Contribution à l'étude du raisonnement temporel. Résolution avec contraintes et application à l'abduction en raisonnement temporel

Nicolas Chleq

Doctorat de l'Ecole Nationale des Ponts et Chaussées

Mathématiques, informatique

présenté par

Nicolas CHLEQ

Contribution à l'étude du raisonnement temporel

Résolution avec contraintes et application à l'abduction en raisonnement temporel

soutenu le 12 janvier 1995

devant le jury composé de :

MM.	René LALEMENT	Président
	Malik GHALLAB Gérard LIGOZAT	Rapporteurs
	Bruno GAUME Patrick TAILLIBERT	Examinateurs
	Bertrand NEVEU	Directeur

Je tiens à remercier ici

Malik Ghallab et *Gérard Ligozat* qui ont accepté, malgré leur emploi du temps chargé, d'être rapporteurs de ce travail et de prendre le temps de le juger ;

René Lalement, *Bruno Gaume* et *Patrick Taillibert* qui me font l'honneur de participer à ce jury ;

Bertrand Neveu qui m'a accueilli dans l'équipe SECOIA à l'INRIA et m'a toujours laissé une grande liberté pour le choix et l'organisation de mon travail ;

Sabine Moisan et *Olivier Corby* qui ont su supporter avec calme et patience les multiples séances de (re)lecture que je leur ai imposées ;

Hany Tolba pour sa bonne humeur et ses conseils pertinents ;

l'équipe de la « doc » pour leur disponibilité, leur compétence et leur aide efficace en matière de recherche bibliographique ;

et enfin, tous les membres des équipes SECOIA, ACACIA et ORION pour leur soutien amical et l'ambiance agréable à laquelle ils contribuent.

Table des matières

Introduction

Dans le cadre des recherches en Intelligence Artificielle, le raisonnement est un sujet très étudié qui joue un rôle important dans la mise en œuvre des outils et des méthodes de l'intelligence artificielle. La capacité à faire du raisonnement de manière automatisée suppose la mise en place d'une combinaison d'un ou de plusieurs formalismes de représentation, de méthodes de résolution de problèmes et d'algorithmes de raisonnement et d'inférence qui travaillent sur la base du formalisme de représentation.

Parmi tous les facteurs qui peuvent concourir à la pertinence et à la qualité d'une modélisation, le temps joue un rôle primordial. Il intervient à différents niveaux du monde réel et de sa modélisation : comme une quantité mesurable, sensible pour tous les acteurs de l'univers ; comme un facteur participant à l'ordonnancement des liens causaux ; et surtout comme un facteur de l'évolution du monde et ceci de manière — en général — irréversible. De plus, il conditionne la notion de vérité, et par là même, le processus d'interprétation qui établit le lien entre l'univers externe et sa représentation dans un système à base de connaissances.

Cet aspect multiforme du temps dans un système intelligent suppose une modélisation adaptée, qui passe par la mise en place de différents formalismes de représentation, de différentes techniques de réalisation de l'inférence, suivant le niveau souhaité pour la modélisation du temps et son utilité envisagée dans le système intelligent à réaliser.

Pour illustrer notre propos de manière concrète, voici quelques exemples de problèmes traités par l'intelligence artificielle et qui sont reconnus comme nécessitant des capacités de raisonnement temporel :

- la *prédiction* (Dean & Boddy, 1988; Nebel & Bäckström, 1991) cherche à construire à partir d'un ensembles de règles causales et d'une description initiale de l'univers une « histoire » de celui-ci. Ce problème est souvent formalisé comme le calcul des conséquences certaines ou possibles d'un ensemble d'événements. Dans un formalisme de ce type, où l'on dispose d'un ensemble d'événements partiellement ordonnés, ce problème est le plus souvent NP ;

- la *planification* (Allen, 1991) où le but est de construire un ensemble d'actions afin que leur réalisation conduise le système considéré dans un état particulier.

Suivant les cas, on est intéressé soit par des actions totalement ordonnées, soit partiellement ordonnées ;

- la *reconnaissance de situations* (Mounir-Alaoui, 1990; Dousson, Gaborit, & Ghallab, 1993) où le problème est d'identifier, parmi un ensemble de situations typiques, celle du système modélisé. Cette reconnaissance permet par exemple de guider l'élaboration d'actions correctrices sur le système.

Beaucoup de travaux ont été réalisés sur le raisonnement temporel. Du point de vue des formalismes de représentation, des logiques temporelles de toutes natures ont été développées, car l'outil logique est un cadre suffisamment général qui possède un cadre formel suffisamment puissant pour être utilisé comme base de développement. Ces formalismes ont généralement pour vocation de modéliser des aspects complexes du temps à un haut niveau d'abstraction. Par opposition, beaucoup d'autres travaux se sont concentrés sur des formalismes plus restreints mais susceptibles d'être mis en œuvre par l'usage de techniques algorithmiques adaptées : c'est le cas des langages de relations temporelles, où l'on exprime les possibilités de placement relatif d'éléments comme les instants ou les intervalles de temps. On utilise souvent pour ceci des techniques de recherche opérationnelle ou de programmation par contraintes.

Malgré tout, on constate qu'il existe un écart significatif entre l'expressivité des formalismes pour lesquels une ou plusieurs procédures de preuve existent, ou plus généralement, pour lesquel un mécanisme d'inférence, au sens large, a pu être développé, et l'expressivité des formalismes de haut niveau, ces derniers n'ayant que peu de mécanismes de raisonnement associé. Les premiers formalismes sont ceux des langages de relations temporelles, les seconds comprennent en particulier les logiques temporelles.

Ceci a motivé notre approche, où nous nous sommes intéressés à deux aspects de la mise en œuvre du raisonnement temporel. Le premier point était de définir un mécanisme d'inférence propre dont l'utilité soit manifeste pour le raisonnement temporel, et deuxièmement, de développer sa mise en œuvre effective dans le cadre d'un mode de raisonnement dont l'intérêt n'est pas négligeable : le raisonnement abductif. En effet, ce type de raisonnement cherche à inférer, à partir d'une théorie d'un domaine et d'une observation particulière, une explication de cette observation, c'est à dire ce qu'il faudrait rajouter à la théorie pour en déduire l'observation. Il s'agit donc d'un raisonnement qui est plus puissant qu'une simple réponse par « oui » ou « non » à une requête. Dans le cadre du raisonnement temporel, l'abduction présente le double intérêt de proposer une solution au problème de la persistance temporelle, et de constituer une technique utilisable pour la planification.

Contribution

Dans un premier temps, nous nous sommes basés sur un formalisme temporel particulier dont l'expressivité est intéressante, les logiques temporelles réifiées,

et pour lequel la mise au point de mécanismes de raisonnement est un problème difficile n'ayant pas l'objet de choix définitifs. Dans cette optique, nous avons identifié le principe de résolution avec contraintes, développé dans le cadre de travaux en démonstration automatique, comme une méthode apte à la réalisation de mécanismes d'inférences pour le raisonnement temporel. La caractéristique de ce genre de mécanismes d'inférence est qu'ils autorisent l'utilisation des techniques spécialisées développées pour les formalismes de relations temporelles, dans le cadre de systèmes déductifs basés sur la résolution. Un des apports est l'efficacité à bas niveau du processus d'inférence, car ce qui est difficile à réaliser avec la résolution est fait plus aisément à l'aide d'algorithmes spécifiques sur les relations temporelles.

Sur la base de l'intérêt présenté par ce principe de résolution, nous avons ensuite étudié la mise au point d'une procédure de raisonnement abductif adaptée au raisonnement temporel. Celle-ci est inspirée de travaux sur l'extension de la programmation logique à l'abduction, c'est à dire principalement à l'extension de la résolution SLD (la plus répandue des procédures de preuve en programmation logique) à la génération d'hypothèses. De ce fait, l'on peut voir notre travail comme une fusion entre la programmation logique abductive et la programmation logique avec contraintes.

Plan de lecture

Le chapitre 1 présente un état de l'art du raisonnement temporel. Dans l'optique de notre travail, nous nous sommes concentrés sur deux grandes classes de formalismes et de techniques développés pour le raisonnement temporel: les logiques temporelles et les systèmes de contraintes temporelles. Dans le chapitre 2, nous explorons la possibilité d'un mécanisme d'inférence simple et efficace basé sur le principe de résolution et adapté au raisonnement temporel. Nous décrivons en particulier comment la résolution avec contraintes peut être utilisée dans ce but, les avantages et les limites de cette approche.

Le chapitre 3 marque le début de la deuxième partie de notre travail portant sur l'abduction dans le raisonnement temporel. Ce chapitre effectue une revue rapide du raisonnement abductif, justifie son intérêt pour le raisonnement temporel et fournit les éléments de choix pour la conception de notre procédure abductive. Cette procédure fait l'objet du chapitre 4 où nous commençons par motiver les choix qui président à sa conception. Partant de la base des travaux en programmation logique abductive, nous expliquons point par point les éléments de notre procédure. Après l'avoir défini formellement, nous présentons ensuite différents exemples de son utilisation et de ses capacités, en mettant l'accent sur la planification.

Pour finir, le chapitre 5 explore quelques aspects de l'implantation de cette procédure: le cadre général, la gestion de contraintes temporelles quantitatives pour lequel nous présentons un algorithme incrémental de complexité intéressante. Puis nous présentons les grandes lignes d'une machine abstraite inspirée de la WAM en

vue de son usage pour l'exécution de notre procédure.

Les annexes de ce mémoire présentent différentes notions qui sont utilisées au cours de la présentation de notre travail. L'annexe A recense les notations utilisées dans ce mémoire. L'annexe B présente quelques bases de logique classique, en particulier les notions d'interprétation. L'annexe C introduit la programmation logique et constitue une présentation de ce domaine suffisante pour servir d'introduction à la programmation logique abductive telle qu'elle est décrite dans les chapitres 3 et 4 de ce mémoire. L'annexe D présente le cadre général de la programmation logique avec contraintes.

Chapitre 1

Le raisonnement temporel

Ce chapitre constitue une introduction au raisonnement temporel. Il est divisé en deux parties. La première partie décrit différents formalismes mis au point pour représenter et raisonner sur le temps. L'aspect opérationnel de ces formalismes, et plus généralement la mise en œuvre du raisonnement temporel, fait l'objet de la deuxième partie de ce chapitre. Y sont décrites d'autres approches, en général plus restreintes quand à leurs capacités expressives, mais pour lesquelles il existe des procédures d'inférences ou de décision.

1.1 Formalismes de représentation du temps

Une première approche du raisonnement temporel, que l'on pourrait qualifier d'approche simpliste, consiste à confondre systématiquement le temps du système modélisé (temps dit *externe*) avec le temps interne de l'exécution du programme. L'inconvénient majeur de cette approche est d'empêcher toute distinction entre une révision des données dûe à une évolution dans le système extérieur, d'une révision dûe à la correction d'une information erronée produite au cours du raisonnement. Cette confusion est source de problèmes complexes qui ne facilitent pas la mise au point d'une base de connaissances qui aurait ainsi été étendue pour prendre en compte le temps.

Un raisonnement où le temps intervient doit identifier celui-ci comme un facteur autonome et une connaissance à part entière. Néanmoins, le temps ne peut pas se résumer à une connaissance ayant le même statut que toutes les autres, ne serait ce que parcequ'il intervient dans la modification de celles-ci. De plus, les propriétés particulière du temps (son irréversibilité, son partage entre tous les acteurs de l'univers) imposent une modélisation propre de la notion de temps dans un système intelligent.

Dans le cadre de l'intelligence artificielle et plus particulièrement du raisonnement temporel, le temps est vu suivant deux acceptions. Le temps dit *de bas nivau* est celui qui se mesure, qui s'ordonne et qui se place sur un axe temporel éventuellement numérique. Ce temps est simplement utilisé pour indexer les éléments de la base de connaissances. On sera donc plus particulièrement intéressé par la gestion d'une base de données et de connaissances temporelles, pour lesquelles la mise en œuvre d'une représentation adéquate du temps se résume à la gestion de l'ordre de celui-ci. Les approches correspondantes font appel à des techniques issues de la recherche opérationnelle et de la programmation par contraintes. Dans cette approche, une date est un point qui doit se placer sur un axe numérique représentant le temps. Les informations manipulées ont une représentation uniforme, par exemple associer une information à une date et représenter cette association par un nœud unique dans un graphe. Les méthodes mises en œuvre dans ce cadre sont celles de la programmation linéaire pour résoudre des systèmes d'équations et d'inéquations, des recherches de plus court chemin, des méthodes de gestion de ressources et de projet (PERT) ou d'ordonnancement.

La deuxième vision du temps est celle d'un facteur dont on cherchera à modéliser les propriétés les plus marquantes comme son irréversibilité, son égale accessibilité à tous les acteurs, ainsi que son intervention dans les liens de cause à effet et son influence sur l'évolution de l'univers modélisé. Ce temps, dit *de haut niveau*, est la visée de nombreux formalismes logiques, construits soit à partir des logiques modales, soit à partir de la logique classique du premier ordre. Ce sont certains de ces formalismes qui vont être décrits maintenant.

Nous commençons par décrire une des premières méthodes utilisées pour formali-

ser le temps en intelligence artificielle. Celle-ci étend simplement la logique classique du premier ordre en accordant un statut spécial au temps et aux termes qui le représentent. Une autre approche est celle des logiques temporelles modales décrite dans le paragraphe 1.1.2. Les logiques temporelles réifiées sont les formalismes qui ont reçus le plus d'attention pour le raisonnement temporel et qui sont à l'origine de la plupart des travaux dans ce domaine : elles sont décrites dans le paragraphe 1.1.3. Pour finir cet exposé sur les formalismes de représentation du temps, nous décrivons le calcul d'événements de Sergot et Kowalski, qui, en raison de ses choix ontologiques et de son cadre formel, possède un statut légèrement à part des formalismes précédents.

1.1.1 La logique classique et la méthode des « arguments temporels »

L'approche la plus simple consiste à rester dans la cadre de la logique classique du permier ordre, et à rajouter à chaque prédicat utilisé un argument supplémentaire représentant l'information temporelle sous la forme d'un instant ou d'un intervalle. Cette méthode, appelée « méthode des arguments temporels » est l'une des premières qui a été étudiée pour la représentation du temps. A première vue, cette méthode semble de peu d'intérêt : en effet, on ne peut pas exprimer simplement de propriétés particulière au temps (antériorité des effets sur les causes, par exemple) sans expliciter celles-ci pour tous les symboles de relation présents dans le langage considéré.

Néanmoins, cette méthode a été étudiée plus complètement par (Haugh, 1987a), qui donne une sémantique précise pour cette forme de logique, et qui en conclut que cette approche, malgré ces défauts, reste viable pour le raisonnement temporel. L'approche de Haugh est un cas particulier de la logique du premier ordre, où les atomes sont de la forme $P(t, x_1, \ldots, x_n)$ avec t qui désigne un terme temporel, et les x_i sont des termes non-temporels. Les différentes entités temporelles sont représentés soit par des instants, soit par des termes fonctionnels qui correspondant aux différentes formes d'intervalles : $closed(t_1, t_2)$, $open(t_1, t_2)$, $open\_l(t_1, t_2)$, ou $open\_r(t_1, t_2)$. Certains prédicats prennent en argument plus d'un terme temporel : $<$ et $=$. Un prédicat particulier $Exists(x)$ décrit l'existence de l'individu dénoté par x.

La sémantique de ce langage est donnée par des structures d'interprétation de la forme (W, O, D, E), où W est un ensemble de points de temps, O une relation d'ordre sur W, D est le domaine d'interprétation des termes non temporels, et E est une relation entre des sous-ensembles de W et des éléments de D. la relation E matérialise les périodes d'existence des éléments de D. La fonction d'interprétation V réalise alors classiquement le lien entre termes temporels et éléments de W, entre termes non-temporels et éléments de D, entre le symbole $<$ et la relation O, entre le prédicat $Exists()$ et la relation E, et enfin associe à un prédicat P un ensemble de n-uplet $\langle t, x_1, \ldots, x_n \rangle$ pris dans $W \times D \times \ldots \times D$.

Pour Haugh, un tel langage présente une expressivité suffisante pour la plupart

des usages du raisonnement temporel. Une autre approche analogue avec une expressivité plus grande est présentée dans (Bacchus, Tenenberg, & Koomen, 1991).

1.1.2 Les logiques temporelles modales

Plus intéressante semble l'utilisation de logiques modales (Mackinson, 1990; Nef, 1990). Dans celles-ci, les formules dont on souhaite voir varier la vérité en fonction du temps sont préfixées par différents opérateurs modaux suivant l'expression recherchée. On introduit en particulier les expressions suivantes :

- $\mathbf{F}\phi$ signifie que ϕ sera vraie dans le futur ;
- $\mathbf{P}\phi$: ϕ a été vraie dans le passé ;
- $\mathbf{G}\phi$: ϕ sera toujours vraie dans le futur ;
- $\mathbf{H}\phi$: ϕ a toujours été vraie dans le passé.

On a les dualités suivantes entre les opérateurs modaux :

$$\begin{aligned} \mathbf{G}\phi &\equiv \neg\mathbf{F}\neg\phi \\ \mathbf{H}\phi &\equiv \neg\mathbf{P}\neg\phi \end{aligned}$$

Les opérateurs classiques de nécessité $\Box$ et de possibilité $\Diamond$ peuvent être liés de deux manières différentes avec les opérateurs temporels :

- suivant la conception Diodoréenne, on a

$$\begin{aligned} \Box\phi &\equiv \phi \wedge \mathbf{G}\phi \\ \Diamond\phi &\equiv \phi \vee \mathbf{F}\phi \end{aligned}$$

  c'est à dire que ce qui est nécessaire est vrai maintenant et le restera dans l'avenir ;

- suivant une conception Aritotélicienne suivant laquelle ce qui est nécessaire a été, est, et sera toujours vrai :

$$\begin{aligned} \Box\phi &\equiv \mathbf{H}\phi \wedge \phi \wedge \mathbf{G}\phi \\ \Diamond\phi &\equiv \mathbf{P}\phi \vee \phi \vee \mathbf{F}\phi \end{aligned}$$

L'interprétation des formules des logiques modales se fait par rapport à des structure, dites de Kripke, qui définissent un ensemble d'univers ou de mondes, une relation entre ces mondes, dite relation d'accessibilité, et ce qu'il faut pour interpréter dans chaque monde une formule sans opérateur modal, en général ce sont des interprétations classiques (Tarskiennes). Dans le cas de la logique temporelle modale, les mondes représentent différents états de l'univers à différents moments (instants ou intervalles), et la relation d'accessibilité correspond à la relation de précédence

entre les moments. Suivant les propriétés désirées pour la relation d'accessibilité, la logique peut être définie axiomatiquement avec les axiomes suivants:

$$\begin{array}{l}\phi \text{ où } \phi \text{ est une tautologie}\\ \mathbf{G}(\phi_1 \supset \phi_2) \supset (\mathbf{G}\phi_1 \supset \mathbf{G}\phi_2)\\ \mathbf{H}(\phi_1 \supset \phi_2) \supset (\mathbf{H}\phi_1 \supset \mathbf{H}\phi_2)\\ \phi \supset \mathbf{HF}\phi\\ \phi \supset \mathbf{GP}\phi\end{array}$$

auxquels on rajoute par exemple les axiomes

$$\begin{array}{l}\mathbf{FF}\phi \supset \mathbf{F}\phi\\ \mathbf{PP}\phi \supset \mathbf{P}\phi\end{array}$$

pour exprimer la transitivité de la relation d'accessibilité entre mondes. De la même manière, l'on peut ajouter des axiomes qui correspondent à la linéarité, à droite ou a gauche, de la relation d'accessibilité, à la densité de l'ensemble des moments, etc.

La logique modale a été utilisée, par exemple, par Manna et Pnuelli pour décrire et raisonner sur l'exécution de programmes parallèles (Manna & Pnuelli, 1981), au besoin en ajoutant d'autres opérateurs modaux comme $\mathcal{U}$ pour *until*, etc. (Fusaoka, Seki, & Takahashi, 1983) utilise une logique temporelles modale pour décrire et contrôler des systèmes de production. Il existe peu de méthodes efficaces (dont le coût en espace ou en temps ne soit pas exponentiel) de raisonnement avec des logiques modales. Une extension de la méthode des tableaux est proposée par P. Wolper (Wolper, 1985) ou des calculs de résolution par (Fisher, 1991; Enjalbert & Cerro, 1989; Ohlbach, 1988) et (Venkatesh, 1986). (Barringer, Fisher, Gabbay, & Hunter, 1991) décrivent un système de raisonnement et de programmation logique appelé METATEM et basé sur une logique temporelle modale. Une logique modale particulière est celle de (Halpern & Shoham, 1986) où les mondes ne sont pas assimilés à des états à un instant donné mais pendant des intervalles de temps.

1.1.3 Les logiques temporelles réifiées

Les logiques temporelles réifiées sont encore un autre exemple d'outils logiques développés pour représenter et raisonner sur le temps. Contrairement aux approches logiques conçues à partir de la logique classique du premier ordre et de la logique modale, les logiques temporelles réifiées mettent en avant l'aspect temporel comme une entité primitive du langage. Suivant les travaux, la représentation primitive du temps se fait, soit par l'instant comme dans la logique de McDermott (McDermott, 1982), soit par l'intervalle comme dans celle d'Allen (Allen, 1984). L'ambition de ces formalismes est variable, depuis la simple description de la répartition temporelle de la vérité de propositions en fonction du temps, jusqu'à la formalisation des liens causaux, de la persistance, etc.

La partie strictement temporelle de ces logiques est en général simple: son but est souvent de pouvoir décrire la répartition temporelle de la vérité des propositions.

Ce langage primitif sert ensuite à une formalisation de l'action, du changement et des liens de cause à effet. Pour l'analyse de ces logiques, on peut donc dans une première approche, ne s'intéresser qu'à la partie strictement temporelle de celles-ci.

La réification est le processus par lequel un objet, une formule, d'un langage devient dans un autre langage un simple terme propositionel de celui-ci. Ici, le premier langage est a-temporel, et une logique temporelle réifiée est alors une logique typée où les formules atemporelles ne sont plus que des termes en argument d'un prédicat unique qui associe ceux-ci avec des entités temporelles. On écrira par exemple :

$$\text{TRUE}(\textit{forme\_atemporelle}, \textit{instant})$$

pour réaliser cette association. Suivant les logiques, la signification réelle d'une telle expression peut être, soit l'occurrence d'un événement à l'instant considéré, soit la vérité d'une propriété à cette instant ou pendant une période comprenant l'instant, etc.

Nous décrivons maintenant plusieurs formalismes de logiques temporelles réifiées, en commençant par les deux cités précédemment, celle de McDermott et celle de Allen. Sur la base de critiques adressées à ces formalismes, Y. Shoham, puis A. Galton, ont développé d'autres systèmes. Shoham (1987) a d'abord récusé l'ontologie pléthorique des logiques de Allen et de McDermott et a proposé une classification des propositions pour y remédier. Galton (1990) a tenté de réintroduire dans la logique d'Allen l'instant à égalité avec l'intervalle, et ceci dans le but de raisonner sur des changements continus en fonction du temps. Ces deux approches seront décrites par la suite.

La logique de McDermott

La logique de McDermott (1982) a pour ambition affichée de modéliser deux aspects : l'indétermination du futur et la continuité du temps. La notion primitive de cette logique est l'*état* qui est une photographie instantanée de l'univers à une date précise. Ces états sont ordonnés partiellement par $<$ et l'ensemble de ces états est dense, c'est à dire qu'entre deux états s_1 et s_2 tels que $s_1 < s_2$ il existe toujours un autre état s_3 telque $s_1 < s_3 < s_2$. Chaque état est associé à une date qui est un nombre réel, l'ordre sur les réels et l'ordre sur les états étant cohérents entre eux.

Pour représenter les « histoires possibles » de l'univers, McDermott définit la notion de *chronique* comme un ensemble complètement ordonné d'états qui s'étend infiniment dans le temps. L'ensemble des chroniques présente donc une structure arborescente et ramifiée dans le futur, comme le montre la figure 1.1.

Un *fait*, qui intuitivement correspond à l'assertion de la vérité d'une proposition à un instant ou durant une période, est décrit dans cette logique comme un ensemble d'états : cet ensemble est celui où la proposition est vraie. Ceci permet à McDermott de distinguer deux faits particuliers : *toujours* qui est l'ensemble de tous les états, et *jamais* qui est l'ensemble vide. A partir de ces définitions, il est aisé de définir une

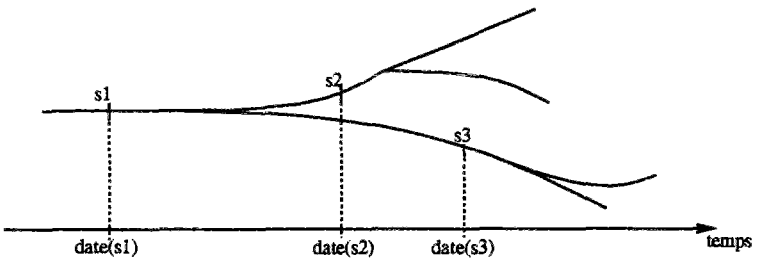


FIG. 1.1 - *Ramification des chroniques dans la logique de McDermott.* s_1, s_2 *et* s_3 *sont des états.*

conjonction de faits comme l'intersection des ensembles d'états, et une disjonction de faits comme une union. Néanmoins, si cette définition paraît solide formellement, il devient impossible de distinguer dans cette logique deux faits distincts qui ont les mêmes occurrences, et seront donc représentés par les mêmes ensembles d'états.

Un *événement*, dont l'objet est de représenter des actions sur l'univers, est défini comme un ensemble d'intervalles d'états. Ceci permet d'assimiler l'événement avec ses intervalles d'occurrence, sachant qu'un intervalle est toujours un sous-ensemble d'une chronique. Un événement a donc comme caractéristique de « prendre du temps », ce qui le distingue de la notion équivalente dans le calcul de situations par exemple, où un événement est un changement instantané entre deux situations. De la même manière, différencier deux événements distincts ayant les mêmes occurrences est impossible dans ce formalisme.

La formalisation des liens causaux est construite à l'aide des notions précédentes : un tel lien se fait d'abord entre un événement et un autre sous condition qu'un fait connexe soit vrai à l'occurrence du premier événement et reste vrai jusqu'à l'occurrence de l'événement causé. Une deuxième forme de lien causal se fait entre un événement et un fait : plus exactement, entre un événement et la persistance d'un fait. Dans ce cas, la persistance est définie comme la propriété d'un fait de durer pendant une période prédéterminée, ou d'être explicitement interrompu avant l'expiration du délai.

McDermott s'intéresse ensuite aux quantités qui varient continuement avec le temps, et auxquelles on associe une valeur numérique. Ces quantités sont appelées « *fluent* » et permettent de définir comme un événement tout changement de leur valeur d'une constante à une autre. Ces changement se font continuement, ce qui est exprimé par un axiome de « la valeur intermédiaire » comme en Analyse. La fin de l'article de McDermott est consacrée à formaliser la notion d'action pour l'appliquer à la planification.

La logique de McDermott comprend au final une quarantaine d'axiomes dont ni la consistance, ni la complétude n'ont été démontrées (ce qui est probablement très difficile à faire). De plus, l'ontologie est complexe, et même si elle est réduite (état, date, fait et évènement) sa définition n'est pas très naturelle.

La logique de Allen

Allen (1984) a proposé une formalisation du temps et de l'action sous la forme d'une logique temporelle basée sur des intervalles. Un intervalle est une période de temps qui peut être éventuellement décomposée en sous-périodes, et cette remarque motive le choix d'Allen pour sa logique. Cette logique temporelle est construite à partir des éléments suivants :

- un prédicat HOLDS qui permet d'associer une *propriété* et un intervalle, avec pour signification que la propriété est vraie pendant toute la période de temps représentée par l'intervalle ;
- un ensemble de six relations entre intervalles décrivant leur positions relatives. Ces six relations et leurs inverses ainsi que la relation d'égalité forment treize relations mutuellement exclusives, communément appelée l'*algèbre d'intervalles*, et qui possède une procédure de décision semi-complète qui est décrite dans le paragraphe 1.2.1 (page 27) sur les contraintes temporelles ;

Ces éléments permettent de disposer d'une logique apte à décrire la répartition temporelle de la vérité des propriétés. Pour Allen, une propriété est décrite par un terme quelconque de la logique du premier ordre. La caractéristique la plus importante des propriétés est l'homogénéité de leur vérité par rapport aux sous-intervalles de ceux sur lesquels elles sont vraies. En d'autres termes, si la propriété P est vraie sur un intervalle i, ce qui se note HOLDS(P,i), alors elle est vraie sur tout intervalle i' inclus dans i. Les propriétés peuvent être combinées et niées à l'aide des fonctions and, or, not, all et exists. Ces fonctions créent de nouvelles propriétés, et leur définition est telle que celles-ci vérifient toujours la propriété d'homogénéité. Il semble donc que le langage des propriétés est celui de la logique du premier ordre, mais Allen ne le dit pas explicitement.

Contrairement à la logique de McDermott, Allen ne formalise pas l'aspect branchant et indéterminé du futur. La logique ne suppose qu'une ligne de temps, et le raisonnement sur le futur n'est qu'un cas particulier de raisonnement hypothétique. De ce fait, pour Allen, il est de la responsabilité de l'implantation et des mécanismes d'inférences de prendre en compte cette indétermination du futur.

Les seules propriétés de la logique d'Allen sont insuffisantes pour décrire tous les aspects temporels du monde. En particulier, un *événement* ne vérifie pas forcément la propriété d'homogénéité. Allen ajoute un nouveau concept à son ontologie : celle d'*occurrence*, qui se subdivise ensuite en *processus* et en *événement*. La différence entre ces deux notions tient à l'existence ou non d'une finalité pour l'occurrence concernée. Un autre critère donné par Allen indique que l'on peut compter le nombre d'occurrences d'un événement, alors qu'il est impossible de le faire pour un processus.

Le prédicat OCCUR s'applique à un événement et un intervalle, et la vérité d'une assertion OCCUR(E,i) se vérifie pour le plus petit intervalle i d'occurrence de l'événement e. Les occurrences de processus sont décrites par le prédicat OCCURRING,

et, contrairement à un événement, ces occurrences ne sont pas forcément décrites par le plus petit intervalle possible : pour un intervalle d'occurrence i d'un processus, il existe un nombre « substantiel » de sous-intervalles i' de i sur lesquels le processus a une occurrence. La caractérisation exacte est que si un processus P se déroule pendant une période i, alors il existe au moins une période i' inclue dans i telle que OCCURRING(P, i') est vérifié.

Sur la base de ce langage, Allen formalise ensuite différents aspects de la causalité. Pour un lien causal entre deux événements, il faut toujours vérifier que l'événement causé est soit simultané, soit postérieur à l'événement déterminant. La relation entre événements qui correspond est transitive, anti-symmétrique et antiréflexive. Sur la base d'une action effectuée par un agent, la distinction se fait entre les *exécutions* (« *performances* ») et les *activités*, suivant que l'action cause un événement ou un processus.

Critiques des logiques de Allen et de McDermott

La première critique de Shoham (Shoham, 1987) concerne l'ontologie des logiques temporelles de McDermott et Allen : celle-ci est pléthorique, trop complexe pour faciliter l'usage des logiques puisque chaque notion est introduite et baptisée par référence à la signification du mot dans le langage commun, ce qui ne peut manquer de varier d'une personne à l'autre et d'une langue à l'autre. Un exemple typique de ceci est la distinction processus-événement dans la logique d'Allen.

La deuxième critique de Shoham concerne le fait que le langage des propriétés est analogue à celui de la logique du premier ordre puisqu'il comprend, entre autres, des fonctions **all** et **exists** qui permettent la quantification à l'intérieur des propriétés. L'absence de sémantique clairement définie pour ces expressions est un obstacle à leur utilisation. De plus, l'absence de l'instant (entité temporelle ponctuelle) dans la logique oblige Allen à des formulations acrobatiques pour exprimer les compositions de propriétés à l'aide des fonctions **and**, **or** et **not**. L'exemple cité par Shoham est celui de la fonction **or** définie par l'axiome suivant :

$$HOLDS(\mathsf{or}(p,q),i)$$
$$\equiv \forall i'\ \ IN(i',i) \Rightarrow (\exists i''\ \ IN(i'',i') \wedge (HOLDS(p,i'') \vee HOLDS(q,i'')))$$

qui aurait eu une formulation plus simple si le point avait été disponible dans la logique : la propriété $\mathsf{or}(p,q)$ est alors vraie sur tout intervalle i tel que à tout instant compris dans i, soit p soit q est vrai à cet instant.

Les critiques de A. Galton (Galton, 1990) rejoignent celles de Shoham sur la nécessité d'introduire l'instant dans la logique d'Allen. Le but n'est pas le même puisque Galton veut pouvoir modéliser des systèmes dont l'état varie continuement avec le temps.

Le premier problème soulevé par Galton est celui d'un corps X en mouvement pendant l'intervalle T et qui passe par la position P. Si p est la propriété qui représente la proposition « X est en P », la négation de p signifie « X n'est pas en

P ». Comme le mouvement de X est continu, il n'y a pas de sous-intervalle pendant lequel la proposition p est vraie : si c'était le cas, X serait au repos pendant cette période. En utilisant les axiomes de Allen, Galton arrive à la conclusion que $HOLDS(not(p), T)$ est vérifié : plus généralement, quelque soit la position P concernée, l'objet X n'y est jamais ! Pour Galton, si une telle logique temporelle décrivait l'univers pour un robot possédant un bras manipulateur, il serait impossible au robot d'attraper l'objet X.

La cause de tout ceci, telle qu'elle est identifiée par Galton, est l'absence de l'instant dans la logique temporelle. L'instant s'entend comme une entité temporelle sans durée propre qui permet par exemple de faire la différence entre le fait que le corps X soit au repos à la position P et le fait qu'il soit à cette position de manière transitoire au cours de son mouvement.

Définition formelle d'une logique temporelle réifiée

Toutes ces remarques amènent Shoham à définir une nouvelle logique temporelle réifiée. L'intérêt de cette logique est double :

- elle est définie formellement avec une syntaxe et une sémantique claire. Sa définition se rapproche alors de celle de la logique du premier ordre, avec des structures d'interprétation dont la forme est sans ambiguïté. Shoham propose aussi bien une version propositionnelle qu'une version en premier ordre de sa logique ;

- elle propose une ontologie très simple, qui ne distingue que les instants et les propositions dont on exprime la vérité en fonction du temps. Les diverses catégories définies par McDermott et Allen se retrouvent dans une classification des propositions établie suivant un critère unique qui lie la vérité d'une proposition sur un intervalle i avec sa vérité sur les sous-intervalles de i ou aux points qui composent i.

La définition syntaxique du cas propositionnel est très simple : P est un ensemble de propositions, T un ensemble de symboles d'instants, et V un ensemble de variables temporelles. Les formules bien formées sont définies récursivement par les conditions suivantes :

- pour deux termes temporels t_1 et t_2 qui appartiennent soit à T soit à V, les expressions $t_1 = t_2$ et $t_1 \preceq t_2$ sont bien formées ;

- pour deux termes temporels t_1 et t_2 et une proposition p de l'ensemble P, alors $TRUE(t_1, t_2, p)$ est bien formée ;

- les compositions habituelles de formules bien formées à l'aide des connecteurs $\vee$, $\wedge$ et $\neg$ sont bien formées ;

- la formule $\forall v\ \phi$, où v appartient à V et ϕ est une formule bien formée, est aussi une formule bien formée.

L'interprétation de ces formules se fait par rapport à des structures représentées par un triplet $\langle W, \leq, M \rangle$, où W est un ensemble d'instants (par exemple des nombres entiers), $\leq$ est une relation d'ordre sur W, et M est la fonction d'interprétation qui associe à chaque élément de T (symbole d'instant) un élément de W, et à chaque symbole de proposition un élément de $2^{W \times W}$ (c'est à dire un ensemble de paires d'instants représentant chacune les deux bornes d'un intervalle). A l'aide d'une telle structure S et d'une affectation de variables α à valeurs dans W, on interprète ainsi les formules atomiques :

- $S, \alpha \models (t_1 = t_2)$ si et seulement si $\mathsf{Val}(t_1) = \mathsf{Val}(t_2)$;
- $S, \alpha \models (t_1 \preceq t_2)$ si et seulement si $\mathsf{Val}(t_1) \leq \mathsf{Val}(t_2)$;
- $S, \alpha \models \mathit{TRUE}(t_1, t_2, p)$ si et seulement si $\langle \mathsf{Val}(t_1), \mathsf{Val}(t_2) \rangle \in M(p)$;

où la fonction Val est M si son argument est un symbole d'instant, et α si son argument est une variable. Les formules composées et avec quantificateur s'interprètent de la manière habituelle.

La version en premier ordre de cette logique est définie en augmentant la syntaxe des expressions temporelles en autorisant des expressions utilisant des fonctions (arithmétiques par exemple) et des constantes numériques. De même les termes non-temporels qui étaient limités aux propositions peuvent maintenant être construits à l'aide de symboles de relation, de symboles de fonction, de variables et de constantes. Ce langage permet alors de construire des formules de la forme :

$$\exists u\ \ \mathit{TRUE}(t_3, t_4, \mathit{ON}(u, B))$$

ou encore

$$\begin{aligned} &\forall v\ \ (1776 \preceq v \wedge v \preceq 1986) \\ &\quad \Rightarrow \mathit{TRUE}(v, v, \mathit{GENDER}(\mathit{PRESIDENT}(\mathit{USA}), \mathit{MALE})) \end{aligned}$$

Dans le cas des termes non-temporels la distinction entre symboles de fonctions et constantes se comprend si l'on considère que dans la formule précédente, USA est une constante dont l'interprétation ne change pas en fonction du temps (« rigide »), alors que le symbole de fonction *PRESIDENT* a une interprétation qui varie en fonction du temps. Plus exactement, l'interprétation du symbole *PRESIDENT* dépendra de l'interprétation du terme temporel v : pour une valeur d'interprétation de v comprise entre 1789 et 1797, l'interprétation du symbole *PRESIDENT* retourne une fonction sur le domaine qui associe « Washington » à « USA », et pour une interprétation de v entre 1977 et 1981, la fonction retournée par l'interprétation de *PRESIDENT* associe « Carter » à « USA », et « Giscard » à « France ».

De la même manière, l'interprétation des symboles relationels est aussi dépendante du temps. Dans la formule $\exists u \ \ TRUE(t_3, t_4, ON(u, B))$ le symbole relationnel ON est interprété différemment selon les valeurs de t_3 et t_4.

A partir de cette logique, Shoham réinterprète toutes les notions des logiques de Allen et de McDermott en fonction d'un critère unique qui lie la vérité entre intervalles et sous-intervalles et entre intervalles et instants. La classification de Shoham est exhaustive, et on y trouve par exemple les catégories suivantes :

liquides : ces propositions vérifient que dès lors qu'elles sont vraies sur un intervalle, elles le sont à tous les instants composant cet intervalle ; et réciproquement, dès lors qu'elles sont vraies à tous les instants d'un intervalle, elles sont vraies sur cet intervalle. Cette catégorie est analogue aux faits de McDermott et aux propriétés d'Allen ;

solides : ces propositions ne sont jamais vraies sur deux intervalles distincts qui se recouvrent. C'est le cas de certains événements dans les logiques de Allen et de McDermott.

L'extension de la logique d'Allen par A. Galton

La proposition de Galton est d'introduire dans la logique une nouvelle entité temporelle pour représenter l'instant. Les liens entre les instants et les intervalles s'expriment avec les relations $WITHIN(t, I)$ [1] pour exprimer que l'instant t est dans l'intervalle I, et $LIMITS(t, I)$ qui signifie que l'instant t est une des bornes de I. Pour les propriétés au sens de Allen, trois prédicats sont définis qui permettent de décrire différentes formes de vérité en fonction du temps :

- $HOLDS\text{-}AT(p, t)$ signife que la propriété p est vraie à l'instant t ;
- $HOLDS\text{-}IN(p, I)$ signifie qu'il existe dans l'intervalle I un instant t où l'on vérifie $HOLDS\text{-}AT(p, t)$;
- $HOLDS\text{-}ON(p, I)$ signifie que pour tout instant t appartenant à l'intervalle I, on vérifie $HOLDS\text{-}AT(p, t)$.

En particulier, on retrouve la propriété d'homogénéité des propriétés, que ce soit d'un intervalle vers ses sous-intervalles ou d'un intervalle vers les points qui le composent.

La négation d'une propriété est définie ainsi :

$$HOLDS\text{-}AT(\mathsf{not}(p), t) \equiv \neg HOLDS\text{-}AT(p, t)$$

Cette définition permet alors de montrer les équivalences suivantes :

$$HOLDS\text{-}IN(\mathsf{not}(p), I) \equiv \neg HOLDS\text{-}ON(p, I)$$
$$HOLDS\text{-}ON(\mathsf{not}(p), I) \equiv \neg HOLDS\text{-}IN(p, I)$$

1. que nous noterons $t \in I$ pour alléger la notation.

Galton introduit de plus une distinction entre les propriétés : les *états de position* (« state of position ») et les *états de mouvement* (« state of motion »). Les premiers sont des propriétés qui peuvent être vrais à des instants isolés et qui vérifient l'axiome suivant :

$$\forall I \, \forall t \quad [(t \in I) \Rightarrow \mathit{HOLDS\text{-}IN}(p, I)] \Rightarrow \mathit{HOLDS\text{-}AT}(p, t)$$

c'est à dire qu'une telle propriété est vraie à l'instant t si pour tout intervalle I comprenant t on vérifie $\mathit{HOLDS\text{-}IN}(p, I)$. De même, les états de mouvement ne sont vrais à un instant que si cet instant est compris dans un intervalle pendant lequel l'état est vrai. Les états de mouvement satisfont l'axiome :

$$\mathit{HOLDS\text{-}AT}(p, t) \quad \Rightarrow \quad \exists I \quad [(t \in I) \wedge \mathit{HOLDS\text{-}ON}(p, I)]$$

Divers théorèmes permettent d'affiner la notion d'états de position. En particulier, ceux-ci sont vrais aux bornes des intervalles où ils sont vrais. Il est aussi possible de redéfinir *HOLDS-AT* et *HOLDS-ON* en fonction de *HOLDS-IN* pour ces propriétés :

$$\begin{aligned} \mathit{HOLDS\text{-}AT}(p, t) &\equiv \forall I \quad [(t \in I) \Rightarrow \mathit{HOLDS\text{-}IN}(p, I)] \\ \mathit{HOLDS\text{-}ON}(p, I) &\equiv \forall I' \quad [(I' \subset I) \Rightarrow \mathit{HOLDS\text{-}IN}(p, I')] \end{aligned}$$

De la même manière, *HOLDS-ON* peut être pris comme l'expression primitive pour les états de mouvement et l'on peut définir les deux autres prédicats en fonction de celui-ci. De plus, il apparaît que la négation d'un état de mouvement est un état de position, et réciproquement.

Les événements sont traités suivant la même approche. Galton définit trois prédicats *OCCUR-AT*, *OCCUR-ON* et *OCCUR-IN* pour désigner respectivement l'occurrence d'un événement ponctuel à un instant, l'occurrence d'un événement durable sur un intervalle, et l'occurrence d'un événement (ponctuel ou durable) quelque part pendant un intervalle.

1.1.4 Critiques des logiques temporelles réifiées

Les logiques temporelles réifiées ont reçus plusieurs critiques. Certaines venaient des personnes ayant étudié des logiques temporelles bâties sur la méthode des arguments temporels (paragraphe 1.1.1, page 13), en particulier Haugh (1987a) et (Bacchus et al., 1991). En premier lieu, l'inexistence d'une sémantique clairement définie pour la quantification à l'intérieur des propositions (termes non temporels) ne permet pas de formaliser des domaines ou l'ensemble des individus de l'univers est variable en fonction du temps. De ce point de vue, le travail de Shoham (1987) évite cet écueil en restreignant la forme des propositions pour n'admettre qu'une forme atomique de celles-ci. L'inconvénient souvent cité de la sémantique définie par Shoham est qu'elle ne repose pas sur celle de la logique classique du premier ordre, et que les notions habituelles de consistance et de complétude doivent être

étudiées dans ce cadre spécifique. De la même manière, l'introduction de connecteurs logiques dans les termes non temporels reste un problème sans réponse claire. Allen (1984) admet cette éventualité mais ne fournit aucune sémantique pour de tels termes complexes.

L'argument de (Haugh, 1987a) est que, compte tenu de ces limites des logiques réifiées, la méthode des arguments temporels ne souffre pas de la comparaison avec celles-ci. Encore faut il reconnaitre que Haugh donne une sémantique pour les logiques construites par la méthode des arguments temporels qui introduit un composant temporel spécifique, tout comme l'approche de Shoham.

Galton récuse la nécessité de la réification sur la base d'arguments d'ordre philosophique. Il propose (Galton, 1991b) une transformation des théories réifiées par l'introduction d'éléments particuliers, les *tokens*, qui sont associés à une expression atemporelle. Ces *tokens* sont ensuite associés, par un prédicat comme *HOLDS*, à une date ou un intervalle particulier: la formule $TRUE(p(u,v),t)$ est transformée en $\exists e \;\; p(u,v,e) \wedge HOLDS(e,t)$, où e désigne un *token*. Plus exactement, e est un *event token* dans la mesure ou il désigne une occurrence particulière de l'événement désigné par $p(u,v)$, ce dernier étant plutôt un *event type*.

1.1.5 Le calcul d'événements

Le calcul d'événements (*event calculus*), présenté dans (Kowalski & Sergot, 1986), est un formalisme un peu particulier par rapport aux logiques décrites précédemment. Il a été développé pour s'affranchir de certains problèmes liés au calcul de situation de McCarthy. Initialement utilisé pour formaliser les ajouts dans les bases de données et pour la compréhension du langage naturel, il a ensuite été utilisé pour de nombreuses applications: la planification (Eshghi, 1988), le raisonnement spatio-temporel (Borillo & Gaume, 1990), etc.

Les entités primitives du calcul d'événements sont l'*événement*, la *période* et la *relation*. Une période est définie par la combinaison d'un intervalle de temps et d'une relation (propriété de l'univers) vraie pendant cet intervalle.

Contrairement aux logiques temporelles réifiées, l'ontologie du calcul d'événements donne un rôle primordial à l'événement. Ceux-ci sont représentés par des termes, et interviennent dans la formation des périodes. Au début et à la fin de chaque période interviennent un événement qui initie la période, ce qui s'exprime par la forme $initiates(e,p)$, et un autre événement qui termine la période, ce qui est noté $terminates(e,p)$. Les liens causaux sont ainsi représentés implicitement à l'aide de ces prédicats, plutôt que par des prédicats specifiques comme dans les formalismes de Allen et de McDermott. Plus précisément, les événements sont représentés par des *event token*, suivant le sens donné à cette expression par (Galton, 1991b). Les seuls éléments réifiés du calcul d'événements sont les relations.

Les événements peuvent n'être que partiellement ordonnés entre eux. De même, il n'est pas nécessaire qu'ils soient associés à une date.

$$\text{holds\_at}(P,T) \leftarrow \text{happens}(E),\ \text{initiates}(E,P),\ \text{succeeds}(E),\ E \prec T,\ \textbf{not}\ \text{clipped}(E,P,T).$$

$$\text{clipped}(E,P,T) \leftarrow \text{happens}(C),\ \text{terminates}(C,P),\ \text{succeeds}(C),\ E \preceq C,\ C \prec T.$$

FIG. 1.2 - *Les clauses du calcul d'événements tel qu'il est décrit par M. Shanahan. P désigne une propriété, T est un instant et E et C désignent des événements. Les symboles $\prec$ et $\preceq$ représentent la relation d'antériorité entre événements.*

Le formalisme du calcul d'événements est celui de la programmation logique : les déductions peuvent se faire à l'aide d'un interprète PROLOG. La négation par échec est utilisée comme mode de raisonnement pour obtenir la persistance par défaut des faits dont on connait le début d'une période de vérité. La formulation initiale du calcul d'événements par Kowalski et Sergot a été simplifiée par Shanahan et est reprise sur la figure 1.2.

Cette formulation du calcul d'événements est la plus simple connue : elle met bien en évidence la persistance par défaut obtenue grâce à la négation par échec. Elle a été utilisée comme base de la plupart des travaux consécutifs, en particulier pour formaliser les changements continus (Shanahan, 1989b). Une autre extension proposée par (Evans, 1990) permet de raisonner à différents niveaux de granularité temporelle. Le calcul d'événements ainsi défini est celui qui a été utilisé en planification abductive par K. Eshghi (Eshghi, 1988) et L. Missiaen (Missiaen, 1991).

1.2 Outils et techniques du raisonnement temporel

Jusqu'à maintenant, nous avons essentiellement décrit des formalismes sans préjuger de leur mise en œuvre concrète. Le but de cette partie est de pallier ce manque et de faire une revue des différentes méthodes et systèmes qui ont été proposés pour rendre opérationnel le raisonnement temporel.

Le premier paragraphe (1.2.1) de cette partie décrit les systèmes de gestion de relations temporelles. Ces systèmes sont destinés à prendre en charge la gestion des relations temporelles entre moments (antériorité, égalité, inclusion, etc.), que ces moments soient des instants ou des intervalles. Le but de ces systèmes est de maintenir une représentation cohérente des relations entre ces entités afin de répondre à des requêtes, inférer des relations nouvelles, etc. Le lien avec des informations non-temporelles n'est pas l'objectif de ces systèmes et doit être réalisé par une autre partie du système. La structure d'un tel système gérant des informations temporelles et proposée par M. Ghallab est représentée sur la figure 1.3.

Un exemple particulièrement représentatif de l'usage des systèmes de gestion

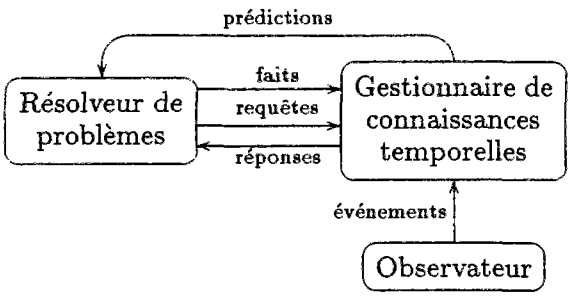


FIG. 1.3 - *Structure générale d'un système gérant des connaissances temporelles*

de relation temporelles est le TMM (*Time Map Manager*) de Dean et McDermott (Dean & McDermott, 1987). Celui-ci est décrit dans le paragraphe 1.2.2.

Pour finir, d'autres approches du raisonnement temporel ont été développées à partir de la théorie des modèles en logique. Cet axe de recherche recoupe souvent celui du raisonnement non-monotone.

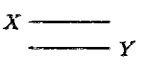
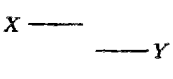
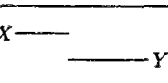
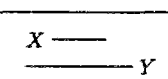
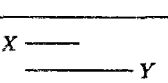
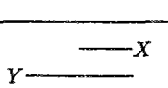
1.2.1 Systèmes de contraintes temporelles

Ce domaine de recherche a été initié par Allen lorsqu'il a décrit un système logique de relations mutuellement exclusives entre des intervalles (Allen, 1983). Ce système de relations, lorsqu'il est muni de lois de composition et d'addition a les propriétés d'une algèbre, ce qui lui donne son nom d'*algèbre d'intervalles*. Allen a proposé un algorithme de propagation qui pemet de décider d'un critère de consistance locale sur un ensemble de ces relations, ainsi que d'approximer le plus petit ensemble de relations liant deux intervalles. La consistance globale d'un tel ensemble de relations étant un problème NP, d'autres algorithmes ont été mis au point pour d'autres critères de consistance plus fort que celui obtenu avec l'algorithme initial d'Allen. Le sous-ensemble maximal de l'algèbre d'intervalle pour lequel la consistance globale n'est pas un problème NP a été identifié par (Nebel & Bürckert, 1993).

Par la suite, l'*algèbre d'instants* a été identifiée par Vilain et Kautz (1986) et des algorithmes de propagation ayant de meilleures propriétés de complexité et de complétude ont été construits. Le temps n'étant pas seulement un facteur supportant des relations symboliques et qualitatives, une partie des travaux sur les contraintes temporelles cherche à intégrer dans un même formalisme des relations de type symboliques (« avant », « après », etc.) et des relations porteuses d'une information numérique, par exemple sur la distance minimale ou maximale entre deux instants.

L'algèbre d'intervalles et les algorithmes de propagation correspondants font l'objet des deux premiers paragraphes de cette partie. Ensuite, l'algèbre d'instants est présentée ainsi que quelques algorithmes de propagation. Pour finir, nous détaillons un certains nombre de représentations mixtes mélangeant des relations symboliques et numériques.

TAB. 1.1 - *Les treize relations de l'algèbre d'intervalles*

Relation	Symbole	Représentation graphique	Relation inverse
X equal Y	=		=
X before Y	b		b^{-1}
X meets Y	m		m^{-1}
X overlaps Y	o		o^{-1}
X during Y	d		d^{-1}
X starts Y	s		s^{-1}
X finishes Y	f		f^{-1}

L'algèbre d'intervalles

L'algèbre d'intervalles a été décrite par Allen (1983). Ce système recense treize relations primitives entre intervalles temporels qui sont mutuellement exclusives et qui forment une base complète, notée B_I, c'est à dire que toute relation entre deux intervalles peut s'exprimer comme une disjonction d'un certain nombre de relations primitives. Ces treize relations sont décrites sur la table 1.1. A l'aide de ces relations, on peut définir par exemple l'inclusion stricte de l'intervalle X dans l'intervalle Y par la disjonction $X\mathrm{d}Y \vee X\mathrm{s}Y \vee X\mathrm{f}Y$, aussi notée sous la forme $X(\mathrm{d,s,f})Y$. Pour toute relation $(b_1, \ldots, b_k)$ la relation inverse est $(b_1^{-1}, \ldots, b_k^{-1})$.

Il existe 2^{13} (8192) relations dans l'algèbre d'intervalles, et sur l'ensemble de ces relations on définit les opérations d'union, d'intersection et de composition :

- l'union, notée $\vee$, entre deux relations est obtenue par l'union (ensembliste) de leur représentation à l'aide des relations primitives ;
- l'intersection, notée $\wedge$ est définie de manière analogue à l'aide de l'intersection ensembliste ;
- la composition de deux relations est définie à l'aide d'une table, donnée dans (Allen, 1983)) qui donne $b_i \circ b_j$ pour tous les couples de relations primitives.

Pour deux relations quelconques $r_1 = (a_1, \ldots, a_n)$ et $r_2 = (b_1, \ldots, b_m)$, la composition est définie par

$$r_1 \circ r_2 = \bigvee_{i,j} a_i \circ b_j$$

Exemple 1.1 *Soient les deux relations* $r = (\mathrm{m}, \mathrm{o}^{-1})$ *entre les intervalles* u *et* v *et* $q = (\mathrm{s}, \mathrm{f})$ *de* v *à* w*, alors la composition de ces relations est calculée ainsi de* u *à* w *:*

$$\begin{aligned} r \circ q &= \mathrm{m} \circ \mathrm{s} \vee \mathrm{m} \circ \mathrm{f} \vee \mathrm{o}^{-1} \circ \mathrm{s} \vee \mathrm{o}^{-1} \circ \mathrm{f} \\ &= (\mathrm{m}) \vee (\mathrm{d}, \mathrm{s}, \mathrm{o}) \vee (\mathrm{o}^{-1}, \mathrm{d}, \mathrm{f}) \vee (\mathrm{o}^{-1}) \\ &= (\mathrm{m}, \mathrm{o}, \mathrm{s}, \mathrm{f}, \mathrm{d}, \mathrm{o}^{-1}) \end{aligned}$$

La relation universelle

$$\Phi = (=, \mathrm{b}, \mathrm{m}, \mathrm{o}, \mathrm{d}, \mathrm{s}, \mathrm{f}, \mathrm{b}^{-1}, \mathrm{m}^{-1}, \mathrm{o}^{-1}, \mathrm{d}^{-1}, \mathrm{s}^{-1}, \mathrm{f}^{-1})$$

est élément neutre de la loi $\wedge$ et absorbant pour $\circ$. Les deux lois sont distributives. (Allen & Hayes, 1985) ont formalisé cette algèbre avec une axiomatique rigoureuse.

Pour un ensemble Θ de relations entre des intervalles, le raisonnement se ramène principalement à deux problèmes :

ISAT : décider de la consistance d'un tel ensemble de relations, c'est à dire si l'on peut extraire de chaque relation de Θ une seule relation primitive telle que les intervalles ainsi contraints soient plaçables sur un axe temporel unidimensionnel. La consistance est identique à la satisfaisabilité de Θ si l'on suppose que les interprétations de ce genre de formules entre intervalles sont une répartition des intervalles concernés sur un axe temporel et qui vérifie pour chaque intervalle que leur borne inférieure est toujours avant leur borne supérieure ;

ISI : trouver pour chaque paire d'intervalles (X, Y) la plus petite relation r de X à Y telle que la relation initiale de X à Y dans Θ implique r. Cette minimalité de la relation est jugée par rapport à la relation d'inclusion.

Vilain et Kautz (Vilain & Kautz, 1986) ont montré que les deux problèmes ISI et ISAT sont NP et équivalents : il existe une transformation polynomiale de l'un à l'autre. Nebel et Bürckert (Nebel & Bürckert, 1993) ont étendu ce résultat d'équivalence à toute sous-classe de l'algèbre d'intervalles. Dans la pratique, les méthodes les plus efficaces pour résoudre ces problèmes utilisent des techniques de propagation de contraintes, en particulier une adaptation de l'algorithme de consistance de chemins (Montanari, 1974; Mohr & Henderson, 1986).

Par transformation des relations entre intervalles en des relations entre les instants (en utilisant $=$, $\leq$ et $\neq$ comme relations primitives) correspondant à leur bornes inférieures et supérieures, (Nebel & Bürckert, 1993) ont identifié le plus grand sous-ensemble, noté $\mathcal{H}$, de l'algèbre d'intervalle pour lequel les deux problèmes ISI et

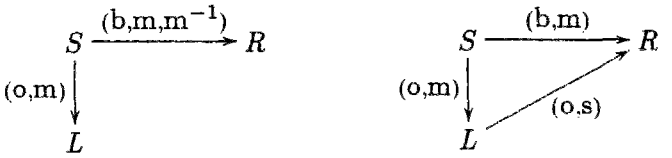


FIG. 1.4 - *Un graphe de contraintes entre intervalles. A gauche le graphe initial, à droite le graphe obtenu après ajout de la relation* (o,s) *de L à R.*

ISAT peuvent être résolus en temps polynomial. Cette analyse est basée sur les propriétés des formules entre instants obtenues par traduction des contraintes entre intervalles en des contraintes entre leurs extrémités. Sachant qu'il est possible de tester la satisfaisabilité d'un ensemble de clauses de Horn propositionelles en un temps polynomial et au mieux linéraire (Dowling & Gallier, 1984), il faut donc que la transformation produise de telles clauses. Par exemple, la relation $(\mathrm{o}, \mathrm{s}, \mathrm{f}^{-1})$ appartient à $\mathcal{H}$. Dans cette classe, le problème ISAT peut être décidé en un temps $O(n^3)$ où n est le nombre d'intervalles présents dans l'ensemble initial de relations. La classe $\mathcal{H}$ contient 868 relations et l'appartenance d'une relation à cette classe se décide en temps polynomial. D'autre part, la transformation des relations de $\mathcal{H}$ ne produit que des clauses unitaires ou des clauses binaires dont les littéraux sont de la forme $(X^- r_1 Y^-)$ et $(X^+ r_2 Y^+)$ où les relations r_i appartiennent à $\{\leq, =, \neq\}$.

De la même manière, on identifie une autre sous classe de l'algèbre d'intervalles telle que la transformation en relations entre instants des expressions de cette classe produit des clauses unitaires (donc pas de disjonctions). Cette classe, appelée *l'algèbre d'intervalles restreinte* et notée $\mathcal{P}$, contient 187 relations (Granier, 1988) et a aussi d'intéressantes propriétés algorithmiques.

L'algèbre d'intervalles a aussi été étudiée sous un autre aspect : celui du voisinage conceptuel entre les différentes relations (Freksa, 1992). Une généralisation de cette approche qui prend aussi en compte les relations spatiales est décrite dans (Ligozat, 1994).

Algorithmes de propagation sur les intervalles

Allen (1983) a présenté en même temps que l'algèbre d'intervalles un algorithme déductif permettant d'inférer de nouvelles relations à partir d'un ensemble initial de relations. Cet algorithme (algorithme 1.1 page 30) est un algorithme de propagation par transitivité qui examine chaque triplet d'intervalles (i, j, k) et qui infère la relation de i à k par composition des relations de i à j et de j à k. Il s'agit en réalité d'une adaptation de l'algorithme de consistance de chemin PC-2 (Montanari, 1974). L'ensemble des relations est représenté dans un graphe dont les nœuds sont les intervalles et dont les arcs portent la relation existant entre leurs nœuds extrémités. La figure 1.4 présente un tel graphe de contraintes.

```
PC-2(E) :
    tant que E ≠ ∅
        soit r_{i,j} ∈ E ; E ← E \ {r_{i,j}}
        pour tout k tel que k ≠ i ∧ k ≠ j faire
            si r_{i,k} ≠ Φ alors
                E ← E ∪ Revise(i, k, r_{i,j} ∘ r_{j,k})
            si r_{j,k} ≠ Φ alors
                E ← E ∪ Revise(j, k, r_{j,i} ∘ r_{i,k})
        fin
    fin

Revise(i, j, r) :
    r ← r ∧ r_{i,j}
    si r = ε alors retourner (Echec)
    si r ≠ r_{i,j} alors
        r_{i,j} ← r
        retourner (r_{i,j})
    sinon retourner ()
```

Algorithme 1.1: *Algorithme de consistance de chemin par propagation de contraintes temporelles. L'argument E de* PC-2 *est l'ensemble des relations à ajouter, et $r_{i,j}$ désigne la relation de i à j.*

A chaque ajout d'une relation entre deux nœuds, que cette relation soit déduite par transitivité ou ajoutée explicitement, un test d'incohérence par intersection entre la nouvelle relation et la relation existante entre i et k permet de satisfaire un critère de consistance locale. Cette consistance est en fait limitée aux groupes de trois intervalles. L'algorithme termine soit lorsqu'une inconsistance est détectée, soit lorsque les relations portées par les arcs sont stables. La complexité en temps de l'algorithme d'Allen est $O(n^3)$ en termes d'opérations élémentaires (composition et intersection) entre relations.

Le problème ISAT peut être vu comme le test d'existence d'une solution au réseau de contraintes : dans ce cas une solution est un étiquetage consistant de chaque arc du graphe par une seule relation primitive présente parmi celles du graphe initial. La consistance d'un tel étiquetage unitaire est équivalent à l'existence d'une solution par projection des intervalles sur un axe temporel linéaire. Une autre appellation de ces solutions est *scénario consistant* (van Beek, 1989) : la recherche de ces scénarios consistants se fait par un processus d'énumération des éléments des contraintes et de retour arrière, ce qui construit donc un ensemble de graphes dont les contraintes sont des singletons, et pour lesquels la consistance est décidable en temps polynomial. Une amélioration proposée par (van Beek, 1990) est d'énumérer, non pas les relations primitives une à une, mais par groupe de relations suivant leur appartenance à la classe $\mathcal{P}$.

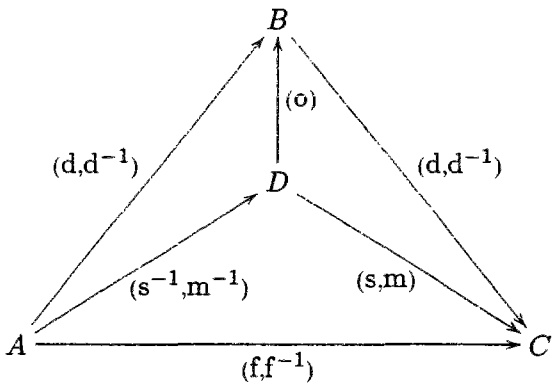


FIG. 1.5 - *Exemple d'inconsistance globale non détectée par l'algorithme d'Allen.*

Par rapport au deux problèmes ISI et ISAT, l'algorithme d'Allen n'a pas de très bonnes propriétes. L'algorithme n'est pas toujours correct pour ISAT car il ne permet de satisfaire qu'un critère de cohérence locale limitée à chaque groupe de 3 intervalles : il existe des ensembles de relations qui ne sont pas consistant globalement et pour lesquels l'algorithme d'Allen termine sans rapporter d'incohérences. La figure 1.5 présente un tel ensemble de relations identifié par (Allen, 1983) : il n'existe aucune solution de cet ensemble de relations avec soit (f), soit (f^{-1}) entre A et C : dans ce cas, la réponse au problème ISI n'est pas correcte non plus.

Pour ISI sur l'algèbre d'intervalle entière, l'algorithme d'Allen n'est pas correct car il laisse subsister dans l'étiquetage final des relations primitives qui ne font partie d'aucune solution. En réalité, les étiquetages obtenus sont minimaux par rapport à tous les sous-graphes de taille 3. Van Beek (van Beek, 1989) a proposé d'étendre ce critère à tous les sous-graphes de taille 4, et a proposé un algorithme en $O(n^4)$ qui, sans être correct non plus, fournit une meilleure approximation pour ISI.

Par contre, l'algorithme d'Allen est suffisant pour décider de ISAT sur l'algèbre d'intervalles restreinte $\mathcal{P}$, et le problème ISI sur cette même classe peut être décidé en $O(n^5)$. Van Beek (van Beek, 1989) a exhibé un contre exemple qui montre que l'algorithme d'Allen n'est pas complet pour ISI sur la classe $\mathcal{P}$. Par contre, l'algorithme de Van Beek en $O(n^4)$ qui permet de satisfaire la consistance des sous-graphes de taille 4 résoud correctement le problème ISI sur la classe $\mathcal{P}$. Van Beek donne aussi un algorithme en $O(n^2)$, adapté de l'algorithme de Dijkstra[2], pour résoudre ce même problème ISI sur $\mathcal{P}$ dans le cas où l'on ne s'intéresse qu'aux relations minimales entre un intervalle particulier et tous les autres.

De manière générale, le coût des algorithmes de propagation sur l'algèbre d'intervalles, bien que polynomial, reste prohibitif pour des applications concrètes où le nombre d'intervalles devient élevé. Des comparaisons entre plusieurs systèmes de gestion de relations temporelles sont présentées dans (Yampratoom & Allen, 1993),

2. L'algorithme de Dijkstra calcule le chemin de poids minimal d'un nœud à un autre dans un graphe valué positivement.

de piètres performances sont rapportées pour les systèmes à base d'intervalles et utilisant des algorithmes de propagation de contraintes: le système TIMELOGIC (Koomen, 1989) demande environ 200 secondes (soit plus de trois minutes) pour enregistrer et propager 1000 assertions de relations entre intervalles.

L'algèbre d'instants

L'algèbre d'instants a été proposée par Vilain et Kautz (1986). Elle a la même structure que l'algèbre d'intervalles sauf que les entités concernées sont des instants et que les relations primitives sont prises dans l'ensemble $B_P = \{<, =, >\}$. Cette algèbre contient donc 8 relations dont certaines sont abrégées ainsi:

$$\begin{array}{lcl} (<,=) & \rightarrow & (\leq) \\ (>,=) & \rightarrow & (\geq) \\ (<,>) & \rightarrow & (\neq) \end{array}$$

Comme précédemment, la relation universelle est notée Φ et la relation vide ϵ.

De même que pour l'algèbre d'intervalles, on définit les opérations de composition, d'intersection et d'union. Le principal intérêt de l'algèbre d'instants est le fait que la consistance peut être décidée en temps polynomial. Cet avantage compense l'expressivité plus faible de l'algèbre d'instants.

Algorithmes de propagation sur les instants

Parmi les algorithmes de propagation pour l'algèbre d'instants, on trouve en premier lieu les adaptations de l'algorithme de consistance de chemins (algorithme 1.1) où les opérations de composition et d'intersection sont celles de l'algèbre d'instants au lieu de celles sur l'algèbre d'intervalles. Comme précédemment, cet algorithme permet de déterminer la consistance (problème ISAT), mais il ne détermine les relations minimales (problème ISI) que si la relation $\neq$ n'apparait pas dans le graphe (van Beek, 1990), par contre, l'algorithme en $O(n^4)$ qui assure la consistance des groupes de quatre instants est correct et calcule bien les relations minimales sur la totalité de l'algèbre d'instants. Un algorithme de même complexité fonctionnant par manipulation formelle et élimination de variables est proposé dans (Koubarakis, 1992).

Van Beek (1990) propose un lemme sur les relations minimales dans un graphe vérifiant déja la consistance de chemin: un tel graphe est minimal si il ne contient pas de groupe de quatre points dans la configuration de la figure 1.6. La particularité de cette configuration est qu'une des relations $\leq$ n'est pas minimale et devrait être réduite à la relation $<$ seule. La preuve de ce lemme donnée par VanBeek n'est pas correcte, bien que le lemme lui-même soit vrai. Une preuve corrigée apparaît dans (Gerevini & Schubert, 1994). Sur la base de ce résultat, VanBeek donne un algorithme pour ISI en deux étapes: la première étape applique l'algorithme de

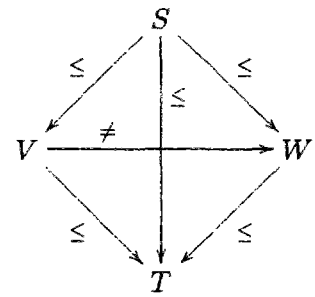


FIG. 1.6 - *Configuration consistante mais non minimale après application de l'algorithme de consistance de chemins. La relation de S à T ne peut par être $=$ car elle est alors contradictoire avec $V \neq W$: la relation minimale entre S et T est donc $<$.*

propagation puis, lors de la deuxième phase, recherche tous les groupes de 4 nœuds ayant une configuration analogue à celle présentée sur la figure 1.6. L'algorithme modifie alors les relations correspondantes dans le graphe. La complexité totale de la procédure est $O(n^4)$ à cause du coût de la recherche des sous-graphes de taille 4 : un argument avancé par Van Beek est qu'en pratique, la plupart du temps de calcul est passé dans l'algorithme de consistance de chemins, et donc que sa complexité en $O(n^3)$ domine celle en $O(n^4)$.

Cependant, l'inconvénient du coût en n^3 de la propagation reste présent, et n'est compensé que par la possibilité de pouvoir déterminer en temps constant la relation minimale entre deux instants par interrogation directe du réseau après propagation. C'est pourquoi d'autres approches qui ne travaillent pas sur le graphe complet des relations ont été proposées. Plus particulièrement, IxTeT (Ghallab & Mounir-Alaoui, 1989; Mounir-Alaoui, 1990), l'algorithme proposé par Van Beek (van Beek, 1990), ainsi que TIMEGRAPH-II (Gerevini & Schubert, 1993) rentrent dans cette catégorie et sont décrits dans la suite de ce paragraphe.

Ghallab et Mounir-Alaoui (1989) ont proposé dans le système IxTeT une approche novatrice qui exploite avantageusement de nombreuses propriétés des arborescences pour optimiser à la fois le temps de mise à jour et le temps de réponse à une requête sur la relation entre deux instants. Une description complète de IxTeT figure dans (Mounir-Alaoui, 1990). Seuls les quelques éléments permettant de comprendre cette approche sont rapportés ici.

Préalablement à toute construction, les relations initiales sont traduites en un graphe qui n'utilise que deux types d'arcs :

- des arcs de type $\leq$;
- des arcs de type $\neq$.

A l'aide de ces deux types d'arcs, toutes les relations de l'algèbre d'instants sont exprimables avec 0, 1, 2 ou 3 arcs entre deux instants. Par exemple, $u < v$ est

représenté par deux arcs $\leq$ et $\neq$ entre u et v, alors que $u = v$ est représenté par un arc $\leq$ de u à v et un arc de même type de v à u. La cohérence de l'ensemble de relations est vérifiée lorsque dans tout circuit d'arcs $\leq$, il n'existe aucun couple de nœuds reliés par un arc $\neq$. Si ce critère est satisfait, les cycles sont fusionnés en un seul nœud. Le graphe obtenu est donc acyclique et représente un ordre partiel des instants. Après avoir rajouté un instant origine t_0 on obtient un treillis dans lequel la détermination de la relation entre deux instants se ramène à une recherche de chemin.

L'optimisation de ces requêtes repose sur une propriété des arborescences d'après laquelle la relation d'ascendance se teste en temps constant si l'on dispose de deux indexations en pré-ordre et en post-ordre des nœuds de l'arbre. Le problème est alors d'extraire un arbre du treillis des instants. Cet arbre est défini par Ghallab et Mounir-Alaoui comme un arbre recouvrant maximal de racine l'instant origine t_0 et empruntant un nombre maximal d'arcs du treillis. Sa construction se fait par un algorithme en $O(m)$ si m est le nombre d'arcs du treillis. Au cours de cet algorithme, un schéma d'indexation original est mis en œuvre pour calculer incrémentalement les index, sans qu'il soit nécessaire de parcourir tout l'arbre pour recalculer ceux-ci, comme cela aurait été le cas si l'on avait gardé des index classiques en pré-ordre et en post-ordre.

Les performances d'IxTeT sont remarquables puisque le coût des opérations d'ajout et de réponse à une requête croît linéairement avec la taille du treillis. Ces avantages sont déterminants lorsque le nombre de relations à gérer devient très grands. Pour mémoire, rappelons que la gestion des relations entre intervalles ou instants à l'aide d'un algorithme de consistance de chemin par propagation a un coût en temps en $O(n^3)$ et en espace de $O(n^2)$.

Van Beek (1990) propose un algorithme qui permet de trouver un scénario consistant, c'est à dire une solution au problème de contraintes. L'idée est d'effectuer un tri topologique[3] des nœuds en suivant les relations $<$. Cependant, un tel tri ne peut pas être réalisé sans précautions à cause des autres relations par nature ambigües comme $\leq$ ou $\neq$.

Etant donné un ensemble de relations, l'algorithme de Van Beek se décompose en deux étapes. La première phase recherche toute les composantes fortement connexes[4] dans le graphe en ne considérant que les arcs étiquetés par $<$, $\leq$ ou $=$. Tous les nœuds faisant partie d'une de ces composantes sont équivalents car ils font tous partie d'un cycle formé de relations $<$, $\leq$ ou $=$. Suivant la forme des relations sur ce cycle, il est possible de déduire soit que tous les nœuds sont égaux lorsque les relations sur le cycle sont $\leq$ ou $=$, soit que le graphe est inconsistant si une des relations de la composante est $<$.

3. aussi appelé *linéarisation*. Un tel ordre des nœuds du graphe s'obtient en général en considérant l'inverse de l'ordre de dépilement des nœuds lors d'un parcours récursif en profondeur du graphe.

4. une composante fortement connexe est un ensemble de nœuds tel que quelque soient u et v appartenant à la composante, il existe un chemin de u à v et un chemin de v à u. En particulier, tout nœud d'une composante appartient à un cycle.

Les composantes fortement connexes $\{S_1, \ldots, S_m\}$ sont fusionnées chacune en un nœud S_i unique qui est relié aux autres nœuds S_j par un arc étiqueté par la relation

$$\bigwedge_{v \in S_i, w \in S_j} C(v, w)$$

où $C(v, w)$ est la relation initiale entre v et w. Si l'un de ces arcs doit être étiqueté par la relation vide ϵ suite à l'opération d'intersection, alors l'ensemble de relations est inconsistant.

La deuxième phase de l'algorithme travaille sur le graphe résultant de la fusion des composantes fortement connexes et remplace toutes les relations $\leq$ restantes par $<$ puis effectue ensuite un tri topologique des nœuds suivant les arcs portant la relation $<$. Ce tri topologique est un scénario consistant pour l'ensemble initial de relations. La restriction des relations $\leq$ en $<$ est licite car la première phase de l'algorithme garantit qu'il n'existe plus de relation $=$ dans le graphe obtenu, et que la restriction ainsi effectuée ne provoque pas l'apparition de nouvelles relations d'égalité.

La complexité de l'algorithme proposé par Van Beek pour calculer un scénario consistant est $O(n^2)$ où n est le nombre de points. La recherche des composantes fortement connexes coûte $O(n^2)$ et le tri topologique est aussi en $O(n^2)$. Si seul le problème ISAT est intéressant, la première phase de l'algorithme suffit.

Gerevini et Schubert (1993) proposent une troisième approche voisine de la précédente. A partir des relations, on construit un graphe dont les nœuds sont les instants, et les arcs sont étiquetés par les relations. Cette représentation sert de départ à la construction d'un « TL-graph » où ne subsistent plus que des arcs étiquetés par $\leq$, $<$ ou $\neq$. De plus, un TL-graph est partitionné en ensembles d'instants linéairement ordonnés, appelés *chaînes*. L'intérêt de ces chaînes est de limiter les recherches dans le graphe lors du calcul de la relation entre deux instants. En effet, pour deux instants u et v, un TL-graph implique $u \leq v$ (resp. $u < v$) si il existe un chemin de u à v formé d'arcs $\leq$ (resp. d'arcs $\leq$ et d'au moins un arc $<$), et la relation $u \neq v$ est vérifiée soit s'il y a un arc $\neq$ entre u et v, soit s'il existe un chemin d'arcs $\leq$ et d'au moins un arc $<$ de u à v ou de v à u. Le partitionnement en chaîne et une indexation des nœuds basée sur la longueur du plus grand chemin depuis un instant origine autorise alors la détermination en temps constant de la relation minimale entre deux instants.

La construction d'un TL-graph se fait en quatre étapes. La première teste la consistance des relations et repose comme précédemment sur la recherche des composantes fortement connexes. La recherche des composantes fortement connexes est la même que dans l'algorithme de Van Beek, et les relations sont inconsistantes si l'une de ces composantes contient un arc $<$ ou $\neq$. Si aucune incohérence n'est détectée, les composantes sont fusionnées en un nœud unique. La deuxième phase indexe les nœuds sur la base de la longueur du plus grand chemin depuis l'origine, puis la troisième phase construit le partitionnement en chaînes de manière à minimiser à la fois le nombre total de chaînes et le nombres d'arcs entre chaînes. La dernière phase

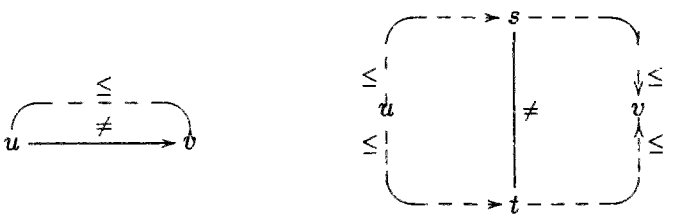


FIG. 1.7 - *Deux cas de relations $u < v$ implicites prises en compte par l'algorithme de Gerevini et Schubert. Les traits pointillés désignent l'existence d'un chemin.*

de l'algorithme a pour but d'expliciter les relations $<$ implicites dans les deux cas montrés sur la figure 1.7. Ces relations sont ajoutées au TL-graph.

Cette phase, qui était aussi présente de manière moins complète dans l'algorithme de Van Beek, a l'avantage d'un coût moins élevé grâce au partitionnement en chaînes. De manière générale, cette structuration du graphe permet à Gerevini et Schubert d'annoncer des complexités en temps linéaires pour la plupart des phases de leur algorithme.

Représentations mixtes : symboliques et numériques

Jusqu'à présent, seules les relations symboliques ont été envisagées. Or le temps a aussi une connotation numérique, puisque l'on utilise souvent des expressions arithmétiques pour représenter des dates. Si t est un instant, $t + 2$ est aussi un instant dont les liens avec t ne sont pas exprimables complètement à l'aide des relations d'ordre symboliques $<$, $\leq$ ou $=$. La relation $t < t + 2$ est juste mais elle induit une perte d'information.

Les approches décrites dans ce paragraphe s'attaquent à ce problèmes et proposent diverses solutions permettant de prendre en compte simultanément des relations symboliques et des relations numériques. Le formalisme de (Dechter, Meiri, & Pearl, 1991) permet de représenter des relations quantitatives entre instants, alors que la méthode proposée par (Kautz & Ladkin, 1991) exprime des relations symboliques entre intervalles, et numériques entre bornes des intervalles. Meiri (1991) se place aussi dans le cadre des relations entre intervalles et instants combinés.

Le modèle développé par (Dechter et al., 1991) est celui des TCSP (*Temporal Constraint Satisfaction Problem*). Comme dans toutes les approches précédentes, il relie les nœuds, représentant des instant, par des arcs étiquetés par un ensemble d'intervalles numériques. Chaque nœud corrrespond, dans le formalisme des CSP, à une variable dont le domaine est continu. Entre deux nœuds X et Y, l'arc orienté de X vers Y et étiqueté par

$$\{[a_1, b_1], \ldots, [a_n, b_n]\}$$

correspond à la disjonction

$$(a_1 \leq Y - X \leq b_1) \vee \ldots \vee (a_n \leq Y - X \leq b_n)$$

et contraint donc la distance entre X et Y.

A cause de la présence des disjonctions, décider de la consistance d'un tel problème est NP-dur (Davis, 1987). La solution proposée par (Dechter et al., 1991) repose sur une énumération avec retour arrière qui génère un nombre exponentiel de graphes dont les arcs ne sont étiquetés que par un seul intervalle. Ce cas particulier de problème temporel, sans disjonction, appelé STP (*Simple Temporal Problem*), possède d'intéressantes propriétés algorithmiques. Décider de la consistance d'un ensemble de relations exprimées dans ce formalisme peut se faire en temps polynomial grâce à l'algorithme de consistance de chemins PC-2 où les opérations de composition et d'intersection de contraintes sont définies ainsi :

$$\begin{aligned}(x : [a_1, b_1] : y) \circ (y : [a_2, b_2] : z) &= x : [a_1 + a_2, b_1 + b_2] : z \\ (x : [a_1, b_1] : y) \wedge (x : [a_2, b_2] : x) &= x : ([a_1, b_1] \cap [a_2, b_2]) : y\end{aligned}$$

Un tel algorithme est complet et calcule aussi les intervalles minimaux sur les arcs du graphe.

L'expressivité du formalisme des STP est légèrement inférieure à celle de l'algèbre d'instants de Vilain et Kautz (1986) : la seule relation qui n'est pas représentable est $\neq$ car elle correspond alors à un arc avec une disjonction de deux intervalles.

Kautz et Ladkin (1991) proposent une approche différente, en ce sens où elle cherche à incorporer dans un même formalisme à la fois les relations de l'algèbre d'intervalles, et des relations métriques, semblables à celles de (Dechter et al., 1991), entre les instants correspondant aux bornes des intervalles. L'algorithme proposé par Kautz et Ladkin repose sur une propagation des contraintes de manière simultanée dans deux réseaux : celui des contraintes entre intervalles et celui des contraintes métriques entre instants. A chaque étape de l'algorithme, s'effectue une conversion des contraintes métriques dans le graphe des contraintes entre intervalles et réciproquement, puis une propagation dans chaque graphe jusqu'à stabilisation des deux réseaux de contraintes. Les opérations de conversion sont les plus délicates puisqu'elles doivent provoquer une perte d'information minimale. La conversion des contraintes métriques retourne le plus petit ensemble de contraintes entre intervalles impliquées par les contraintes métriques, et la conversion inverse retourne les contraintes métriques les plus fortes (intervalles minimaux) impliquées par les contraintes entre intervalles. Ces deux transformations sont polynomiales et ont des complexités respectives en $O(n^3)$ et $O(n^2)$, où n est le nombre d'intervalles représentés.

L'algorithme présenté par Kautz et Ladkin est correct dans la mesure où il ne déduit jamais d'information fausse. Cependant, sa complétude pour ISAT (test de consistance) et même sa correction pour ISI (relations minimales) est un problème ouvert pour lequel Kautz et Ladkin n'apportent aucun résultat.

Un autre formalisme ayant la même puissance d'expression est présenté par Meiri (Meiri, 1991). A la différence du précédent, il n'y a pas de séparation des représen-

TAB. 1.2 - *Complexité du test de consistance dans le formalisme de Meiri. AC désigne la consistance d'arcs et PC la consistance de chemins. Dans les évaluations de complexité, n est le nombre de variables (instants ou intervalles), e le nombre d'arcs et k est la taille maximale des domaines*

	Domaines finis	Intervalle unique	Intervalles multiples
Relations entre instants sans $\neq$	AC $O(ek \log k)$	AC + PC $O(n^3)$	AC + PC $O(n^4k^2)$
Relations entre instants	NP-complet	AC + PC $O(n^3)$	NP-complet
Relations entre instants et intervalles	NP-complet	NP-complet	NP-complet

tations entre les deux types de contraintes. Instants et intervalles sont représentés de manière identique, et en plus des relations entre intervalles (celles de l'algèbre d'intervalles) et entre instants (algèbre d'instants), Meiri définit des relations entre instants et intervalles. Parmi ces relations, on trouve par exemple le fait qu'un instant soit avant, après un intervalle, qu'il appartienne à un intervalle ou qu'il soit égal à une de ses bornes. La restriction de ce formalisme aux intervalles donne l'algèbre d'intervalles et de même pour les instants.

Les algorithmes proposés par Meiri sont des adaptations des algorithmes de consistance d'arcs AC-3 (Mackworth & Freuder, 1985) et de consistance de chemin PC-2. La composition des contraintes fait appel à différentes tables de transitivité suivant le type des éléments contraints. En termes de difficulté, décider de la consistance d'un ensemble de ces relations mixtes est bien sûr toujours un problème NP. Meiri propose tout un ensemble de résultats de complexité suivant les relations présentes dans le problème, la forme des domaines (finis ou continus) des variables, et des contraintes (un ou plusieurs intervalles) : ces résultats sont regroupés dans la table 1.2

Le principal intérêt du formalisme de Meiri est d'intégrer les instants et les intervalles dans un même système. Quelques gains en complexité par rapport au système de Kautz et Ladkin sont obtenus, en particulier parce que les conversions entre contraintes quantitatives et contraintes symboliques sont plus simples dans l'approche de Meiri.

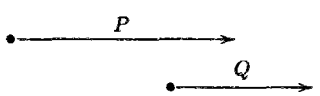
1.2.2 Le TMM de Dean et McDermott

Le TMM (abréviation de *Time Map Manager*)(Dean & McDermott, 1987) est un système complet de raisonnement temporel qui intègre une gestion des relations temporelles par un graphe de contraintes métriques analogues au modèle des STP de

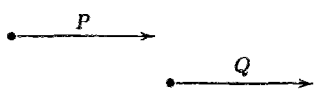
(Dechter et al., 1991), des moteurs d'inférences en chaînage avant et arrière acceptant des règles décrites sous forme de clauses, et un système de maintien de vérité adapté pour le raisonnement temporel.

Dans le langage du TMM, une *persistance* désigne l'association entre un fait et un intervalle. Un intervalle est un ensemble d'instants représenté par ses deux bornes. La logique sous-jacente au TMM est proche de celle de McDermott (1982).

La gestion des relations temporelles entre instants se fait à l'aide d'un graphe dont les nœuds sont des instants et les arcs portent un intervalle qui encadre la distance entre les deux instants reliés par l'arc. Ce formalisme est analogue à celui des « Simple Temporal Problem » (STP) de (Dechter et al., 1991). Cependant, Dean n'utilise pas d'algorithme de consistance de chemin pour gérer ce graphe : la méthode repose sur la recherche des chemins entre deux nœuds. Le long de chacun de ces chemins, l'addition entre elles des bornes inférieures et supérieures produit une estimation de la distance entre les deux instants extrémités du chemin. Ce calcul est la généralisation de la composition des contraintes métriques tel qu'elle a été définie au paragraphe 1.2.1 (page 36). Le graphe est consistant lorsque, pour toute paire d'instants u, v l'intersection de toutes les estimations de $v - u$ produites par cette méthode pour chaque chemin de u à v est non vide : ce qui donne un critère d'inconsistance analogue à celui du formalisme des STP. En complément de ce mécanisme, le TMM incorpore dans la gestion des relations temporelles un mécanisme de coupure (*clipping*) entre intervalles portant sur deux faits déclarés contradictoires. Par défaut la borne supérieure d'une persistance n'est pas contrainte et s'étend aussi loin que possible : la coupure impose à la disjonction des deux intervalles de vérité des faits contradictoires. Soient par exemple P et Q deux faits contradictoires dont les persistances réciproques sont enregistrées ainsi dans le TMM :



Pour résoudre la contradiction, le mécanisme de coupure impose alors une contrainte d'antériorité entre la borne supérieure de la persistance de P et la borne inférieure de celle de Q. Les persistances sont alors dans la configuration suivante :



Divers raffinements de ce schéma sont proposés dans (Groß, Hertzberg, Materne, & Voß, 1992), par exemple en autorisant des coupures sur la deuxième persistance (ici Q) suivant les configurations réciproques des persistances et les faits concernés.

Le TMM incorpore deux mécanismes de déduction : le premier travaille en chaînage arrière à partir de requêtes sur les faits et leurs occurrences, et le deuxième fonctionne en chaînage avant pour produire une « simulation » du système modélisé. Ces deux mécanismes utilisent le système de gestion de relations temporelles pour tester la vérité des premisses, la consistance de certaines relations etc. L'inférence

en chaînage avant peut être contrôlée en faisant appel à un opérateur modal M dans les prémisses qui fait référence à la consistance de l'assertion plutôt qu'à sa vérité. Cet opérateur permet de faire des hypothèses en cours de raisonnement.

Pour effectuer des déductions correctes et tenir à jour les dépendances entre les informations, le TMM utilise un système de maintien de vérité adapté à l'usage du système de contraintes. Comme dans le TMS de (Doyle, 1979), les informations sont insérées dans un graphe de dépendance, y compris les contraintes posées lors du *clipping* des persistances.

Le TMM est un système complet et son ambition est de pouvoir résoudre de nombreux problèmes du raisonnement temporel, comme la prédiction, ou liés au raisonnement temporel comme la planification. Malgré cela, le TMM manque d'un cadre intégrateur entre la gestion des relations temporelles et les mécanismes d'inférence : la présentation du TMM dans (Dean & McDermott, 1987) est peu claire sur cette intégration.

1.2.3 Approches par la théorie des modèles

Les approches par la théorie des modèles partent d'une idée qui recoupe assez largement de nombreux travaux sur le raisonnement non monotone. En particulier, Shoham (1990) relie explicitement le raisonnement non monotone et le raisonnement sur les liens causaux. Une motivation de cette approche vient du constat que les descriptions par des théories logiques d'univers en évolution ne sont pas assez précises dans la mesure où la sémantique habituelle des langages logiques ne rejoint pas toujours les conclusions intuitives que l'on attend de la formalisation. En effet, tous les modèles de ces théories ne correspondent pas forcément à l'intuition qui est derrière l'écriture de la théorie. Soit par exemple le problème très connu du « *Yale Shooting Problem* » (YSP) (Hanks & McDermott, 1987) : au temps 0, on a un pistolet chargé et une dinde vivante, après un temps d'attente, on tire sur la dinde avec le pistolet. La description de ce problème par (Hanks & McDermott, 1987) utilise le calcul de situations de McCarthy : le problème est de formaliser que ce qui était vrai dans la situation au temps 0 (le pistolet est chargé) l'est toujours dans la situation atteinte après le temps d'attente. Ceci est un problème de *persistance*: Parmi tous les modèles de cette théorie, il en existe certains où le pistolet est déchargé (pour une raison inconnue) pendant le temps d'attente. Le modèle sous-jacent à l'intuition de ce problème est celui où le pistolet est toujours chargé au moment du coup de feu. Plus généralement, ce problème est lié à l'existence de plusieurs extensions (Morris, 1988) dans des théories écrites avec des logiques non monotones, la logiques des défauts (Reiter, 1980), ou plus simplement par l'usage d'opérateurs de négation dont le traitement produit par essence un raisonnement non monotone (négation par échec, par exemple). De la même manière, l'usage de l'implication logique, ou A cause B est écrit $A \supset B$ et interprété par $\neg A \vee B$, n'est pas adéquat pour représenter les liens de causes à effet : ici, il est possible que l'implication soit vérifiée et que B soit vrai sans que A ne le soit, ce qui est contraire à l'idée initiale. Shoham identifie

dans (Shoham, 1990) dix propriétés de la relation de cause à effet.

Indépendamment du formalisme utilisé pour la représentation de ce problème, la persistance est un problème récurrent et qui n'est pas solvable dans la définition sémantique du formalisme puisque, en général, tous les modèles sont équivalents pour celle-ci. En introduisant dans la définition sémantique de la logique une relation d'ordre entre interprétations, on définit une sémantique préférée qui ne s'intéresse qu'aux modèles minimaux (ou maximaux) suivant cet ordre. Cet ordre doit être défini de manière à ne conserver que les modèles qui correspondent à ce qu'on attend de la description.

Le premier critère a été proposé par Hanks et McDermott (1987) : ils proposent de minimiser les changements (*minimization of change*), et donc par conséquences les changements anormaux, comme le fait que le pistolet se décharge sans raison valable prévue par la théorie. Shoham (Shoham, 1988) a proposé un autre critère où l'on préfère que les changements arrivent le plus tard possible plutôt que le plus rarement. Son critère, appelé l'*ignorance chronologique*, est défini dans le cadre d'une logique modale de la connaissance qui étend une logique temporelle réifiée définie dans (Shoham, 1987) (et présentée page 20 de ce mémoire). L'usage des opérateurs modaux permet de spécifier des hypothèses potentielles dans le corps des règles causales, comme dans la règle suivante

$$\Box TRUE(t, \textit{Press-button}) \wedge \Diamond TRUE(t, \textit{All-working}) \supset \Box TRUE(t+1, \textit{Bell-rings})$$

où $TRUE(t, \textit{All-working})$ sera éventuellement supposé si l'inférence correspondante satisfait le critère d'ignorance chronologique. Cet usage d'une logique modale n'est pas très clair, et Galton (Galton, 1991a) fait remarquer que l'interprétation des opérateurs modaux, soit comme la connaissance, soit comme la croyance, donne des résultats qui ne concordent pas avec la théorie. Remarquant que les critères précédents n'étaient pas adaptés aux cas où l'information initiale est incomplète ou ambigüe (contrairement à l'exemple du YSP), Haugh (1987b) a proposé un autre critère qui cherche à minimiser les causes des changements.

Pratiquement parlant, il n'y a pas d'implémentation efficace de procédure d'inférence ou de décision pour ces formalismes. Shoham décrit une procédure qui n'accepte qu'un sous ensemble de son langage, où en particulier l'on évite les boucles dans les règles. Sandewall décrit dans (Sandewall, 1988a, 1988b) une logique et un critère de préférence en deux étapes qui maximise d'abord les persistances attendues, puis qui minimise ensuite les actions sur l'univers. Une procédure de décision est décrite dans (Sandewall, 1988c) et son implantation dans (Hansson, 1990). Cette procédure a un coût en mémoire exponentiel puisqu'elle construit, sur la base de la méthode des tableaux (Reeves, 1987), tous les modèles de la théorie puis trie ensuite ceux-ci d'après les critères de préférence.

1.3 Conclusions

La remarque principale que l'on peut faire après ce chapitre est qu'il existe toujours un compromis à trouver entre les capacités expressives d'un formalisme et la difficulté des inférences dans celui-ci. En particulier, si il existe des procédure d'inférence polynomiale pour des langages très réduits exprimant des relations simples entre des instants, le même type de langage pose un problème difficile lorsqu'il concerne des intervalles. Pour des formalismes plus complets et très puissants comme les logiques temporelles réifiées, il n'existe aucune procédure efficace.

Dans le cas des logiques temporelles réifiées, il paraît illusoire de compter sur les capacités grandissantes des systèmes de démonstration automatique de théorèmes pour mettre en œuvre le raisonnement temporel. De tels systèmes ont en général beaucoup de difficultés à prendre en compte les aspects fortement combinatoires des problèmes, ce qui arrive souvent dès que l'on manipule des relations d'ordre ou d'égalité. Cependant, de telles relations peuvent être gérées par des techniques algorithmiques, comme la propagation de contraintes, et nous proposerons dans la suite de ce mémoire un cadre qui permet d'intégrer l'usage de ces procédés algorithmiques dans un système déductif basé sur la logique. Un tel cadre se révèle bien adapté pour le raisonnement temporel avec des logiques temporelles réifiées.

Chapitre 2

Résolution et logiques temporelles

Dans ce chapitre, nous allons étudier quelques moyens de mise en œuvre concrète du raisonnement temporel. Après avoir motivé notre approche, nous nous restreindrons aux logiques temporelles réifiées et aux inférences par résolution. Nous proposons une solution simple au problème d'efficacité de telles inférences. Cette solution est basée sur la résolution avec contraintes, et nous montrons sa faisabilité et l'intérêt qu'elle présente si l'on désire raisonner avec des logiques temporelles réifiées. Un autre intérêt de cette solution est d'intégrer formellement l'usage des systèmes de gestion de contraintes temporelles dans un mécanisme déductif basé sur la résolution.

Les limites de cette approche sont soulignées et nous présentons différentes manières d'étendre les algorithmes usuels de propagation de contraintes pour accepter des expressions temporelles plus riches, en particulier faisant intervenir des termes fonctionnels.

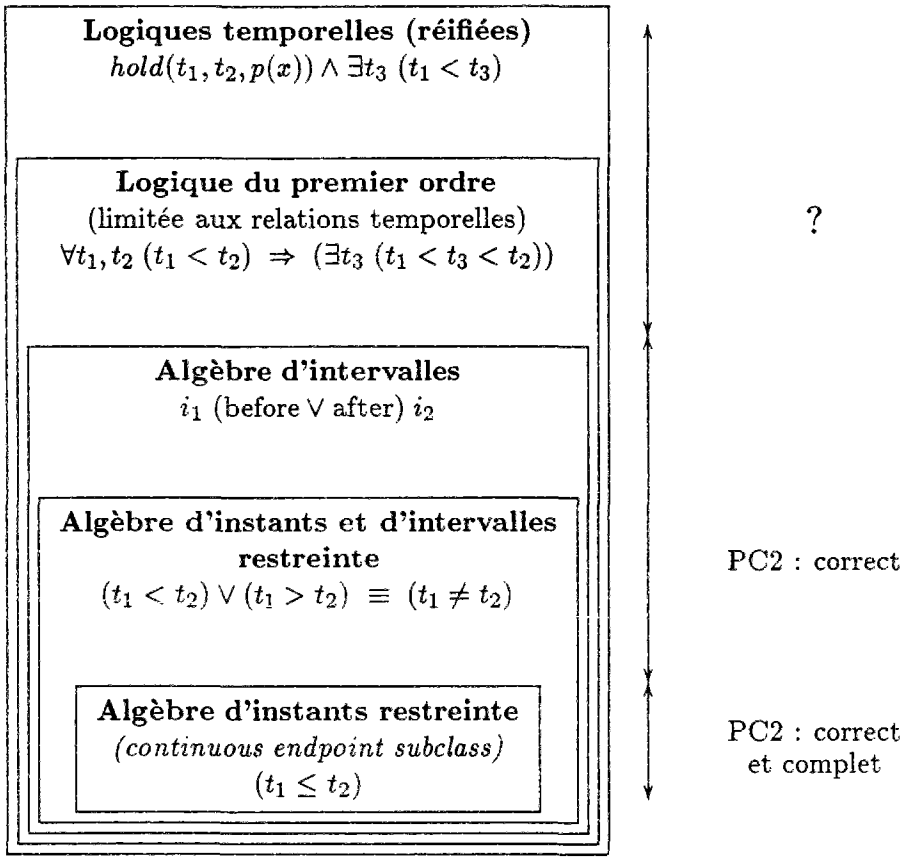


FIG. 2.1 - *Les langages (logiques) utilisables pour le raisonnement temporel et classés par expressivité. La colonne de droite indique l'existence d'une (ou plusieurs) procédures à coût polynomial permettant de décider de la satisfiabilité pour chacun de ces langages.*

2.1 Motivations

A ce point de notre exposé, le paysage est le suivant : l'on dispose d'une variété de langages permettant l'expression du temps et de concepts liés au temps, et ces langages peuvent être classés suivant la figure 2.1 par ordre d'expressivité croissante. En vis à vis de cette classification, inspirée de (Vila, 1994), nous avons porté des indications sur l'existence d'une procédure de test de la satisfiabilité, et dont le coût serait polynomial.

Cette classification est bien entendue très schématique, puisqu'il existe malgré tout des procédures de preuve pour des langages comme la logique ETL de (Sandewall, 1988a), par exemple. Encore ce dernier langage est-il assez limité syntaxiquement, et la procédure correspondante a un coût qui peut devenir exponentiel. Néanmoins, il existe un seuil d'expressivité en delà duquel la mise en pratique du raisonnement temporel devient très difficile. A quels problèmes se heurte cette mise

en œuvre?

efficacité : Il est évident qu'une procédure de preuve polynomiale ne peut s'envisager que pour un langage très restreint, comme l'algèbre d'instants. Néanmoins, en abandonnant cette exigence sur le coût d'une procédure, et donc en se tournant vers des techniques utilisées en démonstration automatique ou en programmation logique, on reste confronté à un manque d'efficacité non négligeable. Ce problème sera plus particulièrement examiné dans la suite de ce chapitre;

persistance : la persistance est un exemple typique de raisonnement par défaut. Cette constatation débouche alors sur tout le cortège de problèmes pratiques rencontrés lors de la mise en pratique de tels modes de raisonnement. Faut-il utiliser des sémantiques préférentielles (Haugh, 1987b; Shoham, 1988; Sandewall, 1988b), sachant que l'implantation en est très difficile (Hansson, 1990), ou s'orienter vers le raisonnement hypothétique comme le pense (Poole, 1988)?

sémantique non classique : les logiques temporelles réifiées, de même que celles que l'on peut construire par la méthode des arguments temporels (Haugh, 1987a), ont une sémantique qui n'est pas habituelle, où l'habitude désigne la sémantique Tarskienne de la logique du premier ordre. En particulier, et ceci est important si l'on songe à appliquer des méthodes de démonstration inspirées de la programmation logique, les structures d'interprétations proposées pour ces langages ne sont pas des interprétations de Herbrand. C'est à dire que certains termes ne peuvent pas être liés à eux mêmes par le processus d'interprétation. Ceci est particulièrement visible dans le cas des logiques temporelles réifiées que décrit Shoham (1987).

Le but de ce chapitre est donc d'étudier ce qui permettrait de progresser vers une mise en œuvre concrète d'un outil de raisonnement temporel. Nous fixerons donc notre attention sur les langages qui semblent les plus intéressants, au regard du critère « expressivité », et qui sont les logiques temporelles réifiées telles que les a présenté (Shoham, 1987). Certains éléments de notre analyse sont aussi valables pour la méthode des arguments temporels (Haugh, 1987a). De la même manière, nous nous restreignons aux systèmes déductifs basés sur le principe de résolution, à la fois parceque ce principe est simple, contrairement à la méthode des tableaux (Reeves, 1987) par exemple, et parcequ'il existe de nombreux travaux sur l'implantation de démonstrateurs par résolution, par exemple OTTER (McCune, 1990), et sur une restriction de la resolution aux clauses de Horn : la programmation logique (Lloyd, 1987).

2.2 Des difficultés de la résolution avec des logiques temporelles

Si l'intérêt du principe de résolution comme mécanisme d'inférence n'est plus à établir, en revanche sa mise en œuvre se heurte souvent à des problèmes induits par certaines caractéristiques des théories manipulés : présence ou non d'un prédicat d'égalité, clauses *auto-résolvantes*, etc. Ces caractéristiques multiplient les possibilités de résolution et contribuent à augmenter significativement la taille de l'espace de recherche. Ces problèmes doivent être résolus au niveau du contrôle de l'inférence par le biais de stratégies et d'heuristiques adaptées.

Il y a deux raisons qui limitent l'usage de la résolution, et plus généralement d'un démonstrateur générique par résolution, comme OTTER (McCune, 1990), pour le raisonnement temporel. Le premier problème vient de l'usage de relations d'ordre entre les instants, ainsi que de l'égalité. Le deuxième est plus général et provient de la difficulté qu'il y a à modéliser une persistance « par défaut » avec une sémantique classique non préférentielle.

2.2.1 Relations d'ordre et d'égalité

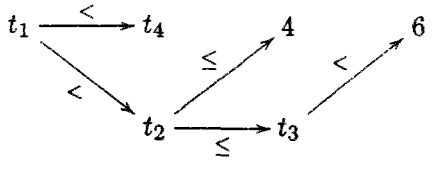
L'utilisation des symboles de relation $<$ et $\leq$ suppose que l'on soit capable de raisonner avec les relations d'ordre et d'égalité dans un mécanisme par résolution. Pour l'égalité, les solutions proposées dans la littérature, celles bâties autour de la *paramodulation* (Wos & Robinson, 1970), ou des systèmes de réécriture (Hsiang, 1987), sont en général lourdes et coûteuses. En ce qui concerne la relation d'ordre $<$, les axiomes qui la définissent ont, lorsqu'ils sont écrits sous forme clausale, le même défaut que les axiomes d'égalité, c'est à dire que ces clauses sont auto-résolvantes (*self-resolving*) : deux copies d'une même clause (par renommage des variables) peuvent produire un résolvant, qui, de manière générale, a peu de chances d'être utile pour une déduction particulière.

Soit par exemple les axiomes suivants qui décrivent (partiellement) les relations d'égalité $=$, d'ordre $<$ et $\leq$ entre instants :

$$\begin{array}{ll} \forall\, t \;\; t \leq t & (O_1) \\ \forall\, t_1, t_2, t_3 \;\; (t_1 < t_2) \wedge (t_2 < t_3) \;\Rightarrow\; (t_1 < t_3) & (O_2) \\ \forall\, t_1, t_2 \;\; [(t_1 \leq t_2) \wedge (t_2 \leq t_1)] \;\Leftrightarrow\; (t_1 = t_2) & (O_3) \\ \forall\, t_1, t_2 \;\; (t1 \leq t2) \;\Leftrightarrow\; [(t_1 < t_2) \vee (t_1 = t_2)] & (O_4) \end{array}$$

et la situation décrite à l'aide des relations suivantes :

$$\begin{array}{ll} t_1 < t_4 & \\ t_1 < t_2 & t_2 \leq 4 \\ t_2 \leq t_3 & t_3 < 6 \end{array}$$



Une conséquence de cette configuration est, par exemple, $t_1 \leq 5$: un démonstrateur par résolution classique comme OTTER demande environ 1 seconde[1] pour le prouver, et génère 390 clauses avec la stratégie *set of support*[2]. Encore faut-il que le démonstrateur possède des prédicats internes « évaluables » permettant, *in fine* après application de la transitivité, de passer de la clause $\neg(4 < 5)$ à la clause vide pour finaliser la preuve.

Si de plus, l'on admet que les instants soient représentés par des expressions arithmétiques, même aussi simples que $t + n$ ou $t - n$ où le deuxième membre est toujours un nombre, la tâche devient hors de portée d'un démonstrateur comme OTTER.

Certaines formes d'axiomes utilisées dans les logiques temporelles posent aussi un problème du même genre : soit par exemple l'axiome suivant sur les propriétés « liquides » telles que les définit Shoham :

$$\forall t_1, t_2, t_3, p \quad TRUE(t_1, t_2, p) \wedge (t_1 \leq t_3 \leq t_2) \Rightarrow TRUE(t_3, p)$$

Comme $TRUE(t_3, p)$ n'est qu'une abréviation de $TRUE(t_3, t_3, p)$, un tel axiome produit une clause auto-résolvante, c'est à dire qu'on peut appliquer la résolution entre cette clause et une copie d'elle-même. Seule une stratégie adaptée pourra empêcher de tels pas de résolution, ou encore l'utilisation récurrente d'une telle clause dans une même branche de l'espace de recherche.

2.2.2 Tout n'est pas toujours vrai...

Le deuxième problème posé par l'usage d'un démonstrateur de théorèmes pour le raisonnement temporel est lié à la difficulté qu'il y a à formaliser la persistance de telle manière que les effets attendus d'un scénario soient des théorèmes de la théorie logique qui formalise le scénario, c'est à dire des formules vraies dans tous les modèles. L'exemple qui suit illustre ceci.

Cet exemple est décrit dans un langage qui est une logique temporelle réifiée très simple, dans laquelle nous donnerons une formalisation possible de la persistance. Cette logique ne permet que de qualifier des propositions par des points ou des intervalles, ceux-ci étant désignés à l'aide de leur deux points extrémités. L'exemple à traiter suppose un raisonnement sur la persistance d'une information.

1. temps CPU mesuré avec la version 2.2 de OTTER sur une SparcStation IPC.

2. un démonstrateur par résolution qui utilise la stratégie SOS manipule deux ensembles de clauses distinctes : C et S. A chaque cycle, on choisit une clause dans S, et on génère tous les résolvants entre cette clause et les clauses de C. Ces résolvants sont ajoutés à S, et la clause choisie est enlevée de S et ajoutée à C. Ceci continue jusqu'à la génération d'une clause vide. En général C contient les axiomes du domaine, et S les clauses propres à l'exemple traité, en particulier la négation de la conjecture à prouver.

Une logique temporelle simple

La logique temporelle que nous allons utiliser est basée sur les instants, et ne permet de qualifier temporellement que des propositions. Ainsi, l'on évite le cas difficile où les termes atemporels auraient une interprétation dépendante du temps (désignateurs « flexibles ») : un simple algorithme d'unification comme l'on en trouve dans un démonstrateur classique est alors capable de décider d'une égalité entre deux propositions, et ceci sans contredire la sémantique de ces propositions.

Dans un souci d'expressivité, nous introduisons les expressions suivantes dans la logique que nous proposons :

$t_1 < t_2$, $t_1 \leq t_2$, $t_1 = t_2$: signifient respectivement que l'instant t_1 est avant l'instant t_2, que t_1 est antérieur ou égal à t_2, et que t_1 est égal à t_2 ;

hold(t_1,t_2, p) : signifie que p est vrai de t_1 à t_2 et exactement sur cet intervalle (ni plus, ni moins) ;

begin(t, p) : signifie que p commence à être vrai à partir de l'instant t ;

end(t, p) : signifie que p cesse d'être vrai juste après l'instant t ;

true(t,p) signifie que p est vrai au temps t comme conséquence du fait que p est vraie pendant une période qui comprend la date t ;

occur(t, p) signifie que p est ponctuellement vrai à l'instant isolé t.

Les liens entre ces différentes expressions sont décrits à l'aide des relations suivantes :

$$\forall\, t_1, t_2, p \;\; hold(t_1, t_2, p) \;\Rightarrow\; (t_1 < t_2) \tag{1}$$

$$\forall\, t_1, t_2, p \;\; hold(t_1, t_2, p) \;\Leftrightarrow\; \left(\begin{array}{l} begin(t_1, p) \;\wedge \\ (\forall\, t_3 \;\; (t_1 \leq t_3 \leq t_2) \Rightarrow true(t_3, p)) \;\wedge \\ end(t_2, p) \end{array} \right) \tag{2}$$

Ce langage est donc assez proche de celui défini dans (Ribeiro & Porto, 1991), puisqu'il met l'accent sur les intervalles de vérité maximaux (exprimés par *hold*()), et avec pour différence l'introduction de prédicats comme *begin*(), *end*(), etc.

Les relation d'ordre $<$ et $\leq$, et d'égalité sont décrites par exemple à l'aide des axiomes O_1 à O_4 du paragraphe précédent, et on suppose de plus que le temps est dense :

$$\forall\, t_1, t_2 \;\; (t_1 < t_2) \Rightarrow [\exists\, t_3 \;\; (t_1 < t_3 < t_2)] \tag{3}$$

A priori, rien n'empêche d'autoriser plusieurs périodes de vérité pour une même proposition. Compte tenu de nature maximale de ces périodes, une précaution doit être prise pour que ces périodes ne se recouvrent pas : en effet, si la fin (ou le début) d'une période I_1 où p est vraie est incluse dans une autre période I_2, il y a une ambiguïté entre la vérité de p à l'intérieur de I_2 et la non-vérité de p aux points

extérieurs à I_1 et infiniment proches du début (ou de la fin) de I_1. Ces situations sont exclues à l'aide des deux axiomes suivants :

$$\forall\, t_1, t_2, t_3, p \;\; [hold(t_1, t_2, p) \wedge begin(t_3, p)] \Rightarrow [(t_3 \leq t_1) \vee (t_2 < t_3)] \qquad (4)$$
$$\forall\, t_1, t_2, t_3, p \;\; [hold(t_1, t_2, p) \wedge end(t_3, p)] \Rightarrow [(t_3 < t_1) \vee (t_2 \leq t_3)] \qquad (5)$$

Pour modéliser la persistance, nous adoptons une solution analogue à celle du TMM, c'est à dire que dès lors que le début d'une période de vérité est connu, on affirme l'existence de la fin de cette période. La seule contrainte imposée à cet instant de fin est d'être postérieur au début de la période :

$$\forall\, t_1, p \;\; begin(t_1, p) \Rightarrow [\exists\, t_2 \;\; (t_1 < t_2) \wedge hold(t_1, t_2, p)] \qquad (6)$$

L'interprétation des formules construites à l'aide de ce langage peut se faire de manière analogue à celle de Shoham (Shoham, 1987). On suppose un ensemble W de points de temps qui comprend aussi bien l'ensemble des réels qu'un ensemble de constantes « symboliques ». Cet ensemble est partiellement ordonné, et cet ordre étend en particulier celui sur les nombres réels. Ensuite, on associe à chaque proposition son interprétation comme un élément d'un domaine D, et chaque élément de D est associé à un ensemble d'éléments de W, ou à un ensemble de couples d'éléments de W. Dans le premier cas, cette association décrit la vérité de la proposition à une date isolée (et sert donc à interpréter le prédicat *occur*), et dans le deuxième cas, on décrit la vérité sur des périodes de temps (et ceci permet d'interpréter *hold*, *begin*, *end* et *true*).

De telles structures d'interprétation ne sont que la formalisation d'une « carte temporelle » décrivant les relations connues entre les points de temps et la répartition de la vérité des propositions par rapport à ces points.

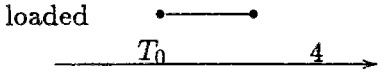
Exemple de raisonnement

Considérons maintenant le scénario très simple, inspiré du *Yale Shooting Problem* (Hanks & McDermott, 1987), où un pistolet est chargé à une date T_0, que l'on supposera antérieure à la date 2 par exemple, puis un coup de feu est tiré, au temps 4, à l'aide de ce même pistolet. Une conséquence naturelle de ce scénario est que le pistolet sera déchargé juste après le temps 4, c'est à dire qu'il est resté chargé avant le coup de feu depuis la date T_0 jusqu'au temps 4. Cet exemple est décrit par les formules suivantes :

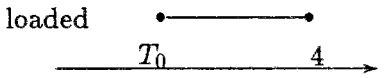
$$begin(T_0, loaded) \wedge T_0 < 2$$
$$occur(4, pulltrigger)$$
$$\forall\, t \;\; true(t, loaded) \wedge occur(t, pulltrigger) \Rightarrow end(t, loaded)$$

Soit $\mathcal{T}$ la théorie constituée de ces formules et de toutes celles qui définissent la logique. Une conséquence souhaitée de $\mathcal{T}$ est $end(4, loaded)$. Cependant, cette

formule n'est pas un théorème de $\mathcal{T}$: elle n'est vraie que dans certains modèles de la théorie. Parmi tous les modèles de $\mathcal{T}$, il y en a une première partie où *loaded* finit avant le temps 4 :



Cette fin intervient sans raison apparente par rapport au scénario mais la configuration correspondante est malgré tout un modèle de $\mathcal{T}$. Dans une autre partie des modèles, *loaded* finit exactement au temps 4 :



Il n'y a par contre aucun modèle où la période de vérité de *loaded* s'étend au dela de 4 : l'axiome (5) ne serait alors pas satisfait.

Cet exemple montre qu'il est impossible de démontrer $end(4, loaded)$ à l'aide d'un démonstrateur de théorèmes qui effectue une preuve par résolution, c'est à dire une preuve d'insatisfiabilité de $\mathcal{T} \cup \{\neg end(4, loaded)\}$, puisque cette nouvelle théorie admet malgré tout certains modèles.

2.2.3 Quelques solutions possibles

En ce qui concerne le premier problème, celui posé par les relations d'ordre et d'égalité, l'exemple donné précédemment montre que même un démonstrateur comme OTTER, réputé puissant, et qui implante la paramodulation pour gérer l'égalité, l'usage de démodulateurs et des prédicats « évaluables », n'est pas très efficace pour prouver des assertions sur l'ordre des instants. *A contrario*, l'on sait que de telles manipulations de relations d'ordre sont très simplement réalisables par des algorithmes spécialisés comme ceux qui ont été décrits dans le premier chapitre (paragraphe 1.2.1, page 26) de ce mémoire. Il est donc intéressant de pouvoir combiner ces algorithmes avec un mécanisme de résolution.

Plusieurs tentatives ont été faites pour incorporer dans un mécanisme de résolution l'usage d'une théorie particulière sans qu'il soit nécessaire d'expliciter, parmi les clauses du problème, les axiomes de la théorie considérée, ce qui contribue grandement à limiter la taille de l'espace de recherche. Le but est d'utiliser pour ces « sous-théories » des procédures spécialisées, plus efficaces que la résolution pour décider de la satisfaisabilité.

La *theory resolution* de (Stickel, 1985) est un de ces mécanismes. Il est basé sur la recherche de littéraux complémentaires dans la règle de résolution, tels que ces littéraux forment un ensemble inconsistant par rapport à la sous-théorie T considérée. L'analogie avec la principe de résolution classique se fait en considérant que lorsque l'on a deux littéraux complémentaires dans deux clauses distinctes, candidates à l'application de la règle de résolution, ces deux littéraux (ou leur instance

par l'unificateur) forment le cas le plus trivial d'inconsistance entre eux : A et $\neg A$. Dans le cas de la *theory-resolution*, il suffit de trouver n clauses :

$$C_1 = \{K_1\} \cup L_1$$
$$\ldots$$
$$C_n = \{K_n\} \cup L_n$$

telles qu'il existe un ensemble de littéraux $R_1, \ldots, R_m$ (éventuellement vide) tels que $K_1 \wedge \ldots \wedge K_n \wedge R_1 \wedge \ldots \wedge R_m$ est insatisfiable par rapport à la théorie T. Alors la clause

$$L_1 \cup \ldots \cup L_n \cup \{\neg R_1, \ldots, \neg R_m\}$$

est un T-résolvant des clauses C_i. En pratique, il est nécessaire (et suffisant) que l'ensemble de littéraux $K_1, \ldots, K_n, R_1, \ldots, R_m$ soit minimal. Les littéraux $K_1, \ldots, K_n$ sont appelés la *clé*, et $R_1, \ldots, R_m$ les *résidus*. La complétude de la *theory-resolution* est obtenue dès lors que l'identification des clés peut se faire de manière complète.

Compte tenu de la relative complexité de la théorie décrivant les relations d'ordre, il parait très difficile de construire un algorithme efficace qui retournerait l'ensemble de toutes clés possibles entre les littéraux en $<$, $\leq$ et $=$ d'un ensemble de clauses.

Par contre, la *theory resolution* présente un avantage pour implanter dans le mécanisme de résolution certains axiomes qui posent problème comme celui qui a été décrit précédemment sur les propriétés « liquides ». Supposons que cet axiome forme la théorie T : alors toute paire de littéraux

$$\{TRUE(t_1, t_2, p), \neg TRUE(t_3, p)\}$$

telle que $t_1 \leq t_3 \leq t_2$ est inconsistante par rapport à T. On peut donc appliquer la *theory resolution* et écrire ceci sous la forme d'une règle d'inférence spécifique :

$$\frac{\begin{array}{c} TRUE(t_1, t_2, p) \vee L_1 \\ \neg TRUE(t_3, p) \vee L_2 \end{array}}{L_1 \vee L_2 \vee \neg(t_1 \leq t_3) \vee \neg(t_3 \leq t_2)}$$

La généralisation de ce mécanisme n'est pas très simple : la taille et la complexité de la théorie T influent directement sur la complexité de l'algorithme chargé d'identifier les clés de la résolution et de générer les résidus correspondants. De plus, pour des raisons de complétude, il est nécessaire de considérer toutes les clés possibles.

La *theory resolution*, dont l'intérêt théorique n'est pas négligeable, ne semble donc pas facilement adaptable au raisonnement temporel. Dans la partie suivante de ce chapitre, nous étudions l'utilisation d'une deuxième extension de la résolution dans le cadre du raisonnement temporel. Ce deuxième principe, la résolution avec contraintes, a l'avantage d'être très facilement adaptable au raisonnement temporel avec des logiques temporelles réifiées. De plus, il fournit un cadre formel simple qui permet d'intégrer des mécanismes déductifs basés sur la résolution et des systèmes de gestion de contraintes tels qu'ils ont été étudiés dans le chapitre 1.

En ce qui concerne le deuxième problème, une solution est de sortir de l'optique de la démonstration de théorèmes, et d'envisager une approche basée sur l'abduction. En effet, la persistance, en particulier telle que nous l'avons formalisé précédemment dans notre logique temporelle, suppose souvent qu'une date inconnue soit placée aussi loin que possible sur l'axe du temps : ce qui correspond à une certaine forme de sémantique préférentielle qui cherche à maximiser les persistances. Si l'on accepte que certaines formes de relations temporelles puissent être supposées si nécessaire, alors il est possible d'utiliser la résolution pour générer ces hypothèses : une clause obtenue par résolution dans laquelle ne subsistent que des littéraux de cette forme peut servir de base à la génération de telles hypothèses.

2.3 La résolution avec contraintes : un cadre intégrateur

La *résolution avec contraintes*, dont l'étude générale a été effectuée par (Bürckert, 1991), est une solution au problème d'intégration entre des méthodes de déduction basées sur le principe de résolution et des procédures spécialisées – sous entendu plus efficaces que la résolution seule – pour déterminer la satisfiabilité de formules dont le langage est un sous-ensemble du langage complet pour lequel on souhaite construire une procédure de décision.

Le cadre de la résolution avec contraintes est celui d'une logique avec *quantificateurs restreints* (Bürckert, 1991), où les deux quantificateurs universel et existentiel sont associés à des formules qui interviennent comme des restrictions sur leur domaine de quantification. Ces formules de restriction, ou *contraintes*, proviennent d'une théorie $\mathcal{T}$, dite *théorie de restriction*, qu'il n'est pas forcément nécessaire de décrire par une théorie logique : il suffit de connaître l'ensemble de ses modèles, et de savoir déterminer la consistance dans ces modèles (appelée aussi $\mathcal{T}$-satisfiabilité par référence à $\mathcal{T}$).

Le langage de la théorie de restriction est donc un sous ensemble du langage complet initial. Plus précisément, les structures d'interprétation de $\mathcal{T}$ sont supposées définies par une structure qui comprend en particulier une signature Δ avec égalité (ensemble de symboles de fonctions et de symboles de prédicats, dont l'égalité, avec leur arité respective), et un ensemble de formules sur Δ considérées comme les restrictions admissibles (satisfaites par la théorie de restriction $\mathcal{T}$).

Pour interpréter les formules de la forme $\forall_{X:R}\ F$ et $\exists_{X:R}\ F$ où R est une formule de restriction et F une formule quelconque, les structures d'interprétation de $\mathcal{T}$ sont étendues avec les symboles de fonctions et de prédicats utilisés dans les formules F, conjointement avec leur mécanisme d'interprétation qui est analogue dans son principe à ce qu'on définit habituellement pour la logique du premier ordre (sémantique à la Tarski). Une telle structure est donc une extension conservatrice d'une structure d'interprétation de $\mathcal{T}$. Si $\mathcal{A}$ est une de ces structures, et α une affectation

de variables, on définit ainsi la satisfaction des formules quantifiées :

$$(\mathcal{A},\alpha) \models \forall_{X:R}\, F \quad \text{ssi} \quad \forall\, u_1,\ldots,u_n \in U^{\mathcal{A}} \text{ tels que } (\mathcal{A}_{/\Delta}, \alpha_{[x_1\leftarrow u_1,\ldots,x_n\leftarrow u_n]}) \models R, \text{ on a } (\mathcal{A}, \alpha_{[x_1\leftarrow u_1,\ldots,x_n\leftarrow u_n]}) \models F$$

$$(\mathcal{A},\alpha) \models \exists_{X:R}\, F \quad \text{ssi} \quad \exists\, u_1,\ldots,u_n \in U^{\mathcal{A}} \text{ tels que } (\mathcal{A}_{/\Delta}, \alpha_{[x_1\leftarrow u_1,\ldots,x_n\leftarrow u_n]}) \models R \text{ et } (\mathcal{A}, \alpha_{[x_1\leftarrow u_1,\ldots,x_n\leftarrow u_n]}) \models F$$

où $U^{\mathcal{A}}$ désigne l'univers d'interprétation de la structure $\mathcal{A}$.

Dans le cas qui nous intéresse, nous restreignons notre attention à la forme clausale des formules avec quantificateurs restreints. Ces clauses sont de la forme $C \parallel R$ où C est un ensemble de littéraux et R une formule de la théorie de restriction, que l'on appelle aussi *restriction* de la clause. Logiquement, la signification de cette expression est $\forall X\ R \Rightarrow C$ où X est le vecteur des variables de la clause. La sémantique de ces clauses est celle qui a été définie précédemment pour les formules $\forall_{X:R}\, C$. En pratique, il faut comprendre qu'une telle clause représente l'ensemble des instances de C par une affectation α qui satisfait R, ou, en d'autres termes, telle que α est une « solution » des contraintes R.

Etant données deux clauses avec contraintes possédant deux littéraux complémentaires, le principe de résolution avec contraintes est défini ainsi :

$$\frac{\{P(x_1,\cdots,x_n)\} \cup C \parallel R \qquad \{\neg P(y_1,\cdots,y_n)\} \cup D \parallel S}{C \cup D \parallel R \wedge S \wedge \Gamma} \; R \wedge S \wedge \Gamma \text{ est } \mathcal{T}\text{-satisfiable} \tag{2.1}$$

où Γ est la conjonction des n équations

$$x_1 = y_1, \ldots, x_n = y_n$$

Ce principe de résolution apparaît comme une généralisation du principe de résolution classique avec unification de Robinson, car si la théorie $\mathcal{T}$ est celle de l'égalité entre termes et que les contraintes admissibles sont des équations, l'algorithme d'unification classique permet de décider de la satisfiabilité d'un ensemble de telles contraintes.

En ce qui concerne l'usage du principe de résolution avec contraintes pour la démonstration automatique, l'approche est analogue à celle de la résolution classique. Dans le cas classique, pour montrer qu'un ensemble de clauses est insatisfiable, il faut dériver par résolution une clause vide. Dans le cas de la résolution avec contraintes, les résultats de (Bürckert, 1991) n'incitent pas à l'optimisme d'un premier abord. Le premier résultat est que si un ensemble de clauses avec contraintes est insatisfiable, alors pour tout modèle A de la théorie de restriction $\mathcal{T}$, il est possible de dériver par résolution avec contraintes une clause vide $\square \parallel R$ telle que $A \models \exists(R)$, et réciproquement. D'un point de vue général, il n'est absolument pas certain qu'il suffise de dériver un nombre fini de clauses vides pour décider de l'inconsistance de l'ensemble de clauses : tout dépend de la théorie $\mathcal{T}$ et de ses modèles. Un deuxième

résultat est que si la théorie $\mathcal{T}$ est axiomatisable en logique du premier ordre, alors il suffit de dériver un nombre fini de clauses vides

$$\square \parallel R_1, \ldots, \square \parallel R_n$$

telles que $\mathcal{T} \models \exists(R_1) \vee \ldots \vee \exists(R_n)$.

Un autre résultat, et le plus intéressant dans notre cas, est que si la théorie de restriction $\mathcal{T}$ a un modèle G qui est *générique*, c'est à dire que pour toute formule de restriction R on vérifie l'équivalence

$$G \models \exists(R) \equiv \mathcal{T} \models \exists(R)$$

alors il suffit de dériver une seule clause vide $\square \parallel R$ telle que $G \models \exists(R)$ pour prouver l'inconsistance de l'ensemble de clauses initiales. La généricité du modèle G comprend les cas où, soit G rend compte de tous les modèles de $\mathcal{T}$, soit $\mathcal{T}$ n'a qu'un modèle unique qu'on peut alors assimiler à G, et ceci pour toutes les formules de restriction envisageables.

Un dernier aspect important, qui fixera les limites de l'approche par la résolution avec contraintes pour le raisonnement temporel, est que la totalité des symboles de fonction du langage doit être prise en compte par la théorie de restriction. Ceci parce que, par construction, ces symboles de fonction apparaissent dans l'ensemble Γ d'équations produit par l'application de la règle de résolution avec contraintes.

2.3.1 Application au raisonnement temporel

L'application de la résolution au raisonnement temporel a bien évidemment pour but de pouvoir utiliser des systèmes de gestion de contraintes temporelles. Pour une formule clausale de la forme

$$L_1 \vee \ldots \vee L_n \vee C_1 \vee \ldots \vee C_m$$

où $C_1, \ldots C_m$ sont des contraintes entre entités temporelles (instants ou intervalles) et $L_1 \ldots L_n$ sont des littéraux, on obtient sa forme contrainte comme

$$L_1 \vee \ldots \vee L_n \parallel \neg C_1 \wedge \ldots \wedge \neg C_m$$

par simple transformation de $\vee$ en $\Rightarrow$ et en écrivant ensuite celle-ci comme une clause avec contraintes.

La justification de cette transformation se fait par analogie entre la logique à quantificateurs restreints définie par (Bürckert, 1991) et une logique temporelle réifiée telle qu'elle est définie par (Shoham, 1987). Cette analogie porte sur la manière dont est définie la sémantique des formules. Tout comme les structures d'interprétations de la théorie de restrictions sont étendues au langage complet, on peut faire la même remarque sur les interprétations $(W, \leq, M)$ des logiques temporelles: un premier élément (W) de la structure d'interprétation définit le support du temps

(nombres entiers ou réels pour des instants, sous-ensembles de ces mêmes nombres pour des intervalles) ainsi que les relations associées (relation d'ordre $\leq$ entre instants, ou relation d'Allen entre intervalles), puis un autre « composant » (M) définit l'interprétation des termes temporels (à image dans W) et atemporels et permet d'associer entités temporelles de W et interprétations des termes atemporels. L'insatisfiabilité d'un ensemble de clauses d'une logique temporelle réifiée est alors équivalente à l'insatisfiabilité de l'ensemble de clauses avec contraintes dans la logique avec quantificateurs restreints, après application de la transformation précédente.

Suivant cette remarque, l'application de la résolution avec contraintes pour le raisonnement temporel se fait d'après les principes suivants :

- le langage complet traité par la résolution avec contraintes est une logique temporelle réifiée sur le modèle de (Shoham, 1987) ;
- la théorie de restriction est celle des relations temporelles entre, soit les instants, soit les intervalles, suivant la logique temporelle considérée ;
- les restrictions envisageables doivent être des conjonctions de contraintes atomiques.

Ceci donne donc le moyen d'intégrer simplement dans un mécanisme par résolution les nombreuses procédures et algorithmes développés pour décider de la consistance d'un ensemble (conjonction) de relations temporelles. Dans ce cas, une procédure de décision correcte et complète joue le rôle d'un modèle générique de la théorie de restriction pour toutes les formules de restriction, tant que celles-ci sont des conjonctions de contraintes.

Cette utilisation de la résolution avec contraintes pour le raisonnement temporel impose que l'algorithme de décision pour la consistance des relations temporelles ait les deux propriétés suivantes :

- l'algorithme doit être complet. Ceci limite le choix de la théorie de restriction, et donc l'expressivité de la logique temporelle, à l'algèbre d'intervalles restreinte, ou à l'algèbre d'instants ;
- il doit être incrémental, c'est à dire qu'il doit être possible d'ajouter une par une de nouvelles contraintes, et que le coût de cet ajout soit (largement) inférieur au coût d'un algorithme qui traiterait l'ensemble des contraintes en une fois sans tenir compte de la consistance déja déterminée lors des pas de résolution antérieurs.

2.3.2 Limites de l'approche

Dans ce qui suit, nous nous plaçons arbitrairement dans le cadre des contraintes entre instants. Comme les remarques que nous allons faire sont essentiellement liées

aux formes syntaxiques utilisées pour dénoter ces instants, il n'y a pas de perte de généralité par rapport à une logique basée sur des intervalles.

Les limites de l'approche par la résolution avec contraintes reposent principalement sur le langage des expressions temporelles, des contraintes entre ces expressions et donc sur la disponibilité d'algorithmes complets et incrémentaux pour décider de la consistance des contraintes. A première vue, dès lors que l'algorithme est complet, il n'y a pas de problème. Cependant, cette vision simpliste doit être amendée par des considérations sur la forme des termes temporels sur lesquels portent les contraintes.

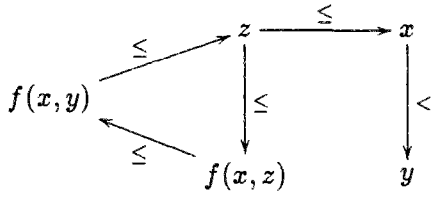
Le premier problème est celui de l'occurrence de symboles fonctionnels quelconques dans les termes temporels désignant des instants, les arguments de ces fonctions étant toujours des termes temporels: nous appellerons de tels termes *purement temporels*, par opposition aux termes temporels où certains arguments de fonction sont des termes atemporels, et que nous appellerons *mixtes*[3]. De tels symboles fonctionnels obligent à étendre la théorie de restriction $\mathcal{T}$ de telle manière que l'équivalence suivante en soit un théorème:

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \Leftrightarrow (s_1 = t_1) \wedge \ldots \wedge (s_n = t_n) \qquad (2.2)$$

et ceci pour tout symbole fonctionnel f d'arité n utilisé dans les contraintes. Ce qui signifie que l'algorithme utilisé pour décider de la consistance tienne compte de cet axiome, comme c'est le cas pour l'algorithme d'unification classique entre termes.

Exemple 2.1 *Soit l'ensemble de contraintes suivant:*

$$\begin{array}{l} f(x,y) \leq z \\ z \leq x \\ z \leq f(x,z) \\ x < y \\ f(x,z) \leq f(x,y) \end{array}$$



Le graphe de contraintes correspondant est consistant, et pourtant cet ensemble de contraintes est inconsistant, puisqu'il implique en particulier l'égalité $f(x,z) = f(x,y)$*, et donc* $y = z$*, ce qui est contradictoire avec la contrainte, obtenue par transitivité,* $z < y$*.*

Un algorithme qui permet de décider de la consistance d'un ensemble d'équations dans ce cadre est présenté dans (Caprotti, 1993). Il ne traite que des ensembles d'équations, et les termes fonctionnels non interprétés ne peuvent pas apparaître dans les termes arithmétiques, c'est à dire que $f(x+1, y)$ est admissible alors que $f(x+1) - y$ ne l'est pas. De plus un tel algorithme est limité aux systèmes d'équations: l'exemple précédent n'est pas traité par ce système. Néanmoins, l'on peut envisager l'usage de cet algorithme avec un propagateur de contraintes temporelles

3. de tels termes temporels apparaissent par exemple suite à la skolémization de variables existentielles, lors de la conversion en clauses de l'énoncé du problème.

qui calcule la relation minimale entre deux points, et qui donc identifiera au cours de la propagation toutes les égalités, données et induites. Un tel algorithme sera décrit dans la deuxième partie de ce chapitre.

Un deuxième facteur limitant cette approche concerne les termes atemporels, et donc les termes temporels mixtes. Lors de l'application de la règle de résolution, il est possible que l'ensemble Γ contienne des égalités entre des termes atemporels. De la même manière que précédemment, il faut disposer d'un algorithme permettant de décider de la consistance de telles égalités. Rappelons que les termes atemporels sont *a priori* construits à l'aide de variables, de symboles de constantes et de symboles fonctionnels tout comme des termes en logique du premier ordre. La différence avec la logique du premier ordre est que l'interprétation des symboles fonctionnels est *a priori* dépendante du temps: on qualifiera de « flexibles » ces termes, le terme « rigide » étant alors réservé aux termes dont l'interprétation est indépendante du temps.

Pour les termes rigides, la consistance d'une égalité peut être déterminée par rapport à l'univers de Herbrand des termes considérés: dans ce cas, un algorithme d'unification classique est parfaitement adapté.

Plus difficile est le cas des termes flexibles: une égalité entre deux de ces termes a des répercussions sur les termes temporels qui interviennent dans leur interprétation. Soit par exemple une modélisation (partielle) du monde des blocs où A, B et C désignent trois blocs; R un robot; le terme $\mathrm{ON}(u,v)$ que le bloc u est sur le bloc v; et $\mathrm{IN\_HAND}(r)$ désigne le bloc présent dans la pince du robot r: ce terme est typiquement un terme flexible. Si les deux clauses entre lesquelles on applique la résolution avec contraintes sont les suivantes:

$$\begin{aligned}&\mathrm{TRUE}(t_1,t_2,\mathrm{ON}(A,B))\vee\ldots\\&\neg\mathrm{TRUE}(t_3,t_4,\mathrm{ON}(\mathrm{IN\_HAND}(R),v))\vee\ldots\end{aligned}$$

alors il faut déterminer la consistance d'un ensemble de contraintes comprenant entre autres les égalités suivantes:

$$\begin{aligned}&t_1=t_3 \qquad t_2=t_4\\&\mathrm{ON}(A,B)=\mathrm{ON}(\mathrm{IN\_HAND}(R),v)\end{aligned}$$

et dont la dernière équation se décompose en:

$$\begin{aligned}&A=\mathrm{IN\_HAND}(R)\\&B=v\end{aligned}$$

Si l'on sait qu'entre les dates 3 et 5 le robot tient dans sa pince le bloc B, la première équation est insatisfaisable pour des valeurs de t_3 et t_4 telles que $3\leq t_3\leq t_4\leq 5$. Cependant, de telles informations sur l'univers ne sont en général pas disponibles à un coût faible, puisque elles peuvent faire l'objet d'un raisonnement complexe, et l'équation concernée est *a priori* indécidable.

Néanmoins, dans les cas des termes flexibles, la décomposition des équations entre deux termes de même symbole fonctionnel est toujours possible et ne contredit pas

l'interprétation non habituelle (par rapport à la logique du premier ordre) de ces termes. En effet, si l'on s'en tient au schéma de (Shoham, 1987), cette interprétation est fonction des arguments temporels des prédicats formant la clé de la résolution. Comme ces arguments temporels sont supposés égaux entre eux, par exemple t_1 et t_3, et t_2 et t_4 dans l'exemple précédent, et font donc référence à la même période de temps, l'interprétation du symbole fonctionnel est identique pour les deux termes, et l'égalité peut être résolue par décomposition. Ainsi, si les deux littéraux de la clé sont

$$\text{TRUE}(t_1, T_2, \text{ON}(\text{IN\_HAND}(x), B)) \quad \text{et} \quad \neg\text{TRUE}(2, t_4, \text{ON}(\text{IN\_HAND}(R), v))$$

où t_1, t_4 sont des variables temporelles, et x et v des variables atemporelles, alors, une solution du système d'équations

$$\begin{aligned} &t_1 = 2 \qquad T_2 = t_4 \\ &\text{ON}(\text{IN\_HAND}(x), B) = \text{ON}(\text{IN\_HAND}(R), v) \end{aligned}$$

peut être représentée par la substitution

$$\{t_1 \mapsto 2, t_4 \mapsto T_2, x \mapsto R, v \mapsto B\}$$

puisque les deux termes fonctionnels $\text{IN\_HAND}(x)$ et $\text{IN\_HAND}(R)$ sont interprétés sur la même période de temps $(2, T_2)$.

Le problème ne se pose que pour les équations non décomposables ainsi, dont les membres sont des fonctions différentes et flexibles. Dans ce cas, une solution éventuelle serait de passer par la génération d'hypothèses supplémentaires: dans l'exemple précédent, une hypothèse adéquate, dont le rôle est analogue à celui d'un *résidu* dans la *theory-resolution*, est $\text{TRUE}(t_1, t_2, \text{IN\_HAND}(R) = A)$, dont la négation serait ajoutée à la clause résolvante. Encore faut il que le langage, c'est à dire la logique temporelle, accepte des termes atemporels contenant une égalité comme $\text{IN\_HAND}(R) = A$: ce qui suppose probablement que la logique, et l'implantation qui en est faite, accepte de gérer des ontologies variables en fonction du temps, c'est à dire que des individus existent ou pas à certaines dates. La logique réifiée présentée dans (Reichgelt, 1989) est un exemple d'un tel formalisme.

Dans le cas de ce que nous avons appelé les termes temporels mixtes, le problème est une combinaison des deux précédents. Fatalement, la décomposition d'une équation entre deux termes temporels mixtes produira une équation entre deux termes atemporels. Décider de la consistance de cette équation peut donc devenir un problème difficile.

2.3.3 L'exemple revisité

Nous reprenons ici l'exemple du paragraphe 2.2.2 dans l'optique de la résolution avec contraintes. La mise sous forme clausale des axiomes de la logique, puis l'application de la transformation en clauses avec contraintes décrite au paragraphe 2.3.1,

$$\neg hold(t_1,t_2,p) \vee begin(t_1,p) \parallel \top$$
$$\neg hold(t_1,t_2,p) \vee end(t_2,p) \parallel \top$$
$$\neg hold(t_1,t_2,p) \vee true(t_3,p) \parallel t_1 < t_3 \leq t_2$$
$$\neg hold(t_1,t_2,p) \vee \neg begin(t_3,p) \parallel t_1 < t_3 \leq t_2$$
$$\neg hold(t_1,t_2,p) \vee \neg end(t_3,p) \parallel t_1 \leq t_3 < t_2$$
$$\neg begin(t_1,p) \parallel f_1(t_1,p) \leq t_1$$
$$\neg begin(t_1,p) \vee hold(t_1,e(t_1,p),p) \parallel \top$$

FIG. 2.2 - *Clauses contraintes produites par la mise sous forme clausale des axiomes de la logique temporelle.*

produit les clauses de la figure 2.2. Le signe $\top$ dans la partie contraintes indique l'absence de contrainte, c'est à dire une contrainte toujours vraie.

La même transformation sur les clauses de l'exemple donne les clauses contraintes suivantes :

$$begin(T_0,loaded) \parallel \top$$
$$\square \parallel \neg(T_0 < 2)$$
$$occur(4,pulltrigger) \parallel \top$$
$$\neg true(t,loaded) \vee \neg occur(t,pulltrigger) \vee end(t,loaded) \parallel \top$$

La clause $\square \parallel \neg(T_0 < 2)$ dans cet exemple est une clause vide : pour préserver la consistance initiale de la théorie, il faut imposer à la théorie de restriction $\mathcal{T}$ de ne pas satisfaire la restriction de cette clause, donc que $T_0 < 2$ soit vrai. Cette méthode de rétablissement de la cohérence a été décrite par (Panitz, 1993). De la même manière, entre les deux clauses

$$\neg begin(t_1,p) \parallel f_1(t_1,p) \leq t_1 \qquad \text{et} \qquad begin(T_0,loaded) \parallel \top$$

on obtient la clause vide

$$\square \parallel f_1(T_0,loaded) \leq T_0$$

dont on déduit que, pour rétablir la consistance de la théorie, il faut que $\mathcal{T}$ vérifie $T_0 < f_1(T_0,loaded)$.

La figure 2.3 montre une preuve « conditionnelle » de $end(4,loaded)$. L'appellation conditionnelle se justifie car la restriction de la clause vide 23 n'est pas satisfaite par $\mathcal{T}$: les seules informations disponibles sont que T_0 est avant 2, et $T_0 < f_1(T_0,loaded)$. Cette réponse fournit une indication sur une hypothèse à faire pour que $end(4,loaded)$ soit une conséquence du scénario décrit. Supposer $4 \leq f_1(T_0,loaded)$ est équivalent à faire l'hypothèse que *loaded* a persisté au moins jusqu'au coup de feu. Malheureusement, seule la solution $f_1(T_0,loaded) = 4$ est acceptable et correspond à un modèle de la théorie : le cas $4 < f_1(T_0,loaded)$ ne satisfait pas l'axiome (5) de la logique temporelle. Cette limite est dûe au fait que le terme $f_1(T_0,loaded)$ est un terme fonctionnel, qui fait en outre référence au terme

$1: \neg end(4, loaded)$
$2: \neg true(t_1, loaded) \vee \neg occur(t_1, pulltrigger) \vee end(t, loaded)$
$3: occur(4, pulltrigger)$
$4: \neg hold(t_2, t_3, p) \vee true(t_4, loaded) \parallel (t_2 \leq t_4) \wedge (t_4 \leq t_3)$
$5: \neg begin(t_5, p) \vee hold(t_5, f_1(t_5, p), p)$
$6: begin(T_0, loaded)$
$20: \neg true(4, loaded)$ [1, 2, 3]
$21: \neg hold(t_2, t_3, loaded) \parallel (t_2 \leq 4) \wedge (4 \leq t_3)$ [20, 4]
$22: \neg begin(t_5, loaded) \parallel (t_5 \leq 4) \wedge (4 \leq f_1(t_5, loaded))$ [21, 5]
$23: \Box \parallel (T_0 \leq 4) \wedge (4 \leq f_1(T_0, loaded))$ [22, 6]

FIG. 2.3 - *Une preuve conditionnelle de $\neg end(4, loaded)$. Pour alléger l'écriture des contraintes, celles-ci sont simplifiées et les égalités qui s'y trouvent sont utilisées pour instancier les variables des clauses lorsque cela est possible.*

atemporel *loaded*, et que tester la consistance de cette hypothèse dépasse le cadre des algorithmes utilisés pour implanter la théorie de restriction $\mathcal{T}$.

De manière générale, l'intérêt de la résolution avec contraintes est de supprimer toute la partie de l'espace de recherche concernant des clauses uniquement formées de littéraux décrivant des relations d'ordre (comme les clauses 23 à 27 sur la preuve du paragraphe 2.2.3). Dans le cas particulier de notre exemple. la résolution avec contraintes ne résoud pas directement le problème de l'inadéquation de la démonstration de *théorèmes* lorsqu'il est nécessaire de manipuler la persistance temporelle. Une solution possible serait d'interpréter abductivement les contraintes associées avec une clause vide : la résolution avec contraintes présente simplement l'avantage de générer plus efficacement ces clauses vides. Néanmoins on ne dispose pas d'algorithme permettant de tester la consistance d'une telle hypothèse.

2.3.4 Pour récapituler...

- La résolution avec contraintes apporte l'efficacité « à un bas niveau » des inférences par résolution, en héritant ainsi des capacités des systèmes dédiés aux langages de relations temporelles.

- La résolution avec contraintes apporte un cadre formel pour l'utilisation de systèmes de contraintes temporelles dans des mécanismes d'inférence par résolution. Ce cadre permet aussi de fixer des exigences minimales pour les systèmes de contraintes temporelles, à savoir la complétude et l'incrémentalité.

- La résolution avec contraintes n'apporte pas de solution aux problèmes de la persistance et de l'implantation d'une procédure de preuve qui matérialiserait une sémantique préférentielle pour la logique temporelle utilisée. De la même

manière, le problème de la décidabilité d'une équations entre termes flexibles (à interprétation dépendante du temps) reste non résolu.

Ces constatations permettent de situer précisément l'intérêt de la résolution avec contraintes. Si l'apport ne parait pas franchement déterminant, il n'est pas interdit d'envisager de développer plus avant certains aspects de son utilisation en raisonnement temporel. C'est l'objet de la partie suivante, où nous allons plus particulièrement étudier des algorithmes de tests de la consistance pour les restrictions de clauses avec contraintes.

2.4 Mise en œuvre

Le but de cette partie est d'étudier de manière plus précise la mise en œuvre de la résolution avec contraintes pour le raisonnement temporel. Le point principal concerne le test de consistance des contraintes, pour lequel nous étudions la possibilité d'étendre des algorithmes connus, afin de repousser, si possible, les limites présentées au paragraphe 2.3.2. Nous présentons quelques considérations générales, puis un exemple concret de mise en œuvre dans le formalisme de contraintes metriques de (Dechter et al., 1991).

2.4.1 Algorithmes de test de la consistance

Dans ce paragraphe, nous allons étudier la mise en œuvre de la résolution avec contraintes en raisonnement temporel, du point de vue des algorithmes permettant de décider de la consistance des restrictions des clauses. Dans tout ce paragraphe, nous nous limitons aux contraintes entre instants, notre propos pouvant être alors étendu aisément au cas de l'algèbre d'intervalles restreinte. Le cas de l'algèbre d'intervalles complète est peu intéressant dans notre cas puisque nous désirons des algorithmes complets et efficaces (polynomiaux) pour tester la consistance d'un ensemble de relations.

Dans la paragraphe 2.3.2 (page 55), nous avons identifié deux sources de difficultés :

- la présence dans les expressions dénotant des instants de symboles fonctionnels que l'on qualifiera de « non interprétés », c'est à dire pour lesquels il n'y a pas d'information accessible sur leur signification, contrairement au fonctions arithmétiques comme + et − par exemple. Ce sont par exemple des fonctions de skolem ;

- la présence dans les termes atemporels de symboles de fonction flexibles, dont l'interprétation varie en fonction du temps.

Un dernier cas difficile résulte de la combinaison des deux précédents : lorsque les termes temporels comprennent des symboles fonctionnels quelconques, dont certains arguments sont des termes atemporels qui peuvent être *a priori* flexibles. Ce cas n'est pas étudié en lui-même : sa solution est aussi une combinaison des solutions des deux premiers cas.

Pour le premier cas, nous n'assimilons pas les symboles fonctionnels arithmétiques, comme + ou −, avec des symboles fonctionnels quelconques. En effet, en ce qui concerne les premiers, il existe plusieurs formalismes de contraintes quantitatives qui permettent de prendre en compte ces expressions arithmétiques, comme le formalisme de (Dechter et al., 1991), où encore les adaptations de l'algorithme du simplexe et de la méthode d'élimination de Gauss (Stuckey, 1991b) comme dans le langage de programmation logique avec contraintes CLP($\mathcal{R}$)(Heintze, Jaffar, Michaylov, Stuckey, & Yap, 1992). Par contre, les symboles fonctionnels quelconques n'ont pas de propriété particulière : la seule relation que la théorie de restriction $\mathcal{T}$ doit satisfaire est :

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \quad \Leftrightarrow \quad (s_1 = t_1) \wedge \ldots \wedge (s_n = t_n)$$

pour tout symbole f d'arité n. Dans ce qui suit, non ne distinguerons donc pas si les contraintes sont purement symboliques ou quantitatives, puisque les algorithmes utilisent tous des techniques de propagation de contraintes.

Il y a deux grandes familles d'algorithmes pour déterminer la consistance d'un ensemble de relations : des algorithmes comme PC-2 qui réalisent la fermeture transitive des relations et qui calculent aussi la relation minimale lorsque les relations excluent $\neq$, et des algorithmes basés sur d'autres critères de consistance et qui ne construisent pas de graphe complet étiqueté par la relation minimale. Cette identification de la relation minimale est importante lorsque celle-ci est une égalité et lie deux termes temporels fonctionnels ayant le même symbole de fonction f. En effet dans ce cas, pour satisfaire l'équivalence 2.2, il faut décomposer une telle égalité en un ensemble d'égalités entre les termes arguments, ces nouvelles égalités étant alors ajoutées aux contraintes déjà testées.

Pour les algorithmes de type PC-2, cette extension est relativement aisée : lors de la relaxation d'une contrainte, si celle-ci devient une égalité et que la règle de décomposition s'applique, les nouvelles contraintes d'égalité sont ajoutées au réseau. Pour un algorithme comme celui de VanBeek (1989), ou de (Gerevini & Schubert, 1993), l'identification des égalités se fait après propagation, en identifiant tous les termes fonctionnels susceptibles d'être concernés et en interrogeant le réseau de contraintes pour déterminer la relation minimale : si celle-ci est une égalité, on ajoute au réseau les nouvelles contraintes d'égalité entre les termes arguments.

Dans le cas des contraintes quantitatives entre instants, c'est à dire lorsqu'apparaissent des termes arithmétiques, un formalisme de représentation applicable est celui de (Dechter et al., 1991). Dans ce formalisme, un algorithme de test de consistance est PC-2, ce qui autorise à utiliser la méthode décrite plus haut.

2.4.2 Un exemple concret de mise en œuvre

Nous allons maintenant étudier dans un cas concret la mise en œuvre de ces principes. Le cadre dans lequel nous nous plaçons est celui de contraintes d'égalité et d'inégalité entre instants, ces instants pouvant être représentés

- soit par des variables ;
- soit par des constantes numériques, ou des symboles dénotant eux-mêmes des constantes (disjoints de l'ensemble des variables) ;
- soit par des termes fonctionnels $f(t_1, \ldots, t_n)$ dont les arguments sont eux-mêmes des instants ;
- soit par des expressions arithmétiques de la forme $t + n$ ou $t - n$ où t est un instant et n un nombre.

Les contraintes sont construites à l'aide des relations $<$, $\leq$ et $=$. Un ensemble de telles contraintes peut être transformé en une représentation équivalente dans le formalisme de (Dechter et al., 1991). Ce formalisme est celui de contraintes entre instants, et dont la forme générale des contraintes est $x : [a, b] : y$ et signifie $a \leq y - x \leq b$.

Un ensemble de ces contraintes peut se représenter dans un graphe orienté dont les nœuds sont les variables et dont les arcs sont étiquetés par l'intervalle de la contrainte. Nous noterons un tel graphe par $G = (V, E)$ où V est l'ensemble des nœuds du graphe et E l'ensemble des arcs. La contrainte portée par l'arc (i, j) est notée T_{ij} et est donc un intervalle de la forme $[a, b]$: en pratique, il n'est pas nécessaire que l'arc inverse (j, i) soit présent dans E puisque l'on peut déterminer immédiatement T_{ji} comme étant l'intervalle $[-b, -a]$.

La traduction des contraintes entre instants dans ce formalisme se fait ainsi :

- les symboles de relation sont $<$, $=$ ou $\leq$, respectivement traduits par les intervalles $]0, +\infty[$, $[0, 0]$ et $[0, +\infty[$;
- les instants, c'est à dire les variables, sont soit des termes « atomiques », soit de la forme $t+n$ ou $t-n$ où t est une variable et n est un nombre. Les contraintes entre éléments de cette dernière forme sont transformées à l'aide des règles suivantes :

$$\begin{aligned}
(t + n) : [a, b] : t' &\rightarrow t : [a + n, b + n] : t' \\
(t - n) : [a, b] : t' &\rightarrow t : [a - n, b - n] : t' \\
t : [a, b] : (t' + n) &\rightarrow t : [a - n, b - n] : t' \\
t : [a, b] : (t' - n) &\rightarrow t : [a + n, b + n] : t'
\end{aligned}$$

Les contraintes négatives sont préalablement transformées ainsi :

$$\begin{aligned}
\neg(x < y) &\rightarrow y \leq x \\
\neg(x \leq y) &\rightarrow y < x
\end{aligned}$$

et les contraintes $x \neq y$ sont prises en compte en sachant que la propriété d'indépendance des contraintes négatives (Lassez & McAloon, 1989, 1992) est vérifiée pour le domaine qui nous concerne. Pour un domaine $\mathcal{D}$ et un ensemble de contraintes positives $c, c_1, \ldots, c_n$, cette propriété est définie par l'equivalence

$$\mathcal{D} \models \exists(c \wedge \neg c_1 \wedge \ldots \wedge \neg c_n) \quad \Leftrightarrow \quad \forall i \in [1,n] \;\; \mathcal{D} \models \exists(c \wedge \neg c_i)$$

c'est à dire que l'inconsistance ne provient jamais de la conjonction entre elles de deux contraintes négatives. Cette propriété est vérifiée pour le domaine des équations et inéquations linéaires sur les réels $\Re$, dans le cas où les contraintes négatives sont des négations d'égalité. L'algèbre d'instants et les contraintes du formalisme de (Dechter et al., 1991) sont des cas triviaux du domaine des équations et inéquations linéaires sur $\Re$, et cette propriété y est donc vérifiée. Ceci donne un moyen simple de traiter les contraintes c_i dont la relation est $\neq$: il suffit de tester séparément la condition $\mathcal{D} \models \exists(c \wedge \neg c_i)$ pour chaque c_i. Cette dernière condition se réécrit ainsi :

$$\begin{aligned} \mathcal{D} \models \exists(c \wedge \neg c_i) \quad &\Leftrightarrow \quad \mathcal{D} \models \neg\forall\neg(c \wedge \neg c_i) \\ &\Leftrightarrow \quad \mathcal{D} \not\models \forall(c \Rightarrow c_i) \end{aligned}$$

ce qui ramène le test précédent à vérifier une implication, ou encore, en termes de contraintes, à déterminer la relation minimale induite par c. L'utilisation de cette propriété n'est bien évidemment valable que pour déterminer la consistance : pour calculer la relation minimale, il faut reprendre en compte explicitement les contraintes négatives.

Un premier algorithme de test de la consistance

L'algorithme que nous allons présenter maintenant suit l'esprit de celui présenté dans (Caprotti, 1993). Il s'agit du couplage entre un algorithme de propagation de contraintes et un algorithme d'unification modifié.

On suppose que l'on dispose des deux clauses avec contraintes suivantes :

$$\{P(x_1, \cdots, x_n)\} \cup C \parallel R \qquad \text{et} \qquad \{\neg P(y_1, \cdots, y_n)\} \cup D \parallel S$$

entre lesquelles on souhaite appliquer le principe de résolution avec contraintes. L'ensemble Γ d'équations est défini comme l'ensemble des équations parmi

$$\{x_1 = y_1, \ldots, x_n = y_n\}$$

dont les deux membres sont des termes temporels. Le but de l'algorithme présenté ici est de déterminer la satisfiabilité de la partie contraintes (la restriction) de la clause résolvante, ou de rapporter une inconsistance.

On suppose donc que l'on dispose d'un algorithme de propagation de contraintes temporelles de type PC2 par exemple, adapté aux contraintes de (Dechter et al., 1991), et implanté de manière incrémentale : c'est à dire que cet algorithme prend

Ct-solve(C, C') :
  soit $Q \leftarrow \bigcup_{r_{i,j} \in C}$ Related-paths(i,j) ; $E \leftarrow \emptyset$
  $C' \leftarrow C' \cup C$
  tant que $Q \neq \emptyset$
    soit $(i,j,k) \in Q$; $Q \leftarrow Q - \{(i,j,k)\}$
    soit $r \leftarrow (r_{i,j} \circ r_{j,k}) \wedge r_{i,k}$
    si $r = \epsilon$ **alors retourner** $(\langle \mathbf{False}, \emptyset \rangle)$
    sinon si $r \subset r_{i,k}$ **alors**
      $Q \leftarrow Q \cup$ Related-paths(i,k)
      si $r = [0,0]$ **alors** $E \leftarrow E \cup \{i = k\}$
    $r_{i,k} \leftarrow r$
  fin
  retourner $(\langle C', E \rangle)$

Algorithme 2.1: *L'algorithme PC2 modifié pour identifier les égalités en cours de propagation. L'argument C est un ensemble de contraintes, et C' est un réseau de contraintes minimales et consistant : $r_{i,j}$ désigne la contrainte portée par l'arc (i,j) dans ce même réseau. La structure C' est supposée modifiée par les affectations à une relation $r_{i,j}$. La fonction* Related-paths *appliquée à un couple (i,j) de nœuds du réseau C' retourne l'ensemble des triplets (i,k,j) où k varie sur tous les nœuds du réseau et est différent de i et de j.*

en entrée deux ensembles de contraintes C et C', tels que C' est consistant, et indique si $C \wedge C'$ est consistant. Cet algorithme est modifié pour retourner une paire $\langle C'', E \rangle$, où C'' est une forme simplifiée de $C \wedge C'$, typiquement un réseau de contraintes minimales, et E est l'ensemble des égalités apparues au cours de la propagation (et qui ne sont donc présentes ni dans C, ni dans C'). Cet algorithme retourne $\langle \mathbf{False}, \emptyset \rangle$ si $C \wedge C'$ est inconsistant. Notons cette opération par

$$\langle C'', E \rangle \leftarrow \text{Ct-solve}(C, C')$$

où C est un ensemble de contraintes, et C' une forme simplifiée d'un ensemble de contraintes consistant, par exemple obtenu précédemment par le même algorithme. L'algorithme 2.1 implante la procédure Ct-solve. Il est possible d'optimiser l'aspect incrémental de cet algorithme en suivant la démarche de (Loganantharaj, Mitra, & Giambrone, 1994).

L'algorithme Ct-solve est caractérisé par les deux propriétés suivantes :

1. Ct-solve(C, C') retourne **False** si et seulement si $C \wedge C'$ est inconsistant. Ceci découle de ce que PC-2 est correct et complet pour décider de la consistance dans le formalisme des STP de Dechter ;

2. si Ct-solve(C, C') retourne $\langle C'', E \rangle$ et que $s = t$ est une égalité telle que $(s =$

Consistance-1(Γ, R, S) :
  soit $\langle C, E\rangle \leftarrow$ Ct-merge(R, S)
  si $(C =$ **False**$)$ **alors retourner** **(False)**
  soit $E' \leftarrow$ Simplify-1$(E \cup \Gamma)$
  $\langle C, E\rangle \leftarrow$ Ct-solve(E', C)
  tant que $(C \neq$ **False**$)$ et $(E \neq \emptyset)$ **faire**
    $E' \leftarrow$ Simplify-1$(E) - E$
    $\langle C, E\rangle \leftarrow$ Ct-solve(E', C)
  fin
  retourner (C)

Algorithme 2.2: *Algorithme de test de la consistance de* $\Gamma \wedge R \wedge S$. *L'argument* Γ *est un ensemble d'égalités, et* R *et* S *sont des ensembles de contraintes sous la forme de réseaux de contraintes minimales.*

$t) \in C''$, alors on vérifie l'équivalence suivante :

$$[(s = t) \notin C \text{ et } (s = t) \notin C'] \Leftrightarrow (s = t) \in E$$

où, dans le cas d'un réseau de contraintes C', la notation $(s = t) \in C'$ signifie qu'il existe dans C' un arc de s à t étiqueté par une égalité (intervalle $[0, 0]$ dane les cas des STP).

Cette dernière équivalence vient de ce que, dans l'algorithme Ct-solve, l'ensemble E ne contient une égalité que si celle-ci est obtenue par propagation et qu'elle est « nouvelle », au sens où la propagation puis l'intersection aboutit à restreindre strictement l'intervalle déja existant dans le réseau. Ceci permet d'éliminer les égalités qui seraient initialement présentes dans C ou dans C'.

On suppose aussi que l'on dispose d'un algorithme qui réalise la fusion de deux réseaux de contraintes, en fusionnant les nœuds étiquetés par le même terme, et qui applique ensuite Ct-solve pour obtenir des égalités et une forme simplifiée et minimale. Cet algorithme retourne éventuellement **False** si une inconsistance est détectée. Notons $\langle C, E\rangle \leftarrow$ Ct-merge(R, S) cette opération.

La partie unification est très simple : il suffit juste d'appliquer une règle de décomposition des termes fonctionnels, et de simplifier les équations triviales. Etant donné un ensemble d'égalités E, on note Simplify-1(E) l'opération qui consiste à appliquer les deux règles suivantes sur E tant que cela est possible, et qui retourne l'ensemble d'égalités ainsi modifié :

$$\begin{aligned} \{x = x\} \cup E &\rightarrow E \\ \{f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)\} \cup E &\rightarrow \{s_1 = t_1, \cdots, s_n = t_n\} \cup E \end{aligned}$$

L'algorithme 2.2 réalise le test de consistance d'un ensemble de contraintes. Il s'agit d'un couplage, d'une manière analogue à celle décrite dans (Caprotti, 1993),

du propagateur de contraintes (Ct-solve) et de l'algorithme d'unification simplifié (Simplify-1). Cet algorithme prend en argument les restrictions R et S des clauses parentes, sous forme de réseau de contraintes, ainsi que l'ensemble d'égalités Γ et retourne **False** si $R \wedge S \wedge \Gamma$ est inconsistant, ou sinon une forme simplifiée et minimale de la restriction de la clause résolvante. Cet algorithme utilise le propagateur modifié pour identifier toutes les égalités susceptibles d'un traitement par Simplify-1 : les nouvelles égalités éventuellement crées ainsi sont rajoutés au réseau de contraintes.

Si $\mathcal{O}$ désigne la théorie des relations quantitatives dont PC2 est une procédure de décision, et $\mathcal{E}$ la théorie de l'égalité définie par les axiomes de la forme $f(x) = f(y) \Leftrightarrow (x = y)$, alors à chaque itération, le réseau C est toujours une conséquence de $\mathcal{O}, \mathcal{E}, R \wedge S \wedge \Gamma$.

Etant donnés les ensembles Γ, R et S, qui sont finis, l'ensemble des contraintes entre les termes apparaissant dans ces ensembles est lui-même fini. L'ensemble des égalités induites par $R \wedge S \wedge \Gamma$ est donc fini. A l'itération k, l'ensemble E'_k contient des égalités qui sont conséquence de $R \wedge S \wedge \Gamma$ et ne contient aucune des égalités de E_{k-1} : ce sera aussi le cas pour E_k. Donc si Ct-solve ne rapporte aucune inconsistance, E devient vide après un nombre d'itérations fini, et l'algorithme termine.

La correction de l'algorithme Consistance-1 se déduit de celle de PC2 : si le résultat est **False** après k itérations, alors le réseau C à cette itération est inconsistant (par rapport à $\mathcal{O}$), comme celui-ci est une conséquence de $\mathcal{O}, \mathcal{E}$ et $\Gamma \wedge R \wedge S$, ce dernier ensemble est inconsistant pour $\mathcal{O}$ et $\mathcal{E}$ réunies.

Si $\Gamma \wedge R \wedge S$ est inconsistant (pour $\mathcal{O}$ et $\mathcal{E}$), alors soit le premier appel de Ct-solve retourne **False**, soit il faut un nombre fini d'itérations pour générer une égalité inconsistante avec les contraintes de C. Comme Ct-solve est exhaustif à chaque opération pour identifier les égalités (car PC2 travaille sur et construit le graphe complet), l'algorithme retourne **False**.

Exemple 2.2 *Soit Γ l'ensemble de contraintes suivant :*

$$\Gamma = \{g(y) = A,\ x = z\}$$

et R et S les réseaux de contraintes correspondant aux ensembles :

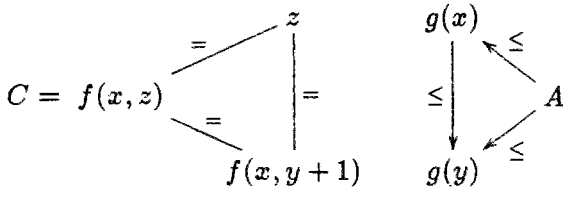
$$R = \{f(x,z) \leq z,\ z \leq f(x,y+1)\}$$
$$S = \{f(x,y+1) \leq f(x,z),\ A \leq g(x),\ g(x) \leq g(y)\}$$

L'ensemble $\Gamma \wedge R \wedge S$ est inconsistant, puisqu'il est possible d'en déduire à la fois $z = x = y + 1$ et $x = y$, ce qui est contradictoire.

L'appel de Consistance-1(Γ, R, S) *se déroule ainsi : le premier appel à* Ct-merge *pour fusionner R et S retourne l'ensemble d'équations*

$$E = \{f(x,z) = z,\ z = f(x,y+1),\ f(x,y+1) = f(x,z)\}$$

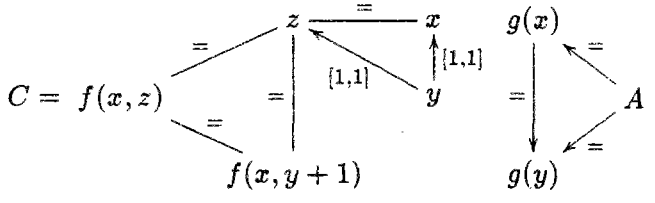
et le réseau de contraintes :



Ensuite, la simplification de $E \cup \Gamma$ *retourne les équations :*

$$E' = \{f(x,z) = z,\ z = f(x,y+1),\ y+1 = z,\ g(y) = A,\ x = z\}$$

et après application de Ct-solve *sur* E' *et* C*, le réseau obtenu contient deux nouveaux points* x *et* y *:*



et les nouvelles équations de ce réseau sont :

$$E = \{g(x) = A,\ g(x) = g(y)\}$$

La première itération produit par simplification de E *l'équation*

$$E' = \{x = y\}$$

qui, ajoutée à C *par* Ct-solve*, produit une inconsistance sur l'arc de* y *à* x*.*

Un deuxième algorithme

De la même manière que précédemment, il est possible de réaliser un compromis entre l'usage coûteux d'un algorithme de propagation de contraintes et certaines des tâches qui sont effectuées par un algorithme d'unification classique. En effet, une substitution est une forme simplifiée d'un ensemble d'équations entre termes, pour lesquels on économisera alors sur la taille de l'ensemble de contraintes à satisfaire.

Le deuxième algorithme de test de la consistance utilise cette idée : il s'agit d'une variante de l'algorithme Consistance-1. On a donc un algorithme d'unification plus sophistiqué qui construit une substitution à partir d'un ensemble d'égalités, sans pour autant produire un échec comme l'algorithme classique. Cet algorithme, appelé Simplify-2 applique sur un ensemble d'égalités E les règles de la figure 2.4 tant que cela est possible. Ce processus est noté $\langle \sigma, E' \rangle$ = Simplify-2(E). Au départ de l'algorithme, la substitution σ est vide.

$$\begin{array}{lll}
\langle \sigma, \{x = x\} \cup E\rangle & \rightarrow \langle \sigma, E\rangle & (S_1)\\
\langle \sigma, \{v = t\} \cup E\rangle & \rightarrow \langle \{v \mapsto t\} \circ \sigma, E_{[v/t]}\rangle & (S_2)\\
\quad v \text{ est une variable et } t \text{ un terme} & & \\
\quad \text{où n'apparaît pas } v & & \\
\langle \sigma, \{t = v\} \cup E\rangle & \rightarrow \langle \{v \mapsto t\} \circ \sigma, E_{[v/t]}\rangle & (S_3)\\
\quad v \text{ est une variable et } t \text{ un terme} & & \\
\quad \text{où n'apparaît pas } v & & \\
\langle \sigma, \{v + a = t\} \cup E\rangle & \rightarrow \langle \{v \mapsto t - a\} \circ \sigma, E_{[v/t-a]}\rangle & (S_4)\\
\quad v \text{ est une variable, } a \text{ un nombre} & & \\
\langle \sigma, \{v - a = t\} \cup E\rangle & \rightarrow \langle \{v \mapsto t + a\} \circ \sigma, E_{[v/t+a]}\rangle & (S_5)\\
\quad v \text{ est une variable, } a \text{ un nombre} & & \\
\langle \sigma, \{f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)\} \cup E\rangle & \rightarrow \langle \sigma, \{s_1 = t_1, \cdots, s_n = t_n\} \cup E\rangle & (S_6)
\end{array}$$

FIG. 2.4 - *Les règles de simplification des contraintes d'égalité, appliquées par* Simplify-2. *L'opérateur* $\circ$ *désigne la composition de deux substitutions.*

L'intérêt de cette transformation est de diminuer la taille de l'ensemble d'égalités, en en extrayant toutes les égalités possibles dont un des membres est une variable : on construit ainsi une substitution σ telle que $\sigma(R \wedge S) \wedge E$ est équivalent à $R \wedge S \wedge \Gamma$. Parallèlement, on traite les termes fonctionnels grâce à la règle de décomposition (S_6).

L'algorithme 2.3 réalise le test de la consistance en utilisant la paire

$$\langle \text{substitution, égalités}\rangle$$

produite par Simplify-2. Cet algorithme retourne aussi une paire $\langle \sigma, C'\rangle$ où σ est une substitution et C un ensemble de contraintes minimales. L'algorithme Consistance-2 utilise la fonction Apply qui applique une substitution à un ensemble de contraintes sous forme d'un réseau de contraintes minimales: comme pour Ct-merge, certains nœuds peuvent être fusionnés si l'application de la substitution leur donne les mêmes étiquettes.

Si les deux clauses initiales sont comme précédemment

$$\{P(x_1, \cdots, x_n)\} \cup C \parallel R \qquad \text{et} \qquad \{\neg P(y_1, \cdots, y_n)\} \cup D \parallel S$$

avec Γ qui est l'ensemble des égalités temporelles parmi

$$\{x_1 = y_1, \ldots, x_n = y_n\}$$

alors, la clause résolvante est calculée par :

$$\sigma(C \cup D) \parallel C'$$

avec $\langle \sigma, C'\rangle = \text{Consistance-2}(\Gamma, R, S)$.

Consistance-2(Γ, R, S) :
  soit $\langle C, E\rangle \leftarrow$ Ct-merge(R, S)
  si ($C =$ **False**) **alors retourner** (**False**)
  soit $\langle \sigma, E'\rangle \leftarrow$ Simplify-2($E \cup \Gamma$)
  $C \leftarrow$ Apply(σ, C)
  $\langle C, E\rangle \leftarrow$ Ct-solve(E', C)
  tant que ($C \neq$ **False**) et ($E \neq \emptyset$) **faire**
    $\langle \sigma', E'\rangle \leftarrow$ Simplify-2(E)
    $E' \leftarrow E' - E$
    $\sigma \leftarrow \sigma' \circ \sigma$
    $C \leftarrow$ Apply(σ', C)
    $\langle C, E\rangle \leftarrow$ Ct-solve(E', C)
  fin
  retourner ($\langle \sigma, C\rangle$)

Algorithme 2.3: *Deuxième algorithme de test de la consistance de l'ensemble de contraintes* $\Gamma \wedge R \wedge S$, *optimisé pour calculer une substitution des variables lorsque cela est possible.*

Ce processus assure que l'ensemble des contraintes portées par les clauses est le plus petit possible : ce qui est toujours intéressant compte tenu du coût des algorithmes de propagation de contraintes. De plus, si la valeur d'une variable est complètement déterminée par les contraintes, cette valeur sera identifiée et représentée par une substitution : l'intérêt ne semble pas immédiat au premier abord, mais n'est pas négligeable lorsque l'on envisage l'application de ce principe en programmation logique avec contraintes. Dans ce dernier cas, les règles usuelles de la négation par échec imposent aux littéraux négatifs d'être sans variables pour la correction de ce principe. Un tel test peut être gardé en l'état puisqu'il n'existe plus dans les clauses de variables dont la valeur puisse être déterminée de manière unique.

2.5 Conclusions

Dans ce chapitre, nous avons présenté l'intérêt de la résolution avec contraintes pour raisonner avec des logiques temporelles réifiées. Ce principe de résolution permet de tirer parti des nombreux travaux sur les systèmes de gestion de contraintes temporelles et fournit un cadre formel qui permet d'intégrer ceux-ci dans des systèmes déductifs généraux basés sur la résolution. En particulier, les caractéristiques indispensables d'un tel système de gestion de contraintes sont la complétude et l'incrémentalité.

Les limites de cette approche sont fixées par la possibilité de décider de la consistance d'un ensemble de relations temporelles. Nous avons vu que les termes fonc-

tionnels posaient un problème à cet égard. Les deux algorithmes présentés dans cette partie travaillent sur le formalisme des contraintes temporelles quantitatives de Dechter, où la syntaxe des termes temporels est augmentée de l'usage de symboles fonctionnels quelconques.

Cependant, la résolution avec contraintes ne répond pas au problème plus général de la mise en œuvre du raisonnement temporel avec une sémantique préférentielle pour la logique utilisée. De la même manière, la persistance n'est pas résolue ainsi. Le seul gain est de l'ordre de l'efficacité à un « niveau bas » des inférences.

Chapitre 3

Abduction et raisonnement temporel

Ce chapitre introduit l'abduction et explique l'intérêt de ce mode de raisonnement pour le raisonnement temporel. Nous proposons ensuite une revue succinte des méthodes et des algorithmes proposés dans la littérature sur le raisonnement abductif et la génération d'hypothèses, et ceci indépendamment de son application au raisonnement temporel. Nous motivons ensuite les choix qui sont à la base de la procédure qui sera décrite dans le chapitre 4.

3.1 L'abduction

La première mention de l'abduction a été faite par Peirce (repris dans (Hartshorne & Weiss, 1958) et (Fann, 1970)) comme un raisonnement de l'effet vers les causes. Le but est de produire pour un fait observé une explication de celui-ci relativement à une théorie donnée. Pour illustrer son propos, Peirce donne les exemples suivants :

Déduction : de la règle « tous les haricots de ce sac sont blancs » et de la proposition « ces haricots viennent de ce sac », on déduit « ces haricots sont blancs » ;

Induction : de « ces haricots viennent de ce sac » et de « ces haricots sont blancs », on déduit « tous les haricots de ce sac sont blancs » ;

Abduction : de « tous les haricots de ce sac sont blancs » et de « ces haricots sont blancs », on déduit « ces haricots viennent de ce sac ».

La déduction est donc l'application de règles générales à des cas particuliers, et dont on infère le résultat, alors que l'induction infère la régle générale à partir d'un cas particulier et du résultat. L'abduction est supposée jouer un grand rôle dans les processus de découverte scientifique. On peut citer le cas de la découverte de la forme elliptique de l'orbite des planètes par Kepler : comme tous les gens de son époque, Kepler a appris que l'orbite des planètes était circulaire. Cependant l'examen de deux points particuliers de l'orbite de la planète Mars lui permet de se rendre compte que l'orbite n'est pas circulaire. L'hypothèse faite alors par Kepler est que l'orbite est elliptique, et la vérification de cette hypothèse se fait déductivement en calculant un troisième point de l'orbite supposée de Mars. L'hypothèse est renforcée lorsque Kepler trouve Mars en ce point, mais l'hypothèse ne devient jamais vraie : les confirmations ultérieures ne font que renforcer sa plausibilité, jusqu'à son éventuelle invalidation par une observation contradictoire.

Dans le cadre de l'intelligence artificielle, l'abduction donne lieu à de nombreux travaux qui trouvent application dans le cadre du diagnostic (Pople, 1973; Cox & Pietrzykowsky, 1987; Ayeb, Marquis, & Rusinowitch, 1990), du langage naturel et de la compréhension de textes (Stickel, 1990), de la planification (Eshghi, 1988; Missiaen, 1991) ou encore de la mise à jour de bases de données (Kakas & Mancarella, 1990a). Les explications produites doivent en général satisfaire un ou plusieurs critères : simplicité, pertinence, etc. Pour l'abduction au sens de l'intelligence artificielle, cette explication est « particulière », c'est à dire qu'elle concerne des objets du domaine et certaines de leur propriétés plutôt qu'une loi générale sur le domaine (ce que fait alors l'induction).

3.2 Intérêt de l'abduction pour le raisonnement temporel

Dans le cas particulier du raisonnement temporel, l'abduction possède plusieurs intérêts. En premier lieu, l'abduction est une forme de raisonnement plus puissante que des procédures de preuve ou d'inférence habituelle. La capacité à répondre à une requête par un ensemble d'hypothèses permettant de satisfaire celle-ci lui ouvre le champ de nombreuses applications, en particulier dans le traitement des langues naturelles où les aspects temporels sont très riches, et en planification lorsqu'on utilise une logique temporelle comme formalisme de représentation des actions et de leurs effets.

3.2.1 L'abduction pour gérer la persistance

Un autre avantage de l'abduction pour le raisonnement temporel concerne la prise en compte de la persistance temporelle des faits. Ce problème, spécifique au raisonnement temporel, est celui qui se pose lorsque l'on souhaite exprimer que ce qui était vrai à un moment donné peut toujours l'être par après, si rien ne s'y oppose. Dans le chapitre 1 nous avons examiné diverses approches permettant de prendre en compte la persistance par le biais d'un critère de préférence sur les modèles des théories logiques. Une autre possibilité, évoquée par (Shanahan, 1989a), est de représenter la persistance comme une hypothèse effectuée lorsqu'elle est nécessaire. Cette approche évite la complexité dûe à la mise en œuvre des raisonnements non monotones et des approches par critères de préférence entre modèles. Il est alors simplement nécessaire de représenter la persistance comme une relation particulière, d'exprimer ses liens avec les autres relations de la logique temporelle, sans incorporer dans la logique le moyen de déduire une persistance par défaut : le mécanisme d'inférence prend en charge cet aspect de la persistance. Cet usage de l'abduction pour remplacer un raisonnement par défaut a aussi été défendu par (Poole, 1988).

A titre d'exemple, dans le cas du langage défini dans le paragraphe 2.2.2 du chapitre 2, et dont les axiomes sont repris sur la figure 3.1, les persistances peuvent être représentées par un nouveau symbole de prédicat, que nous noterons $persist(t_1, t_2, p)$, où t_1 et t_2 sont des instants, et p est une proposition (un terme atemporel).

A priori, avec un tel prédicat, il n'est plus nécéssaire d'essayer de décrire la persistance par un axiome comme (6). Il est juste nécessaire de donner une idée de la persistance « statique », comme étant le fait que la proposition est vraie pendant au moins l'intervalle de persistance. Ceci peut être défini par la relation suivante :

$$\forall\, t_1, t_2, p \;\; persist(t_1, t_2, p) \;\Leftrightarrow\; [\forall\, t_3 \;\; (t_1 \leq t_3 \leq t_2) \Rightarrow true(t_3, p)] \qquad (6')$$

L'axiome (2) est alors remplacé par l'équivalence (2') suivante :

$$\forall\, t_1, t_2, p \;\; hold(t_1, t_2, p) \;\Leftrightarrow\; (begin(t_1, p) \;\wedge\; persist(t_1, t_2, p) \;\wedge\; end(t_2, p))$$

$$\forall\, t_1,t_2,p\ \ hold(t_1,t_2,p) \Rightarrow (t_1 < t_2) \tag{1}$$

$$\forall\, t_1,t_2,p\ \ hold(t_1,t_2,p) \Leftrightarrow \left(\begin{array}{l} begin(t_1,p)\ \wedge \\ (\forall\, t_3\ \ (t_1 \leq t_3 \leq t_2) \Rightarrow true(t_3,p))\ \wedge \\ end(t_2,p) \end{array} \right) \tag{2}$$

$$\forall\, t_1,t_2\ \ (t_1 < t_2) \Rightarrow [\exists\, t_3\ \ (t_1 < t_3 < t_2)] \tag{3}$$

$$\forall\, t_1,t_2,t_3,p\ \ [hold(t_1,t_2,p)\ \wedge\ begin(t_3,p)] \Rightarrow [(t_3 \leq t_1)\ \vee\ (t_2 < t_3)] \tag{4}$$

$$\forall\, t_1,t_2,t_3,p\ \ [hold(t_1,t_2,p)\ \wedge\ end(t_3,p)] \Rightarrow [(t_3 < t_1)\ \vee\ (t_2 \leq t_3)] \tag{5}$$

$$\forall\, t_1,p\ \ begin(t_1,p) \Rightarrow [\exists\, t_2\ \ (t_1 < t_2) \wedge hold(t_1,t_2,p)] \tag{6}$$

FIG. 3.1 - *Rappel des axiomes de la logique temporelle définie au chapitre 2.*

Pour le problème du YSP, exposé au même paragraphe 2.2.2, déduire la formule $end(4, loaded)$, c'est à dire le « succès » du coup de feu au temps 4, est possible en supposant par exemple $persist(T_0, 4, loaded)$. Cette hypothèse permet d'obtenir $true(4, loaded)$, ce qui permet par la règle concluant sur $end(t, loaded)$ d'obtenir $end(4, loaded)$. Remarquons que $begin(T_0, loaded)$ n'intervient pas dans cette preuve, ce que l'on peut interpréter comme le fait que la description de la persistance que nous avons adoptée est trop « statique », puisqu'elle ne prend pas en compte ce qui est vrai (connu ou prouvé) dans le passé par rapport à la période de persistance: il est possible de supposer une persistance sans fondement, c'est à dire qui ne soit pas reliée à une vérité antérieure de la propriété que l'on souhaite voir persister. Il serait alors souhaitable de définir la persistance par un axiome comme:

$$\forall\, t_1,t_2,p\ \ [begin(t_1,p)\ \wedge\ persist(t_1,t_2,p)\ \wedge\ (t_1 < t_2)] \Rightarrow true(t_2,p)$$

en lieu et place de (6'). L'inconvénient de cette relation est d'être plus forte que la précédente, et oblige en particulier à être capable de situer (prouver) le début de toute période de vérité intervenant dans le problème traité.

3.2.2 L'abduction pour la planification

Un domaine où le temps joue un rôle crucial est la planification. L'usage de l'abduction pour la planification a été étudiée en particulier par (Eshghi, 1988) et (Missiaen, 1991) en utilisant le calcul d'événements de Kowalski et Sergot comme formalisme de représentation des actions et des effets de celles-ci. L'intérêt des formalismes développés pour le raisonnement temporel est d'offrir un langage de représentation plus riche que les formalismes d'opérateurs à la STRIPS utilisés en planification (Allen, 1991). La planification est alors vue comme un processus d'inférence dans une logique temporelle (Hertzberg, 1994). En pratique, il est alors possible d'associer une durée aux actions modifiant le monde, de raffiner celles-ci par décomposition de leurs intervalles d'occurrence, d'autoriser des actions simultanées et concurrentes, etc. L'abduction est donc une approche de la planification où un plan est un ensemble de formules closes permettant alors de déduire une formule décrivant l'état final désiré comme conséquence de l'éxécution du plan.

3.3 Méthodes et algorithmes pour l'abduction

Parmi toutes les approches de l'abduction en intelligence artificielle, deux présentent un intérêt pour nous : l'assemblage d'hypotheses (*set-cover-based approach*) (Bylander, Allemang, Tanner, & Josephson, 1991) et l'approche logique (Eshghi & Kowalski, 1989; Kakas & Mancarella, 1990c). Nous examinons brièvement dans cette partie ces deux approches et le type de méthodes et d'algorithmes proposés. Cette partie n'introduit pas d'éléments nouveaux et l'on peut trouver une présentation plus large du raisonnement abductif dans (Paul, 1993).

3.3.1 L'abduction par assemblage d'hypothèses

Dans cette approche, le problème est de sélectionner dans un ensemble d'hypothèses prédéterminées un sous-ensemble de celles-ci expliquant les observations considérées et vérifiant différents critères de minimalité, de plausibilité ou tout autre critère adéquat par rapport au domaine considéré. Une analyse détaillée de la complexité d'un tel problème se trouve dans (Bylander et al., 1991), et les principaux résultats de celle-ci sont décrits dans cette partie.

Plus formellement, un tel problème est défini par un quadruplet (Φ, Ω, e, pl) où Φ est un ensemble fini d'hypothèses, Ω un ensemble fini d'observations, e une fonction qui prend en argument un sous-ensemble de Φ et qui retourne un sous-ensemble de Ω, et pl une fonction qui retourne pour chaque sous-ensemble de Φ une valeur dans un ensemble partiellement ordonné. Intuitivement, la fonction e décrit pour chaque combinaison d'hypothèses prises dans Φ les observations qui en découlent, et pl décrit la plausibilité d'un ensemble d'hypothèses. On se donne ensuite un sous-ensemble Ω' de Ω et l'on doit alors identifier les hypothèses qui expliquent toutes les observations de Ω'. Une *explication* de Ω' est définie comme un sous-ensemble H de Φ vérifiant les deux conditions suivantes :

1. H est *complète* : $e(H) = \Omega'$;
2. H est *parsimonieuse*, c'est à dire qu'il n'existe pas de sous-ensemble propre de H qui explique les mêmes observations que H :

$$\forall H' \subset H \quad e(H) \not\subseteq e(H')$$

De même, H est la *meilleure explication* de Ω' si il n'existe pas d'autre explication H' des mêmes observations ayant une plausibilité supérieure à celle de H.

Un point important dans cette analyse est que les fonction e et pl doivent être calculables. Il y a plusieurs cas particuliers identifiés par (Bylander et al., 1991) où les propriétés de e influent directement sur la complexité du problème d'abduction :

- les hypothèses sont indépendantes, c'est à dire que l'on peut calculer $e(H)$ comme l'union de tous les ensembles $e(\{h\})$ pour tout h appartenant à H.

soit $W \leftarrow \Phi$
pour tout $h \in \Phi$ **faire**
  si $e(W \setminus \{h\}) = \Omega'$ **alors**
    $W \leftarrow W \setminus \{h\}$
retourner W

Algorithme 3.1: Recherche d'une explication de Ω' à partir des hypothèses de l'ensemble Φ.

Dans ce cas, le résultat de (Bylander et al., 1991) est qu'il existe un algorithme en $O(nC_e + n^2)$ qui trouve une explication si il en existe une, où n est le nombre d'hypothèses de Φ et d'observations à expliquer, et C_e est le coût d'un appel à la fonction e. Par contre, trouver toutes les explications est un problème NP, alors que trouver la meilleure explication est un problème P si les plausibilités de chaque hypothèse prise individuellement sont distinctes et totalement ordonnées. Si ceci n'est pas vérifié, ce dernier problème est NP.

- Les hypothèses sont monotones, c'est à dire qu'une hypothèse « composite » n'explique pas moins d'observations qu'un de ses sous-ensembles :

$$\forall H, H' \subseteq \Phi \quad (H \supseteq H' \Rightarrow e(H) \subseteq e(H'))$$

Dans ce cas, et comme précédemment, il existe un algorithme en $O(nC_e + n^2)$ qui trouve une explication s'il en existe une. Les mêmes résultats que précédemment sont valables. Pour les deux cas précédents (hypothèses indépendantes et/ou monotones), l'algorithme 3.1 donné par (Bylander et al., 1991) trouve une explication lorsqu'il en existe une (ce qui se vérifie en testant la condition $\Omega' \subseteq e(\Phi)$), et le même algorithme, mais avec la boucle principale qui examine les hypothèses par ordre croissant de plausibilité, permet de trouver la meilleure explication dans le cas où les plausibilités des hypothèses individuelles sont distinctes et totalement ordonnées.

- Il existe des cas d'incompatibilité entre hypothèses, c'est à dire que toutes les combinaisons d'hypothèses ne sont pas licites. Dans la définition d'un tel problème, on rajoute au triplet (Φ, Ω, e) un ensemble $\mathcal{I}$ dont les éléments sont des paires d'hypothèses qui indiquent tous les cas d'incompatibilités. Pour une hypothèse composite H telle qu'il existe une paire I de $\mathcal{I}$ incluse dans H, on pose $e(H) = \emptyset$. Dans ce cas, les trois problèmes suivants sont NP-complets :

  1. déterminer s'il existe une explication ;
  2. déterminer une explication ;
  3. déterminer la meilleure explication.

- Certaines hypothèses invalident des observations que d'autres hypothèses expliquent. Pour chaque hypothèse h on définit $e^+(h)$ comme l'ensemble des observations de Ω expliquées par h, et $e^-(h)$ comme l'ensemble des observations

invalidées par h. Dans ce cas, et comme précédemment, tous les problèmes sont NP-complets.

Ces résultats sont bien sûr peu encourageants, sachant que la plupart des domaines et des problèmes concrets correspondent aux deux dernières classes de problèmes. Cependant, il est possible d'envisager l'utilisation d'heuristiques pour guider la recherche des explications, et l'on peut utiliser d'autres formes de connaissances non prises en compte dans la description formelle du problème pour éliminer d'emblée certaines hypothèses. De la même manière, une borne supérieure sur la taille des explications réduit le coût de la recherche.

3.3.2 Approches logiques de l'abduction

L'approche de l'abduction par la logique est aussi la plus connue. Dans cette approche, les liens de causes à effet entre hypothèses et observations font partie d'une théorie logique, et sont décrits à l'aide de l'implication. Pour une théorie $\mathcal{T}$ et un ensemble de formules supposables A, une explication d'une formule ω est une formule ϕ prise dans A et qui vérifie les conditions suivantes :

1. $\mathcal{T} \cup \phi \vdash \omega$, c'est à dire que ω est dérivable de $\mathcal{T} \cup \phi$;
2. $\mathcal{T} \cup \phi$ est consistant.

Le principal inconvénient de cette approche est qu'elle est construite sur les relations de dérivabilité et de consistance, qui dans le cadre de la logique du premier ordre sont très difficiles à mettre en œuvre. L'abduction est donc encore un problème difficile suivant cette approche.

Selman et Levesque (Selman & Levesque, 1990) ont classifié différents problèmes d'abduction. Leur analyse suppose que la théorie $\mathcal{T}$ est un ensemble de clauses de Horn propositionnelles, et que l'on s'intéresse aux explications d'une proposition particulière q. Leurs résultats sont les suivants :

Théorème 3.1 *(Selman & Levesque, 1990) Soit p une proposition apparaissant dans $\mathcal{T}$, alors la génération d'une explication de q qui contient p est un problème NP-dur.*

Théorème 3.2 *(Selman & Levesque, 1990) Soit A un ensemble d'hypothèses (propositions), trouver une explication de q formée de propositions prises dans A est un problème NP-dur.*

Il existe malgré tout des cas où le problème de l'abduction n'est pas NP. L'un de ces cas est décrit dans (Eshghi, 1993) en se basant sur le fait que tester la consistance d'une théorie peut être fait en temps linéaire par résolution unitaire moyennant certaines conditions sur la forme des clauses de la théorie. Dans le résultat d'Eshghi, la

théorie $\mathcal{T}$ est donnée sous la forme d'un ensemble de clauses propositionnelles, et l'on suppose de plus qu'une solution Δ au problème d'explication de ω est déductivement close :

$$(a \in A \text{ et } \mathcal{T} \cup \Delta \vdash a) \Rightarrow a \in \Delta$$

Le résultat principal d'Eshghi est le suivant : Δ est une solution minimale qui explique g si et seulement si Δ est une *minimisation* de A par rapport à l'ensemble de clauses $\mathcal{T} \cup \mathsf{Only\text{-}if}(\mathcal{T}, \Theta) \cup \{g\}$. Dans ce résultat, Θ désigne l'ensemble de toutes les propositions apparaissant dans $\mathcal{T}$ et n'appartenant pas à A, et $\mathcal{T} \cup \mathsf{Only\text{-}if}(\mathcal{T}, \Theta)$ désigne la pseudo-complétion de $\mathcal{T}$ par rapport à l'ensemble de propositions Θ. La pseudo-complétion de $\mathcal{T}$, très proche de la notion habituelle de complétion en programmation logique, désigne l'ensemble de clauses comprenant $\mathcal{T}$ et tel que toute proposition non-supposable ne peut pas être vraie autrement que si une clause de $\mathcal{T}$ permet de conclure ainsi. Autrement dit, toute information non supposable qui n'est pas donnée par les clauses de $\mathcal{T}$ est fausse.

L'ensemble de propositions Δ est une minimisation de A par rapport à l'ensemble de clauses C si il existe un modèle M de C minimisant A et tel que $\Delta = M \cap A$. Un modèle est vu comme un ensemble de propositions représentant une affectation de la valeur de vérité Vrai à toutes les propositions du modèle et Faux aux autres. Le modèle M de C minimise A si il n'existe pas d'autre modèle M' de C tel que $(M' \cap A) \subset (M \cap A)$. La minimisation de A par rapport à C est donc le plus petit sous-ensemble de A inclus dans un modèle de C. Cette minimisation se calcule par exemple à l'aide d'un algorithme qui parcourt les hypothèses une à une et les ajoute à Δ si $C \cup M \cup \{a\}$ est consistant, où M est l'ensemble des opposés des hypothèses déja traitées et non retenues, et a l'hypothèse en cours de traitement. Cet algorithme est glouton, c'est à dire qu'il n'est pas nécessaire de revenir sur l'opération de choix d'une hypothèse.

Pour que cette méthode soit praticable, il faut donc que le test de consistance dans l'algorithme de minimisation soit réalisable. En pratique, cela est possible pour les théories uniquement formées de clauses de Horn où ce test se fait en temps polynomial en utilisant la résolution unitaire, et parfois en temps linéaire si la théorie satisfait certaines conditions (Dowling & Gallier, 1984). Eshghi donne en particulier une méthode permettant de tester si une théorie est réfutable par résolution unitaire basée sur la recherche de motifs particuliers dans le graphe de connection des clauses de l'ensemble.

En pratique, les résultats de (Eshghi, 1993) montrent qu'il n'est pas toujours nécessaire que $\mathcal{T} \cup \mathsf{Only\text{-}if}(\mathcal{T}, \Theta)$ satisfasse cette condition. En particulier, le test de consistance n'est alors pas complet, et il faut vérifier explicitement la correction du résultat trouvé pour obtenir un algorithme utilisable, qui reste correct mais qui est incomplet.

3.3.3 Remarques sur les méthodes d'abduction

Tous les résultats et les méthodes qui viennent d'être décrits présentent deux inconvénients majeurs (en plus de leur difficulté) :

- seul le cas propositionnel est pris en compte, et la complexité de ce seul cas n'est qu'une indication pessimiste de la difficulté de l'abduction en logique du premier ordre ;
- l'ensemble A des hypothèses est une donnée du problème, et aucune méthode n'est indiquée pour obtenir cet ensemble, que ce soit la fonction e dans le cadre de l'abduction par assemblage d'hypothèses, ou de l'ensemble A dans les approches logiques. De plus, lorsqu'on envisage le passage à la logique du premier ordre, il est plus réaliste d'envisager de décrire les hypothèses comme des instances de certaines formules ouvertes : énumérer l'ensemble A devient alors impraticable si le langage utilise des symboles fonctionnels par exemple.

Compte tenu de ces remarques, nous allons maintenant nous intéresser à quelques méthodes de génération d'hypothèses proposées dans la littérature. L'intérêt de ces méthodes est de ne pas avoir besoin de la fonction e pour les liens entre les hypothèses et leurs effets : tout est déduit à l'aide de la théorie $\mathcal{T}$.

3.3.4 Méthodes de génération d'hypothèses

Les méthodes de génération d'hypothèses sont toutes basées sur le résultat suivant obtenu d'après le théoreme de la déduction :

$$\mathcal{T}, \phi \vdash \omega \qquad \text{si et seulement si} \qquad \mathcal{T}, \neg\omega \vdash \neg\phi$$

c'est à dire qu'une explication ϕ est une conséquence logique de la théorie $\mathcal{T}$ et de la négation de l'observation à expliquer ω. C'est pourquoi les méthodes de génération d'hypothèses sont basées sur des mécanismes déja développés pour le problème de la déduction, en particulier l'usage du principe de résolution.

Les deux méthodes de génération d'hypothèses décrites dans cette partie sont basée sur la résolution avec une stratégie linéaire :

- La méthode de (Cox & Pietrzykowsky, 1986) permet de générer des explications sous la forme de formules de forme arbitraire, non closes ;
- la procédure de Pople (Pople, 1973).

Pour (Cox & Pietrzykowsky, 1986), étant donnée une théorie $\mathcal{T}$ et une formule ω représentant le fait à expliquer, une cause de ω est une formule ϕ telle que $\mathcal{T} \wedge \phi \vdash \omega$ et qui est consistante avec $\mathcal{T}$. Le calcul d'une cause se fait par la méthode suivante :

1. convertir $\mathcal{T}$ en forme normale ;

2. prendre la négation de ω, la convertir en forme normale et effectuer une dérivation linéaire à partir d'une des clauses de $\neg\omega$;
3. comme $\mathcal{T} \not\vdash \omega$ est *a priori* vérifié, on a deux cas possibles : soit la dérivation ne termine pas, soit elle termine sur une clause d (non vide) ;
4. si D est l'ensemble de toutes les clauses finales non vides de la dérivation, alors pour chaque d appartenant à D, on prend la négation de celle-ci (on obtient donc une conjonction de littéraux), et on applique la skolémisation inverse, c'est à dire que l'on remplace les termes de skolem par des variables universellement quantifiées. La formule ainsi obtenue est une cause de ω.

Les hypothèses ainsi calculées sont minimales par rapport à l'implication, c'est à dire qu'il n'y a pas d'autre cause de ω plus spécifique que ϕ et qui implique ϕ. La seule précaution à prendre est d'écarter les clauses finales d qui seraient subsumées par d'autres clauses de ce type. Les hypothèses générées sont aussi *basiques*, c'est à dire que toute explication d'une de ces hypothèses est triviale (implique directement l'hypothèse sans utiliser $\mathcal{T}$ ou est égale à l'hypothèse). Cet algorithme n'est pas complet, dans certains cas, décrits dans (Cox & Pietrzykowsky, 1986), il ne trouve pas toutes les solutions minimales et basiques.

La méthode de (Pople, 1973) est quelque peu différente. Cette méthode cherche à montrer $\mathcal{T} \supset \omega$. La théorie $\mathcal{T}$ est convertie sous forme normale conjonctive (conjonction de clauses), et l'observation à expliquer mise sous forme normale disjonctive (c'est à dire sous la forme d'une disjonction de conjonctions de littéraux) et après skolémisation des variables universellement quantifiées. Si ω est transformée ainsi :

$$d_1 \vee \ldots \vee d_n$$

où d_i est une conjonction de littéraux, toutes les manières de satisfaire cette disjonction :

$$(\neg d_1 \wedge \ldots \wedge \neg d_{n-1}) \supset d_n$$
$$\ldots$$
$$(\neg d_2 \wedge \ldots \wedge \neg d_n) \supset d_1$$

forment autant de sous-problèmes à résoudre pour montrer ω. La conjonction $\neg d_1 \wedge \ldots \wedge \neg d_{n-1}$ est ajoutée à $\mathcal{T}$ et l'on cherche à montrer d_n, et ceci pour tout d_i. Cette preuve se fait en chaînage arrière (analogue à la résolution SLD) en considérant toutes les contraposées possibles obtenues à partir des clauses de $\mathcal{T}$, et comme précédemment produit un certain nombre de clauses finales qu'on ne peut plus réduire. L'explication est construite comme précédemment à partir de ces clauses.

P. Marquis (Marquis, 1991b) caractérise, dans le cadre propositionnel, les explications minimales et consistantes de ω par rapport à $\mathcal{T}$ comme l'ensemble

$$\mathcal{PI}(\mathcal{T} \Rightarrow \omega) \setminus \mathcal{PI}(\neg\mathcal{T})$$

où $\mathcal{PI}$ représente l'ensemble des premiers implicants. Néanmoins, dans le cas de la logique du premier ordre, un résultat de (Marquis, 1991a) montre que l'abduction ne peut pas être facilement automatisée de cette manière.

3.3.5 L'abduction et la programmation logique

Il existe beaucoup de travaux où l'abduction est envisagée dans le cadre de la programmation logique. A première vue, rien ne devrait les distinguer des approches logiques exposées dans les deux paragraphes précédents. Cependant, un des intérêts, et non des moindres, de la programmation logique, est de fournir un cadre effectif au raisonnement basé sur la logique. Ce cadre est bien sûr très affaibli par rapport à celui de la logique du premier ordre classique, puisque le langage de la programmation logique n'est qu'un sous ensemble de la logique du premier ordre, mais les méthodes développés pour la programmation logique sont en général simples, dans leur principe comme dans leur mise en œuvre, cette dernière pouvant être très efficace, comme en témoignent les travaux sur la compilation et l'exécution des programmes logiques sur une machine abstraite comme la WAM (Warren, 1983).

Rappelons que le langage de la programmation logique (Lloyd, 1987) est celui des clauses de la forme $a \leftarrow l_1, \ldots, l_n$ où a est un atome appelé la tête de la clause, et les littéraux $l_1, \ldots, l_n$ forment le corps de la clause. Les variables des clauses sont implicitement quantifiées universellement. Un but est une clause sans tête $\leftarrow l_1, \ldots, l_n$, et si les variables de cette clause sont $x_1, \ldots, x_m$, elle correspond à la formule $\forall x_1, \ldots, x_m \ (\neg l_1 \vee \ldots \vee \neg l_n)$ ou encore $\neg(\exists x_1, \ldots, x_m \ l_1 \wedge \ldots \wedge l_n)$. Un programme logique est un ensemble de clauses.

L'abduction dans ce cadre est décrite par un triplet (P, IC, A) où P est un programme logique, IC est un ensemble de clauses sans tête appelées contraintes d'intégrité, et A est un ensemble de symboles de prédicats utilisés dans P et que l'on appelle *supposables*. Ceci signifie que les hypothèses constituant une explication sont des instances closes (sans variables) de ces prédicats. Un tel cadre constitue la base de la plupart des approches par la programmation logique, en particulier (Eshghi & Kowalski, 1989) et (Kakas & Mancarella, 1990c) qui seront détaillées dans la suite. Les conditions vérifiées par les hypothèses sont définies à partir de diverses sémantiques possibles pour les programmes logiques. Un exemple de sémantique donnée dans (Kakas & Mancarella, 1990c) utilise la sémantique des modèles stables (Gelfond & Lifschitz, 1988) :

Définition 3.1 *(Kakas & Mancarella, 1990c) la formule q est expliquée par les hypothèses Δ si et seulement si il existe un modèle stable M de $P \cup \Delta$ tel que :*

$$M \models q \qquad et \qquad M \models IC$$

Cette définition n'est qu'une variante sur la notion de *generalized stable model* proposée par les mêmes auteurs pour donner une sémantique à l'abduction (Kakas & Mancarella, 1990b).

Le domaine de l'abduction en programmation logique, aussi appelée *programmation logique abductive*, a été initiée par (Eshghi & Kowalski, 1989), et (Kakas, Kowalski, & Toni, 1992) présente tous les travaux réalisés sur cette base. D'autres procédures ont été proposées, en particulier dans (Satoh & Iwayama, 1991, 1992) et

(Denecker & De Schreye, 1992). Nous décrivons maintenant les procédures proposées par (Kakas et al., 1992) et (Denecker & De Schreye, 1992), car elles sont les plus représentatives de la programmation logique abductive.

La procédure de Kakas et Mancarella

La procédure proposée par Kakas et Mancarella (abrégé K&M dans la suite) pour générer des explications est basée sur la résolution SLD. Elle est décrite comme une extension de la procédure de (Eshghi & Kowalski, 1989) qui interprète abductivement la négation par échec en construisant des hypothèses négatives correspondant aux négations rencontrées au cours de la réfutation. La procédure de K&M étend ce principe à des hypothèses quelconques, et interprète aussi de manière abductive la négation par échec en autorisant des hypothèses qui soient des littéraux négatifs.

Les littéraux supposables (instances de prédicats pris dans l'ensemble A) sont notés à l'aide du signe $\dagger$. Pour interpréter comme une hypothèse la négation par échec, tout littéral négatif $\neg L$ apparaissant dans le corps des clauses de P est remplacé par un nouveau littéral positif noté L^* qui représente donc l'opposé de L. Le programme ainsi transformé est celui sur lequel travaille la procédure. On ajoute aussi à l'ensemble des contraintes d'intégrité toutes les contraintes de la forme $\leftarrow \alpha, \alpha^*$ pour tout littéral supposable α. De plus, on suppose qu'on dispose d'une règle de sélection $\mathcal{R}$ des littéraux dans le corps des clauses buts.

La procédure entrelace deux types de dérivations: la *derivation abductive*, et la *dérivation abductive*, tout en construisant incrémentalement l'ensemble d'hypothèses Δ. Schématiquement, la dérivation abductive est une dérivation SLD étendue de manière à ce que lorsqu'un littéral supposable est sélectionné dans le but courant, celui-ci soit supposé (c'est à dire ajouté à Δ). Ceci n'est effectivement réalisé que si la nouvelle hypothèse est consistante, ce qui est vérifié par une dérivation de consistance. Une telle dérivation part de l'hypothèse α et résoud celle-ci avec les contraintes d'intégrité: les clauses ainsi produites sont considérées comme des buts qui doivent échouer pour que les contraintes d'intégrité soient satisfaites. La dérivation de consistance regroupe donc un ensemble de dérivations simples à partir de ces buts, et qui se termine avec succès lorsque tous les buts ont échoué.

Une *réfutation abductive* est une dérivation abductive qui se finit par la clause vide. Lorsque la procédure termine sur l'état $\langle \square, \Delta^* \rangle$ en partant de $\langle Q, \{\} \rangle$, alors le sous ensemble de Δ^* obtenu en ne conservant que les prédicats supposables avant transformation du programme (on néglige donc les hypothèses correspondant à des littéraux négatifs) est une explication de Q.

Exemple 3.1 *Soit P le programme suivant où D est supposable et sa transforma-*

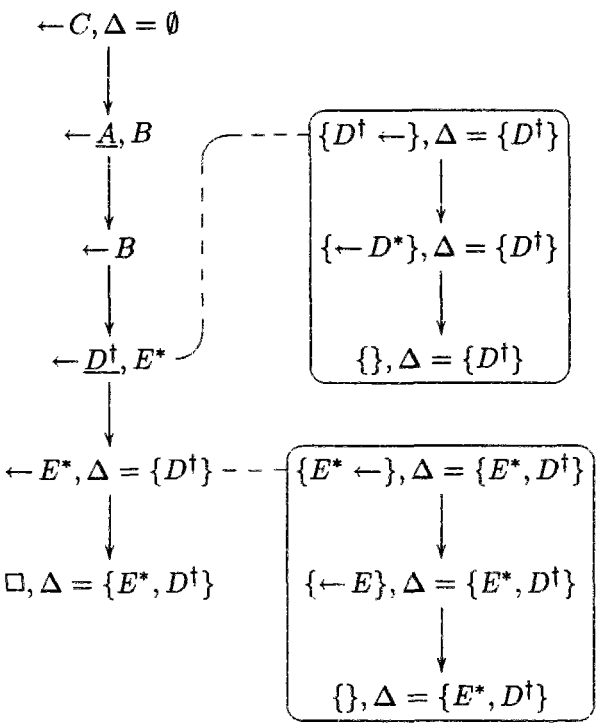


FIG. 3.2 - *Une réfutation abductive de $\leftarrow C$. Le littéral choisi pour continuer soit la réfutation, soit la dérivation est souligné. Les dérivations de consistance sont encadrées.*

tion par réecriture des littéraux négatifs et ajout de nouvelles contraintes d'intégrité :

$$\begin{array}{ll} & C \leftarrow A, B \\ C \leftarrow A, B & A \leftarrow \\ A \leftarrow & B \leftarrow D^\dagger, E^* \\ B \leftarrow D^\dagger, \neg E & \leftarrow E, E^* \\ & \leftarrow D^\dagger, D^* \end{array}$$

La requête $\leftarrow C$ produit la réfutation abductive montrée sur la figure 3.2. Les hypothèses expliquant C sont E^* et $D^\dagger$, c'est à dire qu'il existe un modèle stable $(\{A, B, C, D\})$ de $P \cup \{D\}$ qui satisfait C.

La procédure abductive de K&M suppose que la règle de sélection $\mathcal{R}$ ne sélectionne un littéral supposable que lorsque celui-ci ne contient plus de variables : si ceci n'est pas possible, la procédure échoue. Ceci est une des raisons pour lesquelles la procédure de K&M n'est pas complète. Cette restriction se comprend aisément lorsque le littéral supposable correspond à une négation (par échec) dans le programme initial P, cette condition est celle que l'on appelle *non-floundering query*[1].

1. *to flounder* signifie « patauger ».

Cette limitation de la procédure ne permet d'envisager son utilisation que pour une classe restreinte de programmes.

La procédure SLDNFA

Une autre procédure abductive, conçue comme une extension de la résolution SLDNF, est présenté dans (Denecker & De Schreye, 1992). Elle est utilisée en particulier dans le planificateur abductif CHICA (Missiaen, 1991). Contrairement à la procédure précédente, on ne dispose que d'un programme P et des symboles de prédicats supposables A.

Une caractéristique intéressante de cette procédure concerne le traitement des littéraux supposables qui contiennent des variables. Ceux-ci voient leurs variables remplacées par des constantes de skolem, ce qui est une manière de supposer l'existence d'individus ou d'objets du domaine satisfaisant cette hypothèse. Les algorithmes d'unification utilisés dans cette procédure sont étendus pour prendre en compte la possibilité pour ces constantes de skolem d'être égales à des constantes (ordinaires) de ce même domaine.

La définition de la procédure suppose trois ensembles: $\mathcal{PG}$ est l'ensemble des buts positifs, c'est à dire ceux que l'on veut voir réussir. Typiquement, au début de la procédure, $\mathcal{PG}$ contient le but initial. L'ensemble $\mathcal{NG}$ contient les buts négatifs que l'on veut voir échouer: ces buts servent à réaliser la négation par échec. Un troisième ensemble $\mathcal{NAG}$ joue un rôle analogue aux contraintes d'intégrité de la procédure précédente. Un dernier ensemble Δ collecte les hypothèses faites au cours de l'exécution de la procédure.

La procédure SLDNFA construit une séquence finie de quadruplets de la forme $\langle \mathcal{PG}_i, \mathcal{NG}_i, \mathcal{NAG}_i, \Delta_i \rangle$. Les buts positifs Q pris dans l'ensemble $\mathcal{PG}$ sont traités de manière analogue à ce qui est réalisé par la procédure de K&M: soit ils sont réécrits avec les règles du programme ou les hypothèses de Δ, soit le littéral sélectionné dans la clause est supposable et il est alors ajouté à Δ après avoir été skolémisé. Si un littéral négatif apparaît il est placé dans $\mathcal{NG}$. Les buts pris dans $\mathcal{NG}$ sont traités de manière analogue à ceux de $\mathcal{PG}$, sauf que l'usage d'une hypothèse déja effectuée (présente dans Δ_i) pour effacer un littéral est enregistré à l'aide de l'ensemble $\mathcal{NAG}$ afin que cette branche de la dérivation puisse être reprise plus tard si une nouvelle hypothèse avec le même symbole de prédicat permet de le faire. La procédure SLDNFA termine lorsque $\mathcal{PG}$ ne contient plus que la clause vide.

L'inconvénient de la procédure SLDNFA est de différencier deux cas de résolution suivant que l'on cherche un succès du but concerné (but pris dans l'ensemble $\mathcal{PG}$) ou que l'on recherche un échec (but pris dans $\mathcal{NG}$). Le traitement des buts positifs est analogue à ce que réalise une dérivation abductive telle que la définissent K&M, à la skolémisation des littéraux près. De même il y a une certaine analogie entre une dérivation de consistance et le traitement des buts négatifs par la procédure SLDNFA: les différences sont que SLDNFA autorise la sélection d'un littéral

supposable avec variables, ce qui n'est pas le cas de la procédure de K&M, et qu'une dérivation de consistance de K&M est capable de générer de nouvelles hypothèses permettant de produire un échec fini sur la clause courante.

3.4 Conclusions

Il ressort aussi de ce qui précède que l'abduction apparaît parfaitement réalisable lorsqu'on se limite au langage de la programmation logique. Les procédures décrites plus haut sont réellement utilisables pour peu que l'on puisse décrire la logique temporelle utilisée et le problème traité par un programme logique. A cet égard, le calcul d'événements décrit dans le chapitre 1 de ce mémoire est une solution intéressante, comme en témoignent les travaux sur l'abduction de K. Eshghi (Eshghi, 1988) et L. Missiaen (Missiaen, 1991).

Chapitre 4

Proposition d'une procédure abductive pour le raisonnement temporel

Dans ce chapitre, nous étudions le problème de l'abduction en raisonnement temporel, et nous proposons une procédure abductive. Du chapitre 3, il ressort que le cadre et le langage de la programmation logique permet la mise au point de procédures d'abduction simples. Dans la mesure où le chapitre 2 a permis de justifier l'usage d'un principe de résolution avec contraintes pour le raisonnement temporel, nous proposons dans ce chapitre une procédure abductive qui réalise la fusion de la programmation logique avec contraintes et de la programmation logique abductive.

4.1 Eléments d'une procédure abductive pour le raisonnement temporel

Dans le chapitre 3, nous avons vu que toutes les méthodes de génération d'hypothèses étaient basées sur une forme de résolution, linéaire ou SLD. Comme l'on a vu dans le chapitre 2 que l'on pouvait gagner en efficacité, d'abord sur les pas de résolution individuels, puis de manière générale dans le processus d'inférence, en utilisant la résolution avec contraintes, il est légitime d'imposer qu'une procédure abductive basée sur la résolution utilise ce principe de résolution avec contraintes, au besoin en l'adaptant à la forme de résolution qui est utilisée dans la procédure, par exemple la résolution SLD pour les procédures de programmation logique abductive.

Parmi toutes les procédures d'abduction que nous avons détaillées précédemment, lesquelles sont les plus adaptées à l'usage que l'on souhaite faire de l'abduction en raisonnement temporel ?

- Les procédures de Pople (1973) et de (Cox & Pietrzykowsky, 1986) sont utilisables en pratique dans la plupart des cas. Leurs inconvénient est de générer parfois des hypothèses qui sont des formules non closes arbitraires, ce qui peut devenir contradictoire avec l'usage de l'abduction pour la planification, où l'on souhaite plutôt des hypothèses « particulières » pour chaque occurrence d'action ou d'événement ;

- Les procédures de programmation logique abductive ont pour inconvénient de ne traiter qu'un sous ensemble de la logique du premier ordre, mais en contrepartie d'être plus simple à implanter et à mettre en œuvre ;

- De la même manière, utiliser le cadre de la programmation logique abductive et de la résolution avec contraintes permet *a priori* d'utiliser bon nombre de résultats (théoriques et pratiques) de la programmation logique avec contraintes, en particulier sur CLP($\mathcal{R}$) (Jaffar, Michaylov, Stuckey, & Yap, 1992b), qui se rapproche le plus de notre usage de la résolution avec des contraintes temporelles ;

- La procédure de (Kakas & Mancarella, 1990c) permet d'ajouter à la description du problème des contraintes d'intégrité, ce qui permet d'exclure des combinaisons d'hypothèses, ou d'hypothèses et d'autres conditions, et, par voie de conséquence, de réduire l'espace de recherche. En planification, cela correspond à restreindre dynamiquement les possibilités d'occurrence de certaines actions ;

- *A contrario* la procédure SLDNFA ne manipule pas de contraintes d'intégrité : ce qui se comprend car elle a été principalement utilisée avec le calcul d'événements, qui ne contient pas de telles clauses : l'usage de la négation par échec est ici suffisant. Si l'on souhaite être relativement indépendant du langage (formalisme) temporel utilisé, cela peut être un facteur limitant l'intérêt de SLDNFA.

Pour toutes ces raisons, nous avons donc choisi de développer une procédure abductive sur le modèle de celle de (Kakas & Mancarella, 1990c), et qui utilise la résolution avec contraintes. On peut voir notre travail comme une fusion entre la programmation logique abductive et la programmation logique avec contraintes, spécialisée ensuite pour le raisonnement temporel.

4.2 Description de la procédure

Avant de décrire notre procédure, il semble nécessaire de revenir sur la procédure abductive de Kakas et Mancarella (K&M) décrite dans (Kakas & Mancarella, 1990c), et seulement entrevue dans le chapitre 3. En effet, le principe général de cette procédure est identique à celle qui sera décrite ici, et la procédure de K&M est aussi plus simple que la nôtre, ce qui facilite donc la présentation.

Après avoir décrit et analysé cette procédure, nous étudierons comment celle-ci peut être adaptée au raisonnement temporel, en incorporant l'usage d'un principe de résolution avec contraintes.

4.2.1 La procédure originale de K&M : défauts et remèdes

Pour (Kakas & Mancarella, 1990c), un problème d'abduction est décrit par un triplet (P, IC, A) où P est un programme logique, IC est un ensemble de clauses sans tête appelées contraintes d'intégrité, et A est un ensemble de symboles de prédicats utilisés dans P et que l'on appelle *supposables*. Si $\leftarrow Q$ est un but, l'objet de la procédure est de trouver un ensemble d'hypothèses Δ (instances closes de prédicats supposables, notés avec le signe †) tel que

- $P \cup \Delta \models Q$, c'est à dire que Q est vrai dans tout modèle de $P \cup \Delta$;
- $P \cup \Delta \models IC$, c'est à dire que toutes les clauses de IC sont vraies dans tous les modèles de $P \cup \Delta$.

La notion de modèle pour un programme logique retenue par K&M est celle de modèle stable (Gelfond & Lifschitz, 1988). Satisfaire les contraintes d'intégrité est donc fortement lié à la sémantique associée aux programmes logiques.

Comme cela a été décrit dans le chapitre 3, K&M transforment préalablement le triplet (P, A, IC) en un nouveau triplet (P^*, A^*, IC^*) obtenu par réécriture des littéraux négatifs dans le corps des clauses. Tout littéral négatif $\sim a$ est remplacé par un littéral a^*, qui représente donc l'opposé de a, et a^* est ajouté à l'ensemble des littéraux supposables A. Cette transformation n'est pas strictement indispensable pour présenter la procédure, et nous ne l'appliquerons pas dans ce qui suit.

Sont donc considérés comme supposables tous les littéraux négatifs et les littéraux positifs référencés comme tels par l'ensemble A. On suppose implicitement que

l'ensemble des contraintes d'intégrité IC contient pour chaque littéral supposable a une contrainte de la forme $\leftarrow a, \sim a$. Quel que soit le signe d'un littéral, on garde la notation $\sim a$ pour désigner son opposé.

On suppose aussi que l'on se donne une règle de sélection des littéraux dans le corps d'une clause : cette règle est notée $\mathcal{R}$.

Définition formelle de la procédure de K&M

La procédure entrelace deux types de dérivation : le premier type s'appelle *dérivation abductive* et a pour objet de réécrire une clause but à l'aide des règles du programme, et d'identifier les littéraux susceptibles d'être supposés. Lorsqu'un de ces littéraux est rencontré, il n'est acceptable comme hypothèse que lorsqu'il est consistant. Conformément à la définition choisie, la consistance d'une hypothèse signifie que toutes les contraintes d'intégrité sont satisfaites dans un modèle du programme augmenté des hypothèses.

Définition 4.1 (dérivation abductive) *(Kakas & Mancarella, 1990c) Une* dérivation abductive *de* $\langle G_1, \Delta_1 \rangle$ *à* $\langle G_n, \Delta_n \rangle$ *est une séquence de paires* $\langle G_i, \Delta_i \rangle$ *telle que pour tout* i, G_i *est de la forme* $\leftarrow L_1, \ldots, L_k$, *le littéral* L_j *est sélectionné par* $\mathcal{R}$ *dans* G_i, Δ_i *est un ensemble de littéraux supposables et sans variables, et l'une des trois conditions suivantes est vérifiée :*

(A_1) L_j *est non supposable,* C *est un résolvant de* G_i *et d'une clause de* P *sur le littéral* L_j, *alors :*

$$G_{i+1} = C \qquad \Delta_{i+1} = \Delta_i$$

(A_2) L_j *est supposable,* $L_j \in \Delta_i$, *alors :*

$$G_{i+1} = \leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k \qquad \Delta_{i+1} = \Delta_i$$

(A_3) L_j *est supposable,* $L_j \notin \Delta_i$ *et* $\sim L_j \notin \Delta_i$, *il existe une dérivation de consistance de* $\langle \{L_j\}, \Delta_i \cup \{L_j\} \rangle$ *à* $\langle \{\}, \Delta' \rangle$, *alors :*

$$G_{i+1} = \leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k \qquad \Delta_{i+1} = \Delta'$$

Par rapport à la résolution SLDNF classique (ou encore PROLOG avec une règle de sélection des littéraux), le cas A_1 n'est qu'un pas de résolution classique pour réécrire le but courant G_i à l'aide d'une règle qui conclut sur le prédicat L_j ; le cas A_2 correspond au cas où L_j est déja présent dans l'ensemble des hypothèses, il est alors supposé vrai et effacé dans le but G_i ; et A_3 est appelé lorsqu'on désire faire l'hypothèse L_j. Il faut en particulier que ni L_j ni son opposé ne soient présents dans

Δ_i : L_j sera accepté comme hypothèse si le test de consistance réussit, autrement dit si il existe une dérivation de consistance à partir de ce littéral.

Définition 4.2 (dérivation de consistance) *(Kakas & Mancarella, 1990c) Une* dérivation de consistance *pour un littéral supposable α est une séquence de paires $\langle\{\alpha\}, \Delta_0\rangle, \langle F_1, \Delta_1\rangle, \ldots, \langle F_n, \Delta_n\rangle$ telle que :*

(i) F_1 *est l'union de tous les buts obtenus en résolvant la clause $\alpha \leftarrow$ avec les contraintes d'intégrité, et $\Delta_1 = \Delta_0$;*

(ii) pour $i \geq 1$, l'ensemble F_i est de la forme $\{\leftarrow L_1, \ldots, L_k\} \cup F_i'$ et il existe $j \in [1, k]$ tel que l'on obtient $\langle F_{i+1}, \Delta_{i+1}\rangle$ par une des quatre conditions suivantes :

(C_1) L_j est non supposable, C' est l'ensemble de tous les résolvants du but $\leftarrow L_1, \ldots, L_k$ avec les clauses de P sur le littéral L_j, la clause vide $\square$ n'appartient pas à C', et on a :

$$F_{i+1} = C' \cup F_i \qquad \Delta_{i+1} = \Delta_i$$

(C_2) L_j est supposable, $L_j \in \Delta_i$, $k > 1$, alors :

$$F_{i+1} = \{\leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k\} \cup F_i' \qquad \Delta_{i+1} = \Delta_i$$

(C_3) L_j est supposable, l'opposé de L_j appartient à Δ_i, alors :

$$F_{i+1} = F_i' \qquad \Delta_{i+1} = \Delta_i$$

(C_4) L_j est supposable, ni L_j ni son opposé n'appartiennent à Δ_i, et il existe une réfutation abductive de $\langle \leftarrow\!\sim L_j, \Delta_i\rangle$ à $\langle \square, \Delta'\rangle$, alors :

$$F_{i+1} = F_i' \qquad \Delta_{i+1} = \Delta'$$

Le but d'une dérivation de consistance est de vérifier qu'un littéral clos α peut être ajouté aux hypothèses déja faites. Pour ceci, il faut que les contraintes d'intégrité soient vérifiees : c'est l'objet du passage de F_0 à F_1. Pour le littéral α dont on souhaite faire une hypothèse, c'est à dire le considérer vrai, une contrainte d'intégrité $\leftarrow \alpha, L_1, \ldots, L_n$ sera satisfaite dès lors qu'un des littéraux L_i ne sera pas vrai, ce qui signifie qu'il sera non prouvable : on cherche donc l'échec du but $\leftarrow L_1, \ldots, L_n$. Cette technique est reprise de celle développée par (Sadri & Kowalski, 1988), pour vérifier la satisfaction de contraintes d'intégrité lors d'opérations sur une base de données déductive.

Une fois acquis le principe que F_i représente un ensemble de buts que l'on veut voir échouer, tous les cas de la dérivation sont des moyens d'y arriver. Un échec sur une clause signifie que cette clause est enlevée de l'ensemble F_i.

Dans le cas où α représente une hypothèse négative $\sim a$, le passage de F_0 à F_1 a pour effet que F_1 contient alors au moins le but $\leftarrow a$, car IC contient toujours

au moins une clause $\leftarrow a, \sim a$. Ceci donne le même comportement que la résolution SLDNF.

Exemple 4.1 *Cette procédure permet alors de faire une dérivation comme celle de la figure 4.1 correspondant au programme suivant:*

$$P = \left\{ \begin{array}{l} p \leftarrow q, s \\ q \leftarrow \sim r \\ r \leftarrow l \end{array} \right. \qquad IC = \left\{ \begin{array}{l} \leftarrow s, r \\ \leftarrow r, \sim r \\ \leftarrow s, \sim s \\ \leftarrow l, \sim l \end{array} \right\} \qquad A = \{s, l, \sim r, \sim s, \sim l\}$$

et à la requête $\leftarrow p$.

La dérivation commence par le but $\leftarrow p$ *avec un ensemble d'hypothèses* Δ *vide. Ce but est réecrit à l'aide des règles du programme (cas* A_1 *de la définition de la dérivation abductive), ce qui permet d'obtenir* $\leftarrow s, \sim r$. *Le littéral* s *est sélectionné dans cette clause, est supposable et l'on se trouve donc dans le cas* A_3 *: on commence une dérivation de consistance à partir de* s *comme hypothèse. La résolution de* s *avec les contraintes d'intégrité produit les clauses* $\leftarrow\sim s$ *et* $\leftarrow r$ *qui correspondent à l'ensemble* F_1 *tel qu'il est défini dans la définition de la dérivation de consistance. La première clause de cet ensemble est traité grâce au cas* C_3, *c'est à dire que l'opposé de* $\sim s$ *appartient à* Δ. *La deuxième clause* $\leftarrow r$ *est réécrite (cas* C_1*) et produit comme résolvant* $\leftarrow l$. *Le littéral* l *est supposable, et le cas* C_4 *s'applique. La réfutation abductive qui s'effectue alors cherche à prouver* $\sim l$, *ce qui est possible si* $\sim l$ *est supposé comme hypothèse: c'est ce qui est vérifié à l'aide d'une autre dérivation de consistance démarrée à partir de la réfutation imbriquée au niveau 3. L'ensemble* Δ *construit jusqu'alors contient les hypothèses* $\{s, \sim l\}$, *et la première dérivation abductive reprend avec cet ensemble d'hypothèses. Le littéral qui reste est* $\sim r$, *supposable, et pour lequel est construit une dérivation de consistance. Cette dernière se termine (par le cas* C_3*) puisque* $\sim l$ *est déja présent dans* Δ. *A la fin de la dérivation initiale, l'on obtient une clause vide qui indique le succès de cette dérivation.*

L'ensemble d'hypothèses obtenu $\{s, \sim l, \sim r\}$ *indique donc que l'ensemble* $\{s\}$ *est une explication de* p. *Compte tenu de la définition retenue par K&M, cela signifie qu'il existe un modèle stable de* $P \cup \{s\}$, *ce modèle étant* $M = \{p, q, s\}$, *et qui satisfait toutes les contraintes de* IC.

Pour récapituler, les points particuliers de la procédure abductive de K&M sont les suivants:

- les hypothèses sont générées lorsqu'elles sont nécessaires pour prolonger une réfutation;
- le test de consistance des hypothèses est un test par lequel on vérifie que les contraintes d'intégrité sont toujours satisfaites;

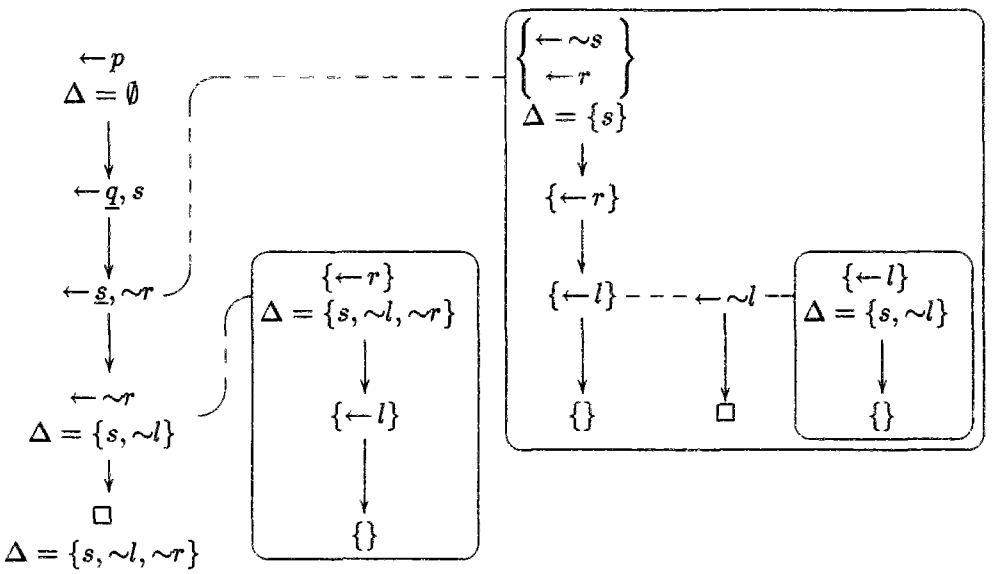


FIG. 4.1 - *Une dérivation pour l'exemple 4.1. Les dérivations de consistance sont encadrées, avec leurs éventuelles réfutations abductives imbriquées.*

- ce dernier test se fait par une recherche exhaustive de l'arbre de dérivation des buts générés à partir de l'hypothèse et des contraintes d'intégrité.

De la même manière, l'on peut considérer que les cas de la dérivation abductive sont des actions par lesquelles on peut faire réussir une réfutation, alors que les différents cas d'une dérivation de consistance représentent différentes possibilités de faire échouer une réfutation. Cette distinction sera utilisée pour étendre la procédure et l'adapter à la résolution avec contraintes.

Défauts de la procédure

La principale restriction est que la règle de sélection $\mathcal{R}$ ne doit sélectionner un littéral supposable que si il est sans variables, et ceci pour les cas A_2, A_3, C_2, C_3 et C_4. Ceci se comprend par analogie avec la résolution SLDNF où la même restriction est nécessaire, car la résolution SLD ne peut pas répondre à des requêtes universellement quantifiées.

Dans le cas de programmes logiques en premier ordre, cette limite sur le choix des littéraux supposables, couplée avec le fait, malgré tout intéressant, que cette procédure soit « paramètrable » par l'ensemble des littéraux supposables, restreint fortement les programmes logiques pour lesquelles cette procédure est utilisable. En pratique, il apparaît donc que cette procédure est très limitée, en dehors du cas propositionnel.

Un autre défaut est que cette procédure reste dans le cadre de la programmation logique pure. En particulier, compte tenu de l'usage prévu de cette procédure pour le raisonnement temporel, il sera nécessaire de l'adapter à l'usage de la résolution SLD avec contraintes.

Quelques remèdes

Si le principal défaut de la procédure de K&M concerne la sélection de littéraux supposables clos, il existe plusieurs moyens de remédier à ceci. Dans le cas où les littéraux supposables correspondent à des hypothèses négatives, on peut appliquer une méthode comme la négation constructive de (Chan, 1988). Cette méthode se base sur le fait qu'effectuer une réfutation SLD à partir d'un but contenant des variables permet d'obtenir une ou plusieurs substitutions pour ces variables. Sous certaines conditions, il est possible d'identifier comme équivalents, par rapport à une sémantique donnée, la disjonction de toutes ces substitutions (des égalites) et la requête initiale. Par exemple, si P est le programme suivant:

(1) $edge(a,b) \leftarrow .$
(2) $edge(c,d) \leftarrow .$
(3) $edge(d,c) \leftarrow .$
(4) $reachable(a) \leftarrow .$
(5) $reachable(X) \leftarrow reachable(Y), edge(Y,X).$
(6) $unreachable(X) \leftarrow \sim reachable(X).$

et la requête est $\leftarrow unreachable(X)$, c'est à dire qu'on cherche des valeurs pour X telles que $unreachable(X)$ est vérifié, alors la résolution SLDNF classique ne peut pas trouver de solution. En se fondant sur le fait que la requête $\leftarrow reachable(X)$ obtient deux réponses, la première avec $X = a$, et la deuxième avec $X = b$, la négation constructive permet de fournir $\neg(X = a \vee X = b)$, c'est à dire $X \neq a \wedge X \neq b$, comme réponse à la requête $\leftarrow unreachable(X)$. Parmi les précautions à prendre, il faut donc s'assurer que l'équivalence $reachable(X) \Leftrightarrow (X = a \vee X = b)$ est vérifiée, ce qui de manière générale suppose que pour toute requête l'on ait la possibilité de la « transformer » (par une réfutation SLD par exemple) en une disjonction d'égalités qui lui soit équivalente, relativement à la sémantique choisie (Przymusinski, 1989a). Ceci est souvent lié à l'absence de branches infinies dans l'arbre de dérivation construit à partir de la requête.

Ceci peut donc permettre en partie de lever la restriction sur l'absence de variables dans les littéraux supposables. Néanmoins, la négation constructive a été étudiée dans le cadre de la programmation logique classique et non abductive. L'usage d'un tel système est donc assez immédiat pour un littéral négatif (donc supposable) sélectionné dans une dérivation abductive. Mais son usage pour les hypothèses positives n'est pas immédiat. Nous proposerons plus loin une solution dans le cadre de la programmation logique avec contraintes qui sera applicable à toutes les formes d'hypothèses: l'intérêt de la résolution avec contraintes est de substituer à l'unification un mécanisme de résolution de contraintes, plus exactement de test de la

consistance, grâce auquel l'on dispose d'une plus grande latitude pour implanter un système analogue et générer des conditions sur les variables.

Sans lien immédiat avec la négation constructive, une autre solution a été étudiée, en particulier dans (Missiaen, 1991) et (Denecker & De Schreye, 1992), qui consiste à remplacer chaque variable d'un littéral candidat à être supposé par une nouvelle constante de skolem. L'idée est de représenter par de telles constantes des éléments du domaine, sans préjuger de leur véritable identité, et dont on ne connait que le fait qu'ils satisfont l'hypothèse considérée. Comme le terme de skolem introduit est factice, au sens où il ne correspond en particulier à aucun élément du domaine, mais peut *a priori* être égal à n'importe lequel de ces éléments, il faut s'assurer qu'il ne puisse pas être identifié à un élément pour lequel l'hypothèse deviendrait contradictoire. La procédure SLDNFA de (Denecker & De Schreye, 1992) utilise des algorithmes d'unification modifiés qui traitent ces constantes de skolem comme des variables, et qui autorise donc une égalité entre une constante de skolem et un autre terme.

Une extension proposée par (Kakas & Mancarella, 1993) sur leur procédure originale, utilise une technique appelée *homogénéisation* qui consiste à remplacer les termes de skolem par des variables et à rajouter au corps de la clause des égalités entre ces variables et les termes qu'elles remplacent. En utilisant l'unification classique, on génère ainsi des égalités faisant intervenir les constantes de skolem et auxquelles on peut appliquer le même traitement que dans la négation constructive. Une fois une constante de skolem liée à un élément du domaine, K&M reprennent une dérivation de consistance pour cette hypothèse ainsi instanciée. Ce processus peut devenir très coûteux, et il serait souhaitable qu'une dérivation de consistance produise et retourne des contraintes sur les constantes de skolem, de telle manière que la consistance d'une hypothèse obtenue par substitution d'une constante de skolem puisse être vérifiable simplement en testant la consistance de l'égalité entre la constante de skolem et le terme qui lui est affecté.

En fait, l'extension de la programmation logique abductive par l'usage de la résolution avec contraintes, ce que l'on peut encore décrire comme l'intégration de la programmation logique abductive et de la programmation logique avec contraintes, permet de résoudre tous ces problèmes de manière assez simple. Ceci fait l'objet du paragraphe suivant.

4.2.2 Extension à la programmation logique avec contraintes

Notre but est maintenant d'étendre la procédure de K&M à l'usage de la résolution avec contraintes, de la même manière que pour la programmation logique avec contraintes. Ensuite, nous souhaitons apporter une solution aux deux problèmes suivants :

1. accepter les hypothèses avec variables, que ces hypothèses soient positives ou négatives ;

2. assurer la consistance de ces hypothèses quelles que soient les valeurs que pourraient prendre par après (dans la suite de la dérivation) les variables de ces hypothèses ;

3. assurer la consistance de ces hypothèses quelles que soient les hypothèses faites ultérieurement.

Chacun de ces points va faire l'objet d'un paragraphe. En réalité, comme nous le verrons, chaque choix a des conséquences sur les problèmes qui se posent pour un autre point : les trois prochains paragraphes contiendront parfois des références en avant qui seront résolues par l'exposé de la solution adoptée pour un autre point.

Usage de la résolution avec contraintes

Dans tous les cas de la procédure où l'on calcule un résolvant, l'usage de la résolution avec contraintes est immédiat. Les clauses buts sont notées $\leftarrow B \parallel C$ où B est un ensemble de littéraux (à interpréter logiquement comme une conjonction), et C une conjonction de contraintes (temporelles dans notre cas). Les règles du programme sont de la forme $H \leftarrow B \parallel C$, où H est un atome (littéral positif) et B est le corps de la clause (comme précédemment un ensemble de littéraux). La notion de résolvant utilisée dans la définition de K&M est remplacée par celle de résolvant au sens de la programmation logique avec contraintes. Si G est la clause but

$$\leftarrow P(x_1, \ldots, x_n), B \parallel R$$

et qu'il existe dans le programme P une clause C de la forme

$$P(y_1, \ldots, y_n) \leftarrow A \parallel S$$

qui soit éventuellement renommée pour ne pas partager de variables avec la clause but précédente, alors le résolvant de G et de C est défini lorsque $R \wedge S \wedge \Gamma$ est consistant, avec $\Gamma = \{x_1 = y_1 \wedge \ldots \wedge x_n = y_n\}$, et ce résolvant s'écrit alors :

$$\leftarrow A, B \parallel R \wedge S \wedge \Gamma$$

Le cas A_1 de la dérivation abductive, où le but courant est réécrit à l'aide d'une clause du programme est immédiatement étendu pour utiliser cette notion de résolvant. Le cas A_2, une fois sélectionné le littéral L (sans variables), utilise une hypothèse déjà présente dans Δ pour effacer L du but courant. Ce cas peut être étendu à un littéral L supposable, sans contraintes sur ses variables, et pour lequel le but courant est remplacé par le résolvant entre ce but et une clause unitaire formée d'un littéral pris dans Δ. Le cas A_3 est plus complexe et sera traité dans le paragraphe suivant concernant la présence de variables dans les hypothèses.

Dans la dérivation de consistance, l'usage de la résolution avec contraintes pour calculer les résolvants (dans les cas (i) et C_1) est aussi simple que précédemment.

Plus complexes sont certains cas de la dérivation de consistance. En premier lieu, le cas C_2 peut être généralisé comme pour le cas A_2, en admettant la présence de variables dans le littéral sélectionné et en calculant non pas un seul résolvant, mais l'ensemble de tous les résolvants entre la clause choisie et les hypothèses de Δ. Ceci permet de garder l'esprit de la dérivation de consistance qui est de développer en parallèle plusieurs branches de dérivation.

Hypothèses avec variables

En autorisant dans les cas A_2 et C_2 la sélection d'un littéral supposable ayant éventuellement des variables, nous avons fait un pas vers une solution au premier problème. Néanmoins, l'intérêt réel d'autoriser des variables dans les littéraux supposables concerne surtout le cas A_3, puisque c'est à ce moment là que les hypothèses sont générées et que leur consistance est testée. Pour A_3, il y deux possibilités :

1. soit l'on garde le littéral supposable avec ses variables, par exemple $p(X)$, ce qui signifie que l'hypothèse a pour signification $\exists X\ p(X)$;
2. soit l'on remplace toutes les variables du littéral par des constantes de skolem non encore utilisées, comme dans (Denecker & De Schreye, 1992) ;

Dans le premier cas, la dérivation de consistance, dont le but est de vérifier la satisfaction de toutes les contraintes d'intégrité de la forme $\leftarrow p(Y), B[Z,Y] \parallel C$ sous l'hypothèse $\exists X\ p(X)$, commence par l'ensemble $F_0 = \{\exists X\ p(X)\}$. Le calcul de F_1 fait intervenir les contraintes d'intégrité et F_1 contient entre autres $\leftarrow\sim p(X)$ et $\leftarrow B[Z,Y] \parallel C \wedge (X = Y)$ qui ne doivent donc pas avoir de solutions. L'inconvénient de cette solution est que la variable X de l'hypothèse reste anonyme, c'est à dire qu'il serait nécessaire de garder tout au long de la dérivation de consistance l'indication que X est une variable existentiellement quantifiée dans une hypothèse. De plus, la clause $\leftarrow\sim p(X)$ de F_1 doit être interprétée comme $\exists X\ \leftarrow\sim p(X)$, ce qui est plus faible que l'interprétation habituelle d'une telle clause par $\forall X\ \leftarrow\sim p(X)$. De ce fait, il est difficile d'implanter l'équivalent du cas C_3 de la dérivation de consistance : ce cas traite de la présence de $\sim p$ dans la clause courante, lorsque p est dans Δ, ce qui est un moyen d'éliminer immédiatement cette clause car $\sim p$ ne peut alors pas être prouvé. Si l'on garde les variables dans les hypothèses de Δ, ce mécanisme doit identifier les variables dans l'hypothèse et dans la clause, et ceci en tenant compte de tous les éventuels renommages ayant pu intervenir dans les pas de résolution antérieurs.

La deuxième solution remplace X par une constante de skolem sk et commence une dérivation de consistance avec $F_0 = \{p(sk)\}$. Skolémiser $p(X)$ est une manière de dire $\exists X\ p(X)$ en disposant d'un nom pour désigner X. L'intérêt de cette solution est que l'on peut voir l'hypothèse comme sans variables, et que la constante introduite possède un statut qui permet de la différencier à la fois des variables et des autres constantes ordinaires.

Un schéma de négation constructive

Dans tous les cas de figure, que l'on garde des variables ou que l'on crée des contantes de skolem, il est nécessaire de se prémunir contre un cas d'échec de la dérivation de consistance par occurrence d'une clause vide dans un des ensembles F_i. En réalité, avec la résolution avec contraintes, une dérivation qui finit par une clause vide $\square \parallel R$ n'est pas synonyme d'une preuve de la requête initiale : obtenir une preuve demande au minimum que R soit consistant, et peut éventuellement demander la dérivation de plusieurs clauses vides distinctes. L'idée est que si l'on ajoute des contraintes telles que R devient inconsistante, alors l'on garde la possibilité d'un succès de la dérivation de consistance. Cette manière de forcer l'échec d'une branche de la dérivation est un cas qui vient enrichir les quatre cas déja prévus par la définition de K&M.

Dans un premier temps, la réalisation la plus simple d'un tel schéma est que toute clause vide $\square \parallel R$ avec des contraintes R qui sont consistantes, et apparaissant dans les clauses d'une dérivation de consistance (ensemble F_i) oblige à faire une hypothèse telle que R n'a pas de solution (est insatisfaisable). Ainsi, l'on transforme un succès potentiel en un échec certain pour une branche de l'arbre de refutation.

Plus généralement, ce schéma peut être appliqué à toutes les clauses d'une dérivation de consistance. D'après (Stuckey, 1991a), si P est un programme logique avec contraintes sur un domaine $\mathcal{A}$, et $G = \leftarrow Q$ un but, alors si l'on considère l'arbre de dérivation de racine G pour une règle de sélection donnée, on définit une *frontière* de cet arbre comme étant un ensemble fini de nœuds (de cet arbre), tel que tout chemin depuis G dans l'arbre est soit fini et conduit à un échec, soit passe une et une seule fois par un nœud de la frontière.

Par exemple, si l'arbre de dérivation d'un but G obtenu en suivant une règle de sélection donnée est celui de la figure 4.2, alors les ensembles $\{G_1, G_2, G_3\}$, $\{G_4, G_5, G_2, G_7\}$ et $\{G_4, G_6, G_3\}$ sont des frontières de cet arbre.

Le résultat apporté par Stuckey est le suivant. Si $\{G_1 = \leftarrow Q_1, \ldots, G_n = \leftarrow Q_n\}$ est une frontière d'un arbre de dérivation d'un but $G = \leftarrow Q$, alors l'équivalence suivante est vérifiée :

$$\mathcal{A}, P^* \models_3 (Q \Leftrightarrow (\exists Y_1\, Q_1) \vee \ldots \vee (\exists Y_n\, Q_n))$$

où Y_i désigne les variables de Q_i qui n'apparaissent pas dans Q. Ce résultat est valable pour une relation de conséquence trivaluée ($\models_3$) et l'équivalence ($\Leftrightarrow$) est satisfaite dès lors que les deux formules en relation ont la même valeur de vérité (parmi Vrai, Faux ou Indéfini). Le symbole P^* désigne la complétion du programme P, c'est à dire que pour chaque prédicat p défini par un ensemble de règles

$$\begin{array}{c} p(t_1) \leftarrow B_1 \parallel c_1 \\ \vdots \\ p(t_n) \leftarrow B_n \parallel c_n \end{array}$$

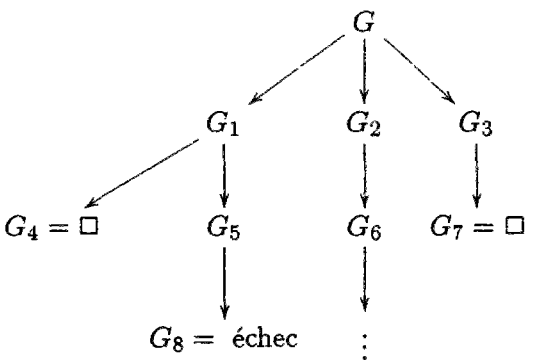


FIG. 4.2 - *Exemple d'un arbre de dérivation. Les ensembles de formules étiquetant les nœuds* $\{G_1, G_2, G_3\}$, $\{G_4, G_5, G_2, G_7\}$ *et* $\{G_4, G_6, G_3\}$ *sont des* frontières *de cet arbre de dérivation (Stuckey, 1991a).*

la complétion P^* contient la formule

$$\forall X\ p(X) \Leftrightarrow \begin{cases} \exists\, Y_1\ (x = t_1\ \wedge\ c_1\ \wedge\ B_1) & \vee \\ \qquad\qquad \vdots & \\ \exists\, Y_n\ (x = t_n\ \wedge\ c_n\ \wedge\ B_n) & \end{cases}$$

où x est une nouvelle variable, et Y_i désigne les variables de la $i^{\text{ième}}$ clause définissant p. Si un symbole p n'est défini par aucune clause de P, alors P^* contient $\forall x\ \neg p(X)$.

L'intérêt de ce résultat est d'établir un cadre général pour la négation constructive en programmation logique avec contraintes. Si dans le cours d'une dérivation, un but négatif de la forme $\leftarrow\sim G(Z) \parallel c$ est rencontré, une deuxième dérivation de $\leftarrow G(Z) \parallel c'$, où c' est un sous ensemble des contraintes c, est effectuée jusqu'à atteindre une frontière

$$\{\ (\leftarrow B_1 \parallel c' \wedge c_1\ ),\ \ldots\ ,(\leftarrow B_n \parallel c' \wedge c_n\ )\ \}$$

Les formules de cette frontière sont regroupées en une disjonction

$$(\exists Z, Y_1\ B_1 \wedge c_1)\ \vee \ldots \vee\ (\exists Z, Y_n\ B_n \wedge c_n)$$

où Y_i dénote l'ensemble des variables de $B_i \wedge c_i$ qui n'apparaissent pas dans le but initial $\leftarrow\sim G(Z) \parallel c$. On prend ensuite la négation de cette formule que l'on convertit en une disjonction de buts qui seront utilisés pour prolonger la dérivation initiale dans autant de branches. Cette conversion depuis les formes $\neg(\exists Y_i\ B_i \wedge c_i)$ s'effectue en séparant les contraintes de manière à obtenir la formule

$$(\neg\exists Y_i\ c_i) \vee (\neg\exists Y_i\ B_i \wedge c_i)$$

qui est équivalente à la précédente. Toutes ces formules sont rassemblées en une conjonction dont on prend ensuite la forme normale disjonctive.

Ce schéma est très général, presque trop pour une application pratique, puisqu'il laisse la possibilité du choix de c', c'est à dire du sous ensembles des contraintes passées à la sous-dérivation, et de la frontière atteinte dans cette sous-dérivation. Pour une utilisation pratique de ce schéma, nous remarquons qu'il existe une frontière particulière qui contient entre autres toutes les clauses vides de l'arbre. En ce limitant à cette frontière, l'on obtient un ensemble de formules où ne subsistent que des contraintes : en fait, une conjonction de contraintes dans chaque formule de la frontière. En réalité, si l'arbre de dérivation d'un but ne contient aucune branche infinie, l'ensemble de ses feuilles étiquetées par une clause vide est une frontière.

Le schéma de « négation constructive » que nous proposons est le suivant : toutes les fois qu'une clause vide $\square \parallel R$ apparaît dans un des ensembles F_i d'une dérivation de consistance, l'on choisit dans R une contrainte sans variables dont on ajoute la négation à l'ensemble des hypothèses. De cette manière, l'ensemble des hypothèses qui sont des contraintes sont vraies dans certains modèles de la théorie de restriction, modèles dans lesquels la restriction de cette clause vide n'a pas de solution.

A la lumière de l'étude de (Stuckey, 1991a), ce principe est une restriction de celui qu'il propose avec les différences suivantes :

- il ne tient pas compte des branches infinies qui pourraient apparaître dans certaines dérivations : ce qui limite son usage à des programmes logiques localement stratifiés ;
- il ne prend pas en considération toutes les contraintes associées à une clause vide : en réalité, le schéma de Stuckey produit une disjonction de contraintes. Dans notre cas, comme l'ensemble des hypothèses est logiquement une conjonction, cette disjonction de contraintes correspond à autant de points de choix conduisant chacun vers une solution distincte de la dérivation de consistance.

Une dernière remarque sur un tel principe de négation constructive est qu'il s'applique bien sous une hypothèse de non fermeture du domaine. C'est à dire que lorsqu'une contrainte comme $sk \neq a$ est supposée, où sk est une constante de skolem et a une constante ordinaire, le domaine d'interprétation de cette contrainte doit avoir d'autres éléments que a pour que cette contrainte soit consistante (puisqu'elle correspond en fait à $\exists X\ X \neq a$. Cette hypothèse est bien sûr valable pour le raisonnement temporel où les points de temps sont par exemple les nombres réels ou rationels.

Maintenir la consistance des hypothèses

A ce point de notre exposé, nous avons adopté les solutions suivantes :

- la résolution avec contraintes est utilisée pour tout calcul du résolvant entre deux clauses ;

- il est possible de sélectionner dans une dérivation abductive un littéral avec des variables. Dans ce cas, ces variables sont transformées en constantes de skolem avant de faire l'hypothèse et de tester sa consistance ;
- lorsque l'on teste la consistance d'un ensemble de contraintes, ces constantes de skolem ont le même statut que des variables, par contre, si l'on sélectionne un littéral, elles comptent pour des constantes ;
- un schéma analogue à la négation constructive est mis en œuvre dans la dérivation de consistance pour forcer l'échec d'une branche d'une dérivation si celle-ci finit sur une clause vide.

Nous allons maintenant étudier une solution au deuxième problème mentionné précédemment, à savoir conserver (ou rétablir) la consistance des hypothèses lorsque d'autres hypothèses sont rajoutées ou que des contraintes additionnelles peuvent forcer une constante de skolem à prendre une valeur qui invalide une hypothèse dont la consistance a été précédemment testée.

Dans le premier cas, c'est à dire lorsque des contraintes additionnelles invalident des dérivations de consistance déja effectuées, il est donc nécessaire d'identifier de manière exhaustive toutes les contraintes pesant sur les constantes de skolem et ayant été nécessaires au bon déroulement (succès) des dérivations de consistance. L'exemple qui suit est typique de ce problème.

Exemple 4.2 *Soit P le programme suivant :*

$$p(X) \leftarrow \sim q(X).$$
$$q(X) \leftarrow \sim r(X).$$
$$r(a).$$

pour lequel la requête $\leftarrow p(X), \sim r(X)$ produit la réfutation abductive de la figure 4.3. Dans cette réfutation, on suppose que l'on utilise la résolution avec contraintes et que les constantes de skolem sont considérées comme des variables lorsqu'il s'agit de déterminer la consistance d'une égalité. La première hypothèse $\sim q(sk)$ est consistante si $sk = a$ est vérifiée car c'est la seule condition pour que $r(sk)$ soit vrai. L'inconvénient est que cette hypothèse reste locale à la réfutation abductive interne à la première dérivation de consistance, et que cette condition est contradictoire avec la deuxième hypothèse $\sim r(sk)$ pour laquelle il est nécessaire que sk soit différent de a. En effet, l'hypothèse $\sim r(sk)$ signifie en réalité $\exists X \neg r(X)$, ou encore $\neg(\forall X\ r(X))$. Or $r(a)$ est vérifiée, il faut donc qu'il existe au moins un autre élément, et différent de a, pour qui r ne soit pas satisfait.

Cet exemple montre qu'il est nécessaire d'être exhaustif pour identifier au cours de la dérivation de consistance toutes les contraintes pesant sur les constantes de skolem et qui conditionnent le succès de la dérivation. De cette manière, vérifier la consistance à travers plusieurs dérivations de consistance successives se ramène à vérifier la consistance d'un ensemble de contraintes.

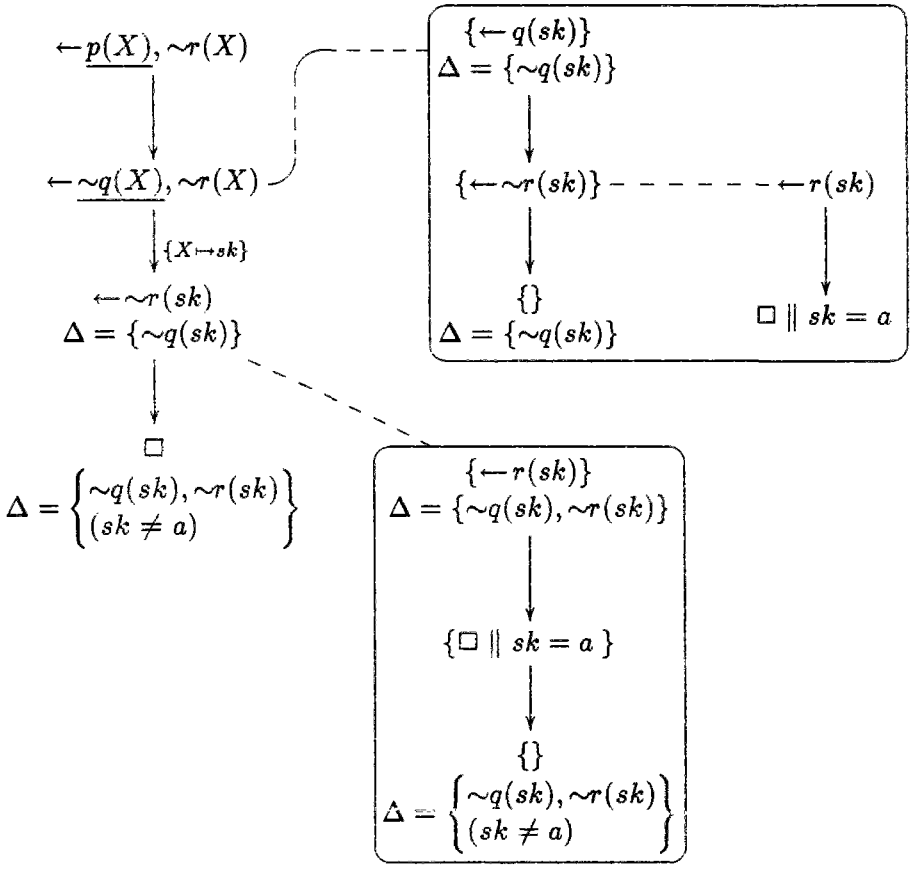


FIG. 4.3 - *Exemple d'une dérivation qui est incorrecte par manque de cohérence entre les hypothèses nécessaires pour valider une hypothèse ($\sim q(sk)$) et les contraintes qui portent sur la constante de skolem sk en fin de la dérivation.*

Une correction nécessaire pour la procédure est que lorsqu'une dérivation abductive interne à une dérivation de consistance rencontre un littéral supposable, les contraintes obtenues en fin de cette dérivation soient ajoutées comme hypothèses. De cette manière, si d'autres contraintes rajoutées ultérieurement imposent des valeurs incompatibles pour les constantes de skolem, ces contraintes seront tous simplement inconsistantes avec les hypothèses précédemment enregistrées: comme il s'agit de contraintes, l'on dispose bien sûr d'un algorithme pour décider de cette consistance, ce qui reporte donc sur cet algorithme le soin de vérifier la consistance des valeurs prises par les constantes de skolem au cours de la dérivation.

Il existe un deuxième forme de consistance à maintenir: lorsque l'on génère de nouvelles hypothèses, il est possible que celles-ci modifient des dérivations de consistances précédemment effectuées. Ceci se passe essentiellement par la possibilité de continuer une dérivation de consistance antérieure à partir d'un de ses points où l'échec en l'absence de résolvant permet d'éliminer des clauses, c'est à dire les cas C_1, C_2 et C_3.

Exemple 4.3 *Soit le programme P et les contraintes d'intégrité IC suivantes:*

$$P = \left\{ \begin{array}{l} p(X,Y) \leftarrow s(X), q(X,Y), r(Y). \\ q(a,b) \leftarrow . \\ q(X,c) \leftarrow r(X) \parallel X \neq a\,. \end{array} \right\}$$

$$IC = \left\{ \begin{array}{l} \leftarrow s(X), q(Y,Z) \parallel (Y \neq X) \wedge (Z \neq X)\,. \\ \leftarrow r(X), q(Y,Z) \parallel (Y \neq X) \wedge (Z \neq X)\,. \end{array} \right\}$$

pour lequel on considère les prédicats s et r comme supposables. Si l'on applique la procédure à la requête $\leftarrow p(a,b)$, l'on obtient une réponse $\Delta = \{s(a), r(b)\}$ conformément à la figure 4.4. Suivant la définition adoptée jusqu'à présent, la première dérivation de consistance réussit par absence de résolvant entre la clause $\leftarrow r(Y) \parallel Y \neq a$ et les littéraux de Δ puisque $r(Y)$ est un littéral supposable, et qu'il n'existe pas d'hypothèse en r dans Δ. Néanmoins, lorsque la deuxième dérivation de consistance, qui vérifie l'hypothèse $r(b)$ se produit, elle réussit aussi car le résolvant entre la clause $\leftarrow r(Y) \parallel (Y \neq b) \wedge (Y \neq a)$ et l'hypothèse $r(b)$ a une restriction qui est inconsistante.

Le problème est que l'ensemble de cette dérivation est incorrect. Si l'on écrit le programme P comme un programme propositionnel, et que l'on cherche le plus petit modèle de $P \cup \{s(a), r(b)\}$, l'on trouve l'ensemble de littéraux

$$\{p(a,b), s(a), r(b), q(a,b), q(b,c)\}$$

qui viole une des contraintes d'intégrité. En réalité, il n'y a donc pas d'explication à la requête $\leftarrow p(a,b)$ qui satisfasse les contraintes d'intégrité.

La correction de ce défaut se fait en gardant une trace des clauses sur lesquelles un échec est enregistré à un moment donné dans une dérivation de consistance. Ces

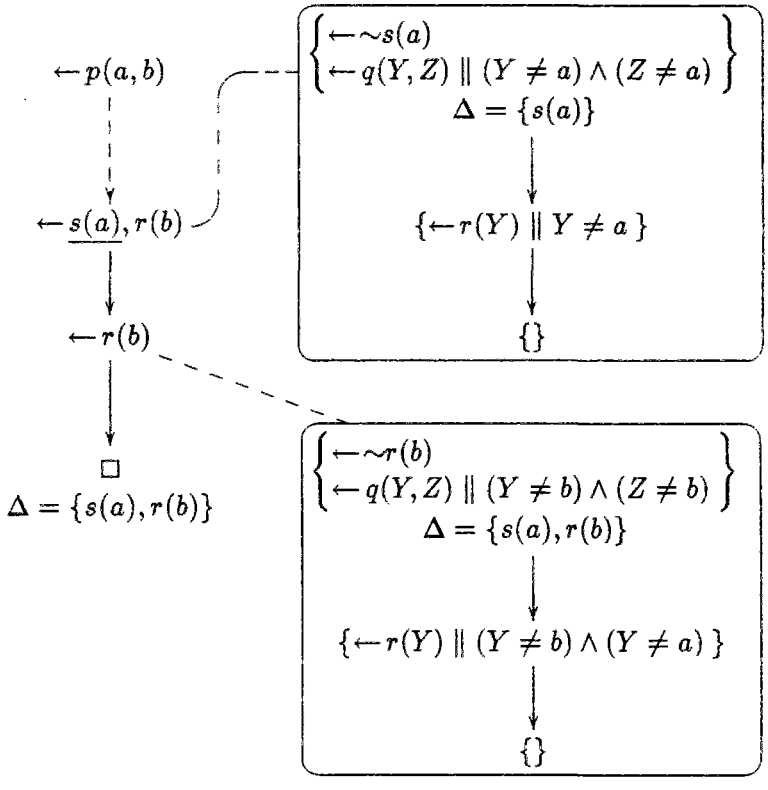


FIG. 4.4 - *Exemple de dérivation incorrecte par invalidation d'une dérivation de consistance par une hypothèse subséquente.*

clauses ont alors la même fonction que les contraintes d'intégrité et l'on peut les garder dans cet ensemble. Les passages $F_0 \rightarrow F_1$ ultérieurs prendront ces clauses et génèreront éventuellement des buts qui assureront la continuation des dérivations antérieures. Ceci doit donc se faire dans le cas C_1 si l'ensemble des résolvants est vide, et que la clause contient au moins un littéral supposable (qui est donc distinct de celui qui a été sélectionné). Dans le cas C_2, il faut garder cette trace de manière systématique, car rien n'empêche deux hypothèses de même symbole de prédicat. Dans le cas C_3, la clause est de toutes les façons consistante et le restera: il n'est donc pas nécessaire de garder celle-ci.

Dans le cas de l'exemple précédent, la deuxième dérivation de consistance commence donc avec les clauses

$$\left\{\begin{array}{l} \leftarrow \sim r(b) \\ \leftarrow q(Y,Z) \parallel (Y \neq b) \wedge (Z \neq b) \\ \square \parallel b \neq a \end{array}\right\}$$

qui contient une clause vide avec une restriction qui est satisfaite. Cette deuxième dérivation est donc un échec et cet échec est aussi celui de la dérivation abductive initiale.

4.2.3 Définition formelle de la procédure abductive

Pour récapituler ce qui précède, la procédure que nous allons formellement définir a les caractéristiques suivantes:

- Elle est construite sur le modèle de celle de (Kakas & Mancarella, 1990c);
- La résolution SLD avec contraintes est la règle d'inférence de base utilisée pour le calcul des résolvants;
- Toutes les fois qu'un littéral supposable est sélectionné pour être converti en hypothèse, ses (éventuelles) variables sont remplacées par des constantes de skolem;
- Un procédé analogue à la négation constructive est utilisée en cours de dérivation de consistance pour collecter les hypothèses portant sur les constantes de skolem et assurant la validité du test de consistance de l'hypothèse;
- De la même manière, les contraintes nécessaires au succès des réfutations abductives à l'intérieur d'une dérivation de consistance sont conservées comme hypothèses;
- Pour faciliter l'implantation des deux cas précédents, les hypothèses qui sont des contraintes (temporelles) sont conservées séparément des autres hypothèses;

- Dans une dérivation de consistance, les clauses sur lesquelles un échec est enregistré par absence de résolvant sont gardées comme des contraintes d'intégrité pour produire des résolvants avec les hypothèses ultérieures: ces résolvants sont utilisés comme clauses de départ des dérivations de consistance.

Ceci est repris dans les deux définitions suivantes. Comme pour la procédure de K&M, le calcul entrelace deux types de dérivation, tout en calculant incrémentalement l'ensemble des hypothèses Δ et l'ensemble Δ_t des hypothèses qui sont des contraintes temporelles.

Dans ces définitions, nous appelons $\mathcal{T}$-satisfaisabilité d'une contrainte R le fait que $\mathcal{T} \models \exists(R)$. Cette notation est étendue à un ensemble Δ_t de contraintes: si R est $\mathcal{T}, \Delta_t$-satisfaisable, cela signifie $\mathcal{T}, \Delta_t \models \exists(R)$.

Définition 4.3 (Dérivation abductive) *Soit G une clause but de la forme $\leftarrow B$. Une* dérivation abductive *de G est une séquence finie de nuplets:*

$$\langle G_1, \Delta_1^t, \Delta_1, I_1 \rangle \ldots \langle G_n, \Delta_n^t, \Delta_n, I_n \rangle$$

où G_i est une clause but, Δ_i^t est un ensemble de contraintes temporelles sans variables, Δ_i est un ensemble de littéraux sans variables, et I_i est un ensemble de contraintes d'intégrité. De plus, l'on a:

$$G_1 = G \qquad et \qquad I_1 = IC$$

Pour tout $i = 1, \ldots, n$, G_i est de la forme $\leftarrow L, L' \parallel R$ où L est le littéral sélectionné. Connaissant l'état i, l'état suivant $\langle G_{i+1}, \Delta_{i+1}^t, \Delta_{i+1}, I_{i+1} \rangle$ est obtenu par un des cas suivants:

(A_1) L est positif et non supposable, C est un résolvant de G_i et d'une variante d'une clause de P sur le littéral L, la restriction de ce résolvant étant $\mathcal{T}, \Delta_i^t$-satisfaisable, alors:

$$G_{i+1} = C \qquad \Delta_{i+1}^t = \Delta_i^t$$
$$\Delta_{i+1} = \Delta_i \qquad I_{i+1} = I_i$$

(A_2) L est soit positif et supposable, soit négatif, il existe dans Δ_i un littéral H de même symbole que L, tel que si Γ est l'ensemble des égalités terme à terme entre arguments de L et arguments de H, alors $R \wedge \Gamma$ est $\mathcal{T}, \Delta_i^t$-satisfaisable, alors:

$$G_{i+1} = \leftarrow L' \parallel R \wedge \Gamma \qquad I_{i+1} = I_i$$
$$\Delta_{i+1}^t = \Delta_i^t \qquad \Delta_{i+1} = \Delta_i$$

(A_3) L est soit positif et supposable, soit négatif, L ne satisfait pas la condition précédente, son opposé n'appartient pas à Δ_i, et il existe une dérivation de consistance depuis $\langle F_0, \Delta_i^t, \Delta_i \cup \{\sigma L\}, I_i \rangle$ qui finit en $\langle \{\}, \Delta_t', \Delta', I' \rangle$, alors:

$$G_{i+1} = \leftarrow \sigma L' \parallel \sigma R \qquad I_{i+1} = I'$$
$$\Delta_{i+1}^t = \Delta_t' \qquad \Delta_{i+1} = \Delta'$$

où σ est une substitution qui associe à chaque variable de L un nouvelle constante de skolem, et F_0 est l'ensemble de tous les résolvants de la clause $\sigma L \leftarrow$ avec les clauses de I_i.

Une dérivation abductive est donc la recherche des hypothèses à effectuer: la consistance de celles-ci étant alors vérifiée par une dérivation de consistance. Les cas prévus par la dérivation abductive sont analogues à ceux de K&M. La seule différence est la possibilité de sélectionner un littéral supposable avec variables dans les cas A_2 et A_3, ce dernier cas remplaçant ces variables par des constantes de skolem avant d'effectuer le test de consistance.

Définition 4.4 (Dérivation de consistance) *Une* dérivation de consistance *est une séquence finie de nuplets de la forme:*

$$\left\langle F_1, \Delta_1^t, \Delta_1, I_1 \right\rangle \ldots \left\langle F_m, \Delta_m^t, \Delta_m, I_m \right\rangle$$

telle que pour tout $i \in [1, m]$, F_i est un ensemble de clauses buts, F_m est l'ensemble vide, Δ_i^t est un ensemble de contraintes temporelles sans variables, Δ_i est un ensemble de littéraux sans variables, et I_i est un ensemble de contraintes d'intégrité. Chaque ensemble de clauses F_i peut s'écrire $\{\leftarrow L, L' \parallel R\} \cup F_i'$ de manière à identifier une clause particulière dans cet ensemble, et L est un littéral sélectionné dans $\leftarrow L, L' \parallel R$. Etant donné l'état $\langle F_i, \Delta_i^t, \Delta_i, I_i \rangle$, l'état suivant $\langle F_{i+1}, \Delta_{i+1}^t, \Delta_{i+1}, I_{i+1} \rangle$ est obtenu par l'un des cas suivants:

(C_1) il existe dans F_i une clause vide $C = \square \parallel R'$, alors:

$$\begin{array}{ll} F_{i+1} = F_i'' & \Delta_{i+1}^t = \Delta_i^t \cup \{D'\} \\ \Delta_{i+1} = \Delta_i & I_{i+1} = I_i \end{array}$$

où D' est une contrainte sans variables telle que $R' \wedge D'$ est inconsistant et Δ_{i+1}^t est $\mathcal{T}$-satisfaisable, et F_i'' est l'ensemble des clauses de $F_i - \{C\}$ dont la restriction est toujours satisfaisable compte tenu de Δ_{i+1}^t.

(C_2) L est positif et non supposable, C est l'ensemble de tous les résolvants de $\leftarrow L, L' \parallel R$ avec les clauses de P sur le littéral L, alors:

$$\begin{array}{l} F_{i+1} = C \cup F_i' \\ \Delta_{i+1}^t = \Delta_i^t \qquad \Delta_{i+1} = \Delta_i \\ I_{i+1} = \begin{cases} I_i & si\ C \neq \emptyset \\ I_i \cup \{\leftarrow L, L' \parallel R\} & si\ C = \emptyset \end{cases} \end{array}$$

(C_3) L est soit positif et supposable, soit négatif, C est l'ensemble de tous les résolvants de $\leftarrow L, L' \parallel R$ avec les éléments de Δ_i, alors:

$$\begin{array}{ll} F_{i+1} = C \cup F_i' & \Delta_{i+1}^t = \Delta_i^t \\ \Delta_{i+1} = \Delta_i & I_{i+1} = I_i \cup \{\leftarrow L, L' \parallel R\} \end{array}$$

(C_4) L est soit positif et supposable, soit négatif, L est sans variables (ou ses variables ont des valeurs complètement déterminées par les contraintes R), et l'opposé de L appartient à Δ_i, alors :

$$F_{i+1} = F'_i \qquad \Delta^t_{i+1} = \Delta'_t$$
$$\Delta_{i+1} = \Delta' \qquad I_{i+1} = I_i$$

(C_5) L est soit positif et supposable, soit négatif, L est sans variables (ou ses variables ont des valeurs complètement déterminées par les contraintes R), et il existe une dérivation abductive de $\langle \leftarrow \neg L, \Delta^t_i, \Delta_i, I_i \rangle$ à $\langle \square \parallel R', \Delta'_t, \Delta', I' \rangle$ alors :

$$F_{i+1} = F'_i \qquad \Delta_{i+1} = \Delta'$$
$$I_{i+1} = I'$$
$$\Delta^t_{i+1} = \begin{cases} \Delta'_t & \text{si } \mathcal{T}, \Delta'_t \models \exists(R') \\ \Delta'_t \wedge R'' & \text{sinon.} \end{cases}$$

où R'' est tel que $\mathcal{T}, \Delta'_t \wedge R'' \models \exists(R')$ et $\Delta'_t \wedge R''$ est $\mathcal{T}$-satisfaisable.

Si une telle dérivation n'existe pas, et si L est négatif, alors :

$$F_{i+1} = \{\leftarrow L' \parallel R\} \cup F'_i$$
$$\Delta_{i+1} = \Delta_i \qquad \Delta^t_{i+1} = \Delta^t_i \qquad I_{i+1} = I_i$$

Le cas C_1 s'applique lorsqu'une clause vide est présente dans l'ensemble courant de clauses F_i. La contrainte D' qui est calculée à cette occasion est une contrainte sans variables, qui est typiquement l'opposée d'une des contraintes de R'. Dans le cas C_2, la branche de la dérivation qui finit sur la clause choisie est étendue à l'aide des résolvants obtenus avec les clauses du programme. Ce cas exclut la possibilité pour le littéral L d'être supposable, ce qui empêche de mettre dans le programme certaines instances de prédicats supposables. Si tel était le cas, il est plus simple d'initialiser Δ avec ces instances : elles seront alors traitées par le cas C_3, et une trace en sera gardée dans l'ensemble des contraintes d'intégrité.

Le cas C_4 élimine la clause choisie car elle est déja consistante avec les hypothèses. La restriction sur l'absence de variables dans le littéral L se comprend car Δ ne contient que des hypothèses sans variables et en aucun cas des hypothèses de la forme $\forall X\ p(X)$. Pour finir, le cas C_5 permet de relancer une dérivation abductive interne à la dérivation de consistance. De même que pour le cas précédent (C_4) la restriction sur L est nécessaire car une dérivation abductive ne peut répondre qu'à des requêtes existentielles.

Du bon usage des contraintes

Les deux définitions précédentes font un usage non négligeable de la notion de $\mathcal{T}$-satisfaisabilité, que nous allons maintenant analyser avec plus de détails. Si, comme

précédemment, $\mathcal{T}$ désigne la théorie des relations d'ordre entre instants, cette notation signifie que

$$\mathcal{T} \models \exists(C)$$

c'est à dire, qu'il existe une substitution σ des variables libres de C telle que σC est satisfait par $\mathcal{T}$, ou encore est prouvable à partir de $\mathcal{T}$. La généralisation à $\mathcal{T}, \Delta_t$ comme cela est utilisée dans les définitions précédentes, est immédiate.

La consistance telle qu'elle est vérifiée par un algorithme de propagation de contraintes est définie par l'existence d'une solution aux contraintes, c'est à dire d'une affectation à chaque instant d'une valeur (dans $\mathcal{R}$ par exemple), telle que le placement de ces nœuds sur l'axe des nombres réels (donc un ordre total) satisfasse toutes les contraintes de l'ensemble C. Cette notion de consistance n'accorde aucune importance au fait que les nœuds soient étiquetés par des constantes ou par des variables.

Il est donc nécessaire de compléter le résultat retourné par l'algorithme de propagation de contraintes par un traitement qui différencie les instants suivant leur forme logique, constantes, constantes de skolem ou variables. Comme les constantes de skolem n'existent que pour nommer des points de temps qui correspondent par ailleurs à des variables quantifiées existentiellement, leur statut est donc équivalent à celui des variables lorsqu'il s'agit de déterminer la satisfaisabilité.

Un ensemble de contraintes $C = c_1 \wedge \ldots \wedge c_n$ entre des instants sera donc $\mathcal{T}, \cdot_{\sqcup}$-satisfiable si les conditions suivantes sont satisfaites :

1. un algorithme de propagation de contraintes correct et complet indique que l'ensemble de contraintes (ou le graphe correspondant) est consistant :

2. si C' désigne la fermeture transitive des contraintes de C (calculée par exemple à l'aide d'un algorithme tel que PC2), alors toute contrainte de C' qui ne contient ni variable ni constante de skolem peut être déduite à partir de $\mathcal{T} \wedge \Delta_t$;

3. de la même manière toute contrainte entre une constante de skolem et une autre constante, ordinaire ou de skolem, n'est pas contradictoire avec $\mathcal{T}, \Delta_t$.

Le test de prouvabilité d'une contrainte (sans variables) à partir de $\mathcal{T} \wedge \Delta_t$ est aisément implanté par comparaison des relations minimales induites par Δ_t et celles induites par C : le même algorithme est utilisé pour obtenir les deux représentations.

L'intérêt de cette définition est de tenir compte de la densité du temps telle qu'elle est définie habituellement par la relation suivante :

$$\forall t_1, t_2 \ \ (t_1 < t_2) \ \Rightarrow \ (\exists t_3 \ \ t_1 < t_3 < t_2)$$

Cette relation n'est que partiellement implantée par l'algorithme de propagation de contraintes, par exemple si l'on utilise le formalisme de contraintes quantitatives de (Dechter et al., 1991) en représentant les bornes des intervalles portés par les

contraintes par des nombres réels. Par exemple, si C est l'ensemble de contraintes $(a < T) \wedge (T < 3)$, cet ensemble est consistant pour un algorithme de propagation de contraintes, mais ne sera satisfaisable que si $a < 3$ est vérifié.

Compte tenu de ce qui précède, il est aisé de déterminer comment doit être choisie une contrainte dans le cas C_1 de la dérivation de consistance. Ce cas cherche, en modifiant $\mathcal{T}$ par l'ajout d'une contrainte D' dans Δ_t, à rendre insatisfaisable les contraintes d'une clause vide $\square \parallel R'$ apparue parmi les clauses courantes de la dérivation. La troisième condition de la satisfaisabilité est la plus évidente à invalider: une contrainte D' choisie dans le cas C_1 sera donc la négation d'une contrainte de R' liant une constante de skolem avec une autre constante.

De la même manière, dans le cas C_5, le but est de modifier $\mathcal{T}$ de manière à ce que R' soit satisfaisable. Les contraintes R'' rajoutées à cette occasion sont typiquement l'ensemble des contraintes sans variables présentes dans R'. L'on peut aussi utiliser l'algorithme présenté dans le chapitre 2 à la page 70 qui extrait des contraintes du résolvant une substitution: comme celle-ci ne lie que des variables avec d'autres termes, elle ne restreint pas le choix des contraintes que l'on peut ajouter à Δ_t dans les cas C_1 et C_5 de la dérivation de consistance.

4.3 Quelques exemples

Préalablement à l'étude spécifique de notre procédure en planification, nous allons maintenant illustrer le fonctionnement de cette procédure à l'aide de deux exemples inspirés de problèmes classiques du raisonnement temporel.

Ces exemples sont décrits à l'aide du formalisme présenté dans le chapitre 2, paragraphe 2.2.2 (page 47). Les clauses contraintes correspondantes apparaissent sur la figure 4.5. Par rapport à la présentation qui avait été faite de cette logique, nous avons donc introduit le prédicat $persist(T, T', P)$ qui prend en argument deux instants et une proposition et qui décrit le fait que la proposition P est vrai au moins de T à T' : ce qui, plus formellement, signifie qu'il existe une période de vérité de la proposition P dans laquelle est inclue la période de T à T'. Cette définition permet de tirer parti de l'abduction pour manipuler la persistance (cf. chapitre 3, paragraphe 3.2.1 page 75), et c'est l'objet du premier exemple d'illustrer ceci.

4.3.1 Raisonnement avec la persistance

Le but de l'exemple présenté maintenant est d'illustrer la prise en compte de la persistance par l'abduction. Cet exemple est donc décrit avec le formalisme temporel précédent, cependant, les résultats sont généralisables à tout formalisme qui dispose d'un prédicat pour matérialiser la persistance. Conformément à l'optique de l'abduction, il n'est pas nécessaire de prévoir les mécanismes d'inférence pour déduire cette persistance, il suffit de décrire comment elle intervient dans la formation des

$$
\begin{array}{ll}
\leftarrow hold(T_1, T_2, P) \parallel \neg(T_1 < T_2) & A_1 \\
\leftarrow persist(T_1, T_2, P) \parallel \neg(T_1 < T_2) & A_2 \\
begin(T_1, P) \leftarrow hold(T_1, T_2, P) & A_3 \\
end(T_2, P) \leftarrow hold(T_1, T_2, P) & A_4 \\
hold(T_1, T_2, P) \leftarrow begin(T_1, P),\ persist(T_1, T_2, P),\ end(T_2, P) & A_5 \\
true(T, P) \leftarrow hold(T_1, T_2, P) \parallel (T_1 < T) \wedge (T \leq T_2) & A_6 \\
true(T_2, P) \leftarrow begin(T_1, P),\ persist(T_1, T_2, P) \parallel T_1 < T_2 & A_7 \\
\leftarrow persist(T_1, T_2, P),\ begin(T_3, P) \parallel (T_1 < T_3) \wedge (T_3 \leq T_2) & A_8 \\
\leftarrow persist(T_1, T_2, P),\ end(T_3, p) \parallel (T_1 \leq T_3) \wedge (T_3 < T_2) & A_9
\end{array}
$$

FIG. 4.5 - *Les clauses avec contraintes qui définissent le formalisme temporel utilisé pour les exemples. Les clauses A_1, A_2, A_8 et A_9 sont des contraintes d'intégrité.*

périodes de vérité et les conditions qui assurent sa consistance: l'abduction fait le reste...

L'exemple que nous présentons est un sous-ensemble du « *Yale Shooting Problem* » où nous nous focalisons sur la persistance. Cet exemple suppose un pistolet qui est chargé à un temps t_0, et dont on actionne la gachette au temps 4. Pour prouver le succès du coup de feu, il faut donc que le pistolet soit resté chargé entre les instants t_0 et 4. Cet exemple est décrit par les clauses suivantes:

$$
\begin{array}{ll}
occur(t_0, loading) & R_1 \\
occur(4, pulltrigger) & R_2 \\
begin(T, loaded) \leftarrow occur(T, loading) & R_3 \\
end(T, loaded) \leftarrow occur(T, unloading),\ true(T, loaded) & R_4 \\
end(T, loaded) \leftarrow occur(T, pulltrigger),\ true(T, loaded) & R_5
\end{array}
$$

Le succès du coup de feu est par exemple représenté par la formule $end(4, loaded)$. Cette requête est réfutée à l'aide de notre procédure abductive: les étapes de cette dérivation abductive sont présentées sur la figure 4.6

L'hypothèse nécessaire pour satisfaire la requête est $persist(t_0, 4, loaded)$. La consistance de celle-ci est testée par la dérivation de consistance qui apparaît sur la figure 4.7. Le premier ensemble de clauses contient une clause vide pour laquelle il est possible en supposant $t_0 < 4$ de rendre inconsistant les contraintes qu'elle porte. Au fur et à mesure de la dérivation la plupart des clauses générées sont éliminées, soit par absence de résolvants comme pour la clause

$$\leftarrow occur(T_3, unloading), true(T_3, loaded) \parallel (t_0 \leq T_3) \wedge (T_3 < 4)$$

soit parce que leur restriction est inconsistante, comme pour la clause

$$\leftarrow occur(T_3, loading) \parallel (t_0 < T_3) \wedge (T_3 \leq 4)$$

qui, avec le littéral $occur(t_0, loading)$, produit $\square \parallel (t_0 < t_0) \wedge (t_0 < 4)$.

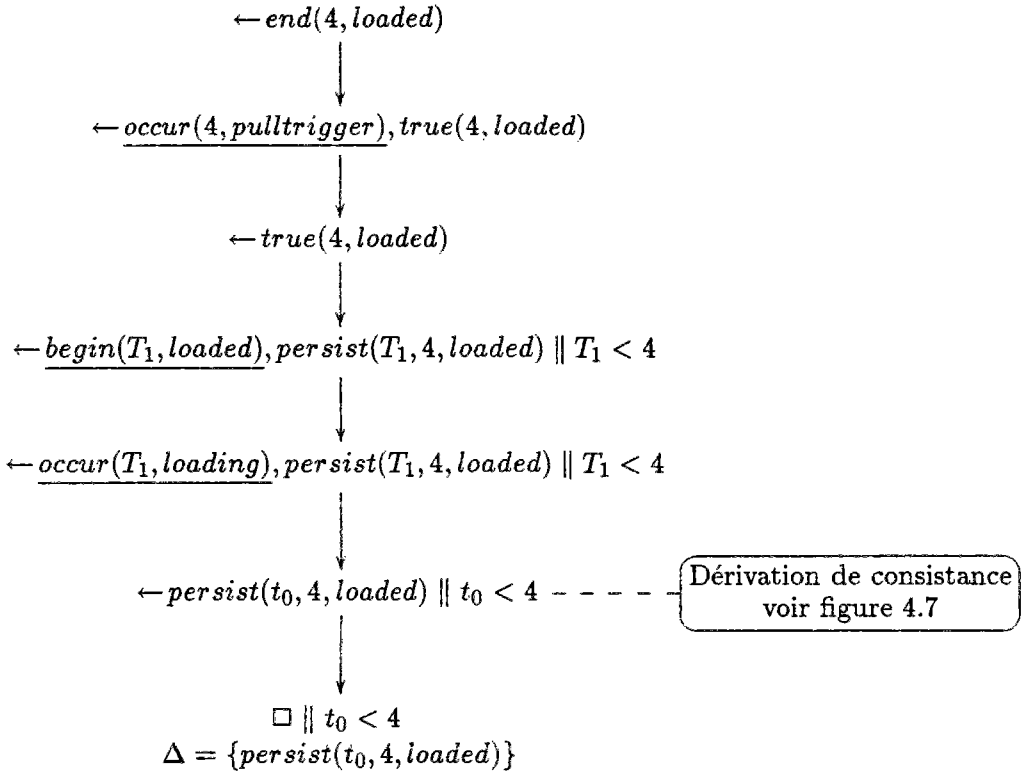


FIG. 4.6 - *Les étapes de la dérivation abductive du but* $\leftarrow end(4, loaded)$ *dans le problème du YSP.*

$$\left\{\begin{array}{l} \leftarrow begin(T_3, loaded) \parallel (t_0 < T_3) \wedge (T_3 \leq 4) \\ \leftarrow end(T_3, loaded) \parallel (t_0 \leq T_3) \wedge (T_3 < 4) \end{array}\right\}$$
$$\Delta = \{persist(t_0, 4, loaded)\} \qquad \Delta_t = \{\}$$

$$\downarrow$$

$$\left\{\begin{array}{l} \leftarrow occur(T_3, loading) \parallel (t_0 < T_3) \wedge (T_3 \leq 4) \\ \leftarrow hold(T_3, T_4, loaded) \parallel (t_0 < T_3) \wedge (T_3 \leq 4) \\ \leftarrow occur(T_3, unloading), true(T_3, loaded) \parallel (t_0 \leq T_3) \wedge (T_3 < 4) \\ \leftarrow occur(T_3, pulltrigger), true(T_3, loaded) \parallel (t_0 \leq T_3) \wedge (T_3 < 4) \\ \leftarrow hold(T_4, T_3, loaded) \parallel (t_0 \leq T_3) \wedge (T_3 < 4) \end{array}\right\}$$

$$\downarrow$$

$$\{\ \}$$
$$\Delta = \{persist(t_0, 4, loaded)\}$$

FIG. 4.7 - *Les grandes phases de la dérivation de consistance pour l'hypothèse* $persist(t_0, 4, loaded)$ *dans le problème du YSP.*

4.3.2 Génération d'explications

L'exemple qui suit demande la génération d'explications *a posteriori* pour une situation inattendue. Il s'agit du « *Stolen Car Problem* » (Kautz, 1986). Celui-ci suppose que l'on gare une voiture dans un garage (proposition *parked*) à la date 12, en fin de journée (proposition *day*), et deux jours plus tard la voiture a disparu. Ce que l'on sait, c'est que voler une voiture (*steal*) cause sa disparition, et que ceci ne peut se produire que la nuit (*night*). Tout ceci est décrit à l'aide des clauses suivantes :

$hold(0, 12, day) \leftarrow .$
$hold(T, T + 12, day) \leftarrow end(T, night).$
$hold(T, T + 12, night) \leftarrow end(T, day).$
$begin(12, parked) \leftarrow .$
$end(T, parked) \leftarrow true(T, parked),\ occur(T, steal).$
$\leftarrow occur(T, steal),\ hold(T_1, T_2, day) \parallel (T_1 < T \leq T_2).$

La requête permettant de savoir quand et pourquoi la voiture peut disparaître est $end(T, parked)$. La dérivation abductive correspondante apparaît sur la figure 4.8.

Cette dérivation abductive rencontre tout d'abord le littéral $occur(T, steal)$ qui est skolémisé avant de tester la consistance de cette hypothèse. Ce test est représenté sur la figure 4.9 : la particularité de ce test est qu'il existe plusieurs solutions, car les restrictions des clauses vides qui apparaissent dans la dérivation sont des conjonctions de plusieurs contraintes. Les deux solutions représentées sont celles qui

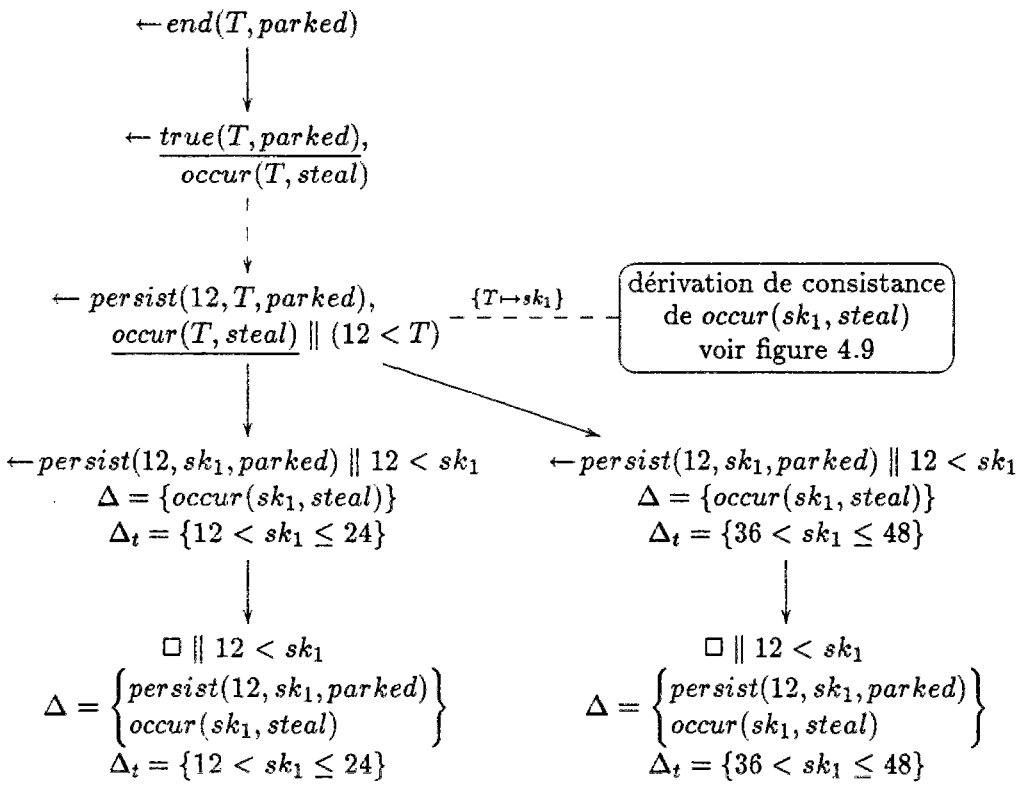


FIG. 4.8 - *La dérivation abductive de $end(T, parked)$ dans le Stolen Car Problem. Cette dérivation se sépare en deux après la première dérivation de consistance car celle-ci retourne deux solutions.*

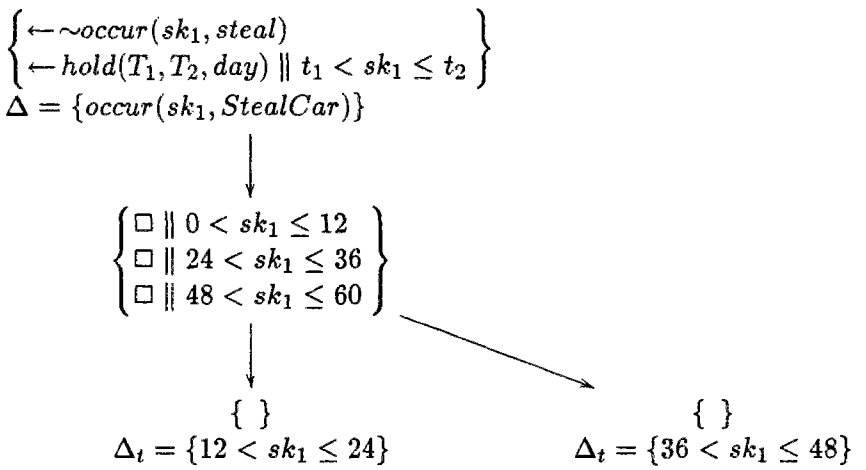


FIG. 4.9 - *La dérivation de consistance de l'hypothèse $occur(sk_1, steal)$ dans le Stolen Car Problem. La dérivation de consistance représentée ici a deux solutions correspondant à différents choix de contraintes dans les clauses vides obtenues. On représente ceci par les deux branches finales.*

permettent à la dérivation abductive initiale (figure 4.8) de produire à son tour deux solutions. Dans chacune des branches finales de cette dérivation abductive le test de l'hypothèse $persist(12, sk_1, parked)$ est identique et réussit sans problème.

Les deux solutions retournées correspondent chacune à une occurrence du vol de la voiture pendant une des deux nuits (entre les dates 12 et 24, ou 36 et 48) : dans chacun des cas, l'hypothèse de persistance de *parked* impose la présence de la voiture dans le garage jusqu'au moment du vol.

4.4 Remarques préliminaires sur l'implantation

A ce point de notre définition, il est important de faire quelques remarques sur la manière d'implanter une telle procédure. Nous reviendrons dans le chapitre 5 sur cette implantation, mais certains points de la procédure doivent faire l'objet d'un compromis qui pourra limiter la puissance de celle-ci. Cette courte étude est motivée par l'intérêt que présentent les caractéristiques de la procédure avec certains choix d'implantation, en particulier pour la planification qui sera étudiée dans le paragraphe 4.5.

Il existe une grande marge de manœuvre pour implanter cette procédure : la définition suppose à plusieurs endroits le choix d'un élément, clause ou littéral, pour lequel il n'y a pas d'*a priori* possible en dehors d'une application concrète. Nous

pouvons malgré tout identifier les points suivants :

- La règle de sélection des littéraux n'est pas décrite dans la définition de la procédure. D'un point de vue pratique, il est souhaitable que celle-ci soit adaptable aux désirs de l'utilisateur, sans pour autant être complexe, et donc imprédictible, pour ne pas compliquer la tâche d'écriture des clauses d'un problème particulier. A cet égard, la règle de sélection de Prolog, où l'on choisit toujours le premier littéral du but courant, est très facile d'emploi, mais elle ne garantit pas la complétude du processus de réfutation car elle ne permet pas d'éviter les branches infinies. Cependant, la distinction entre littéraux ordinaires et littéraux supposables nécessite parfois que ces derniers soient retardés le long d'une branche de dérivation jusqu'à l'épuisement des littéraux ordinaires : ceci permet d'instancier autant que possible les littéraux supposables et limite le nombre de constantes de skolem à générer et à contraindre pour la suite de la dérivation.

- De la même manière, il est parfois souhaitable de sélectionner conjointement plusieurs littéraux supposables. Ceci apparaît nécessaire lorsque les hypothèses ne sont pas indépendantes : disposer de l'ensemble des clauses dans une même dérivation de consistance permet de mener à bon port une dérivation qui sinon se perdrait dans l'espace de recherche. Ce type de sélection implique en particulier l'usage de la sélection retardée décrite précédemment.

- Le choix des clauses dans la dérivation de consistance a aussi des conséquences complexes. Parmi toutes les clauses d'un ensemble F_i, il ne semble pas évident d'identifier un critère de sélection dont l'application soit locale, et qui tienne compte des dépendances entre les hypothèses, déjà faites et à venir. C'est pour cela que nous préférons imposer une règle simple, la première clause d'abord, en imposant bien sur que ces clauses soient rangées au début de la dérivation de consistance dans l'ordre correspondant aux contraintes d'intégrité dans la description du problème. Ceci est une règle simple et prédictible que l'on peut prendre en compte pour l'écriture du problème.

- Le choix des contraintes temporelles dans le cas C_1 de la dérivation de consistance peut être raffiné de multiples manières. Une première heuristique est de choisir une contrainte qui fait intervenir au moins une constante de skolem. Une autre est de choisir la première contrainte dans l'ordre où elles apparaissent dans la restriction de la clause vide. Nous préférons la première solution qui maximise « l'utilité » de la contrainte ajoutée à l'ensemble Δ_t. C'est à ce niveau qu'il serait possible d'implanter des heuristiques permettant, par exemple, de minimiser l'étendue temporelle du plan généré si l'on se place dans le cadre de la planification.

Un autre point important est la faculté pour les différentes étapes de la procédure (dérivation abductive et dérivation de consistance) de retourner un nombre borné de

solutions. Par exemple, compte tenu du fait que plusieurs choix possibles dans le cas C_1 de la dérivation de consistance peuvent conduire à plusieurs solutions distinctes pour celle-ci, ces solutions donnant en retour la possibilité à la dérivation abductive appelante d'explorer autant de solutions, il est souhaitable d'imposer une limitation, forcément arbitraire, au nombre des solutions retournées. On peut remarquer que, comme précédemment, le choix le plus simple et le plus prédictible est de se limiter à une solution dans tous les cas de figures.

Pour résumer, une implantation suffisament souple de cette procédure doit offrir les possibilités suivantes :

- une règle de sélection à la Prolog, le littéral le plus à gauche en premier ;
- une règle de sélection retardée, qui sélectionne les littéraux supposables après tous les littéraux ordinaires ;
- une sélection conjointe de tous les littéraux supposables dans la clause but courante. Cette règle présente un intérêt si l'on utilise aussi la sélection retardée ;
- les clauses courantes d'une dérivation de consistance sont rangées dans l'ordre de leur production par les contraintes d'intégrité, celles-ci étant prises dans l'ordre de la description du problème ;
- le nombre de solutions retournées à chaque étape, dérivation abductive et dérivation de consistance, doit être paramétrable, la valeur par défaut étant 1.

Nous allons maintenant étudier plus précisément comment cette procédure peut être utilisée pour la planification, et les capacités qu'elle présente pour cette application. Comme la plupart des travaux sur la planification abductive ont été réalisés à l'aide du formalisme du Calcul d'Evénements de (Kowalski & Sergot, 1986), nous nous restreindrons à cet usage. La procédure abductive de (Missiaen, 1991) nous servira de repère tout au long de cette étude.

4.5 Planification avec le calcul d'événements

L'usage de l'abduction pour la planification a été étudié principalement par (Missiaen, 1991). Son étude se plaçait dans le cadre du calcul d'événements, et l'implantation de sa procédure dans le planificateur CHICA (Missiaen, Bruynooghe, & Denecker, 1993), est très liée à ce formalisme. Dans ce qui suit, sa procédure nous servira de référence, et, afin de faciliter les comparaisons entre celle-ci et la procédure que nous proposons, nous étudierons celle-ci avec le calcul d'événements.

Nous présentons tout d'abord le calcul d'événements et nous donnons les clauses avec contraintes qui le décrivent pour notre procédure. Ensuite, nous étudierons, sur l'exemple du monde des blocs, l'apport de notre procédure à la planification

$$(1.1)\quad \text{Holds\_at}(P,T) \leftarrow \text{Happens}(E),\ \text{Initiates}(E,P),\text{Succeeds}(E),\ \sim\text{Clipped}(date(E),P,T) \parallel (date(E) < T).$$

$$(1.2)\quad \text{Clipped}(T_1,P,T_2) \leftarrow \text{Happens}(E),\ \text{Terminates}(E,P),\ \text{Succeeds}(E) \parallel (T_1 \leq date(E)) \wedge (date(E) < T_2).$$

FIG. 4.10 - *Les axiomes du calcul d'événements sous forme de clauses contraintes. Les variables E désignent des événements, P désigne une propriété, et les variables* $T_1, T_2, \ldots$ *désignent des instants.*

abductive. Tout au long de cette partie, la plupart des exemples que nous présentons sont tirés de (Missiaen, 1991).

4.5.1 Le calcul d'événements

Pour l'usage de notre procédure, il est d'abord nécessaire d'écrire les clauses du calcul d'événements comme des clauses avec contraintes. Pour cela, par rapport à la version qui avait été introduite dans le chapitre 1 (page 25), nous introduisons explicitement la date d'occurrence d'un événement par la fonction *date* appliquée à celui-ci.

Les clauses contraintes du calcul d'événements ainsi modifié apparaissent sur la figure 4.10. En plus de ces clauses, on définit un événement particulier *start* qui arrive toujours (Happens(*start*)), réussit toujours (Succeeds(*start*)), et dont la date d'occurrence est antérieure à toute autre date.

Le calcul d'événements utilisé par Missiaen a une formulation plus complexe que celui de Shanahan. En premier lieu, il impose un certain nombre de contraintes sur la forme des clauses décrivant le domaine, et sur la forme des buts générés, afin que sa procédure soit correcte. De telles contraintes sont, par exemple, que toute variable d'une règle qui représente un événement doit au moins apparaître dans un littéral *Happens*/1 (Missiaen, 1991, page 80), ou encore qu'un littéral *Happens*/1 dans un but doit toujours être sélectionné avant tout autre littéral de ce but (Missiaen, 1991, page 81). On peut ajouter l'absence de *context dependent terminating events*, qui a été décrite précédemment. Ces contraintes sont extérieures au langage et doivent être traitées par l'implantation, ce qui en limite la généralité.

Une autre modification apportée par Missiaen est l'introduction d'un prédicat supposable Persists(E,P,T), avec une signification contraire à Clipped(E,P,T), et utilisé à la place de $\sim$Clipped(E,P,T) dans la définition de *Holds_at*/2. L'inconvénient de la négation par échec est que la preuve subséquente d'un littéral Clipped(e,p,t) nécessite de connaître totalement l'ordre entre les événements et les instants, afin que l'hypothèse de persistance reste valable dans toutes les linéarisations de l'ordre (partiel) du plan. La définition de la persistance de p entre e et t telle qu'elle est implantée dans le calcul d'événements de Missiaen est basée sur la

remarque suivante :

> « *pour tout événement réussi qui termine p, soit cet événement arrive avant e, soit il doit arriver après l'instant t* » (Missiaen, 1991, page 34)

et utilise la négation par échec sur les prédicat d'ordonnancement entre événements dans la définition de Clipped3. L'implantation par Missiaen de *Persists*/3 pose aussi un problème car elle repose sur le (méta-)prédicat *findall*/3 de Prolog pour identifier tous les événements susceptibles de terminer P avant l'instant T. L'usage de *findall*/3 est une source d'incomplétude pour sa procédure, principalement parce que cet appel ne retourne que les événements couramment supposés qui peuvent terminer P. Ceci oblige alors à revérifier toutes les hypothèses de persistance à chaque nouvel événement supposé. Notre procédure utilise un mécanisme analogue grâce à l'ensemble des contraintes d'intégrité I construit incrémentalement au cours de la procédure : tout nouvel événement supposé est résolu avec ces contraintes d'intégrité, ce qui a pour effet de ne revérifier que les dérivations de consistance antérieures qui sont concernées.

Une dernière modification est apportée par Missiaen pour réduire les retours arrière dans sa procédure. Il introduit un prédicat Maintain(E,P,T) dont la signification est toujours opposée à celle de *Clipped*/3, et qui est aussi utilisée dans la définition de *Holds_at*/2 à la place de l'appel en négation par échec de *Clipped*/3. Ce prédicat *Maintain*/3 possède une première définition à l'aide de *Persists*/3, et une définition alternative (utilisée en cas de retour arrière) qui permet d'implanter un mécanisme analogue à celui décrit par (Chapman, 1987) : si la propriété P est interrompue avant la date T, l'occurrence d'un autre événement, le « *white knight* » (chevalier blanc), est supposée pour rétablir la propriété P avant la date T. Le seul retour arrière qui se produit est celui qui intervient pour prouver un but Maintain(e,p,t) à l'aide de sa deuxième définition.

Sans modification du calcul d'événements par rapport à la formulation de Shanahan, notre procédure peut implanter un mécanisme analogue, dont l'efficacité est moyenne car il suppose malgré tout un retour arrière entre deux instanciations possibles d'un même but. Soient trois événements e_1, e_2 et e_3 tels que e_1 et e_3 peuvent commencer p (*ie.* Initiates(e_1,p) et Initiates(e_3,p) peuvent être prouvés), et e_2 peut terminer p. Seuls e_1 et e_2 ont des occurrences, et on connaît l'ordre de celles-ci :

$$date(e_1) \xrightarrow{<} date(e_2) \xrightarrow{<} t$$

Prouver que p est vrai à la date t produit la dérivation de la figure 4.11. La première branche de la dérivation (avec E lié à e_1) produit un échec à cause de l'échec de la dérivation de consistance subséquente : l'on dérive dans cette dérivation de consistance une clause vide $\square \parallel date(e_1) \leq date(e_2) < t$ dont la restriction est satisfaite et qu'aucune hypothèse d'ordre ne peut rendre inconsistante, compte tenu de ce que l'on sait de l'ordre entre les événements. La deuxième branche utilise e_3 comme événement initiateur, dont il faut supposer l'occurrence (hypothèse Happens(e_3)) et la

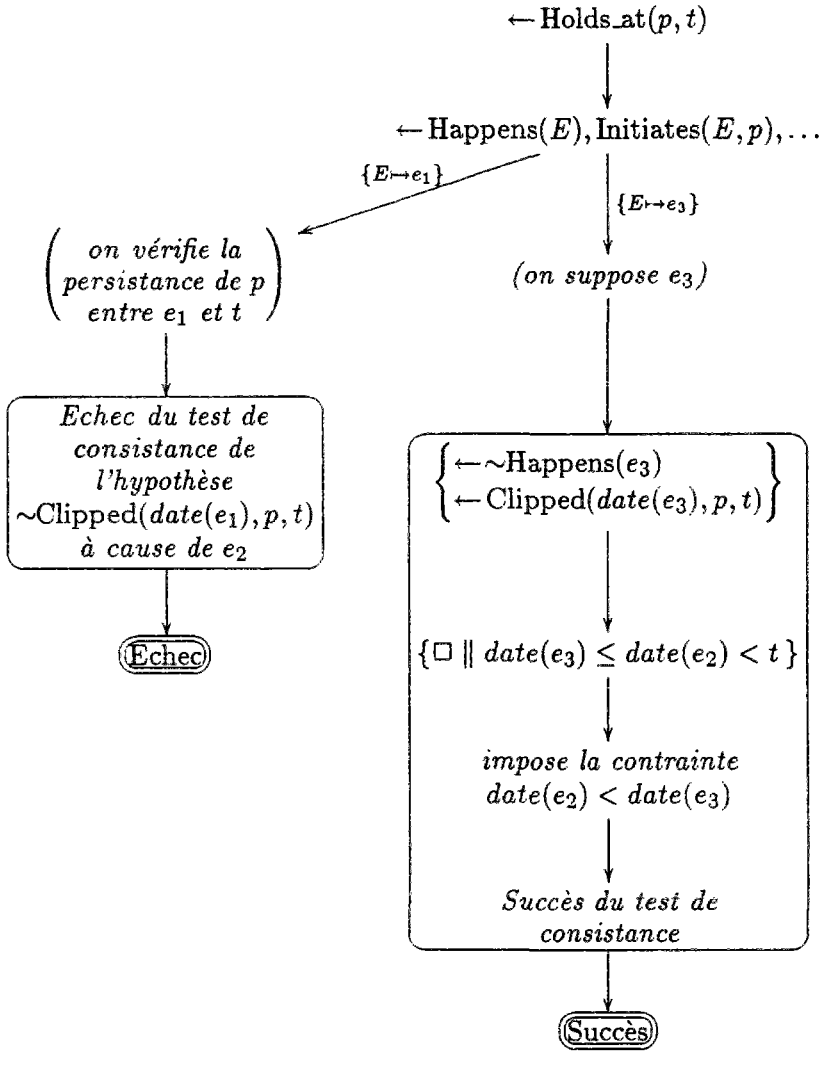


FIG. 4.11 - *Introduction d'un événement qui joue le rôle d'un* white knight, *pour prouver que p est vrai à la date t.*

persistance de p depuis cette occurrence (hypothèse Clipped($date(e_3), p, t$)). Cette deuxième dérivation de consistance contraint l'occurence de e_3 à intervenir après celle de e_2, faisant de e_3 l'équivalent d'un « *white knight* ».

L'efficacité de ce mécanisme n'est pas très bonne. Dans chacune des branches de la dérivation abductive, on effectue une dérivation de consistance sur une hypothèse négative ~Clipped(...), avec pour seule différence l'événement initiateur de la propriété p, et donc la borne inférieure de la période de persistance.

4.5.2 Exemples simples avec le calcul d'événements

Les deux exemples qui vont suivre ont été proposés par (Missiaen, 1991) comme des exemples qui illustraient certaines incapacités de sa procédure.

Le premier exemple rentre dans la catégorie qu'il appelle « *context dependent terminating events* », c'est à dire que le succès des événements dépend de conditions à la date de leur occurrence. L'exemple de Missiaen est le suivant :

$$\begin{array}{l}\text{Initiates}(start, r).\\ \text{Initiates}(e_1, p).\\ \text{Initiates}(e_2, q).\\ \text{Succeeds}(e_1).\\ \text{Succeeds}(e_2).\\ \text{Terminates}(e_1, r) \leftarrow \text{Holds\_at}(q, date(e1)).\\ \text{Terminates}(e_2, r) \leftarrow \text{Holds\_at}(p, date(e2)).\end{array}$$

pour lequel on cherche à satisfaire le but :

$$\leftarrow \text{Holds\_at}(t_1, p), \text{Holds\_at}(t_1, q), \text{Holds\_at}(t_1, r)$$

avec t_1 qui est bien sûr une date postérieure à $date(start)$. Sont référencés comme supposables les instances du prédicat Happens, et l'on se limite à des occurrences de e_1 et e_2 qui ont lieu entre $date(start)$ et t_1, car ce sont les seules à être pertinentes.

La difficulté de cet exemple est qu'il n'y a pas de plan linéaire pour atteindre le but. Quel que soit l'ordre entre les événements e_1 et e_2, le premier de ceux-ci active la condition permettant au deuxième de terminer r ce qui invalide donc la possibilité que r soit toujours vrai à la date t_1, en même temps que p et q. La procédure de Missiaen trouve une solution incorrecte en ordonnant strictement ces deux actions, e_2 avant e_1.

Notre procédure trouve la solution suivante :

$$\Delta = \left\{ \begin{array}{l} \text{Happens}(e_1), \text{Happens}(e_2),\\ \sim\text{Clipped}(date(e_1), p, t_1), \sim\text{Clipped}(date(e_2), q, t_1),\\ \sim\text{Clipped}(date(start), r, t_1) \end{array} \right\}$$

avec les hypothèses d'ordre suivantes :

$$\Delta_t = \{date(e_1) \leq date(e_2),\ \ date(e_2) \leq date(e_1)\}$$

qui imposent en réalité l'occurrence simultanée de e_1 et de e_2. Cette solution est correcte, et s'appuie sur une particularité du calcul d'événements tel qu'il a été défini précédemment, à savoir que lorsque une propriété est débutée par l'occurrence d'un événement e, alors cette propriété n'est pas vraie à la date d'occurrence de e, et le devient juste après. C'est ce qui permet à notre procédure de construire une solution au problème précédent.

En ce qui concerne les « *context dependent terminating events* », Missiaen donne une méthode d'élimination de ceux-ci dans la description d'un problème avec le calcul d'événements. Ici, un tel prétraitement des clauses ne semble pas nécessaire.

Le deuxième exemple proposé ici provient toujours de (Missiaen, 1991), et illustre un cas d'incomplétude de sa procédure. Cet exemple est décrit par les clauses suivantes :

$$\begin{array}{l}\text{Initiates}(start, r).\\ \text{Initiates}(start, q).\\ \text{Initiates}(e_1, p).\\ \text{Succeeds}(e_1).\\ \text{Succeeds}(e_2).\\ \text{Terminates}(e_1, r) \leftarrow \text{Holds\_at}(q, date(e_1)).\\ \text{Terminates}(e_2, q).\end{array}$$

et le but cherché est :

$$\leftarrow \text{Holds\_at}(t_1, p), \text{Holds\_at}(t_1, r)$$

Pour que r soit toujours vrai à t_1, il ne faut pas que cette propriété soit terminée par l'occurrence de e_1, qui est pourtant nécessaire pour prouver que p est vrai à la date t_1. La solution est que e_2 intervienne strictement avant e_1 pour terminer q, ce qui enlève alors la possibilité pour e_1 de terminer r.

La procédure de Missiaen ne trouve pas de solution à ce problème. Notre procédure trouve la solution suivante :

$$\Delta = \left\{ \begin{array}{l} \text{Happens}(e_1), \text{Happens}(e_2),\\ \sim\text{Clipped}(date(e_1), p, t_1), \sim\text{Clipped}(date(start), r, t_1) \end{array} \right\}$$

avec la contrainte temporelle

$$date(e_2) < date(e_1)$$

Ces deux exemples illustrent un certain avantage de notre procédure par rapport à celle de Missiaen. L'avantage est ici de se placer dans le cadre de la programmation

logique avec contraintes et de se reposer entièrement sur des techniques de propagation de contraintes pour gérer les relations d'ordre. De ce fait, les différentes étapes de la procédure pendant lesquelles sont sélectionnées des hypothèses sous forme de contraintes temporelles ne sont pas limitées dans leur choix, comme cela serait le cas si l'égalité était par exemple traitée avec un algorithme d'unification, et les relations d'ordre par des clauses spécifiques et le mécanisme de résolution SLD.

4.5.3 Planification dans le mode des blocs

Les deux exemples précédents ont montré que notre procédure avait des capacités que ne possédait pas celle de Missiaen, qui nous sert de référence, au moins pour la planification avec le calcul d'événements. Nous allons maintenant développer cet usage de la procédure en planification, en nous concentrant sur le « monde des blocs » comme domaine d'application.

Nous commençons par développer une formalisation du monde des blocs qui tire parti des capacités de notre procédure, et de la possibilité de prendre en compte des contraintes d'intégrité. Nous présentons ensuite une extension de la procédure par des contraintes sur domaine finis qui permet de planifier en gérant des ressources finies, comme des robots par exemple.

Formalisation du domaine des blocs

Le « monde des blocs » est un univers très simple où n'existent que des blocs que l'on peut empiler, des endroits (des tables) où poser ces blocs, et des robots pour effectuer le travail. Les actions et les propriétés grâce auxquelles nous allons décrire cet univers sont les mêmes que celles utilisées habituellement, en particulier par (Missiaen, 1991). Nous utilisons aussi le prédicat $Act/2$, qui décrit l'association entre un événement et l'action effectuée sur l'univers à l'occurence de cet événement. Les actions possibles sont :

put(X, Y, R) qui exprime que le robot R pose le bloc X sur le bloc Y ;

pick(X, R) qui exprime que le robot R attrape le bloc X ;

On écrira par exemple $\mathrm{Act}(e_1, put(a, b, rob_1))$ pour dire que l'action associée à l'événement e_1 est que le robot rob_1 pose le bloc a sur le bloc b. Pour décrire l'état du monde, nous utiliserons les propriétés suivantes [1] :

on(X, Y) signifie que le bloc X est posé sur le bloc Y ;

clasped(X, R) : le bloc X est dans la pince du robot R ;

1. Il serait possible de réunir en une seule action, par exemple $move(X, Y, Z, R)$, la succession des actions $pick(X, R)$ et $put(X, Z, R)$, mais une telle formulation ne permet pas de raisonner sur l'occupation du robot comme on souhaite le faire.

free(R) : le robot R est libre, c'est à dire que sa pince est vide ;

clear(X) : le bloc X n'a pas de bloc posé sur lui.

Ces propriétés sont débutées et terminées par les deux actions précédentes suivant les règles (2.1) à (2.8) de la figure 4.12. Afin de faciliter, dans un premier temps de manière simple mais pas très efficace, l'usage d'un nombre fini de robots, en jouant sur les différentes instanciations d'une règle, nous rajoutons l'antécédent Robot(r) à toutes les règles décrivant les conséquences des actions.

De manière générale, les prédicats supposables sont *Happens*/1 et *Act*/2, c'est à dire qu'un plan définit les occurrences des événements et le type d'action qui leur est associé.

Pour imposer en particulier un enchainement correct des actions entre *pick*() et *put*(), Une première solution est celle de Missiaen qui utilise les conditions de succès des événements de la manière suivante :

$$\begin{aligned}
&\text{Succeeds}(E) \leftarrow \\
&\qquad \text{Act}(E, put(X,Y,R)), \text{Holds_at}(clasped(X,R), date(E)), \\
&\qquad \text{Holds_at}(clear(Y), date(E)). \\
&\text{Succeeds}(E) \leftarrow \\
&\qquad \text{Act}(E, pick(X,R)), \text{Holds_at}(free(R), date(E)), \\
&\qquad \text{Holds_at}(clear(X), date(E)).
\end{aligned}$$

Son objectif est que lorsque l'on a sélectionné, puis supposé, l'occurrence d'un événement et sa liaison avec une action, l'appel du prédicat *Succeeds*/1 pour cet événement aboutisse à faire les hypothèses qui garantissent son succès. Pour un événement e dont l'action est $put(a,b,r)$, la preuve de Succeeds(e) va amener la génération, si elle est nécessaire, de l'occurrence d'un nouvel événement antérieur à e, et avec l'action $pick(a,r)$, afin de prouver Holds_at($clasped(a,r), date(e)$.

Cependant, une condition indispensable pour que ce schéma fonctionne est que les antécédents Act() dans les définitions de *Succeeds*/1 agissent comme des « gardes » pour le corps de ces clauses (Missiaen, 1991, pages 38 et 39), c'est à dire que l'appel de ces littéraux doit réussir sans qu'ils puissent être supposés, alors qu'ils peuvent l'être lorsqu'ils sont introduits par une clause définissant *Initiates*/2 ou *Terminates*/2.

Ce genre de contrainte nuit à la simplicité de la procédure et lie celle-ci à un formalisme particulier. Nous proposons une solution qui exploite la capacité à manipuler des contraintes d'intégrité de notre procédure. Avec ces contraintes, il est plus simple de décrire ces enchaînements obligatoires comme des contraintes d'intégrité sur les hypothèses de la forme Act(e,a). L'équivalent des clauses précédentes est réalisé à l'aide des contraintes d'intégrité (3.1) à (3.4) de la figure 4.12.

Lorsque l'hypothèse faite est Act($e, put(a,b,r_1)$), les deux premières contraintes d'intégrité font apparaître les buts suivants dans la dérivation de consistance :

$$\begin{aligned}
&\leftarrow \sim\text{Holds_at}(clasped(a,r_1), date(e)) \\
&\leftarrow \sim\text{Holds_at}(clear(b), date(e)
\end{aligned}$$

{ *Les conséquences des actions* }

(2.1) Initiates$(E, on(X,Y)) \leftarrow$ Act$(E, put(X,Y,R))$, Robot(R).
(2.2) Initiates$(E, clasped(X,R)) \leftarrow$ Act$(E, pick(X,R))$, Robot(R).
(2.3) Initiates$(E, free(R)) \leftarrow$ Act$(E, put(X,Y,R))$, Robot(R).
Initiates$(E, clear(X)) \leftarrow$
(2.4) Act$(E, pick(Y,R))$, Robot(R),
Holds_at$(on(Y,X), date(E))$.
(2.5) Terminates$(E, on(X,Y)) \leftarrow$ Act$(E, pick(X,R))$, Robot(R).
(2.6) Terminates$(E, clasped(X,R)) \leftarrow$ Act$(E, put(X,Y,R))$, Robot(R).
(2.7) Terminates$(E, free(R)) \leftarrow$ Act$(E, pick(X,R))$, Robot(R).
(2.8) Terminates$(E, clear(X)) \leftarrow$ Act$(E, put(Y,X,R))$, Robot(R).

{ *Les contraintes d'intégrité* }

(3.1) $\leftarrow$ Act$(E, put(X,Y,R))$, $\sim$Holds_at$(clasped(X,R), date(E))$.
(3.2) $\leftarrow$ Act$(E, put(X,Y,R))$, $\sim$Holds_at$(clear(Y), date(E))$.
(3.3) $\leftarrow$ Act$(E, pick(X,R))$, $\sim$Holds_at$(clear(X), date(E))$.
(3.4) $\leftarrow$ Act$(E, pick(X,R))$, $\sim$Holds_at$(free(R), date(E))$.

FIG. 4.12 - *Les clauses définissant le monde des blocs.*

Ces buts n'ont plus de variables, et sont donc traités par le cas C_5 de la dérivation de consistance, c'est a dire que l'on va en particulier essayer de prouver le but

$$\leftarrow \text{Holds\_at}(clasped(a, r_1), date(e))$$

par une dérivation abductive, ce qui va amener à faire l'hypothèse d'un événement dont l'action est $pick(a, r_1)$, avec une date d'occurrence antérieure à celle de e. De la même manière, les hypothèses de la forme Act$(e, pick(a,r))$ produisent les deux buts

$$\leftarrow \sim Holds\_at(free(r), date(e))$$
$$\leftarrow \sim Holds\_at(clear(a), date(e)$$

en tête de la dérivation de consistance.

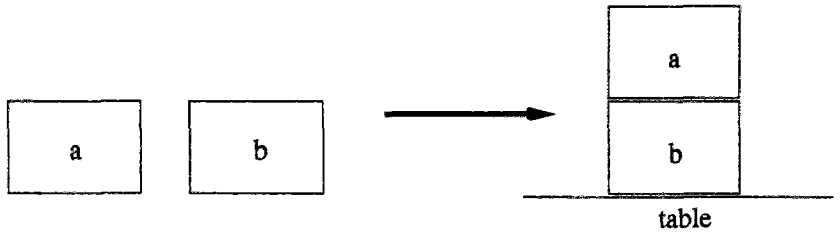
La solution que nous proposons à l'aide des contraintes d'intégrité est plus simple que celle proposée par Missiaen. En particulier, il n'est donc plus nécessaire de traiter différemment les littéraux $Act/2$ suivant la clause où ils apparaissent.

Comment pallier à la faiblesse de la persistance dans le calcul d'événements

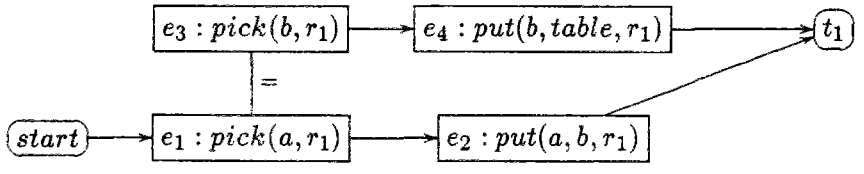
Cette formalisation du monde des blocs est malgré tout insuffisante. En particulier elle ne permet pas de s'affranchir du problème identifé par (Missiaen, 1991) concernant la validité des hypothèses de persistance dans toutes les extensions de

l'ordre partiel du plan généré. Indépendamment de l'usage pour la planification, ce problème a aussi été étudié par (Chittaro, Montanari, & Provetti, 1994) qui propose une reformulation de la persistance dans le calcul d'événements qu'il qualifie alors de « sceptique ».

Ce problème est par exemple illustré par l'exemple suivant. Soient deux blocs a et b posés côte à côte, donc libres, et que l'on désire empiler sur la table à l'aide d'un robot r_1 :

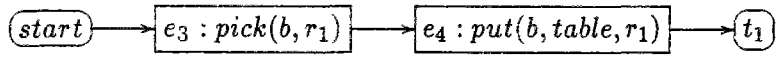


Le plan suivant est un plan pour lequel il existe une dérivation abductive en utilisant les clauses de la figure 4.12 :



Ce plan est bien évidemment incorrect du point de vue physique, pourtant il satisfait toutes les contraintes d'intégrité de la formalisation du domaine des blocs. Le problème est le suivant :

1. La procédure construit tout d'abord un plan partiel qui amène b sur la table :



2. Ensuite, les actions e_1 et e_2 nécessaires pour amener a au dessus de b sont construites et ordonnées entre elles par $date(start) < date(e_1) < date(e_2) < t_1$. Par contre, aucune contrainte n'est imposée entre la date de e_1 et d'autres événements du plan. Pour valider l'occurence de e_1, il faut que le robot r_1 soit libre, ce qui va être prouvée dans une dérivation abductive du but

$$\leftarrow \text{Holds\_at}(free(r_1), date(e_1))$$

Cette dérivation possède deux réponses, la première où l'événement initiateur de $free(r_1)$ est *start*, et la deuxième où c'est l'événement e_4 qui joue ce rôle. La première réponse avec l'événement *start* suppose que l'hypothèse de persistance

$$\sim\text{Clipped}(date(start), free(r_1), date(e_1))$$

soit vérifiée. La dérivation de consistance correspondante s'effectue ainsi :

$$\{ \leftarrow \text{Clipped}(date(start), free(r_1), date(e_1)) \}$$
$$\downarrow$$
$$\{ \square \parallel date(start) \leq date(e_3) < date(e_1) \}$$

et la clause vide obtenue a une restriction consistante, mais pas satisfaite. L'hypothèse de persistance est donc possible si l'on impose $date(e_1) \leq date(e_3)$, cette contrainte étant licite puisqu'aucune autre contrainte ne lie $date(e_1)$.

Ce problème vient de ce que la preuve de consistance de l'hypothèse de persistance dans la formulation initiale du calcul d'événements repose sur la complétion de l'ordre entre les événements. La solution retenue par Missiaen consiste en l'introduction du prédicat *Persists*/3 décrit précédemment au paragraphe 4.5.1 (page 120). Le but de cette modification étant, ramené à notre exemple, que si l'on ne peut pas prouver que e_3 est en dehors de l'intervalle $[date(start), date(e_1)[$, alors l'on considère que e_3 est dans cette période, ce qui invalide donc la première solution permettant de prouver $\text{Holds\_at}(free(r_1), date(e_1))$.

De la même manière le calcul d'événement « sceptique » de (Chittaro et al., 1994) définit ainsi la persistance :

$$\begin{aligned}\text{Clipped}(T_1, P, T_2) \leftarrow \\ &\text{Happens}(E),\ \text{Terminates}(E, P), \\ &\text{Succeeds}(E),\ \sim(date(E) < T_1),\ \sim(T_2 \leq date(E)).\end{aligned}$$

où la négation par échec est aussi utilisée sur les contraintes d'ordonancement entre événements. Dans cette formulation, et toujours en nous ramenant à notre exemple, si soit $date(e_3) < date(start)$, soit $date(e_1) \leq date(e_3)$ ne peut pas être prouvé, alors la réfutation de $\text{Clipped}(date(start), free(r_1), date(e_1))$ est un succès et l'hypothèse de persistance n'est pas acceptable.

Malheureusement, ces solutions ne sont pas transposables dans notre procédure, puisque nous traitons les relations d'ordre comme des contraintes. Si cette méthode nous permet de gagner en eficacité, par contrecoup nous perdons en souplesse puisque nous ne pouvons pas utiliser les contraintes avec une mécanisme comme la négation par échec. Lorsqu'une contrainte est présente dans le corps d'une clause, elle n'est pas réfutée, mais simplement ajoutée à celles qui sont héritées des pas de résolution antérieurs : seule l'inconsistance de ces contraintes peut produire un échec. Dans l'optique de la programmation logique avec contrainte, et plus encore dans la manière dont notre procédure fonctionne, il n'y a pas d'échec d'un but qui soit causé par la non satisfaction des contraintes : celles-ci sont des hypothèses potentielles pour valider la réfutation.

La seule solution adéquate compte tenu de notre procédure et de son fonctionnement est d'ajouter des contraintes d'intégrité supplémentaires pour éliminer les plans

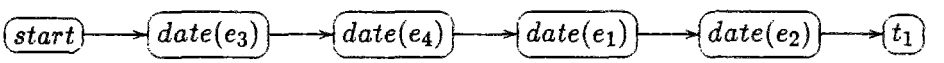
qui ne sont pas réalisables. En particulier, il suffit ici d'insérer entre les contraintes d'intégrité (3.3) et (3.4) la contrainte suivante :

$$\leftarrow \text{Act}(E, pick(X, R)),\ \text{Holds\_at}(clasped(Y, R), date(E))$$

où X et Y désignent des blocs et R un robot. L'insertion de cette contrainte à la position précitée va donc forcer l'apparition de la clause

$$\leftarrow \text{Holds\_at}(clasped(Y, r_1), date(e_1))$$

dans la dérivation de consistance de l'hypothèse $\text{Act}(e_1, pick(a, r_1))$. Soit ce but connait un échec fini si il n'existe pas d'autre hypothèses $\text{Act}(E, pick(Y, r_1))$ strictement antérieure à e_1, soit ce but va être résolu avec les actions d'événements comme e_3 avec le bloc b, pour lesquels la non-persistance de $clasped(b, r_1)$ entre $date(e_3)$ et $date(e_1)$ est obtenue avec les contraintes $date(e_3) \leq date(e_4) < date(e_1)$. La deuxième de ces contraintes correspond à un plan linéaire correct pour le problème initial :



Pour résumer, les contraintes d'intégrité portant sur les hypothèses de la forme $\text{Act}(e, pick(x, r))$ sont donc les suivantes :

(3.3) $\leftarrow \text{Act}(E, pick(X, R)),\ \sim\text{Holds\_at}(clear(X), date(E))$.
(3.4) $\leftarrow \text{Act}(E, pick(X, R)),\ \text{Holds\_at}(clasped(Y, R), date(E))$.
(3.5) $\leftarrow \text{Act}(E, pick(X, R)),\ \sim\text{Holds\_at}(free(R), date(E))$.

Résolution de problèmes de planification

Soit le problème de « l'anomalie de Sussman » (Sussman, 1975) illustré par la figure 4.13, et ainsi nommé par Sussman car son planificateur HACKER ne pouvait pas produire un plan pour cet exemple. Lorsque l'on dispose d'un seul robot comme c'est le cas dans cet exemple, il existe un plan linéaire qui permet d'atteindre le but final : on prend et on pose successivement les blocs c, b puis a.

Compte tenu des remarques effectuées au paragraphe 4.4, si l'on choisit une règle de sélection des littéraux dans l'ordre, avec retardement et sélection conjointe des litteraux supposables, la démarche suivie par la procédure pour établir un plan sera de supposer d'abord les actions $put(c, table, rob_1)$, $put(b, c, rob_1)$ et $put(a, b, rob_1)$, puis, dans l'ordre de celles-ci, de supposer les actions $pick()$ correspondantes. Cet exemple est aussi un cas où les preuves de consistance nécessitent la complétion de l'ordre entre les événements, et compte tenu de la solution que nous avons apportée à ce problème, notre procédure génere une solution correcte sous la forme d'un plan linéaire.

Supposons le même problème où l'on déclare disposer de deux robots pour effectuer le travail. On complète la description du problème par les clauses suivantes :

$$\text{Robot}(rob_1) \leftarrow .$$
$$\text{Robot}(rob_2) \leftarrow .$$

{ *La situation initiale* }
Initiates($start, on(c, a)$) $\leftarrow$.
Initiates($start, clear(c)$) $\leftarrow$.
Initiates($start, clear(b)$) $\leftarrow$.
Initiates($start, clear(table)$) $\leftarrow$.
Initiates($start, free(rob_1)$) $\leftarrow$.

{ *Le robot disponible* }
Robot(rob_1) $\leftarrow$.

{ *Le but à atteindre* }
$\leftarrow$ Holds_at($on(c, table), t_1$), Holds_at($on(b, c), t_1$), Holds_at($on(a, b), t_1$)

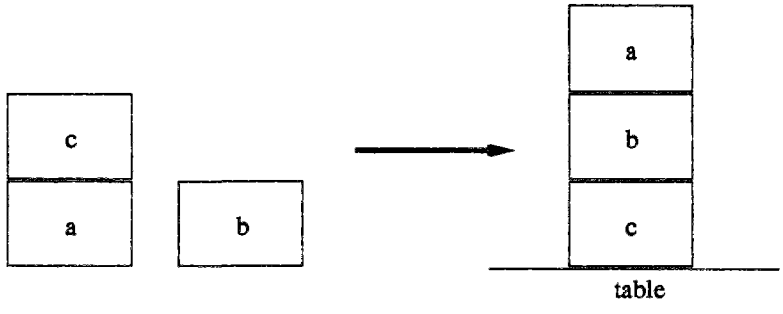


FIG. 4.13 - *Le problème de « l'anomalie de Sussman ». La situation initiale est décrite par le schéma de gauche, et le but final, à la date t_1, par celui de droite.*

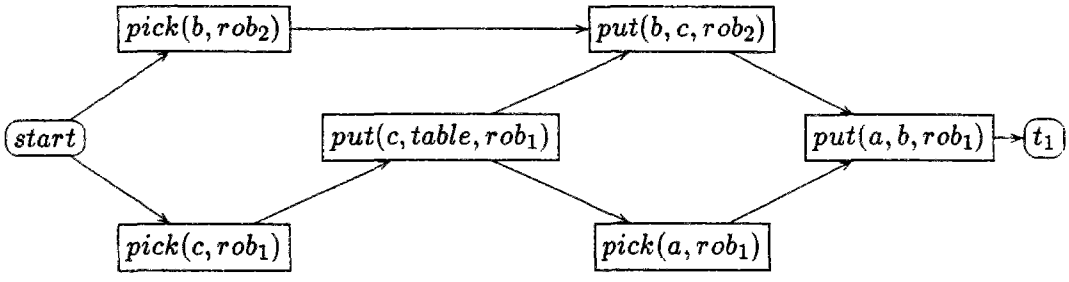


FIG. 4.14 - *Une solution à deux robots pour le problème de Sussman.*

et l'on ajoute les contraintes d'intégrité suivantes qui permettent d'éviter les conflits entre les robots :

$\leftarrow \text{Act}(E, put(X, Y, R_1))$, Holds_at$(clasped(Y, R_2), date(E))$, $R_1 \neq R_2$.
$\leftarrow \text{Act}(E, pick(X, R_1))$, Holds_at$(clasped(X, R_2), date(E))$, $R_1 \neq R_2$.

Ces contraintes excluent le cas où l'un des robots souhaiterait poser un bloc sur un bloc qui est tenu par un autre robot, et le cas où les deux robots souhaiteraient attraper le même bloc.

En forçant la dérivation abductive initiale à retourner plusieurs valeurs correspondant à différentes instanciations des antécédents Robot(R) dans les définitions des actions, la procédure retourne par exemple la solution qui apparaît sur la figure 4.14.

De la même manière, on peut n'imposer aucune limite sur les robots utilisés. La méthode la plus simple est de compter sur la skolémisation des variables lors de la génération des hypothèses pour créer ces robots. On complète simplement la description du problème de Sussman par la clause

$$\text{Robot}(R) \leftarrow .$$

La solution produite fait intervenir trois robots, nommés par les constantes de skolem r_1, r_2 et r_3. L'ordre entre les événements est celui de la figure 4.15

Une remarque sur les solutions à ce problème avec de multiples robots est que du point de vue du plan généré, les robots sont *a priori* interchangeables. Si dans la solution de la figure 4.14, l'on échange rob_1 et rob_2 le plan reste valide. Cependant, pour obtenir cette autre solution il faut relancer la procédure, et le travail de génération du deuxième plan sera identique au premier cas. Il serait donc agréable de disposer d'une solution unique et synthétique où deux robots anonymes (représentés par des constantes de skolem r_1 et r_2 par exemple) seraient utilisés pour décrire cette solution avec des contraintes comme

$$r_1 \in \{rob_1, rob_2\} \qquad r_2 \in \{rob_1, rob_2\} \qquad r_1 \neq r_2$$

Ces expressions rentrent dans le cadre des contraintes sur domaines finis. Tous comme les contraintes temporelles, il est possible d'envisager l'utilisation de telles

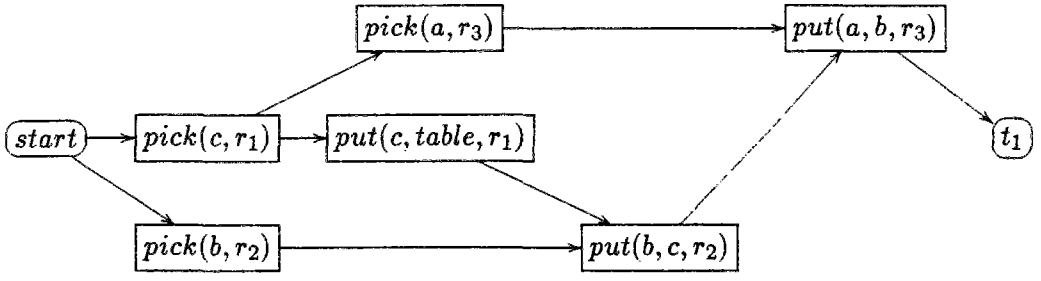


FIG. 4.15 - *Une solution à trois robots pour le problème de Sussman.*

contraintes dans une procédure de programmation logique avec contraintes. L'intégration de telles contraintes dans notre procédure abductive permettrait donc de raisonner avec des variables dont le domaine est fini, ce qui correspond à l'idée de ressources qui peuvent s'épuiser et dont il faut gérer l'affectation.

La combinaison de deux systèmes de contraintes demande que l'on étudie plusieurs points :

- la disponibilité d'algorithme de test de la consistance dans chacun des domaines ;
- la disponibilité d'un algorithme de test de la consistance sur un ensemble de contraintes de deux types différents ;
- dans le cas où un tel algorithme n'existepas, la possibilité de déterminer la consistance globale de toutes les contraintes à partir de la consistance pour chaque type de contraintes.

Nous allons maintenant étudier comment un tel système peut être mis en œuvre.

Les contraintes sur domaines finis et la gestion des ressources

L'extension aux contraintes sur domaines finis que nous allons présenter est volontairement très simple afin de rester utilisable. Les contraintes sur domaines finis n'ont pas d'aussi bonnes propriétés que les contraintes temporelles de l'algèbre d'instants restreinte, et, en particulier, la consistance est un problème NP, et ne peut être déterminée que par un processus d'énumération, eventuellement aidé par un filtrage préliminaire des domaines des variables.

Le langage des contraintes que nous allons utiliser est réduit au trois expressions suivantes :

$x \in \{s_1, \ldots, s_n\}$ décrit le domaine du terme x comme étant l'ensemble $\{s_1, \ldots, s_n\}$ où les s_i sont des termes sans variables ;

$x = y$ signifie que le terme x est égal au terme y ;

$x \neq y$ signifie que le terme x est différent de y.

Ces contraintes sont utilisées dans la partie restriction des clauses, tout comme les contraintes temporelles. La conjonction des contraintes lors du calcul des résolvants se fait en appliquant les règles suivantes :

$$\frac{(t \in S_1) \quad (t \in S_2)}{t \in (S_1 \cap S_2)} \qquad \frac{(t_1 \in S) \quad (t_1 = t_2)}{t_2 \in S}$$

qui permettent entre autre de n'avoir qu'une déclaration de domaine par terme. Si un terme est contraint à appartenir à un ensemble vide, on considère qu'il n'y a pas de résolvant car sa restriction est alors inconsistante.

Ce critère n'est pas suffisant pour décider de la consistance globale d'un ensembles de contraintes temporelles et de contraintes sur domaines finis. En particulier, la consistance de chacun des ensembles de contraintes, temporelles d'un côté, sur domaines finis de l'autre, n'est pas suffisante pour établir la consistance de l'ensemble total. Il peut y avoir des contraintes sur des termes à domaine finis, des robots par exemple, où une égalité entre deux de ces termes invalide les relations d'ordre, par ailleurs consistantes temporellement, entre les actions des robots concernés.

Ceci est une raison suffisante pour limiter la portée de ces contraintes sur domaine finis. En particulier, elles ne sont pas utilisées comme les contraintes temporelles pour produire des contraintes additionnelles pendant une dérivation de consistance (cas C_1). L'intérêt principal de ces contraintes est d'assister la phase de skolémisation des variables avant les tests de consistance des hypothèses.

En effet, créer une constante de skolem est équivalent à postuler l'existence d'un élément, qui ne sera contraint que par le fait qu'il vérifie l'hypothèse pour laquelle il est créé, et qui est *a priori* distinct de tout autre élément, sauf si une contrainte l'impose. Ceci est parfaitement réaliste dans le cas des instants, par exemple, qui sont supposés pris dans un ensemble infini et dense. Mais dans le cas de termes limités à un domaine fini, la skolémisation n'est pas toujours licite. Par exemple, si l'on ne dispose que de deux robots, en créer trois *a priori* distincts comme dans la solution de la figure 4.15 est impossible, et le plan correspondant n'est pas valide.

Pour implanter ce mécanisme, l'on étend la procédure abductive pour gérer un deuxième ensemble Δ_d de contraintes qui seront d'une des trois formes décrites précédemment. Lors d'une skolémisation d'une variable, si celle-ci n'est pas contrainte à un domaine fini dans la restriction de la clause, on lui affecte simplement une nouvelle constante de skolem. Par contre, si cette variable v a un domaine fini S, il faut lui substituer une constante dont l'existence soit acquise, c'est à dire que dès lors que l'on suppose que cette nouvelle constante est différente de toutes les autres, il faut que ces contraintes soient consistantes, c'est à dire qu'il existe une solution qui respecte toutes les contraintes d'inégalité et d'appartenance au domaine.

Lorsque les domaines sont tous identiques et de taille n, par exemple lorsque tous les robots peuvent *a priori* effectuer toutes les tâches, il suffit de s'assurer que l'on n'a pas créé plus de n constantes de skolem représentant des robots, et éventuellement de réutiliser des constantes de skolem déja créées. Par contre, lorsque les domaines ne sont pas identiques, il faut utiliser des méthodes plus générales. Ce type de problème et une comparaison entre différentes approches ont été étudiés par le projet BAHIA du (PRC-IA, 1992). Pour notre usage, nous proposons la méthode suivante lors de la skolémisation d'une variable associée à un domaine finie, pour déterminer à la fois la constante de skolem qui sera substituée à la variable v de domaine S, et le nouvel ensemble Δ_d :

1. l'on crée une nouvelle constante de skolem sk pour cette variable, et l'on initialise un ensemble C de contraintes avec la contrainte $sk \in S$ où S est le domaine de v ;
2. pour chaque constante de skolem sk' apparaissant dans Δ_d on ajoute dans C une contrainte $sk \neq sk'$;
3. si $C \cup \Delta_d$ est consistant, l'on ajoute C à Δ_d et on retourne sk ;
4. sinon, l'on effectue les actions suivantes :
   (a) on choisit dans C une inéquation $sk \neq sk'$, telle que
   $$\begin{cases} S \cap \mathcal{D}(sk') \neq \emptyset, \text{ et} \\ \Delta_d \cup (C - \{sk \neq sk'\}) \cup \{sk = sk'\} \text{ est consistant} \end{cases}$$
   où $\mathcal{D}(sk')$ est le domaine de sk' tel qu'il est induit par les contraintes de Δ_d ;
   (b) on restreint dans Δ_d le domaine de sk' à $S \cap \mathcal{D}(sk')$;
   (c) on retourne sk'.

La validité de cette méthode découle du résultat suivant. Nous notons sous la forme $\langle \{v_1, \ldots, v_n\}, D \rangle$ un graphe de contraintes où les domaines des variables sont $D(v_i)$ et qui contient toutes les contraintes $v_i \neq v_j$ pour $i \neq j$. Soit $G = \langle \{v_1, \ldots, v_n\}, D \rangle$ un tel graphe consistant, et v_{n+1} une variable de domaine S telle que le graphe

$$G' = \langle \{v_1, \ldots, v_n, v_{n+1}\}, D' \rangle \qquad \begin{cases} D'(v_i) = D(v_i) & 1 \leq i \leq n \\ D'(v_{n+1}) = S \end{cases}$$

soit inconsistant. Si V est l'ensemble de variables

$$\{v_i : 1 \leq i \leq n \text{ et } D'(v_i) \cap D'(v_{n+1}) \neq \emptyset\}$$

alors, il existe $v_k \in V$ tel que le graphe $G'' = \langle \{v_1, \ldots, v_n\}, D'' \rangle$ où les domaines D'' sont définis par

$$\begin{cases} D''(v_i) = D'(v_i) & 1 \leq i \leq n \text{ et } i \neq k \\ D''(v_k) = D'(v_k) \cap D'(v_{n+1}) \end{cases}$$

est consistant.

En effet, soit s une séquence de valeurs $(x_1, \ldots, x_n)$, solution de G. On sait que G'est inconsistant, c'est à dire que la séquence $s' = (x_1, \ldots, x_n, x_{n+1})$ où $x_{n+1} \in S$ est telle qu'il existe i, j distincts et tels que $x_i = x_j$. En fait, comme s est une solution de G, on peut affiner ceci en affirmant qu'il existe $k \in [1, n]$ tel que $x_k = x_{n+1}$. Soit alors G'' le graphe de contraintes $\langle \{v_1, \ldots, v_n\}, D'' \rangle$ où D'' est défini par

$$\begin{cases} D''(v_i) = D'(v_i) & 1 \leq i \leq n \text{ et } i \neq k \\ D''(v_k) = D'(v_k) \cap D'(v_{n+1}) \end{cases}$$

alors s est une solution de G'' parce que pour tous i, j distincts on a $x_i \neq x_j$, et $x_i \in D''(v_i) = D(v_i)$ pour $i \neq k$, et l'on sait que $x_k \in D(v_k)$ et $x_k \in S$, donc $x_k \in D''(v_k)$. Donc G'' est consistant.

L'application de cet algorithme de « réparation » produit un ensemble Δ_d qui est consistant, et dont la représentation sous forme de graphe de contraintes est un graphe complet.

Un inconvénient des contraintes sur domaines finis est le coût du test de la consistance d'un ensemble de contraintes. Les contraintes sont converties en un graphe dont les nœuds sont étiquetés par le domaine correspondant, les égalités étant représentées en fusionnant les nœuds et en prenant l'intersection des domaines correspondants. Ne restent dans le graphe que des contraintes d'inégalités, et par construction de Δ_d et de C dans l'algorithme précédent, le graphe correspondant est complet. La solution pour déterminer la consistance est d'utiliser un algorithme d'énumération avec *forward checking*, qui semble être ici une heuristique satisfaisante (Dechter & Meiri, 1989). Ce processus est bien sûr très coûteux, mais, en considérant que l'usage de ces contraintes est de gérer des ressources finies et peu nombreuses (parce que chères par exemple), le coût peut rester acceptable.

L'application de ce système pour la planification dans le monde des blocs se fait en définissant ainsi le prédicat *Robot*/1 :

$$\text{Robot}(R) \leftarrow R \in \{rob_1, rob_2\}$$

En adoptant une règle de sélection retardée et conjointe de tous les littéraux supposables, l'on obtient le plan de la figure 4.16, avec un ensemble Δ_d de contraintes qui est

$$r_1 \in \{rob_1, rob_2\} \qquad r_2 \in \{rob_1, rob_2\} \qquad r_1 \neq r_2$$

où r_1 et r_2 sont les constantes de skolem représentant les robots. Une autre solution dont la figure 4.14 est alors une instance est obtenue en considérant une autre inéquation lors de l'étape (a) de l'algorithme de skolémisation.

4.5.4 Remarques sur l'usage en planification

De manière générale, l'abduction est une solution possible au problème de la génération de plans non linéaires. La caractéristique des procédures abductives est

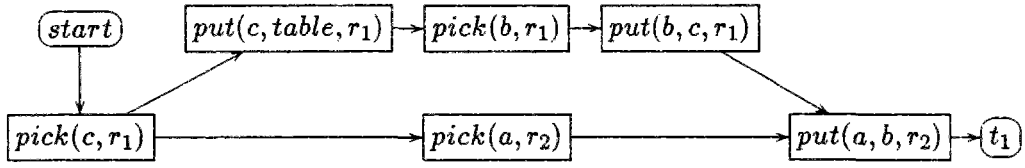


FIG. 4.16 - *Un plan à deux robots pour le problème de Sussman. Ce plan est obtenu à l'aide de contraintes sur domaines finis.*

qu'elles génèrent un ensemble minimal de contraintes nécessaire pour établir la consistance des hypothèses générées, et donc le respect des contraintes d'intégrité par les actions du plan, dans le cas de notre procédure.

Cependant, un des avantages de notre procédure est qu'elle est relativement indépendante du formalisme temporel utilisée. Ceci peut présenter quelques inconvénients, en particulier pour la planification avec un formalisme spécifique comme le calcul d'événements, car elle n'est pas optimisée pour ce formalisme comme la procédure de Missiaen. En pratique, il s'avère qu'elle ne supporte pas la comparaison en termes de temps de calcul avec celle de Missiaen dès que le problème de planification est de taille importante. En effet, comme cela a été entrevu précédemment, le fait que la procédure de Missiaen soit spécifiquement adaptée au Calcul d'Evénements lui permet d'imposer un ensemble de contraintes sur la forme des règles et des buts dont la réfutation permet d'obtenir un plan correct.

Missiaen a développé un système pour le contrôle de son planificateur abductif basé sur un critère de « localité », inspiré des travaux de (Lansky, 1986, 1988, 1990). Ce système lui permet donc de traiter avec succès des problèmes importants. Un tel travail reste à faire en ce qui concerne notre procédure : il est alors peu probable qu'il soit possible de rester indépendant du formalisme temporel.

Du point de vue des problèmes qui restent susceptibles de contribuer à l'amélioration de notre procédure pour son usage en planification, nous citerons en particulier les deux points suivants :

- la sélection conjointe de tous les littéraux pourrait se faire par sous-ensembles de ces littéraux. Est il possible d'identifier des heuristiques permettant de séparer les hypothèses en paquets indépendants ? Nous pensons que le travail de Missiaen sur la « localité » peut présenter un intérêt de ce point de vue ;

- la persistance telle qu'elle est réalisée par notre procédure souffre de quelques manques (cf. paragraphe 4.5.3). Nous avons vu que les solutions qui reformulent la définition de la persistance dans le calcul d'événements à l'aide de la négation par échec sur les relations d'ordre temporel ne sont pas applicables avec notre procédure. Existe-t'il une formulation générale ou des techniques de prise en compte des contraintes temnporelles qui apporteraient une solution à ce problème ? *A priori* l'introduction d'un opérateur modal **M** appliqué à des

contraintes temporelles comme le fait Dean dans le TMM (Dean & McDermott, 1987), et qui signifie la consistance de la contrainte avec Δ_t, est peut être une solution. La définition du prédicat *Clipped*/3 serait la suivante :

$$\begin{aligned}\text{Clipped}(T_1, P, T_2) \leftarrow \\ &\text{Happens}(E),\ \text{Terminates}(E, P), \\ &\text{Succeeds}(E) \ \|\ \mathbf{M}(T_1 \leq date(E)) \wedge \mathbf{M}(date(E) < T_2).\end{aligned}$$

ce qui invaliderait toute hypothèse de persistance dès lors que l'occurrence de l'événement E interrompant P serait consistante avec l'ordre (partiel) entre les événements. Cette solution doit être étudiée de manière plus approfondie.

4.6 Conclusions

Nous avons décrit dans ce chapitre une procédure abductive pour le raisonnement temporel basée sur la fusion entre les techniques de programmation logique avec contraintes et une procédure de preuve en programmation logique abductive. Notre procédure est en particulier utilisable pour des problèmes de planification, comme l'ont montré les exemples réalisés avec le calcul d'événements.

En dehors de l'implantation, qui fera l'objet du chapitre suivant, plusieurs points sont encore à étudier pour cette procédure :

- Du point de vue théorique, nous supputons que cette procédure est correcte par rapport à la sémantique des modèles stables (Gelfond & Lifschitz, 1988) lorsque celle-ci est adaptée aux programmes logiques avec contraintes, entre autre parce que la procédure qui nous sert de modèle, celle de Kakas et Mancarella, est déja correcte de ce point de vue. Nous n'avons actuellement pas la preuve de cette conjecture.

- De la même manière, la complétude de la procédure n'est pas prouvée. Il y a de bonnes raisons de penser qu'elle ne pourrait être complète que pour une classe limitée de programmes incluant les programmes localement stratifiés (plus exactement une adaptation de cette propriété aux programmes avec contraintes).

- De manière concrète, nous avons pu voir l'intérêt qu'il y avait à manipuler simultanément des contraintes de diverses formes, dans notre cas des contraintes temporelles et des contraintes sur domaines finis. Cette combinaison de différentes formes de contraintes ayant chacune des procédures de décision appropriées serait intéressant, en particulier pour les contraintes spatiales (Baykan & Fox, 1987). Le problème de la consistance globale déterminée à partir des consistances établies par chaque procédure spécialisée reste un point important à résoudre.

- L'étude sur la planification a permis d'identifier un problème lié à la prise en compte de la persistance par l'abduction. Ce problème avait déja été identifié dans le calcul d'événements par Missiaen et d'autres auteurs, dans d'autres domaines que la planification. La logique temporelle utilisée pour les deux premiers exemples d'utilisation de la procédure n'échappe pas non plus à ce problème, car sa définition de la persistance est analogque à celle du calcul d'événements. Des deux solutions proposées, seule la première qui consiste à rajouter à la description du domaine des contraintes d'intégrité *ad'hoc* a pu être testée. Nous espérons pouvoir étudier la deuxième de ces solutions qui suppose l'extension du langage de contraintes par un opérateur modal de consistance.

Chapitre 5

Implantation de la procédure abductive

Dans ce chapitre, nous étudions brièvement l'implantation de la procédure de preuve. Ce qui suit est dicté par les enseignements tirés de l'implantation d'un prototype de la procédure. Nous décrivons tout d'abord le cadre général de l'implantation, en particulier l'introduction du typage, puis nous présentons un algorithme incrémental et efficace pour la gestion de contraintes temporelles métriques. Pour finir, nous étudions la faisabilité d'une machine abstraite pour l'exécution de la procédure abductive.

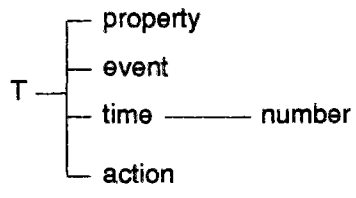
5.1 Cadre général de l'implantation

Il y a plusieurs possibilités pour l'implantation de la procédure abductive. Une première solution est d'en faire un méta-interprète au dessus de Prolog, comme c'est le cas pour le planificateur CHICA de Missiaen. Cette solution présente l'avantage d'utiliser un cadre où la résolution SLD est déja implantée, et où le parcours de l'espace de recherche est une notion naturelle. Par contre, Prolog souffre de sa trop grande simplicité, en particulier de son absence de système de typage natif et bien intégré. Nous avons pu voir, durant la description de la procédure, que beaucoup de notions devaient être adaptées à la sémantique des termes utilisés : par exemple, l'égalité ne s'interprète pas de la même manière entre deux instants, qu'entre deux événements, ou entre deux robots. Dans le premier cas, cette égalité signifie que les deux instants sont à la même position sur une droite temporelle, alors que dans le calcul d'événements, où chaque événement est représenté par un terme, l'égalité entre deux événements est équivalente à l'égalité (isomorphie) entre les deux termes.

L'introduction des types dans la programmation logique, et en particulier pour le langage Prolog, a reçu beaucoup d'attention et fait l'objet de nombreux travaux. On peut citer le système de (Mycroft & O'Keefe, 1984), dont (Lakshman & Reddy, 1991) présente une reconstruction formelle. L'inconvénient de beaucoup de ces sytèmes est l'absence de possibilités de sous-typage, car celle-ci détruit la propriété de « consistance des types ». Cette propriété signifie que l'application d'une règle d'inférence, la résolution SLD dans ce cas, à des expressions (clauses) bien formées produit une expression bien formée, ce qui dispense de la vérification des types lors de l'exécution (Deitrich & Hagl, 1988).

Il faut noter qu'une des motivations de H.J. Bürckert pour la mise au point du mécanisme général de la résolution avec contraintes était d'utiliser des systèmes comme KL-ONE (Brachman & Levesque, 1985) pour représenter les connaissances taxonomiques d'un domaine particulier, en parallèle avec des systèmes déductifs basés sur la résolution. Dans cette approche, les descriptions de typage, les relations entre les types des termes, sont des contraintes dont la satisfaisabilité est déterminée à l'aide d'un système à la KL-ONE.

Notre solution est plus proche de cette dernière que des extensions habituelles de la programmation logique par des types. De la même manière que Bürckert, la résolution SLD avec contraintes est utilisée aussi avec des contraintes de typage sur les termes des clauses. Chaque clause est donc associée à des contraintes temporelles, à des contraintes de typage et éventuellement à des contraintes de domaine. Ces contraintes sont combinées lors du calcul de résolvant et leur consistance est vérifiée en évaluant le type des termes et en le comparant à celui indiqué par la contrainte de typage. Il n'y a pas d'intéraction entre ce système de contraintes de typage et les contraintes temporelles, et la consistance globale est équivalente à la consistance de chaque type de contraintes, si l'on ne prend pas en compte les contraintes sur domaines finis.



Act : event × action
Clipped : time × property × time
Happens : event
Holds_at : property × time
Initiates : event × property
Succeeds : event
Terminates : event × property

date : event → time

FIG. 5.1 - *types et signatures pour le calcul d'événements.*

Nous définissons les informations taxonomiques à l'aide d'une hiérarchie de types, qui sont représentés par des symboles, et chaque symbole de fonction et de prédicat est associé à une signature. Les variables sont déclarées avec un type lors de l'écriture des clauses. Par exemple, pour le calcul d'événements, les types et les signatures sont définis sur la figure 5.1.

L'intérêt du typage est de pouvoir définir simplement la manière dont sont gérées les différentes contraintes simplement à partir du type des termes qui les composent. L'expression d'une contrainte de finitude du domaine peut être associée à un type et vient restreindre automatiquement toute variable déclarée de ce type, d'autres restrictions pouvant toujours être apportées au cours de la résolution. La résolution avec contraintes calcule les résolvants en réunissant les contraintes de chaque type provenant des clauses parentes dans la clause résolvante. Chaque forme de contrainte, temporelle, de typage, ou de domaine, est gérée à l'aide des algorithmes correspondants. Cependant, suivant les domaines d'interprétation mis en jeu, la consistance globale n'est pas forcément acquise dès lors que les consistance de chaque ensemble de contraintes est prouvée. En particulier, les contraintes sur domaine finis doivent être manipulées avec précaution (cf. chapitre 4, paragraphe 4.5).

5.2 Un algorithme efficace et incrémental pour la résolution de contraintes temporelles

Nous décrivons ici une étude réalisée dans le cadre d'un système de contraintes temporelles métriques entre instants en vue de son intégration dans un système déductif basé sur la procédure de preuve présentée au chapitre 4. L'objectif principal de ce travail est de fournir un ensemble complet de procédures permettant de :

1. décider de manière complète de la satisfiabilité d'un ensemble de contraintes temporelles ;
2. disposer d'un algorithme incrémental pour cette tâche ;

3. déterminer la relation minimale entre deux instants.

Le premier chapitre de ce mémoire a décrits plusieurs de ces systèmes, et nous en avons proposé des extensions dans le chapitre 2 pour les termes temporels avec constructeurs. Ici, notre motivation est principalement de prendre en compte l'information quantitative, en accordant un statut identique à tous les termes temporels utilisés, c'est à dire sans distinguer les variables, les constantes, les termes fonctionnels, etc.

En dehors des études sur CLP($\Re$), où les relations quantitatives peuvent être exprimées à l'aide d'équations et d'inéquations linéaires, et où les techniques de satisfaction sont basées sur l'élimination de Gauss et la méthode du Simplexe (Jaffar et al., 1992b), les seuls systèmes qui permettent l'expression de relations quantitatives sont basés sur des modèles analogues à celui des STP de (Dechter et al., 1991). Notre propos se place toujours dans ce cadre, mais nous proposons, sur la base d'une idée proposée dans (Dechter et al., 1991), un ensemble d'algorithmes réalisant les objectifs précités, pour un coût intéressant, puisque plus faibles que celui du plus connu des algorithmes applicables ici : PC-2.

5.2.1 Le cadre : contraintes métriques entre instants

Le cadre dans lequel nous nous plaçons est celui des contraintes métriques entre instants décrit par (Dechter et al., 1991) sous l'appellation de « Simple Temporal Problem » (STP). Ce cadre a déja fait l'objet d'une description dans le chapitre 1 de ce mémoire, aussi, nous ne rappellerons que l'essentiel.

Les contraintes manipulées dans ce formalisme lient chacune deux variables en bornant la distance entre ces deux variables. Entre deux variables x et y, la contrainte $[a, b]$ signifie $a \leq y - x \leq b$. Un ensemble de ces contraintes peut se représenter dans un graphe orienté dont les nœuds sont les variables et dont les arcs sont étiquetés par l'intervalle de la contrainte. Nous noterons un tel graphe par $G = (V, E)$ où V est l'ensemble des nœuds du graphe et E l'ensemble des arcs. La contrainte portée par l'arc (i, j) est notée T_{ij} et est donc un intervalle de la forme $[a, b]$: en pratique, il n'est pas nécessaire que l'arc inverse (j, i) soit présent dans E puisque l'on peut déterminer immédiatement T_{ji} comme étant l'intervalle $[-b, -a]$.

Un tel formalisme permet donc de représenter toutes les relations telles que :

- les symboles de relation sont $<$, $=$ ou $\leq$, respectivement traduits par les intervalles $]0, +\infty[$, $[0, 0]$ et $[0, +\infty[$;

- les instants, c'est à dire les variables, sont soit des termes « atomiques », soit de la forme $t+n$ ou $t-n$ où t est une variable et n est un nombre. Les contraintes entre éléments de cette dernière forme sont transformées à l'aide des règles

suivantes :

$$\begin{aligned}(t+n):[a,b]:t' &\rightarrow t:[a+n,b+n]:t'\\ (t-n):[a,b]:t' &\rightarrow t:[a-n,b-n]:t'\\ t:[a,b]:(t'+n) &\rightarrow t:[a-n,b-n]:t'\\ t:[a,b]:(t'-n) &\rightarrow t:[a+n,b+n]:t'\end{aligned}$$

Pour un graphe de contraintes $G = (V, E)$, on définit le *graphe des distances* comme un graphe orienté qui comporte le même ensemble V de nœuds, et tel qu'à chaque arc $[a, b]$ de x à y dans G correspond dans le graphe des distances un arc de x à y valué par b et un arc réciproque de y à x valué par $-a$ [1].

Ce domaine, au sens de la programmation logique avec contraintes (Jaffar & Maher, 1994), est un sous ensemble du domaine des équations et inéquations linéaires sur $\Re$. L'intérêt de cette remarque est que, dans un cas particulier, ce domaine vérifie la propriété dite « d'indépendance des contraintes négatives » (Lassez & McAloon, 1989), ce qui permet de traiter simplement les contraintes de la forme $t_1 \neq t_2$ comme cela a été expliqué au chapitre 2. Il est donc possible de transformer tout ensemble de contraintes positives ou négatives en un ensemble de contraintes où les seules contraintes négatives utilisent le symbole $=$. Pour cela, il suffit de réécrire les contraintes à l'aide des deux règles suivantes :

$$\begin{aligned}\neg(x < y) &\rightarrow y \leq x\\ \neg(x \leq y) &\rightarrow y < x\end{aligned}$$

Cette transformation seulement partielle des contraintes permet d'éviter la réécriture des contraintes $\neg(x = y)$ en $(x < y) \vee (y < x)$ qui donne potentiellement lieu à une explosion combinatoire, soit dans le nombre des clauses, soit lors du test de consistance des restrictions de celles-ci.

Une fois que l'on dispose d'un ensemble de contraintes où les seules contraintes négatives sont des négations d'égalités, l'inconsistance d'une contrainte $\neg(x = y)$ avec la contrainte positive (ou conjonction de contraintes positives) C se ramène à tester l'implication $C \Rightarrow (x = y)$. Comme cela a été établi précédemment, il suffit de déterminer la relation minimale entre x et y : si celle-ci est une égalité (intervalle $[0, 0]$ dans le formalisme qui nous intéresse) la contrainte est inconsistante avec C, sinon elle ne l'est pas.

5.2.2 Determination de la consistance

La première solution possible pour déterminer de manière complète la consistance est d'utiliser un algorithme de consistance de chemin par propagation de type PC-2 comme cela a été décrit dans le chapitre 1. Ce type d'algorithme a un coût en $O(n^3)$ où n est le nombre de nœuds du graphe. De plus, il est aussi possible, sur le modèle de (Loganantharaj et al., 1994) par exemple, d'en dériver une version incrémentale.

1. En général, il est inutile de construire le graphe des distances puisqu'il se déduit très simplement à partir du graphe de contraintes.

Un tel algorithme calcule aussi la relation minimale entre tous les nœuds et il est alors possible de la déterminer en temps constant par simple examen du réseau de contraintes produit.

Néanmoins, le coût en n^3 présente une croissance trop élevée pour des applications pratiques comme l'ont montré (Yampratoom & Allen, 1993). Il est donc intéressant de rechercher d'autres algorithmes permettant d'accomplir la même tache pour une complexité plus faible.

Il y a deux critères équivalents (Dechter et al., 1991) à la consistance globale d'un réseau de contraintes métriques :

1. le premier critère est que le graphe de contraintes ne doit avoir aucun circuit non valide, c'est à dire tel que la composition de tous les intervalles le long de ce circuit produit un intervalle ne contenant pas 0 ;
2. le deuxième critère est que le graphe des distances associé ne contienne aucun circuit de poids négatif. Ce deuxième critère est équivalent au premier.

Une première méthode plus efficace que PC-2 pour déterminer la consistance est d'examiner tous les cycles du graphe de contraintes ou de distances. La nécessité d'examiner *tous* les cycles, et pas seulement ceux dont on peut déterminer les éléments par recherche des composantes fortement connexes (algorithme linéaire en $O(|V|+|E|)$) fait perdre de son intérêt à cette solution. De plus, contrairement aux approches semblables dans l'algèbre d'instants (où les relations sont seulement symboliques) il n'est pas possible de détecter par ce moyen les égalités implicites entre instants.

Dechter (1991) donne un autre moyen de tester la consistance d'un ensemble de contraintes métriques. Celui-ci repose sur une condition de consistance de chemins plus faible que celle établie par l'algorithme PC-2 puisqu'elle repose sur un ordre entre les nœuds : cette condition, appelée *consistance de chemin directionnelle* est la suivante

Définition 5.1 (Consistance de chemins directionnelle) *Soit $G = (V, E)$ un graphe de contraintes métriques et $\prec$ un ordre sur les éléments de V. On note alors $V = \{1, \ldots, n\}$ suivant l'ordre $\prec$. Alors G vérifie la* consistance de chemins directionnelle, *que l'on notera $\prec$-dPC, si et seulement si pour tous nœuds i et j tels que $i \prec j$, et pour tout nœud k telque $i \prec k$ et $j \prec k$, on a :*

$$T_{ij} \subseteq T_{ik} \circ T_{kj}$$

où T_{ij} désigne l'intervalle de la contrainte de i à j et $\circ$ la composition de ces contraintes.

L'algorithme 5.1 permet d'établir cette forme de consistance. Dechter montre que cet algorithme est correct et complet pour déterminer la consistance globale

DPC(V, E) :
  pour $k = n$ a 1 **faire**
    pour tout (i, j) **tel que** $(i, k), (j, k) \in E$ et $i, j \prec k$ **faire**
      $T_{i,j} \leftarrow T_{i,j} \wedge (T_{i,k} \circ T_{k,j})$
      $E \leftarrow E \cup \{(i, j)\}$
      si $T_{ij} = \emptyset$ **alors retourner** (Echec)

Algorithme 5.1: *Algorithme DPC qui transforme un graphe (V, E) en un graphe qui satisfait la propriété de consistance de chemin directionnelle par rapport à l'ordre $\prec$. Pour faciliter la notation, les nœuds du graphe sont représentés par des entiers suivant l'ordre $\prec$.*

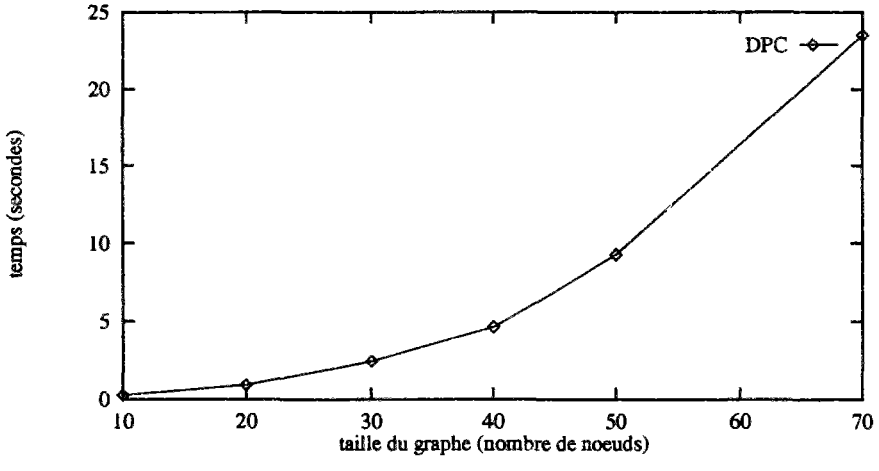


FIG. 5.2 - *Coût de la propagation totale par l'algorithme DPC en fonction du nombre de nœuds du graphe.*

d'un ensemble de contraintes. L'algorithme retourne « Echec » si un tel graphe est inconsistant sinon le graphe obtenu à la fin de l'algorithme est $\prec$-dPC.

Le premier intérêt de cet algorithme est que l'ordre $\prec$ est arbitraire. Le deuxième est sa complexité qui est linéaire en fonction de n : plus exactement, cette complexité est $O(nW^2)$ où W est le nombre maximal de parents qu'un nœud possède dans le graphe (en ne considérant que les arcs dans le sens de $\prec$). La figure 5.2 montre l'évolution du temps de calcul[2] en fonction de la taille du graphe[3].

L'algorithme DPC présente un inconvénient important : il n'est pas incrémental.

2. Les algorithmes ont été programmés avec Le_Lisp version 15.25, et les temps ont été mesurés sur le code interprété avec une station Sun IPC. Les chiffres présentés sont plus élevés que ce que donnerait une implémentation compilée puisque les tests ont montré que tous les algorithmes présentés ici étaient accélérés d'un facteur compris entre 20 et 25 par la compilation avec le compilateur modulaire Complice.

3. Tous les tests de cette partie ont été réalisés avec des graphes de contraintes consistants et générés aléatoirement.

C'est à dire que si l'on dispose de l'ensemble des contraintes élément par élément et que l'on est intéressé par la consistance après chaque ajout d'une contrainte, l'utilisation de DPC tel qu'il est donné par Dechter (1991) engendre un surcoût non négligeable car l'algorithme ne tient pas compte de la consistance directionnelle déjà établie lors des passages précédents.

Le problème est donc d'obtenir une version incrémentale de cet algorithme. Pour cela, nous allons d'abord détailler le fonctionnement de DPC. La boucle principale de cet algorithme examine une fois chaque nœud en descendant dans l'ordre $\prec$: chaque étape sur un nœud k établit la propriété $\prec$-dPC pour tous les triplets de nœuds (i, j, k) tels qu'il existe un arc de i à k et un arc de j à k dans le graphe. L'arc éventuellement créé ou modifié par l'opération de relaxation est (i, j) dont la valeur ne dépend alors que des contraintes des arcs (i, k) et (j, k). Ces observations permettent de concevoir un algorithme incrémental basé sur les principes suivants :

- l'algorithme doit fonctionner à partir d'un graphe vérifiant déja la propriété $\prec$-dPC ;
- lors de l'ajout ou de la modification d'une contrainte de l à m (que l'on peut sans perte de généralité supposer tels que $l \prec m$), tous les triplets (i, j, k) dont le nœud k est supérieur à m dans $\prec$ sont inchangés au regard de la propriété $\prec$-dPC. On peut donc envisager de ne commencer l'examen par la boucle principale qu'à partir du nœud m ;
- seuls les nœuds k qui sont à l'extrémité supérieure d'une contrainte précédemment modifiée (ou créée) par l'algorithme doivent être considérés dans la boucle principale ;
- de la même manière, seuls les parents de k tels que l'arc de ce parent à k a été modifié ou créé auront une influence sur le graphe.

Nous proposons l'algorithme 5.2 comme un algorithme incrémental pour rétablir la propriété $\prec$-dPC sur un graphe $G = (V, E)$ auquel on ajoute (ou restreint) une contrainte de i_1 à i_2. Cet algorithme utilise une file Q pour conserver les nœuds à partir desquels il faudra rétablir la propriété $\prec$-dPC. Le nœud à traiter k est choisi dans cette file comme le plus grand dans l'ordre $\prec$. Pour n'examiner que les parents de k concernés par les modifications précédentes, Π est une table qui retient pour chaque nœud j les nœuds i tels que la contrainte portée par l'arc (i, j) a été modifiée.

L'algorithme IDPC termine : en effet, lorsque l'on traite le nœud k, les seuls nœuds éventuellement rajoutés à la file Q sont strictement inférieurs à k dans l'ordre $\prec$ et de plus l'ensemble des nœuds est fini. La correction de IDPC découle de celle de l'algorithme DPC et des remarques faites précédemment. La complexité de IDPC peut être évaluée ainsi :

- la boucle principale est parcourue un nombre de fois inférieur ou égal au rang de k dans l'ordre $\prec$;

IDPC(i_1, i_2) :
  soit $Q \leftarrow \{i_2\}$; **soit** $\Pi[i_2] = \{i_1\}$
  $E \leftarrow E \cup \{(i_1, i_2)\}$
  tant que $Q \neq \emptyset$ **faire**
    soit $k \leftarrow \max_{\prec}(Q)$; $Q \leftarrow Q \setminus \{k\}$
    pour tout (i,j) **tel que** $\begin{cases} i \prec j \prec k \text{ et } (i,k),(j,k) \in E \\ \text{et (soit } i \in \Pi[k], \text{ soit } j \in \Pi[k]) \end{cases}$
      $T_{i,j} \leftarrow T_{i,j} \wedge (T_{i,k} \circ T_{k,j})$
      $E \leftarrow E \cup \{(i,j)\}$
      si $T_{ij} = \emptyset$ **alors retourner** (Echec)
      si T_{ij} est modifiee **alors**
        $Q \leftarrow Q \cup \{j\}$
        $\Pi[j] \leftarrow \Pi[j] \cup \{i\}$

Algorithme 5.2: *Algorithme IDPC qui ajoute la contrainte* (i_1, i_2) *(supposée telle que* $i_1 \prec i_2$*) au graphe* (V, E) *et qui propage celle-ci pour rétablir la consistance de chemin directionnelle par rapport à* $\prec$.

- le choix du nœud à traiter k et son extraction de la file Q peut se faire en un temps linéaire en fonction de la longueur de Q si celle-ci est implantée par une liste simple, ou mieux encore en un temps logarithmique si l'on utilise un *tas binaire* (« binary heap ») pour implanter Q ;
- si $W(k)$ est le nombre de parents de k, le nœud couramment traité par l'algorithme, l'opération de relaxation des arcs (i,j) est faite un nombre de fois inférieur où égal à $W(k)^2$;
- le test de la condition pour le choix des arcs (i,j) à relaxer peut être réalisé en temps constant si la structure Π est une table indexée par les nœuds (plus exactement par leur rang dans $\prec$) et que chaque élément de cette table est lui même un vecteur de booléens indiquant l'appartenance de chaque nœud à l'ensemble correspondant ;
- avec les choix de structure exposés plus haut, l'ajout d'un nœud à la file Q prend un temps constant si celle-ci est est représenté par une liste simple, ou un temps logarithmique dans le cas d'un tas binaire. L'ajout d'un nœud à une position de Π prend toujours un temps constant quelle que soit la structure choisie pour les éléments de Π.

Il ressort de ce qui précède que la complexité en pire cas de IDPC n'est pas meilleure que celle de DPC. En pratique cependant, le travail effectué par IDPC pour l'ajout d'une contrainte est largement inférieur à celui qu'effectue DPC pour le même travail. La figure 5.3 montre l'évolution du temps de calcul pour rétablir la propriété $\prec$-dPC après ajout d'une contrainte dans le graphe. Les contraintes ajoutées sont toutes

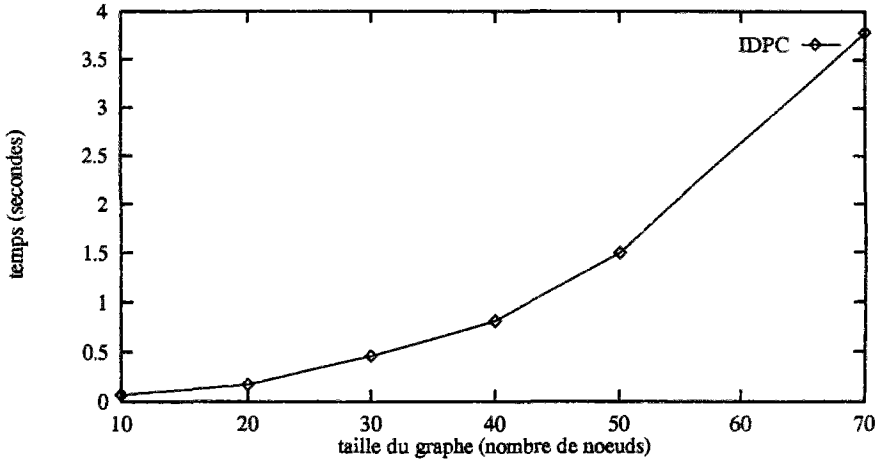


FIG. 5.3 - *Coût d'ajout et de propagation d'une contrainte par l'algorithme IDPC.*

tirées aléatoirement et sont connues pour être consistantes avec le graphe dans lequel elles sont insérées : ce cas est celui où l'algorithme de propagation effectue le plus de travail.

Pour le même travail, on peut estimer à partir de la courbe de la figure 5.2 le temps que prend l'algorithme DPC pour rétablir la propriété $\prec$-dPC : cette comparaison souligne bien le caractère incrémental de notre algorithme.

Dans le cadre d'un système basé sur la résolution avec contraintes, il est souvent nécessaire d'ajouter, non pas une, mais plusieurs contraintes à la fois. L'algorithme IDPC permet ceci : lorsque l'on souhaite ajouter à un graphe G un ensemble de contraintes dont les arcs sont $E' = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, il suffit d'initialiser la file Q de l'algorithme par $\{y_1, \ldots, y_m\}$ et d'initialiser $\Pi[y]$ par $\{x : (x, y) \in E'\}$ pour chaque $y \in Q$. Nous avons évalué la performance de IDPC pour ce type d'ajout. La figure 5.4 montre l'évolution du temps de propagation nécessaire pour rétablir la consistance de chemins directionnelle en fonction du nombre de contraintes ajoutées conjointement.

Ces derniers tests ont été réalisés sur des graphes consistants comportant vingt nœuds, connectés (comprenant donc au moins 19 arcs formant un arbre), et qui vérifiaient la propriété $\prec$-dPC. Les contraintes ajoutées étaient tirées aléatoirement et telles que les graphes finaux étaient toujours consistants. A titre de comparaison, l'agorithme DPC prend environ 4 secondes pour établir la consistance sur ces mêmes graphes. Ceci donne une indication sur la limite au delà de laquelle l'ajout incrémental et simultané de plusieurs contraintes devient équivalent à l'algorithme non incrémental : ici cette limite s'établit aux environs de 35 arcs pour 20 nœuds. Sur ces mêmes graphes, la propagation de la totalité des arcs initiaux par IDPC prend environ 5 secondes : ce qui correspond à un surcoût d'environ 25% par rapport à DPC pour la même tâche.

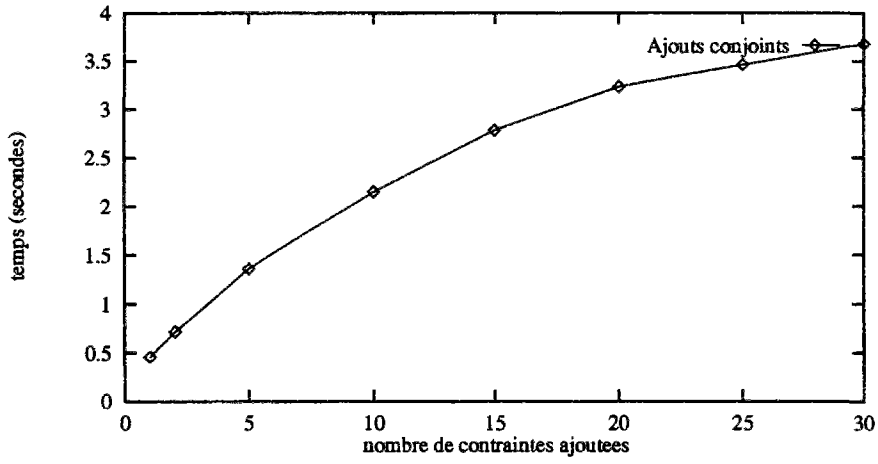


FIG. 5.4 - *Coût d'ajout et de propagation simultanée de plusieurs contraintes par l'algorithme IDPC.*

5.2.3 Détermination de la relation minimale

La relation minimale entre deux instants l et m est définie comme l'intervalle $[a, b]$ tel que a est l'opposé du poids $\delta(m, l)$ du chemin de poids minimal de m à l dans le graphe des distances du graphe de contraintes initial, et b est le poids $\delta(l, m)$ du chemin de poids minimal de l à m dans ce même graphe des distances.

Cette relation minimale peut donc être calculée par deux recherches de chemin minimal dans le graphe des distances initial. Un algorithme adéquat est par exemple celui de Bellman et Ford dont la complexité est en $O(n|E|)$. En pratique, l'on dispose d'un graphe qui vérifie la propriété de consistance de chemins directionnelle et pour lequel on peut montrer la résultat suivant sur les chemins de poids minimal :

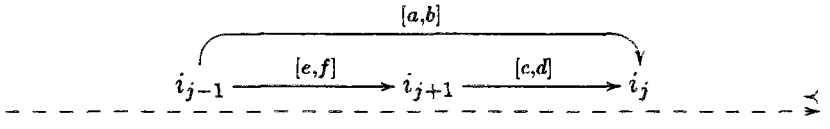
Théorème 5.1 *Soit $G = (V, E)$ un graphe de contraintes métriques consistant, et $G_d = (V, E_d)$ le graphe $\prec$-dPC obtenu par application de l'algorithme DPC sur G. Pour tous nœuds l et m, si $\delta(l, m)$ est le poids du chemin minimal de l à m dans le graphe des distances de G, alors il existe un chemin minimal de l à m dans le graphe des distances de G_d de même poids d, noté $(l, i_1, \ldots, i_k, m)$, et qui vérifie la propriété suivante :*

(i) aucun nœud i_j sur ce chemin n'est tel que $i_{j-1} \prec i_j$ et $i_{j+1} \prec i_j$.

où i_{j-1} est le prédécesseur direct de i_j sur le chemin minimal et i_{j+1} son successeur direct.

Preuve : Il suffit de montrer que, lorsqu'il existe un chemin minimal de l à m dans G[4], et qu'il ne satisfait pas la propriété (i), alors l'application de l'algorithme DPC sur G construit un chemin minimal de même poids dans le graphe des distances de G_d et tel que ce chemin vérifie alors la propriété citée.

Supposons que le chemin minimal dans G ne vérifie pas la propriété (i). Alors il existe au moins un nœud qui est supérieur (dans l'ordre $\prec$) à ses deux voisins directs dans le chemin. Soit i_j le plus grand de ces nœuds. Ce nœud i_j est tel que son prédécesseur direct i_{j-1} sur le chemin vérifie $i_{j-1} \prec i_j$ et son successeur direct i_{j+1} vérifie $i_{j+1} \prec i_j$:



Lorsque le nœud i_j est traité par l'algorithme DPC, la phase de relaxation va créer, s'il n'existe pas déja, un arc (i_{j-1}, i_{j+1}) étiqueté par

$$T_{i_{j-1},i_{j+1}} \wedge T_{i_{j-1},i_j} \circ T_{i_j,i_{j+1}} \quad = \quad [e,f] \cap [a-d, b-c]$$

et la contribution de cet arc au poids du chemin $(l, \ldots, i_{j-1}, i_{j+1}, \ldots, m)$ est :

$$\min(f, b-c)$$

Comme le chemin $(l, \ldots, i_{j-1}, i_j, i_{j+1}, \ldots, m)$ est minimal, on vérifie :

$$\min(f, b-c) = b-c$$

On a donc un nouveau chemin minimal de l à m de la forme $(l, \ldots, i_{j-1}, i_{j+1}, \ldots, m)$ et dont le poids est identique au chemin minimal initial. L'application de DPC va ainsi supprimer dans le chemin minimal tous les nœuds i_j qui sont supérieurs dans $\prec$ à leurs deux voisins directs dans le chemin minimal. L'ensemble des nœuds étant fini, et ceux-ci étant traités par ordre descendant de $\prec$, l'on est donc assuré que le chemin minimal dans le graphe final G_d vérifie la propriété (i). □

Un corollaire immédiat de ce résultat est le suivant :

Corollaire 5.1 *Soit G un graphe de contraintes métriques consistant, et G_d le graphe obtenu après application de l'algorithme DPC. Alors, pour tous nœuds l et m de ce graphe, le chemin minimal de l à m dans G_d vérifie que pour tout nœud i_j de ce chemin (autres que l et m), on a $i_j \prec \max_{\prec}(l, m)$.*

Ce résultat permet de calculer la relation minimale par deux recherches de chemin minimal qui, compte tenu du résultat précédent, peuvent se faire plus efficacement que par l'algorithme de Bellman et Ford, puisque la propagation des poids peut

4. Par abus de langage et pour alléger l'écriture, nous appellerons « chemin minimal dans G », le chemin de poids minimal dans le graphe des distances de G.

```
Min-path(s, t):
    pour tout u ∈ V faire d(u) ← ∞
    d(s) ← 0
    pour tout u = {s, ..., 1} faire
        pour tout (u, v) ∈ E tel que v ≺ u faire
            d(v) ← min(d(v), d(u) + W(u, v))
    pour tout u = {1, ..., t} faire
        pour tout (u, v) ∈ E tel que u ≺ v faire
            d(v) ← min(d(v), d(u) + W(u, v))
    retourner (d(t))
```

Algorithme 5.3: *Algorithme de calcul du poids $\delta(s,t)$ du chemin minimal du nœud s au noeud t dans le graphe des distances (V,E) d'un graphe de contraintes satisfaisant la propriété de consistance de chemin directionnelle. Les nœuds du graphe sont notés $\{1,\ldots,n\}$ dans l'ordre de $\prec$.*

se faire suivant certaines directions privilégiées. En effet, le résultat montre que le chemin minimal de s à t, notons le $(s, i_1, \ldots, i_k, t)$, ne passe que par des nœuds inférieurs dans $\prec$ à $\max_{\prec}(s,t)$ et que chacun de ces nœuds i_j (exceptés les nœuds extrémités s et t) vérifie une des trois conditions suivantes :

(1) $i_{j+1} \prec i_j \prec i_{j-1}$;

(2) $i_j \prec i_{j-1}$ et $i_j \prec i_{j+1}$, et un tel nœud, s'il existe, est unique ;

(3) $i_{j-1} \prec i_j \prec i_{j+1}$;

Nous proposons l'algorithme 5.3 qui est construit à partir d'une variante, par Yen, de l'algorithme de Bellman et Ford.

Après initialisation, la première boucle de l'algorithme propage les distances depuis s pour tous les nœuds inférieurs à s dans l'ordre $\prec$ et atteignables depuis s par un chemin qui ne comprend que des arcs dans le sens inverse de $\prec$. A la fin de cette première boucle, tous les nœuds $u \in \{1,\ldots,s\}$ tels que tous les nœuds d'un chemin minimal de s à u vérifient le cas (1) de la remarque précédente, sont étiquetés par $d(u) = \delta(s,u)$. La deuxième boucle prend en compte les cas (2) et (3) et propage ces distances à partir de ces nœuds vers les nœuds supérieurs dans l'ordre. Compte tenu de ceci, à la fin des deux boucles, tous les nœuds $u = 1,\ldots,t$ sont étiquetés par $\delta(s,u)$, et donc l'algorithme retourne $\delta(s,t)$.

La complexité de cet algorithme est linéaire en fonction de n et son coût est inférieur en pratique à celui de l'algorithme de Bellman et Ford puisque, pour un nœud donné, l'on ne considère à chaque fois que les arcs qui sont, soit dans le sens de $\prec$, soit dans le sens inverse. La figure 5.5 montre l'évolution du coût du calcul de

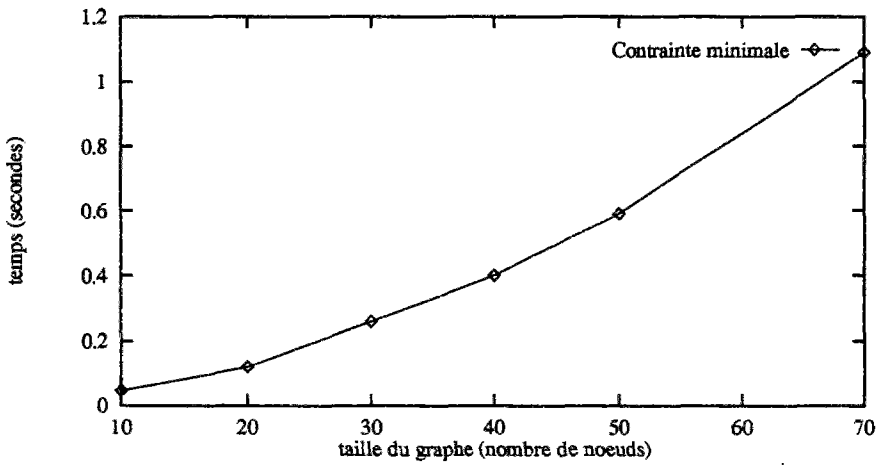


FIG. 5.5 - *Coût de la recherche de la contrainte minimale dans un graphe $\prec$-dPC.*

la contrainte minimale (c'est à dire deux recherches successives du chemin minimal) par cet algorithme en fonction de la taille du graphe.

5.3 Architecture d'une machine abstraite

Dans l'histoire de la programmation logique, la WAM (Warren Abstract Machine) (Warren, 1983) a joué un rôle primordial dans la diffusion et l'intérêt qui a été porté à la programmation logique, et plus particulièrement à Prolog. La WAM est une machine abstraite, dont la structure est particulièrement bien adaptée à l'exécution des programmes Prolog. Ces programmes sont compilés en une suite d'instructions qui réalisent l'enchaînement des buts, tout en construisant dans une zone de mémoire spéciale les termes qui seront donnés comme réponse pour les valeurs des variables du but initial. La simplicité de l'exécution de Prolog — la règle de sélection dans l'ordre, le parcours en profondeur d'abord et le retour arrière chronologique — permet donc cette compilation et les gains de temps en exécution sont particulièrement importants.

De plus, la WAM a ensuite été étendue pour prendre en compte différentes extensions de la programmation logique, en particulier la programmation logique avec contraintes avec CLP($\mathcal{R}$) (Jaffar, Michaylov, Stuckey, & Yap, 1992a), ou encore sur domaines finis avec CLP(fd) (Diaz & Codognet, 1993). On peut citer aussi la résolution SLG implantée sur une extension de la WAM (Swift & Warren, 1994).

Dans tous les cas le bénéfice de l'exécution sur une machine abstraite a compensé la rigidité introduite par la compilation et l'exécution déterministe des buts. Ces remarques motivent donc l'intérêt que nous portons à cette méthode d'implantation pour notre procédure abductive, malgré sa complexité plus grande que celle de la simple résolution SLDNF de Prolog.

Nous commençons par donner une description de la WAM originale, description qui est inspirée par celle de (Aït-Kaci, 1990), puis nous essayons d'identifier les raisons pour lesquelles la WAM permet un tel gain dans l'exécution des programmes logiques. Nous décrivons ensuite la manière dont cette architecture est étendue à la programmation logique avec contraintes. Pour finir, nous proposerons les grandes lignes d'une extension de la WAM pour la programmation logique abductive et l'implantation de notre procédure.

5.3.1 Qu'est ce que la WAM?

La WAM est une machine dont la structure générale apparaît sur la figure 5.6. Elle possède plusieurs registres spécialisés (P, CP, H, etc.), un ensemble extensible de registres dits « d'arguments », et plusieurs zones de données dont les rôles sont les suivants :

- La zone de code contient les instructions résultant de la compilation du programme logique. Chaque prédicat défini par une clause correspond à une adresse dans cette zone. Le registre P pointe sur la prochaine instruction à exécuter et CP retient l'adresse de l'instruction à exécuter après un appel réussi à un prédicat ;

- Le tas est une zone de données où sont construits dynamiquement des termes au cours de l'exécution du programme, c'est en particulier dans cette zone que sont construits les termes retournés comme réponses à une requête. Le registre H pointe sur la prochaine adresse à utiliser pour créer une structure dans cette zone. S et HB sont utilisés pendant l'exécution du programme.

- La pile est utilisée pour sauvegarder au cours de l'exécution les points de choix (pointés à l'aide du registre B) et les valeurs des registres arguments et CP avant qu'un appel à une procédure ne soit effectué.

- Le *trail* est une zone qui fonctionne comme une pile pour retenir les adresses des variables qui doivent être déliées lors des retours arrière.

- la zone PDL (*Push Down List*) est une pile utilisée par l'algorithme d'unification comme zone temporaire pour stocker des données.

L'idée principale de la WAM est qu'une clause qui définit un prédicat p/n peut être vue comme une procédure à n arguments qui lie les variables de ceux-ci à des termes construits dynamiquement dans la zone du tas. Ainsi, une clause de la forme

$$p_0(\ldots) \leftarrow p_1(\ldots), \ldots, p_n(\ldots).$$

est compilée en une suite d'instructions de la forme

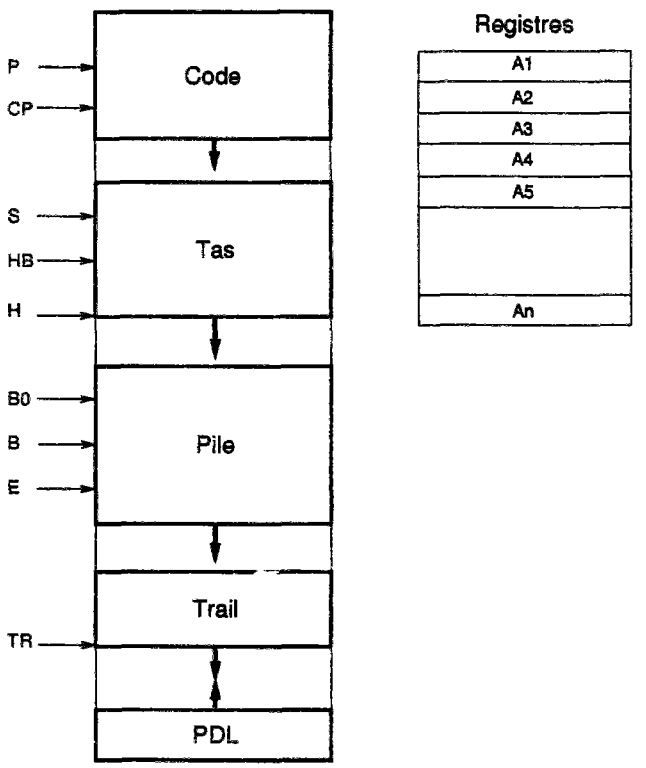


FIG. 5.6 - *Structure de la WAM.*

p_0 : `allocate`
obtenir les arguments de p_0
ranger les arguments de p_1
`call` p_1
⋮
ranger les arguments de p_n
`call` p_n
`deallocate`

Le principe du passage des arguments est d'utiliser les registres A_1, A_2, etc. pour cette tâche, en allouant ces registres dans l'ordre des arguments. L'expression « obtenir les arguments » signifie récupérer les contenus de ces registres arguments, soit par l'unification, soit sous la forme d'une liaison d'une variable si l'argument correspondant est une variable, soit en construisant une structure dans le tas matérialisant le terme argument si celui-ci est complexe (fonctionnel). L'expression « ranger les arguments » signifie placer dans les registres arguments la valeur des arguments du prédicat, qui est ensuite appelé par l'instruction `call`.

La représentation des termes dans le tas se fait comme un tableau de cellules de deux types différents : STR et REF, et complétées chacune d'une adresse dans ce tas. Les cellules STR représentent les termes complexes de la forme $f(t_1, \ldots, t_n)$: l'adresse placée dans la cellule est celle où sont stockés le nom et l'arité du symbole de fonction (ici f/n) et après cette adresse viennent les arguments du terme dans les n cellules suivantes. Les cellules REF représentent les variables, et elles ne sont pas instanciées si l'adresse associée à la cellule REF est celle de la cellule elle-même ; sinon la variable est liée (instanciée) par le terme pointé par l'adresse placée dans la cellule. Les variables identiques dans un terme sont regroupées en une seule cellule REF, c'est à dire que la forme générale de représentation des termes est celle de graphes acycliques et orientés (DAG).

Une requête, qui est un terme, est compilée en une suite d'instructions spécifiques qui construisent la structure du terme sur le tas, obtiennent les arguments par unification, placent ceux-ci dans les registres arguments, puis appellent le code de chaque prédicat de la requête. L'effet de l'exécution est de construire la liaison des variables de la requête à des termes construits dynamiquement sur le tas. Les différentes informations sauvegardées sur la pile retiennent les points de choix à l'intérieur d'un même prédicat (plusieurs clauses pour définir un prédicat), et les valeurs des registres.

5.3.2 Pourquoi la WAM est elle efficace?

Une première explication de l'efficacité de la WAM est que la résolution SLD est instanciée dans Prolog avec des règles de sélection et de choix qui sont très simples : on prend les littéraux dans l'ordre dans lequel ils apparaissent, on essaie les clauses dans l'ordre où elles sont données, etc. La structure de l'espace de recherche est celle

d'un arbre parcouru en profondeur d'abord, ce qui est très simple à faire sur une machine qui possède au moins une pile. La structure du code généré pour la WAM à partir d'un programme logique, et son exécution, suivent fidèlement cet ordre de parcours.

De plus, des instructions spécifiques sont prévues pour gérer des structures de données courantes dans les programmes Prolog comme les listes. L'unification est aussi optimisée : elle évite l'*occur-check* et utilise la structure des termes sous forme de DAG, ce qui lui donne une complexité très favorable, qui peut devenir linéaire (Baader & Siekmann, 1993).

5.3.3 Les extensions de la WAM pour la programmation logique avec contraintes

Les principes de fonctionnement sont identiques à ceux de la WAM originale. Une zone est ajoutée pour collecter les contraintes, couplée avec les algorithmes permettant de vérifier la consistance de celles-ci. Les contraintes apparaissant dans le corps des clauses sont compilées en des instructions spécifiques qui ajoutent des contraintes dans la zone concernée. Après chaque ajout des éléments d'une contrainte, la consistance est vérifiée, et un retour arrière est déclenché si celle-ci n'est pas obtenue (Jaffar et al., 1992a).

La méthodologie d'écriture des clauses impose que les contraintes soient placées en tête du corps de celles-ci, ce qui permet de les placer dans la zone concernée dès le début de l'exécution du corps de la clause. Les échecs par inconsistance des contraintes sont alors immédiatement détectés.

Par contre, les littéraux ordinaires des clauses sont compilés comme dans la WAM originale, et le passage d'arguments se réalise toujours par unification et à l'aide des registres arguments. L'opération d'unification détecte les types des variables et des termes en jeu, suivant qu'il s'agit de variables qui n'apparaissent que dans les contraintes, ou que dans les littéraux, ou encore dans les deux à la fois à l'intérieur du corps d'une clause.

Cette technique d'extension de la WAM est suffisament simple, dans son principe, à défaut de l'être dans sa réalisation, pour que son usage dans une machine abstraite adaptée à la procédure abductive soit possible. Nous allons maintenant esquisser les grandes lignes d'une telle machine.

5.3.4 Extension de la WAM pour la procédure abductive

Afin d'exécuter la procédure abductive, il faut noter les avantages et les inconvénients que les principes de la WAM imposent à l'implantation de notre procédure. Une première remarque est que la WAM est assez « rigide » : les règles de sélection sont simples et non adaptables, et le programme est une donnée statique qui n'est

pas modifiable à l'exécution. *A contrario* notre procédure suppose de pouvoir traiter différemment les littéraux suivants qu'ils sont supposables ou non, suivant que l'on retarde leur sélection, ou qu'on les sélectionne tous ensemble.

Malgré tout, toutes les dérivations se font à l'aide du même programme si l'on trouve le moyen de gérer les littéraux supposables et les hypothèses sans en faire un programme. Ceci est un avantage qui peut être utilisé.

Une solution serait de s'inspirer des techniques déja utilisées dans le compilateurs de Prolog sur la WAM, où des buts spécifiques, comme $op/2$ qui permet de définir de nouveaux opérateurs et leurs règles d'association, permettent d'influer sur la résolution. Ici, la déclaration des littéraux supposables pourrait se faire par un mécanisme analogue, tout comme la règle de sélection à utiliser, et diriger la compilation du programme en conséquence :

- les littéraux non supposables sont compilés de manière habituelle :
- suivant la règle de sélection, soit les littéraux sont réordonnées dans le corps des clauses (si l'on a une règle de sélection retardée), soit ils restent à leur place ;
- le code généré pour chaque littéral supposable tente de résoudre celui-ci avec les hypothèses existantes, et en cas d'échec de prouver leur consistance en tant qu'hypothèse.

Une nécessité est de représenter l'ensemble des hypothèses Δ de manière dynamique, ce qui nécessite une zone spécifique dont l'organisation serait analogue au tas de la WAM. Il faut alors implanter la résolution entre un littéral supposable et les hypothèses par une instruction spécifique de la WAM.

La phase de résolution avec les contraintes d'intégrité peut être préparée lors de la compilation puisque les littéraux supposables sont connus lors de cette phase. Pour une hypothèse de prédicat h/n, le code généré enchainerait les motifs suivants :

```
h:  allocate
    obtenir les arguments de h
    ranger les arguments de a1
    call a1
    ⋮
    ranger les arguments de an
    call an
    deallocate
```

pour chaque contrainte d'intégrité de la forme

$$\leftarrow a_1, \ldots, h, \ldots, a_n$$

Globalement, il faut aussi que la machine fonctionne suivant deux modes, l'un pour les dérivations abductives où la génération d'une clause vide (c'est à dire

l'exécution complète d'un appel sans échec) retourne les liaisons des variables, les contraintes et l'ensemble courant d'hypothèses, et un autre mode pour les dérivations de consistance, où l'exécution complète d'un appel déclenche l'équivalent du cas C_1 de la dérivation de consistance, et où l'occurrence d'un échec force la continuation de l'exécution sur la clause suivante parmi celles générées entre l'hypothèse et les contraintes d'intégrité.

5.4 Conclusions

Dans ce chapitre, nous avons étudié différents éléments de l'implantation de la procédure abductive décrite au chapitre 4 de ce mémoire. Nous avons choisi un cadre de programmation logique typée qui s'est révélé à l'usage suffisamment souple pour implanter et étendre notre procédure. L'introduction du typage s'est en particulier révélé une aide précieuse lors de l'ajout de différents systèmes de contraintes, en particulier les contraintes sur domaine finis qui ont été exposées dans le paragraphe 4.5 du chapitre 4.

Les idées présentées dans la dernière partie de ce chapitre sur l'architecture d'une machine abstraite ne sont que le résultat d'une étude prospective. Elles restent à valider de manière concrète. Nous pensons qu'une telle implantation, malgré la rigidité qu'elle introduit dans la procédure, peut être utile pour une classe assez large de problèmes. Les exemples en planification avec le calcul d'événements présentés dans le chapitre 4 montrent que l'on peut se contenter de règles de sélection simples et prédictibles et obtenir malgré tout des résultats significatifs.

Conclusions, perspectives

Nous avons exposé dans ce mémoire notre contribution au raisonnement temporel en intelligence artificielle.

Le premier objectif a été d'identifier quelques raisons qui limitaient la mise en œuvre du raisonnement temporel à l'aide de langages comme les logiques temporelles réifiées. Nous avons en particulier montré que le principe très général de la résolution avec contraintes fournissait une base d'analyse et un cadre formel pour l'intégration des systèmes de gestion de contraintes temporelles dans les systèmes déductifs basés sur le principe de résolution.

Si certaines considérations sur les sémantiques des logiques temporelles limitent pour le moment l'intérêt d'une approche inspirée de la démonstration de théorèmes, nous avons développé une application de la résolution avec contraintes pour l'abduction en raisonnement temporel. L'usage du raisonnement abductif permet de résoudre certains problèmes comme la persistance, et aussi de proposer une autre approche pour la planification temporelle. Notre contribution sur ce deuxième aspect est inspiré de travaux en programmation logique abductive, et a débouché sur une procédure de preuve qui étend les possibilités de la programmation logique abductive, en intégrant des techniques de la programmation logique avec contraintes.

L'usage de cette procédure abductive pour quelques problèmes de planification a permis de montrer que si les actions du plan comme les persistances nécessaires à l'accomplissement de celui-ci pouvaient être générés comme des hypothèses, il était nécessaire d'accorder un soin particulier aux hypothèses de persistance surtout si le plan généré était non linéaire, ce qui est *a priori* le cas dans la planification par abduction.

Cette procédure abductive peut être implantée relativement aisément, et nous avons développé quelques aspects de cette implantation dans le dernier chapitre de ce mémoire.

Les points qui pourraient faire l'objet de recherches supplémentaires sont les suivants :

- La correction de la procédure : conformément à l'habitude en programmation logique, cette correction se vérifie par rapport à une sémantique donnée des programmes logiques. Ici, il semble que partir de la correction de la procédure

de Kakas et Mancarella, qui sert de modèle à la nôtre, permettrait d'arriver à ce résultat.

- Du point de vue de l'usage de notre procédure, il manque un moyen de contrôler le parcours de l'espace de recherche, et de limiter la combinatoire possible lorsqu'on choisit les contraintes temporelles permettant de rétablir la consistance de l'explication produite.

- Dans le cadre d'un usage en planification non linéaire, il semble nécessaire que les hypothèses de persistance soient traitées avec attention. Existe t'il d'autres moyens en plus des deux que nous avons cités au chapitre correspondant, à savoir contraindre les solution par des contraintes d'intégrité supplémentaires et introduire un opérateur modal de consistance dans le langage des contraintes temporelles ?

Toujours dans le cadre d'une utilisation en planification, cette procédure est *a priori* capable de compléter un plan partiel, puisque cela correspond à lui fournir en entrée un ensemble d'hypothèses et de contraintes temporelles non vide. Cette possibilité pourrait être utilisée pour raffiner ou corriger un plan dont on ne fournirait que les grandes lignes.

De la même manière, l'introduction d'un système permettant de gérer la granularité et le changement d'echelle dans la représentation temporelle (Kuipers, 1987) pourrait fournir un moyen de contrôle de la recherche, en limitant le niveau de détail de l'explication générée, ainsi qu'en offrant la possibilité de focaliser la recherche à une échelle particulière.

Bibliographie

Aït-Kaci, H. (1990). The WAM: A (real) tutorial. Tech. rep. 5, DEC Paris Research Laboratory, Paris, France.

Allen, J. F. (1991). Planning as temporal reasoning. In Allen, J., Fikes, R., & Sandewall, E. (Eds.), *Principles of Knowledge Representation and Reasoning: Proc. of the Second International Conference (KR'91)*, pp. 3–14. Kaufmann, San Mateo, CA.

Allen, J. F., & Hayes, P. J. (1985). A common-sense theory of time. In *Proc. of the 9 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 528–531 Los Angeles, CA.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, 26*(11), 832–843.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence, 23*(2), 123–154.

Ayeb, B. E., Marquis, P., & Rusinowitch, M. (1990). A new diagnosis approach by deduction and abduction. In Gottlob, G., & Nejdl, W. (Eds.), *Expert Systems in Engineering: Principles and Applications: Proc. of the International Workshop, Vienna, Austria*, Vol. 462 of *Lecture Notes in Computer Science*, pp. 32–46. Springer, Berlin, Heidelberg.

Baader, F., & Siekmann, J. (1993). Unification theory. In Gabbay, D., Hogger, C., & Robinson, J. (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, Oxford, UK.

Bacchus, F., Tenenberg, J., & Koomen, J. (1991). A non-reified temporal logic. *Artificial Intelligence, 52*, 87–108.

Barringer, H., Fisher, M., Gabbay, D., & Hunter, A. (1991). Meta-reasonning in executable temporal logic. In *Principles of Knowledge Representation and Reasoning: Proc. of the Second International Conference (KR'91)*, pp. 40–49 San Mateo, CA.

Barth, P. (1992). CLP($\mathcal{PB}$): A meta-interpeter in CLP($\mathcal{R}$). Research report MPI-I-92-233, Max Planck Institute.

Baykan, C., & Fox, M. (1987). An investigation of opportunistic constraint satisfaction in space planning. In *Proc. of the 10 th Int. Joint Conference on Artificial Intelligence (IJCAI)* Milan, Italy.

Borillo, M., & Gaume, B. (1990). Une extension cognitive du calcul d'évènemts de Kowalski et Sergot et son application au raisonnement spatio-temporel. *Revue d'intelligence artificielle*, *4*, 7–26.

Brachman, R., & Levesque, H. (Eds.). (1985). *Readings in Knowledge Representation.* Morgan Kaufmann.

Bürckert, H.-J. (1991). *A Resolution Principle for a Logic with Restricted Quantifiers*, Vol. 568 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin Heidelberg.

Bylander, T., Allemang, D., Tanner, M. C., & Josephson, J. R. (1991). The computational complexity of abduction. *Artificial Intelligence*, *49*, 25–60.

Caprotti, O. (1993). Extending RISC-CLP(Real) to Handle Symbolic Functions. In Miola, A. (Ed.), *DISCO '93: International Symposium on Design and Implementation of Symbolic Computation Systems*, Vol. 722 of *Lecture Notes in Computer Science*, pp. 241–255 Gmunden, Austria. Springer-Verlag.

Cavedon, L., & Lloyd, J. (1989). A completeness theorem for SLDNF resolution. *Journal of Logic Programming*, *6*(7), 177–191.

Chan, D. (1988). Constructive negation based on the completed database. In *Logic Programming: Proc. of the Sixth International Conference and Symposium*, pp. 111–125.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, *32*, 333–378.

Chittaro, L., Montanari, A., & Provetti, A. (1994). Skeptical and credulous event calculi for supporting modal queries. In *Proc. of the 11 th ECAI*, pp. 361–365 Amsterdam.

Clark, K. (1978). Negation as failure. In Gallaire, H., & Minker, J. (Eds.), *Logic and Databases*, pp. 293–322. Plenum Press.

Cox, P., & Pietrzykowsky, T. (1986). Causes for events: Their computation and applications. In *Proc. of the 8th International Conference on Automated Deduction (CADE 86)*, Vol. 230 of *Lecture Notes in Artificial Intelligence*, pp. 608–621.

Cox, P., & Pietrzykowsky, T. (1987). General diagnosis by abductive inference. In *Proc. of Symposium on Logic Programing*, pp. 183–189.

Davis, E. (1987). Constraint propagation with interval labels. *Artificial Intelligence*, *32*(3), 281–331.

Dean, T. L., & Boddy, M. (1988). Reasoning about partially ordered events. *Artificial Intelligence, 36*, 375–399.

Dean, T. L., & McDermott, D. V. (1987). Temporal data base management. *Artificial Intelligence, 32*(1), 1–55.

Dechter, R., & Meiri, I. (1989). Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proc. of the 11 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 271–277 Detroit, MI.

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraints networks. *Artificial Intelligence, 49*, 61–95.

Deitrich, R., & Hagl, F. (1988). A polymorphic type system with subtypes for Prolog. In Ganzinger, H. (Ed.), *Proc. of 2nd European Symposium on Programming (ESOP)*, Vol. 300 of *Lecture Notes in Computer Science*, pp. 79–93.

Denecker, M., & De Schreye, D. (1992). SLDNFA: an abductive procedure for normal abductive programs. In *Proc. of the Joint International Conference and Symposium on Logic Programming*, pp. 686–700 Washington.

Diaz, D., & Codognet, P. (1993). A minimal extension of the wam for CLP(FD). In *Proc. of the 10th International Conference on Logic Programming.*

Dousson, C., Gaborit, P., & Ghallab, M. (1993). Situation recognition. In *Proc. of the 13 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 166–172 Chambery, France.

Dowling, W., & Gallier, J. (1984). Linear time algorithms for testing the satisfiability of propositional Horn formulæ. *The Journal of Logic Programming, 1*(3), 267–284.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence, 12*, 231–272.

Enjalbert, P., & Cerro, L. F. D. (1989). Modal resolution in clausal form. *Theoretical Computer Science, 65*, 1–33.

Eshghi, K., & Kowalski, R. A. (1989). Abduction compared with negation by failure. In Levi, G., & Martelli, M. (Eds.), *Logic Programming: Proc. of the Sixth International Conference*, pp. 234–254. MIT Press, Cambridge, MA.

Eshghi, K. (1988). Abductive planning with event calculus. In *Proc. of the 5th Int. Conf. on Logic Programming*, pp. 562–579.

Eshghi, K. (1993). A tractable class of abduction problems. In *Proc. of the 13 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3–8 Chambery, France.

Evans, C. (1990). The Macro Event Calculus: Representing temporal granularity. Tech. rep., Imperial College, Dept of Computing, Logic Programming Group.

Fann, K. T. (1970). *Peirce's Theory of Abduction.* Nijhoff, The Hague, Holland.

Fisher, M. (1991). A resolution method for temporal logic. In *Proc. of the 12 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 99–104 Sidney, Australia.

Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence, 54*, 199–227.

Fusaoka, A., Seki, H., & Takahashi, K. (1983). A description and reasoning of plant controllers in temporal logic. In *Proc. of the 8 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 405–408 Karlsruhe, FRG.

Galton, A. (1990). A critical examination of Allen's theory of action and time. *Artificial Intelligence, 42*(1), 159–188.

Galton, A. (1991a). A critique of Yoav Shoham's theory of causal reasoning. In *Proc. of AAAI-91*, pp. 355–359.

Galton, A. (1991b). Reified temporal theories and how to unreify them. In *Proc. of the 12 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1177–1182 Sidney, Australia.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proc. of ICLP'88*, pp. 1070–1080.

Gerevini, A., & Schubert, L. (1993). Efficient temporal reasoning through timegraphs. In *Proc. of the 13 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 648–654 Chambery, France.

Gerevini, A., & Schubert, L. (1994). On computing the minimal labels in time point algebra networks. Technical report 495, Univ. of Rochester, Computer Science Dept.

Ghallab, M., & Mounir-Alaoui, A. (1989). Managing efficiently temporal relations through indexed spanning trees. In *Proc. of the 11 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1297–1303 Detroit, MI.

Granier, T. (1988). Contribution à l'étude du temps objectif dans le raisonnement. Rapport lifia RR 716-I-73, LIFIA.

Groß, E., Hertzberg, J., Materne, S., & Voß, H. (1992). On clipping persistence (or whatever must be clipped) in time maps. In *ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation*, pp. 139–148 Pisa, Italy.

Halpern, J., & Shoham, Y. (1986). A propositional modal logic of time intervals. In *Proc. of Logic in Computer Science (LICS)*, pp. 279–292.

Hanks, S., & McDermott, D. V. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence, 33*, 379–412.

Hansson, C. (1990). A prototype system for logical reasoning about time and action. thesis, Linköping University.

Hartshorne, C., & Weiss, P. (Eds.). (1958). *Collected Papers of Charles Sanders Peirce*. Harvard Universty Press.

Haugh, B. A. (1987a). Non-standard semantics for the method of temporal arguments. In *Proc. of the 10 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 449–455 Milan, Italy.

Haugh, B. A. (1987b). Simple causal minimizations for temporal persistence and projection. In *Proc. of AAAI-87*, pp. 218–223 Seattle, WA.

Heintze, N. C., Jaffar, J., Michaylov, S., Stuckey, P. J., & Yap, R. H. (1992). *The CLP($\mathcal{R}$) Programmer's Manual.*

Hertzberg, J. (1994). Theoretical planning and its contributions to practical and applied planning. In *Proc. of the 11 th ECAI*, pp. 811–812 Amsterdam.

Höhfeld, M., & Smolka, G. (1988). Definite relations over constraint languages. Lilog report 53, IBM Deutschland, Stuttgart, Germany.

Hsiang, J. (1987). Rewrite method for theorem proving in first order theory with equality. *Journal of Symbolic Computation*, *3*(1), 133–151.

Jaffar, J., & Lassez, J.-L. (1987). Constraint logic programming. In *Proc. of the 14th ACM Symposium of the Principles of Programming Languages*, pp. 111–119.

Jaffar, J., & Maher, M. J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming*, *??*

Jaffar, J., Michaylov, S., Stuckey, P., & Yap, R. (1992a). An abstract machine for CLP($\mathcal{R}$). In *Proceedings ACM SIGPLAN Symposium on Programming Language Design and Implementation (PLDI)*, pp. 128–139 San Francisco.

Jaffar, J., Michaylov, S., Stuckey, P., & Yap, R. (1992b). The CLP($\mathcal{R}$) language and system. *ACM Transactions on Programming Languages and Systems*, *14*(3), 339–395.

Kakas, A. C., Kowalski, R., & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770.

Kakas, A. C., & Mancarella, P. (1990a). Database updates through abduction. In *Proc. of the 16th Conference on Very Large Databases.*

Kakas, A. C., & Mancarella, P. (1990b). Generalized stable models: A semantics for abduction. In *Proc. of the 9 th ECAI*, pp. 385–391 Stockholm.

Kakas, A. C., & Mancarella, P. (1990c). On the relation between truth maintenance and abduction. In *Proc. of PRICAI'90*, pp. 438–443.

Kakas, A. C., & Mancarella, P. (1993). Constructive abduction in logic programming. Compulog deliverable report, University of Cyprus.

Kautz, H. A. (1986). The logic of persistence. In *Proc. of AAAI-86*, p. 401 Philadelphia, PA.

Kautz, H. A., & Ladkin, P. B. (1991). Integrating metric and qualitative temporal reasoning. In *Proc. of AAAI-91*, pp. 241–246.

Koomen, J. (1989). The TIMELOGIC temporal reasoning system. Tech. rep. 231, University of Rochester.

Koubarakis, M. (1992). Dense time and temporal constraints with $\neq$. In *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, pp. 24–35.

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, *4*, 67–95.

Kuipers, B. (1987). Abstraction by time-scale in qualitative simulation. In *Proc. of AAAI-87*, pp. 621–625 Seattle, WA.

Kunen, K. (1989). Signed data dependencies in logic programs. *Journal of Logic Programming*, *6*(7), 231–246.

Lakshman, T., & Reddy, U. S. (1991). Typed prolog: A semantic reconstruction of the mycroft-o'keefe type system. Tech. rep., University of Illinois at Urbana-Champaign. Also in Proc. of ILPS'91.

Lansky, A. L. (1986). A representation of parallel activity based on events, structure, and causality. In Georgeff, M. P., & Lansky, A. L. (Eds.), *Reasoning about actions and plans : proceedings of the 1986 Workshop*, pp. 123–159 Los Altos, CA.

Lansky, A. L. (1988). Localized event-based reasoning for multiagent domains. *Computational Intelligence*, *4*(4), 319–340.

Lansky, A. L. (1990). Localized search for controlling automated reasoning. In *Proc. of the 1990 DARPA Planning Workshop*, pp. 115–125 San Diego, CA.

Lassez, J.-L., & McAloon, K. (1989). Independence of negative constraints. In *TAPSOFT'89 Int. Joint Conference on Theory and Practice of Software Development*, Vol. 351 of *Lecture Notes in Computer Science*, pp. 19–27 Barcelona, Spain.

Lassez, J.-L., & McAloon, K. (1992). A canonical form for generalized linear constraints. *Journal of Symbolic Computation*, *13*(1), 1–24.

Ligozat, G. (1990). Weak representations of interval algebras. In *Proc. of AAAI-90*, pp. 715–720 Boston, MA.

Ligozat, G. (1994). Towards a general characterization of conceptual neighborhoods in temporal and spatial reasoning. In *Proc. of the AAAI-94 Workshop on Spatial and Temporal Reasoning*, pp. 55–59 Seattle, Washington, USA.

Lloyd, J. (1987). *Foundations of Logic Programming* (second edition). Springer-Verlag, Berlin Heidelberg.

Loganantharaj, R., Mitra, D., & Giambrone, S. (1994). An efficient algorithm for incremental temporal constraint propagation. In *Proc. of the Seventh Florida Artificial Intelligence Research Symposium*, pp. 342–346 Pensacola. FLAIRS.

Mackinson, D. (1990). Logique modale : Quelques jalons essentiels. In Iturrioz, L., & Dussauchoy, A. (Eds.), *Modèles logiques et systèmes d'Intelligence Artificielle.* Hermès.

Mackworth, A. K., & Freuder, E. C. (1985). The complexity of some polynomial network consistency algorithms for Constraint Satisfaction Problems. *Artificial Intelligence, 25*, 65–74.

Manna, Z., & Pnuelli, A. (1981). Verification of concurrent programs, part 1: The temporal framework. Report STAN-CS-81-836, Stanford University.

Marquis, P. (1991a). Extending abduction from propositional to first-order logic. In Jorrand, P., & Kelemen, J. (Eds.), *Fundamentals of Artificial Intelligence Research: International Workshop FAIR'91 Proc.*, Vol. 535 of *Lecture Notes in Computer Science*, pp. 141–155. Springer, Berlin, Heidelberg.

Marquis, P. (1991b). *Contribution à l'étude des méthodes de construction d'hypothèses en intelligence artificielle.* Ph.D. thesis, Université de Nancy 1.

McCune, W. W. (1990). OTTER 2.0 users guide. Tech. rep. ANL-90/9, Argonne National Laboratory.

McDermott, D. V. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science, 6*, 101–155.

Meiri, I. (1991). Combining qualitative and quantitative constraints in temporal reasoning. In *Proc. of AAAI-91*, pp. 260–267.

Missiaen, L. (1991). *Localized Abductive Planning with the Event Calculus.* Ph.D. thesis, K.U. Leuven.

Missiaen, L., Bruynooghe, M., & Denecker, M. (1993). CHICA, an abductive planning system based on event calculus. Draft, Department of Computing, K.U. Leuven.

Mohr, R., & Henderson, T. (1986). Arc and path consistency revisited. *Artificial Intelligence, 28*, 225–233.

Montanari, U. (1974). Networks of constraints: fundamental properties and applications to picture processing. *Information Science*, *7*, 95–132.

Morris, P. H. (1988). The anomalous extension problem in default reasoning. *Artificial Intelligence*, *35*, 383–399.

Mounir-Alaoui, A. (1990). *Raisonnement temporel pour la planification et la reconnaissance de situations.* Ph.D. thesis, Université Paul Sabatier, Toulouse, France.

Mycroft, A., & O'Keefe, R. (1984). A polymorphic type system for prolog. *Artificial Intelligence*, *??*, 295–307.

Nebel, B., & Bäckström, C. (1991). On the computational complexity of temporal projection and some related problems. Research report RR-91-34, German Center for Artificial Intelligence (DFKI).

Nebel, B., & Bürckert, H.-J. (1993). Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. Tech. rep. RR-93-11, DFKI.

Nef, F. (1990). Quelques systèmes de logique temporelle propositionnelle. In Iturrioz, L., & Dussauchoy, A. (Eds.), *Modèles logiques et systèmes d'Intelligence Artificielle.* Hermès.

Ohlbach, H.-J. (1988). A resolution calculus for modal logics. *Lecture Notes in Computer Science*, *310*, 500–516.

Panitz, S. E. (1993). Default reasoning with a constrained resolution principle. In *Int. Conf. on Logic Programming and Automated Reasoning (LPAR'93)*, pp. 265–276 St. Petersbourg.

Paul, G. (1993). Approaches to abductive reasoning: an overview. *Artificial Intelligence Review*, *7*, 109–152.

Poole, D. (1988). A logical framework for default reasoning. *Artificial Intelligence*, *36*(1), 27–47.

Pople, H. E. (1973). On the mechanization of abductive logic. In *Proc. of the 3 rd Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 147–151 Stanford, MA.

PRC-IA (Ed.). (1992). *Actes des quatrièmes journées nationales PRC-IA*, Marseille. Teknea.

Przymusinski, T. C. (1989a). On constructive negation in logic programming. In *Proc. of The North American Conf. On Logic Programming* Cleveland. preprint.

Przymusinski, T. C. (1989b). On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, *5*, 167–205.

Reeves, S. (1987). Semantic tableaux as a framework for automated theorem-proving. In Hallam, J., & Mellish, C. (Eds.), *Advances in Artifical Intelligence: Proc. of the 1987 AISB Conference*, pp. 125–139 Edinburgh.

Reichgelt, H. (1989). A comparison of first-order and modal logic of time. In Jackson, P., & van Harmelen, H. (Eds.), *Logic-based Knowledge Representation*, pp. 143–176. MIT Press.

Reiter, R. (1978). On closed world data bases. In Gallaire, H., & Minker, J. (Eds.), *Logic and Data Bases*, pp. 55–76. Plenum Press, New York.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence, 13*, 81–132.

Ribeiro, C., & Porto, A. (1991). Maximal intervals: an approach to temporal reasoning. In *Proc. of EPIA '91*, Vol. 541 of *Lecture Notes in Artificial Intelligence*, pp. 180–194.

Sadri, F., & Kowalski, R. A. (1988). A theorem proving approach to database integrity. In Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufman.

Sandewall, E. (1988a). Non-monotonic entailment for reasoning about time and action Part I: Sequential actions. Research report LiTH-IDA-R-88-27, Linköping University.

Sandewall, E. (1988b). Non-monotonic entailment for reasoning about time and action Part II: Concurrent actions. Research report LiTH-IDA-R-88-28, Linköping University.

Sandewall, E. (1988c). Non-monotonic entailment for reasoning about time and action Part III: Decision procedure. Research report LiTH-IDA-R-88-29, Linköping University.

Satoh, K., & Iwayama, N. (1991). Computing abduction by using the TMS. In Furukawa, K. (Ed.), *Logic Programming: Proc. of the Eighth International Conference*, pp. 505–518. MIT Press, Cambridge, MA.

Satoh, K., & Iwayama, N. (1992). A query evaluation method for abductive logic programming. In *Proc. of the Joint International Conference and Symposium on Logic Programming*, pp. 671–685 Washington.

Selman, B., & Levesque, H. J. (1990). Abductive and default reasoning: A computational core. In *Proc. of AAAI-90*, pp. 343–348 Boston, MA.

Shanahan, M. (1989a). Prediction is deduction but explanation is abduction. In *Proc. of the 11 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1055–1060 Detroit, MI.

Shanahan, M. (1989b). Representing continuous change in the Event Calculus. Tech. rep., Imperial College, Dept. of Computing, Logic Programming Group. see also (Shanahan, 1990).

Shanahan, M. (1990). Representing continuous change in the Event Calculus. In *Proc. of the 9 th ECAI* Stockholm.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, *33*, 89–104.

Shoham, Y. (1988). Chronological ignorance: Experiments in nonmonotonic temporal reasoning. *Artificial Intelligence*, *36*, 279–331.

Shoham, Y. (1990). Nonmonotonic reasoning and causation. *Cognitive Science*, *14*, 213–252.

Stickel, M. E. (1990). Rationale and methods for abductive reasoning in natural-language interpretation. In Studer, R. (Ed.), *Natural Language and Logic: Proc. of the International Scientific Symposium, Hamburg, FRG*, pp. 233–252. Springer, Berlin, Heidelberg.

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, *1*, 333–355.

Stuckey, P. J. (1991a). Constructive negation for contraint logic programming. In *Proc. of LICS, Logic in Computer Science*, pp. 328–339.

Stuckey, P. J. (1991b). Incremental linear constraint solving and detection of implicit equalities. *ORSA Journal on Computing*, *3*(4), 269–274.

Sussman, G. J. (1975). *A Computer Model of Skill Acquisition.* Elsevier Science Publisher.

Swift, T., & Warren, D. S. (1994). An abstract machine for SLG resolution. Tech. rep., Dept. of Computer Science, University at Stony Brook.

van Beek, P. (1990). Reasoning about qualitative temporal information. In *Proc. of AAAI-90*, pp. 728–734 Boston, MA.

van Beek, P. (1989). Approximation algorithms for temporal reasoning. In *Proc. of the 11 th Int. Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1291–1296 Detroit, MI.

Venkatesh, G. (1986). A decision method for temporal logic based on resolution. *Lecture Notes in Computer Science*, *206*, 272–289.

Vila, L. (1994). A survey on temporal reasoning in artificial intelligence. *Ai Communications*, *7*(1), 4–28.

Vilain, M., & Kautz, H. (1986). Constraint propagation algorithms for temporal reasoning. In *Proc. of AAAI-86*, pp. 377–382 Philadelphia, PA.

Warren, D. H. (1983). An abstract prolog instruction set. Technical note 309, SRI International, Menlo Park, CA.

Wolper, P. (1985). The Tableau method for temporal logic: an overview. *Logique et Analyse*, *110-111*, 119–136.

Wos, L., & Robinson, G. A. (1970). Paramodulation and set of support. In Laudet, M., Lacombe, D., & Nolin, L. (Eds.), *Symposium on Automatic Demonstration*, pp. 276–310 Versailles.

Yampratoom, E., & Allen, J. F. (1993). Performance of temporal reasoning systems. TRAINS technical note 93-1, University of Rochester.

Annexe A

Notations utilisées dans ce mémoire

Les notations utilisées dans ce mémoire sont les suivantes :

$\vee$ ou logique ;

$\wedge$ et logique ;

$\Rightarrow$ impliquation logique ;

$\Leftrightarrow$ et $\equiv$ équivalence logique ;

$E_{[v/t]}$ l'expression E dans laquelle toutes les occurrences de v sont remplacées par t ;

$\sigma_1 \circ \sigma_2$ composition des deux substitutions σ_1 et σ_2 ;

$\mathcal{R}, \Re$ ensemble des nombres réels ;

$\sim a$ négation par échec dans les programmes logiques ;

$\models$ déduction logique, relation de conséquence ;

$\forall(f)$ fermeture universelle de f, c'est à dire la formule obtenue en préfixant f par des quantificateurs universels portant sur chacune des variables libres de f ;

$\exists(f)$ fermeture existentielle de f, même principe que la fermeture universelle mais avec des quantificateurs existentiels ;

Annexe B

La logique du premier ordre

La logique du premier ordre, encore appelée *calcul des prédicats*, est une classe particulière de langage logique autorisant la formation d'énoncés dont les valeurs de vérité dépendent de variables qui peuvent être quantifiées. De tels langages, qu'on appelle parfois réalisation d'un langage du premier ordre, ont une syntaxe qui permet de définir la forme des énoncés exprimables, et une manière de définir la sémantique, ou encore l'interprétation, des formules ainsi construites.

B.1 Aspects syntaxiques

Un langage logique du premier ordre suppose que l'on se donne trois ensembles de symboles. Le premier, V, est celui des *variables*: il s'agit de l'ensemble des symboles qui seront utilisés dans les formules pour faire référence de manière indéfinie à des objets ou individus du domaine. Le deuxième ensemble, P, est celui des *prédicats* ou encore des symboles de relation. Cet ensemble regroupe les noms des prédicats utilisés dans le langage. Le troisième et dernier ensemble F est celui des symboles de fonction : par exemple $+$ ou $-$.

Définition B.1 *Un* terme *est défini récursivement par les énoncés suivants :*

1. *une variable est un terme ;*
2. *si f est un symbole de fonction, et $t_1, \ldots, t_n$ un ensemble (éventuellement vide) de termes, alors $f(t_1, \ldots, t_n)$ est un terme (appelé fonctionnel). Dans ce cas les termes t_i sont les arguments de f. Une* constante *est représentée par un terme fonctionnel sans argument dont le symbole de fonction est le nom de la constante.*

De la même manière, on définit un *atome* (aussi appelé une *formule atomique*) comme étant une expression de la forme $p(t_1, \ldots, t_n)$ où p est un symbole de prédicat

provenant de l'ensemble P, et $t_1, \ldots, t_n$ sont des termes.

Définition B.2 *Les formules de la logique du premier ordre dites* bien formées *sont d'une des formes suivantes :*

1. *un atome ;*
2. *la négation $\neg f$ d'une formule bien formée f ;*
3. *la conjonction , notée $f_1 \wedge f_2$, la disjonction $f_1 \vee f_2$, et l'implication $f_1 \Rightarrow f_2$ de deux formules bien formées f_1 et f_2 ;*
4. *$\forall v\ f$ et $\exists v\ f$ où f est une formule bien formée et v est une variable.*

Parmi les formes particulières de formule, il faut citer le *littéral* qui est défini comme un atome ou la négation d'un atome. Un *littéral positif* est un atome, alors qu'un *littéral négatif* est la négation d'un atome.

Définition B.3 *Pour une formule de la forme $\forall x\, f$ ou $\exists x\, f$, la* portée *du quantificateur $\forall x$ (ou $\exists x$) est la formule f. On définit une* occurrence liée *d'une variable dans une formule comme une occurrence de cette variable, soit juste après un quantificateur, soit dans une formule qui se trouve dans la portée d'un quantificateur portant sur cette variable. Toute autre occurrence d'une variable est dite* libre.

Exemple B.1 *Soit la formule $\exists x\, p(x,y) \wedge q(x)$. La portée du quantificateur $\exists$ est la formule $p(x,y)$, les deux premières occurrences de la variable x sont liées, et la troisième est libre. L'occurrence de y est libre.*

Si la formule avait été $\exists x\ (p(x,y) \wedge q(x))$ alors toutes les occurrences de x auraient été liées.

Par abus de langage, et lorsqu'il n'y a pas d'ambiguïté possible, on parle de variables libres et de variables liées d'une formule pour désigner toutes les variables n'ayant que des occurrences libres (resp. liées) dans la formule.

Définition B.4 *Une formule* close *est une formule dans laquelle toutes les occurrences de variables sont liées.*

Définition B.5 *La* fermeture universelle *(resp.* existentielle*) d'une formule f, notée $\forall(f)$ (resp. $\exists(f)$), est la formule close obtenue en préfixant f par un quantificateur universel (resp. existentiel) pour chaque variable ayant une occurrence libre dans f.*

B.2 Aspects sémantiques et interprétation

La seule définition syntaxique d'un langage logique du premier ordre n'est jamais que la description d'une suite de symboles qui ne possède aucune signification. Le but de l'interprétation est de donner à toute formule bien formée une valeur de vérité en se référant à un *monde* particulier. Un énoncé comme *x est un nombre pair* n'a pas en lui même de valeur de vérité : mais si l'on se place dans l'univers de l'arithmétique, il est possible de lui donner une valeur de vérité pour chaque choix possible de x dans cet univers.

Une *interprétation* est un objet (mathématique) qui determine le monde dans lequel on se place pour interpréter les formules : sa structure est couramment définie comme un couple $(\mathcal{D}, I)$ où $\mathcal{D}$ est le *domaine* d'interprétation, c'est à dire l'ensemble des individus ou des objets constituant l'univers, et I est la fonction d'interprétation qui associe :

- à tout symbole de fonction f d'arité n, une fonction I_f de $\mathcal{D}^n$ dans $\mathcal{D}$;
- à tout symbole de prédicat P d'arité n, une fonction I_P de $\mathcal{D}^n$ dans l'ensemble $\{\mathsf{Vrai}, \mathsf{Faux}\}$.

L'interprétation d'une formule se fait dans une interprétation donnée et pour une affectation des variables libres de celle-ci. Une *affectation* est une fonction de l'ensemble des variables V dans le domaine $\mathcal{D}$. Une formule f est satisfaite par l'interprétation $\mathcal{I} = (\mathcal{D}, I)$ compte tenu de l'affectation des variables a, ce qui se note $\mathcal{I}, a \models f$, si l'interprétation de f est Vrai. Cette interprétation de f par $\mathcal{I}$ et a est notée $\mathcal{I}_a(f)$ en assimilant la structure d'interprétation $\mathcal{I}$ et l'affectation a à une fonction notée $\mathcal{I}_a$ dépendante de a et qui associe une valeur de vérité parmi $\{\mathsf{Vrai}, \mathsf{Faux}\}$ à toute formule bien formée.

De la même manière, la sémantique d'un terme t, noté $\mathcal{I}_a(t)$, est définie ainsi :

- si t est une variable, alors $\mathcal{I}(t) = a(t)$;
- si t est de la forme $f(s_1, \ldots, s_n)$, alors la sémantique de t est l'élément du domaine $\mathcal{D}$ défini par $I_f(\mathcal{I}_a(s_1), \ldots, \mathcal{I}_a(s_n))$.

La sémantique d'une formule atomique $P(t_1, \ldots, t_n)$ est définie comme

$$I_P(\mathcal{I}_a(t_1), \ldots, \mathcal{I}_a(t_n))$$

ce qui donne donc une des deux valeurs Vrai ou Faux. Si cette sémantique donne Vrai, on écrit :

$$\mathcal{I}, a \models P(t_1, \ldots, t_n)$$

Pour les formules composées, on utilise les règles suivantes :

- $\mathcal{I}, a \models \neg f$ si et seulement si $\mathcal{I}_a(f) = \mathsf{Faux}$;

- $\mathcal{I}, a \models f_1 \vee f_2$ si et seulement si $\mathcal{I}, a \models f_1$ ou $\mathcal{I}, a \models f_2$;
- $\mathcal{I}, a \models f_1 \wedge f_2$ si et seulement si $\mathcal{I}, a \models f_1$ et $\mathcal{I}, a \models f_2$;
- $\mathcal{I}, a \models \forall x\, f$ si et seulement si pour toute affectation de variables a' coïncidant avec a sauf en x on a $\mathcal{I}, a' \models f$;
- $\mathcal{I}, a \models \exists x\, f$ si et seulement si il existe une affectation a' coïncidant avec a sauf en x tel que l'on a $\mathcal{I}, a' \models f$.

B.3 Déduction en logique du premier ordre

On définit un *modèle* d'une formule f comme une interprétation $\mathcal{I}$ telle que $\mathcal{I}, a \models f$ pour toute affectation de variables a. On note ceci : $\mathcal{I} \models f$. La même définition s'étend à une *théorie*, c'est à dire un ensemble de formules.

Définition B.6 *Les conditions suivantes définissent quelques notions liées à la sémantique et utilisées en logique :*

- *Une formule f est* valide, *ce qu'on note $\models f$, si elle est vraie pour toute interprétation.*
- *Une formule f est* satisfaisable *s'il existe une interprétation $\mathcal{I}$ et une affectation de variables a telle que $\mathcal{I}, a \models f$.*
- *Une formule* contradictoire *est une formule qui n'est pas satisfaisable.*
- *Une formule* contingente *est une formule qui n'est ni valide ni contradictoire.*

La relation de *conséquence logique* se définit entre deux formules f_1 et f_2 comme le fait que la validité de f_1 entraine la validité de f_2. On note ceci $f_1 \models f_2$.

En ce qui concerne la complexité, la relation INSAT qui est vérifiée pour une formule lorsque celle-ci est contradictoire, est indécidable. Il n'existe pas d'algorithme général permettant de décider en un temps fini si une formule est contradictoire. De même, la relation de déduction ($\models$) entre deux formules est indécidable. En realité ces deux relations sont seulement semi-décidables, c'est à dire qu'il existe des procédures qui terminent lorsque la formule est contradictoire mais qui ne terminent pas forcément si elle ne l'est pas.

Annexe C

La programmation logique

La programmation logique trouve son origine à partir des années 70 dans les travaux sur la démonstration automatique, domaine qui fut initié par Herbrand puis par Robinson en particulier. La programmation logique, c'est à dire le fait que la logique soit utilisée comme un langage de programmation, a été introduite par les travaux de Kowalski puis de Colmerauer sur le langage Prolog.

La présentation qui suit ne reprend que les éléments nécessaires à une bonne compréhension du mémoire. Pour une introduction plus complète, le lecteur intéressé est invité à consulter (Lloyd, 1987).

C.1 Le langage de la programmation logique

La programmation logique a pour langage une forme particulière de clauses : les clauses de Horn. Une clause de Horn est une clause (c'est à dire logiquement une disjonction de littéraux) comportant au plus un littéral positif. Une telle clause a la forme suivante :

$$A \vee \neg L_1 \vee \ldots \vee \neg L_n$$

où A et $L_1, \ldots, L_n$ sont des atomes, et on la note ainsi :

$$A \leftarrow L_1, \ldots, L_n$$

où A est le seul littéral positif de la clause, et les littéraux $L_1, \ldots, L_n$ sont les opposés des autres littéraux de la disjonction. L'atome A est appelé la *tête* de la clause, et les littéraux $L_1, \ldots, L_n$ constituent le *corps* de la clause.

Une *clause définie* est une clause qui contient exactement un littéral positif, et un *but* est une clause dont la tête est vide. On définit ensuite un *programme logique* comme un ensemble de clauses de Horn : un tel programme est aussi appelé *défini* car il ne contient que des clauses avec des littéraux positifs.

Par convention, les variables d'un programme logique sont notées par des symboles commençant par une majuscule. Ceci permet d'éviter des ambiguïtés entre variables et constantes de même nom.

Soit P un programme logique du premier ordre (avec variables) : l'*univers de Herbrand* de P, noté U_P est l'ensemble de tous les termes sans variables qu'il est possible de construire à l'aide des constantes et des symboles de fonctions apparaissant dans P ; et la *base de Herbrand* de P, notée B_P est l'ensemble de tous les atomes sans variables qu'il est possible de construire avec tous les symboles de prédicat apparaissant dans P et tous les termes de l'univers de Herbrand de P.

C.2 Interprétation des programmes logiques

L'interprétation des programmes logiques, contrairement aux langages du premier ordre, repose sur ce qu'on appelle les interprétations de Herbrand. La base de telles interprétations a pour caractéristique que tous les termes clos sont affectés à eux-mêmes par les fonctions d'interprétation. Plus précisément :

1. le domaine d'interprétation est l'univers de Herbrand U_L ;

2. toute constante est associée à elle-même par la fonction d'interprétation ;

3. pour chaque symbole de fonction f d'arité n, la fonction d'interprétation correspondante associe le terme $f(t_1, \ldots, t_n)$ au n-uplet de termes $t_1, \ldots, t_n$.

Une *interprétation de Herbrand* est toute interprétation construite sur une structure vérifiant les trois conditions précédentes.

Comme l'interprétation des constantes et des termes sans variables est fixe dans le cas d'une interprétation de Herbrand, il est souvent plus commode de désigner de telles interprétations par un sous-ensemble de la base de Herbrand U_L du programme L : on ne retient dans cet ensemble que les atomes qui sont vrais dans l'interprétation de Herbrand considérée. Il faut noter qu'une telle interprétation ne contient que des informations positives, ce qui posera un problème certain lors de l'introduction d'un opérateur de négation dans les programmes logiques (cf. partie C.4).

Pour un programme logique L clos, c'est à dire sans variables libres, un *modèle de Herbrand* de L est une interprétation de Herbrand de L qui satisfait toutes les clauses de L. Ce modèle est toujours décrit comme un sous-ensemble de la base de

Herbrand U_L.

Exemple C.1 *Soit P_1 le programme logique suivant qui décrit un graphe et une condition d'accessibilité des nœuds de celui-ci:*

$$\begin{array}{ll} (1) & edge(a,b) \leftarrow . \\ (2) & edge(c,d) \leftarrow . \\ (3) & edge(d,c) \leftarrow . \\ (4) & reachable(a) \leftarrow . \\ (5) & reachable(X) \leftarrow reachable(Y), edge(Y,X). \end{array}$$

L'univers de Herbrand de ce programme est l'ensemble $\{a,b,c,d\}$ et un modèle de Herbrand est l'ensemble

$$M_1 = \{edge(a,b), edge(c,d), edge(d,c), reachable(a), reachable(b)\}$$

Exemple C.2 *Soit P_2 le programme logique suivant (extrait de (Lloyd, 1987)):*

$$\begin{array}{ll} (1) & sort(X,Y) \leftarrow sorted(Y), perm(X,Y). \\ (2) & sorted(nil) \leftarrow . \\ (3) & sorted(X.nil) \leftarrow . \\ (4) & sorted(X.Y.Z) \leftarrow X \leq Y, sorted(Y.Z). \\ (5) & perm(X.Y, U.V) \leftarrow delete(U, X.Y, Z), perm(Z,V). \\ (6) & delete(X, X.Y, Y) \leftarrow . \\ (7) & delete(X, Y.Z, Y.W) \leftarrow delete(X,Z,W). \\ (8) & 0 \leq X \leftarrow . \\ (9) & f(X) \leq f(Y) \leftarrow X \leq Y. \end{array}$$

Ce programme décrit, par le prédicat sort, que la liste Y est la version triée de la liste X. Les listes sont représentées par l'expression $X.Y$ où X est le premier élément de la liste, et Y est le reste de la liste. La fonction f représente la fonction successeur *sur les nombres entiers: 1 est représenté par $f(0)$, 2 par $f(f(0))$, etc.*

Le langage du programme P_2 utilise les constantes 0 et nil, les symboles de fonction f et « . », et les symboles de prédicats sort, sorted, perm, delete, et $\leq$. L'univers de Herbrand U_{P_2} est l'ensemble

$$\{0, f(0), f(f(0)), \ldots, f^k(0), \ldots, nil, 0.nil, 0.0.nil, \ldots, f(0).nil, f(0).0.nil, \ldots\}$$

qui est bien sûr infini.

La base de Herbrand de P_2 comprend toutes les instances des prédicats avec en argument les termes de l'univers de Herbrand. Un modèle de Herbrand de P_2 contient, entre autres, toutes les instances $sort(m,n)$ du prédicat sort, telles que soit m et n sont la liste vide nil, soit m est la version triée de n lorsque celle-ci est une liste de termes $f^k(0)$.

Parmi tous les modèles de Herbrand d'un programme, on est en général intéressé par les plus petits d'entre eux (au sens de l'inclusion ensembliste). Ces modèles sont les *modèles de Herbrand minimaux* d'un programme. Ils vérifient la condition qu'il n'existe pas de sous-ensemble propre de ces modèles qui soit lui-même un modèle du programme considéré. Un résultat utile est que tout programme logique constitué de clauses définies admet un unique modèle de Herbrand minimal.

C.3 La résolution SLD

La résolution SLD est une méthode de preuve basée sur le principe de résolution que l'on applique aux buts et aux clauses telles qu'on les manipule en programmation logique. Cette forme de résolution est définie pour tous les programme logiques définis pour lesquels elle constitue une méthode de preuve correcte et complète.

La partie suivante rappelle quelques définitions nécessaires pour l'exposé de la résolution SLD dans les programmes logiques avec variables.

C.3.1 Substitution et unification

Définition C.1 (Substitution) *Une* substitution *est un ensemble fini de paires (variable, terme) noté sous la forme* $\{v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\}$, *et qui associe à chaque variable* v_i *le terme* t_i.

Le domaine *d'une substitution est l'ensemble des variables y apparaissant en partie gauche des paires.*

L'application d'une substitution θ *à une expression* E *(terme, prédicat ou formule), noté* $E\theta$ *produit l'expression* E *dans laquelle toutes les occurrences des variables du domaine de* θ *sont remplacées par le terme correspondant indiqué par la substitution.*

Les substitutions peuvent être composées comme des fonctions: si $\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ et $\eta = \{Y_1 \mapsto s_1, \ldots, Y_m \mapsto s_m\}$, la composition de θ et η, notée $\eta \circ \theta$ ou $\theta\eta$, est obtenue à partir de l'ensemble:

$$\{Y_1 \mapsto s_1, \ldots, Y_m \mapsto s_m, X_1 \mapsto t_1\eta, \ldots, X_n \mapsto t_n\eta\}$$

en enlevant de celui-ci les paires $X_i \mapsto t_i\eta$ telles que $X_i = t_i\eta$, et les paires $Y_i \mapsto s_i$ telles que la variable Y_i appartient à l'ensemble $\{X_1, \ldots, X_n\}$.

Définition C.2 (Unificateur) *Un* unificateur *de deux expressions* A *et* B *est une substitution* σ *telle que* $A\sigma = B\sigma$. *Cette unificateur est appelé* le plus général *si pour toute autre unificateur* η *de* A *et* B, *il existe une substitution* γ *telle que* $\eta = \gamma \circ \sigma$.

$$
\begin{cases}
\{f(t_1,\ldots,t_m) = f(s_1,\ldots,s_m)\} \cup E & \rightarrow \{t_1 = s_1,\ldots,t_m = s_m\} \cup E \\
\{f(t_1,\ldots,t_m) = g(s_1,\ldots,s_n)\} \cup E & \rightarrow \text{échec} \\
\quad \text{avec } f \neq g & \\
\{X = X\} \cup E & \rightarrow E \\
\{t = X\} \cup E & \rightarrow \{X = t\} \cup E \\
\quad t \text{ n'est pas une variable} & \\
\{X = t\} \cup E & \rightarrow \{X = t\} \cup E(\{X \mapsto t\}) \\
\quad X \text{ n'est pas égal à } t \text{ et} & \\
\quad \text{n'apparaît pas dans } t &
\end{cases}
$$

FIG. C.1 - *Règles de simplification d'un ensembles d'équations pour obtenir l'unificateur le plus général. X désigne une variable, et t est, sauf précisions supplémentaires, un terme quelconque.*

L'unificateur le plus général entre deux atomes $p(t_1,\ldots,t_n)$ et $p(s_1,\ldots,s_n)$ est obtenu par application des règles de simplification de la figure C.1 sur l'ensemble d'équations $\{t_1 = s_1,\ldots,t_n = s_n\}$. Si l'application d'une règle produit un échec, il n'y a pas d'unificateur entre les deux atomes.

C.3.2 La résolution SLD

Définition C.3 (Dérivation SLD) *Soit P un programme défini et G un but de la forme $\leftarrow A_1,\ldots,A_n$, et $\mathcal{R}$ une règle de sélection des littéraux dans un but. Une* dérivation SLD *à partir du but G est une séquence (finie ou infinie) de buts $(G_i)_i$ telle que $G_0 = G$ et pour tout $i \geq 0$, G_{i+1} est obtenu à partir de G_i de la manière suivante :*

1. *A_m est un atome sélectionné par $\mathcal{R}$ dans le corps de G_i ;*
2. *$A \leftarrow B_1,\ldots,B_q$ est une clause du programme P qui ne partage aucune variable avec G_i (moyennant un renommage) ;*
3. *θ est l'unificateur le plus général de A_m et A ;*
4. *G_{i+1} est le but $\leftarrow (A_1,\ldots,A_{m-1},B_1,\ldots,B_q,A_{m+1},\ldots,A_n)\theta$.*

Définition C.4 (Réfutation SLD) *Une* réfutation SLD *d'un but G avec un programme P est une dérivation SLD à partir de ce but et qui finit par la clause vide $\square$. On écrit que cette réfutation est une réfutation de $P \cup \{G\}$.*

La *Résolution SLD* est définie comme le système qui utilise la dérivation SLD comme mécanisme d'inférence.

Définition C.5 (Substitution réponse) *Soit P un programme logique défini et*

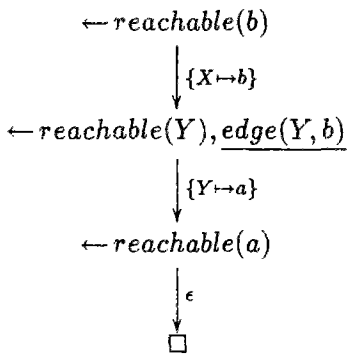


FIG. C.2 - *Une réfutation SLD de* $\leftarrow reachable(b)$ *avec le programme* P_1. *Le littéral sélectionné est souligné sauf s'il n'y a pas de confusion possible.*

G un but. Une substitution réponse *à la requête G est une substitution obtenue en projetant la composition* $\theta_1 \ldots \theta_n$ *sur les variables de G, où* $\theta_1, \ldots, \theta_n$ *est la séquence de tous les unificateurs les plus généraux obtenus au cours de la réfutation SLD de G.*

Les théorèmes suivants, qui établissent la correction et la complétude de la résolution SLD ont été établis par Lloyd (Lloyd, 1987).

Théorème C.1 (Correction de la résolution SLD) *Soit P un programme défini et G un but. Alors toute substitution réponse* θ *obtenue par résolution SLD est correcte, c'est à dire que* $P \models \forall(G\theta)$.

Théorème C.2 (Complétude de la résolution SLD) *Soit P un programme logique défini and G un but. Alors, pour toute substitution correcte* θ *pour le but G avec le programme P (c'est à dire* $P \models \forall(G\theta)$*), il existe une réfutation SLD qui calcule la substitution réponse* σ, *et une substitution* γ *telle que* $\theta = \gamma \circ \sigma$.

On définit *l'ensemble des succès* d'un programme P comme l'ensemble de tous les atomes A de la base de Herbrand de P tels qu'il existe une réfutation de $\leftarrow A$ avec P. Alors, un corollaire du théorème de correction de la résolution SLD est que l'ensemble des succès d'un programme défini P est inclus dans son modèle de Herbrand minimal. Le théorème de complétude a alors pour corollaire que l'ensemble des succès est égal au modèle de Herbrand minimal.

Un exemple de réfutation SLD apparaît sur la figure C.2 : $reachable(b)$ est une conséquence du programme P_1, et la substitution résultat calculée par la résolution SLD est la substitution identité ϵ.

Une propriété intéressante de la résolution SLD est son indépendance par rapport à la règle de sélection des littéraux. Ceci signifie que s'il existe une réfutation de

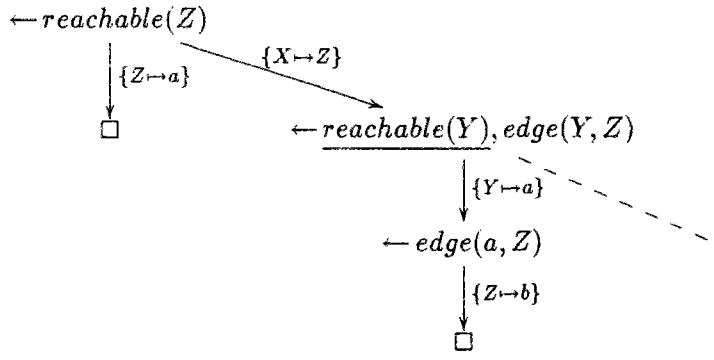


FIG. C.3 - *Un arbre SLD de* $P_1 \cup \{\leftarrow reachable(Z)\}$.

$P \cup \{G\}$ dont la substitution réponse est σ, alors pour toute règle de sélection R, il existe une réfutation SLD de $P \cup \{G\}$ utilisant R et dont la substitution réponse σ' est telle que $G\sigma'$ est une variante de $G\sigma$.

Si l'on s'intéresse à l'espace de recherche, celui-ci a la forme d'un arbre dont chaque nœud est un but, la racine de cet arbre est le but initial G, et un nœud a pour fils tous les résolvents obtenus par résolution SLD entre ce but et les clauses du programme. Une branche d'un tel arbre représente une dérivation SLD de G. L'indépendance par rapport à la règle de sélection permet de fixer celle-ci à l'avance et de ne considérer que l'arbre obtenu avec cette seule règle.

L'ensemble des branches d'un arbre SLD se répartit en trois catégories :

- les succès, c'est à dire que la branche est finie et a pour feuille extrémité une clause vide ;
- les échecs, c'est à dire les branches finies qui ne finissent pas par la clause vide ;
- les branches infinies.

La figure C.3 présente un arbre SLD produit à partir de la requête $reachable(Z)$: cet arbre a une branche infinie qui correspond à l'utilisation perpétuelle de la règle (5) du programme P_1 pour prolonger la dérivation. La présence de branches infinies dans un arbre SLD pose en général un problème de complétude si la stratégie de recherche utilisée est en profondeur d'abord.

Sur le même programme P_1 et le même but $\leftarrow reachable(Z)$, mais avec une règle de sélection différente, on obtient l'arbre de la figure C.4. Celui-ci contient deux branches « succès » comme le précédent, deux branches échec et aucune branche infinie. Dans les deux arbres, on retrouve les mêmes substitutions réponses.

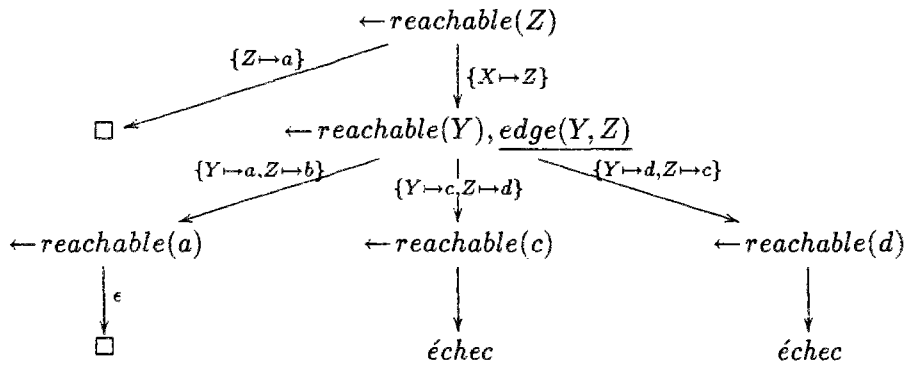


FIG. C.4 - *Autre arbre SLD de* $P_1 \cup \{\leftarrow reachable(Z)\}$.

C.4 La négation dans les programmes logiques

Les programmes logiques définis sont souvent assez restreints dans leur capacité expressive. En particulier, on souhaite disposer d'un opérateur de négation dans le corps des clauses. Cet opérateur de négation dans un programme logique induit un problème lié à l'incapacité pour la résolution SLD de déduire une information négative : si P est un programme défini et A un atome de sa base de Herbrand B_P, la résolution SLD ne permet pas de prouver que $\neg A$ est une conséquence de P. Ceci ne serait le cas que si $P \cup \{A\}$ était insatisfiable, ce qui n'est pas vrai car $P \cup \{A\}$ est satisfiable et a B_P comme modèle.

Ce problème est en général contourné par l'utilisation de *l'hypothèse du monde clos* introduite par (Reiter, 1978) qui postule que l'on peut considérer $\neg A$ vrai si A n'est pas une conséquence logique de P, et ceci pour tout atome clos A. De manière générale, comme la déduction est indécidable en logique du premier ordre, on n'implémente toujours que des versions affaiblies de cette hypothèse. Dans le cadre de la programmation logique, la règle usuelle est appelée *négation par échec fini* (Clark, 1978).

Un programme logique avec négation est dit *normal*. Le mécanisme utilisé pour calculer les substitutions réponses à une requète est la résolution SLDNF, extension de la résolution SLD par la règle de négation par échec fini. Ce mécanisme est décrit dans la partie suivante (C.4.1), et la partie C.4.2 introduit quelques éléments sur la sémantique des programmes logiques normaux.

C.4.1 La résolution SLDNF

La résolution SLDNF est simplement l'extension de la résolution SLD par la règle suivante :

> $\neg A$ est prouvé si et seulement si A ne peut pas être prouvé

où « A est prouvé » signifie qu'il existe un arbre SLD de $P \cup \{A\}$ qui contient une clause vide, ou encore qu'il existe une réfutation SLD de $P \cup \{A\}$. De la même manière, « A ne peut pas être prouvé » s'interprète comme le fait qu'il existe un arbre SLD de $P \cup \{A\}$ dont aucune branche n'est infinie, et qui ne contienne pas de clause vide. Le problème de la résolution SLDNF est que ces deux cas ne sont pas les seuls possibles pour un arbre SLD : il peut exister des arbres SLD qui ne contienne pas la clause vide et qui sont infinis.

Un cas pour lequel la résolution SLDNF n'est ni réussie ni un échec est celui où le littéral sélectionné est négatif et contient encore des variables. Ce cas particulier est évité si les conditions suivantes sont respectées :

1. toute clause du programme P est *admissible*, c'est à dire que toute variable de la clause apparaît soit dans la tête de celle-ci, soit dans au moins un littéral positif de son corps ;
2. la requête G est *permise*, c'est à dire que toute variable y apparaissant possède au moins une occurrence dans un littéral positif du corps de G ;
3. toute clause de P dont la tête est un prédicat qui apparaît aussi dans le corps d'une autre clause de P doit être permise.

La correction de la résolution SLDNF est définie par rapport à une théorie particulière obtenue à partir du programme P et appelée la *complétion* de P, notée $\mathsf{Comp}(P)$. Cette opération s'effectue ainsi :

1. toute clause C de la forme $p(t_1, \ldots, t_n) \leftarrow L_1, \ldots, L_m$ est transformée en :

   $$p(x_1, \ldots, x_n) \leftarrow \exists Y \ \ (x_1 = t_1), \ldots, (x_n = t_n), L_1, \ldots, L_m$$

   où Y est le vecteur des variables de C, et les variables $x_1, \ldots, x_n$ n'apparaissent pas dans C ;
2. pour tout symbole de prédicat p d'arité n, si ce symbole n'apparaît dans aucune tête de clause de P, alors $\mathsf{Comp}(P)$ comprend la clause :

   $$\forall X \ \ \neg p(x_1, \ldots, x_n)$$

   Sinon, soit $C_1, \ldots, C_k$ les clauses dont p est le symbole de prédicat de leur tête. On appelle *définition complétée* de p la formule

   $$\forall x_1, \ldots, x_n \ \ p(x_1, \ldots, x_n) \leftrightarrow E_1 \vee \ldots \vee E_k$$

où les formules E_k sont les parties droites des transformées de $C_1, \ldots, C_k$ par la transformation précédente.

La complétion de P est alors l'ensemble des définitions complétées de chaque prédicat du programme P ainsi que les axiomes qui définissent la relation d'identité $=$.

Les axiomes de $=$ imposent, entre autre, que toutes les constantes soient distinctes deux à deux, que deux fonctions f et g ne produisent jamais de résultats identiques, etc.

Intuitivement, la complétion du programme P postule que toute information qui n'est pas fournie par le programme est réputée fausse. En particulier, un atome n'est vrai que si une clause du programme définit son symbole de prédicat et que cette clause, correctement instanciée, permet de déduire l'atome.

Le résultat suivant, qui définit la correction de la résolution SLDNF, est dû à (Clark, 1978).

Théorème C.3 (Correction de la résolution SLDNF) *Soit P un programme logique normal et G un but permis. Alors toute substitution réponse θ obtenue par résolution SLDNF de $P \cup \{G\}$ est telle que* $\mathsf{Comp}(P) \models \forall(G\theta)$.

La complétude de la résolution SLDNF a été prouvée par différents auteurs, parmi lesquels (Clark, 1978; Lloyd, 1987; Cavedon & Lloyd, 1989). Ce résultat est valable pour les programmes logiques normaux et *hiérarchiques*, c'est à dire les programmes tels que l'on peut affecter à tout symbole de prédicat un entier positif, appelé *niveau*, et pour toute règle du programme le niveau des symboles de prédicat dans le corps est strictement inférieur au niveau du symbole du prédicat de la tête. Intuitivement, cette condition impose que le programme ne contienne pas de « boucle ».

Théorème C.4 (Complétude de la résolution SLDNF) *Soit P un programme logique normal et hiérarchique, G un but permis, et R une règle de sélection* sûre, *c'est à dire qui ne sélectionne un littéral négatif que lorsque celui-ci ne contient pas de variables. Alors, les deux propriétés suivantes sont vérifiées:*

1. *il existe un arbre SLDNF pour G construit avec la règle R et cet arbre est fini;*
2. *si θ est une substitution telle que $G\theta$ ne contient plus de variables et vérifiant* $\mathsf{Comp}(P) \models G\theta$, *alors θ est une substitution réponse qui peut être obtenue par résolution SLDNF.*

Un autre résultat de complétude de la résolution SLDNF est dû à Kunen (Kunen, 1989) et s'applique à tout programme logique dit « *call-consistent* ». Cette condition est plus faible que la hiérarchisation d'un programme, car elle permet la récursion d'un prédicat (boucle dans le programme) à travers un nombre pair de négations.

C.4.2 Sémantique des programmes logique avec négation

Un inconvénient d'un opérateur de négation est qu'un programme l'utilisant n'a plus un seul modèle de Herbrand minimal. Par exemple, le programme propositionnel P_3 suivant :

$$a \leftarrow b, \neg c$$
$$b \leftarrow$$

a deux modèles de Herbrand minimaux : $\{a, b\}$ et $\{b, c\}$. Bien évidemment, on préfèrera en général le premier au deuxième, comme correspondant plus à l'intuition sous-jacente de ce programme.

Plusieurs tentatives ont été faites de définir une sémantique des programmes logiques normaux qui ne repose pas sur un mécanisme opérationnel (Przymusinski, 1989b). La sémantique des *modèles stables* (Gelfond & Lifschitz, 1988), dont la définition suit, est une de ses approches. Elle s'appuie, pour un programme logique normal P, sur le programme propositionnel Π obtenu en instanciant P par tous les éléments de son univers de Herbrand.

A partir de ce programme propositionnel et d'un ensemble d'atomes M inclus dans la base de Herbrand de P, on construit un programme Π_M ainsi :

1. on enlève de Π toutes les règles qui ont dans leur corps un littéral $\neg B$ tel que $B \in M$;
2. on enlève des règles restantes tous les littéraux négatifs.

Ce programme Π_M ne contient plus de négation et admet donc un unique modèle de Herbrand minimal. Un *modèle stable* de Π est défini par (Gelfond & Lifschitz, 1988) comme un ensemble d'atomes M tel que le modèle de Herbrand minimal de Π_M est égal à M. Ce modèle stable est aussi un modèle de Herbrand minimal du programme propositionnel initial Π.

La sémantique des modèles stables a l'avantage d'être définie pour des programmes qui ne sont pas hiérarchiques. Soit par exemple le programme P_4 :

$$p(1,2) \leftarrow$$
$$q(x) \leftarrow p(x,y), \neg q(y)$$

Un modèle stable de ce programme est $M = \{p(1,2), q(1)\}$. Le programme Π_M correspondant est :

$$p(1,2) \leftarrow$$
$$q(1) \leftarrow p(1,2)$$
$$q(2) \leftarrow p(2,2)$$

Une remarque de (Gelfond & Lifschitz, 1988) est que la procédure de Prolog (qui applique la résolution SLDNF avec une règle de sélection dans l'ordre des littéraux

de la clause, et avec une stratégie en profondeur d'abord) produit la réponse *yes* sur ce type de programme si la requête ne contient pas de variables.

Dans le cas du programme propositionnel P_3 décrit plus haut, $\{a, b\}$ est un modèle stable de P_3, alors que l'ensemble $\{b, c\}$ ne l'est pas.

Annexe D

La programmation logique avec contraintes

La programmation logique avec contraintes a été présentée par (Jaffar & Lassez, 1987). Leur attention s'est porté principalement sur les contraintes sur les nombres réels, et ils ont proposé un système, appelé CLP($\mathcal{R}$), et qui a fait l'objet d'une description complète dans (Jaffar et al., 1992b).

Depuis l'émergence de ce paradigme, de nombreux travaux ont été mené, soit sur les aspects théoriques (sémantique, propriétés des domaines de contraintes et des procédures de preuve, etc.), soit sur les aspects pratiques (implantation, interprétation abstraite, nouveaux résolveurs de contraintes, etc.). Un revue générale des travaux en programmation logique avec contraintes apparaît dans (Jaffar & Maher, 1994).

D.1 Approche intuitive

Intuitivement, la programmation logique avec contraintes cherche à étendre la programmation logique classique en remplaçant le processus d'unification par un processus plus général de satisfaction de contraintes.

En programmation logique, l'algorithme d'unification prend en entrée un ensemble d'égalité entre termes, obtenues à partir des termes arguments d'un littéral choisi dans le but courant, et des termes arguments d'un littéral qui soit une tête de clause du programme. Si le but courant est

$$\leftarrow B_1, \ldots, B_{i-1}, P(x_1, \ldots, x_n), B_{i+1}, \ldots, B_m$$

et qu'une clause du programme logique est

$$P(y_1, \ldots, y_n) \leftarrow A_1, \ldots, A_l$$

alors, l'algorithme d'unification reçoit en entrée l'ensemble d'équations

$$\Gamma = \{x_1 = y_1, \ldots, x_n = y_n\}$$

Ce que réalise cet algorithme, en plus du calcul d'un unificateur, est principalement un test de « satisfaisabilité » de cet ensemble d'équations. C'est à dire que l'on cherche à déterminer l'existence de solution (des valeurs pour les variables) tels que ces égalités soient vérifiées. La substitution produite par l'algorithme n'est qu'une représentation d'une solution: en particulier, celle qui est calculée est la plus générale possible. Ce qui est important est le sens du mot « vérifié » dans ce qui précède: cette satisfaction se détermine par rapport à un *domaine* d'interprétation des termes, qui est ici l'univers de Herbrand, et une axiomatisation formelle de la relation d'égalité sur cet univers. L'algorithme d'unification constitue une procédure de décision correcte et complète pour ce domaine particulier.

La programmation logique avec contraintes étend cette vision de la programmation logique, en posant comme prérequis la définition d'un domaine particulier $\mathcal{A}$, qui permet de donner le sématique d'un langage $\mathcal{L}$. Ce langage est le langage des contraintes, et l'on parle par exemple du domaine des « équations et inéquations linéaires sur les nombres réels »: $X + Y > 7, 3X - Z = 3Y$ sont des contraintes sur le domaine des nombres réels.

Une fois obtenu la notion de domaine pour les contraintes, la programmation logique avec contraintes utilise celui-ci en lieu et place de l'unification. En particulier, un but est alors une clause sans tête qui contient dans son corps, aussi bien des littéraux « classiques » que des contraintes. Les contraintes sont différenciées des autres littéraux dans la mesure où elles sont traitées par des algorithmes spécifiques au domaine. Lorsqu'un littéral est sélectionné, il est réecrit à l'aide d'une clause du programme, et les contraintes associées à la clause résolvante sont une conjonction de celles du but, de celles de la clause du programme, et des égalités Γ entre sous-termes des deux littéraux.

D.2 Présentation formelle de CLP

Le modèle le plus général de CLP est celui décrit par (Höhfeld & Smolka, 1988) et qui est repris ici.

On suppose que l'on a un langage de contraintes $\mathcal{L}$, qui est caractérisé par:

- un ensemble de variables V;
- un ensemble Φ de formules, qui seront appelées *contraintes*;
- une fonction $Var : \Phi \rightarrow V$, qui associe à toute contrainte ϕ l'ensemble $Var(\phi)$ des *variables contraintes* par ϕ;
- une classe d'interprétations $\mathcal{A}$ sur un domaine $D^{\mathcal{A}}$;

Cette définition du langage $\mathcal{L}$ est suffisamment générale pour accepter n'importe quelle syntaxe pour les contraintes. Les interprétations de $\mathcal{A}$ servent à décrire la signification des contraintes, et l'on supose juste que, pour chaque interprétation A de cette classe, l'on dispose d'une fonction qui associe à chaque contrainte $\phi \in \Phi$ l'ensemble de ses solutions dans A : ce qui est noté $[\![\phi]\!]^A$, et qui est donc un ensemble de valuations des variables de ϕ.

Si α est une valuation qui est une solution de ϕ dans une interprétation A, on écrira aussi de $A, \alpha \models \phi$. C'est à dire qu'une fois les variables libres de ϕ instanciées par leur valeurs correspondantes données par α, cette contraintes est satisfaite (logiquement vraie) par A.

Le langage complet qui nous intéresse est une extension de $\mathcal{L}$ par des symboles de relations (prédicat) et des connecteurs logiques pour former les clauses. soit $\mathcal{R}(\mathcal{L})$ ce langage. Il est défini par :

- le même ensemble de variables V que $\mathcal{L}$;
- l'ensemble $\mathcal{R}(\Phi)$ des formules, en particulier :
  - toutes les contraintes de Φ ;
  - tous les atomes $r(X_1, \ldots, X_n)$ où r est un symbole de relation et les termes $X_1, \ldots, X_n \in V$;

  et toutes les formules composables à l'aide des connecteurs, c'est à dire :
  - ρ_1 & ρ_2 (conjonction) où ρ_1 et ρ_2 appartiennent à $\mathcal{R}(\Phi)$;
  - $\rho_1 \rightarrow \rho_2$ (implication) où ρ_1 et ρ_2 appartiennent à $\mathcal{R}(\Phi)$;
- la fonction $Var : \mathcal{R}(\Phi) \rightarrow V$, qui étend celle sur Φ en associant à chaque formule ρ l'ensemble $Var(\rho)$ des variables contraintes par ρ :
  - $Var(r(X_1, \ldots, X_n)) = \{X_1, \ldots, X_n\}$;
  - $Var(\rho_1 \,\&\, \rho_2) = Var(\rho_1) \cup Var(\rho_2)$;
  - $Var(\rho_1 \rightarrow \rho_2) = Var(\rho_1) \cup Var(\rho_2)$;
- la classe des interprétations « admissibles » $\mathcal{A}$ sur un domaine $D^{\mathcal{A}}$, telles qu'une interprétation $\mathcal{A}$ de cette classe étend une interprétation $\mathcal{A}_0$ de $\mathcal{L}$ sur le domaine $D^{\mathcal{A}} = D^{\mathcal{A}_0}$ par des relations $r^{\mathcal{A}}$ pour chaque symbole de relation r dans $\mathcal{R}$;

De la même manière que pour $\mathcal{L}$, et pour chaque interprétation $\mathcal{A}$ admissible de $\mathcal{R}(\mathcal{L})$, l'on définit une fonction qui associe à une formule ρ de $\mathcal{R}(\Phi)$ l'ensemble $[\![\rho]\!]^{\mathcal{A}}$ de valuations des variables de ρ qui sont appelées *solutions* de ρ et que l'on définit ainsi :

- $[\![r(X_1, \ldots, X_n)]\!]^{\mathcal{A}} = \{\alpha : (\alpha(X_1), \ldots, \alpha(X_n)) \in r^{\mathcal{A}}\}$;

- $[\![\rho_1 \,\&\, \rho_2]\!]^{\mathcal{A}} = [\![\rho_1]\!]^{\mathcal{A}} \cap [\![\rho_2]\!]^{\mathcal{A}}$;
- $[\![\rho_1 \rightarrow \rho_2]\!]^{\mathcal{A}} = (Val(\mathcal{A}) - [\![\rho_1]\!]^{\mathcal{A}}) \cup [\![\rho_2]\!]^{\mathcal{A}}$, où $Val(\mathcal{A})$ désigne l'ensemble de toutes les valuations possibles (fonctions de V dans $D^{\mathcal{A}}$).

Dans ce cadre, la règle de résolution est la suivante : si R est la clause

$$\leftarrow B_1 \,\&\, \ldots \,\&\, r(X_1, \ldots, X_n) \,\&\, \ldots \,\&\, B_k \,\&\, \phi$$

et que C est une clause du programme éventuellement renommée ainsi :

$$r(X_1, \ldots, X_n) \leftarrow A_1 \,\&\, \ldots \,\&\, A_m \,\&\, \phi'$$

où ϕ et ϕ' sont des contraintes, alors le résolvant de ces deux clauses est :

$$\leftarrow B_1 \,\&\, \ldots \,\&\, A_1 \,\&\, \ldots \,\&\, A_m \,\&\, \ldots \,\&\, B_k \,\&\, \phi \,\&\, \phi'$$

Les notions habituelles de sémantique des programmes que l'on trouve ne programmation logique peuvent être adapté aisément au cas de la programmation logique avec contraintes.

Si $\mathcal{C}$ est un ensemble de clauses de $\mathcal{R}(\mathcal{L})$ définies, c'est à dire où les clauses sont de la forme :

$$r(\vec{X}) \leftarrow r_1(\vec{X}_1) \,\&\, \ldots \,\&\, r_m(\vec{X}_m) \,\&\, \phi$$

où les $r_i(\vec{X}_i)$ sont des atomes relationels de $\mathcal{R}(\mathcal{L})$, et ϕ est une conjonction de formules de $\mathcal{L}$, alors on définit un *modèle* de $\mathcal{C}$ comme une interprétation $\mathcal{M}$ de $\mathcal{R}(\mathcal{L})$ telle que toute valuation $\alpha : V \rightarrow D^{\mathcal{M}}$ est une solution de chaque formule (clause) de $\mathcal{C}$, ou, autrement dit, $[\![\rho]\!]^{\mathcal{M}} = Val(\mathcal{M})$, pour tout $\rho \in \mathcal{C}$.

Il est possible, (Höhfeld & Smolka, 1988), d'étendre toute interprétation du langage de contraintes $\mathcal{L}$ en un *modèle minimal* $\mathcal{M}$ de l'ensemble de clauses $\mathcal{C}$. La minimalité s'entendant au sens ou la structure rajoutée pour étendre l'interprétation est minimale : si $\mathcal{M}'$ est un autre modèle de $\mathcal{C}$, alors $r^{\mathcal{M}} \subseteq r^{\mathcal{M}'}$ pour toute relation $r \in \mathcal{R}$.

Ce modèle minimal peut aussi être construit comme la limite $\mathcal{M} = \bigcup_{i \geq 0} \mathcal{A}_i$ d'une séquence d'interprétations de $\mathcal{R}(\mathcal{L})$ définies ainsi :

- $r^{\mathcal{A}_0} = \emptyset$;
- $r^{\mathcal{A}_{i+1}} = \{(\alpha(X_1), \ldots, \alpha(X_n)) : \alpha \in [\![\rho]\!]^{\mathcal{A}_i}$ pour tout $r(X_1, \ldots, X_n) \leftarrow \rho \in \mathcal{C}\}$;
- $r^{\mathcal{M}} = \bigcup_{i \geq 0} r^{\mathcal{A}_i}$.

et ceci pour tout symbole r de $\mathcal{R}$.

Par rapport à cette sémantique, la règle de résolution avec contraintes décrite précédemment est correcte, c'est à dire que le résolvant R' est satisfait par toute

interprétation $\mathcal{A}$ du programme $\mathcal{C}$ et toute valuation qui satisfait R, ce que l'on note $[\![R']\!]^{\mathcal{A}} \subseteq [\![R]\!]^{\mathcal{A}}$.

De la même manière, l'om peut montrer que si $\mathcal{M}$ est un modèle minimal de $\mathcal{C}$, et que $\alpha \in [\![R]\!]^{\mathcal{M}}$ est une solution de la formule R dans $\mathcal{M}$, alors il existe une séquence de réduction (application de la règle de résolution) de la formule R en une contrainte ϕ (c'est à dire une clause vide où ne subsistent que des contraintes) telle que $\alpha \in [\![\phi]\!]^{\mathcal{M}}$.

D.3 Instances de CLP

Les dialectes de CLP sont assez nombreux. On peut citer le plus célèbre CLP($\mathcal{R}$) sur le domaine des équations et inéquations sur l'ensemble $\mathcal{R}$ des nombres réels (Jaffar et al., 1992b). D'autres exemples sont CLP(fd) (Diaz & Codognet, 1993), sur les contraintes sur domaines finis, ou encore CLP(PB) sur les pseudos-booléens (Barth, 1992).

Contribution à l'étude du raisonnement temporel : Résolution avec contraintes et application à l'abduction en raisonnement temporel

Ce travail présente notre contribution au domaine du raisonnement temporel (RT) en intelligence artificielle. Nous avons défini et mis en œuvre un mécanisme de raisonnement abductif (génération d'hypothèses) pour le RT. Un tel mode de raisonnement présente en particulier l'intérêt d'être une alternative possible au raisonnement par défaut pour prendre en compte la non monotonie inhérente au RT. Une autre motivation est que le raisonnement abductif appliqué au RT est une approche possible pour la planification.

Préalablement à la présentation de la procédure d'abduction, nous étudions l'intérêt pour le raisonnement temporel du principe de résolution avec contraintes de Bürckert. Ce principe s'avère être un cadre formel intéressant pour décrire l'intégration des systèmes de contraintes temporelles dans des systèmes déductifs utilisant le principe de résolution comme règle d'inférence.

Dans la deuxième partie, nous proposons une procédure de génération d'hypothèses qui est basée conjointement sur des travaux de Kakas et Mancarella sur l'abduction en programmation logique, et sur l'idée de la résolution avec contraintes. La procédure que nous proposons possède des mécanismes originaux facilitant en premier lieu son application au raisonnement temporel, et permettant ensuite de ramener la conservation de la consistance des hypothèses déja générées à des tests de satisfaction de contraintes temporelles.

Nous présentons des exemples d'utilisation, en particulier pour la planification en utilisant le Calcul d'Evénements de Sergot et Kowalski comme formalisme de représentation. Le cadre de la programmation logique avec contraintes, sous jacent à notre travail, nous permet d'étendre notre procédure à d'autres systèmes de contraintes. Nous décrivons en particulier l'utilisation de contraintes sur domaines finis, ce qui permet de décrire et gérer des ressources finies en planification. L'implantation de la procédure utilise, en plus des contraintes temporelles et des contraintes sur domaines finis, des contraintes sur le typage des termes, ce qui nous permet de proposer un cadre logique avec types et sous-typage qui facilite la description du formalisme temporel et des problèmes à résoudre.