \in EXEC_i(\mathcal{P})$ satisfies the correspondence property between the (distinct) users U and V iff for each j, t_j does not contain a subterm matching:*

$$w(\text{first}_S(U) - 1, x_s, u, x_t, x_\ell, s(x_c)).w(\text{first}_S(V) - 1, x'_s, x'_r, x'_t, x'_\ell, x_c)$$
$$\textit{or } w(\text{first}_S(V) - 1, x_s, v, x_t, x_\ell, s(x_c)).w(\text{first}_S(U) - 1, x'_s, x'_r, x'_t, x'_\ell, x_c),$$

where $U : u$ and $V : v$ occur in the same line of the field `session_instance`.

The critical state $t_{\text{goal}}(\mathcal{P})$ is therefore any of the two above terms in Definition 6. Again, these terms are independent from $\mathcal{P}$.

Example 13. A critical state for the protocol in Figure 1, symmetric key version, is: $t_{\text{goal}}(\mathcal{P}) := w(0, x_1, tv, x_{M_1}, x_{l_1}, xc).w(1, x_2, scard, x_{M_2}, x_{l_2}, s(xc))$

5.5 Key compromising attack

A classical goal of cryptographic protocols is the exchange between two users A and B of new keys – symmetric or public keys. In such a scenario, A may propose to B a new shared symmetric key K or B may ask a trusted server for A's public key K, see Section 5.6 below for this particular second case. In this setting, a technique of attack for the intruder is to introduce a compromised key K': I has built some key K' and he let B think that K' is the key proposed by A or that this is A's public key for instance (see Example 14 for a key compromising attack). The compromising of K may be obtained by exploiting for instance a type flaw as described below. Such an attack is not properly speaking a secrecy attack. However, it can of course be exploited if later on B wants to exchange some secret with A using K (actually the compromised K').

Therefore, a key compromising attack is defined as a secrecy attack for an extended protocol $\mathcal{P}'$ obtained from a protocol $\mathcal{P}$ of the above category as follows:

1. declare a new `identifier` X : `number`;
2. add a rule: $n + 1.\ B \to A : \{X\}_K$ where n is the number of messages in $\mathcal{P}$ and K is the key to compromise,
3. add the declaration `goal` : `secrecy_of` X;

5.6 Binding attack

This is a particular case of key compromising attack, and therefore a particular case of secrecy attack, see Section 5.5. It can occur in protocols where the public keys are distributed by a trusted server (who knows a table K of public keys) because the principals do not know in advance the public keys of others. In some case, the intruder I can do appropriate diverting in order to let some principal learn a fake binding name – public key. For instance, I makes some principal B believe that I's public key $K[I]$ is the public key of a third party A (binding A–$K[I]$). This is what can happen with the protocol SLICE/AS, see [7].

5.7 Type flaw

This flaw occurs when a principal can accept a term of the wrong type. For instance, he may accept a pair of numbers instead of a new symmetric key, when numbers, pair of numbers and symmetric keys are assumed to have the same type. Therefore, a type flaw refers more to implementation hypotheses than to the protocol itself. Such a flaw may be the cause of one of the above attack, but its detection requires a modification of the sort system of $\mathcal{F}$. The idea it to collapse some sorts, by introducing new sorts equalities. For instance, one may have the equality `symmetric_key` = `text` = `number`. By definition of profiles of $\{\_\}_\_$ and $\langle \_, \_ \rangle$, ciphers and pairs are in this case `number`s, and be accepted as `symmetric_key`.

Example 14. A known key compromising attack on Otway-Rees protocol, see [7], exploits a type flaw of this protocol. We present here the extended version of Otway-Rees, see Section 5.5.

`protocol Ottway Rees`
`identifiers`
A, B, S : `user`;
K_{as}, K_{bs}, K_{ab} : `symmetric_key`;
M, N_a, N_b, X : `number`;
`messages`
1. $A \to B : \langle M, A, B, \{N_a, M, A, B\}_{K_{as}} \rangle$
2. $B \to S : \langle M, A, B, \{N_a, M, A, B\}_{K_{as}}, N_b, M, A, B_{K_{bs}} \rangle$
3. $S \to B : \langle M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}} \rangle$
4. $B \to A : \langle M, \{N_a, K_{ab}\}_{K_{as}} \rangle$
5. $A \to B : \{X\}_{Kab}$

`knowledge`
$A : B, S, K_{as}$;
$B : S, K_{bs}$;
$S : A, B, K_{as}, K_{bs}$;
`session_instance` $[A : a, B : b, S : s, k_{as} : k_{as}, K_{bs} : k_{bs}]$;
`intruder` : `divert, impersonate`;
`intruder_knowledge` : ;
`goal` : `secrecy_of` X;

The symmetric keys K_{as} and K_{bs} are supposed to be only known by A and S, resp. B and S. The identifiers M, N_a, and N_b re nonces. The new symmetric K_{ab} is generated by the trusted server S and transmitted to B and indirectly to A, by mean of the cipher $\{N_a, K_{ab}\}_{K_{as}}$.

If the sorts `numbers`, `text`, and `symmetric_key` are assumed to collapse, then we have the following scenario:

1. $A \to I(B) : \langle M, A, B, \{N_a, M, A, B\} K_{as} \rangle$
4. $I(B) \to A : \langle M, \{N_a, M, A, B\} K_{as} \rangle$
5. $A \to I(B) : \{X\}_{\langle M, A, B \rangle}$

In rule 1, I diverts (and memorises) A's message. In next step 4, I impersonates B and makes him think that the triple $\langle M, A, B\rangle$ is the new shared symmetric key K_{ab}. We recall that $\langle \_, \_ \rangle$ is right associative and thereafter $\langle N_a, M, A, B\rangle$ can be considered as identical to $\langle N_a, \langle M, A, B\rangle\rangle$

6 Verification: deduction techniques and experiments

We have implemented the construction of $R(\mathcal{P})$ in OCaml® and performed experiments using the theorem prover daTac [33] with paramodulation modulo AC. Each rule $l \to r \in R(\mathcal{P})$ is represented as an oriented equation $l = r$, the initial state is represented as a unit positive clause $P\big(t_{init}(\mathcal{P})\big)$ and the critical state as a unit negative clause $\neg P\big(t_{\text{goal}}(\mathcal{P})\big)$.
As for multiset rewriting [8], an ac-operator will take care of concurrency. On the other hand unification will take care of communication in an elegant way. The deduction system combines paramodulation steps with equational rewriting by $\mathcal{S}_0$.

6.1 Deduction techniques. Generalities

The main deduction technique consists in replacing a term by an equal one in a clause: given a clause $l = r \vee C'$ and clause $C[l']$, the clause $(C' \vee C[r])\sigma$ is deduced, where σ is a unifier of l and l', that is a mapping from variables to terms such that $l\sigma$ is equal to $l'\sigma$.

This deduction rule is called *paramodulation*. It has been introduced by Robinson and Wos [27]. Paramodulation (together with resolution and factoring) was proved refutationally complete by Brand [6] who also shown that applying a replacement in a variable position is useless.
For reducing the number of potential deduction steps, the paramodulation rule has been restricted by an ordering, to guarantee it replaces big terms by smaller ones. This notion of ordered paramodulation has been applied to the Knuth-Bendix completion procedure [16] for avoiding failure in some situations (see [14] and [2]). A lot of work has been devoted to putting more restrictions on paramodulation in order to limit combinatorial explosion [23].

In particular paramodulation is often inefficient with axioms such as associativity and commutativity since these axioms allow for many successful unifications between their subterms and subterms in other clauses. Typically word problems in finitely presented commutative semigroups cannot be decided by standard paramodulation. This gets possible by building the associativity and commutativity in the paramodulation rule using the so-called paramodulation modulo AC and rewriting modulo AC rules.

The integration of associativity and commutativity axioms within theorem-proving systems has been first investigated by Plotkin [26] and Slagle [31]. Rusinowitch and Vigneron [29] have built-in this theory in a way that is compatible with the ordered paramodulation strategy and rewriting and preserves refutational completeness. These techniques are implemented in the daTac system [33].

Another approach has been followed by Wertz [35] and Bachmair and Ganzinger [3], consisting of using the idea of extended clauses developed for the equational case by Peterson and Stickel [25].
In all the approaches, the standard unification calculus has to be replaced by unification modulo associativity and commutativity. This may be very costly since some unification problems have doubly exponentially many minimal solutions [12].

6.2 Deduction rules for protocol verification

We present here the version of paramodulation we have applied for simulating and verifying protocols. States are built with the specific ac-operator "." for representing the multiset of information components: sent and expected messages, and the knowledge of the intruder.

The definition of our instance of the paramodulation rule is the following.

Definition 7 (Paramodulation). $\dfrac{l = r \qquad P(l')}{P(r.z)\sigma}$ *if* σ *is an ac-unifier of* $l.z$ *and* l'*, and* z *is a new variable.*

This rule is much simpler than the general one in [29]. We only need to apply replacements at the top of the term. In addition the equations are such that the left-hand side is greater than the right-hand side and each clause is unit. So we do not need any strategy for orienting the equations or selecting a literal in a clause.

In the verification of protocols, we encounter only simple unification problems. They reduce to unifying multisets of standard terms, where one of the multisets has no variable as argument of ".". Only one argument of the other multiset is a variable. Hence for handling these problems we have designed a unification algorithm which is more efficient than the standard ac-unification algorithm of daTac.

Let us illustrate this with an example.

Example 15. For performing a paramodulation step from $f(x_1).g(a) = c$ into $P(a.g(x_2).f(b).h(x_3))$, trying to unify $f(x_1).g(a)$ and $a.g(x_2).f(b).h(x_3)$ will not succeed. We have to add a new variable in the left-hand side of the equation for capturing the additional arguments of the ac-operator. The unification problem we have to solve is $f(x_1).g(a).z =^?_{ac} a.g(x_2).f(b).h(x_3)$. Its unique solution σ is $\{x_1 \mapsto b, x_2 \mapsto a, z \mapsto a.h(x_3)\}$. The deduced clause is $P(c.z)\sigma$, that is $P(c.a.h(x_3))$.

The paramodulation rule is used for generating new clauses. We need a rule for detecting a contradiction with the clause representing the goal.

Definition 8 (Contradiction). $\dfrac{P(t) \qquad \neg P(t')}{\square}$ *if* σ *is an ac-unifier of* t *and* t'*.*

In addition to these two deduction rules, we need to simplify clauses by term rewriting, using equations of S_0 (rewrite rules (1)–(6)). For this step we have to compute a match σ of a term l into l', that is a substitution such that $l\sigma = l'$.

Definition 9 (Simplification). $\dfrac{l = r \qquad P(t[l'])}{P(t[r\sigma])}$ *if* σ *is a match of* l *into* l'.

Applying this rule consists in replacing the initial clause by the simplified one.

6.3 Deduction strategy

We basically apply a breadth first search strategy. The compilation of the protocol generates four sets of clauses:

(0) the rewrite rules of S_0;
(1) the clauses representing transitions rules (including intruder's rules);
(2) the clause representing the initial state, $P\big(t_{init}(\mathcal{P})\big)$;
(3) the critical state $(\neg P\big(t_{\text{goal}}(\mathcal{P})\big))$;

The deduction strategy used by daTac is the following:

```
Repeat:
    Select a clause C in (2), C contains only a positive literal
    Repeat:
        Select a clause D in (1), D is an equation l = r
        Apply Paramodulation from D into C:
            Compute all the most general ac-unifiers
            For each solution σ,
                Generate the resulting clause C'σ
        Simplify the generated clauses:
            For each generated clause C'σ,
                Select a rewrite rule l → r in (0)
                For each subterm s in C'σ,
                    If s is an instance lφ of l
                    Then Replace s by rφ in C'σ
        Add the simplified generated clauses into (2)
        Try Contradiction between the critical state and each new clause:
            If it applies, Exit with message "contradiction found".
    Until no more clause to select in (1)
Until no more clause to select in (2)
```

Note that any derivation of a contradiction $\square$ with this strategy is a linear derivation from the initial state to the goal and it can be directly interpreted as a scenario for a flaw or an attack.

6.4 Results

The approach has been experimented with several protocols described in [7]. We have been able to find the known flaws with this uniform method in several protocols, in less than 1 minute (including compilation) in every case, see Figure 2.

Protocol	Description	Flaw	Intruder abilities
Encrypted Key Exchange	Key distribution	Correspondence attack	divert impersonate
Needham Shroeder Public Key	Key distribution with authentication	Secrecy attack	divert impersonate
Otway Rees	Key distribution with trusted server	Key compromising = secrecy attack type flaw	divert impersonate
Shamir Rivest Adelman	Transmission of secret information	Secrecy attack	divert impersonate
Tatebayashi Matsuzaki Newman	Key distribution	Key compromising = secrecy attack	eaves_dropping impersonate
Woo and Lam Π	Authentication	Correspondence attack	divert impersonate

Fig. 2. Experiments

See http://www.loria.fr/equipes/protheo/SOFTWARES/CASRUL/ for more details.

7 Conclusion

We have presented a complete, compliant translator from security protocols to rewrite rules and how it is used for the detection of flaws. The advantages of our system are that the automatic translation covers a large class of protocols and that the narrowing execution mechanism permits to handle several aspects like timeliness. A drawback of our approach is that the produced rewrite system can be complex and therefore flaw detection gets time-consuming. However, simplifications should be possible to shorten derivations. For instance, composition and reduction with rules $\mathcal{S}_0$ may be performed in one step.

The translation can be directly extended for handling key systems satisfying algebraic laws such as commutativity (cf. RSA). It can be extended to other kinds of flaws: binding, typing... We plan to analyse E-commerce protocols where our management of freshness should prove to be very useful since fresh data are ubiquitous in electronic forms (order and payment e.g.). We plan to develop a

generic daTac proof strategy for reducing the exploration space when searching for flaws. We also conjecture it is possible to modify our approach in order to prove the absence of flaws under some assumptions.

References

1. R. Anderson. Programming Satan's computer. volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag.
2. L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without Failure. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 1–30. Academic Press inc., 1989.
3. L. Bachmair and H. Ganzinger. Associative-Commutative Superposition. In N. Dershowitz and N. Lindenstrauss, editors, *Proc. 4th CTRS Workshop, Jerusalem (Israel)*, volume 968 of *LNCS*, pages 1–14. Springer-Verlag, 1995.
4. D. Basin. Lazy infinite-state analysis of security protocols. In *Secure Networking — CQRE [Secure] '99*, LNCS 1740, pages 30–42. Springer-Verlag, Berlin, 1999.
5. D. Bolignano. Towards the formal verification of electronic commerce protocols. In *IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Society, 1997.
6. D. Brand. Proving Theorems with the Modification Method. *SIAM J. of Computing*, 4:412–430, 1975.
7. J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.
8. G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *Formal Methods and Security Protocols*, 1998. LICS '98 Workshop.
9. G. Denker and J. Millen. Capsl intermediate language. In *Formal Methods and Security Protocols*, 1999. FLOC '99 Workshop.
10. N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. North-Holland, 1990.
11. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29:198–208, 1983. Also STAN-CS-81-854, May 1981, Stanford U.
12. E. Domenjoud. A technical note on AC-unification. the number of minimal unifiers of the equation $\alpha x_1 + \cdots + \alpha x_p \doteq_{AC} \beta y_1 + \cdots + \beta y_q$. *JAR*, 8:39–44, 1992.
13. R. Focardi and R. Gorrieri. Cvs: A compiler for the analysis of cryptographic protocols. In *12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 1999.
14. J. Hsiang and M. Rusinowitch. Proving Refutational Completeness of Theorem-Proving Strategies : the Transfinite Semantic Tree Method. *JACM*, 38(3):559–587, July 1991.
15. J.-M. Hullot. Canonical forms and unification. In *5th International Conference on Automated Deduction*, volume 87, pages 318–334. Springer-Verlag, LNCS, july 1980.
16. D. E. Knuth and P. B. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
17. G. Lowe. Casper: a compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

18. G. Lowe. Towards a completeness result for model checking of security protocols. In *11th IEEE Computer Security Foundations Workshop*, pages 96–105. IEEE Computer Society, 1998.
19. C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.
20. C. Meadows. The NRL protocol analyzer: an overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
21. J. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.
22. J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, pages 141–154. IEEE Computer Society, 1997.
23. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 2000.
24. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
25. G. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *JACM*, 28:233–264, 1981.
26. G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.
27. G. A. Robinson and L. T. Wos. Paramodulation and First-Order Theorem Proving. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 135–150. Edinburgh University Press, 1969.
28. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.
29. M. Rusinowitch and L. Vigneron. Automated Deduction with Associative-Commutative Operators. *Applicable Algebra in Engineering, Communication and Computation*, 6(1):23–56, January 1995.
30. B. Schneier. *Applied Cryptography*. John Wiley, 1996.
31. J. R. Slagle. Automated Theorem-Proving for theories with Simplifiers, Commutativity and Associativity. *JACM*, 21(4):622–642, 1974.
32. P. Syverson, C. Meadows, and I. Cervesato. Dolev-Yao is no better than Machiavelli. In *WITS'00. Workshop on Issues in the Theory of Security*, 2000.
33. L. Vigneron. Positive deduction modulo regular theories. In *Proceedings of Computer Science Logic, Paderborn (Germany)*, pages 468–485. LNCS 1092, Springer-Verlag, 1995.
34. C. Weidenbach. Towards an automatic analysis of security protocols. In *Proceedings of the 16th International Conference on Automated Deduction*, pages 378–382. LNCS 1632, Springer-Verlag, 1999.
35. U. Wertz. First-Order Theorem Proving Modulo Equations. Technical Report MPI-I-92-216, MPI Informatik, April 1992.
36. T. Woo and S. Lam. A semantic model for authentication protocols. In *IEEE Symposium on Research in Security and Privacy*, pages 178–194. IEEE Computer Society, 1993.

Appendix A: design flaws

Example 16.

`identifiers`	M, B, C : `user`;
	O, N : `number`;
	K_b, K_c : `public_key`;
	$hash$: `function`;
`messages`	1. $M \rightarrow C : \{O\}_{K_c}$
	2. $C \rightarrow M : \langle B, \{N\}_{K_b}, hash(N)\rangle$
	3. $M \rightarrow B : \{N\}_{K_b}, hash(O)$
	4. $B \rightarrow M : \left\{ hash\big(hash(N), hash(O)\big) \right\}_{K_b^{-1}}$
`knowledge`	C : $B, K_b, hash$;
	M : $C, O, K_c, K_b, hash$;
	B : $K_b, K_b^{-1}, hash$;
`session_instance`	$[M : Merchant,\ B : Bank,\ C : Customer,$
	$\ O : car,\ K_b : k_b,\ K_c : k_c]$
	$[M : Merchant,\ B : Bank,\ C : Customer,$
	$\ O : peanut,\ K_b : k_b,\ K_c : k_c]$

This is a flawed e-commerce protocol. While browsing an online commerce site, the customer C is offered an object O (together with an order form, price information *etc*) by merchant M. Then, C transmits M a payment form N with his bank account information and the price of O, in order for M to ask directly to C's bank B for the payment. For confidentiality reasons, M must never read the contents of N, and B must not learn O. Therefore, O is encrypted in message 1 with the public key K_c of C. Also, in message 2, N is transmitted by C to M in encrypted form with the bank's public key K_b and in the form of a digest computed with the $hash$ one-way function. Then M relays the cipher $\{N\}_{K_b}$ to B together with a digest of O. The bank B makes the verification for the payment and when it is possible, gives his certificate to M in the form of a dual signature.

The problem is that in message 2, there is no occurrence of O, so there may be some interference between two executions of the protocol. Imagine that C is performing simultaneously two transactions with the same merchant M. In the two concurrent execution of the protocol, M sends 1. $M \rightarrow C : \{car\}_{K_c}$ and 1. $M \rightarrow C : \{peanut\}_{K_c}$. C will reply with two distinct corresponding payment forms (the price field will vary) 2. $C \rightarrow M : \langle B, \{N_{car}\}_{K_b}, hash(N_{car})\rangle$ and 2. $C \rightarrow M : \langle B, \{N_{peanut}\}_{K_b}, hash(N_{peanut})\rangle$. But after receiving these two messages, M may be confused about which payment form is for which offer (recall that M can not read N_{car} and N_{peanut}), and send the wrong requests to B: 3. $M \rightarrow B : \{N_{car}\}_{K_b}, hash(peanut)$ and 3. $M \rightarrow B : \{N_{peanut}\}_{K_b}, hash(car)$. If the bank refuses the payment of N_{car} but authorises the one of N_{peanut}, it will give a certificate for buying a car and paying peanuts! Fortunately for M, the check of dual signature (by M) will fail and transaction will by aborted, but

there is nevertheless a serious interference flaw in this protocol, that can occur even only between two honest agents (without an intruder).

Appendix B: a correspondence attack

Trace obtained by daTac of a correspondence attack for the symmetric key TV protocol (Figure 1).

$$
\begin{aligned}
&t_{init}(\mathcal{P}) = \\
&h(x_1).w(0, x_2, tv, x_3, \langle tv, scard, key\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, \{x_5\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(scard) \\
&\rightsquigarrow_{(\mathrm{tvs}_1)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key}\rangle, \langle tv, scard, key, x_2, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(scard) \\
&\rightsquigarrow_{(26)} \\
&h(x_1).w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key}\rangle, \langle tv, scard, key, x_2, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(tv).i(scard).i(\langle tv, \{nonce(x_1)\}_{key}\rangle) \\
&\rightsquigarrow_{(28)} \\
&h(x_1).w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key}\rangle, \langle tv, scard, key, x_2, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(tv).i(scard).i(\{nonce(x_1)\}_{key}) \\
&\rightsquigarrow_{(37)} \\
&h(x_1).m(2, I, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key}\rangle, \underline{0}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key}\rangle, \langle tv, scard, key, x_2, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(scard).i(tv).i(\{nonce(x_1)\}_{key}) \\
&\rightsquigarrow_{(\mathrm{tvs}_3)} \\
&h(x_1).w(0, x_2, tv, x_3, \langle tv, scard, key\rangle, s(\underline{1})) \\
&\qquad .w(1, x_3, scard, \langle x_3, \{x_4\}_{key}\rangle, \langle scard, key\rangle, \underline{1}) \\
&\qquad .i(scard).i(tv).i(\{nonce(s(x_1))\}_{key})
\end{aligned}
$$

One subterm (of the last term) matches the pattern $t_{\mathrm{goal}}(\mathcal{P})$.

Appendix C: a secrecy attack

Trace obtained by daTac of a secrecy attack for the public key TV protocol (Figure 1).

$$
\begin{aligned}
&t_{init}(\mathcal{P}) = \\
&h(x_1).w(0, x_2, tv, x_3, \langle tv, scard, key, key[tv]^{-1}\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .i(key) \\
&\rightsquigarrow^{(\mathrm{tvp}_1)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \\
&\qquad\quad \langle tv, scard, key, key[tv]^{-1}, x_2, x_3, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .secret(nonce(x_1), f(\underline{1})) \\
&\qquad .i(key) \\
&\rightsquigarrow^{(27)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \\
&\qquad\quad \langle tv, scard, key, key[tv]^{-1}, x_2, x_3, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .secret(nonce(x_1), f(\underline{1})) \\
&\qquad .i(key).i(tv).i(scard).i(\langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle) \\
&\rightsquigarrow^{(28)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \\
&\qquad\quad \langle tv, scard, key, key[tv]^{-1}, x_2, x_3, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .secret(nonce(x_1), f(\underline{1})) \\
&\qquad .i(key).i(tv).i(scard).i(\{nonce(x_1)\}_{key[tv]^{-1}}) \\
&\rightsquigarrow^{(35)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \\
&\qquad\quad \langle tv, scard, key, key[tv]^{-1}, x_2, x_3, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .secret(nonce(x_1), f(\underline{1})) \\
&\qquad .i(key).i(tv).i(scard).i(\{nonce(x_1)\}_{key[tv]^{-1}}).i(key[tv]) \\
&\rightsquigarrow^{(31)} \\
&h(x_1).m(1, tv, tv, scard, \langle tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \underline{1}) \\
&\qquad .w(2, scard, tv, \langle scard, tv, \{nonce(x_1)\}_{key[tv]^{-1}}\rangle, \\
&\qquad\quad \langle tv, scard, key, key[tv]^{-1}, x_2, x_3, nonce(x_1)\rangle, \underline{1}) \\
&\qquad .w(1, x_4, scard, \langle x_4, x_5\rangle, \langle scard, key, key[scard]^{-1}\rangle, \underline{1}) \\
&\qquad .secret(nonce(x_1), f(\underline{1})) \\
&\qquad .i(key).i(tv).i(scard).i(\{nonce(x_1)\}_{key[tv]^{-1}}).i(key[tv]).i(nonce(x_1))
\end{aligned}
$$

The subterm $secret(nonce(x_1), f(\underline{1})).i(nonce(x_1))$ matches the pattern $t_{\mathrm{goal}}(\mathcal{P})$.

Article de référence du Chapitre 5

Y. Chevalier and L. Vigneron.
Strategy for Verifying Security Protocols with Unbounded Message Size.
Journal of Automated Software Engineering, 11(2) : 141-166,
April 2004.
Kluwer Academic Publishers.

Strategy for Verifying Security Protocols with Unbounded Message Size

Y. Chevalier and L. Vigneron *

LORIA - UHP - UN2
Campus Scientifique, B.P. 239
54506 Vandœuvre-lès-Nancy Cedex, France
E-mail: {chevalie,vigneron}@loria.fr

Abstract

We present a system for automatically verifying cryptographic protocols. This system manages the knowledge of principals and checks if the protocol is runnable. In this case, it outputs a set of rewrite rules describing the protocol itself, the strategy of an intruder, and the goal to achieve. The protocol specification language permits to express commonly used descriptions of properties (authentication, short term secrecy, and so on) as well as complex data structures such as tables and hash functions. The generated rewrite rules can be used for detecting flaws with various systems: theorem proving in first-order logic, on-the-fly model-checking, or SAT-based state exploration. These three techniques are being experimented in the European Union project AVISPA.
The aim of this paper is to describe the analysis process. First, we describe the major steps of the compilation process of a protocol description by our tool Casrul. Then, we describe the behavior of the intruder defined for the analysis. Our intruder is based on a lazy strategy, and is implemented as rewrite rules for the theorem prover daTac. Another advantage of our model is that it permits to handle parallel executions of the protocol and composition of keys. And for sake of completeness, it is possible, using Casrul, to either specify an unbounded number of executions or an unbounded message size.
The combination of Casrul and daTac has permitted successful studying of various protocols, such as NSPK, EKE, RSA, Neumann-Stubblebine, Kao-Chow and Otway-Rees. We detail some of these examples in this paper. We are now studying the SET protocol and have already very encouraging results.

Keywords: Security protocols, Verification, Flaw detection, Intruder model, Automatic strategies.

*This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-39252 AVISPA project.

1 Introduction

The verification of cryptographic protocols has been intensively studied these last years. A lot of methods have been defined for analyzing particular protocols [32, 6, 14, 35, 42, 23]. Some tools, such as Casper [26], CVS [19] and CAPSL [15], have also been developed for automating one of the most sensitive step: the translation of a protocol specification into a low-level language that can be handled by different verification systems.

Our work is in this last line. We have designed a protocols compiler, Casrul [25], that translates a cryptographic protocol specification into a set of rewrite rules. This translation step permits, through static analysis of the protocol, to rule out many errors.
The output of our compiler can be used to get a representation of protocols in various systems:

- As Horn clauses, it can be used either by theorem provers in first-order logic, or as a Prolog program.
- As rewrite rules, it can be used by inductive theorem provers, or as an input for a rewrite systems compiler, such as ELAN.
- As propositional formulas, the search for an attack can be seen as a planning problem, which can, after translation into SAT formulas, be solved by SAT solvers such as Chaff [34, 3].

In our case, we use the theorem prover daTac for trying to find flaws in protocols. The technique implemented in daTac is narrowing. This unification-based technique permits us to handle infinite state models (by not limiting the size of messages), and also to guarantee the freshness of the randomly generated information, such as nonces or keys [25]. Note that there was a first approach with narrowing by Meadows in [27], where the narrowing rules are restricted to symbolic encryption equations (see also [30]); transitions are handled by a Prolog-like backward search from a goal stating insecurity.

The main objective of this paper is, after giving a general presentation of Casrul in Section 2, to present in Section 3 an innovative model of the Intruder behavior, based on the definition of a lazy model. This lazy approach, briefly described in [8], is completely different and much more efficient than the model of the Intruder presented in [25]. It permits to handle untyped variables, and hence messages of unbounded size. In this setting, and when the number of principals is finite, the analysis terminates. This strategy has first been implemented in daTac [9], and has also been independently developed later by other authors [12, 31]. In Section 4, we show that our method can be successfully applied to many different kinds of protocols. We explain the results obtained for two protocols and we give a summary of flaws found in other protocols, with the timings. In Section 5, we compare the

Casrul compiler with the other compilers available. We also compare our analysis strategy with other tools.

2 Input Protocols

We present in this section the syntax used for describing security protocols, illustrated in Figure 1. This syntax has been partially detailed in [25], and is close to one of CAPSL [29] or Casper [26] though it differs on some points – for instance, on those in Casper which concern CSP. All the notions we will use for protocols are classical and can be found in [40].
In the following, we present the features added in the syntax for a more powerful expressiveness. We also present some algorithms for verifying the correctness and run-ability of a protocol.
These algorithms are implemented in our compiler, Casrul[1], that transforms a protocol given as in Figure 1 into a set of rewrite rules. In [25], we have proved that this compilation defines a non-ambiguous operational semantics for protocols and Intruder behavior.

The information given for describing a protocol can be decomposed into two parts: the description of the protocol itself, and the execution environment. This last part includes the legitimate participants and the abilities given to an intruder.

2.1 Protocol Information

The description of a protocol is the composition of three types of information: the identifiers, the initial knowledge, and the messages. Let us present each of these.

2.1.1 Identifiers

This section contains the declaration of all the identifiers used in the protocol messages. This includes principals (users), keys (symmetric, public/private, table), piece of text or numbers, hash functions. Some of these identifiers will be used as fresh information, i.e. they will be generated during the execution of the protocol.
Let us give more details about the four kinds of supported encryption algorithms, the last two being new features, added in the last version of Casrul:

- A key K is an asymmetric key if the key K^{-1} permitting to decrypt a cipher $\{M\}_K$ encrypted by K *cannot* be easily derived from K. The keys K and K^{-1} have to be created at the same time. One is made publicly available, the *public key*, and the other shall be kept secret and is called the *private key*. In our system, public keys are not accessible to anyone by default, since the goal of a protocol may be to ensure that each participant associates a public key to the right person (*i.e.* the possessor of the private key).

[1] `http://www.loria.fr/equipes/cassis/softwares/casrul/`

Protocol `WLMA`;
Identifiers
A, B, S : *User*;
Na, Nb : *Number*;
Kab, Kas, Kbs : *Symmetric_key*;
Knowledge
A : B, S, Kas;
B : S, Kbs;
S : A, B, Kas, Kbs;
Messages
1. $A \rightarrow B$: A, Na
2. $B \rightarrow A$: B, Nb
3. $A \rightarrow B$: $\{A, B, Na, Nb\}Kas$
4. $B \rightarrow S$: $\{A, B, Na, Nb\}Kas$, $\{A, B, Na, Nb\}Kbs$
5. $S \rightarrow B$: $\{B, Na, Nb, Kab\}Kas$, $\{A, Na, Nb, Kab\}Kbs$
6. $B \rightarrow A$: $\{B, Na, Nb, Kab\}Kas, \{Na, Nb\}Kab$
7. $A \rightarrow B$: $\{Nb\}Kab$

Role
$A[A : b; B : I; S : se; Kas : kbs]$,
$B[B : b; S : se; Kbs : kbs]$;
Parallel
$S[A : I; B : b; S : se; Kas : kis; Kbs : kbs]$,
$S[A : b; B : I; S : se; Kas : kbs; Kbs : kis]$;
Secret kbs;
Intruder *Divert*, *Impersonate*;
Intruder_knowledge b, se, kis;
Goal B *authenticates* S *on* Kab;

Figure 1: Woo and Lam Mutual Authentication Protocol.

- A key K is a *symmetric key* if the key K^{-1} permitting to decrypt a cipher encrypted by K is K itself, or a key that can easily be derived from K.
- A *table* T associates a public and a private key to the name of a principal A: $T[A]$ and $T[A]^{-1}$. Initially, only the owner of the table knows those keys.
- A *function* f is a one-way, collision-free, hash function algorithm. Thus, for a message M and a *function* f, $f(M)$ is the hash of M calculated by the algorithm f.

2.1.2 Initial Knowledge

For defining the initial state of a protocol, we have to list the initial knowledge of each principal.

An identifier (key or number) that is not in any initial knowledge will be used as a *fresh* information, created at its first use (for example, Na, Nb and Kab in Figure 1). Note that this is possible to give *messages* in the initial knowledge of a principal as long as all identifiers appearing in these messages are defined.

2.1.3 Messages

They describe the different steps of the protocol with, for each one, its index, the name of its sender, the name of its receiver, and the body of the message itself. The syntax for encoding is very classical: $\{M\}_K$ means the message M encoded by the key K. The encoding is supposed to be a public/private key encoding if K is a *public* or *private* key, or an element of a table. If K is a *symmetric* key (or a *compound message*, as used in SSL for instance), it is assumed that a symmetric encryption/decryption algorithm is used to encode the ciphers. We also allow Xor encryption with the notation $(M)\mathsf{xor}(T)$, in which we assume M and T are two expressions of the same size, thus getting rid of block properties of Xor encryption.

All this information brings a precise view of the proposed protocol, and at this point we should be able to run the protocol. However, the model of a principal is not complete: we have to check that the protocol is correct and runnable by verifying the evolution of the knowledge of each principal.

2.2 Correctness of the Protocol

The knowledge of the principals in a protocol is always changing. One has to verify that all the messages can be composed and sent to the right person to guarantee the protocol can be run.
The knowledge of each participant can be decomposed into three parts:

- the initial knowledge, declared in the protocol,
- the acquired knowledge, obtained by decomposition of the received messages,
- the generated knowledge, created for composing a message (fresh knowledge).

A protocol is correct and runnable with respect to the initial specification if each principal can compose the messages it is supposed to send. For some messages, principals will use parts of the received messages. Thus, a principal has to update its knowledge as soon as it receives a message: it has to store the new information, and check if it can be used for decoding old ciphers (i.e. parts of received messages it could not decode because it did not have the right key).

The function *compose* defined in Figure 2 describes the composition of a message M by a principal U at the step i of the protocol. The knowledge of U before running this function is therefore the union of its initial knowledge and the information it could get in the received and sent messages, until step $i-1$ (included). For an easier reuse of this knowledge, a name is assigned to each information.

$$
\begin{aligned}
compose(U, M, i) &= t && \text{if } M \text{ is known by } U \text{ and named } t && (1)\\
compose(U, \langle M_1, M_2\rangle, i) &= \langle compose(U, M_1, i), compose(U, M_2, i)\rangle && && (2)\\
compose(U, (M_1)\mathsf{xor}(M_2), i) &= (compose(U, M_1, i))\mathsf{xor}(compose(U, M_2, i)) && && (3)\\
compose(U, \{M\}_K, i) &= \{compose(U, M, i)\}_{compose(U,K,i)} && && (4)\\
compose(U, T[A], i) &= compose(U, T, i)[compose(U, A, i)] && && (5)\\
compose(U, T[A]^{-1}, i) &= compose(U, T, i)[compose(U, A, i)]^{-1} && && (6)\\
compose(U, f(M), i) &= compose(U, f, i)(compose(U, M, i)) && && (7)\\
compose(U, M, i) &= fresh(M) && \text{if } M \text{ is a fresh identifier} && (8)\\
compose(U, M, i) &= \texttt{Fail} && \text{else} && (9)
\end{aligned}
$$

Figure 2: Verification that a message can be composed.

Note that when a fresh identifier (key, nonce, ...) is encountered for the first time, a unique new term is automatically generated.
As the message M has to be sent by U at step i, any problem will generate a failure in this function. This case implies that the principal does not have enough knowledge to compose the message to send, and hence the protocol is not *runnable*. The compilation process will abort and an error message indicating which piece of knowledge could not be composed is output.

In addition to being able to compose the messages, a principal has also to be able to verify the information received in messages: if it is supposed to receive an information it already knows, it has to check this is really the same. A principal also knows the shape of the messages it receives. So it has to be able to check that everything it can access in a received message corresponds to what it expects.
These verifications are pre-computed during the compilation process by the function *expect* defined in Figure 3. This function describes the behavior of a principal U that tries to give an as accurate as possible value to every part of a message it will receive. Each unknown cipher is replaced by a new variable $x_{U,M}$ that can be seen as a new name.
Rules 12 and 13 mean that U needs to know one of the arguments of the Xor operator to get the other one and study it. Rules 14, 15 and 16 describe that U has to be able to compose the key decoding the analyzed cipher, i.e. the inverse key of the one coding the message, for studying the contents of the message M.
Note that K stands for a public key, K^{-1} for a private key (possibly through the use of a table), and SK for a symmetric key or a compound term.

$$
\begin{aligned}
expect(U, M, i) &= compose(U, M, i) && \text{if no } \texttt{Fail} \quad (10)\\
expect(U, \langle M_1, M_2\rangle, i) &= \langle expect(U, M_1, i), expect(U, M_2, i)\rangle && (11)\\
expect(U, (M_1)\mathsf{xor}(M_2), i) &= (compose(U, M_1, i))\mathsf{xor}(expect(U, M_2, i)) && \text{if no } \texttt{Fail} \quad (12)\\
expect(U, (M_1)\mathsf{xor}(M_2), i) &= (expect(U, M_1, i))\mathsf{xor}(compose(U, M_2, i)) && \text{if no } \texttt{Fail} \quad (13)\\
expect(U, \{M\}_K, i) &= \{expect(U, M, i)\}_{compose(U,K^{-1},i)^{-1}} && \text{if no } \texttt{Fail} \quad (14)\\
expect(U, \{M\}_{K^{-1}}, i) &= \{expect(U, M, i)\}_{compose(U,K,i)^{-1}} && \text{if no } \texttt{Fail} \quad (15)\\
expect(U, \{M\}_{SK}, i) &= \{expect(U, M, i)\}_{compose(U,SK,i)} && \text{if no } \texttt{Fail} \quad (16)\\
expect(U, M, i) &= x_{U,M} && \text{else} \quad (17)
\end{aligned}
$$

Figure 3: Verification of a received message.

These algorithms are implemented in Casrul. This compiler can therefore generate rewrite rules modeling the behavior of principals: principals wait until a message is received, and immediately send a response. This can be summarized by the following kind of rule:

$$expect(U, M_i, i) \Rightarrow compose(U, M_{i+1}, i+1)$$

Considering the Woo and Lam protocol given in Figure 1, for each role, the transitions are modeled by the following rewrite rules, where x_A, x_{Na}, x_{Kbs}, ... are names of known knowledge, and x_1, x_2, ... are new names (i.e. variables) different for each role and representing previously unknown pieces of messages.

- Role A: (initial knowledge: x_A, x_B, x_S, x_{Kas})
  - Composed for x_B:
    $\Rightarrow x_A, fresh(Na)$
    A generates a new nonce and will name it x_{Na}.
  - Expected from x_B, composed for x_B:
    $x_B, x_1 \Rightarrow \{x_A, x_B, x_{Na}, x_1\}_{x_{Kas}}$
    In this case, A does not know the second information sent by B and names it x_1. But since this has to be a nonce, later it will name it x_{Nb}.
  - Expected from x_B, composed for x_B:
    $\{x_B, x_{Na}, x_{Nb}, x_2\}_{x_{Kas}}, \{x_{Na}, x_{Nb}\}_{x_2} \Rightarrow \{x_{Nb}\}_{x_2}$
    A is able to find x_2 in the first part of the received message; therefore it is able to check the composition of the second part.
- Role B: (initial knowledge: x_B, x_S, x_{Kbs})
  - Expected from x_1, composed for x_1:
    $x_1, x_2 \Rightarrow x_B, fresh(Nb)$

- Expected from x_A, composed for x_S:
  $x_3 \Rightarrow x_3, \{x_A, x_B, x_{Na}, x_{Nb}\}_{x_{Kbs}}$
  B cannot decrypt the received message ($\{A, B, Na, Nb\}_{Kas}$) and gives it the name x_3.
- Expected from x_S, composed for x_A:
  $x_4, \{x_A, x_{Na}, x_{Nb}, x_5\}_{x_{Kbs}} \Rightarrow x_4, \{x_{Na}, x_{Nb}\}_{x_5}$
- Expected from x_A:
  $\{x_{Nb}\}_{x_{Kab}} \Rightarrow$

- Role S: (initial knowledge: x_S, x_A, x_B, x_{Kas}, x_{Kbs})
  - Expected from x_B, composed for x_B:
    $\{x_A, x_B, x_1, x_2\}_{x_{Kas}}, \{x_A, x_B, x_1, x_2\}_{x_{Kbs}}$
    $\Rightarrow \{x_A, x_1, x_2, fresh(Kab)\}_{x_{Kas}}, \{x_B, x_1, x_2, fresh(Kab)\}_{x_{Kbs}}$
    S puts the same fresh key Kab in each part of the composed message.

2.3 Execution Environment

Verifying a protocol consists in trying to simulate what an intruder could do for disturbing the run of a protocol, without some participants noticing. In the previous section, we have defined how generic honest participants of the protocol behave according to the message sequence. However, trying all the possible instantiations of a protocol does not terminate since there are infinitely many.
In this section, we describe how to specify an environment of execution for the protocol. An initial state is given, representing the principals who may run the protocol, together with the roles they might assume. This environment permits to ensure termination of the verification, and also to define this initial state. It is composed using either *Roles* or *Parallel roles* declarations. We also describe how to specify the abilities and knowledge of an attacker, together with the possible goals of this attacker.

2.3.1 Roles

This field describes the possible instances of roles taken by principals in the protocol. Formally, roles such as A, B,... are instantiated by principals. One role may be instantiated zero, one or more times. This is possible to define independently the participants, thus permitting a large flexibility in the definition of the initial state of the protocol.
For example, in the protocol of Figure 1, two roles are defined: b can play the role A with the Intruder I; b can also play the role B.

2.3.2 Parallel Roles

This field is used to specify instances of roles that can be run an unbounded number of times in parallel. As is the case for Roles declaration, it is only possible to

specify a finite number of *different* instances. It corresponds to a weakening of a role definition, because we do not allow those instances to create different nonces. It is used in conjunction with a field `Secret`, that defines instances the Intruder should never be able to know. These instances permit to reduce the number of rules generated by Casrul. A secrecy goal is generated for each of those secret instances. The parallel roles will be played by honest users, but who will be manipulated by the Intruder.
For example, in the protocol of Figure 1, in parallel to the official roles, the Intruder can use a server with whom he can either pretend to be b and play role B, or play role B with its real identity.

2.3.3 Intruder

The `Intruder` field describes which strategies the Intruder can use, among three possibilities: passive `eaves_dropping`, `divert` and `impersonate`. These properties depend on the assumptions made on the execution environment of the protocol. If nothing is specified, this means that we want a simulation of the protocol in a safe network.
When communicating through an unsafe media, one should assume an Intruder is present. Depending on the network, he can have the ability `divert`, `eaves_dropping`, `impersonate` or any combination of these. If `divert` is selected, he can remove messages from the network; if `eaves_dropping` is selected, he can just record the contents of messages exchanged.
The Intruder is then able to reconstruct terms as he wishes, using all the information he got. He can send arbitrary messages in his own name. If moreover `impersonate` is selected, he can also send messages in the name of another principal.

In the description of the Intruder model (Section 3), we will focus on the case where he may divert messages and impersonate principals.

2.3.4 Intruder Knowledge

The `Intruder_knowledge` is the list of information known from the beginning by the Intruder. Contrarily to the initial knowledge of other principals, each element of the messages in the Intruder knowledge has to have been introduced in `Role` or `Parallel` role, as an effective knowledge (and not a formal one used for describing the messages).

2.3.5 Goal

This field gives the kind of flaw we want to detect. There are several possibilities, but the two main ones are `authenticates` and `Secrecy_of`.
– *Secrecy* means that some secret information (e.g. a key or a number) exchanged during the protocol is kept secret.

– An authentication goal is defined for instance by

$$B \texttt{ authenticates } S \texttt{ on } Kab$$

which means that if a principal b playing the role B ends its part of the protocol and believes it has interacted with a principal se playing the role S, and if it believes it has received Kab sent by se, then, at some point, the principal se must have sent Kab to b.
Note that the authentication goal relies on the freshness of information in the system of rewrite rules. For example, in the system devised by Blanchet [5], nonces are not guaranteed to be different from one session of the protocol to another, and in this abstraction, fake authentication attacks can be found on all protocols.
– A last goal has been introduced in order to automatically handle the case of some compromised secrets: the `Short_term_secrecy` goal. In this case, one can define identifiers that should remain secret in a part of the protocol (usually during the session instance where they have been created). Then they are released, i.e. they are added to the knowledge of the Intruder.

3 Intruder's Model

One of the biggest problem in the area of cryptographic protocols verification is the definition of the Intruder. The most referred model is the one defined by *Dolev and Yao* in [17]. It says that the Intruder controls entirely the communication network. This means that he can intercept, record, modify, compose, send, crypt and decrypt (if he knows the appropriate key) each message. He can also send messages in the name of another principal.
However, the Dolev-Yao's model of an Intruder is not scalable, since there are rules for composing messages, and these rules such as building a couple from two terms, do not terminate: given a term, it is possible to build a couple with two copies of this term, and to do it again with that couple, and so on.

In some approaches people try to bound the size of the messages, but these bounds are valid only when one considers specific kinds of protocols and/or executions. We want to be able to study all the protocols definable within the Casrul syntax, and to get a system that is as independent as possible w.r.t. the initial state. Thus, those bounds are not relevant in our approach, and this has led us to bring a new model of the Intruder.

A common approach to deal with infinite-space problem is to use a lazy exploration of the state space while analyzing the protocol by model-checking. This approach was proposed for example in [4]. In a totally different way, our work can be connected to this since we have developed a lazy version of Dolev-Yao's Intruder: we replace the terms building step of the Intruder by a step in which, at the same time, the Intruder analyzes his knowledge and tests if he can build a term matching the message awaited by a principal; the pattern of the awaited message is given by the principal, instead of being blindly composed by the Intruder. This defines our

model as a lazy one, where a symbolic analysis is performed on *classes* of possible executions, those classes being lazily constructed. More information on this is given in [10].
This strategy may look similar to the one described in [37] (Chapter 15), but our lazy model is applied dynamically during the execution of the protocol, while Roscoe's model consists in looking for the messages that can be composed by the Intruder before the execution of the protocol. The messages are prepared statically in advance, and some type information permits to bound the total size of the system. This is a strong restriction to the Dolev-Yao model. One advantage of our method is that we can find some type flaws (in the Otway-Rees protocol, for instance) that cannot be found when the size of messages is bounded by typing.
In the first version of Casrul [25], we used to have a static method similar to Roscoe's, where we were generating many rules in which the Intruder was impersonating the principals. This method was found unsuitable for complex protocols, where a given rule can be applied in many different, yet often equivalent, ways.

In the following, we first briefly present the system testing if terms can be built. Then, we define a system for decomposing the Intruder's knowledge, relying on the testing system. It is remarkable that the knowledge decomposition using this system now allows decomposition of ciphers with composed key (see the Otway-Rees example in Section 4.2) and even the *Xor*-encryption, whereas other similar models such as [1] only allow atomic symmetric keys.

For the next two sections, we have to give the meaning of the terms in the rewrite rules generated by Casrul.

- Atomic terms are those constants declared in the `Role` and `Parallel` fields;
- Some unary operators are used to type those constants, such as MR to describe a principal; we also use F for representing any of those operators;
- The C operator stands for building a couple (i.e. a concatenation) of messages;
- CRYPT, SCRYPT and XCRYPT operators stand respectively for public or private key encryption, symmetric key encryption and *Xor*-encryption;
- TABLE(t_1, t_2) is valid if t_1 is the name of a table, and t_2 the name of a principal. In this case, TABLE(t_1, t_2) stands for the public key of t_2 registered in table t_1;
- FUNC(t_1, t_2) is valid if t_1 is a function symbol and t_2 is a message. In this case, FUNC(t_1, t_2) is the hash of t_2 computed with the algorithm t_1.

To describe our lazy version of Dolev and Yao's intruder using a set of rewrite rules, we also use other operators whose meaning should be clear from the name. For instance, *Comp* is used for the composition of a message; note that the "." operator is not a list constructor, but an associative and commutative (AC) operator. These rules originate from the implementation of the lazy intruder in daTac.

3.1 Test of Composition of a Term

The heart of our Intruder's model is to *test* if a term matching a term t can be composed from a knowledge set C. The rewriting system described in Figure 4 tries to reduce the expression $Comp(t)$ *from* C ; Id , building a substitution τ. In this expression, Id stands for the identity substitution, and $Comp(t)$ *from* C is a constraint for the Intruder meaning that the term t has to be composable from the knowledge list C.
This test is a constraint solving algorithm; it cannot be compared with functions *compose* and *expect* described in Section 2.2, which are defined for normal principals: *compose* is used for checking that the protocol is runnable; *expect* is used for verifying the contents of received messages.

$$Comp(t).T \text{ from } t.C \ ; \ \tau \ \rightarrow \ T \text{ from } t.C \ ; \ \tau \tag{18}$$

$$Comp(r).T \text{ from } s.C \ ; \ \tau \ \rightarrow \ T\sigma \text{ from } s\sigma.C\sigma \ ; \ \tau\sigma \quad \text{if } r\sigma = s\sigma \tag{19}$$

$$Comp(\text{C}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{20}$$

$$Comp(\text{CRYPT}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{21}$$

$$Comp(\text{SCRYPT}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{22}$$

$$Comp(\text{XCRYPT}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{23}$$

$$Comp(\text{TABLE}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{24}$$

$$Comp(\text{TABLE}(t_1, t_2)^{-1}).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{25}$$

$$Comp(\text{FUNC}(t_1, t_2)).T \text{ from } C \ ; \ \tau \ \rightarrow \ Comp(t_1).\, Comp(t_2).T \text{ from } C \ ; \ \tau \tag{26}$$

Figure 4: System testing if a term may be composed from some given knowledge.

The system of Figure 4, being complete in the sense that it can find all the ways of composing a term, cannot be confluent since two different ways will lead to two different normal forms. Further investigation is needed to handle the case of the *Xor*-encryption. The rule (23) cannot handle, for example, that:

$$(x \oplus y) \oplus (y \oplus z) = x \oplus z$$

The effectiveness of this system heavily relies on the fact that we *do not* use the rule (19) when the term r is a variable, thereby reducing the test of composability of a term to the test of composability of some of its variables, which can then be instantiated later. Moreover, termination is ensured by restricting the applications of rules (20-26) to the cases where they apply on terms which are not variables. This last restriction is mandatory to ensure termination, since the Intruder would otherwise test terms of unbounded depth.

Theorem of Completeness. For any set of constraints, if there exists a substitution that satisfies all its constraints, our strategy permits to transform this set into either an empty set, or a set containing only simple constraints, i.e. of the form $Comp(x_1). \ldots . Comp(x_n)$ *from* C where the x_i are variables (such a constraints are solved by replacing x_i by anything).

This completeness result can be proved in two steps. First, one shows that forbidding unification between a variable and another term is complete when one considers satisfiability of the system. The main point of the proof here is that if a variable appears in the knowledge set I of the Intruder, it also appears in a constraint for a knowledge set I', with $I' \subset I$. This constraint can be satisfied by I', hence it is possible to replace a unification between a term t in a constraint and a variable x in the knowledge set by another derivation leading to the satisfaction of the constraint.
The second step is to show that the restriction on the application of rules (20-26) is complete, which can be easily done once one can ensure there is no unification between a variable and another term. Further details are given in [10]. Note also that the proof of completeness and correctness of a system derived from the one presented here was given in [38] as a NP-completeness result for finding attacks in the case of a finite number of principals.

For example, from the Intruder's knowledge $\text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb))$, we may test if a term matching $\text{CRYPT}(\text{C}(\text{MR}(b), x_1), \text{SCRYPT}(sk(kbs), x_2))$ can be built:

$$
\begin{aligned}
& Comp(\text{CRYPT}(\text{C}(\text{MR}(b), x_1), \text{SCRYPT}(sk(kbs), x_2))) \\
& \qquad \textit{from } \text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb)) \ ; \ Id \\
\rightarrow^{(21)} \quad & Comp(\text{C}(\text{MR}(b), x_1)).\, Comp(\text{SCRYPT}(sk(kbs), x_2)) \\
& \qquad \textit{from } \text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb)) \ ; \ Id \\
\rightarrow^{(20)} \quad & Comp(\text{MR}(b)).\, Comp(x_1).\, Comp(\text{SCRYPT}(sk(kbs), x_2)) \\
& \qquad \textit{from } \text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb)) \ ; \ Id \\
\rightarrow^{(18)} \quad & Comp(x_1).\, Comp(\text{SCRYPT}(sk(kbs), x_2)) \\
& \qquad \textit{from } \text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb)) \ ; \ Id \\
\rightarrow^{(19)} \quad & Comp(x_1) \\
& \qquad \textit{from } \text{MR}(b).\text{SCRYPT}(sk(kbs), \text{NONCE}(Nb)) \ ; \ \sigma
\end{aligned}
$$

The test is successful, generating the substitution $\sigma : x_2 \leftarrow \text{NONCE}(Nb)$ in the last step. This is the only solution. In general, we have to explore all the possible solutions. Note that we stop, accepting the composition, as soon as there are only variables left in the *Comp* terms.

3.2 Decomposition of the Intruder's Knowledge

In Dolev-Yao's model, all the messages sent by the principals acting in the protocol are sent to the Intruder. The Intruder has then the possibility to decompose the terms he knows, including the last message, and build a new one, faked so as it looks like it has been sent by another principal (chosen by the Intruder). We define

$$C.UFO(\text{F}(t).C') \rightarrow C.\text{F}(t).UFO(C') \quad (27)$$

$$C.UFO(\text{C}(t_1, t_2).C') \rightarrow C.UFO(t_1.t_2.C') \quad (28)$$

$$C.UFO(\text{CRYPT}(t_1, t_2).C') \rightarrow C.\text{CRYPT}(t_1, t_2).UFO(\text{TEST}(\text{CRYPT}(t_1, t_2)).C') \quad (29)$$

$$C.UFO(\text{TEST}(\text{CRYPT}(t_1, t_2)).C') \rightarrow C.UFO(t_2.C')\sigma \quad \textit{if } A(t_1{}^{-1}, C, \sigma) \quad (30)$$

$$C.UFO(\text{SCRYPT}(t_1, t_2).C') \rightarrow C.\text{SCRYPT}(t_1, t_2).UFO(\text{TEST}(\text{SCRYPT}(t_1, t_2)).C') \quad (31)$$

$$C.UFO(\text{TEST}(\text{SCRYPT}(t_1, t_2)).C') \rightarrow C.UFO(t_2.C')\sigma \quad \textit{if } A(t_1, C, \sigma) \quad (32)$$

$$C.UFO(\text{XCRYPT}(t_1, t_2).C') \rightarrow C.\text{XCRYPT}(t_1, t_2).UFO(\text{TEST}(\text{XCRYPT}(t_1, t_2)).C') \quad (33)$$

$$C.UFO(\text{TEST}(\text{XCRYPT}(t_1, t_2)).C') \rightarrow C.UFO(t_2.C')\sigma \quad \textit{if } A(t_1, C, \sigma) \quad (34)$$

$$C.UFO(\text{TEST}(\text{XCRYPT}(t_1, t_2)).C') \rightarrow C.UFO(t_1.C')\sigma \quad \textit{if} A(t_2, C, \sigma) \quad (35)$$

$$C.UFO(\text{TABLE}(t_1, t_2).C') \rightarrow C.\text{TABLE}(t_1, t_2).UFO(C')\sigma \quad (36)$$

$$C.UFO(\text{TABLE}(t_1, t_2)^{-1}.C') \rightarrow C.\text{TABLE}(t_1, t_2)^{-1}.UFO(C')\sigma \quad (37)$$

$$C.UFO(\text{FUNC}(t_1, t_2).C') \rightarrow C.\text{FUNC}(t_1, t_2).UFO(C')\sigma \quad (38)$$

Figure 5: System Simplifying the Intruder's Knowledge.

a system that keeps in a predicate, *UFO*, the data that are not already treated by the Intruder, and moves the non-decomposable knowledge out of *UFO*. For the decryption of a cipher (but this should also apply to hash functions), we use a predicate (TEST) and a conditional rewrite rule. The resulting system described in Figure 5 only deals with *decomposing* the knowledge of the Intruder, where we are always using, together with the fourth rule, the equality $t^{-1-1} = t$.

We note this system $A(t, C, \sigma)$, a predicate that is true whenever the term t can be built from the knowledge C using a substitution σ. The system of Figure 4 shows that this predicate can be implemented with rewrite rules similar to those that are used to test if a principal can compose a message that matches the pattern of an awaited message.

3.3 Use of this Model for Flaws Detection

We can decompose the sequence of steps the Intruder uses to send a message:

1. For each active state s of the protocol, and for each principal p, the Intruder creates a new active state s' where he tries to send a message to the principal p: the principal p brings a pattern m that the Intruder's message should match. At the same time, p gives the pattern of the message t that the Intruder will receive if he succeeds in sending a message;

2. Second, the Intruder analyzes his knowledge and tests if he can compose a

message matching this pattern m;

3. If he can compose a message matching the pattern m, he goes back to step 1 with s' as active state. If not, this state s' fails and is removed from the active states.

3.4 Properties of our Model

In our model, the Intruder has to keep track of all the previously sent messages. Thus, we maintain a list of previously sent messages with the knowledge at the time the messages were sent:

$$l \stackrel{\text{def}}{=} (T_1 \ \mathit{from}\ C_1) : \ldots : (T_n \ \mathit{from}\ C_n)$$

This is used, for instance in the example of Section 3.1, to prove it is sound to substitute $\text{NONCE}(Nb)$ for x_2.
We also maintain a set of knowledge C representing the Intruder's knowledge evolution whenever he succeeds in sending an appropriate message. We model a protocol step with the rule:

$$(C, l) \rightarrow (C.t, l : (m \ \mathit{from}\ C))$$

Comparing this model to an execution model where an Oracle tells a message (ground term) that is accepted by the principal and where the Intruder has to verify he can send this message, this exhaustive exploration system turns out to be both *sound* (see the semantics given in [25]) and *complete* as long as we consider only a bounded number of principals [10, 38]. The variables here are untyped, thus allowing *messages of unbounded size* and the discovery of *type flaws*.

The *complexity* of our strategy is exponential in the number of roles. For parallel roles, the initial computation is exponential, but once done, the execution depends only on the number of constraints generated.

3.5 Example: Woo-Lam Protocol

The execution of the specification given in Figure 1 gives the following result:

$I \rightarrow b$	$: I, x_3$
$I \rightarrow b$	$: x_5$
$I(se) \rightarrow b$	$: x_4, \{I, x_3, Nb, x_2\}kbs$
$I \rightarrow b$	$: \{Nb\}x_2$

This corresponds to the messages sent by the Intruder, in the run of the protocol where I plays role A and b plays role B, for getting an authentication flaw between b and se. They are messages 1, 3, 5 and 7 of the protocol. Nb is the nonce generated by b.
Variables x_2, x_3, x_4 and x_5 represent parts of messages that are independent of this protocol run. b cannot check their value.

The most tricky message is the third. How can I compose $\{I, x_3, Nb, x_2\}kbs$ without knowing the key kbs? This is a constraint that the Intruder has to solve. Our model has concluded that this constraint is solvable and I can send this message. This conclusion is the result of the study of the Intruder's knowledge, of the protocol messages and of the roles and parallel roles. I cannot compose this part of the message, but he can get one that matches with it:

$A(b) \rightarrow I$	$: b, Na$
$I \rightarrow A(b)$	$: I, Nb$
$A(b) \rightarrow I$	$: \{b, I, Na, Nb\}kbs$
$I \rightarrow se$	$: \{b, I, Na, Nb\}kbs, \{b, I, Na, Nb\}kis$
$se \rightarrow I$	$: \{I, Na, Nb, Kai\}kbs, \{b, Na, Nb, Kai\}kis$

In the first part, he uses b as player of role A and Nb as its own nonce; in the second part, he uses the server se as a parallel role. The consequence is that the first part of the last message ($\{I, Na, Nb, Kai\}kbs$) matches with the message to build in the main run, $\{I, x_3, Nb, x_2\}kbs$. The result is a flaw because b thinks that the server has generated Kai for the protocol run where it has played role B; and the server se thinks it has generated Kai for b playing role A.

Thanks to our model of the Intruder, the role b as A and the parallel role se as S are not run. This is an important improvement for the efficiency of flaw detection (see Table 1), that is possible because the Intruder knows the nonces Na and Nb.

4 Experimentations

We give a few hints on how to use our system through two examples of protocol analysis taken from the literature. First, we study the Otway-Rees un-amended protocol, which has a type flaw leading to a secrecy flaw. Then, with the EKE protocol, we show how we deal with concurrent runs of the protocol. We also list the results obtained for other protocols that can be found in [11].
But first, let us give a short presentation of the prover used for looking for flaws from the rewrite rules generated by Casrul.

4.1 The Prover daTac

For studying the protocols with the rules generated by Casrul, we have used the theorem prover daTac[2], specialized for automated deduction in first-order logic with equality and associative-commutative (AC) operators. This last property is important, since we use an AC operator for representing the list of messages at a given state. Hence, asking for one message in this list consists in trying all the possible solutions. A more pertinent use is the possibility we have to express commutative properties of constructors. This enables us, for example, to express the commutativity of encryption in the RSA protocol.

[2] `http://www.loria.fr/equipes/cassis/softwares/daTac/`

The deduction techniques used by daTac are Resolution and Paramodulation [39]. They are combined with efficient simplification techniques for eliminating redundant information. Another important property is that this theorem prover is refutationally complete. Our model being complete with respect to the Dolev-Yao's model, we are certain to find all expressible flaws.

For connecting Casrul and daTac, we have designed a tiny tool, Casdat, running Casrul and translating its output into a daTac input file.

4.2 The Otway-Rees Protocol

The Casrul specification of this well-known protocol is given in Figure 6. To study

Protocol `Otway_Rees`;
Identifiers
A, B, S : *User*;
Kas, Kbs, Kab : *Symmetric_key*;
M, Na, Nb, X : *Number*;
Knowledge
A : B, S, Kas;
B : S, Kbs;
S : A, B, Kas, Kbs;
Messages
1. $A \rightarrow B$: $M, A, B, \{Na, M, A, B\}Kas$
2. $B \rightarrow S$: $M, A, B, \{Na, M, A, B\}Kas$, $\{Nb, M, A, B\}Kbs$
3. $S \rightarrow B$: $M, \{Na, Kab\}Kas, \{Nb, Kab\}Kbs$
4. $B \rightarrow A$: $M, \{Na, Kab\}Kas$
5. $A \rightarrow B$: $\{X\}Kab$
Role
$A[A : a; B : b; S : se; Kas : kas]$,
$B[B : b; S : se; Kbs : kbs]$,
$S[A : a; B : b; S : se; Kas : kas; Kbs : kbs]$;
Intruder *Divert*, *Impersonate*;
Intruder_knowledge a;
Goal *Secrecy_Of* X;

Figure 6: Otway-Rees Protocol.

this protocol, we only have to compile this specification to daTac rules and to apply the theorem prover daTac on the generated file, leaving the result in the `Otway-Rees.exe` file:

```
% casdat Otway-Rees.cas
% datac -i Otway-Rees.dat -r o Otway-Rees.exe
```

The trace of an execution is quite hard to analyze if one is not familiar with the techniques implemented in daTac, but hopefully, the result is the sequence of derivations leading to the discovery of the flaw *(in 6s)*:

```
> Inference steps to generate the empty clause:
  60 = Resol(1,56)      60 = Simpl(11,60)      ...   60 = Simpl(34,60)
  63 = Resol(5,60)      63 = Simpl(11,63)      ...   63 = Simpl(30,63)
  66 = Resol(44,63)     66 = Simpl (14,66)     ...   66 = Simpl (52,66)
  66 = Clausal Simpl({45},66)
```

Now, we just have to look at the given trace to figure out the scenario that leads to the secrecy flaw. Only the clauses generated by a resolution step and fully simplified matter.

The first one is pretty simple, since it is nothing but the first principal sending its first message. All the simplifications following correspond to the decomposition of this message to Intruder's knowledge. We thus have:

$$a \rightarrow \_ \quad : M, a, b, \{Na, M, a, b\}kas$$

The second resolution ($63 = Resol(5, 60)$) is much more exotic, since it is the reception of the message labelled 4 in the protocol by principal a. Using the protocol's specification, it is first read as:

$$\begin{array}{ll} a \rightarrow \_ & : M, a, b, \{Na, M, a, b\}kas \\ \_ \rightarrow a & : M, \{Na, x5\}kas \\ a \rightarrow \_ & : \{X\}x5 \end{array}$$

At this point, we can only say that the Intruder has tried to send to the principal a a message *matching* $M, \{Na, x5\}kas$. He has no choice but to unify ($66 = Resol(44, 63)$) the term yielded after the first message with the required pattern. Now, the sequence of messages becomes:

$$\begin{array}{ll} a \rightarrow \_ & : M, a, b, \{Na, M, a, b\}kas \\ \_ \rightarrow a & : M, \{Na, M, a, b\}kas \\ a \rightarrow \_ & : \{X\}(M, a, b) \end{array}$$

The Intruder has proved that he can send a term matching the pattern of the awaited message, so we can go on to the next step, the decomposition of the second message sent by a ($66 = Simpl(52, 66)$). But after that, decomposing what he knows, the Intruder finds himself knowing X, that should have remained secret. The last move (Clausal Simplification) stamps this contradiction out, thus ending the study of this protocol.

4.3 The Encrypted Key Exchange *(EKE)* Protocol

We shall now study the EKE protocol, known to have a parallel authentication attack. The Casrul specification of this protocol is given in Figure 7.

Protocol EKE;
Identifiers
A, B : *User*;
Na, Nb : *Number*;
Ka : *Public_key*;
P, R : *Symmetric_key*;
Knowledge
A : B, P;
B : P;
Messages
1. $A \rightarrow B$: $\{Ka\}P$
2. $B \rightarrow A$: $\{\{R\}Ka\}P$
3. $A \rightarrow B$: $\{Na\}R$
4. $B \rightarrow A$: $\{Na, Nb\}R$
5. $A \rightarrow B$: $\{Nb\}R$
Role
$A[A : a; B : b; P : p]$;
Parallel
$B[B : a; P : p; R : re]$;
Secret p, re;
Intruder *Divert*, *Impersonate*;
Intruder_knowledge ;
Goal *A authenticates B on Nb*;

Figure 7: Encrypted Key Exchange Protocol.

The trace of the execution does also, in this case, lead to a flaw. This time, this is an authentication flaw:

> Inference steps to generate the empty clause:

87 = Resol(1,85)	87 = Simpl(9,87)	...	87 = Simpl(76,87)
88 = Resol(2,87)	88 = Simpl(9,88)	...	88 = Simpl(31,88)
89 = Resol(81,88)	89 = Simpl(49,89)	...	89 = Simpl(4,89)
90 = Resol(4,89)	90 = Simpl(9,90)	...	90 = Simpl(31,90)
92 = Resol(78,90)	92 = Simpl(27,92)	...	92 = Simpl(70,92)

92 = Clausal Simpl({33},92)

We can study in deeper details this trace in order to find the scenario of the attack. Since the principal a appears in two instances, we will give, right after its name, a string (*seq* or //) that identifies the principal either as the one defined in the `Role` or in the `Parallel` field. First of all, the principal of the `Role` field starts with sending its first message:

$a(seq) \rightarrow \_$: $\{Ka\}p$

Here, $a(seq)$ has to generate a fresh key Ka. Then, the Intruder tries to send a message to the principal a defined in the `Role` field:

$$\begin{array}{lll} a(seq) & \rightarrow \_ & : \{Ka\}p \\ \_ & \rightarrow a(seq) & : \{\{x_1\}Ka\}p \\ a(seq) & \rightarrow \_ & : \{Na\}x_1 \end{array}$$

The Intruder now has to prove he could send the message $\{\{x_1\}Ka\}p$. He cannot compose this message using his current knowledge, but he can have a term unifying with this by interacting with $a(//)$. This is what is done in the next resolution:

$$\begin{array}{lll} a(seq) & \rightarrow \_ & : \{Ka\}p \\ \_ & \rightarrow a(//) & : \{Ka\}p \\ a(//) & \rightarrow \_ & : \{\{re\}Ka\}p \\ \_ & \rightarrow a(seq) & : \{\{re\}Ka\}p \\ a(seq) & \rightarrow \_ & : \{Na\}re \end{array}$$

Now, the Intruder can go on like this until he arrives at this point:

$$\begin{array}{lll} a(seq) & \rightarrow \_ & : \{Ka\}p \\ \_ & \rightarrow a(//) & : \{Ka\}p \\ a(//) & \rightarrow \_ & : \{\{re\}Ka\}p \\ \_ & \rightarrow a(seq) & : \{\{re\}Ka\}p \\ a(seq) & \rightarrow \_ & : \{Na\}re \\ \_ & \rightarrow a(//) & : \{Na\}re \\ a(//) & \rightarrow \_ & : \{Na, Nb\}re \\ \_ & \rightarrow a(seq) & : \{Na, Nb\}re \\ a(seq) & \rightarrow \_ & : \{Nb\}re \end{array}$$

The principal $a(seq)$ has finished its part of the protocol, and it is possible to see if Nb was a nonce created by a principal b communicating with a. This is not the case here, so we reach an authentication flaw, as indicated by the last clausal simplification ($92 = \mathit{Clausal\ Simpl}(33, 92)$). The total time of execution is a few seconds.

One can note that, in this case, we perform much better than in [8], where we used two instantiations of both roles running concurrently. The time needed to reach this flaw was around 4 minutes, and the number of states explored before reaching the flaw was around 200 (here, only 7!).

4.4 Some of the Other Protocols Already Studied

The study of the protocols given in Table 1 is straightforward, and is done in an automatic way similar to the one used for the two previously detailed examples. The table shows the difference of timings between the original version of Casrul,

Protocol	Lazy	Lazy //	Kind of Flaw
Andrew Secure RPC	7s	7s	Authenticate Flaw
Encrypted Key Exchange	268s	3s	Authenticate Flaw
Kehne-Schoene-Langendorf	69s	20s	Authenticate Flaw
Kao Chow (unamended)	24s	2s	Authenticate Flaw
Needham Schroeder Conventional Key Protocol	11s	11s	Compromised Key/Authenticate Flaw
Needham Schroeder Public Key Protocol	18s	3s	Authenticate Flaw
Neumann-Stubblebine (initial part)	8s	2s	Authenticate Flaw
Neumann-Stubblebine (repeated part)	34s	4s	Authenticate/Secrecy Flaw
Otway Rees	6s	6s	Authenticate Flaw
RSA protocol	2s	2s	Secrecy Flaw
SPLICE	53s [3]		Authenticate Flaw
Intruder impersonates client		6s	Authenticate Flaw
Intruder impersonates server		26s	Authenticate Flaw
Davis Swick Authentication Protocol	181s	18s	Authenticate Flaw
TMN (compromised key flaw)	67s	67s	Short Term Secrecy Flaw
TMN (authentication flaw)	41s	41s	Authenticate Flaw
Woo-Lam $\pi - (3)$ Protocol	10s	10s	Authenticate Flaw
Woo-Lam π Protocol	59s	1s	Authenticate Flaw
Woo-Lam Mutual Authentication Protocol	512s	30s	Authenticate Flaw

Table 1: Results obtained with Casrul+daTac for several cryptographic protocols.

with the lazy model of the Intruder, and the new one that permits in addition to use parallel roles. All those results have been obtained with a PC under Linux (Pentium 3, 800 MHz, 128 Mb RAM).

All the flaws have been found *automatically* by trying several roles and/or parallel roles; the knowledge of existing flaws has sometimes guided us for limiting those roles, for avoiding useless computations.

We point out that, in all the protocols studied up to now, we have, every time, obtained an attack when there is one, and we have not found any attack when

[3] In this case, both errors are found. We give the time for finding the first flaw (Intruder impersonates the client).

no attack was reported in the literature. One shall also note that the number of explored clauses, using a breadth-first search strategy, is always smaller than a few hundreds. This demonstrates that our lazy strategy for the Intruder, represented by the rewrite rules produced by Casrul, can be turned into a time and space efficient procedure, independently of the tool used afterwards for finding flaws. This result is obtained *without limiting the number of possible messages*, contrarily to many other tools. This is due to the fact that we do not type informations, i.e. we use a pure symbolic calculus.

Our objective is to continue to improve our verification technique, and to study more and more protocols. But we are also working on the positive verification of protocols. For instance, we have already studied some parts of the SET protocol of VISA and Mastercard, and we are currently studying other parts of this protocol. Since the new version of Casrul permits it, we are also studying protocols that use composed keys, such as SSL (Secure Sockets Layer) [22].

5 Discussion and Related Works

In this section, we summarize the results presented in this paper, compare Casrul with other protocol compilers, and compare our technique with other protocol analysis methods.

5.1 Summary of our Results

We believe that a strong advantage of our method (shared with Casper and CAPSL) is that it is not ad-hoc: the translation is working without user interaction for a wide class of protocols and therefore does not run the risk to be biased towards the detection of a known flaw. Protocols specification are usually given together with the knowledge of participants. Thus, the only part of the verification process where a user of Casrul has to be imaginative is the description of the environment of execution. The Intruder can always be given the strongest abilities `divert` and `impersonate`. And it is now possible, after a work by M. Bouallagui and J. Himanshu [7], to only specify the number of participants instead of their instances. This permits to incrementally search for flaws in the protocol by increasing the number of participants, thereby reducing user's intervention to the important part of the protocol, that is the specification of role instances.

Another important advantage of our method is the ability to handle infinite state models, and to be closer to the original Dolev-Yao model [17] because of our dynamic lazy intruder model.

A limitation of our method is that two fresh information are always different. This means that we do not consider the case where two nonces, for example, are equal by accident. Another limitation is that we cannot consider Diffie-Hellman protocols because we cannot deal with the exponential yet. Otherwise, any protocol that can be expressed in our specification can be studied by our method.

In the following, we summarize the results presented in this paper, depending on the limitation or not of the number of sessions.

5.1.1 Unbounded Number of Sessions

For studying the case of an unbounded number of runs of a protocol, we bound both the size of messages and the number of different nonces in order to ensure termination. The result of the analysis (that uses parallel roles) is expressed as rewrite rules on the knowledge of the Intruder. Thus, no flaw is searched on the principals which can run an unbounded number of concurrent executions. This enables us to study authentication flaws when an unbounded number of concurrent executions are involved in a terminating system. This method is very efficient and complete: if a flaw exists, it will find it; if it terminates without finding any flaw, the protocol is correct. The main drawback is that it is not sound: the abstraction on nonces implies that the attacks found may be fake.

5.1.2 Bounded Number of Sessions

When considering a bounded number of runs of a protocol, we do not need to bound the size of messages. In that case, our method terminates, is sound and is complete for finding flaws.

5.2 Other Protocols Compilers

Let us compare Casrul with the most well-known three similar tools. Casper [26] and CVS [19] are compilers from protocols descriptions to process algebra (CSP for Casper, and SPA for CVS). Both have been applied to a large number of protocols. The Casper approach is oriented towards finite-state verification by model-checking with FDR [36]. The syntax we use for Casrul is similar to the Casper syntax for protocols description. However, our verification techniques, based on theorem proving methods, relax many of the strong assumptions for bounding the information (to get a finite state model) in model checking. For instance, our technique based on narrowing ensures directly that all randomly generated nonces are pairwise different. This guarantees the freshness of information over different executions.
Casper and CVS are similar at the specification level, but CVS has been developed to support the so-called Non-Interference approach [21] to protocol analysis. This approach requires the tedious extra-work of analyzing interference traces in order to check whether they are real flaws. Our verification technique is also based on analyzing traces, but it captures automatically the traces corresponding to attacks.

CAPSL [29] is a specification language for authentication protocols in the flavor of Casper's input. There exists a compiler from CAPSL to an intermediate formalism, CIL, which may be converted to an input for automated verification tools such as Maude, PVS, NRL [28]. A CIL basically contains a set of rules expressing the state transitions of a protocol. Initially, every protocol rule from a standard notation gives rise to two transition rules, one for the sender and one for the receiver. The

transition rules can be executed by multiset and standard pattern matching. The rewrite rules produced by our compilation is also an intermediate language, which has the advantage to be an idiom understood by many automatic deduction systems. In our case, we have a single rule for every protocol message exchange, as opposite to the initial CIL which has two rules. More recently, an optimized version of CIL has been defined [16]: it generates one rule per protocol rule, as we do.
Another difference between Casrul and CAPSL is that it is not possible to define an initial state in CAPSL. Thus, back-ends of the CAPSL parser need to define this information separately. Note this is not a limitation for Casrul users, since if initial states currently have to be defined, the receive/send rules of the protocol are independent w.r.t. this initial state. Thus, it can be dropped by tools having no use for it.
Another important difference between the CIL and the rules generated by Casrul is that the evolution of the knowledge of the principals is automatically handled by our system. Hence, we get immediately an optimized version of the rewrite system, which minimizes the number of transitions for verification.
CAPSL takes the advantage on Casrul when it comes to expressiveness. It is already possible to specify choice points and to have a modular specification of protocols whereas this is not implemented yet in Casrul. The extension MuCAPSL of CAPSL also permits to define group protocols.

5.3 Other Protocol Analysis Methods

A lot of other methods have been implemented to verify or find attacks on cryptographic protocols. In this discussion, we focus on automated methods, which can be divided into three main categories.

First, there are methods where an attack is a search starting from an initial state of the protocol. Generally speaking, search for attacks in this setting is most of the time both fast and sound. The drawback of these methods being that no conclusion can be drawn from the absence of attack when starting from an initial state. One can gather in a second category model-checkers aiming at proving correctness of a protocol without starting from an initial state. Undecidability of search for attacks in an unbounded number of executions, where either message size or number of different nonces is not bounded, makes these methods either not sound or not terminating, or both. A third category of methods relies on *abstraction*, and often also on the use of tree automata, to model the behavior of the Intruder and of the principals. These methods permit to prove the correctness of a protocol, but the abstraction done can also lead to the discovery of false flaws.

5.3.1 Model-checking with an Initial State

Lowe uses, together with Casper, the FDR model-checker. The aim of the analysis, in this case, is to prove that the protocol is a refinement of a security property. If a counter-example (*i.e.* an attack) is found, it is given as the result of the computation.

Time comparisons with Casper are difficult, since it is proprietary system. However, one can note that its use by Lowe has permitted the discovery of numerous flaws in the protocols given in the Clark and Jacob library [11, 18]. The main drawback of this tool is that possible type flaws have to be specified by the user.

The Murφ [32] tool integrates a compiler for a high-level protocol specification language relying on guarded commands with a model-checker using aggressive state space reduction through the use of symmetries. While this reduction permits a fast analysis of the protocols, it has been reported that some attacks found using this tool are fake, because they relied on the confusion of nonces created by two different principals. One shall note that the use of symbolic reduction achieves the same state space reduction as the use of symmetries, but in our case the classes of executions are sound and no false attack is found.

A third tool, among those starting from an initial state, is the one developed by Lugiez, Amadio and Vanackère [1]. The lazy intruder strategy is a generalization of their symbolic reduction that permits to handle type flaws and composed keys. It is, to our knowledge, one of the first methods using constraints to analyze cryptographic protocols.

5.3.2 Model-checking without an Initial State

The method proposed by Debbabi et al. [13] is a tool using symbolic constraints but in a somewhat different fashion. The capacities of the Intruder and of the principals are abstracted as inference rules. The inference system is non-terminating, but an algorithm is given permitting to transform this inference system into a terminating one. When this algorithm terminates, it is possible to conclude whether the protocol is flawed. One shall note that in this system, no bound is assumed neither on the message size nor on the number of different nonces. The drawback of this method, however, is that the algorithm used permits only to find special kinds of attacks, *i.e.* those respecting an order on the application of inference rules. Moreover, the method relies on some properties of messages of the protocol to be used, and it is not clear whether it can be extended to generic protocols definitions.

A very similar method is used by Athena [41]. This tool recursively constructs the set of possible executions of the protocol without bound on the number of the executions. Typing permits to bound the size of messages, but the number of different nonces is unbounded, thus implying an infinite state space. The major improvement of this method is that halting conditions can be specified by the user, hence permitting more flexibility in the analysis. However, it seems that the performance of this tool relies on the strategy of the exploration of the state space.

A last tool using *backward search* is described by Blanchet in [5]. The main part of the analysis algorithm builds a finite number of rules describing the possible actions of an intruder between two messages. If this construction terminates, it is possible to analyze an unbounded number of simultaneous runs of the protocol. However, in this case, the number of different nonces is finite. While such construction is very useful for the analysis of secrecy properties of a protocol, it is unsuited

for the analysis of authentication, since messages from one execution of any protocol can be replayed later, the nonces being identical. Moreover, the abstraction on nonces leads to an unsound system: some fake secrecy flaws can be found.

5.3.3 Abstraction and Tree Automata

Another possibility for automatic analysis of cryptographic protocols is to use tree automata in conjunction with an abstraction on the protocol. Along this line, one can note the work by Klay and Genet [23] or by Monniaux [33], where the set of messages that the Intruder can compose from a finite set of knowledge is over-approximated by a regular tree language. Such systems can find attacks or verify systems where only a finite number of executions of the protocol is considered. Their drawback is that even in this case, they are not sound: the over-approximation of the knowledge of the Intruder may lead to false attacks. These approaches were extended using an intricate setting in [24] to the case of an unbounded number of parallel executions of the protocol. Logical properties are used to characterize the knowledge of the Intruder, and the states of the automata are logical formulas. One important point is that principals running the protocol in parallel are also used as accomplices of the Intruder. However, further investigation is needed to compare our system with Goubault-Larrecq's.

6 Conclusion

We have designed and implemented in Casrul a compiler of cryptographic protocols, transforming a general specification into a set of rewrite rules. The user can specify some strategies for the verification of the protocol, such as the number of simultaneous executions, the initial knowledge, the general behavior of the Intruder, and the kind of attack to look for.
The transformation to rewrite rules is fully automatic and high level enough to permit further extensions or case specific extensions. For example, one can model specific key properties such as key commutativity in the RSA protocol.

The protocol model generated is general enough to be used for various verification methods, as exemplified in the European Union project AVISPA[4] [2].
In our case, we have used narrowing with the theorem prover daTac. The AC properties proposed by this system permit us to handle general rewrite rules, simplifying the translation from the Casrul output to the daTac input. The timings obtained for verifying protocols could be much better, but using a general theorem prover such as daTac shows how efficient are the rules generated by Casrul. This is confirmed by the large number of protocols that have been verified entirely automatically, the most well-known being listed in Table 1. Recently, we have even found a new attack on the Denning-Sacco symmetric key protocol (see [10]).

[4] `http://www.avispa-project.org/`

We are currently using all the expressiveness of Casrul for studying large protocols, such as SET and SSL, and the first results are very positive. We also plan to work on the study of an unbounded number of sequential executions, which should be useful in the study of One Time Password protocols, for example. In this case, each session would have its own nonces. But, because of undecidability results [20], we would have to restrain our model in order to keep implementability.

Acknowledgments: We would like to thank the referees for their numerous interesting comments that helped us to improve this article.

References

[1] R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. Technical report, INRIA Research Report 3915, Marseille (France), 2000.

[2] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Vigano, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *14th International Conference on Computer Aided Verification, CAV'2002*, LNCS 2404, pages 349–353, Copenhaguen (Denmark), 2002. Springer.

[3] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, Houston, Texas, 2002.

[4] D. Basin. Lazy Infinite-State Analysis of Security Protocols. In *Secure Networking — CQRE'99*, LNCS 1740, pages 30–42, Düsseldorf (Germany), 1999. Springer.

[5] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop*, 2001.

[6] D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *10th IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Society, 1997.

[7] M. Bouallagui and J. Himanshu. Automatic Generation of Session Instances. Technical report, LORIA, Nancy (France), 2003.

[8] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols (short paper). In *Proceedings of ASE-2001: The 16th IEEE Conference on Automated Software Engineering*, San Diego (CA), 2001. IEEE CS Press.

[9] Y. Chevalier and L. Vigneron. Towards Efficient Automated Verification of Security Protocols. In *Proceedings of the Verification Workshop (VERIFY'01) (in connection with IJCAR'01), Università degli studi di Siena, TR DII 08/01*, pages 19–33, 2001.

[10] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *14th International Conference on Computer Aided Verification, CAV'2002*, LNCS 2404, pages 324–337, Copenhaguen (Denmark), 2002. Springer.

[11] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.

[12] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symposium (SAS)*, LNCS 2477, pages 326–341, Madrid (Spain), 2002. Springer.

[13] M. Debbabi, M. Mejri, M. Tawbi, and I. Yahmadi. Formal Automatic Verification of Authentication Cryptographic Protocols. In *Proceedings of the First IEEE International Conference on Formal Engineering Methods (ICFEM'97)*, 1997.

[14] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *LICS' Workshop on Formal Methods and Security Protocols*, 1998.

[15] G. Denker and J. Millen. CAPSL Intermediate Language. In *FLOC's Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[16] G. Denker, J. Millen, J. Kuester-Filipe, and A. Grau. Optimizing Protocol Rewrite Rules of CIL Specifications. In *13th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 2000.

[17] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29:198–208, 1983. Also STAN-CS-81-854, 1981, Stanford U.

[18] B. Donovan, P. Norris, and Lowem G. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[19] A. Durante, R. Focardi, and R. Gorrieri. A Compiler for Analysing Cryptographic Protocols Using Non-Interference. *ACM Transactions on Software Engineering and Methodology*, 9(4):489–530, 2000.

[20] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *FLOC's Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[21] R. Focardi, A. Ghelli, and R. Gorrieri. Using Non Interference for the Analysis of Security Protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[22] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol — Version 3.0, 1996. `draft-freier-ssl-version3-02.txt`.

[23] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *17th International Conference on Automated Deduction*, LNCS 1831, pages 271–290, Pittsburgh (PA, USA), 2000. Springer.

[24] J Goubault-Larrecq. A Method for Automatic Cryptographic Protocol Verification. In *Proc. 15 IPDPS 2000 Workshops, Cancun, Mexico*, LNCS 1800, pages 977–984. Springer, 2000.

[25] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Logic for Programming and Automated Reasoning*, LNCS 1955, pages 131–160, St Gilles (Réunion, France), 2000. Springer.

[26] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[27] C. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. *Journal of Computer Security*, 1(1):5–36, 1992.

[28] C. Meadows. The NRL Protocol Analyzer: an Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[29] J. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.

[30] J. Millen and H.-P. Ko. Narrowing Terminates for Encryption. In *PCSFW: Proceedings of the 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[31] J. Millen and V. Shmatikov. Constraint Solving for Bounded-process Cryptographic Protocol Analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175. ACM SIGSAC, 2001.

[32] J. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, pages 141–154. IEEE Computer Society, 1997.

[33] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Sixth International Static Analysis Symposium (SAS'99)*, LNCS 1694. Springer, 1999.

[34] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[35] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[36] A. W. Roscoe. Modelling and Verifying Key-exchange Protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.

[37] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[38] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop.* IEEE, 2001.

[39] M. Rusinowitch and L. Vigneron. Automated Deduction with Associative-Commutative Operators. *Applicable Algebra in Engineering, Communication and Computation*, 6(1):23–56, 1995.

[40] B. Schneier. *Applied Cryptography.* John Wiley, 1996.

[41] D. Song, S. Berezin, and A. Perrig. Athena, a Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9((1,2)):47–74, 2001.

[42] C. Weidenbach. Towards an Automatic Analysis of Security Protocols. In *16th International Conference on Automated Deduction*, LNCS 1632, pages 378–382, Trento (Italy), 1999. Springer.