Vérification et Spécification des Systèmes Distribués

Benjamin Lerman

Université Paris 7 Denis Diderot
UFR d'informatique

Thèse pour l'obtention du titre de
Docteur de l'Université Paris 7
Spécialité Informatique

Vérification et Spécification des Systèmes Distribués

Benjamin Lerman

Cette thèse a été soutenue le 28 novembre 2005.
Le jury était constitué de

Paul Gastin (directeur)
Rémi Morin
Madhavan Mukund (rapporteur)
Igor Walukiewicz (rapporteur)
Marc Zeitoun
Wieslaw Zielonka

Remerciements

Au terme de cet important travail, je tiens à remercier toutes les personnes qui, des près ou de loin, m'ont facilité la tâche.

En particulier je souhaite remercier les deux rapporteurs de cette thèse : Igor Walukiewicz pour nos discussions fructueuses au cours de cette thèse et Madhavan Mukund pour ses remarques pertinentes sur le manuscrit. Un grand merci également à Rémi et Morin et Wieslaw Zielonka d'avoir accepté de faire partie de mon jury.

Pour l'orientation scientifique et les échanges constructifs je remercie mon directeur de thèse, Paul Gastin ainsi que Marc Zeitoun qui a beaucoup participé à l'encadrement de mon travail.

Pour leur accueil chaleureux et leur aide dans les diverses tâches administratives, je remercie Jean-Éric Pin, Anca Muscholl et Ahmed Bouajjani ainsi que de tout coeur Noëlle Delgado, secrétaire de l'unité.

Je garderai un très bon souvenir de mes années de thèse au L.I.A.F.A. grâce à la bonne humeur et l'amitié de tous les thésards du laboratoire et je leur souhaite une bonne continuation à tous.

Je remercie aussi mes amis pour m'avoir rappelé assez souvent que la vie ne s'arrête pas aux portes du laboratoire, voire même à celles de notre univers.

Pour leur soutien indéfectible et leur constante attention je remercie mes parents, ma sœur et toute ma famille.

Enfin, mes remerciements vont tout particulièrement vers celle qui partage ma vie au jour le jour, qui me soutient et me supporte. Merci Cécile.

Table des matières

Première partie

Préliminaires

Chapitre 1

Introduction

1.1 Présentation

It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise–this thing gives out and [it is] then that "Bugs"–as such little faults and difficulties are called–show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.

Edison to Puskas, 13 November 1878

Depuis que la notion de processeur existe, la hantise du programmeur a un nom : "bug". L'histoire qui veut que l'étymologie de ce terme vienne de la présence d'un insecte dans un relais, empêchant ainsi son bon fonctionnement, est malheureusement fausse, et ces "bugs" que nous craignons tant sont bien dus à des erreurs humaines. Le "graal" de générations de programmeurs est alors de trouver un moyen automatique de détecter et de corriger ces erreurs. Malheureusement, depuis qu'en 1936 Alan Turing montre qu'il n'est pas possible de prouver automatiquement si un programme s'arrête ou non, ce "graal" semble inaccessible.

Cependant, des méthodes approchées ont été trouvées. Les méthodes de vérification formelle se basent sur des abstractions des programmes et permettent d'obtenir des résultats partiels. Il s'agit, d'une part, de modéliser le système et, d'autre part, de modéliser le comportement attendu, puis de vérifier automatiquement que le modèle ne présente pas de comportements aberrants.

Cependant une grand majorité de ces résultats s'intéresse aux systèmes séquentiels, c'est à dire à des systèmes qui n'exécutent qu'une seule action à la fois de manière ordonnée. Aujourd'hui, de plus en plus de système sont distribués, c'est à dire qu'ils sont composés de plusieurs systèmes séquentiels qui effectuent des calculs en parallèle et qui communiquent entre eux. En effet, que l'on considère un ensemble d'ordinateurs reliés par un réseau ou alors un seul ordinateur possédant plusieurs processeurs, nous ne pouvons pas facilement modéliser ces systèmes par des systèmes séquentiels. Ces considérations nous poussent donc à chercher des moyens d'étendre les technique de vérification formelle aux systèmes distribués et cette thèse a pour but de présenter un certain nombre de nouvelles approches que nous avons suivies.

1.2 Les problèmes

Ce travail a cherché à résoudre plusieurs problèmes ayant trait aux systèmes distribués.

Tout d'abord, nous nous sommes intéressés au problème de la spécification. Il s'agit de se donner un formalisme qui nous permet de décrire le comportement attendu d'un système. En effet, le but de la vérification est de vérifier qu'un modèle du système qui nous intéresse est correct. Pour définir dans quel sens un système est correct, il nous faut pouvoir définir formellement les comportements attendus et les comportements interdits de notre système. Nous avons donc besoin d'un outil qui soit tout d'abord suffisamment riche pour permettre d'exprimer les propriétés recherchées, mais qui soit également suffisamment simple pour qu'il soit aisé d'écrire ces propriétés sans risquer de se tromper. Il faut également qu'il soit possible et de préférence facile de vérifier si un comportement donné appartient ou non à la spécification. Il s'agit donc de trouver un compromis entre pouvoir d'expression et simplicité d'utilisation.

Nous nous sommes ensuite intéressés au problème de modélisation. Nous savons que nous ne pouvons pas vérifier si la plupart des systèmes réels vérifient une spécification. Il nous faut donc choisir un cadre dans lequel ce problème peut être résolu. Nous devons donc définir une classe de systèmes qui soit capable d'approximer le comportement de nos systèmes tout en permettant de vérifier si ce modèle vérifie ou non notre spécification. Encore une fois nous devons trouver un compromis : un modèle permettant d'exprimer des comportements complexes risque de rendre le problème de la vérification indécidable (nous ne pourrons pas déterminer si un système décrit dans ce modèle vérifie ou non une spécification) alors qu'un modèle ne permettant pas d'exprimer suffisamment de comportements ne nous permettra pas de modéliser de manière satisfaisante les systèmes réels.

Enfin nous nous sommes intéressés au problème de la synthèse de contrôleur. Il s'agit de se donner un système en interaction avec un environnement et une spécification pour ce système, puis de calculer de manière automatique quel doit être le comportement de ce système afin qu'il vérifie sa spécification quelles que soient les actions de l'environnement. Il s'agit donc de trouver automatiquement quel doit être le comportement d'un système quand il interagit avec une entité que nous ne pouvons pas contrôler, que ce soit un autre ordinateur à l'autre bout d'un réseau ou bien l'utilisateur en face de son clavier. Nous ne pouvons pas être assuré que le comportement de l'environnement sera correct et il faut donc nous assurer que nous pouvons imposer un comportement adéquat en toutes circonstances.

1.3 Plan

Préliminaires Nous avons commencé par introduire un certain nombre de notations et de notions classiques que nous utiliserons tout au long de ce document. En particulier nous introduisons les traces de Mazurkiewicz qui nous serviront à modéliser les exécutions de nos systèmes distribués.

Logiques de Traces Pour le problème de spécification, nous nous sommes intéressés aux logiques temporelles. Il s'agit d'un cadre général permettant

d'écrire des formules décrivant l'évolution du système au cours du temps. Après avoir introduit quelques logiques temporelles sur les traces de Mazurkiewicz et décrit les différences entre les logiques globales et locales, nous avons montré une nouvelle preuve de la complexité de $\text{ISTL}^{\diamond}$ qui lie cette logique globale aux logiques locales. Puis, nous avons introduit une nouvelle logique globale : la logique des filtres et nous avons décrit une méthode pour vérifier si une exécution vérifiait une formule de cette logique.

Automates Asynchrones Alternants Cellulaires Pour le problème de la modélisation, nous avons étendu les automates asynchrones cellulaires de Zielonka en des automates asynchrones alternants cellulaires dans l'espoir de pouvoir résoudre aisément le problème de la complémentation. Nous avons malheureusement montré que ces automates ne résolvaient malheureusement pas ce problème de la manière attendue, en revanche nous avons montré que sur les alphabets séries paralléles, ces automates avaient exactement la même expressivité que les automates asynchrones cellulaires.

Jeux distribués Enfin, pour le problème de la synthèse de contrôleur, nous avons introduit un nouveau type de jeu. En effet, le problème de la synthèse de contrôleur est un problème qui se réduit aisément à un problème de théorie des jeux. Si nous modélisons notre système et son environnement comme un jeu où chaque exécution est une partie et que nous considérons que les parties gagnantes sont celles où le système a vérifié sa spécification, trouver un contrôleur est équivalent à trouver une stratégie gagnante dans ce jeu. Nous avons donc introduit des jeux prenant en compte la particularité des systèmes distribués : c'est à dire des jeux se jouant à plus de deux joueurs et sur lesquels plusieurs joueurs peuvent jouer en même temps. Nous avons alors décrit un méthode pour trouver des stratégies gagnantes sans mémoire, puis, nous avons montré que sur les alphabets séries parallèles, il était décidable de savoir s'il existait une stratégie gagnante avec une certaine mémoire que nous avons appelée mémoire causale.

Chapitre 2

Notations et Définitions

2.1 Notations diverses

Étant donné un ensemble Q, on appellera $\mathcal{B}^+(Q)$ l'ensemble des formules booléennes positives sur Q, à savoir, les formules construites à partir des éléments de Q en utilisant les connecteurs logiques $\wedge$ et $\vee$. On interdit la conjonction vide (tt) et la disjonction vide (ff) dans ces formules. Un sous ensemble X de Q *satisfait* $\varphi \in \mathcal{B}^+(Q)$ (on écrira $X \models \varphi$) si l'assignation $\chi_X : Q \to \{\mathsf{tt}, \mathsf{ff}\}$ définie par $\chi_X(q) = \mathsf{tt}$ si et seulement si $q \in X$, satisfait φ. Du fait que tt n'est pas autorisé dans les formules, $X \models \varphi$ nous assure que X est non vide.

Soit $f : X \to Y$ une fonction et $X' \subseteq X$, on écrira $f(X')$ pour l'ensemble $\bigcup_{x' \in X'} f(x')$.

Par abus de notation, pour un sous ensemble Y d'un ensemble X, on écrira $\overline{Y}$ à la place de $X \setminus Y$ quand l'ensemble X est clair dans le contexte.

Soient X, I deux ensembles et $J \subseteq I$, pour $x = (x_i)_{i \in I} \in X^I$, on écrira $x_J = (x_j)_{j \in J} \in X^J$. De même si $(X_i)_{i \in I}$ sont des ensembles et $J \subseteq I$, on écrira $X_J = \prod_{j \in J} X_j$.

Si Σ est un alphabet fini, on notera Σ^* l'ensemble des mots finis sur l'alphabet Σ, Σ^ω l'ensemble des mots infinis sur l'alphabet Σ et $\Sigma^\infty = \Sigma^* \uplus \Sigma^\omega$ l'ensemble des mots finis et infinis sur l'alphabet Σ.

2.2 Ordres partiels

Soir $(V, \leq)$, un ensemble partiellement ordonné et $H \subseteq V$, on écrira $\downarrow_V H = \{e \in V \mid \exists h \in H, e \leq h\}$. Quand il n'y a pas d'ambiguïté, on écrira cet ensemble $\downarrow H$. On dira que H est *fermé par le bas* si $\downarrow H = H$. On écrira $\downarrow x$ à la place de $\downarrow\{x\}$. On écrira $\Downarrow x = \downarrow x \setminus \{x\}$.

Si $\leq$ est un ordre partiel, on écrira $\lessdot$ la relation de successeur, à savoir $\lessdot = < \setminus <^2$.

Définition 1 (Ensembles partiellement ordonnés étiquetés).
Soit Σ un alphabet fini. Un ensemble partiellement ordonné étiqueté *par Σ est un triplet $(V, \leq, \ell)$ où $(V, \leq)$ est un ensemble partiellement ordonné et $\ell : V \to \Sigma$ est*

la fonction d'étiquetage. Un isomorphisme *entre deux ensembles partiellement ordonnés étiquetés $t = (V, \leq, \ell)$ et $t' = (V', \leq', \ell')$ est une bijection $\varphi : V \to V'$ telle que $\ell' \circ \varphi(x) = \ell(x)$ et $x \leq y \Rightarrow \varphi(x) \leq \varphi(y)$. On écrit $t \sim_\varphi t'$ (abrégé en $t \sim t'$) s'il existe un isomorphisme φ entre t et t' et on dit que t et t' sont isomorphes.*

2.3 Traces de Mazurkiewicz

Définition 2 (Alphabets de dépendance).
Un alphabet de dépendance *est une paire (Σ, D) où Σ est un ensemble fini d'actions et $\mathrm{D} \subseteq \Sigma \times \Sigma$ est une relation réflexive et symétrique nommée* relation de dépendance*. Son complémentaire $\mathrm{I} = (\Sigma \times \Sigma) \setminus \mathrm{D}$ est appelé* relation d'indépendance*.*

Pour un alphabet de dépendance (Σ, D), pour tout $a \in \Sigma$, nous noterons $\mathrm{D}(a) = \{b \in \Sigma \mid a \,\mathrm{D}\, b\}$ et pour tout $A \subseteq \Sigma$, nous noterons $\mathrm{D}(A) = \bigcup_{a \in A} \mathrm{D}(a)$.

Parmi les alphabets de dépendance, nous nous intéresserons particulièrement à un cas particulier : les alphabets *séries parallèles* ou *cographes*. Il s'agit de la plus petite famille d'alphabet de dépendance contenant les singletons et close par les opérations $\cdot$ (*produit série*) et $\|$ (*produit parallèle*) définies ci-dessous :

Soit (A, D_A) et (B, D_B) deux alphabets de dépendance, alors $(A, \mathrm{D}_A) \cdot (B, \mathrm{D}_B) = (A \uplus B, \mathrm{D}_A \cup D_B \cup A \times B \cup B \times A)$. Il s'agit donc de l'union des alphabets de dépendances A et B où tous les éléments de A sont dépendants de tous les éléments de B.

Soit (A, D_A) et (B, D_B) deux alphabets de dépendance, alors $(A, \mathrm{D}_A) \| (B, \mathrm{D}_B) = (A \uplus B, \mathrm{D}_A \cup D_B)$. Il s'agit de l'union des alphabets de dépendances A et B où tous les éléments de A sont indépendants de tous les éléments de B.

Définition 3 (Traces de Mazurkiewicz).
Les traces de Mazurkiewicz *sur l'alphabet (Σ, D) sont, à isomorphisme près, une restriction des ensembles partiellement ordonnés étiquetés par Σ.*

Une trace de Mazurkiewicz t sur (Σ, D) est, à isomorphisme près, un ensemble fini ou infini, partiellement ordonné étiqueté par Σ, $t = (V, \leq, \ell)$ tel que :

$$\begin{array}{lll} \textit{(T1)} & \forall x \in V & \downarrow x \textit{ est un ensemble fini.} \\ \textit{(T2)} & \forall x, y \in V & x \lessdot y \Rightarrow \ell(x) \,\mathrm{D}\, \ell(y). \\ \textit{(T3)} & \forall x, y \in V & \ell(x) \,\mathrm{D}\, \ell(y) \Rightarrow x \leq y \textit{ ou } y \leq x. \end{array}$$

Nous prenons les conventions et notations suivantes :

Soit $t = (V, \leq, \ell)$ une trace sur l'alphabet de dépendance (Σ, D), nous disons que t est finie si V est un ensemble fini.

Nous notons $\mathbb{R}(\Sigma, \mathrm{D})$ l'ensemble des traces finies et infinies sur (Σ, D) et $\mathbb{M}(\Sigma, \mathrm{D})$ l'ensemble des traces finies sur (Σ, D).

Soit $t = (V, \leq, \ell) \in \mathbb{R}(\Sigma, \mathrm{D})$. Nous définissons $\mathrm{Alph}(t) = \ell(V)$ et $\mathrm{Alph}_\infty(t) = \{a \in \Sigma \text{ telles que } |\ell^{-1}(a)| = \infty\}$.

Nous nommons linéarisation d'une trace $t \in \mathbb{R}(\Sigma, \mathrm{D})$, un élément $\omega \in \Sigma^\infty$ tel que :

▷ Il existe une bijection e de $[\![0..|V|[\![$ dans V.
▷ $\forall i \in [\![0..|V|[\![$, $\omega_i = \ell(e(i))$.
▷ $\forall i, j \in [\![0..|V|[\![$, $e(i) < e(j) \Rightarrow i < j$.

Toute trace de $\mathbb{R}(\Sigma, \mathrm{D})$ admet au moins une linéarisation dans Σ^∞ et tout élément ω de Σ^∞ est une linéarisation d'une unique trace de $\mathbb{R}(\Sigma, \mathrm{D})$. Cette propriété n'est pas vraie sur les ordres partiels en général, elle est propre aux traces. Elle est due au fait que l'ordre est déterminé, en grande partie, par l'étiquetage. Nous noterons $[\omega]$ la trace dont ω est une linéarisaion. De plus, si T est un ensemble de traces, nous nommerons $\mathrm{Lin}(T)$, l'ensemble de toutes les linéarisations des traces de T.

Une trace $r = (V_r, \leq_r, \ell_r)$ est un préfixe d'une trace $s = (V_s, \leq_s, \ell_s)$, si
▷ $V_r \subseteq V_s$
▷ $\leq_r$ et $\leq_s$ sont égaux sur V_r
▷ ℓ_r et ℓ_s sont égaux sur V_r
▷ $\forall x \in V_r$, $\forall y \in V_s \backslash V_r$, $y \not\leq_s x$

En particulier, si r est une trace finie, r est un préfixe d'une trace s s'il existe une linéarisation de r qui est un préfixe d'une linéarisation de s.

Si une trace r est un préfixe de s, nous noterons $r \leq s$.

Si on deux préfixes r et s d'une trace t, on note $r \cup s$ l'unique préfixe minimale de t contenant r et s, c'est à dire tel que $r \leq r \cup s$ et $s \leq r \cup s$.

On écrira $\lessdot$ la relation de successeur, à savoir $\lessdot = < \setminus <^2$.

Soit deux mots $w_1, w_2 \in \Sigma^\omega$, on écrira $w_1 \sim w_2$ si $[w_1] = [w_2]$. De même on écrira $w_1 \lesssim w_2$ si $[w_1] \leq [w_2]$.

Soit $r = (V_1, \leq_1, \ell_1)$ et $s = (V_2, \leq_2, \ell_2)$ deux traces de $\mathbb{R}(\Sigma, \mathrm{D})$, telles que $\mathrm{D}(\mathrm{Alph}_\infty(r)) \cap \mathrm{Alph}(s) = \emptyset$, nous définissons $r \cdot s$ comme l'unique trace $t = (V, \leq, \ell)$ de $\mathbb{R}(\Sigma, \mathrm{D})$ telle que :
▷ $V = V_1 \uplus V_2$.
▷ $\ell_{|V_1} = \ell_1$.
▷ $\ell_{|V_2} = \ell_2$.
▷ $\leq$ est confondu avec $\leq_1$ sur V_1.
▷ $\leq$ est confondu avec $\leq_2$ sur V_2.
▷ $\forall x_1 \in V_1$, $\forall x_2 \in V_2$, $x_2 \not\leq x_1$

On vérifie aisément que l'ordre partiel ainsi défini est encore une trace.

Si $t = (V, \leq, \ell)$ est une trace. Soit $r = (C, \leq_{|C}, \ell_{|C})$ un préfixe fini de t. Nous définissons les éléments maximaux de r comme les éléments maximaux de C pour $\leq$, noté $\max(r)$.

Pour toutes traces r et s telles que $r \leq s$, nous définissons $r^{-1} \cdot s$ comme l'unique trace u telle que $s = r \cdot u$.

Lemme 1 (Lemme de Levy) *Soit u, u', v, $v'i$, quatres traces telles que u, $u' \in \mathbb{M}(\Sigma, \mathrm{D})$, $v, v' \in \mathbb{R}(\Sigma, \mathrm{D})$ et $u \cdot v = u' \cdot v'$, alors* $\mathrm{Alph}(u \setminus u')\ \mathrm{I}\ \mathrm{Alph}(u' \setminus u)$.

2.4 Automates Asynchrones

Zielonka [61, 62] a introduit les Automates Asynchrones comme un outil reconnaissant les langages reconnaissables de traces finies. Des travaux consécutifs les ont utilisés pour reconnaître les langages reconnaissables de traces infinies [24]

ou des ordres partiels [19]. Ils fonctionnent directement sur les ordres partiels et non sur les linéarisations.

Définition 4 (Architecture).
Une architecture *est un quadruplet* $(\Sigma, \mathcal{P}, R, W)$ *où* Σ *est un ensemble fini d'actions,* $\mathcal{P}$ *est un ensemble fini de processus,* $R : \Sigma \to 2^{\mathcal{P}}$ *associe à chaque élément* a *de* Σ *son domaine de lecture et* W *associe à chaque élément* a *de* Σ *son domaine d'écriture.*

On considérera dans la suite uniquement les architectures $(\Sigma, \mathcal{P}, R, W)$ vérifiant les propriétés suivantes :

$$\begin{array}{ll} \forall a \in \Sigma, & \emptyset \neq W(a) \subseteq R(a) \\ \forall a, b \in \Sigma, & W(a) \cap R(b) \neq \emptyset \Leftrightarrow W(b) \cap R(a) \neq \emptyset \end{array}$$

La première condition assure tout d'abord qu'il n'y a pas d'action complètement inutile. En effet une action dont le domaine d'écriture est vide ne peut agir sur le système. Cette condition assure également qu'une action ne peut pas écrire sur un processus dont elle ne peut lire la valeur.

La seconde condition assure que les communications sont symétriques. En effet, si une action a peut communiquer avec b via le processus i, c'est à dire si une action a peut écrire sur un processus i qui soit dans le domaine de lecture de b, alors il existe un processus j via lequel b peut communiquer avec a, c'est à dire un processus j sur lequel b peut écrire et qui est dans le domaine de lecture de a.

Ces architectures ont déjà été considérées, par exemple, dans [63]. Elles sont suffisantes pour définir un alphabet de dépendance sur Σ. On définit la relation de dépendance sur Σ par $\mathrm{D} = \{(a, b) \in \Sigma \times \Sigma \mid R(a) \cap W(b) \neq \emptyset\}$.

On dira de plus qu'une architecture est *cellulaire*, s'il existe une bijection entre $\mathcal{P}$ et Σ. On confond alors $\mathcal{P}$ et Σ et on exige également que pour tout a dans Σ, $W(a) = \{a\}$. Dans ces conditions, on peut se contenter de donner la relation de dépendance car $R(a) = W(a) = \{b \in \Sigma \mid a \,\mathrm{D}\, b\}$.

On introduit alors la notation suivante : si $t = (V, \leq, \ell)$ est une trace et J un sous ensemble de $\mathcal{P}$, la trace $\partial_J t$ est le préfixe de t défini par l'ensemble de sommets $U = \downarrow\{x \in V \mid W(\ell(x)) \cap J \neq \emptyset\}$.

Définition 5 (Automates Asynchrones).
Un automate asynchrone *sur l'architecture* $(\Sigma, \mathcal{P}, R, W)$ *est un quadruplet* $\mathcal{A} = ((Q_i)_{i\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, q^0, \mathcal{F})$ *où :*

- ▷ *Chaque* Q_i *est un ensemble fini* d'états locaux *pour le processus* $i \in \mathcal{P}$. *Les éléments de* $Q_{\mathcal{P}}$ *sont les états globaux de* $\mathcal{A}$.
- ▷ $q^0 \in Q_{\mathcal{P}}$ *est* l'état initial global *de* $\mathcal{A}$.
- ▷ $\delta_a : Q_{R(a)} \to 2^{Q_{W(a)}}$ *est la* fonction de transition *locale pour* a *dans* Σ.
- ▷ $\mathcal{F} \subseteq Q_{\mathcal{P}}$ *est la* condition d'acceptation.

Une exécution d'un automate asynchrone $\mathcal{A} = ((Q_i)_{i\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, q^0, \mathcal{F})$ sur une trace $t = (V, \leq, \ell)$ finie est une trace $t' = (V, \leq, \ell, \sigma)$ de $\mathbb{M}(\Sigma', \mathrm{D}')$ où σ est une fonction d'étiquetage telle que pour tout x dans V, $\sigma(x)$ est un élément de $Q_{W(\ell(x))}$.

Nous définissons alors le morphisme lw (*last write*): $\mathbb{M}(\Sigma', \mathrm{D}') \to Q^\bullet = (Q_i \uplus \{\bullet\})_{i\in\mathcal{P}}$ de la manière suivante :

- ▷ $\mathrm{lw}(r)_i = \bullet$ si pour tout sommet x de r, $i \notin W(\ell(x))$.

▷ $\mathrm{lw}(r)_i = \sigma(x)_i$ où x est le dernier sommet de r tel que $i \in W(\ell(x))$ sinon.

On introduit l'opération $q_0 * q_1 \in Q^\bullet$ où q_0 et q_1 sont dans $Q^\bullet$ par $(q_0 * q_1)_i = q_{1i}$ si $q_{1i} \neq \bullet$ et $(q_0 * q_1)_i = q_{0i}$ sinon.

On vérifie facilement que lw muni de cette opération est bien un morphisme. De plus pour tout t, $\mathrm{lw}(t) = \mathrm{lw}(\varepsilon) \Rightarrow t = \varepsilon$.

La trace étiquetée t' est alors une exécution de $\mathcal{A}$ sur t si elle vérifie la condition suivante :

Quelque soit e dans V, $\sigma(e) \in \delta_{\ell(e)}((q^0 * \mathrm{lw}(\Downarrow e))_{\mathrm{D}(\ell(e))})$.

Une exécution t' de l'automate $\mathcal{A}$ sur la trace t est acceptée par l'automate $\mathcal{A}$ si et seulement si $q^0 * \mathrm{lw}(t')$ est un élément de $\mathcal{F}$. On appelle $\mathcal{L}(\mathcal{A})$ l'ensemble des traces de $\mathbb{M}(\Sigma, \mathrm{D})$ sur lesquelles il existe une exécution acceptée par $\mathcal{A}$.

Si la trace $t' \in \mathbb{M}(\Sigma', \mathrm{D}')$ est une exécution de l'automate $\mathcal{A}$ sur la trace t, on appellera état global de t' l'élément $q^0 * \mathrm{lw}(t')$ de $Q_{\mathcal{P}}$.

2.5 Structures d'événements

Les ensembles partiellement ordonnés, étiquetés, munis d'une relation de conflit, introduits dans [40] sont appelés *structures d'événements premières*. Nous nous intéresserons uniquement à des *structures d'événements premières* et les appellerons donc simplement *structures d'événements*.

Définition 6 (Structures d'événements).
Une structure d'événements *(SE) sur Σ est un quadruplet $\rho = (V, \leq, \#, \ell)$, où $(V, \leq, \ell)$ est un ensemble partiellement ordonné étiqueté par Σ et $\# \subseteq V \times V$ est une relation symétrique non réflexive appelée la* relation de conflit *vérifiant la propriété d'héritage suivante :*

$$(e \# f \text{ et } f \leq g) \Longrightarrow e \# g \tag{2.1}$$

Les éléments de V sont les événements *de ρ. Si $\Sigma = \Sigma_1 \times \Sigma_2$ et $\ell(e) = (\ell_1(e), \ell_2(e))$, on écrira $(V, \leq, \#, \ell_1, \ell_2)$ au lieu de $(V, \leq, \#, \ell)$.*

Un ensemble partiellement ordonné étiqueté peut être considéré comme une structure d'événement dont la relation de conflit est vide. Étant donné une structure d'événement $\rho = (V, \leq, \#, \ell)$, un sous ensemble H de V est *sans conflit* si $(H \times H) \cap \# = \emptyset$. Comme la relation de conflit est non réflexive et héritée, on en déduit que $\leq \cap \# = \emptyset$, donc que si $H \subseteq V$ est sans conflit, $\downarrow H$ l'est également. Une *configuration* de ρ est un sous ensemble de V, fermé par le bas et sans conflit. L'ensemble de toutes les configurations de ρ sera noté $\mathcal{C}(\rho)$, l'ensemble des configurations finies de ρ sera noté $\mathcal{C}_{\mathrm{fin}}(\rho)$ et l'ensemble de toutes les configurations maximales de ρ sera noté $\mathcal{C}_{\max}(\rho)$.

Étant donné un ensemble partiellement ordonné étiqueté t, on dira que ρ est une *structure d'événement sur t* (noté t-SE) si tous les éléments de $\mathcal{C}_{\max}(\rho)$ sont isomorphes à t.

On dira que $C, D \subseteq V$ sont *compatibles* (on écrira $C \natural D$) si $C \cup D$ est sans conflit. On écrira simplement $e \natural f$ pour $\{e\} \natural \{f\}$. Deux événements compatibles non ordonnés seront dit *concurrents*. Un *conflit initial* est une paire d'éléments (e, f) telle que $e \# f$ et pour tout $e' \leq e$, $f' \leq f$, $e' \# f' \Rightarrow e = e'$ et $f = f'$. On écrit alors $e \#^i f$ si (e, f) est un conflit initial.

Si $\rho_1 = (V_1, \leq_1, \#_1, \ell_1)$ et $\rho_2 = (V_2, \leq_2, \#_2, \ell_2)$ sont deux structures d'événements, on définira la structure d'événements $\rho = \rho_1 \# \rho_2 = (V, \leq, \#, \ell)$ par :

- $V = V_1 \uplus V_2$
- $\leq = \leq_1 \uplus \leq_2$
- $\# = \#_1 \uplus \#_2 \uplus V_1 \times V_2 \uplus V_2 \times V_1$
- $\ell = \ell_1 \uplus \ell_2$

On dira que $\rho' = (V', \leq', \#', \ell')$ est une sous structure d'événement *induite* de $\rho = (V, \leq, \#, \ell)$ que l'on écrira $\rho' \subseteq \rho$ si $V' \subseteq V$ et $\leq'$, $\#'$, ℓ' sont les restrictions de $\leq$, $\#$, ℓ à V'.

Soit $\rho = (V, \leq, \#, \ell)$ et $\rho' = (V', \leq', \#', \ell')$ deux structures d'événements. Un *morphisme* $\varphi : \rho \to \rho'$ est une fonction de V sur V' telle que :

- $\ell'(\varphi(e)) = \ell(e)$ pour tout $e \in V$.
- $e \leq f$ entraine $\varphi(e) \leq' \varphi(f)$ pour tout $e, f \in V$.
- $e \natural f$ entraine $\varphi(e) \natural' \varphi(f)$ pour tout $e, f \in V$.

On définit $\varphi(\rho)$ comme la sous structure d'événement de ρ' induite par $\varphi(V)$.

Un *isomorphisme* est un morphisme bijectif $\varphi : \rho \to \rho'$ tel que $\varphi^{-1} : \rho' \to \rho$ est également un morphisme. On dit que ρ et ρ' sont isomorphes, noté $\rho \overset{\varphi}{\sim} \rho'$ (ou $\rho \sim \rho'$), s'il existe un isomorphisme φ entre ρ et ρ'. Ces définitions s'appliquent également aux ordres partiels étiquetés (considérés comme des structures d'événements avec une relation de conflit vide).

Lemme 2 *Soit ρ et ρ' deux structures d'événements finies. S'il existe deux morphismes bijectifs $\varphi : \rho \to \rho'$ et $\psi : \rho' \to \rho$, alors $\rho \sim \rho'$.*

Preuve Soit $\rho = (V, \leq, \#, \ell)$. Comme φ et ψ sont bijectifs, la fonction $\psi \circ \varphi$ est une permutation de V et il existe $n > 0$ tel que $(\psi \circ \varphi)^n = \mathrm{id}_{|V}$. Dans ces conditions, $\varphi^{-1} = (\psi \circ \varphi)^{n-1} \circ \psi$ est un morphisme. □

Deuxième partie

Logiques de Traces

Chapitre 3

Motivations

Le problème du "Model Checking" consiste à s'assurer algorithmiquement qu'un modèle d'un système vérifie une spécification. L'une des approches les plus fructueuses a été obtenue en utilisant des automates finis pour modéliser les systèmes et des logiques (en particulier les logiques temporelles) pour modéliser les spécifications ([43, 11, 25, 28, 48, 57, 60]).

Ces résultats étant particulièrement fructueux sur les systèmes séquentiels, il existe de nombreux travaux visant à les étendre à des systèmes distribués. En effet, un certain nombre de logiques (telle LTL par exemple) ont pour modèle des mots. Or un mot définit un ordre total sur un ensemble d'événements. Par exemple le mot ab place l'événement a avant l'événement b. Cependant, un système distribué où les événements a et b seraient concurrents ne devrait pas pouvoir distinguer les deux exécutions ab et ba.

Pour étendre le problème du "Model Checking" aux systèmes distribués, il est tout d'abord utile de considérer des modèles prenant en compte ce phénomène. Les ordres partiels, et en particulier les traces de Mazurkiewicz, que nous avons introduits section 2.3 sont un modèle naturel pour ce type de problème. L'indépendance de deux événements est alors symbolisée par le fait que ces deux événements sont incomparables, l'ordre représentant la dépendance d'un événement par rapport à ses prédécesseurs.

Une fois le modèle choisi, nous pouvons étendre le concept de logique temporelle à celui-ci. Cependant une difficulté se pose dès le départ. Pour les systèmes séquentiels, une formule est évaluée sur un mot à une certaine position, pour les systèmes distribués, il nous faut dès lors définir ce qu'est une position dans une trace.

Une position doit représenter l'état du système à un moment donné. Pour les systèmes distribués il existe deux types d'états :

- ▷ L'état global du système représente l'état exact à un moment donné de toutes les composantes du système distribué. Si on considère une trace de Mazurkiewicz t, on peut représenter les différents états globaux de t au cours du temps par l'ensemble des préfixes finis de t.
- ▷ L'état local d'un processus représente la vue du système à un moment donné pour un processus donné. Si on considère une trace de Mazurkiewicz t, on peut représenter les différents états locaux de t au cours du temps par l'ensemble des événements de t.

Ces deux possibilités de définition de l'état d'un système engendrent deux

types de logiques temporelles sur les traces. Quand une formule s'évalue sur l'état global du système, on parle de logique globale et lorsqu'une formule s'évalue sur un état local du système, on parle de logique locale.

Nous allons ci-après décrire quelques unes de ces logiques, en nous intéressant en particulier à la complexité du problème de satisfaisabilité. Nous montrerons alors que la relativement faible complexité de la logique globale ISTL$^\diamond$ peut s'expliquer par sa relation avec les logiques locales. Enfin nous nous intéresserons à une logique globale en particulier : la logiques des filtres.

Chapitre 4

Rappels sur les logiques

4.1 Les logiques temporelles sur les mots

Les logiques temporelles sont des logiques de propositions dont les formules permettent de spécifier le comportement d'un système au cours du temps. La plus connue d'entre elles est LTL.

Pour modéliser une formule de LTL, on considère un système qui évolue au cours du temps. Ce dernier est discret, on peut donc le modéliser par les entiers naturels, chaque entier représentant une date. À chaque date, le système vérifie un certain nombre de propriétés prises dans un ensemble fini P. On peut donc décrire une exécution d'un système par un mot sur l'alphabet 2^P. La première lettre contient l'ensemble des propriétés vérifiées par le système à l'instant 0. La seconde, l'ensemble des propriétés vérifiées par le système à l'instant 1 et ainsi de suite. À partir de maintenant, nous ne parlerons plus de propriétés. Nous nous plaçons directement sur l'alphabet $\Sigma = 2^P$, et une exécution du système est donc un mot de Σ^∞. Ces deux modélisations sont équivalentes.

La syntaxe de LTL est alors la suivante :

$$\mathrm{LTL}(\Sigma) ::= tt \mid a \in \Sigma \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\psi$$

Intuitivement, la formule $\mathsf{X}\,\varphi$ (*successeur*) est vraie au temps n, si la formule φ est vraie au temps $n+1$ et la formule $\varphi\,\mathsf{U}\,\psi$ (*"until"*) est vraie au temps n, s'il existe un temps n' plus grand que n tel que ψ est vraie au temps n' et que φ est vraie entre les temps n (inclus) et n' (exclus).

La sémantique formelle de LTL est la suivante.

Soit $w \in \Sigma^\infty$ un mot et $n \in \mathbb{N}$ un entier, nous avons :

$$\begin{cases} t,n \models tt & \\ t,n \models a & \text{si} \quad t_n = a \\ t,n \models \neg\varphi & \text{si} \quad t,n \not\models \varphi \\ t,n \models \varphi \wedge \psi & \text{si} \quad t,n \models \varphi \text{ et } t,n \models \psi \\ t,n \models \mathsf{X}\,\varphi & \text{si} \quad t,n+1 \models \varphi \\ t,n \models \varphi\,\mathsf{U}\,\psi & \text{si} \quad \exists m \geq n \text{ tel que } t,m \models \psi \text{ et pour tout } n' \text{ tel que} \\ & \qquad n \leq n' < m \text{ on a } t,n' \models \varphi \end{cases}$$

Cette syntaxe minimale nous permet de définir de nouveaux opérateurs :

$$
\begin{array}{lll}
\varphi \vee \psi & \Leftrightarrow & \neg(\neg\varphi \wedge \neg\psi) \\
\mathsf{F}\,\varphi \quad \text{(futur)} & \Leftrightarrow & tt\ \mathsf{U}\ \varphi \\
\mathsf{G}\,\varphi \quad \text{(toujours)} & \Leftrightarrow & \neg\,\mathsf{F}\,\neg\varphi \\
\varphi\ \mathsf{R}\ \psi \quad \text{(“release”)} & \Leftrightarrow & \neg(\neg\varphi\ \mathsf{U}\ \neg\psi)
\end{array}
$$

Il existe également des versions de LTL ayant des opérateurs pouvant exprimer des propriétés sur le passé [29, 20, 34].

Il existe de nombreuses autres logiques temporelles. Parmi elles, nous pouvons citer CTL [46, 10] qui est une logique branchante permettant de parler des arbres d'exécution des systèmes

4.2 Le problème de satisfaisabilité

Le problème de satisfaisabilité d'une logique est le suivant :

Étant donnée une formule φ de notre logique, nous voulons savoir s'il existe un modèle M tel que $M \models \varphi$.

Pour LTL ce problème est PSpace-Complet [49, 14]. L'algorithme consiste, à partir d'une formule φ de LTL, à construire un automate $\mathcal{A}_\varphi$ qui reconnaît exactement l'ensemble des mots qui modélisent la formule φ. Il suffit alors de vérifier si le langage de cet automate est non vide.

4.3 Le problème du “model checking”

Le problème du “model checking” d'une logique est le suivant :

Étant donnée une formule φ représentant une propriété, et étant donné un système nous souhaitons savoir si toutes les exécutions du système vérifient la propriété exprimée par φ.

Dans notre cas, le système est représenté par un automate et la formule est une formule de LTL. Encore une fois, ce problème est alors PSpace-Complet [49, 14]. Il suffit de faire l'intersection de l'automate représentant le système avec l'automate représentant tous les modèles de la formule $\neg\varphi$. Cet automate reconnaît alors un langage autre que le langage vide si et seulement s'il existe une exécution du système qui ne vérifie pas la propriété décrite par φ.

4.4 Satisfaisabilité et “model checking”

Nous avons vu ci-dessus qu'un moyen pour résoudre le problème du “model checking” consistait, pour une formule φ à calculer une représentation, sous forme d'un automate, de tous les modèles de φ. Si un tel procédé est possible, il nous donne le moyen de vérifier si une formule φ est satisfaisable. Il suffit en effet de calculer l'automate reconnaissant tous les modèles de φ, puis de vérifier que cet automate reconnaît au moins un mot.

Nous en déduisons que cette méthode pour résoudre le problème du “model checking” est au moins aussi difficile algorithmiquement que le problème de la satisfaisabilité.

Nous nous intéressons donc au problème de la satisfaisabilité des logiques de traces afin de trouver des bornes au problème du “model checking”.

Chapitre 5

Les logiques locales

5.1 Tlc

La logique Tlc (Temporal Logic for Causality) a été introduite en [3]. Il s'agit d'une extension de LTL aux traces de Mazurkiewicz.

La syntaxe de Tlc est la suivante :

$$\text{Tlc}(\Sigma) ::= tt \mid a \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{EX}\,\varphi \mid \mathsf{EY}\,\varphi \mid \mathsf{Eco}\,\varphi \mid \mathsf{EG}\,\varphi \mid \varphi\,\mathsf{EU}\,\psi \mid \varphi\,\mathsf{ES}\,\psi$$

De manière intuitive, la formule $\mathsf{EX}\,\varphi$ (successeur existentiel) est vraie sur le sommet x, s'il existe un sommet immédiatement successeur de x qui vérifie la formule φ. De même, la formule $\mathsf{EY}\,\varphi$ (prédécesseur existentiel) est vraie sur le sommet x, s'il existe un sommet y immédiatement prédécesseur de x qui vérifie la formule φ. La formule $\mathsf{Eco}\,\varphi$ (concurrent existentiel) est vraie sur le sommet x, s'il existe un sommet concurrent de x qui vérifie la formule φ. La formule $\mathsf{EG}\,\varphi$ (toujours existentiel) est vraie sur le sommet x s'il existe un chemin maximal partant de x pour lequel la formule φ est vraie sur chaque sommet. La formule $\varphi\,\mathsf{EU}\,\psi$ ("until" existentiel) est vraie sur un sommet x, s'il existe un sommet y dans le futur de x et un chemin de x à y tels que la formule φ est vraie sur tous les sommets du chemin sauf peut-être y et que la formule ψ est vraie en y. Enfin la formule $\varphi\,\mathsf{ES}\,\psi$ ("since" existentiel) est vraie sur un sommet x, s'il existe un sommet y dans le passé de x et un chemin de y à x tels que la formule φ est vraie sur tous les sommets du chemin sauf peut-être y et que la formule ψ est vraie en y.

La sémantique formelle de Tlc est la suivante :

Soit $t = (V, \leq, \ell)$ une trace sur l'alphabet de dépendance (Σ, D), soit $x \in V$

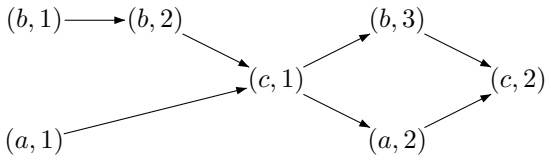
un sommet de t, nous avons :

$$\begin{cases} t,x \models tt & \\ t,x \models a & \text{si } \ell(x) = a \\ t,x \models \neg\varphi & \text{si } t,x \not\models \varphi \\ t,x \models \varphi \wedge \psi & \text{si } t,x \models \varphi \text{ et } t,x \models \psi \\ t,x \models \mathsf{EX}\,\varphi & \text{si } \exists y \in V \text{ tel que } x \lessdot y \text{ et } t,y \models \varphi \\ t,x \models \mathsf{EY}\,\varphi & \text{si } \exists y \in V \text{ tel que } y \lessdot x \text{ et } t,y \models \varphi \\ t,x \models \mathsf{Eco}\,\varphi & \text{si } \exists y \in V \text{ tel que } x \not\leq y \text{ et } y \not\leq x \text{ et } y \models \varphi \\ t,x \models \mathsf{EG}\,\varphi & \text{si il existe un chemin maximal } y_0, y_1 \dots y_n \dots \text{ tel que} \\ i & \quad y_0 = x \text{ et } \forall i \;\; t,y_i \models \varphi \\ t,x \models \varphi\,\mathsf{EU}\,\psi & \text{si } \exists n \in \mathbb{N} \text{ et } y_0 \cdots y_n \in V \text{ tels que } y_0 = x, \\ & \quad \forall i < n \;\; y_i \lessdot y_{i+1}, \forall i < n \;\; t,y_i \models \varphi \text{ et } t,y_n \models \psi \\ t,x \models \varphi\,\mathsf{ES}\,\psi & \text{si } \exists n \in \mathbb{N} \text{ et } y_0 \cdots y_n \in V \text{ tels que } y_0 = x, \\ & \quad \forall i < n \;\; y_{i+1} \lessdot y_i, \forall i < n \;\; t,y_i \models \varphi \text{ et } t,y_n \models \psi \end{cases}$$

Nous pouvons alors définir les opérateurs suivants :

$\varphi \vee \psi$		$\Leftrightarrow$	$\neg(\neg\varphi \wedge \neg\psi)$
$\mathsf{AX}\,\varphi$	(successeur universel)	$\Leftrightarrow$	$\neg\,\mathsf{EX}\,\neg\varphi$
$\mathsf{AY}\,\varphi$	(prédécesseur universel)	$\Leftrightarrow$	$\neg\,\mathsf{EY}\,\neg\varphi$
$\mathsf{EF}\,\varphi$	(futur existentiel)	$\Leftrightarrow$	$tt\;\mathsf{EU}\,\varphi$
$\mathsf{AG}\,\varphi$	(toujours universel)	$\Leftrightarrow$	$\neg\,\mathsf{EF}\,\neg\varphi$

Considérons alors la trace t suivante (le sommet (l,n) est étiquetté par l) :



Nous avons donc par exemple :

$$t,(a,1) \models a\,\mathsf{EU}\,c$$
$$t,(b,1) \models b\,\mathsf{EU}\,c$$
$$t,(a,1) \models \mathsf{EX}\,c$$
$$t,(c,1) \models \mathsf{AX}\,\neg c$$

Il nous reste à définir la notion de formule initiale. En effet, si on veut définir le fait qu'une trace satisfait une formule, il nous faut décider sur quel sommet on va évaluer cette formule. Sur les mots, la question se résout en évaluant la formule à la position 0, mais sur les traces, il peut exister plusieurs sommets initiaux. Plusieurs solutions ont été proposées pour résoudre ce problème (voir par exemple [16]). On peut introduire une nouvelle modalité (EM) qui s'évalue directement sur les traces. Sa sémantique est la suivante :

$t \models \mathsf{EM}\,\varphi$ si et seulement s'il existe un sommet minimal de t, x, tel que $t,x \models \varphi$.

Une autre solution est d'introduire une nouvelle lettre à l'alphabet. On nomme $\bullet$ cette lettre. On étend la relation de dépendance D à $\bullet$ de la manière suivante : $\bullet\,\mathrm{D}\,a$ pour tout a dans $\Sigma \uplus \{\bullet\}$, et on dira qu'une trace $t \in \mathbb{R}(\Sigma, \mathrm{D})$ vérifie une formule φ si et seulement si $\bullet \cdot t, \bullet \models \varphi$.

Ces deux solution ne sont pas équivalentes pour le pouvoir d'expression. Cependant tout ce qui est exprimable à l'aide de la modalité EM l'est également avec l'ajout du sommet $\bullet$.

Maintenant que nous pouvons parler du problème de satisfaisabilité de TLC, il a été montré dans [3], que ce problème était dans PSPACE en utilisant une construction de tableaux. Ce résultat est également une conséquence du résultat plus général dont nous parlons dans la section 5.2. Un résultat d'expressivité à été exprimé dans [16]. Il a été montré que TLC a le même pouvoir d'expression que $\mathrm{FO}_\Sigma(<)$ (la logique du première ordre) si et seulement si l'alphabet de dépendance est série parallèle. En effet comme la modalité EU n'est pas exprimable par une formule du premier ordre sur des alphabets plus généraux, TLC n'est pas contenu dans $\mathrm{FO}_\Sigma(<)$ de manière générale. De plus des extensions de TLC ont été proposé dans [27].

5.2 Logiques temporelles définissables en MSO

Dans [21], Paul Gastin et Dietrich Kuske introduisent un cadre général pour définir des logiques locales sur les traces. Ils s'intéressent à toutes les logiques locales ayant un nombre fini de modalités, toutes définissables en logique du second ordre. Ce cadre englobe un très grand nombre de logiques locales introduites auparavant. De plus, il donne un outil général pour concevoir de nouvelles logiques locales correspondant à des besoins particuliers.

Définition 7 (Logiques temporelles définissables en MSO).
Pour définir une logique temporelle, on commence par se donner un nombre fini de modalités B et une fonction d'arité : $\mathrm{arity} : B \to \mathbb{N}$.

La syntaxe d'une formule de $\mathrm{TL}(B)$ *est alors :*

$$\varphi ::= \sum_{M \in B} M(\underbrace{\varphi, \cdots, \varphi}_{\mathrm{arity}(M)})$$

La logique $\mathrm{TL}(B)$ *est alors définissable en* MSO *si et seulement si pour tout M dans B, il existe une formule* MSO, $[\![B]\!]$ *avec* $\mathrm{arity}(B)$ *variables libres du second ordre et une variable libre du première ordre telle que la sémantique d'une formule φ de* $\mathrm{TL}(B)$ *puisse alors se définir par induction de la manière suivante :*

Si $\varphi = M(\varphi_1, \cdots, \varphi_n)$ et $[\![\varphi]\!] = \alpha(X_1, \cdots X_n, p)$, alors $t, x \models \varphi$, si et seulement si $t, \nu \models \alpha(X_1, \cdots X_n, p)$ où ν est l'assignation qui à p associe x et à X_k associe l'ensemble V_k des sommets y tels que $t, y \models \varphi_k$.

En particulier la logique TLC est alors définissable en MSO.

On commence par introduire la formule $\mathrm{CHAIN}(X)$ qui est satisfaite si et seulement si X est une chaîne finie pour la relation successeur :

$\mathrm{CHAIN}(X) \Leftrightarrow \exists x \in X \big(\forall y \in X \quad y \geq x\big) \wedge \exists x \in X \big(\forall y \in X \big[y \neq x \Rightarrow \big(y < x \wedge \exists! z \in X \big[y \lessdot z\big]\big)\big]\big)$

Cette formule signifie que X est un ensemble qui contient un unique élément maximal et un unique élément minimal et que pour tout élément y de X qui n'est pas son élément maximal, il existe un unique élément successeur de y.

On définit également la formule $\mathrm{MAXCHAIN}(X)$ qui est satisfaite si et seulement si X est une chaîne qui ne peut pas être prolongée sur la droite :

$\text{MAXCHAIN}(X) \Leftrightarrow \exists x \in X\big(\forall y \in X\big(y \geq x \wedge (\forall z \;\; y < z \Rightarrow \exists! e \in X \;\; y \lessdot e\big)\big)\big)$

La modalité $\exists! x\varphi(x)$ est la modalité qui exprime qu'il existe un unique x rendant la formule $\varphi(x)$ et peut s'exprimer de la manière suivante $\exists! x\varphi(x) \Leftrightarrow \exists x\big(\varphi(x) \wedge \forall y\big(y \neq x \Rightarrow \neg\varphi(y)\big)\big)$.

La formule $\text{MAXCHAIN}(X)$ signifie que X est un ensemble qui contient un unique élément minimal, et que pour chaque élément y de X, si y a au moins un successeur, alors un et un seul successeur de y est dans X.

On peut alors définir :

$$
\begin{array}{lcl}
[\![tt]\!](x) & = & tt \\
[\![a]\!](x) & = & (\ell(x) = a) \\
[\![\neg]\!](X, x) & = & x \notin X \\
[\![\wedge]\!](X_1, X_2, x) & = & x \in X_1 \wedge x \in X_2 \\
[\![\mathsf{EX}]\!](X, x) & = & \exists y \in X \;\; x \lessdot y \\
[\![\mathsf{EY}]\!](X, x) & = & \exists y \in X \;\; y \lessdot x \\
[\![\mathsf{Eco}]\!](X, x) & = & \exists y \in X (x \not\leq y \wedge y \not\leq x) \\
[\![\mathsf{EG}]\!](X, x) & = & \exists Y \big(\text{MAXCHAIN}(Y) \wedge x \in Y \wedge \forall y \in Y \big(y \geq x \Rightarrow y \in X\big)\big) \\
[\![\mathsf{EU}]\!](X_1, X_2, x) & = & \exists Y \exists y \big(\text{CHAIN}(Y) \wedge x \in Y \wedge y \in Y \wedge x \leq y \wedge y \in X_2 \wedge \\
 & & \forall z \big(z \in Y \wedge z \geq x \wedge z < y \Rightarrow z \in X_1\big)\big) \\
[\![\mathsf{ES}]\!](X_1, X_2, x) & = & \exists Y \exists y \big(\text{CHAIN}(Y) \wedge x \in Y \wedge y \in Y \wedge x \geq y \wedge y \in X_2 \wedge \\
 & & \forall z \big(z \in Y \wedge z \leq x \wedge z > y \Rightarrow z \in X_1\big)\big)
\end{array}
$$

La plupart des ces formules sont très simples. Revenons cependant sur $[\![\mathsf{EU}]\!](X_1, X_2, x)$ ($[\![\mathsf{ES}]\!](X_1, X_2, x)$ est quasiment identique). Cette formule dit qu'il existe une chaîne finie qui contient x et un élément y de cette chaîne qui est plus grand que x, telle que tous les élément situés entre x (compris) et y (non compris) sont dans X_1 et telle que y est dans X_2. Si X_1 contient tous les sommets vérifiant la formule φ et X_2 tous les sommets vérifiant la formule ψ, cette formule est alors bien équivalente au fait que x vérifie la formule $\varphi\,\mathsf{EU}\,\psi$.

Dans [21], les auteurs montrent que toutes les logiques entrant dans ce cadre ont une complexité pour le problème de satisfaisabilité et pour le problème du "model checking" dans PSpace. Pour montrer ce résultat, ils commencent par le montrer sur les mots, puis étendent le résultat aux traces en utilisant le fait que l'on peut exprimer en MSO le fait que deux sommets de la linéarisation d'une traces sont ordonnés dans la trace d'origine.

Ce résultat étend de nombreux résultats précédemment connus sur les logiques locales sur les traces. De plus, il assure que toutes les nouvelles logiques que l'on peut construire dans ce cadre ne sont pas plus dures pour les problèmes de satisfaisabilité et de "model checking" que LTL sur les mots. Cela nous offre donc un outil générique pour spécifier des systèmes distribués. En particulier nous allons utiliser ce cadre et ce résultat pour montrer que la complexité du problème de satisfaisabilité de la logique globale $\text{ISTL}^{\diamond}$ est EXPSpace.

Chapitre 6

Les logiques globales

6.1 LTrL

La syntaxe de LTrL est la suivante :

$$\mathrm{LTrL}(\Sigma) ::= tt \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a\rangle\varphi \mid \varphi \,\forall\mathsf{U}\, \psi \mid \langle a^{-1}\rangle tt$$

Il s'agit d'une logique inspirée de LTL. De plus elle n'est pas pure future en raison des constantes $\langle a^{-1}\rangle tt$.

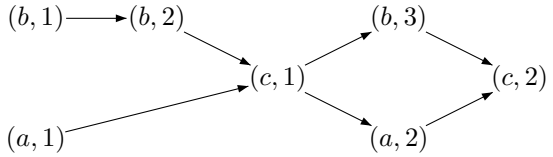
L'opérateur $\langle a\rangle\varphi$ est un opérateur de type *Next*, souvent noté $\exists\mathsf{X}_a$, cependant nous avons préféré adopter cette nouvelle notation afin d'être cohérents avec la notation $\langle A\rangle\varphi$ qui apparaît pour d'autres logiques globales.

Sa sémantique est la suivante :

Soit t une trace sur l'alphabet de dépendance (Σ, D), soit r un préfixe fini de t, nous avons :

$$\begin{cases} t, r \models tt \\ t, r \models \neg\varphi & \text{si} \quad t, r \not\models \varphi \\ t, r \models \varphi \wedge \psi & \text{si} \quad t, r \models \varphi \text{ et } t, r \models \psi \\ t, r \models \langle a\rangle\varphi & \text{si} \quad r \cdot a \leq t \text{ et } t, r \cdot a \models \varphi \\ t, r \models \varphi \,\forall\mathsf{U}\, \psi & \text{si} \quad \exists s \text{ tel que } r \leq s \leq t \;\; t, s \models \psi \text{ et } \forall u,\, r \leq u < s \Rightarrow t, u \models \varphi \\ t, r \models \langle a^{-1}\rangle tt & \text{si} \quad a \in \ell \circ \max(r) \end{cases}$$

Si nous reprenons la trace d'exemple de la section 5.1 :



Nous avons alors, par exemple :

$$\begin{array}{l} t, \varepsilon \models \langle a\rangle\langle b\rangle tt \\ t, \varepsilon \models \langle b\rangle\langle a\rangle tt \\ t, \varepsilon \models \langle a\rangle\langle a^{-1}\rangle tt \\ t, \varepsilon \models \langle a\rangle(\langle b\rangle tt \,\forall\mathsf{U}\, \langle c\rangle tt) \end{array}$$

mais :

$$t, \varepsilon \not\models \langle a \rangle tt \; \forall \mathsf{U} \; \langle c \rangle tt$$

Cette logique a été introduite dans [50]. Le problème de satisfaisabilité de cette logique est non élémentaire. La preuve en a été donnée dans [59] en simulant une machine de Turing à l'aide d'un codage astucieux de compteurs. Une procédure de décision non élémentaire a été donnée dans [22]. Les deux opérateurs qui, dans cette construction, entraînent l'explosion de la procédure de décision sont la négation et le $\forall \mathsf{U}$.

6.2 ISTL

Il s'agit d'une logique plus riche que LTrL, sa syntaxe est la suivante :

$$\mathrm{ISTL}(\Sigma) ::= tt \mid \langle a^{-1} \rangle tt \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi \mid \varphi \; \forall \mathsf{U} \; \psi \mid \varphi \; \exists \mathsf{U} \; \psi \mid \; \exists \mathsf{G} \; \varphi$$

La sémantique des opérateurs non encore définis est la suivante :

Soit t une trace sur l'alphabet de dépendance (Σ, D), soit r un préfixe fini de t, nous avons :

$$\begin{cases} t, r \models \varphi \; \exists \mathsf{U} \; \psi \quad \text{si} & \text{il existe une suite finie } r_0 \dots r_n \text{ telle que :} \\ & 1)\ r = r_0 \\ & 2)\ r_i \lessdot r_{i+1} \text{ pour } 0 \leq i < n \\ & 3)\ r_n \models \psi \\ & 4)\ r_i \models \varphi \text{ pour } 0 \leq i < n \\ t, r \models \exists \mathsf{G} \varphi \quad \text{si} & \text{il existe une suite finie ou infinie } r_0 r_1 \dots \text{ telle que :} \\ & 1)\ r_0 = r \\ & 2)\ \forall i \; r_i \lessdot r_{i+1} \\ & 3)\ \forall i \; r_i \models \varphi \\ & 4)\ \forall s \in \mathbb{M}(\Sigma, \mathrm{D}) \;\; s \leq t \Rightarrow \exists i \;\; s \leq r_i \end{cases}$$

Nous avons par exemple :

$$t, \varepsilon \models \langle a \rangle \langle a^{-1} \rangle tt$$
$$t, \varepsilon \models \neg \; \exists \mathsf{G} \; \langle a^{-1} \rangle tt$$
$$t, \varepsilon \models \langle a \rangle tt \; \exists \mathsf{U} \; \langle c \rangle tt$$

Ce dernier exemple illustre la différence entre l'opérateur $\forall \mathsf{U}$ de LTrL et ce nouvel opérateur $\exists \mathsf{U}$ qui est moins contraignant.

Cette logique est la première logique globale sur les traces à avoir été étudiée. Elle a été introduite dans [30, 31]. Le problème de satisfaisabilité a été montré indécidable dans [41], puis il a été montré dans [2] qu'il suffisait de l'opérateur $\exists \mathsf{U}$ pour arriver à ce résultat. La preuve montre que le problème de décidabilité est équivalent au problème de l'arrêt d'une machine de Turing. Le codage de la machine est très proche de celui utilisé dans [59].

6.3 ISTL$^{\diamond}$

Il s'agit d'un fragment de ISTL, sa syntaxe est la suivante :

$$\text{ISTL}^{\diamond}(\Sigma) ::= tt \mid \langle a^{-1}\rangle tt \mid \neg\varphi \mid \varphi \wedge \psi \mid \exists \mathsf{F}\, \varphi$$

La sémantique des opérateurs non encore définis est la suivante :

Soit t une trace sur l'alphabet de dépendance (Σ, D), soit r un préfixe fini de t, nous avons :

$$t, r \models \exists \mathsf{F} \varphi \quad \text{si} \quad \exists s \in \mathbb{M}(\Sigma, \mathrm{D}) \text{ tel que } r \leq s \leq t \text{ et } t, s \models \varphi$$

L'opérateur $\exists \mathsf{F}$ peut s'obtenir à partir de $\forall \mathsf{U}$ ou $\exists \mathsf{U}$ de la façon suivante :

$$\exists \mathsf{F} \varphi = tt\, \forall \mathsf{U}\, \varphi = tt\, \exists \mathsf{U}\, \varphi$$

L'intérêt de cette logique par rapport aux logiques présentées jusqu'à présent est la faible complexité du problème de satisfaisabilité. En effet le problème de satisfaisabilité de cette logique a été démontré EXPSpace dans [2].

Chapitre 7

Preuve de la complexité de ISTL$^{\diamond}$

Nous allons tout d'abord présenter la preuve de [2] montrant que le problème de satisfaisabilité de ISTL$^{\diamond}$ est EXPSpace. Puis partant de certains résultats intermédiaires, nous allons faire un lien entre ISTL$^{\diamond}$ et une nouvelle logique locale crée dans ce but. Nous allons alors présenter une nouvelle preuve du résultat de [2] en utilisant cette nouvelle logique locale. C'est donc la faible complexité des logiques locales que nous avons montré à la section 5.2 qui explique la faible complexité (comparée aux autres logiques globales) de ISTL$^{\diamond}$.

7.1 Preuve historique

Nous allons tout d'abord reprendre la preuve de décidabilité de ISTL$^{\diamond}$ contenue dans [2].

Pour ce faire, nous commençons par définir un sous ensemble de formules de ISTL$^{\diamond}$ que nous considérerons comme étant en forme normale. Ces formules sont définies ainsi :

- ▷ Toute constante (à savoir les formules du type $\langle a^{-1}\rangle tt$) est en forme normale.
- ▷ $\exists \mathsf{F} \bigwedge_k \varphi_k$ est en forme normale, si tous les φ_k sont en forme normale.
- ▷ La négation d'une formule en forme normale est en forme normale.

De plus nous nommons "eventuality formula" , tout formule de ISTL$^{\diamond}$ de la forme $\exists \mathsf{F}\varphi$.

À partir de cette forme, nous pouvons alors considérer 3 propositions introduites dans [2].

La première relie les "*eventuality formula*" aux "*eventuality formula*" en forme normale.

Proposition 3 *Pour toute "eventuality formula" φ de* ISTL$^{\diamond}$*, il existe une formule ψ de* ISTL$^{\diamond}$ *telle que :*

1. *φ et ψ sont équivalentes.*
2. *ψ est une disjonction d'au plus $2^{|\varphi|}$ "eventuality formula" en forme normale.*

3. *Le nombre de sous "eventuality formula" distinctes dans ψ est au plus* $2^{|\varphi|+1}$.

La seconde proposition donne des propriétés sur les différents préfixes d'une même trace qui vérifie une "eventuality formula" en forme normale.

Proposition 4 *Soit φ une "eventuality formula" en forme normale, soit $t \in \mathbb{R}(\Sigma, \mathrm{D})$, soit r et s deux préfixes de t, alors :*

$$t, r \models \varphi \text{ et } t, s \models \varphi \Rightarrow t, r \cup s \models \varphi$$

Enfin la troisième proposition introduit la notion de préfixe maximal vérifiant une "eventuality formula" en forme normale.

Proposition 5 *Pour toute "eventuality formula" φ en forme normale, et pour toute trace t, il existe un préfixe maximal, notée $\max_\varphi(t)$, qui est l'unique trace telle que $\max_\varphi(t) \leq t$ et telle que $\forall r \in \mathbb{M}(\Sigma, \mathrm{D})\ \ t, r \models \varphi \Leftrightarrow r \leq \max_\varphi(t)$.*

La preuve de ces trois propositions se trouve dans [2]. De plus, les auteurs montrent également que nous ne perdons pas de pouvoir d'expression en ne considérant que les "eventuality formula" car toute formule de ISTL$^\diamond$ est équivalente à une combinaison booléenne de "eventuality formula".

Nous allons maintenant considérer une "eventuality formula" φ et une trace t. Soit ψ la formule obtenue à l'aide de la proposition 3 et soit Ξ_ψ l'ensemble des sous "eventuality formula" en forme normale contenues dans ψ. Nous savons que $|\Xi_\psi| \leq 2^{|\varphi|+1}$ et que ψ est disjonction d'élément de Ξ_ψ d'après la proposition 3.

Nous considérons alors le nouvel alphabet de dépendance $(\overline{\Sigma}, \overline{\mathrm{D}})$ avec :

- $\overline{\Sigma} = \Sigma \times 2^{\Xi_\psi}$
- $\forall (a, \Xi_1), (b, \Xi_2) \in \overline{\Sigma}\ \ (a, \Xi_1)\ \overline{\mathrm{D}}\ (b, \Xi_2) \Leftrightarrow a\ \mathrm{D}\ b$

Pour tout élément $e = (a, \xi)$ de $\overline{\Sigma}$, nous définissons $\ell(e) = a$ et $\mu(e) = \xi$.

Nous considérons alors l'ensemble des traces $\overline{t} = (E, \leq, \ell \times \mu) \in \mathbb{R}(\overline{\Sigma}, \overline{\mathrm{D}})$ sur ce nouvel alphabet. Il existe une projection de $\mathbb{R}(\overline{\Sigma}, \overline{\mathrm{D}})$ dans $\mathbb{R}(\Sigma, \mathrm{D})$ qui à la trace $\overline{t}$ associe la trace $t = (E, \leq, \ell)$, du fait de la forme de la relation $\overline{\mathrm{D}}$.

Ceci nous permet d'étiqueter chacun des sommets d'une trace t par un sous ensemble de Ξ_ψ.

Pour tout $\xi \in \Xi_\psi$, nous appelons c_ξ l'ensemble des sommets x tels que $\xi \in \mu(x)$.

Nous avons alors la proposition suivante [2] :

Proposition 6 *Soit ψ une "eventuality formula" en forme normale de* ISTL$^\diamond(\Sigma, \mathrm{D})$. *La forumule ψ est satisfaisable si et seulement si l'une des deux conditions suivante est vérifiée :*

- $\varepsilon, \varepsilon \models \psi$
- $\exists \overline{t} = (E, \leq, \ell \times \mu) \in \mathbb{R}(\overline{\Sigma}, \overline{\mathrm{D}})$ *qui satisfait les conditions suivantes :*

  1. *Pour tout $\xi \in \Xi_\psi$, $\xi = \exists \mathsf{F} \bigwedge_k \xi_k$, $\overline{t}$ est bien étiquetée pour ξ. C'est à dire :*

(a) *L'ensemble* c_ξ *est un préfixe, c'est à dire :*

$$\forall x \in c_\xi \;\; \forall y \leq x \;\; y \in c_\xi$$

(b) *Tout préfixe fini* $\overline{r}$ *contenu dans* c_ξ *est tel que* $t, r \models \xi$. *(Si* c_ξ *est un ensemble fini, cela revient à dire que le préfixe* $\overline{r}$ *défini par* c_ξ *est tel que* $t, r \models \xi$*).*

(c) *Il n'existe pas de préfixe fini* $\overline{r}$*, non contenu dans* c_ξ *tel que* $t, r \models \bigwedge_k \xi_k$.

2. c_ψ *n'est pas vide.*

Dans ces conditions l'ensemble des traces $t \in \mathbb{R}(\Sigma, \mathrm{D})$ *qui satisfont* ψ *est la projection de l'ensemble des traces* $\overline{t} \in \mathbb{R}(\overline{\Sigma}, \overline{\mathrm{D}})$ *qui vérifient ces propriétés.*

À partir de cette propriété, nous pouvons construire un automate de Büchi qui reconnaît l'ensemble des traces de $\mathbb{R}(\overline{\Sigma}, \overline{\mathrm{D}})$ qui vérifient ces contraintes.

Si nous considérons maintenant une "eventuality formula" $\varphi \in \mathrm{ISTL}^\diamond$, d'après la proposition 3, nous pouvons écrire φ comme une disjonction de "eventuality formula" en forme normale. Pour chacune d'entre elles, nous considérons l'automate présenté ci-dessus, puis nous en faisons la disjonction. [2] montre que nous obtenons un automate en $O(2^{2^{|\psi|}})$ états. Le problème de satisfaisabilité est alors équivalent au problème du vide de cet automate et se résout donc en temps $O(2^{2^{|\psi|}})$.

7.2 Nouvelle preuve

Nous allons reprendre la preuve de [2] en montrant qu'il existe un lien relativement important avec les logiques locales. Ainsi nous allons montrer que la satisfaisabilité de $\mathrm{ISTL}^\diamond$ se réduit au problème de satisfaisabilité de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ où $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ est une logique locale.

7.2.1 $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$

La syntaxe de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ est la suivante :

$$\begin{aligned}\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})(\Sigma) ::= \quad & tt \mid a \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{EX}\,\varphi \mid \mathsf{EF}\,\varphi \\ & \mid \mathsf{Conf}(A, B, \varphi, \psi) \;\; A, B \subseteq \Sigma\end{aligned}$$

Pour définir la sémantique des opérateurs non encore définis, nous allons introduire une nouvelle formule globale : $\mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi)$ telle que $t, r \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi)$ si et seulement si :

- $A \subseteq \ell \circ \max(r)$
- $B \cap \ell \circ \max(r) = \emptyset$
- $\forall y \in \max(r) \;\; t, y \models \varphi$
- $\exists y \in \max(r) \;\; t, y \not\models \psi$

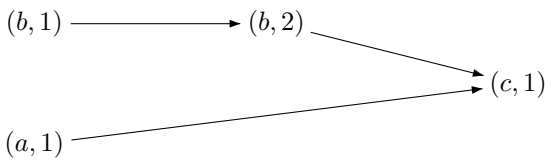
Dans ces conditions, on a :

$$\begin{cases} t, x \models \mathsf{EF}\,\varphi & \text{si} \quad \exists y \geq x \text{ tel que } t, y \models \varphi \\ t, x \models \mathsf{Conf}(A, B, \varphi, \psi) & \text{si} \quad \text{il existe un préfixe fini } r \text{ de } t \text{ tel que} \\ & \quad 1)\ x \in \max(r) \\ & \quad 2)\ t, r \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi) \end{cases}$$

Par exemple, si nous considérons la relation de dépendance suivante :

$$a \quad \text{——} \quad c \quad \text{——} \quad b$$

nous considérons la trace suivante :



qui est telle que :

$$\overline{t}, (a, 1) \models \mathsf{Conf}(\{a, b\}, \{c\}, a \vee (b \wedge \mathsf{EX}\, b), \mathsf{EX}\, b)$$
$$\overline{t}, (a, 1) \models \mathsf{Conf}(\{a, b\}, \{c\}, \mathsf{EX}\, c, b)$$

Cette logique n'est pas une logique naturelle, mais elle présente l'avantage de relier les logiques globales et les logiques locales grâce à l'opérateur Conf qui permet de considérer les éléments maximaux d'un préfixe.

7.2.2 Une autre approche du processus de décision de ISTL$^\diamond$

Notre travail va consister à construire une formule de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ qui va assurer les contraintes de la proposition 6.

Nous introduisons la notation suivante :

Si α est une "eventuality formula" en forme normale, nous écrivons $\mathrm{Now}(\alpha)$ la formule α privée de la modalité $\exists\mathsf{F}$ initiale. Nous étendrons la fonction Now aux formules de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ de la forme $\varphi = \mathsf{EF}\,\psi$ par $\mathrm{Now}(\varphi) = \psi$.

Nous considérons donc α une "eventuality formula" en forme normale de ISTL$^\diamond$. Nous prenons alors comme ensemble Ξ_α l'ensemble des sous "eventuality formula" en forme normale de α.

Soit $\beta \in \Xi_\alpha$, alors β est une "eventuality formula" en forme normale et β s'écrit :

$$\beta = \exists\mathsf{F}\left(\left(\bigwedge_{a\in A}\langle a^{-1}\rangle tt\right) \wedge \left(\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt\right) \wedge \left(\bigwedge_{\varphi\in\Phi}\varphi\right) \wedge \left(\bigwedge_{\psi\in\Psi}\neg\psi\right)\right)$$

où les $A, B \subseteq \Sigma$ et $\Phi, \Psi \subseteq \Xi_\alpha$.

Nous construisons alors par induction la formule suivante de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$:

$$\text{Etiquetage}(\beta) = \mathsf{EF}\left(\bigwedge_{\psi\in\Psi} \mathsf{Conf}\left(A, B, \bigwedge_{\varphi\in\Phi} \text{Etiquetage}(\varphi), \text{Etiquetage}(\psi)\right)\right)$$

La formule φ étant de taille strictement plus petite que la formule β, nous finissons par devoir calculer la fonction Etiquetage appliqué au vide à laquelle nous donnons la valeur tt.

Nous étendons la fonction Etiquetage à la formule $\text{Now}(\beta)$ par :

$$\text{Etiquetage}(\text{Now}(\beta)) = \bigwedge_{\psi\in\Psi} \mathsf{Conf}\left(A, B, \bigwedge_{\varphi\in\Phi} \mathsf{EF}\,\text{Etiquetage}(\text{Now}(\varphi)), \mathsf{EF}\,\text{Etiquetage}(\text{Now}(\psi))\right)$$

Nous remarquons que $\text{Now}(\text{Etiquetage}(\beta)) = \text{Etiquetage}(\text{Now}(\beta))$.

Nous introduisons la formule Etiquetage, car nous allons montrer la propriété suivante ($\bullet$ est une lettre supplémentaire, dépendante de toutes les lettres de Σ, voir section 5.1) :

Proposition 7 *Soit t une trace de $\mathbb{R}(\Sigma, \mathrm{D})$, alors pour tout $r \leq t$ on a $t, r \models \beta$ si et seulement si pour tout $x \in \max(\bullet r)$, $\bullet t, x \models \text{Etiquetage}(\beta)$.*

Pour montrer ce résultat, nous allons avoir besoin d'un résultat semblable à la proposition 4 pour la logique $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$.

Lemme 8 *Soit t une trace de $\mathbb{R}(\Sigma, \mathrm{D})$, soient $A, B \subseteq \Sigma$ et φ, ψ_1, ψ_2 des formules de $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$, telles que $\psi_1 = \mathsf{EF}\,\varphi_1$ et $\psi_2 = \mathsf{EF}\,\varphi_2$. Soient r et s deux préfixes de t, alors si $t, r \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi_1)$ et $t, s \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi_2)$, alors $t, r \cup s \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi_1) \wedge \mathsf{Glob}_{\mathsf{Conf}}(A, B, \varphi, \psi_2)$.*

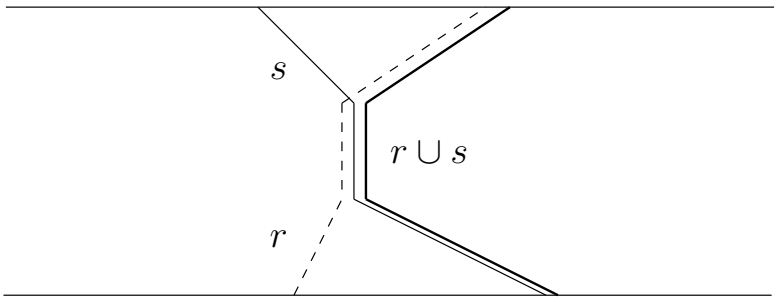


FIG. 7.1 – Union de deux traces

Preuve Nous montrons tout d'abord que $r \cup s$ vérifie les conditions liées à B et φ. Pour ce faire, il suffit de remarquer que $\max(r \cup s) \subseteq \max(r) \cup \max(s)$. Dans ces conditions, pour tout x dans $\max(r \cup s)$, x est soit dans $\max(r)$, soit dans $\max(s)$. Dans ces conditions, $\ell(x) \notin B$ et $t, x \models \varphi$.

Ensuite, dans une trace de Mazurkiewicz, nous avons la propriété suivante : si x est un élément de $\max(r)$ et y un élément de $\max(s)$, alors $x \leq y$ entraîne

$y \in \max(r \cup s)$. En effet, supposons que $y \notin \max(r \cup s)$, alors il existe $z > y$ tel que $z \in \max(r \cup s) \setminus s$, mais alors $z \in r$. De plus on a $z > x$ ce qui est impossible car x est dans $\max(r)$.

Maintenant, comme r et s vérifient les conditions liées à A de la modalité $\mathsf{Glob}_{\mathsf{Conf}}$, on en déduit que pour tout $a \in A$, il existe $x \in \max(s)$ tel que $\ell(x) = a$ et $y \in \max(r)$ tel que $\ell(y) = a$. Comme x et y sont étiquetés par la même lettre, ils sont ordonnés, donc $x \leq y$ ou $y \leq x$, et dans ces conditions x ou y est dans $\max(r \cup s)$ et on en déduit que $r \cup s$ vérifie bien les conditions liées à A.

Il nous reste à montrer que $r \cup s$ vérifie les conditions liées à ψ_1 et à ψ_2. La preuve étant symétrique, nous montrons que $r \cup s$ vérifie la condition liée à ψ_1. Comme r vérifie la condition liée à ψ_1, il existe un sommet x dans $\max(r)$ tel que $t, x \not\models \psi_1$. De plus il existe un sommet y dans $\max(r \cup s$ tel que $y \geq x$, donc $y \not\models \psi_1$ car ψ_1 est une formule du type $\mathsf{EF}\,\varphi_1$. Dans ces conditions, on en déduit que $r \cup s$ vérifie les conditions de la modalité $\mathsf{Glob}_{\mathsf{Conf}}$. □

Nous allons maintenant pouvoir montrer la proposition.

Preuve de la proposition 7 Pour montrer le résultat, on va procéder par induction sur le nombre de sous "eventuality formula" de β.

- ▷ Si β ne contient pas de sous "eventuality formula" (Φ et Ψ sont vides), alors β s'écrit $\exists\mathsf{F}((\bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt))$.
  Soit t un trace de $\mathbb{R}(\Sigma, \mathrm{D})$ et r un préfixe de t.
  Montrons que $t, r \models \beta$ entraîne que pour tout x dans $\max(\bullet r)$, $\bullet t, x \models \mathsf{EF}\,\mathsf{Conf}(A, B, tt, ff)$. Comme $t, r \models \beta$, il existe un préfixe r' de t avec $r' \geq r$ tel que $t, r' \models \mathrm{Now}(\beta)$. Donc $\bullet t, \bullet r' \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, tt, ff)$ et donc pour tout y dans $\max(\bullet r')$, $\bullet t, y \models \mathsf{Conf}(A, B, tt, ff)$. Maintenant pour tout x dans $\max(\bullet r)$, il existe $y \in \max(\bullet r')$ tel que $x \leq y$, donc pour tout x dans $\max(\bullet r)$, on a $\bullet t, x \models \mathsf{EF}\,\mathsf{Conf}(A, B, tt, ff) = \text{Etiquetage}(\beta)$.
  Maintenant supposons que pour tout x dans $\max(\bullet r)$, $\bullet t, x \models \mathsf{EF}\,\mathsf{Conf}(A, B, tt, ff)$. On considère alors r_x le préfixe de t tel qu'il existe $y \geq x$ avec $y \in \max(\bullet r_x)$ et $\bullet t, \bullet r_x \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, tt, ff)$. Dans ces conditions, on appelle s le préfixe de t égale à $\cup_{x\in\max(r)} r_x$. D'après le lemme 8, $t, s \models \mathsf{Glob}_{\mathsf{Conf}}(A, B, tt, ff)$, de plus par construction $r \leq s$, donc $t, r \models \exists\mathsf{F}\,\mathsf{Glob}_{\mathsf{Conf}}(A, B, tt, ff) = \beta$.
  Ceci termine le cas de base de l'induction.
- ▷ On suppose maintenant que $\beta = \exists\mathsf{F}(\bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt) \wedge (\bigwedge_{\varphi\in\Phi}\varphi) \wedge (\bigwedge_{\psi\in\Psi}\neg\psi)$.
  Soit t un trace de $\mathbb{R}(\Sigma, \mathrm{D})$ et r un préfixe de t.
  Montrons que $t, r \models \beta$ entraîne que pour tout x dans $\max(\bullet r)$, $\bullet t, x \models \mathsf{EF}\bigwedge_{\psi\in\Psi}\mathsf{Conf}(A, B, \bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi), \text{Etiquetage}(\psi))$. Comme $t, r \models \beta$, il existe un préfixe r' de t avec $r' \geq r$ tel que $t, r' \models \mathrm{Now}(\beta)$. Considérons $\varphi \in \Phi$, $t, r' \models \varphi$, donc par induction, on en déduit que pour tout y dans $\max(\bullet r')$, $\bullet t, y \models \text{Etiquetage}(\varphi)$. De même, pour tout $\psi \in \Psi$, $t, r' \not\models \psi$, donc il existe y dans $\max(\bullet r')$, tel que $\bullet t, y \not\models \text{Etiquetage}(\psi)$. Donc pour tout ψ dans Ψ, pour tout y dans $\max(\bullet r')$, on a $\bullet t, y \models \mathsf{Conf}(A, B, \bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi), \text{Etiquetage}(\psi))$. Maintenant pour tout x dans $\max(\bullet r)$, il existe $y \in \max(\bullet r')$ tel que $x \leq y$, donc pour tout x dans $\max(\bullet r)$, on a $\bullet t, x \models \mathsf{EF}\,\mathsf{Conf}(A, B, \bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi), \text{Etiquetage}(\psi))$. Ce résultat étant vrai pour tout ψ dans Ψ, on en déduit que $\bullet t, x \models$

$\bigwedge_{\psi\in\Psi} \mathsf{EF}\,\mathsf{Conf}(A,B,\bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi),\text{Etiquetage}(\psi))$. Maintenant, en utilisant le lemme 8, on en déduit que :
$t,x \models \mathsf{EF}\bigwedge_{\psi\in\Psi}\mathsf{Conf}(A,B,\bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi),\text{Etiquetage}(\psi)) = \text{Etiquetage}(\beta)$.
Maintenant supposons que pour tout x dans $\max(\bullet r)$, $\bullet t,x \models \mathsf{EF}\bigwedge_{\psi\in\Psi}\mathsf{Conf}(A,B,\bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi),\text{Etiquetage}(\psi))$. On considère alors ψ dans Ψ et $r_{\psi,x}$ le préfixe de t tel qu'il existe $y \geq x$ avec $y \in \max(\bullet r_{\psi,x})$ et $\bullet t, \bullet r_{\psi,x} \models \mathsf{Glob}_{\mathsf{Conf}}(A,B,\bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi),\text{Etiquetage}(\psi))$. Dans ces conditions, on appelle s_ψ le préfixe de t égale à $\cup_{x\in\max(r)} r_{\psi,x}$. D'après le lemme 8, on a :
$t, s_\psi \models \mathsf{Glob}_{\mathsf{Conf}}(A,B,\bigwedge_{\varphi\in\Phi}\text{Etiquetage}(\varphi),\text{Etiquetage}(\psi))$
Par induction, on en déduit que $t, s_\psi \models \bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt) \wedge (\bigwedge_{\varphi\in\Phi}\varphi) \wedge \neg\psi$. Maintenant on appelle $s = \cup\psi \in \Psi s_\psi$. D'après le lemme 4, on déduit que $r, s \models \bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt) \wedge (\bigwedge_{\varphi\in\Phi}\varphi)$. De plus pour tout $\psi \in \Psi$, $s \geq s_\psi$, ψ est une "eventuality formula", et $s_\psi \models \neg\psi$, donc $s \models \neg\psi$, donc finalement $s \models (\bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt) \wedge (\bigwedge_{\varphi\in\Phi}\varphi) \wedge (\bigwedge_{\psi\in\Psi}\neg\psi)$. Or $s \geq r$, donc :
$r \models \exists\mathsf{F}((\bigwedge_{a\in A}\langle a^{-1}\rangle tt) \wedge (\bigwedge_{b\in B}\neg\langle b^{-1}\rangle tt) \wedge (\bigwedge_{\varphi\in\Phi}\varphi) \wedge (\bigwedge_{\psi\in\Psi}\neg\psi)) = \beta$.
Ceci termine la preuve. □

Nous avons donc une nouvelle procédure de décision de la satisfaisabilité d'une formule de ISTL$^\diamond$:

Comme dans [2], nous nous contentons de travailler sur les "eventuality formula". Nous considérons donc α une "eventuality formula" de ISTL$^\diamond$.

Nous commençons par écrire α comme une disjonction de "eventuality formula" en forme normale à l'aide de la proposition 3, puis pour chacune de ces "eventuality formula" β nous construisons la formule de la proposition Etiquetage(β). Nous faisons alors la disjonction de ces formules, et α est satisfaisable si et seulement si cette nouvelle formule est satisfaisable.

Malheureusement cette formule peut être beaucoup trop longue pour nous donner un résultat équivalent à celui de [2]. Pour trouver un résultat équivalent, nous allons devoir passer par le même processus que dans [2].

Nous reprenons donc l'alphabet de dépendance $(\overline{\Sigma},\overline{\mathrm{D}})$ défini à la section 7.1.

La première chose à remarquer est que la complexité de $\mathrm{TL}^{\mathrm{loc}}_{(\overline{\Sigma},\overline{\mathrm{D}})}(\mathsf{EX},\mathsf{EF},\mathsf{Conf})$ est la même que $\mathrm{TL}^{\mathrm{loc}}_{(\Sigma,\mathrm{D})}(\mathsf{EX},\mathsf{EF},\mathsf{Conf})$ car la dépendance de la complexité de ces logiques en fonction de l'alphabet ne dépend que des composantes connexes de l'alphabet de dépendance et ces composantes connexes sont les mêmes pour (Σ,D) et pour $(\overline{\Sigma},\overline{\mathrm{D}})$.

Ensuite, pour toute formule β de Ξ_α, on introduit la formule atomique μ_β dont la sémantique est $t,x \models \mu_\beta$ si et seulement si $\beta \in \mu(x)$.

Une fois ceci introduit, nous devons modifier la formule Etiquetage(β) de la manière suivante :

$$\text{Etiquetage}'(\beta) = \mathsf{EF}\left(\bigwedge_{\psi\in\Psi}\mathsf{Conf}\left(A,B,\bigwedge_{\varphi\in\Phi}\mu_\varphi,\mu_\psi\right)\right)$$

Nous reprenons alors notre formule α comme étant une disjonction de "eventuality formula" en forme normale. Nous avons donc $\alpha = \beta_1 \vee \beta_2 \vee \cdots \vee \beta_n$. Nous écrivons alors la formule suivante :

$$\alpha' = \big(\bigwedge_{\xi \in \Xi_\alpha} (\mu_\xi \Leftrightarrow \text{Etiquetage}'(\xi)) \big) \wedge \big(\bigvee_{1 \leq i \leq n} \mu_{\beta_i} \big)$$

Nous obtenons alors en utilisant un raisonnement similaire à celui de [2] que la formule α de ISTL$^\diamond$ est satisfaisable si et seulement si la formule α' de $\mathrm{TL}^{\mathrm{loc}}_{(\overline{\Sigma},\overline{\mathrm{D}})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ est satisfaisable.

Il nous reste à calculer la taille de la formule α'.

D'après la proposition 3, la nouvelle forme d'α contient au plus $2^{|\alpha|+1}$ sous "eventuality formula" en forme normale et est la disjonction d'au plus $2^{|\alpha|}$ "eventuality formula" en forme normale, donc la taille de la formule de $\mathrm{TL}^{\mathrm{loc}}_{(\overline{\Sigma},\overline{\mathrm{D}})}(\mathsf{EX}, \mathsf{EF}, \mathsf{Conf})$ est borné par $2^{|\alpha|} + 2^{|\alpha|+1}$ multiplié par la taille d'un opérateur Conf, soit $2 * 2^{|\alpha|+1} + 2 * |\Sigma|$, donc en tout nous avons une formule de taille $2^{O(|\alpha|)}$ en considérant la taille de Σ comme une constante. Puis nous devons décider si cette formule est satisfaisable. Ce problème est un problème PSPACE, donc nous obtenons donc bien que le problème de satisfaisabilité de ISTL$^\diamond$ est en temps $O(2^{2^{|\alpha|}})$.

Chapitre 8

La logique des filtres

8.1 Présentation de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle)$

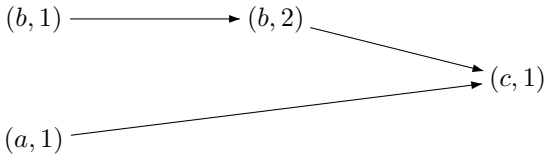
La syntaxe de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle)$ est la suivante :

$$\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle) ::= tt \mid \neg\alpha \mid \alpha \wedge \beta \mid \forall A \subseteq \Sigma; \langle A^* \rangle \alpha \mid \forall A \subseteq \Sigma; \langle A \rangle \alpha$$

La sémantique des opérateurs non encore définis est la suivante :
Soit t une trace sur l'alphabet de dépendance (Σ, D), soit r un préfixe fini de t, nous avons :

$$\begin{cases} t, r \vDash \langle A^* \rangle \alpha & \text{si} \quad \exists s > r \text{ tel que } t, s \vDash \alpha \text{ et } \mathrm{Alph}(r^{-1} \cdot s) \subseteq A \\ t, r \vDash \langle A \rangle \alpha & \text{si} \quad \exists s \text{ tel que } r \lessdot s \text{ et } t, s \vDash \alpha \text{ et } \ell(r^{-1} \cdot s) \in A \end{cases}$$

Nous appelons t la trace suivante :



L'alphabet de dépendance étant :

$$a \quad \text{——} \quad c \quad \text{——} \quad b$$

En reprenant la trace donnée en exemple dans la section précédente, nous avons :

$$t, \varepsilon \vDash \langle b^* \rangle \langle a \rangle \langle c \rangle tt$$
$$t, \varepsilon \vDash \langle \{a, b\}^* \rangle \langle c \rangle tt$$

Cette logique est une logique à base de filtres. En effet l'opérateur $\langle A^* \rangle$ permet de filtrer certains types d'actions. Ces filtres ont été introduits dans

[15] dans le but d'obtenir une logique temporelle globale linéaire sans opérateur passé qui a le même pouvoir d'expression que la logique du premier ordre sur les traces finies ou infinies.

La complexité du problème de satisfaisabilité de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$ n'est cependant pas connue. Nous gardons espoir que cette complexité reste élémentaire, principalement pour deux raisons : tout d'abord cette logique tout comme $\mathrm{ISTL}^{\diamond}$ a des propriétés de type "flat until". Or si nous exceptons la négation (problème qu'il est envisageable de résoudre à l'aide d'automates alternants), le caractère non élémentaire de la complexité de la décidabilité de LTrL est dû à des emboîtements de $\forall\mathsf{U}$ sur la gauche.

De plus, il existe un lien avec $\mathrm{ISTL}^{\diamond}$ au niveau des résiduels des langages de ces logiques.

Si φ est une formule d'une logique globale pur futur sur l'alphabet de dépendance (Σ, D), on écrira $\mathcal{L}(\varphi)$, l'ensemble des traces t de $\mathbb{R}(\Sigma, \mathrm{D})$ telles que $t, \varepsilon \models \varphi$. De plus on écrira $a^{-1}\varphi$ la formule telle que $t, \varepsilon \models a^{-1}\varphi$ si et seulement si $at, \varepsilon \models \varphi$. On remarque que $\mathcal{L}(a^{-1}\varphi) = a^{-1}\mathcal{L}(\varphi)$.

Considérons une formule $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$ qui s'écrit $\varphi = \langle\Sigma^*\rangle\psi$, cette formule est équivalente à $\exists\mathsf{F}\psi$. Avec les notations précédemment introduites nous avons alors l'égalité : $\mathcal{L}(a^{-1}\varphi) = \mathcal{L}(\varphi \vee \langle \mathrm{I}(a)^*\rangle(a^{-1}\psi))$. Nous voyons alors naturellement apparaître les filtres. Comme le calcul d'un automate représentant l'ensemble des modèles d'une formule fait également apparaître naturellement les résiduels d'un langage, la logique $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$ nous avait semblé être une bonne candidate pour une logique globale dont l'expressivité n'est pas trop faible et qui garde pourtant un problème de décidabilité de complexité raisonnable.

Malheureusement nous n'avons pas été capables de trouver un algorithme élémentaire pour ce problème, nous allons montrer maintenant un algorithme non élémentaire construisant un automate alternant reconnaissant l'ensemble des modèles d'une formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$.

8.2 Processus de décision de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$

Pour décider la satisfaisabilité d'une formule α de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$, nous allons calculer un automate alternant qui va accepter toutes les linéarisations des traces r sur (Σ, D), qui vérifient $r, \varepsilon \vDash \alpha$. La satisfaisabilité de α sera donc obtenue en vérifiant qu'un tel automate reconnaît autre chose que le vide. Problème dont la complexité est dans NL. Pour ce faire, nous commençons par enrichir la syntaxe de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$ et déterminer une forme normale de manière à ne pas avoir à traiter la négation.

Nous commençons donc par déterminer notre nouvelle syntaxe :

$$\begin{aligned}\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A]) ::= tt \quad & | \mathit{ff} \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \forall A \subseteq \Sigma; \langle A^*\rangle\alpha \\ & | \forall A \subseteq \Sigma; [A^*]\alpha \mid \forall A \subseteq \Sigma; \langle A\rangle\alpha \\ & | \forall A \subseteq \Sigma; [A]\alpha\end{aligned}$$

La sémantique des opérateurs non encore définis est la suivante :

Soit t une trace sur l'alphabet de dépendance (Σ, D), soit r un préfixe fini de t, nous avons :

$$\begin{cases} t, r \vDash [A^*]\alpha & \Leftrightarrow & t, r \vDash \neg\langle A^*\rangle\neg\alpha \\ t, r \vDash [A]\alpha & \Leftrightarrow & t, r \vDash \neg\langle A\rangle\neg\alpha \end{cases}$$

Ceci entraîne les égalités suivantes :

$$\begin{array}{rcl} \langle\emptyset^*\rangle\alpha & = & \alpha \\ [\emptyset^*]\alpha & = & \alpha \\ \langle\emptyset\rangle\alpha & = & ff \\ [\emptyset]\alpha & = & tt \end{array}$$

Nous pouvons également remarquer que les opérateurs $\langle A\rangle$ et $\vee$ commutent, de même que $\langle A^*\rangle$ et $\vee$, $[A]$ et $\wedge$ et $[A^*]$ et $\wedge$. Ainsi pour tout α, β formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A])$, pour tout $A \subseteq \Sigma$:

$$\begin{array}{rcl} \langle A\rangle(\alpha \vee \beta) & = & \langle A\rangle\alpha \vee \langle A\rangle\beta \\ \langle A^*\rangle(\alpha \vee \beta) & = & \langle A^*\rangle\alpha \vee \langle A^*\rangle\beta \\ [A](\alpha \wedge \beta) & = & [A]\alpha \wedge [A]\beta \\ [A^*](\alpha \wedge \beta) & = & [A^*]\alpha \wedge [A^*]\beta \end{array}$$

Tout formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle)$ peut se réécrire dans notre nouvelle syntaxe, il suffit de faire descendre les négations au niveau des atomes en utilisant les opérateurs conjugués.

Cette logique est de plus une logique pur futur, à savoir soit φ une formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A])$, t une trace et r un préfixe fini de t, nous avons :

$$t, r \vDash \varphi \Leftrightarrow r^{-1}t, \varepsilon \vDash \varphi$$

Profitant de cette propriété, nous nous permettrons l'abus de notation suivante :

$$t \vDash \alpha \text{ si } t, \varepsilon \vDash \alpha$$

Cette propriété permet de plus de donner un sens canonique aux résiduels d'une formule. En effet nous considérerons :

$$t \vDash a^{-1}\alpha \Leftrightarrow a \cdot t \vDash \alpha$$

Dans ces conditions, nous obtenons les calculs de résiduels suivants :

$$\begin{array}{rcl} a^{-1}tt & = & tt \\ a^{-1}ff & = & ff \\ a^{-1}(\alpha \vee \beta) & = & a^{-1}\alpha \vee a^{-1}\beta \\ a^{-1}(\alpha \wedge \beta) & = & a^{-1}\alpha \wedge a^{-1}\beta \\ a^{-1}\langle A^*\rangle\alpha & = & \begin{cases} \langle (A \cap I(a))^*\rangle a^{-1}\alpha \vee \langle A^*\rangle\alpha & \text{si } a \in A \\ \langle (A \cap I(a))^*\rangle a^{-1}\alpha & \text{sinon} \end{cases} \\ a^{-1}[A^*]\alpha & = & \begin{cases} [(A \cap I(a))^*]a^{-1}\alpha \wedge [A^*]\alpha & \text{si } a \in A \\ [(A \cap I(a))^*]a^{-1}\alpha & \text{sinon} \end{cases} \\ a^{-1}\langle A\rangle\alpha & = & \begin{cases} \langle A \cap I(a)\rangle a^{-1}\alpha \vee \alpha & \text{si } a \in A \\ \langle A \cap I(a)\rangle a^{-1}\alpha & \text{sinon} \end{cases} \\ a^{-1}[A]\alpha & = & \begin{cases} [A \cap I(a)]a^{-1}\alpha \wedge \alpha & \text{si } a \in A \\ [A \cap I(a)]a^{-1}\alpha & \text{sinon} \end{cases} \end{array}$$

Nous allons nous inspirer de ce calcul pour trouver un automate alternant $\mathcal{A}_\alpha$ qui étant donné la formule α de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A])$ reconnaît l'ensemble des mots ω tels que :

$$\exists t \in \mathbb{R}(\Sigma, \mathrm{D}) \text{ tel que } t \vDash \alpha \text{ et } \omega \text{ est une linéarisation de } t$$

Nous appellerons $\mathcal{A}_\alpha$ l'automate des linéarisations de α. De plus, si Q_α est l'ensemble des états de l'automate $\mathcal{A}_\alpha$, pour $q \in Q_\alpha$, nous écrirons $\mathcal{L}(\mathcal{A}_\alpha, q)$ pour le langage des mots défini par l'automate $\mathcal{A}_\alpha$ avec pour état initial l'état q.

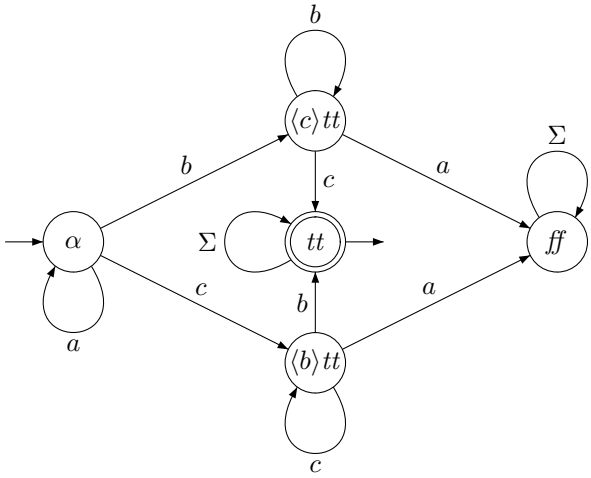
8.2.1 Automates des linéarisations

Exemple

Considérons la formule $\alpha = \langle a^*\rangle(\langle b\rangle tt \wedge \langle c\rangle tt)$ sur l'alphabet de dépendance suivant :

$$b \quad\text{——}\quad a \quad\text{——}\quad c$$

Nous pouvons alors considérer l'automate des linéarisations suivant :



Cet automate reconnaît les linéarisations des traces finies et infinies vérifiant la formule α.

L'exemple est un automate déterministe pour des raisons de clarté, mais nous allons construire des automates alternants.

Description de l'automate

Nous allons construire l'automate par induction structurelle sur la formule α.

Soit α une formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A])$, nous appelons Q_α l'ensemble des états de $\mathcal{A}_\alpha$ et $\delta : (B^+(Q_\alpha), \Sigma), \to B^+(Q_\alpha)$ sa fonction de transition.

Nous définissons alors Q_α de manière inductive :

$$\begin{cases} Q_{tt} & = & \{tt\} \\ Q_{ff} & = & \{ff\} \\ Q_{\alpha\wedge\beta} & = & Q_\alpha \cup Q_\beta \\ Q_{\alpha\vee\beta} & = & Q_\alpha \cup Q_\beta \\ Q_{\langle A^*\rangle\alpha} & = & \{\langle C^*\rangle q \mid C \subseteq A \text{ et } q \in B^+(Q_\alpha)\} \cup Q_\alpha \\ Q_{[A^*]\alpha} & = & \{[C^*]q \mid C \subseteq A \text{ et } q \in B^+(Q_\alpha)\} \cup Q_\alpha \\ Q_{\langle A\rangle\alpha} & = & \{\langle C\rangle q \mid C \subseteq A \text{ et } q \in B^+(Q_\alpha)\} \cup Q_\alpha \\ Q_{[A]\alpha} & = & \{[C]q \mid C \subseteq A \text{ et } q \in B^+(Q_\alpha)\} \cup Q_\alpha \end{cases}$$

Puis nous définissons également δ de manière inductive :

$$\begin{cases} \delta(tt,a) & = & tt \\ \delta(ff,a) & = & ff \\ \delta(\beta\wedge\gamma,a) & = & \delta(\beta,a)\wedge\delta(\gamma,a) \\ \delta(\beta\vee\gamma,a) & = & \delta(\beta,a)\vee\delta(\gamma,a) \\ \delta(\langle A^*\rangle\beta,a) & = & \begin{cases} \langle (A\cap I(a))^*\rangle\delta(\beta,a)\vee\langle A^*\rangle\beta\vee\delta(\beta,a) & \text{si } a\in A \\ \langle (A\cap I(a))^*\rangle\delta(\beta,a)\vee\delta(\beta,a) & \text{sinon} \end{cases} \\ \delta([A^*]\beta,a) & = & \begin{cases} [(A\cap I(a))^*]\delta(\beta,a)\wedge[A^*]\beta\wedge\delta(\beta,a) & \text{si } a\in A \\ [(A\cap I(a))^*]\delta(\beta,a)\wedge\delta(\beta,a) & \text{sinon} \end{cases} \\ \delta(\langle A\rangle\beta,a) & = & \begin{cases} \langle (A\cap I(a))\rangle\delta(\beta,a)\vee\beta & \text{si } a\in A \\ \langle (A\cap I(a))\rangle\delta(\beta,a) & \text{sinon} \end{cases} \\ \delta([A]\beta,a) & = & \begin{cases} [(A\cap I(a))]\delta(\beta,a)\wedge\beta & \text{si } a\in A \\ [(A\cap I(a))]\delta(\beta,a) & \text{sinon} \end{cases} \end{cases}$$

Nous prenons comme états répétés l'état tt et les états du type $[A^*]\beta$ et $[A]\beta$.

Nous prenons comme états finaux les états β tels que $\varepsilon \vDash \beta$. Pour ce faire il suffit de montrer que nous savons décider si une formule est satisfaisable par la trace ε, la preuve se faisant par induction structurelle sur la forme de la formule :

1. tt est satisfaisable par la trace ε.
2. ff n'est pas satisfaisable par la trace ε.
3. $\beta\wedge\gamma$ est satisfaisable par la trace ε si β et γ sont satisfaisables par la trace ε.
4. $\beta\vee\gamma$ est satisfaisable par la trace ε si β ou γ est satisfaisable par la trace ε.
5. $\langle A^*\rangle\beta$ est satisfaisable par la trace ε si β l'est.
6. $[A^*]\beta$ est satisfaisable par la trace ε si β l'est.
7. $\langle A\rangle\beta$ n'est pas satisfaisable par la trace ε.
8. $[A]\beta$ est satisfaisable par la trace ε.

Nous avons alors le théorème suivant :

Théorème 9 *Soit ω un mot de Σ^∞, soit q un état de $B^+(Q_\alpha)$, alors :*

$$[\omega] \vDash q \Leftrightarrow \omega \in \mathcal{L}(\mathcal{A}_\alpha, q)$$

Nous commençons par remarquer que les fonctions de transitions d'une part pour $\langle A^*\rangle\beta$ et $[A^*](\neg\beta)$ et d'autre part pour $\langle A\rangle\beta$ et $[A](\neg\beta)$ sont duales. Or ces deux opérateurs étant des opérateurs duaux, nous en déduisons que nous n'avons

pas besoin de montrer le théorème pour des états du type $[A^*]\beta$ ou $[A]\beta$. En effet, l'automate construit est un alternant très faible, donc son complémentaire est son automate automate dual.

Pour montrer ce théorème nous allons commencer par montrer la proposition suivante :

Proposition 10 *Soit $\langle A^*\rangle q$ et $\langle A\rangle q' \in Q_\alpha$, soit $\omega \in \Sigma^*$, alors :*

$$\delta(\langle A^*\rangle q, \omega) = \bigvee_{\substack{\omega \sim u \cdot v \\ \mathrm{Alph}(u) \subseteq A}} (\langle (A \cap I(v))^*\rangle \delta(q, v) \vee \delta(q, v))$$

et

$$\delta(\langle A\rangle q', \omega) = \langle A \cap I(w)\rangle \delta(q', w) \vee \bigvee_{\substack{\omega \sim a \cdot v \\ a \in A}} \delta(q', v)$$

avec la notation : $u \sim v$ si $[u] = [v]$.

Corollaire 11 *Nous pouvons déduire de cette proposition que quel que soit $q \in Q_\alpha$, quel ques soient $\omega_1, \omega_2 \in \Sigma^*$, $[\omega_1] = [\omega_2] \Leftrightarrow \delta(q, \omega_1) = \delta(q, \omega_2)$. Cette propriété se déduit immédiatement de la syntaxe même de δ.*

Preuve de 10 Nous allons montrer cette proposition par récurrence sur la taille de ω.

La proposition est vraie pour $|\omega| \leq 1$ par définition de δ.

Supposons que la proposition est vraie pour ω et montrons qu'elle est vraie pour $\omega \cdot a$:

Nous n'allons le montrer que pour $\langle A^*\rangle q$ le second résultat se montrant de la même manière.

$$\begin{aligned}
\delta(\langle A^*\rangle q, \omega \cdot a) &= \bigvee_{\substack{\omega \sim u \cdot v \\ \mathrm{Alph}(u) \subseteq A}} (\delta(\langle (A \cap I(v))^*\rangle \delta(q, v), a) \vee \delta(\delta(q, v), a)) \\
&= \bigvee_{\substack{\omega \sim u \cdot v \\ \mathrm{Alph}(u) \subseteq A}} (\langle (A \cap I(v \cdot a))^*\rangle \delta(q, v \cdot a) \vee \delta(q, v \cdot a) \vee \\
&\qquad \bigvee_{a \in A \cap I(v)} \langle (A \cap I(v))^*\rangle \delta(q, v))
\end{aligned}$$

Nous voulons comparer ce résultat à :

$$\bigvee_{\substack{\omega \cdot a \sim u' \cdot v' \\ \mathrm{Alph}(u') \subseteq A}} (\langle (A \cap I(v'))^*\rangle \delta(q, v') \vee \delta(q, v'))$$

Pour ce faire, nous allons montrer que chacun des éléments des disjonctions est contenu dans les deux formulations.

1. Soit u', v' tels que $\omega \cdot a \sim u' \cdot v'$ et $\mathrm{Alph}(u') \subseteq A$:
   D'après le lemme de Levi (voir section 2.3) nous avons :
   (a) S'il existe v tel que $v' \sim v \cdot a$ alors $\langle (A \cap I(v'))^*\rangle \delta(q, v') = \langle (A \cap I(v \cdot a))^*\rangle \delta(q, v \cdot a)$ et $\delta(q, v') = \delta(q, v \cdot a)$ qui sont des éléments de la première formulation puisque $\omega \sim u' \cdot v$.

(b) sinon, il existe u_1, u_2 tels que $u' = u_1 \cdot a \cdot u_2$ avec $\mathrm{Alph}(u_2) \subseteq (I(a))$ et $\omega \sim u_1 \cdot u_2 \cdot v'$. Nous appelons alors $u = u_1 \cdot u_2$ et $v = v'$. Nous avons alors $\delta(p, v) = \delta(p, v')$ et $\langle (A \cap I(v'))^* \rangle \delta(q, v') = \langle (A \cap I(v)^* \rangle \delta(q, v)$.

Nous en déduisons donc que la seconde expression est contenue dans la première.

2. La réciproque se montre de la même manière.

□

Maintenant nous allons pouvoir montrer notre théorème :

Preuve de 9 Soit q un état de A_α. Nous supposons le résultat vrai pour tous les sous-états de q (on appelle sous-états de q, l'ensemble $Q_q \backslash q$).

1. $q = tt$
   Le résultat est évident car l'automate pour q accepte tous les mots et toute trace t satisfait tt.
2. $q = ff$
   Le résultat est évident car l'automate pour q n'accepte aucun mot et aucune trace t ne satisfait ff.
3. $q = q_1 \vee q_2$
   Le résultat est évident par les propriétés même de l'automate alternant.
4. $q = q_1 \wedge q_2$
   Le résultat est évident par les propriétés même de l'automate alternant.
5. $q = \langle A^* \rangle p$
   (a) $[\omega] \vDash q \Rightarrow \omega \in \mathcal{L}(\mathcal{A}_\alpha, q)$
   Nous avons alors :

$$\omega \sim u \cdot v \text{ avec } u \in A^* \text{ et } [v] \vDash p$$

   Nous pouvons alors trouver ω_1 et ω_2 tels que :

$$\omega = \omega_1 \cdot \omega_2 \text{ avec } u \lesssim \omega_1 \text{ et } \mathrm{Alph}(\omega_2) \subseteq \mathrm{Alph}(v)$$

   où $\omega_1 \lesssim \omega_2$ si $[\omega_1] \leq [\omega_2]$.
   Dans ces conditions, nous avons $v = v_1 \cdot v_2$ avec $u \cdot v_1 \sim \omega_1$ et $v_2 \sim \omega_2$
   Dans ces conditions, en utilisant la proposition 10, nous obtenons qu'il existe un chemin dans l'automate de l'état $\langle A^* \rangle p$ à l'état $\delta(p, v_1)$ en lisant le mot $\omega_1 \sim u \cdot v_1$.
   Or par hypothèse, $v_1 \cdot v_2 \in \mathcal{L}(\mathcal{A}_\alpha, p)$, donc $v_2 \in \mathcal{L}(\mathcal{A}_\alpha, \delta(p, v_1))$.
   Donc il existe une exécution acceptante de l'automate à partir de $\delta(p, v_1)$ en lisant le mot v_2. En recollant cette exécution à celle qui part de $\langle A^* \rangle p$ jusqu'à $\delta(p, v_1)$, nous obtenons une exécution acceptante de l'automate à partir de $\langle A^* \rangle p$ en lisant le mot ω.
   (b) $\omega \in \mathcal{L}(\mathcal{A}_\alpha, q) \Rightarrow [\omega] \vDash q$
   Nous considérons une exécution acceptante à partir de $\langle A^* \rangle p$ en lisant ω, alors il existe ω_1 et ω_2 tels que $\omega = \omega_1 \cdot \omega_2$ et que l'exécution vienne d'entrer dans un état de type $\delta(p, v)$ après la lecture de ω_1 ce

qui arrive forcément car les états de type $\langle C^* \rangle p$ ne sont pas des états répétés (Proposition 10).

Dans ces conditions, $[v \cdot \omega_2] \vDash p$ par hypothèse de récurrence, et de plus il existe u tel que $\omega \sim u \cdot v \cdot \omega_2$ avec $u \in A^*$ ce qui montre que $[\omega] \vDash \langle A^* \rangle p = q$.

6. $q = \langle A \rangle p$

   La preuve est identique à celle pour $\langle A^* \rangle p$.

□

Ceci prouve donc que l'automate ainsi construit est bien l'automate des linéarisations qui nous intéressait. Le problème de satisfaisabilité de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle, [A^*], [A])$ donc de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle)$ se réduit alors au problème du vide de cet automate. Cependant, cet automate possède un nombre d'états non élémentaire en fonction de la taille de la formule α, donc notre algorithme de décision est également de complexité non élémentaire.

8.2.2 Implémentation

L'implémentation qui a été faite de cet algorithme procède de façon a l'optimiser quelque peu. Le but est d'éviter les calculs inutiles, et ce de deux manières différentes :

▷ Éviter de calculer plusieurs fois les même résiduels.

▷ Éviter de calculer des états qui ne seront pas accessibles.

Pour le premier élément, nous commençons par réécrire α en utilisant la forme normale suivante :

1. *tt* et *ff* sont en forme normale.
2. Si α est en forme normale, $[A^*]\alpha$ et $[A]\alpha$ sont en forme normale si et seulement si α n'est pas une conjonction.
3. Si α est en forme normale, $\langle A^* \rangle \alpha$ et $\langle A \rangle \alpha$ sont en forme normale si et seulement si α n'est pas une disjonction.
4. Si α est en forme normale et n'est ni une conjonction, ni une disjonction, toutes les combinaisons booléennes de α en forme normale disjonctive sont en forme normale.

N'importe quelle formule de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle, [A^*], [A])$ peut se mettre sous cette forme grâce aux propriétés de commutativité et de distributivité des opérateurs présentées au début du chapitre.

Cette forme normale nous donne un moyen rapide de comparer deux formules.

Nous utilisons également un certain nombre d'optimisations pour simplifier les formules de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^* \rangle, \langle A \rangle, [A^*], [A])$. Par exemple :

$$\langle a \rangle \varphi \vee \langle \{a, b\} \rangle \varphi = \langle \{a, b\} \rangle \varphi$$

$$\langle a \rangle \varphi \wedge \langle \{a, b\} \rangle \varphi = \langle a \rangle \varphi$$

$$[a] \varphi \vee [\{a, b\}] \varphi = [a] \varphi$$

$$[a] \varphi \wedge [\{a, b\}] \varphi = [\{a, b\}] \varphi$$

De plus nous introduisons une fonction de hachage des formules de $\mathrm{TL}^{\mathrm{glob}}_{(\Sigma,\mathrm{D})}(\langle A^*\rangle, \langle A\rangle, [A^*], [A])$ et nous gardons en permanence l'ensemble des formules dont nous avons déjà calculé les résiduels, associées à leurs hachés. Ainsi avant de calculer un nouveau résiduel, nous commençons par calculer le haché de la formule et nous vérifions si nous n'avons pas déjà fait ce calcul.

Enfin pour la seconde proposition, nous calculons les nouveaux états au fur et à mesure qu'ils apparaissent dans les résiduels plutôt que calculer tous les états possibles.

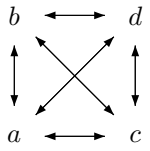
8.3 Exemples

Exemple pour les traces infinis :

Nous allons considérer la formule $[\{a,c\}^*]\langle\{b,d\}^*\rangle\langle a\rangle tt$ pour différents alphabets de dépendance. Ces automates ont été obtenus à l'aide de notre implémentation. Nous ne représentons l'automate que dans un seul cas, les autres étant peu esthétiques du fait de leur complexité.

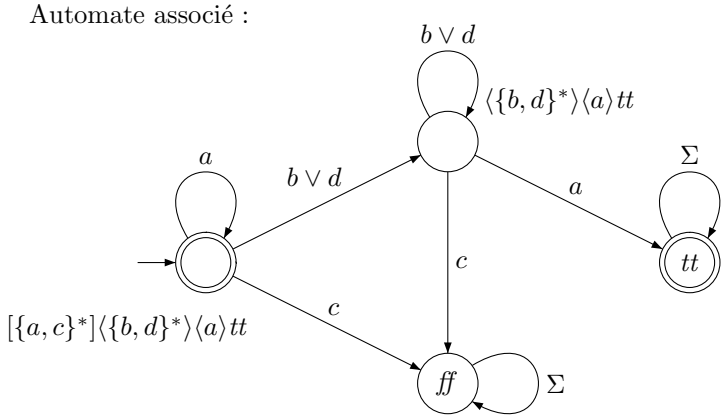
8.3.1 Cas séquentiel

Alphabet de dépendance :



Il s'agit du cas où le monoïde de trace se réduit au cas du monoïde libre.

Automate associé :



Exemple de calcul :

$\delta(\alpha, a) = \alpha \wedge \delta(\langle\{b,d\}^*\rangle\langle a\rangle tt, a)$

or

$\delta(\langle\{b,d\}^*\rangle\langle a\rangle tt, a) = \langle\emptyset^*\rangle\delta(\langle a\rangle tt, a) \vee \delta(\langle a\rangle tt, a)$

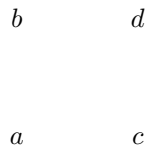
et

$\delta(\langle a\rangle tt, a) = \langle\emptyset\rangle tt \vee tt = tt$

donc
$\delta(\langle\{b,d\}^*\rangle\langle a\rangle tt, a) = tt$
et
$\delta(\alpha, a) = \alpha$

8.3.2 Cas parallèle

Alphabet de dépendance :



Nous obtenons l'automate suivant :
États :

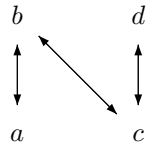
$$\begin{array}{lcl} 1 & = & [\{a,c\}^*]\langle\{b,d\}^*\rangle\langle a\rangle tt \\ 2 & = & [a^*]\langle\{b,d\}^*\rangle\langle a\rangle tt \\ 3 & = & \langle\{b,d\}^*\rangle\langle a\rangle tt \\ 4 & = & \langle b^*\rangle\langle a\rangle tt \\ 5 & = & \langle d^*\rangle\langle a\rangle tt \\ 6 & = & \langle a\rangle tt \\ 7 & = & tt \end{array}$$

L'état initial est l'état 1, les états répétés sont les états 1, 2 et 7.
Fonction de transition :

$$\begin{array}{lcllcl} \delta(1,a) & = & 1 & \delta(1,b) & = & (1\wedge 3)\vee(1\wedge 5)\vee(1\wedge 6) \\ \delta(1,c) & = & 1\wedge 2 & \delta(1,d) & = & (1\wedge 3)\vee(1\wedge 4)\vee(1\wedge 6) \\ \\ \delta(2,a) & = & 2 & \delta(2,b) & = & (2\wedge 3)\vee(2\wedge 5)\vee(2\wedge 6) \\ \delta(2,c) & = & (2\wedge 3)\vee(2\wedge 6) & \delta(2,d) & = & (2\wedge 3)\vee(2\wedge 4)\vee(2\wedge 6) \\ \\ \delta(3,a) & = & 7 & \delta(3,b) & = & 3\vee 5\vee 6 \\ \delta(3,c) & = & 3\vee 6 & \delta(3,d) & = & 3\vee 4\vee 6 \\ \\ \delta(4,a) & = & 7 & \delta(4,b) & = & 4\vee 6 \\ \delta(4,c) & = & 4\vee 6 & \delta(4,d) & = & 4\vee 6 \\ \\ \delta(5,a) & = & 7 & \delta(5,b) & = & 5\vee 6 \\ \delta(5,c) & = & 5\vee 6 & \delta(5,d) & = & 5\vee 6 \\ \\ \delta(6,a) & = & 7 & \delta(6,b) & = & 6 \\ \delta(6,c) & = & 6 & \delta(6,d) & = & 6 \\ \\ \delta(7,a) & = & 7 & \delta(7,b) & = & 7 \\ \delta(7,c) & = & 7 & \delta(7,d) & = & 7 \end{array}$$

8.3.3 Cas linéaire

Alphabet de dépendance :



Nous obtenons l'automate suivant :
États :

$$\begin{array}{rcl} 1 & = & [\{a,c\}^*]\langle\{b,d\}^*\rangle\langle a\rangle tt \\ 2 & = & [a^*]\langle\{b,d\}^*\rangle\langle a\rangle tt \\ 3 & = & \langle\{b,d\}^*\rangle\langle a\rangle tt \\ 4 & = & [a^*]\langle a\rangle tt \\ 5 & = & \langle b^*\rangle\langle a\rangle tt \\ 6 & = & \langle a\rangle tt \\ 7 & = & tt \\ 8 & = & ff \end{array}$$

L'état initial est l'état 1, les états répétés sont les états 1, 2, 4 et 7.
Fonction de transition :

$$\begin{array}{rclrcl} \delta(1,a) & = & 1 & \delta(1,b) & = & 3 \\ \delta(1,c) & = & 1 \wedge 4 \wedge 6 & \delta(1,d) & = & (2 \wedge 3) \vee (2 \wedge 5) \vee (2 \wedge 6) \\ & & & & & \\ \delta(2,a) & = & 2 & \delta(2,b) & = & 3 \\ \delta(2,c) & = & 4 \wedge 6 & \delta(2,d) & = & (2 \wedge 3) \vee (2 \wedge 5) \vee (2 \wedge 6) \\ & & & & & \\ \delta(3,a) & = & 7 & \delta(3,b) & = & 3 \\ \delta(3,c) & = & 6 & \delta(3,d) & = & 3 \vee 5 \vee 6 \\ & & & & & \\ \delta(4,a) & = & 4 & \delta(4,b) & = & 8 \\ \delta(4,c) & = & 4 \wedge 6 & \delta(4,d) & = & 4 \wedge 6 \\ & & & & & \\ \delta(5,a) & = & 7 & \delta(5,b) & = & 5 \\ \delta(5,c) & = & 6 & \delta(5,d) & = & 5 \vee 6 \\ & & & & & \\ \delta(6,a) & = & 7 & \delta(6,b) & = & 8 \\ \delta(6,c) & = & 6 & \delta(6,d) & = & 6 \\ & & & & & \\ \delta(7,a) & = & 7 & \delta(7,b) & = & 7 \\ \delta(7,c) & = & 7 & \delta(7,d) & = & 7 \\ & & & & & \\ \delta(8,a) & = & 8 & \delta(8,b) & = & 8 \\ \delta(8,c) & = & 8 & \delta(8,d) & = & 8 \end{array}$$

Les automates ainsi obtenus sont souvent très loin d'être optimaux. Le seul cas qui donne des automates simples est celui où le monoïde de trace se réduit au monoïde libre, c'est à dire quand le cas des traces se réduit au cas des mots.

Troisième partie

Automates Asynchrones Alternants Cellulaires

Chapitre 9

Motivations

Parmi les différents outils permettant de modéliser les systèmes séquentiels, les automates alternants sont des outils permettant de calculer des langages rationnels en utilisant moins d'états que les automates non déterministes classiques [7, 9]. Ils ont ensuite été utilisés pour faire de la vérification [55, 23].

Parmi les propriétés intéressantes des automates alternants : il est particulièrement simple de les complémenter. Alors que cette opération est de complexité exponentielle en espace sur les automates non déterministes usuels, elle est linéaire sur les automates alternants. Ceci en fait un outil particulièrement adéquat pour calculer l'ensemble des modèles d'une formule, car si un l'ensemble des modèles d'une formule est représenté par un automate alternant, l'ensemble des modèles de la négation de cette même formule est alors facile à calculer.

D'un autre côté, dans le cadre des systèmes distribués, Zielonka [61, 62] a introduit les automates asynchrones (cellulaires) (voir section 2.4).

La complémentation des automates asynchrones n'est pas une opération aisée. Or c'est une opération qui intervient régulièrement dès que l'on veut utiliser des automates pour décrire les modèles d'une formule logique qui contient des négations. Dans ces conditions, il nous a semblé intéressant d'étudier une extension des automates asynchrones cellulaires qui ferait intervenir la notion d'alternance.

Nous avons donc introduit les automates alternants asynchrones cellulaires, et nous avons défini la notion d'exécution et d'exécution acceptante sur ces automates. Nous avons ensuite montré que, malheureusement, les résultats espérés sur le complémentaire ne s'étendent pas aisément au cas asynchrone. Nous nous sommes alors intéressé à la caractérisation de l'ensemble des langages définissables par un automate alternant asynchrone cellulaire.

Nous allons ci-après introduire la notion d'automate alternant asynchrone cellulaire. Puis nous allons montrer que sur un certain nombre d'alphabets de dépendance, ces automates ne reconnaissent pas plus de langages que les automates asynchrones usuels.

Chapitre 10

Définitions

Définition 8 (Automates Alternants Asynchrones Cellulaires).
Un automate alternant asynchrone cellulaire *(AAAC)* $\mathcal{A}$ *sur l'architecture cellulaire* (Σ, D) *est un quadruplet* $((Q_a)_{a\in\Sigma}, q^0, (\delta_a)_{a\in\Sigma}, \mathcal{F})$ *où :*

- ▷ *Chaque* Q_a *est un ensemble fini* d'états locaux *pour le processus* $a \in \Sigma$. *Les éléments de* Q_Σ *sont les états globaux de* $\mathcal{A}$.
- ▷ $q^0 \in Q_\Sigma$ *est* l'état initial global *de* $\mathcal{A}$.
- ▷ $\delta_a : Q_{\mathrm{D}(a)} \to \mathcal{B}^+(Q_a)$ *est la* fonction de transition *locale pour* a *dans* Σ.
- ▷ $\mathcal{F} \subseteq Q_\Sigma$ *est la* condition d'acceptation.

Une exécution de l'automate alternant asynchrone cellulaire $\mathcal{A}$ sur une trace finie t est une structure d'événements $\rho = (V, \leq, \#, \ell, \sigma)$ où $\ell : V \to \Sigma$ est un étiquetage des événements par les lettres alors que $\sigma : V \to \bigcup_{a\in\Sigma} Q_a$ est étiquetage des événements par les états, et qui vérifie les conditions (10.1-10.4) :

$$\forall e, f \in V, \qquad e \#^i f \Leftrightarrow e \neq f \text{ et } \ell(e) = \ell(f) \text{ et } \Downarrow e = \Downarrow f \tag{10.1}$$

$$\forall C \in \mathcal{C}_{\max}(\rho), \quad (C, \leq, \ell) \sim t \tag{10.2}$$

$$\forall e \in V, \qquad \sigma(e) \in Q_{\ell(e)}, \tag{10.3}$$

Pour e dans V, on se permet d'écrire $\mathrm{D}(e)$ pour $\mathrm{D}(\ell(e))$. Pour e dans V, nous définissons $H(e) = \{f \in V \mid \ell(f) = \ell(e) \text{ et } \Downarrow f = \Downarrow e\}$ comme l'ensemble des *cousins* de e.

$$\forall e \in V, \qquad \sigma(H(e)) \models \delta_{\ell(e)}((q^0 * \mathrm{lw}(\Downarrow e))_{\mathrm{D}(e)}) \tag{10.4}$$

L'exécution ρ est acceptée par l'automate $\mathcal{A}$ si $q^0 * \mathrm{lw}(C) \in \mathcal{F}$ pour tout $C \in \mathcal{C}_{\max}(\rho)$. On note $\mathcal{L}(\mathcal{A})$ l'ensemble des traces de $\mathbb{M}(\Sigma, \mathrm{D})$ sur lesquelles il existe une exécution acceptée par $\mathcal{A}$.

Nous pouvons par exemple considérer l'automate alternant asynchrone cellulaire suivant : $\Sigma = \{a, b\}$ avec $a\mathrm{I}b$. $Q_a = Q_b = \{0, 1\}$, $q_0 = (0, 0)$, $\delta_a(0) = 0 \wedge 1$, $\delta_b(0) = 0 \vee 1$ et $\mathcal{F} = \{(0, 0), (1, 1)\}$. Il existe alors deux exécution de $\mathcal{A}$ sur la trace ab qui sont :

Appelons ρ la première de ces deux exécutions. Les deux éléments t_1 et t_2 de $\mathcal{C}_{\max}(\rho)$ sont les suivants :

$$\begin{array}{cc} b_0 & b_0 \\ a_0 & a_1 \end{array}$$

Dans ces conditions $q_0 * \mathrm{lw}(t_1) = (0,0)$ et $q_0 * \mathrm{lw}(t_2) = (1,0)$. On en déduit donc que cette exécution n'est pas une exécution acceptante de $\mathcal{A}$. Il en est de même pour la seconde exécution. Nous en déduisons que la trace ab n'est pas reconnue par l'automate $\mathcal{A}$.

Cet automate nous intéresse également car il montre que, malheureusement, les automates alternants asynchrones cellulaires ne se complémentent pas aussi aisément que les automates alternants sur les mots. En effet, pour obtenir le complémentaire d'un automate alternant sur les mots, il suffit de complémenter la relation d'acceptation et de prendre le dual de la relation de transition. Si nous faisons ceci pour l'automate $\mathcal{A}$ présenté ci-dessus, nous obtenons l'automate suivant : $Q_a = Q_b = \{0,1\}$, $q_0 = (0,0)$, $\delta_a(0) = 0 \vee 1$, $\delta_b(0) = 0 \wedge 1$ et $\mathcal{F} = \{(0,1),(1,0)\}$. Cet automate est très similaire à l'automate de départ et on vérifie facilement que la trace ab n'est pas acceptée par cet automate.

Pour $U \subseteq V$, on définit $\mathsf{last}(U) = \{e \in U \mid \forall f \in U,\ (\ell(e) = \ell(f) \wedge e \leq f) \Rightarrow e = f\}$. On écrit alors $\mathsf{last}(\rho)$ pour la sous structure d'événements induite par $\mathsf{last}(V)$. On étend cette définitions aux ensembles de structures d'événements de la manière suivante : $\mathsf{last}(S) = \bigcup_{U \in S} \mathsf{last}(U)$. Ces définitions s'appliquent également aux ordres partiels étiquetés. On utilisera fréquemment les observations suivantes, valides dès que 10.2 est vraie.

Lemme 12 *Soit t une trace de $\mathbb{M}(\Sigma, \mathrm{D})$, soit $\rho = (V, \leq, \#, \ell)$ une t-SE, alors pour tout $C \in \mathcal{C}_{\max}(\rho)$, on a $\mathsf{last}(C) = \mathsf{last}(V) \cap C$.*

Preuve L'inclusion $\mathsf{last}(V) \cap C \subseteq \mathsf{last}(C)$ est toujours vraie. Il suffit donc de prouver $\mathsf{last}(C) \subseteq \mathsf{last}(V) \cap C$. Soit e dans $\mathsf{last}(C)$, alors e est dans C. Soit $f \in V$ tel que $\ell(e) = \ell(f) = a$ et $e \leq f$. Supposons $e < f$. Tous les éléments étiquetés par a dans C doivent être ordonnés avec e, et comme $e \in \mathsf{last}(C)$, ils appartiennent tous à $\downarrow e$. Donc, $\downarrow f$ contient strictement plus d'éléments étiquetés par a que C, et donc une configuration maximale contenant $\downarrow f$ ne peut être isomorphe à C, ce qui est en contradiction avec 10.2. On en déduit donc que $e = f$, donc e est dans $\mathsf{last}(V)$. □

Corollaire 13 *Soit t une trace de $\mathbb{M}(\Sigma, \mathrm{D})$, soit $\rho = (V, \leq, \#, \ell)$ une t-SE. Dans ces conditions $\mathsf{last}(\mathcal{C}_{\max}(\rho)) = \mathcal{C}_{\max}(\mathsf{last}(\rho))$. En particulier, $\mathsf{last}(\rho)$ est une $\mathsf{last}(t)$-SE.*

Preuve Soit $C \in \mathcal{C}_{\max}(\rho)$. D'après le lemme 12, nous savons que $\mathsf{last}(C) = \mathsf{last}(V) \cap C$ est une configuration de $\mathsf{last}(\rho)$. Nous allons prouver que cette configuration est maximale. Soit $e \in \mathsf{last}(V)$ tel que $\mathsf{last}(C) \cup \{e\}$ est sans conflit. Alors, $C \cup \{e\} = \downarrow \mathsf{last}(C) \cup \{e\}$ est sans conflit, et comme C est maximal, on en déduit que $e \in C$. Donc, $e \in \mathsf{last}(V) \cap C = \mathsf{last}(C)$ et donc $\mathsf{last}(C)$ est maximal.

Soit $C \in \mathcal{C}_{\max}(\mathsf{last}(\rho))$, alors $\downarrow C$ est une configuration et nous trouvons $C' \in \mathcal{C}_{\max}(\rho)$ tel que $\downarrow C \subseteq C'$. Maintenant nous avons $C \subseteq C' \cap \mathsf{last}(V) = \mathsf{last}(C')$ d'après le lemme 12. Comme C est maximal, on en déduit que $C = \mathsf{last}(C')$. $\square$

Chapitre 11

Automate réduit

Dans ce chapitre, nous allons construire à partir d'un automate alternant asynchrone cellulaire $\mathcal{A}$, un automate séquentiel usuel $\mathcal{R}_\mathcal{A}$ qui acceptera toutes les linéarisations des traces acceptées par $\mathcal{A}$. Les états de $\mathcal{R}_\mathcal{A}$ sont des structures d'événements particulières. Afin de pouvoir définir ces états, nous introduisons les notions de $\mathcal{T}$-SE et de structures d'événements réduites.

Nous identifions un ordre partiel étiqueté $(V, \leq, \ell)$ où $\ell : V \to \Sigma$ est une bijection avec l'ordre partiel sur Σ défini par $a < b$ si et seulement si $\ell^{-1}(a) < \ell^{-1}(b)$. Nous notons $\mathcal{T}$ la classe des ordres partiels sur Σ.

Soit $T = (\Sigma, \leq)$. Une *T-structure d'événements* (T-SE) est une structure d'événements $g = (V, \leq, \#, \ell, \sigma)$ telle que $(x, \leq, \ell) \sim T$ pour tout x dans $\mathcal{C}_{\max}(g)$. Dans ces conditions, nous disons que T est le *type* de g. Nous disons que g est une $\mathcal{T}$-SE, si c'est une T-SE pour un certain ordre partiel T dans $\mathcal{T}$.

Remarque 1 *Notons que si $g = (V, \leq, \#, \ell, \sigma)$ est une T-SE et que x est un sous-ensemble sans conflit de V tel que $(x, \leq, \ell) \sim T$, alors x est dans $\mathcal{C}_{\max}(g)$.*

Pour définir $\mathcal{R}_\mathcal{A}$, nous avons également besoin de la notion de réduction. Soit $g = (V, \leq, \#, \ell, \sigma)$ une $\mathcal{T}$-SE. Un morphisme $\varphi : g \to g$ tel que $\varphi_{|\varphi(V)} = \mathrm{id}_{|\varphi(V)}$ est une *reduction* de g sur $\varphi(g)$. La $\mathcal{T}$-SE g est *réduite* si la seule réduction de g est id_g.

Lemme 14 *Soit g une T-SE et soit φ une réduction de g. Alors, $\varphi(g)$ est une T-SE. De manière plus précise $\mathcal{C}_{\max}(\varphi(g)) = \varphi(\mathcal{C}_{\max}(g)) \subseteq \mathcal{C}_{\max}(g)$.*

Preuve Soit $g = (V, \leq, \#, \ell, \sigma)$. Soit y dans $\mathcal{C}_{\max}(\varphi(g))$. Comme y est sans conflit, il existe x dans $\mathcal{C}_{\max}(g)$ tel que $y \subseteq x$. Alors $y = \varphi(y) \subseteq \varphi(x)$ qui est un sous-ensemble sans conflit de $\varphi(V)$. Comme y est un ensemble maximal, on obtient $y = \varphi(x)$, donc $\mathcal{C}_{\max}(\varphi(g)) \subseteq \varphi(\mathcal{C}_{\max}(g))$.

Considérons maintenant x dans $\mathcal{C}_{\max}(g)$. Comme φ est un morphisme, $\varphi(x)$ est sans conflit. Soit y dans $\mathcal{C}_{\max}(g)$ tel que $\varphi(x) \subseteq y$. Comme g est une T-SE, et φ est un morphisme, on a $|y| = |x| = |\ell(x)| = |\ell(\varphi(x))| = |\varphi(x)|$. On en déduit que $\varphi(x) = y \in \mathcal{C}_{\max}(g)$ et donc on obtient $\varphi(\mathcal{C}_{\max}(g)) \subseteq \mathcal{C}_{\max}(g)$. De plus $\varphi(x) \subseteq \varphi(V)$ est un sous-ensemble sans conflit de $\varphi(g)$ qui est maximal en tant que configuration de g, donc c'est également une configuration maximale de sa sous structure d'événements $\varphi(g)$. On en déduit donc, $\varphi(\mathcal{C}_{\max}(g)) \subseteq \mathcal{C}_{\max}(\varphi(g))$. □

Lemme 15 *Pour toute $\mathcal{T}$-SE g, il existe une réduction $\varphi : g \to g$ telle que $\varphi(g)$ est réduite. De plus, il existe une unique, à isomorphisme près, sous structure d'événements de g, noté $\overline{g}$, qui est à la fois réduite et l'image de g par un morphisme.*

Preuve Nous prouvons ce résultat par induction sur la taille de g. Si g est réduit il suffit de prend $\varphi = \mathrm{id}$. Dans le cas contraire, par définition, il existe une réduction $\varphi : g \to g$ telle que $\varphi(g) \neq g$. D'après le lemme 14, $\varphi(g)$ est une $\mathcal{T}$-SE, de plus elle a strictement moins d'éléments que g. Par induction, il existe une réduction ψ de $\varphi(g)$ telle que $\psi \circ \varphi(g)$ est réduite. Nous pouvons alors conclure car la composition de deux réductions est encore une réduction.

Supposons maintenant qu'il existe deux morphismes φ_1 et φ_2 de g tels que $h_1 = \varphi_1(g)$ et $h_2 = \varphi_2(g)$ soiensoient tous les deux réduits. Posons $h_i = (V_i, \leq, \#, \ell, \sigma)$. Dans ces conditions $\varphi_2 \circ \varphi_1 : h_2 \to h_2$ est un morphisme et nous obtenons $V_2 = \varphi_2 \circ \varphi_1(V_2) \subseteq \varphi_2(V_1) \subseteq V_2$, donc $\varphi_2(V_1) = V_2$. De même $\varphi_1(V_2) = V_1$. De plus, comme $V_2 = \varphi_2 \circ \varphi_1(V_2)$, les deux morphismes φ_1 et φ_2 sont bijectifs et le lemme 2 entraîne donc $h_1 \sim h_2$. □

Nous allons maintenant décrire comment on étend une T-SE g par l'exécution d'une action a de Σ. Nous devons étendre les configurations maximales y de g par des événements étiquetés par a. Cette extension ne dépend que des événements dans $y \cup \ell^{-1}(\mathrm{D}(a))$ qui sont les événements lus par a. Formellement pour une T-SE $g = (V_g, \leq_g, \#_g, \ell_g, \sigma_g)$ on définit :

$$\chi_a(g) = \{y \cap \ell^{-1}(\mathrm{D}(a)) \mid y \in \mathcal{C}_{\max}(g)\} \tag{11.1}$$

On considère alors $\mathcal{H} : \chi_a(g) \to 2^{Q_a} \setminus \{\emptyset\}$. Pour x dans $\chi_a(g)$, on écrira H_x à la place de $\mathcal{H}(x)$. On définit alors $g \cdot_a \mathcal{H} = \mathsf{last}(V, \leq, \#, \ell, \sigma)$ où :

- ▷ $V = V_g \uplus \biguplus_{x \in \chi_a(g)} H_x$,
- ▷ (ℓ, σ) coïncide avec (ℓ_g, σ_g) sur V_g, et pour tout e dans H_x, $\ell(e) = a$ et $\sigma(e) = e$,
- ▷ $e < f$ si et seulement si $e <_g f$, ou $e \in \downarrow_g x$ et f est dans H_x pour un certain x de $\chi_a(g)$,
- ▷ $e \#^i f$ si et seulement si $e \#^i_g f$, ou e et f sont deux éléments distincts de H_x pour un certain x de $\chi_a(g)$.

Ceci définit bien $g \cdot_a \mathcal{H}$ comme une structure d'événements.

Soit $\mathcal{A} = ((Q_a)_{a \in \Sigma}, q^0, (\delta_a)_{a \in \Sigma}, \mathcal{F})$ un automate alternant asynchrone cellulaire. L'automate $\mathcal{R}_\mathcal{A}$ sur Σ^* est défini de la manière suivante. Les états de $\mathcal{R}_\mathcal{A}$ sont les $\mathcal{T}$-SE réduites. L'état initial de $\mathcal{R}_\mathcal{A}$ est $\emptyset$. Un état g est un état acceptant si et seulement si $(q^0 * \mathrm{lw}(\mathcal{C}_{\max}(g))) \subseteq \mathcal{F}$. Finalement les transitions de $\mathcal{R}_\mathcal{A}$ sont de la forme $g \xrightarrow{a} \overline{g \cdot_a \mathcal{H}}$, où $\mathcal{H} : \chi_a(g) \to 2^{Q_a} \setminus \{\emptyset\}$ est tel que :

$$\forall x \in \chi_a(g),\ H_x \models \delta_a((q^0 * \sigma_g(x))_{\mathrm{D}(a)}) \tag{11.2}$$

On remarque que $\mathcal{R}_\mathcal{A}$ est *a priori* un automate infini. Pour justifier cette définition, nous devons prouver que lorsque g est un état de $\mathcal{R}_\mathcal{A}$, alors $\overline{g \cdot_a \mathcal{H}}$ est également un état de $\mathcal{R}_\mathcal{A}$. Du fait du lemme 14, il suffit de montrer que lorsque f est une $\mathcal{T}$-SE, alors $g \cdot_a \mathcal{H}$ est également une $\mathcal{T}$-SE, ce qui est le but du lemme 16.

Étant donnés deux ordres partiels $T_1 = (\Sigma_1, \leq_1)$ et $T_2 = (\Sigma_2, \leq_2)$, nous définissons $T_1 \cdot T_2 = (\Sigma_1 \cup \Sigma_2, \leq)$ par $a < b$ si $a, b \in \Sigma_2$ et $a <_2 b$, ou $a, b \in \Sigma_1 \setminus \Sigma_2$

et $a <_1 b$, ou $a \in \Sigma_1 \setminus \Sigma_2$, $b \in \Sigma_2$ et il existe $a' \in \Sigma_1$, $b' \in \Sigma_2$ telles que $a \leq_1 a'$, $b' \leq_2 b$ et $a' \,\mathrm{D}\, b'$.

Lemme 16 *Soit $g = (V_g, \leq_g, \#_g, \ell_g, \sigma_g)$ une T-SE et $\mathcal{H} : \chi_a(g) \to 2^{Q_a} \setminus \{\emptyset\}$, alors $h = g \cdot_A \mathcal{H} = \mathsf{last}(V, \leq, \#, \ell, \sigma)$ est une $(T \cdot a)$-SE. Plus précisément, pour tout z dans $\mathcal{C}_{\max}(h)$, il existe y dans $\mathcal{C}_{\max}(g)$ et e dans $H_{y \cap \ell^{-1}(\mathrm{D}(a))}$ tels que $z = (y \setminus \ell^{-1}(a)) \cup \{e\}$.*

Preuve Nous réutilisons les notations introduites ci-dessus. Nous commençons par montrer que $\mathsf{last}(V) = V \setminus \ell_g^{-1}(a)$. Soit $e \in \ell_g^{-1}(a)$. Il existe z dans $\mathcal{C}_{\max}(g)$ tel que e soit dans z. De plus e appartient à $x = z \cap \ell_g^{-1}(\mathrm{D}(a)))$. De ce fait, pour tout f dans H_x, nous avons $e < f$ et $\ell_g(e) = a = \ell(f)$. Comme H_x est non vide, nous en déduisons que e n'appartient pas à $\mathsf{last}(V)$. Dans l'autre sens, considérons e dans $V \setminus \ell_g^{-1}(a)$ et f dans V avec $e \leq f$ et $\ell(e) = \ell(f)$. Comme $\ell(e) \neq a$, f est un élément de V_g, et il existe donc $y \in \mathcal{C}_{\max}(g)$ tel que $f \in y$. Alors e et f appartiennent à $\downarrow y = y$ et comme g est une T-SE, nous en déduisons que $e = f$.

Nous allons maintenant montrer que $H_x \times H_y \subseteq \#$ pour tous x, y dans $\chi_a(g)$ tels que $x \neq y$. En effet, comme g est une T-SE, il existe $e \in x$ et $f \in y$, tels que $e \neq f$ et $\ell(e) = \ell(f)$. Comme g est une T-SE, ceci implique $e \# f$. Maintenant, quel que soit e' dans H_x et f' dans H_y, nous avons $e < e'$ et $f < f'$, donc par héritage, nous obtenons $e' \# f'$.

Considérons z dans $\mathcal{C}_{\max}(h)$. Nous montrons que $z \setminus V_g$ est un singleton. Supposons $z \subseteq V_g$. Alors il existe y dans $\mathcal{C}_{\max}(g)$ tel que $z \subseteq y$. Soit $x = y \cap \ell^{-1}(\mathrm{D}(a))$ et f dans H_x, alors $z \cup \{f\}$ est sans conflit, et $z \cup \{f\} \subseteq \mathsf{last}(V)$ ce qui est en contradiction avec la maximalité de z. On en déduit donc que $z \setminus V_g$ est non vide. Comme deux événements quelconques de $V \setminus V_g$ sont en conflit, nous en déduisons donc que $z \setminus V_g = \{e\}$ est un singleton. Remarquons que $z \cap V_g = z \setminus \ell^{-1}(a)$. Donc $z = (z \setminus \ell^{-1}(a)) \cup \{e\}$.

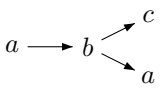
Comme z est sans conflit dans h, sa fermeture par le passé $z' = \downarrow z$ dans $(V, \leq, \#, \ell, \sigma)$ est également sans conflit. Remarquons que $z' = z$ si $\ell_g^{-1}(a) = \emptyset$ et $z' = z \cup \{e'\}$ avec $e' < e$ et $\ell(e') = a$ sinon. Considérons y dans $\mathcal{C}_{\max}(g)$ tel que $z' \cap V_g \subseteq y$. Nous avons $\Downarrow e \cap \ell^{-1}(\mathrm{D}(a)) = z' \cap V_g \cap \ell^{-1}(\mathrm{D}(a)) \subseteq y \cap \ell^{-1}(\mathrm{D}(a))$. De ce fait, nous en déduisons $e \in H_{y \cap \ell^{-1}(\mathrm{D}(a))}$ et $y \cup \{e\}$ est sans conflit. Nous obtenons $z = (z \setminus \ell^{-1}(a)) \cup \{e\} \subseteq (y \setminus \ell^{-1}(a)) \cup \{e\}$ qui est un ensemble sans conflit et contenu dans $\mathsf{last}(V)$. Par maximalité de z, nous obtenons $z = (y \setminus \ell^{-1}(a)) \cup \{e\}$.

Finalement, on remarque aisément que $((y \setminus \ell^{-1}(a)) \cup \{e\}, \leq, \ell)$ est isomorphe à $T \cdot a$ quand y est dans $\mathcal{C}_{\max}(g)$ et e dans $H_{y \cap \ell^{-1}(\mathrm{D}(a))}$. □

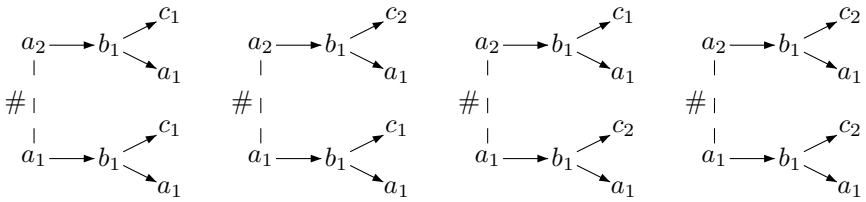
Il faut remarquer que $\mathcal{R}_{\mathcal{A}}$ n'est pas un automate diamant. Considérons en effet l'alphabet $\{a, b, c\}$ avec $\mathrm{D} = a - b - c$ et considérons $\mathcal{A}$ d'état initial $(0, 0, 0)$ et dont les transitions sont :

- $\delta_a(a_0, b_0) = 1 \wedge 2$,
- $\delta_b(a_1, b_0, c_0) = \delta_b(a_2, b_0, c_0) = 1$,
- $\delta_c(b_1, c_0) = 1 \vee 2$,
- $\delta_a(a_1, b_1) = \delta_a(a_2, b_1) = 1$.

On considère alors l'exécution de cet automate sur la trace suivante :

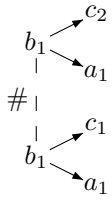


Il existe exactement 4 exécutions de $\mathcal{A}$ sur cette trace qui sont :



Maintenant, si on considère les exécutions de $\mathcal{R}_\mathcal{A}$ sur les linéarisations ce cette trace, on remarque qu'il y a exactement deux exécutions de $\mathcal{R}_\mathcal{A}$ sur le mot *abac*. Ces exécutions atteignent les deux états suivants :

Cependant, si on considère les exécutions de $\mathcal{R}_\mathcal{A}$ sur le mot *abca*, en sus des deux états précédents, il existe deux exécutions qui atteignent l'état suivant :



Néanmoins, nous allons montrer que l'automate $\mathcal{R}_\mathcal{A}$ accepte exactement les linéarisations des traces acceptées par $\mathcal{A}$.

Proposition 17 *Soit $\mathcal{A}$ un automate asynchrone alternant cellulaire. Alors* $\mathrm{Lin}(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{R}_\mathcal{A})$.

Avant de prouver cette proposition, nous définissons la notion de restriction d'une exécution de $\mathcal{A}$ et la notion de a-successeur à l'intérieur d'une exécution de $\mathcal{A}$, et nous prouvons plusieurs lemmes.

Soit $\rho = (V, \leq, \#, \ell, \sigma)$ une exécution d'un automate asynchrone alternant cellulaire $\mathcal{A}$ sur une trace t et soit t_1 un préfixe de t. Nous notons $\rho_{|t_1}$ la sous structure d'événements de ρ induite par $V_1 = \{e \in V \mid (\downarrow e, \leq, \ell) \leq t_1\}$. Nous remarquons que lorsque e est dans V_1, alors $\downarrow e \subseteq V_1$ et $H(e) \subseteq V_1$. On en déduit alors que $\rho_{|t_1}$ est une exécution de $\mathcal{A}$ sur t_1.

Soit $\rho = (V, \leq, \#, \ell, \sigma)$ une exécution de $\mathcal{A}$ sur une trace t. Soit $sa \leq t$ pour un certain s dans $\mathbb{M}(\Sigma, \mathrm{D})$ et a dans Σ. Soit $g = (S, \leq, \#, \ell, \sigma)$ une $\mathsf{last}(s)$-SE avec $S \subseteq \mathsf{last}(\rho_{|s})$. Pour x dans $\chi_a(g)$, soit $K_x = \{e \in V \mid \ell(e) = a \text{ et } \Downarrow e = \downarrow x\}$. Le a-successeur de g dans ρ, noté $\mathrm{next}_\rho(g, a)$, est alors la sous structure d'événements induite par $S' = (S \setminus \ell^{-1}(a)) \cup \bigcup_{x \in \chi_a(g)} K_x$.

De plus, si il existe une trace $u = u'a$, tel que $su \leq t$, on définit par induction $\mathrm{next}_\rho(g, u) = \mathrm{next}_\rho(\mathrm{next}_\rho(g, u'), a)$ avec $\mathrm{next}_\rho(g, \varepsilon) = g$.

Lemme 18 *En reprenant les notations ci-dessus, nous avons* $\mathrm{next}_\rho(g,a) = g \cdot_a \mathcal{H}$ *où* $\mathcal{H}$ *est la fonction qui à* x *dans* $\chi_a(g)$ *associe* $H_x = \sigma(K_x) \neq \emptyset$. *En particulier,* $\mathrm{next}_\rho(g,a)$ *est une* $\mathsf{last}(sa)$*-SE. De plus,* $S' \subseteq \mathsf{last}(\rho_{|sa})$ *et* $\mathrm{next}_\rho(g,a)$ *est la sous structure d'événements de* $\rho_{|sa}$ *induite par* S'.

Preuve On commence par montrer que K_x est non vide.

Soit $x \in \chi_a(g)$ et $y \in \mathcal{C}_{\max}(g)$ tel que $x = y \cap \ell^{-1}(\mathrm{D}(a))$. Comme g est une $\mathsf{last}(s)$-SE, nous avons $(y, \leq, \ell) \sim \mathsf{last}(s)$. Puisque $y \subseteq \mathsf{last}(\rho_{|s})$ qui est aussi une $\mathsf{last}(s)$-SE (corrolaire 13), on obtient $y \in \mathcal{C}_{\max}(\mathsf{last}(\rho_{|s}))$ (remarque 1). Soit $u \in \mathcal{C}_{\max}(\rho_{|s})$ tel que $y = \mathsf{last}(u)$ (corrolaire 13). On a $(u, \leq, \ell) \sim s$ et puisque $sa \leq t$ il existe $e \in V$ tel que $v = u \cup \{e\}$ soit une configuration et $(v, \leq, \ell) \sim sa$. On a alors, $\Downarrow e = \downarrow(\mathsf{last}(u) \cap \ell^{-1}(\mathrm{D}(a))) = \downarrow x$. Donc $e \in K_x$ qui n'est pas vide.

Maintenant, si $e \in K_x$ alors par définition de K_x et de $H(e)$, on obtient $K_x = H(e)$. Comme ρ est une exécution, la condition 10.4 nous assure que $\sigma(H(e)) \models \delta_{\ell(e)}((q^0 * \mathrm{lw}(\Downarrow e))_{\mathrm{D}(e)}) = \delta_{\ell(e)}((q^0 * \mathrm{lw}(\downarrow x))_{\mathrm{D}(e)}) = \delta_a((q^0 * \sigma(x))_{\mathrm{D}(a)})$.

Nous montrons maintenant que $S' = \mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$.

Soit e un élément de S', alors soit e est dans $S \setminus \ell^{-1}(a)$ soit e est dans $\bigcup_{x \in \chi_a(g)} K_x$. Si e est dans $\bigcup_{x \in \chi_a(g)} K_x$, alors e est dans $\mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$ car e est maximal dans $\rho_{|sa}$ et donc il n'existe pas d'élément f dans $S \cup \bigcup_{x \in \chi_a(g)} K_x$ tel que $f > e$. Si e est dans $S \setminus \ell^{-1}(a))$, alors pour tout s dans $\rho_{|s}$, $f > e \Rightarrow \ell(f) \neq \ell(e)$, comme de plus, $\ell(e) \neq a$, on en déduit que e est dans $\mathsf{last}(\rho_{|sa})$, et donc e est dans $\mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$. On en déduit que $S' \subseteq \mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$.

Considérons maintenant e dans $\mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$, alors e peut ne pas être dans S' seulement si e est dans S et $\ell(e) = a$, mais dans ce cas, il existe $x \in \chi_a(g)$ tel que e est dans x. Alors pour tout f dans $K_x \neq \emptyset$, $e < f$ et $\ell(e) = \ell(f) = a$, donc e n'est pas dans $\mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$. Nous en déduisons que e est dans S'. Donc finalement $S' = \mathsf{last}(S \cup \bigcup_{x \in \chi_a(g)} K_x)$.

Nous pouvons maintenant conclure. Par définition de $g \cdot_a \mathcal{H}$, nous avons $\mathrm{next}_\rho(g,a) = g \cdot_a \mathcal{H}$ où $\mathcal{H}$ est la fonction qui à x dans $\chi_a(g)$ associe $H_x = \sigma(K_x) \neq \emptyset$. □

Nous allons maintenant montrer qu'à partir d'une exécution de l'automate réduit, nous pouvons reconstruire une exécution de l'automate alternant asynchrone cellulaire.

Lemme 19 *Soit* τ *une exécution de* $\mathcal{A}$ *sur* t *une trace de* $\mathbb{M}(\Sigma, \mathrm{D})$ *et soit* $g = \overline{\mathsf{last}(\tau)}$. *Soit* $\mathcal{H} : \chi_a(g) \to 2^{Q_a} \setminus \{\emptyset\}$ *qui satisfait l'équation 11.2. Nous avons alors une transition :* $g \xrightarrow{a} \overline{h}$ *dans* $\mathcal{R}_\mathcal{A}$ *où* $h = g \cdot_a \mathcal{H}$. *Il existe alors une exécution* ρ *de* $\mathcal{A}$ *sur* ta *telle que* $\rho_t = \tau$, $h = \mathrm{next}_\rho(g,a)$ *et telle qu'il existe une réduction* ψ *de* $\mathsf{last}(\rho)$ *sur* h.

Ce lemme nous donne un moyen de construire inductivement une exécution de l'automate alternant asynchrone cellulaire $\mathcal{A}$ à partir d'une exécution de l'automate $\mathcal{R}_\mathcal{A}$.

Preuve Soit $\tau = (W, \leq, \#, \ell, \sigma)$ et soit $\varphi : \mathsf{last}(\tau) \to \overline{\mathsf{last}(\tau)} = g$ une réduction. Nous étendons τ en une exécution $\rho = (V, \leq, \#, \ell, \sigma)$ de $\mathcal{A}$ sur ta de la manière suivante :

- ▷ $V = W \uplus \biguplus_{x \in \chi_a(\mathsf{last}(\tau))} K_x$, où pour tout x dans $\chi_a(\mathsf{last}(\tau))$, K_x est une copie de $H_{\varphi(x)}$. On remarque que pour x dans $\chi_a(\mathsf{last}(\tau))$ nous avons $x = y \cap \ell^{-1}(\mathrm{D}(a))$ pour un certain y dans $\mathcal{C}_{\max}(\mathsf{last}(\tau))$. D'après le lemme 14, $\varphi(y) \in \mathcal{C}_{\max}(g)$, donc $\varphi(x) = \varphi(y) \cap \ell^{-1}(\mathrm{D}(a)) \in \chi_a(g)$ ce qui assure que $H_{\varphi(x)}$ est bien défini.
- ▷ Les fonctions d'étiquetage ℓ et σ sont étendues à V par $\ell(e) = a$ pour tout e dans $V \setminus W$ et $\sigma(K_x) = \sigma(H_{\varphi(x)})$ pour x dans $\chi_a(\mathsf{last}(\tau))$.
- ▷ On a $e < f$ si e et f sont dans W et $e < f$ ou e est dans $\Downarrow x$ et f est dans K_x pour un certain x dans $\chi_a(\mathsf{last}(\tau))$.
- ▷ On a $e \#^i f$ si e et f sont dans W et $e \#^i f$ ou $e \neq f$ et e et f sont dans K_x pour un certain x dans $\chi_a(\mathsf{last}(\tau))$.

Nous montrons alors que ρ est une exécution de $\mathcal{A}$ sur ta. Pour cela nous devons montrer que ρ vérifie les conditions 10.1 à 10.4.

Les conditions 10.1 et 10.3 sont vérifiées par définition de ρ.

Nous montrons maintenant que ρ vérifie la condition 10.2.

On considère C dans $\mathcal{C}_{\max}(\rho)$. Il y a au plus un élément dans $C \setminus W$, car quels que soient e et f dans $V \setminus W$, $e \neq f$ entraîne $e \# f$. Nous allons montrer qu'il y a exactement un élément dans $C \setminus W$. Si ce n'est pas le cas, C est inclus dans W, et comme C est maximal, C est dans $\mathcal{C}_{\max}(\tau)$. Soit $x = \mathsf{last}(C) \cap \ell^{-1}(\mathrm{D}(a))$, x est l'unique élément de $\chi_a(\mathsf{last}(\tau))$ contenu dans C. Par définition de $\mathcal{H}$, K_x n'est pas vide, et on peut considérer un élément e de K_x. Comme C est maximal dans ρ, il existe f dans C tel que $e \# f$. Ce conflit ne peut être initial, car f n'est pas dans K_x. Nous en déduisons qu'il existe $e' < e$ tel que $e' \# f$. Maintenant, par définition de ρ, e' est un élément de $\downarrow x$ qui est inclus dans C. Ceci est une contradiction, car par définition C est un ensemble sans conflit. On en déduit qu'il y a exactement un élément de $V \setminus W$ dans C. Nous appelons e cet élément.

Nous montrons maintenant que $(C \setminus \{e\}, \leq, \ell) \sim t$.

Nous pouvons étendre $C \setminus \{e\}$ dans τ en $C' \in \mathcal{C}_{\max}(\tau)$. Si $C' \neq C \setminus \{e\}$, alors il existe f dans $C' \setminus C$ et comme C est maximal dans ρ nous avons $f \# e$. Par le même raisonnement que ci dessus, nous trouvons e' dans $\downarrow x \subseteq C \setminus \{e\} \subseteq C'$ tel que $f \# e'$. Ceci contredit le fait que C' est un ensemble sans conflit. On en déduit que $(C \setminus \{e\}) = C'$ et comme τ est un exécution de $\mathcal{A}$ sur t, on en déduit que $(C', \leq, \ell) = (C \setminus \{e\}, \leq, \ell) \sim t$.

Nous montrons maintenant que $(C, \leq, \ell) \sim ta$.

Il suffit de montrer que $(C, \leq, \ell) \sim (C \setminus \{e\}, \leq, \ell) \cdot a$ en utilisant la définition de la concaténation des traces. C'est bien le cas puisque pour $f \in C \setminus \{e\}$, on a $f < e$ si et seulement si il existe $f' \in x = \mathsf{last}(C \setminus \{e\}) \cap \ell^{-1}(\mathrm{D}(a))$ avec $f \leq f'$, c'est à dire si et seulement si il existe $f' \in C \setminus \{e\}$ avec $f \leq f'$ et $\ell(f') \in D(a)$.

Nous en déduisons finalement que $(C, \leq, \ell) \sim ta$, ce qui prouve que ρ vérifie la condition 10.2.

Il nous reste à montrer que ρ vérifie la condition 10.4.

Soit e dans V. Si e est dans W alors e est dans τ et il vérifie la propriété 10.4.

Supposons maintenant que e est dans $V \setminus W$. Nous avons $\ell(e) = a$ et il existe $x \in \chi_a(\mathsf{last}(\tau))$ tel que e est dans K_x.

Nous montrons que $K_x = H(e)$.

Tout d'abord, pour tout f dans K_x, $\Downarrow f = \Downarrow e = \downarrow x$ et $\ell(e) = \ell(f) = a$, donc $K_x \subseteq H(e)$.

Nous avons $\Downarrow f = \Downarrow e = \downarrow x$ et $\ell(f) = \ell(e) = a$. Considérons maintenant $f \in H(e)$. Comme ρ vérifie la condition 10.2, nous avons $|\downarrow f \cap \ell^{-1}(a)| = |\downarrow e \cap \ell^{-1}(a)| > |\downarrow f' \cap \ell^{-1}(a)|$ pour tout f' dans W, donc f est dans $V \setminus W$. Si f n'est pas dans K_x, alors il existe y dans $\chi_a(\mathsf{last}(\tau))$, tel que f est dans K_y, mais dans ce cas e et f ne sont pas en conflit initial. Il existe donc $e' < e$ tel que $e' \# f$, mais alors on a $e' < f$. Ceci est une contradiction, ce qui prouve que f est dans K_x, donc $H(e) = K_x$.

Maintenant $\mathrm{lw}(\Downarrow e)_{\mathrm{D}(e)} = \mathrm{lw}(\downarrow x)_{\mathrm{D}(e)} = \sigma(x)$ et $\sigma(K_x) = \sigma(H_{\varphi(x)}) \models \delta_a((q^0 * \sigma(\varphi(x)))_{\mathrm{D}(a)}) = \delta_a((q^0 * \sigma(x))_{\mathrm{D}(a)})$ ce qui prouve que :

$$\sigma(H(e)) \models \delta_a((q^0 * \mathrm{lw}(\Downarrow e))_{\mathrm{D}(e)})$$

Nous en déduisons donc que ρ vérifie la condition 10.4, et donc finalement ρ est bien une exécution de $\mathcal{A}$ sur ta. De plus $\rho_{|t} = \tau$ par construction de ρ.

Nous montrons maintenant que $h = \mathrm{next}_\rho(g, a)$.

Considérons $S \subseteq W$ tel que $g = (S, \leq, \#, \ell, \sigma)$. D'après le lemme 18, h est isomorphe à la sous structure d'événements de $\mathsf{last}(\rho)$ induite par $S' = (S \setminus \ell^{-1}(a)) \uplus \biguplus_{x \in \chi_a(g)} K_x$. Nous en déduisons donc que $h = (S', \leq, \#, \ell, \sigma) = \mathrm{next}_\rho(g, a)$.

Enfin, nous montrons qu'il existe une réduction ψ de $\mathsf{last}(\rho)$ sur h.

Tout d'abord $\mathsf{last}(V) = (\mathsf{last}(W) \setminus \ell^{-1}(a)) \uplus \biguplus_{x \in \chi_a(\mathsf{last}(\tau))} K_x$. On définit alors $\psi : \mathsf{last}(\rho) \to h$ par :

$$\psi(e) = \begin{cases} \varphi(e) & \text{si } e \in \mathsf{last}(W) \setminus \ell^{-1}(a) \\ e' \text{ où } e' \text{ est l'unique élément de } K_{\varphi(x)} \text{ tel que } \sigma(e') = \sigma(e) & \text{si } e \in K_x \text{ avec } x \in \chi_a(\mathsf{last}(\tau)) \end{cases}$$

Il nous faut donc montrer que ψ est bien une réduction de $\mathsf{last}(\rho)$ sur h.

Nous commençons par montrer que $\psi(\mathsf{last}(V)) = S'$.

$\psi(\mathsf{last}(V)) = \psi((\mathsf{last}(W) \setminus \ell^{-1}(a)) \uplus \biguplus_{x \in \chi_a(\mathsf{last}(\tau))} K_x) = \varphi((\mathsf{last}(W) \setminus \ell^{-1}(a))) \uplus \bigcup_{x \in \chi_a(\mathsf{last}(\tau))} \psi(K_x) = (S \setminus \ell^{-1}(a)) \uplus \bigcup_{x \in \chi_a(\mathsf{last}(\tau))} \psi(K_x)$. Par définition de ψ, $\psi(K_x) = K_{\varphi(x)}$ donc, nous avons $\bigcup_{x \in \chi_a(\mathsf{last}(\tau))} \psi(K_x) = \bigcup_{x \in \chi_a(\mathsf{last}(\tau))} K_{\varphi}(x)) = \biguplus_{y \in \varphi(\chi_a(\mathsf{last}(\tau)))} K_y = \biguplus_{y \in \chi_a(\varphi(\mathsf{last}(\tau)))} K_y$ et $\varphi(\mathsf{last}(\tau)) = g$, donc $\bigcup_{x \in \chi_a(\mathsf{last}(\tau))} \psi(K_x) = \biguplus_{y \in \chi_a(g)} K_y$ et finalement $\psi(\mathsf{last}(V)) = S'$.

Nous savons maintenant que ψ est à valeur sur h, nous montrons maintenant que ψ est bien une réduction.

Pour ce faire, il suffit de montrer que ψ est un morphisme car par définition de ψ, $\psi_{|h} = \mathrm{id}_{|h}$.

Tout d'abord $\ell(\psi(e)) = \ell(e)$ et $\sigma(\psi(e)) = \sigma(e)$ par construction de ψ.

Nous montrons que pour tout e, f dans $\mathsf{last}(V)$, $e < f \Rightarrow \psi(e) < \psi(f)$.

Si e et f sont dans W, alors $\psi(e) = \varphi(e)$ et $\psi(f) = \varphi(f)$ et le résultat est donc vrai car φ est une réduction. Maintenant si e est dans $V \setminus W$, alors $e < f$ n'est pas possible. Il reste à considérer le cas $e \in W$ et $f \in V \setminus W$. Alors il existe $x \in \chi_a(\mathsf{last}(\tau))$ et $e' \in x$ tel que f est dans K_x et $e \leq e' \lessdot f$. Nous avons alors $\varphi(e) = \psi(e) \leq \psi(e') = \varphi(e')$ et $\psi(f) \in K_{\varphi(x)}$ et $\varphi(e') \in \varphi(x)$, donc $\varphi(e') = \psi(e') < \psi(f)$. Nous en déduisons donc $\psi(e) < \psi(f)$.

Montrons enfin que pour tout e et f dans $\mathsf{last}(V)$, $e \natural f \Rightarrow \psi(e) \natural \psi(f)$.

Si e et f sont dans W, alors $\psi(e) = \varphi(e)$ et $\psi(f) = \varphi(f)$ et le résultat est donc vrai car φ est une réduction. Si e et f sont dans $V \setminus W$, $e \neq f \Rightarrow e \# f$ et le résultat est encore vrai. Il reste à considérer le cas où e est dans W et f dans $V \setminus W$. Il existe alors x dans $\chi_a(\mathsf{last}(\tau))$ tel que f est dans K_x et $x \natural e$ sinon e et f seraient en conflit. Comme φ est une réduction, on a $\psi(e) = \varphi(e)$ et cet élément est compatible avec $\varphi(x)$. Maintenant $\psi(e)$ et $\psi(f)$ ne peuvent pas être en conflit initial par définition de ρ et de ψ. De même aucun $e' < \psi(e)$ ne peut être en conflit initial avec f. Puisque $\Downarrow\psi(f) = \Downarrow\varphi(x)$ et $\varphi(x)$ est compatible avec $\psi(e)$, nous en déduisons que $\psi(e)$ est compatible avec $\psi(f)$.

Finalement, ψ est donc bien une réduction de $\mathsf{last}(\rho)$ sur h. □

Nous allons maintenant pouvoir montrer que l'automate réduit reconnaît bien l'ensemble des linéarisations des traces reconnues par l'automate alternant asynchrone cellulaire.

Preuve de la proposition 17 Montrons $\mathrm{Lin}(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{R}_\mathcal{A})$.

Soit t une trace de $\mathbb{M}(\Sigma, \mathrm{D})$ dans $\mathcal{L}(\mathcal{A})$, soit ρ une exécution acceptante de $\mathcal{A}$ sur t. Soit w une linéarisation de t. Nous allons montrer par induction sur $|w|$ qu'il existe une exécution de $\mathcal{R}_\mathcal{A}$ sur w atteignant un état $g \subseteq \mathsf{last}(\rho)$.

Si $|w| = 0$, alors $\emptyset$ est une exécution de $\mathcal{R}_\mathcal{A}$ et le résultat est vrai.

Si $w = va$, alors par induction, il existe une exécution $\emptyset \xrightarrow{v} g'$ telle que $g' \subseteq \mathsf{last}(\rho_{|[v]})$. D'après le lemme 18, nous avons $\mathrm{next}_\rho(g', a) = g' \cdot_a \mathcal{H}$ et $\mathrm{next}_\rho(g', a) \subseteq \rho$, donc $\overline{\mathrm{next}_\rho(g', a)} \subseteq \rho$ et il existe une transition $g' \xrightarrow{a} g = \overline{g' \cdot_a \mathcal{H}} = \overline{\mathrm{next}_\rho(g', a)} \subseteq \rho$ ce qui termine l'induction.

Considérons maintenant x dans $\mathcal{C}_{\max}(g)$, comme g est inclus dans $\mathsf{last}(\rho)$, x est dans $\mathcal{C}_{\max}(\mathsf{last}(\rho)) = \mathsf{last}(\mathcal{C}_{\max}(\rho))$ (corrolaire 13). Soit $y \in \mathcal{C}_{\max}(\rho)$ tel que $x = \mathsf{last}(y)$. On a $\mathrm{lw}(x) = \mathrm{lw}(y)$, donc $q^0 * \mathrm{lw}(x) = q^0 * \mathrm{lw}(y) \in \mathcal{F}$ car ρ est une exécution acceptante, ce qui prouve que $q^0 * \mathrm{lw}(\mathcal{C}_{\max}(g)) \subseteq \mathcal{F}$ et donc que $\emptyset \xrightarrow{w} g$ est une exécution acceptante de $\mathcal{R}_\mathcal{A}$. Donc $\mathrm{Lin}(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{R}_\mathcal{A})$.

Montrons maintenant $\mathcal{L}(\mathcal{R}_\mathcal{A}) \subseteq \mathrm{Lin}(\mathcal{L}(\mathcal{A}))$

Soit un mot w dans $\mathcal{L}(\mathcal{R}_\mathcal{A})$ et $\emptyset \xrightarrow{w} g$ une exécution acceptante de $\mathcal{R}_\mathcal{A}$. Nous montrons par induction sur $|w|$ qu'il existe une exécution ρ de $\mathcal{A}$ sur $[w]$ telle que $\overline{\mathsf{last}(\rho)} = g$.

Si w est vide, prendre pour ρ l'exécution vide est solution.

Supposons maintenant que $\emptyset \xrightarrow{w} g \xrightarrow{a} \overline{h}$ est une exécution de $\mathcal{R}_\mathcal{A}$ avec a dans Σ, $\mathcal{H} : \chi_a(g) \to 2^{Q_a} \setminus \{\emptyset\}$ satisfaisant l'équation 11.2 et $h = g \cdot_a \mathcal{H}$. Par induction, il existe une exécution $\tau = (W, \leq, \#, \ell, \sigma)$ de $\mathcal{A}$ sur $t = [w]$ telle que $g = \overline{\mathsf{last}(\tau)}$. D'après le lemme 19, nous trouvons une exécution ρ de $\mathcal{A}$ sur $ta = [wa]$ et une réduction $\psi : \mathsf{last}(\rho) \to h$. Nous en déduisons que $\overline{\mathsf{last}(\rho)} = \overline{h}$ ce qui termine l'induction.

Remarquons au passage que ceci prouve le corollaire du lemme 19 suivant :

Corollaire 20 *Soit $\mathcal{A}$ un automate asynchrone alternant cellulaire. Soit $t = t_1 t_2 = (V, \leq, \ell)$ une trace de $\mathbb{M}(\Sigma, \mathrm{D})$ avec $t_i = (V_i, \leq, \ell)$. Soit w_i une linéarisation de t_i et $\emptyset \xrightarrow{w_1} g \xrightarrow{w_2} h$ une exécution de $\mathcal{R}_\mathcal{A}$. Alors il existe un exécution ρ de $\mathcal{A}$ telle que $g = \overline{\mathsf{last}(\rho_{|t_1})}$ et $h = \overline{\mathrm{next}_\rho(g, t_2)} = \overline{\mathsf{last}(\rho)}$.*

Considérons maintenant φ une réduction de $\mathsf{last}(\rho)$ sur $\overline{\mathsf{last}(\rho)}$. Soit $y \in \mathcal{C}_{\max}(\rho)$ et $x = \mathsf{last}(y) \in \mathcal{C}_{\max}(\mathsf{last}(\rho))$. D'après le lemme 14, $\varphi(x) \in \mathcal{C}_{\max}(\overline{\mathsf{last}(\rho)})$. Comme φ est un morphisme, $\sigma(x) = \sigma(\varphi(x))$, donc $q_0 * \mathrm{lw}(y) = q_0 * \mathrm{lw}(x) = q_0 * \mathrm{lw}(\varphi(x))) \in F$. Ceci prouve que $\mathcal{L}(\mathcal{R}_{\mathcal{A}}) \subseteq \mathrm{Lin}(\mathcal{L}(\mathcal{A}))$.

Finalement nous en déduisons que $\mathcal{L}(\mathcal{R}_{\mathcal{A}}) = \mathrm{Lin}(\mathcal{L}(\mathcal{A}))$.

□

Chapitre 12

Reconnaissabilité sur les cographes

Nous allons maintenant montrer que le langage de traces définies par un automate alternant asynchrone cellulaire est un langage régulier quand l'alphabet de dépendance est un alphabet série parallèle (voir section 2.3).

Pour ce faire, nous allons montrer que l'automate des linéarisations d'un tel automate est un automate fini. Dans ce but, nous allons travailler sur un automate alternant asynchrone cellulaire particulier que nous appellerons *automate universel* dont l'automate réduit contiendra tous les automates réduits de tous les automates alternants asynchrones cellulaires sur le même alphabet et avec les mêmes états. Nous allons ensuite procéder par induction structurelle sur l'alphabet et montrer que l'automate réduit de l'automate universel ne contient qu'un nombre fini d'état. Comme tout automate réduit pourra alors être considérer comme une partie de cet automate universel, ce résultat prouvera que tout automate réduit est un automate fini, et donc que les langages reconnus par les automates alternants asynchrones cellulaires sur les alphabets séries parallèles sont réguliers.

12.1 Automate universel

Nous nous plaçons sur l'alphabet de dépendance (Σ, D) et nous nous donnons un ensemble d'états $Q = (Q_a)_{a\in\Sigma}$, nous définissons alors l'automate universel $\mathcal{U}_{\Sigma,\mathrm{D},Q}$ par $\mathcal{U}_{\Sigma,\mathrm{D},Q} = (Q, q^0, (\delta_a)_{a\in\Sigma}, Q_\Sigma)$ avec $\delta_a(q) = tt$ pour tout q dans $Q_{\mathrm{D}(a)}$ et pour tout a dans Σ.

Nous considérons alors un automate cellulaire asynchrone alternant quelconque sur le même alphabet et avec les mêmes états locaux, et nous allons montrer que toute exécution de cette automate est une exécution de l'automate universel, et que toute exécution de l'automate des linéarisations de cet automate est une exécution de l'automate des linéarisations de l'automate universel.

Lemme 21 *Soit $\mathcal{A} = (Q, q^0, (\delta'_a)_{a\in\Sigma}, \mathcal{F})$ un automate cellulaire asynchrone alternant, et t une trace de $\mathbb{M}(\Sigma, \mathrm{D})$. Toute exécution de $\mathcal{A}$ sur t est une exécution de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$ sur t.*

Preuve Soit $\rho = (V, \leq, \#, \ell)$ une exécution de $\mathcal{A}$, pour montrer que c'est également une exécution de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$, il suffit de montrer que ρ vérifie les conditions 10.1 à 10.4. Les conditions 10.1, 10.2 et 10.3 sont vérifiées par ρ car elles ne dépendent pas de la fonction de transition ni de la condition d'acceptation de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$. La condition 10.4 est également vérifiée car quel que soit e dans V, $\delta_{\ell(e)}((q^0 * \mathrm{lw}(\Downarrow e))_{\mathrm{D}(e)}) = tt$. On en déduit donc que ρ est bien une exécution de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$ sur t. □

Lemme 22 *Soit $\mathcal{A} = (Q, q^0, (\delta'_a)_{a\in\Sigma}, \mathcal{F})$, soit $\mathcal{R}_\mathcal{A}$ l'automate des linéarisations de $\mathcal{A}$. Soit $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ l'automate des linéarisations de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$. Quel que soit l'état q de $\mathcal{R}_\mathcal{A}$, q est un état de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$.*

Preuve Pour montrer ce résultat, on va considérer un mot v tel qu'il existe une exécution de $\mathcal{R}_\mathcal{A}$ menant à q en lisant le mot v. On va alors montrer par induction sur la taille de v qu'il existe une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ menant à q en lisant v.

- ▷ Si $v = \varepsilon$, alors q est l'état $\emptyset$ et c'est également un état de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$.
- ▷ Si $v = ua$, alors il y a une exécution de $\mathcal{R}_\mathcal{A}$ de la forme $\emptyset \xrightarrow{u} q' \xrightarrow{a} q$. Par induction, il existe une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ de la forme $\emptyset \xrightarrow{u} q'$, il nous suffit donc de montrer qu'il existe dans $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ une transition de q' à q par la lettre a. Par construction, il existe $\mathcal{H} : \chi_a(q') \to 2^{Q_a} \setminus \{\emptyset\}$ tel que $q = \overline{q' \cdot_a \mathcal{H}}$. Dans ces conditions, $\mathcal{H}$ vérifie également l'équation 11.2 pour l'automate $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$, donc la transition $q' \xrightarrow{a} q$ est également une transition de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ ce qui nous permet de dire que $\emptyset \xrightarrow{u} q' \xrightarrow{a} q$ est une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ ce qui termine l'induction.

Ceci prouve que tous les états de $\mathcal{R}_\mathcal{A}$ sont des états de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. □

12.2 Induction

Pour montrer que le nombre d'état de l'automate universel est fini, nous allons montrer comment construire les états de cet automate à partir des états de l'automate universel sur un alphabet plus petit.

Nous définissons $\mathcal{G}(\Sigma, \mathrm{D})$ comme l'ensemble des structures d'événements g telles qu'il existe un mot w de Σ^* tel qu'il existe un chemin $\emptyset \xrightarrow{w} g$ dans $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. On remarque que $\mathcal{G}(\Sigma, \mathrm{D})$ est exactement l'ensemble des états accessibles de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$.

Nous allons donc montrer la proposition suivante par induction :

Proposition 23 *Soit (Σ, D) un alphabet série parallèle. L'ensemble $\mathcal{G}(\Sigma, \mathrm{D})$ est un ensemble fini.*

12.2.1 Base de l'induction

Si l'alphabet Σ ne contient qu'une seule lettre, $\mathcal{G}(\Sigma, \mathrm{D}) = 2^Q$. Dans ces conditions, il s'agit bien d'un ensemble fini et la proposition est prouvée.

12.2.2 Cas paralléle

L'alphabet Σ est tel que $\Sigma = A \uplus B$ avec $A \times B \cap \mathrm{D} = \emptyset$. Dans ces conditions, considérons une exécution ρ de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$ sur une trace $t \in \mathbb{R}(\Sigma, \mathrm{D})$. Comme (Σ, D) est un alphabet série parallèle, nous pouvons décomposer t en deux traces t_A et t_B sur les alphabets $\mathbb{R}(A, \mathrm{D}_A)$ et $\mathbb{R}(B, \mathrm{D}_B)$ telles que $t = t_A t_B = t_B t_A$.

Nous pouvons alors décomposer $\rho = (V, \leq, \#, \ell, \sigma)$ en deux exécution $\rho_A = (V_A, \leq_A, \#_A, \ell_A, \sigma_A)$ et $\rho_B = (V_B, \leq_B, \#_B, \ell_B, \sigma_B)$ telles que :

- $V_A = \ell^{-1}(A)$
- $V_B = \ell^{-1}(B)$
- $\leq_A = \leq_{|V_A}$
- $\leq_B = \leq_{|V_B}$
- $\#_A = \#_{|V_A}$
- $\#_B = \#_{|V_B}$
- $\ell_A = \ell_{|V_A}$
- $\ell_B = \ell_{|V_B}$
- $\sigma_A = \sigma_{|V_A}$
- $\sigma_B = \sigma_{|V_B}$

Nous avons alors les propriétés suivantes :

- $\leq = \leq_A \uplus \leq_B$
- $\# = \#_A \uplus \#_B$
- ρ_A est une exécution de $\mathcal{U}_{A,\mathrm{D},Q}$ sur t_A.
- ρ_B est une exécution de $\mathcal{U}_{B,\mathrm{D},Q}$ sur t_B.

Ces propriétés montrent que l'on peut donc considérer ρ comme la mise en parallèle des deux exécutions ρ_A et ρ_B.

Dans ces conditions, $\mathsf{last}(\rho) = \mathsf{last}(\rho_A) \uplus \mathsf{last}(\rho_B)$ et si on considère un morphisme φ_A de $\mathsf{last}(\rho_A)$ sur g_A et un morphisme de $\mathsf{last}(\rho_B)$ sur g_B, alors $\varphi = \varphi_A \uplus \varphi_B$ est un morphisme de $\mathsf{last}(\rho)$ sur $g = g_A \uplus g_B$. De même, si on considère un morphisme de $\mathsf{last}(\rho)$ sur g, on peut le décomposer en deux morphismes φ_A et φ_B tels que $\varphi_A(\mathsf{last}(\rho_A)) = g_A$, $\varphi_B(\mathsf{last}(\rho_B)) = g_B$ et $g = g_A \uplus g_B$.

Dans ces conditions, on en déduit qu'une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ sur t se décompose en deux exécutions : une de $\mathcal{R}_{\mathcal{U}_{A,\mathrm{D},Q}}$ sur t_A et une de $\mathcal{R}_{\mathcal{U}_{B,\mathrm{D},Q}}$ sur t_B et qu'un état de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ est l'union d'un état de $\mathcal{R}_{\mathcal{U}_{A,\mathrm{D},Q}}$ et d'un état de $\mathcal{R}_{\mathcal{U}_{B,\mathrm{D},Q}}$.

Par induction, le nombre d'états de $\mathcal{R}_{\mathcal{U}_{A,\mathrm{D},Q}}$ et $\mathcal{R}_{\mathcal{U}_{B,\mathrm{D},Q}}$ sont finis, donc le nombre d'état de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ qui est le produit des nombres d'états de $\mathcal{R}_{\mathcal{U}_{A,\mathrm{D},Q}}$ et de $\mathcal{R}_{\mathcal{U}_{B,\mathrm{D},Q}}$ est également fini, ce qui prouve la proposition dans ce cas.

12.2.3 Cas série

Nous nous plaçons maintenant sur l'alphabet $\Sigma = A \uplus B$ avec $A \times B \subseteq \mathrm{D}$.

Pour décrire la construction qui nous permettra de construire les états de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ à partir des états de $\mathcal{R}_{\mathcal{U}_{A,\mathrm{D},Q}}$ et $\mathcal{R}_{\mathcal{U}_{B,\mathrm{D},Q}}$, nous allons avoir besoin de définir la notion de *projection* et celle de *produit série*.

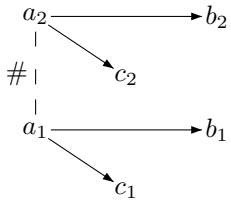
Définition 9 (Projection).
Soit $T = (\Sigma, \leq)$ un type et $g = (V, \leq, \#, \ell, \sigma)$ une T-SE. Soit $\Sigma_1 \subseteq \Sigma$ un sous ensemble de Σ. La projection $\Pi_{\Sigma_1}(g)$ est la structure d'événements $(\ell^{-1}(\Sigma_1), \leq, \#, \ell, \sigma)$.

La projection s'étend au type de g, en effet nous avons le lemme suivant :

Lemme 24 *Soit $T = (\Sigma, \leq)$ un type et $g = (V, \leq, \#, \ell, \sigma)$ une T-SE. Soit $\Sigma_1 \subseteq \Sigma$ un sous ensemble de Σ. $\Pi_{\Sigma_1}(g)$ est une $\Pi_{\Sigma_1}(T)$-SE avec $\Pi_{\Sigma_1}(T) = (\Sigma_1, \leq)$.*

Preuve Pour prouver ce résultat, il suffit de montrer que toutes les configurations maximales $\Pi_{\Sigma_1}(g)$ sont isomorphes au type $\Pi_{\Sigma_1}(T)$. Considérons donc un configuration x de $\mathcal{C}_{\max}(\Pi_{\Sigma_1}(g))$, nous pouvons l'étendre en une configuration y de $\mathcal{C}_{\max}(g)$. Dans ces conditions, y est isomorphe à T, donc la projection de y sur Σ_1 est isomorphe à $\Pi_{\Sigma_1}(T)$, or cette projection est exactement x, donc x est isomorphe à $\Pi_{\Sigma_1}(T)$, donc $\Pi_{\Sigma_1}(g)$ est une $\Pi_{\Sigma_1}(T)$-SE. □

Remarque 2 *Même si g est dans $\mathcal{G}(\Sigma, \mathrm{D})$, $\Pi_{\Sigma_1}(g)$ peut ne pas être dans $\mathcal{G}(\Sigma, \mathrm{D})$. En effet plaçons nous sur l'alphabet de dépendance $b - a - c$ et considérons alors l'exécution suivante :*



Si nous considérons maintenant la projection sur $\{b, c\}$, nous obtenons la structure d'événements suivante :

Cette structure d'événements n'est pas une exécution car les conflits initiaux ne sont pas sur des sommets étiquetés par la même lettre. En effet, entre autres, le conflit entre b_1 et c_2 est un conflit initial.

Nous allons maintenant définir le *produit série* de deux structures d'événements.

Pour ce faire nous allons tout d'abord définir le produit série sur les ordres partiels.

Soit deux ordres partiels $T_1 = (\Sigma_1, \leq_1)$ et $T_2 = (\Sigma_2, \leq_2)$. Nous définissons le produit série de T_1 et T_2 : $T_1 \odot T_2 = (\Sigma_1 \cup \Sigma_2, \leq)$ par $a < b$ si :

- ▷ $a, b \in \Sigma_2$ et $a <_2 b$
- ▷ $a, b \in \Sigma_1 \setminus \Sigma_2$ et $a <_1 b$
- ▷ $a \in \Sigma_1 \setminus \Sigma_2$ et $b \in \Sigma_2$

Nous pouvons maintenant définir le produit série sur les structures d'événements.

Définition 10 (Produit série).
Soit $T_1 = (\Sigma_1, \leq_1)$ et $T_2 = (\Sigma_2, \leq_2)$. Soit $\mathcal{T}_i$ la classe des T_i-SE pour $i \in \{1, 2\}$.

Soit $g = (S_g, \leq_g, \#_g, \ell_g, \sigma_g) \in \mathcal{T}_1$ *et* $\gamma : \mathcal{C}_{\max}(g) \to \mathcal{T}_2$ *avec* $\gamma(x) = (S_{\gamma(x)}, \leq_{\gamma(x)}, \#_{\gamma(x)}, \ell_{\gamma(x)}, \sigma_{\gamma(x)}) \in \mathcal{T}_2$ *pour tout* x *dans* $\mathcal{C}_{\max}(g)$. *Nous définissons alors le produit série* $g \odot \gamma = \mathsf{last}(S, \leq, \#, \ell, \sigma)$ *avec :*

- $S = S_g \uplus \biguplus_{x \in \mathcal{C}_{\max}(g)} S_{\gamma(x)}$
- $\ell = \ell_g \uplus \biguplus_{x \in \mathcal{C}_{\max}(g)} \ell_{\gamma(x)}$
- $\sigma = \sigma_g \uplus \biguplus_{x \in \mathcal{C}_{\max}(g)} \sigma_{\gamma(x)}$
- $e \leq f$ *si :*
  - $e \leq_g f$
  - *Il existe* x *dans* $\mathcal{C}_{\max}(g)$ *tel que* $e \leq_{\gamma(x)} f$
  - *Il existe* x *dans* $\mathcal{C}_{\max}(g)$ *tel que* $e \in x$ *et* $f \in S_{\gamma(x)}$
- $e \# f$ *s'il existe* $e' \leq e$ *et* $f' \leq f$ *tels que :*
  - $e' \#_g f'$
  - *Il existe* x *dans* $\mathcal{C}_{\max}(g)$ *tel que* $e' \#_{\gamma(x)} f'$

Nous noterons de plus $\mathcal{T}_1 \odot \mathcal{T}_2$ *l'ensemble des* $g \odot \gamma$ *avec* g *un élément de* $\mathcal{T}_1$ *et* γ *une fonction de* $\mathcal{C}_{\max}(g) \to \mathcal{T}_2$.

Nous allons maintenant montrer que le type de $g \odot \gamma$ est $T_1 \odot T_2$. Pour se faire, il suffit de considérer un élément y de $\mathcal{C}_{\max}(g \odot \gamma)$ et de montrer qu'il est isomorphe à $T_1 \odot T_2$. Considérons deux éléments de e_1 et e_2 de y et montrons qu'il sont ordonnés de la même manière que $\ell^{-1}(e_1)$ et $\ell^{-1}(e_2)$ dans $T_1 \odot T_2$. Il faut considérer 3 cas :

1. $\ell^{-1}(e_1) \in \Sigma_2$ et $\ell^{-1}(e_2) \in \Sigma_2$. Alors il existe $x \in \mathcal{C}_{\max}(g)$ tel que e_1 et e_2 soient dans la même configuration maximale de $S_\gamma(x)$, dans ces conditions e_1 et e_2 sont ordonnés comme $\ell^{-1}(e_1)$ et $\ell^{-1}(e_2)$ dans T_2, donc dans $T_1 \odot T_2$.
2. $\ell^{-1}(e_1) \in \Sigma_1 \setminus \Sigma_2$ et $\ell^{-1}(e_2) \in \Sigma_1 \setminus \Sigma_2$. Alors il existe $x \in \mathcal{C}_{\max}(g)$ tel que e_1 et e_2 soient dans x, donc e_1 et e_2 sont ordonnés comme $\ell^{-1}(e_1)$ et $\ell^{-1}(e_2)$ dans T_1, donc dans $T_1 \odot T_2$.
3. $\ell^{-1}(e_1) \in \Sigma_1 \setminus \Sigma_2$ et $\ell^{-1}(e_2) \in \Sigma_2$. Alors il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e_2 \in S_{\gamma(x)}$. Pour tout $e \in S_g \setminus x$, il existe $f \in x$ tel que $\ell(e) = \ell(f)$ et donc $e \# f < e_2$. On en déduit que $e \# e_2$ pour tout $e \in S_g \setminus x$. Par conséquent, $e_1 \in x$. Nous avons donc $e_1 < e_2$ de même que $\ell^{-1}(e_1) < \ell^{-1}(e_2)$ dans $T_1 \odot T_2$.

Nous allons maintenant énoncer deux lemmes techniques nous décrivant des propriétés du produit série dont nous allons avoir besoin par la suite.

Le premier de ces lemmes décrit le comportement du produit série vis à vis de certaine réduction :

Lemme 25 *Soit* g *une* T_1*-SE ,* $\gamma : \mathcal{C}_{\max}(g) \to \mathcal{T}_2$ *une fonction de* $\mathcal{C}_{\max}(g)$ *vers l'ensemble des* T_2*-SE. Supposons que pour tout* x *dans* $\mathcal{C}_{\max}(g)$, φ_x *est une réduction de* $\gamma(x)$. *On considère alors* $\gamma' : \mathcal{C}_{\max}(g) \to \mathcal{T}_2$ *qui à tout* x *de* $\mathcal{C}_{\max}(g)$ *associe* $\varphi_x(\gamma(x))$. *Alors il existe une réduction* φ *de* $g \odot \gamma$ *sur* $g \odot \gamma'$. *En particulier* $\overline{g \odot \gamma} = \overline{g \odot \gamma'}$.

Preuve Pour montrer ce résultat on commence par définir φ et on montre qu'il s'agit bien d'un réduction.

Soit $e \in g \odot \gamma$, si $e \in g$, alors on pose $\varphi(e) = e$, sinon il existe un unique $x \in \mathcal{C}_{\max}(g)$ tel que $e \in \gamma(x)$ et on pose alors $\varphi(e) = \varphi_x(e)$.

Par construction de $g \odot \gamma'$, on a bien $\varphi(g \odot \gamma) = g \odot \gamma'$, il nous suffit donc de montrer que φ est bien une réduction.

Soit e_1 et e_2 dans $g \odot \gamma$.

- ▷ Si $e_1 < e_2$, considérons 3 cas :
  - ◇ Si e_1 et e_2 sont dans g, alors $\varphi(e_1) = e_1$, $\varphi(e_2) = e_2$ et on a bien $\varphi(e_1) < \varphi(e_2)$.
  - ◇ Si ni e_1 ni e_2 ne sont dans g, alors il existe $x_1 \in \mathcal{C}_{\max}(g)$ tel que $e_1 \in \gamma(x_1)$ et $x_2 \in \mathcal{C}_{\max}(g)$ tel que $e_2 \in \gamma(x_2)$. De plus si $x_1 \neq x_2$, alors par construction $\gamma(x_1)$ est en conflit avec $\gamma(x_2)$ ce qui n'est pas possible car e_1 et e_2 sont ordonnés. Donc $x_1 = x_2 = x$ et $\varphi(e_1) = \varphi_x(e_1)$ et $\varphi(e_2) = \varphi_x(e_2)$. Comme φ_x est une réduction, nous en déduisons que $\varphi(e_1) < \varphi(e_2)$.
  - ◇ Si l'un est dans g et l'autre non, nous avons forcément $e_1 \in g$ par construction de $g \odot \gamma$. De plus il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e_1 \in x$ et $e_2 \in \gamma(x)$. Dans ces conditions, nous obtenons $\varphi(e_1) = e_1$ et $\varphi(e_2) = \varphi_x(e_2) \in \gamma'(x)$, donc $\varphi(e_1) < \varphi(e_2)$.
- ▷ Si $e_1 \natural e_2$, considérons encore les 3 mêmes cas :
  - ◇ Si e_1 et e_2 sont dans g, alors $\varphi(e_1) = e_1$, $\varphi(e_2) = e_2$ et on a bien $\varphi(e_1) \natural \varphi(e_2)$.
  - ◇ Si ni e_1 ni e_2 ne sont dans g, alors il existe $x_1 \in \mathcal{C}_{\max}(g)$ tel que $e_1 \in \gamma(x_1)$ et $x_2 \in \mathcal{C}_{\max}(g)$ tel que $e_2 \in \gamma(x_2)$. De plus si $x_1 \neq x_2$, alors par construction $\gamma(x_1)$ est en conflit avec $\gamma(x_2)$ ce qui n'est pas possible. Donc $x_1 = x_2 = x$ et $\varphi(e_1) = \varphi_x(e_1)$ et $\varphi(e_2) = \varphi_x(e_2)$. Comme φ_x est une réduction, nous en déduisons que $\varphi(e_1) \natural \varphi(e_2)$.
  - ◇ Si l'un est dans g et l'autre non, considérons $e_1 \in G$. Nous avons alors $e_1 < e_2$, car il ne peut pas y avoir d'élément en dehors de g qui soit compatible avec un élément de g sans être ordonné avec lui. En réutilisant le résultat montré ci-dessus, on obtient donc $\varphi(e_1) < \varphi(e_2)$, donc $\varphi(e_1) \natural \varphi(e_2)$.

Finalement φ est bien la réduction recherché. □

Le second lemme que nous allons montrer décrit le comportement du produit série vis à vis de certaine projection :

Lemme 26 *Soit g une T_1-SE , $\gamma : \mathcal{C}_{\max}(g) \to \mathcal{T}_2$ une fonction de $\mathcal{C}_{\max}(g)$ vers l'ensemble des T_2-SE. Posons $\Sigma_1 = \mathrm{Alph}(t_1)$ et $\Sigma_2 = \mathrm{Alph}(t_2)$. Alors nous avons $g \odot \gamma = \Pi_{\Sigma_1 \setminus \Sigma_2}(g) \odot \gamma'$ où :*

$$\gamma'(x) = \#_{\substack{y \in \mathcal{C}_{\max}(g) \\ \Pi_{\Sigma_1 \setminus \Sigma_2}(y) = x}} \gamma(y)$$

Preuve Commençons par montrer $g \odot \gamma \subseteq \Pi_{\Sigma_1 \setminus \Sigma_2}(g) \odot \gamma'$.

Soit $e \in g \odot \gamma$.

- ▷ Si $e \in g$, alors $\ell(e) \in \Sigma_1 \setminus \Sigma_2$, car sinon, il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e \in x$, et il existe $e' \in \gamma(x)$ tel que $\ell(e') = \ell(e)$. Dans ces conditions on a $e < e'$ et donc $e \notin g \odot \gamma$. Nous en déduisons donc que $e \in \Pi_{\Sigma_1 \setminus \Sigma_2}(g)$. Maintenant, il existe donc $y \in \mathcal{C}_{\max}(\Pi_{\Sigma_1 \setminus \Sigma_2}(g))$ tel que $e \in y$. Quelque soit $e' \in \gamma'(y)$, $\ell(e') \in \Sigma_2$, donc $\ell(e') \neq \ell(e)$, nous en déduisons donc que $e \in \Pi_{\Sigma_1 \setminus \Sigma_2}(g) \odot \gamma'$.

▷ Sinon, il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e \in \gamma(x)$, mais dans ces conditions, $\Pi_{\Sigma_1\setminus\Sigma_2}(x) \in \mathcal{C}_{\max}(\Pi_{\Sigma_1\setminus\Sigma_2}(g))$ et $e \in \gamma'(\Pi_{\Sigma_1\setminus\Sigma_2}(x))$, donc $e \in \Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.

Montrons maintenant la réciproque.

Considérons $e \in \Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.

▷ Si $e \in \Pi_{\Sigma_1\setminus\Sigma_2}(g)$, alors $e \in g$ et il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e \in x$. De plus quelque soit $e' \in \gamma(x)$, $\ell(e) \neq \ell(e')$, donc $e \in g \odot \gamma$.

▷ Sinon, il existe $y \in \mathcal{C}_{\max}(\Pi_{\Sigma_1\setminus\Sigma_2}(g))$ tel que $e \in \gamma'(y)$. Donc il existe $x \in \mathcal{C}_{\max}(g)$ tel que $\Pi_{\Sigma_1\setminus\Sigma_2}(y) = x$ et $e \in \gamma(x)$, donc $e \in g \odot \gamma$.

Il nous reste maintenant à montrer que la relation d'ordre et la relation de conflit est la même dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.

Pour ce faire, nous considérons e_1 et e_2 dans $g \odot \gamma$ et nous allons devoir considérer 5 cas :

1. Si e_1 et e_2 sont dans g, alors e_1 et e_2 sont dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g)$, et donc la relation d'ordre et la relation de conflit est la même pour e_1 et e_2 dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.
2. Si $\exists x \in \mathcal{C}_{\max}(g)$ tel que $e_1, e_2 \in \gamma(x)$, alors e_1 et e_2 sont dans $\gamma'(\Pi_{\Sigma_1\setminus\Sigma_2}(x))$, et donc la relation d'ordre et la relation de conflit est la même pour e_1 et e_2 dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.
3. Si $\exists x, y \in \mathcal{C}_{\max}(g)$ tels que $e_1 \in \gamma(x)$ et $e_2 \in \gamma(y)$ avec $x \neq y$, alors $e_1 \# e_2$ dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.
4. Si $\exists x \in \mathcal{C}_{\max}(g)$ tel que $e_1 \in x$ et $e_2 \in \gamma(x)$, alors $e_1 \in \Pi_{\Sigma_1\setminus\Sigma_2}(x)$ et $e_2 \in \gamma'(\Pi_{\Sigma_1\setminus\Sigma_2}(x))$, donc $e_1 < e_2$ dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.
5. Si aucun des autres cas ne s'appliquent alors $\exists x, y \in \mathcal{C}_{\max}(g)$ avec $e_1 \in x$ et $e_2 \in \gamma(y)$ et $\Pi_{\Sigma_1\setminus\Sigma_2}(x) \neq \Pi_{\Sigma_1\setminus\Sigma_2}(y)$, dans ces conditions $e_1 \in \Pi_{\Sigma_1\setminus\Sigma_2}(x)$ et $e_2 \in \gamma'(\Pi_{\Sigma_1\setminus\Sigma_2}(y))$, donc $e_1 \# e_2$ dans $g \odot \gamma$ et dans $\Pi_{\Sigma_1\setminus\Sigma_2}(g) \odot \gamma'$.

Nous en déduisons le résultat. □

Nous allons maintenant énoncer un lemme intermédiaire qui nous permettra de construire les élément de $\mathcal{G}(\Sigma, \mathrm{D})$ en fonction des éléments de $\mathcal{G}(A, \mathrm{D})$ et $\mathcal{G}(B, \mathrm{D})$.

Lemme 27 *Soit $t = t_1 t_2 = (V, \leq, \ell)$ une trace de $\mathbb{M}(\Sigma, \mathrm{D})$ avec $t_i = (V_i, \leq, \ell)$ telle que $V_1 \times V_2 \subseteq <$. Soit $\Sigma_2 = \mathrm{Alph}(t_2)$. Soit w_i une linéarisation de t_i et $\emptyset \xrightarrow{w_1} g \xrightarrow{w_2} h$ une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. Alors il existe $\gamma : \mathcal{C}_{\max}(g) \to G(\Sigma_2, D)$ telle que $h = \overline{g \odot \gamma}$*

Preuve Nous commençons par écrire $g = (V_g, \leq_g, \#_g, \ell_g, \sigma_g)$ et $h = (V_h, \leq_h, \#_h, \ell_h, \sigma_h)$. Puis, en utilisant le corollaire 20, nous considérons une exécution ρ sur t tel que $g = \overline{\mathsf{last}(\rho_{|t_1})}$ et $h = \overline{\mathsf{last}(\rho)}$. La structure d'événements ρ peut s'écrire $(W_1 \uplus W_2, \leq, \#, \ell, \sigma)$ où $(W_1, \leq, \#, \ell, \sigma)$ est exactement la structure d'événements $\rho_{|t_1}$.

Nous allons alors maintenant considérer $x \in \mathcal{C}_{\max}(\rho_{|t_1})$ et nous lui associons $\mathrm{rest}(x) = \{e \in W_2 \mid x \subseteq {\downarrow}e\}$.

Nous allons tout d'abord montrer que pour $x, y \in \mathcal{C}_{\max}(\rho_{|t_1})$, $x \neq y \Rightarrow \mathrm{rest}(x) \times \mathrm{rest}(y) \subseteq \#$.

Cette propriété est très simple à montrer. Comme x et y sont dans $\mathcal{C}_{\max}(\rho_{|t_1})$, il existe un élément e dans x et un élément f dans y tels que $e \# f$. En effet

sinon, $x \cup y$ serait sans conflit, or $x \neq y$, donc $x \cup y$ est plus grand que x et y, donc ni x ni y ne serait dans $\mathcal{C}_{\max}(\rho_{|t_1})$. Considérons maintenant e' dans $\mathrm{rest}(x)$ et f' dans $\mathrm{rest}(y)$. Par définition, nous avons $x \subseteq \downarrow e'$, donc $e < e'$ et de même $f < f'$, donc $e' \# f'$. Ceci montre donc la propriété, qui entraîne de plus que pour tout $x, y \in \mathcal{C}_{\max}(\rho_{|t_1})$, si $x \neq y$, alors $\mathrm{rest}(x)$ et $\mathrm{rest}(y)$ sont disjoint.

Nous montrons maintenant que pour tout élément e' de W_2, il existe un (unique) $x \in \mathcal{C}_{\max}(\rho_{|t_1})$ tel que $e' \in \mathrm{rest}(x)$. L'unicité est dûe à la propriété que nous venons de démontrer.

Considérons un élément e' de W_2, il existe une configuration y de $\mathcal{C}_{\max}(\rho)$ tel que $e' \in y$, alors il existe une configuration x de $\mathcal{C}_{\max}(\rho_{|t_1})$ tel que $x = y \cap W_1$, donc $x \subseteq y$, mais comme $V_1 \times V_2 \subseteq <$, nous en déduisons que quelque soit e dans x, on a $e < e'$, donc e' est dans $\mathrm{rest}(x)$.

Ceci montre de plus que $\downarrow \mathrm{rest}(x) \cap W_1 = x$.

Nous allons maintenant montrer que $\mathrm{rest}(x)$ est une exécution de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$. Les conditions (10.3) et (10.4) sont évidentes.

Nous montrons la condition (10.1). Considérons e'_1 et e'_2 dans $\mathrm{rest}(x)$ tels que l'on a $e'_1 \#^i e'_2$ dans $\mathrm{rest}(x)$. Nous allons alors montrer que $e'_1 \#^i e'_2$ dans ρ. En effet, si ce n'est pas le cas, il existe $e_1 < e'_1$ avec $e_1 \# e'_2$. Comme $e'_1 \#^i e'_2$ dans $\mathrm{rest}(x)$, on en déduit que e_1 n'est pas dans $\mathrm{rest}(x)$. Mais alors $e_1 \in x \subseteq \Downarrow e'_2$ ce qui est impossible puisque $\downarrow e'_2$ est sans conflit.

Nous montrons la condition (10.2). Nous considérons donc un élément z de $\mathcal{C}_{\max}(\mathrm{rest}(x))$, nous pouvons l'étendre en un élément $z' = z'_1 z'_2$ de $\mathcal{C}_{\max}(\rho)$ avec $z'_1 \sim t_1$ et $z'_2 \sim t_2$ puisque ρ est une exécution sur $t = t_1 t_2$. Pour montrer que $\mathrm{rest}(x)$ vérifie la condition (10.2), il suffit de montrer que $z = z_2$. Nous avons $x \subseteq z'_1 z'_2$ car quelque soit $e \in \mathrm{rest}(x)$, nous avons $x \subseteq \downarrow e$, de plus $x = z'_1$ car $x \in \mathcal{C}_{\max}(\rho_{|t_1})$, il suffit donc de montrer que quelque soit $e \in z'$, si e n'est pas dans x alors e est dans z. Si e n'est pas dans z, alors e appartient à W_2, car x est un ensemble maximal sans conflit de W_1. Dans ces conditions il existe $y \in \mathcal{C}_{\max}(\rho_{|t_1})$ tel que $e \in \mathrm{rest}(y)$, mais dans ces conditions $y = x$ car sinon $\mathrm{rest}(y)$ et $\mathrm{rest}(x)$ sont en conflit. Nous en déduisons donc que e est dans $\mathrm{rest}(x)$ et e est sans conflit avec z qui est un élément maximal de $\mathrm{rest}(x)$, donc e est un élément de z ce qui prouve la condition (10.2).

Finalement, nous en déduisons que $\mathrm{rest}(x)$ est une exécution de $\mathcal{U}_{\Sigma,\mathrm{D},Q}$.

Nous considérons maintenant un élément x de $\mathcal{C}_{\max}(g)$. La configuration $\downarrow x$ est alors un élément de $\mathcal{C}_{\max}(\rho_{|t_1})$. Nous définissons alors la fonction γ de $\mathcal{C}_{\max}(g)$ dans $\mathcal{T}_2$ de la manière suivante : $\gamma(x) = \mathsf{last}(\mathrm{rest}(\downarrow x))$, puis nous définissons $\overline{\gamma} : \mathcal{C}_{\max}(g) \to \mathcal{G}(\Sigma_2, \mathrm{D})$ par $\overline{\gamma}(x) = \overline{\gamma(x)}$.

Nous montrons maintenant que $h = \overline{g \odot \gamma} = \overline{g \odot \overline{\gamma}}$.

Tout d'abord le lemme 25 nous assure que $\overline{g \odot \gamma} = \overline{g \odot \overline{\gamma}}$.

Il nous suffit donc de montrer que $h = \overline{g \odot \gamma}$. Pour ce faire, nous appelons ρ' l'exécution induite par l'ensemble des sommets $W' = \downarrow g \cup \bigcup_{x \in \mathcal{C}_{\max}(g)} \mathrm{rest}(\downarrow x)$.

Il nous faut déjà montrer que ρ' est bien une exécution, donc qu'elle vérifie les conditions (10.1) et (10.2), les conditions (10.3) et (10.4) étant évidentes.

Nous montrons la condition (10.1). Il suffit de vérifier que l'ensemble $W' = \downarrow W'$ dans ρ. Comme ρ vérifie la condition (10.1) cette propriété suffit à montrer que ρ' la vérifie également. Considérons donc un élément e_1 dans ρ tel qu'il existe e_2 dans ρ' avec $e_1 < e_2$. Si e_2 est un élément de $\downarrow g$ alors e_1 est également un élément de $\downarrow g$ et donc e_1 est dans ρ'. Sinon il existe $x \in \mathcal{C}_{\max}(g)$ tel que

e_2 est dans $\text{rest}(\downarrow x)$. Quelque soit $y \in \mathcal{C}_{\max}(\rho_{|t_1})$, si $y \neq \downarrow x$, e_1 ne peut pas appartenir à $\text{rest}(y)$ car $\text{rest}(y)$ et $\text{rest}(\downarrow x)$ sont en conflits. Si e_1 appartient à $\text{rest}(\downarrow x)$ alors e_1 est dans ρ'. Sinon e_1 est donc un élément de W_1. Dans ces conditions, $e_1 \in \downarrow e_2 \cap W_1 = \downarrow x \subseteq \downarrow g$. Finalement e_1 est donc bien dans ρ' ce qui montre la condition (10.1).

Nous montrons maintenant la condition (10.2). Nous considérons donc une configuration y de $\mathcal{C}_{\max}(\rho')$. Elle s'écrit $y_1 \uplus y_2$ avec $y_1 \subseteq \downarrow g$ et $y_2 \subseteq \text{rest}(\downarrow x)$ pour un certain x dans $\mathcal{C}_{\max}(g)$, en effet, il ne peut pas y avoir d'élément dans deux ensembles $\text{rest}(\downarrow x_1)$ et $\text{rest}(\downarrow x_2)$ pour x_1 et x_2 distincts dans $\mathcal{C}_{\max}(g)$ car alors $\text{rest}(\downarrow x_1)$ et $\text{rest}(\downarrow x_2)$ sont en conflits. De plus y_1 est inclus dans $\downarrow x$ car nous avons montrer dans la preuve de la condition (10.1) que tout élément de $W_1 \setminus \downarrow x$ est en conflit avec $\text{rest}(\downarrow x)$. De plus $\downarrow x \cup \text{rest}(\downarrow x)$ est un ensemble sans conflit, on en déduit donc que $y = \downarrow x \cup \text{rest}(\downarrow x)$. On a alors déjà montrer que $\downarrow x \cup \text{rest}(\downarrow x)$ est isomorphe à t, et on en déduit donc la condition (10.2).

Par définition de ρ (voir corollaire 20), nous avons $h = \overline{\mathsf{last}(\rho)} = \overline{\text{next}_\rho(g, t_2)}$. Or par construction $\text{next}_\rho(g, t_2) = \mathsf{last}(\rho')$, donc $h = \overline{\mathsf{last}(\rho')}$. Il nous suffit donc de montrer que $\mathsf{last}(\rho') = g \odot \gamma$

Montrons tout d'abord que $\mathsf{last}(\rho') \subseteq g \odot \gamma$. Soit e dans $\mathsf{last}(\rho')$. Le sommet e est alors dans $\mathsf{last}(\rho)$. Si $\ell(e) \in \Sigma_2$, alors e est dans $\text{rest}(\downarrow x)$ pour $x = \downarrow e \cap g \in \mathcal{C}_{\max}(g)$, donc e est dans $\gamma(x)$ et donc $e \in g \odot \gamma$, sinon e est dans $\mathsf{last}(\downarrow g) = g$ et par définition de $g \odot \gamma$, e est dans $g \odot \gamma$.

Il nous faut maintenant montrer que $g \odot \gamma \subseteq \mathsf{last}(\rho')$. Considérons donc e dans $g \odot \gamma$. S'il existe $x \in \mathcal{C}_{\max}(g)$ tel que $e \in \gamma(x)$, alors $e \in \mathsf{last}(\text{rest}(\downarrow x))$ donc $e \in \mathsf{last}(\rho')$ et sinon e est dans g et $\ell(e) \notin \Sigma_2$, donc e est encore une fois dans $\mathsf{last}(\rho')$.

Il nous reste maintenant à montrer que la relation d'ordre et la relation de conflit est la même dans $g \odot \gamma$ et dans $\mathsf{last}(\rho')$.

Pour ce faire, nous considérons e_1 et e_2 dans $g \odot \gamma$ et nous allons devoir considérer 5 cas :

1. Si e_1 et e_2 sont dans g, alors e_1 et e_2 sont dans $\mathsf{last}(\rho')$, et donc la relation d'ordre et la relation de conflit est la même pour e_1 et e_2 dans $g \odot \gamma$ et dans $\mathsf{last}(\rho')$.
2. Si $\exists x \in \mathcal{C}_{\max}(g)$ tel que $e_1, e_2 \in \gamma(x)$, alors e_1 et e_2 sont dans $\text{rest}(\downarrow x)$, et donc la relation d'ordre et la relation de conflit est la même pour e_1 et e_2 dans $g \odot \gamma$ et dans $\mathsf{last}(\rho')$.
3. Si $\exists x, y \in \mathcal{C}_{\max}(g)$ tels que $e_1 \in \gamma(x)$ et $e_2 \in \gamma(y)$ avec $x \neq y$, alors $e_1 \# e_2$ dans $g \odot \gamma$. On en déduit également $e_1 \in \text{rest}(\downarrow x)$ et $e_2 \in \text{rest}(\downarrow y)$, donc on a aussi $e_1 \# e_2$ dans $\mathsf{last}(\rho')$.
4. Si $\exists x \in \mathcal{C}_{\max}(g)$ tel que $e_1 \in x$ et $e_2 \in \gamma(x)$, alors on a $e_2 \in \text{rest}(\downarrow x)$ et on a donc $e_1 < e_2$ dans $g \odot \gamma$ et dans $\mathsf{last}(\rho')$.
5. Si aucun des autres cas ne s'appliquent alors $\exists x, y \in \mathcal{C}_{\max}(g)$ avec $x \neq y$, $e_1 \in x \setminus y$ et $e_2 \in \gamma(y)$. Dans ces conditions $e_1 \# e_2$ dans $g \odot \gamma$. De plus il existe $e_1' \in y$ tel que $e_1 \# e_1'$. On a également $e_2 \in \text{rest}(\downarrow y)$, donc on a $e_1' < e_2$ et donc $e_1 \# e_2$ dans $\mathsf{last}(\rho')$.

Ceci montre donc que $g \odot \gamma = \mathsf{last}(\rho')$ et donc $h = \overline{\mathsf{last}(\rho')} = \overline{g \odot \gamma}$.
Finalement, nous en déduisons le résultat.

□

Nous allons maintenant affiner le résultat précédent.

Soit T un type sur Σ. Nous dirons qu'une T-SE h appartient à $2^{\mathcal{G}(\Sigma,\mathrm{D})}$, s'il existe $g_1, g_2, \ldots, g_n$, n T-SE *deux à deux non isomorphes* de $\mathcal{G}(\Sigma,\mathrm{D})$ telles que $h = g_1 \# g_2 \# \cdots \# g_n$. On peut remarquer que même si $g_1, g_2, \ldots, g_n$ sont deux à deux non isomorphes, h peut tout à fait ne pas être réduit. En effet considérons deux T-SE non isomorphes g_1 et g_2 telles qu'il existe une réduction de g_1 sur g_2, alors $\overline{g_1 \# g_2} = g_2 \neq (g_1 \# g_2)$.

Lemme 28 *Soit $t = t_1 t_2 = (V, \leq, \ell)$ une trace de $\mathbb{M}(\Sigma,\mathrm{D})$ avec $t_i = (V_i, \leq, \ell)$ telle que $V_1 \times V_2 \subseteq <$. Soit $\Sigma_1 = \mathrm{Alph}(t_1)$, $\Sigma_2 = \mathrm{Alph}(t_2)$ et $\Sigma_1' = \Sigma_1 \setminus \Sigma_2$. Soit w_i une linéarisation de t_i et $\emptyset \xrightarrow{w_1} g \xrightarrow{w_2} h$ une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. Alors nous avons $h \in \overline{\Pi_{\Sigma_1'}(g) \odot 2^{\mathcal{G}(\Sigma_2,\mathrm{D})}}$.*

Preuve Pour montrer ce résultat, nous allons utiliser les lemmes 25 et 26. Le lemme précédent nous assure qu'il existe une fonction $\gamma : \mathcal{C}_{\max}(g) \to \mathcal{G}(\Sigma_2,\mathrm{D})$ telle que $h = \overline{g \odot \gamma}$. De plus pour $x \in \mathcal{C}_{\max}(\Pi_{\Sigma_1'}(g))$, nous posons $\gamma'(x) = \#_{\substack{y \in \mathcal{C}_{\max}(g) \\ \Pi_{\Sigma_1'}(y) = x}} \gamma(x)$ et le lemme 26 nous assure que $g \odot \gamma = \Pi_{\Sigma_1'}(g) \odot \gamma'$.

Maintenant si nous considérons $\gamma'(x)$, il existe un élément $\gamma''(x)$ de $2^{\mathcal{G}(\Sigma_2,\mathrm{D})}$ tel qu'il y aie une réduction de $\gamma'(x)$ sur $\gamma''(x)$: il suffit de supprimer les graphes de $\mathcal{G}(\Sigma_2,\mathrm{D})$ qui apparaissent plusieurs fois. Le lemme 25 nous assure donc que $h = \overline{g \odot \gamma} = \overline{\Pi_{\Sigma_1'}(g) \odot \gamma''} \in \overline{\Pi_{\Sigma_1'}(g) \odot 2^{\mathcal{G}(\Sigma_2,\mathrm{D})}}$. □

Maintenant que nous avons ce lemme, il nous reste à montrer que nous pouvons construire $\Pi_{\Sigma_1}(g)$ à partir des élément de $\mathcal{G}(\Sigma_1,\mathrm{D})$. Pour ce faire, nous avons besoin du lemme suivant :

Lemme 29 *Soit $t = t_0 t_1 = (V, \leq, \ell)$ une trace de $\mathbb{M}(\Sigma,\mathrm{D})$ avec $t_i = (V_i, \leq, \ell)$ telle que $V_0 \times V_1 \subseteq <$. Soit $\Sigma_0 = \mathrm{Alph}(t_0)$, $\Sigma_1 = \mathrm{Alph}(t_1)$ et $\Sigma_1' \subseteq \Sigma_1$. Soit w_i une linéarisation de t_i et $\emptyset \xrightarrow{w_0} f \xrightarrow{w_1} g$ une exécution de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. Alors chaque composante connexe de $\Pi_{\Sigma_1'}(g)$ est contenu dans un graphe de $\Pi_{\Sigma_1'}(\mathcal{G}(\Sigma_1,\mathrm{D}))$.*

Preuve Pour prouver ce résultat, on commence par utiliser le lemme 27 qui nous permet d'écrire $g = \overline{f \odot \gamma}$ où γ est une fonction de $\mathcal{C}_{\max}(f)$ dans $\mathcal{G}(\Sigma_1,\mathrm{D})$. Nous obtenons alors que $\Pi_{\Sigma_1'}(f \odot \gamma) = \Pi_{\Sigma_1'}(\#_{x \in \mathcal{C}_{\max}(f)} \gamma(x))$ et donc $\Pi_{\Sigma_1'}(f \odot \gamma) = \#_{x \in \mathcal{C}_{\max}(f)} \Pi_{\Sigma_1'}(\gamma(x))$. Dans ces conditions, comme g est incluse dans $f \odot \gamma$ car le réduit d'une structure d'événements est toujours inclus dans cette structure d'événements, nous en déduisons que $\Pi_{\Sigma_1'}(g)$ est incluse dans $\Pi_{\Sigma_1'}(f \odot \gamma)$. Il s'en suit que pour chaque composante connexe de $\Pi_{\Sigma_1'}(g)$ il existe $x \in \mathcal{C}_{\max}(f)$ tel que cette composante est incluse dans $\Pi_{\Sigma_1'}(\gamma(x))$ qui est un élément de $\Pi_{\Sigma_1'}(\mathcal{G}(\Sigma_1,\mathrm{D}))$ ce qui prouve le lemme. □

Nous pouvons maintenant montrer la proposition 23 dans le cas série. En effet, considérons une structure d'événements h de $\mathcal{G}(\Sigma,\mathrm{D})$. Il existe donc une exécution $\emptyset \xrightarrow{w} h$ de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$.

Si $\mathrm{Alph}(w) \neq \Sigma$, alors $h \in \mathcal{G}(\Sigma',\mathrm{D})$ pour $\Sigma' = \mathrm{Alph}(w)$. Par hypothèse d'induction, $\mathcal{G}(\Sigma',\mathrm{D})$ est fini, et le résultat est prouvé.

Maintenant si $\mathrm{Alph}(w) = \Sigma$, nous pouvons considérer la factorisation de w en $v_0 v_1 v_2 \cdots v_n$ où pour chaque i, v_i est non vide et $\mathrm{Alph}(v_i) \subseteq A$ ou $\mathrm{Alph}(v_i) \subseteq B$

et telle que $v_i \times v_{i+1} \subseteq <$. Il existe alors un k tel que $\mathrm{Alph}(v_k \cdots v_n) = \Sigma$ et $\mathrm{Alph}(v_{k+1} \cdots v_n) \subsetneq \Sigma$. De plus, ce k est différent de n, car $\mathrm{Alph}(v_n) \subseteq A$ ou $\mathrm{Alph}(v_n) \subseteq B$ et A et B sont strictement inclus dans Σ. On appelle alors $w_0 = v_0 \cdots v_{k-1}$, $w_1 = v_k$ et $w_2 = v_{k+1} \cdots v_n$. Les facteurs w_1 et w_2 sont alors non vides et on a les propriétés suivantes : $w_0 \times w_1 \subseteq <$, $w_1 \times w_2 \subseteq <$, $\mathrm{Alph}(w_1 w_2) = \Sigma$, $\emptyset \neq \mathrm{Alph}(w_1) \neq \Sigma$ et $\emptyset \neq \mathrm{Alph}(w_2) \neq \Sigma$. On pose alors $\Sigma_i = \mathrm{Alph}(w_i)$ et $\Sigma'_1 = (\Sigma_0 \cup \Sigma_1) \setminus \Sigma_2$. Comme $\Sigma_1 \cup \Sigma_2 = \Sigma$, on a $\Sigma'_1 = \Sigma_1 \setminus \Sigma_2 \subseteq \Sigma_1$. On remarque que Σ_1 et Σ_2 sont strictement inclus dans Σ.

L'exécution $\emptyset \xrightarrow{w} h$ s'écrit alors $\emptyset \xrightarrow{w_0} f \xrightarrow{w_1} g \xrightarrow{w_2} h$.

Nous utilisons alors le lemme 28 qui nous permet d'écrire $h = \overline{\Pi_{\Sigma'_1}(g) \odot \gamma'_2}$ où γ'_2 est une fonction de $\mathcal{C}_{\max}(\Pi_{\Sigma'_1}(g))$ dans $2^{\mathcal{G}(\Sigma_2,\mathrm{D})}$. Nous utilisons alors le lemme 29 qui nous assure que chaque composante connexe de $\Pi_{\Sigma'_1}(g)$ est inclus dans une composante connexe d'un graphe de $\Pi_{\Sigma'_1}(\mathcal{G}(\Sigma_1, \mathrm{D}))$.

Maintenant par induction, nous savons que $\mathcal{G}(\Sigma_1, \mathrm{D})$ est un ensemble fini, donc chaque composante connexe de $\Pi_{\Sigma'_1}(g)$ est dans un ensemble fini. On peut alors écrire $\Pi_{\Sigma'_1}(g) = g^1 \# g^2 \# \cdots \# g^n$ où chacun des g^k est à prendre dans un ensemble fini. Dans ces conditions on peut séparer γ'_2 en ${\gamma'}_2^1$, ${\gamma'}_2^2$, …, ${\gamma'}_2^n$ et $\Pi_{\Sigma'_1}(g) \odot \gamma'_2 = (g^1 \odot {\gamma'}_2^1) \# (g^2 \odot {\gamma'}_2^2) \# \cdots \# (g^n \odot {\gamma'}_2^n)$ Maintenant, encore une fois par induction, $\mathcal{G}(\Sigma_2, \mathrm{D})$ est un ensemble fini, donc $2^{\mathcal{G}(\Sigma_2,\mathrm{D})}$ est également dans un ensemble fini. Nous en déduisons que chacun des $g^k \odot {\gamma'}_2^k$ fait partie d'un ensemble fini. De plus, si il existe deux indices pour lesquels $g^k \odot {\gamma'}_2^k$ prend la même valeur, alors on peut en supprimer un des deux sans changer le réduit. Nous en déduisons finalement que nous pouvons construire h à l'aide des élément de $\Pi_{\Sigma'_1}(\mathcal{G}(\Sigma_1, \mathrm{D}))$ et des éléments de $2^{\mathcal{G}(\Sigma_2,\mathrm{D})}$, nous avons donc montrer que h appartient à un ensemble fini.

Finalement nous avons donc prouvé la proposition 23.

12.3 Résultat

La proposition 23 nous permet donc de prouver le théorème qui nous intéresse :

Théorème 30 *Soit $A = (Q, q^0, (\delta_a)_{a\in\Sigma}, \mathcal{F})$ un automate alternant cellulaire asynchrone sur un alphabet (Σ, D) série parallèle, alors $\mathcal{L}(\mathcal{A})$ est un langage reconnaissable.*

Pour prouver ce résultat, il suffit de montrer que $\mathcal{R}_\mathcal{A}$ est un automate fini. Or les états de $\mathcal{R}_\mathcal{A}$ sont contenus dans les états de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$. Il suffit donc de montrer que l'ensemble des états de $\mathcal{R}_{\mathcal{U}_{\Sigma,\mathrm{D},Q}}$ est un ensemble fini. Or ce résultat est vrai d'après la proposition 23. Nous en déduisons donc le théorème.

Quatrième partie

Jeux distribués

Chapitre 13

Motivations

La synthèse de contrôleur est un problème qui a été considéré par de nombreux auteurs pour différents types de systèmes (séquentiels, concurrents, temporisés, probabilistes) et pour différents types de spécifications (logique temporelle linéaire ou branchante par exemple) [8, 26, 1, 47, 44, 38, 4, 51, 53, 56, 5, 13, 32, 36, 37].

Le problème de synthèse de contrôleur peut également se décrire comme un problème de théorie des jeux. En effet, le système est modélisé par l'arène du jeu, le contrôleur par un joueur et l'environnement par un second joueur. La spécification est alors modélisée par la condition de gain et trouver une implémentation correcte revient alors à trouver une stratégie gagnante pour le joueur contrôleur. Lorsque le système est un système distribué, nous parlerons du problème de synthèse de contrôleurs distribués.

Nos travaux sur les automates asynchrones nous ont amenés à considérer un nouveau type de jeux distribués et donc un nouveau problème de synthèse de contrôleurs distribués. Ces jeux se différencient des différents types de jeux présentés jusqu'à présent par la notion de mémoire que nous utilisons. Nous avons montré que cette notion de mémoire que nous appellerons *mémoire causale* nous permettait de calculer effectivement l'existence de stratégies gagnantes dans le cadre de certaines architectures.

Nous allons présenter différents modèles de synthèse de contrôleurs distribués, puis nous présenterons notre cadre et nous montrerons alors un algorithme pour décider de l'existence d'une stratégie gagnante sans mémoire et pour la calculer le cas échéant. Enfin nous montrerons que sur certaines architectures, il est décidable de savoir s'il existe une stratégie gagnante.

Chapitre 14

État de l'art

14.1 Synthèse de contrôleur

Nous allons nous intéresser à la synthèse de contrôleur distribué. Nous nous donnons un *système distribué réactif* qui exécute un programme en interaction avec un environnement. Nous modélisons ce système par un système de transition *asynchrone* [63] constitué de plusieurs processus. Il peut exécuter des action locales et des actions de synchronisation mettant en oeuvre plusieurs processus. Une telle action commence par lire les états des processus mis en jeux, puis, en fonction de ce qu'elle a lu, choisit une transition changeant les états des processus impliqués. Si nous interprétons les processus comme des mémoires, ce paradigme suggère une communication par mémoire partagée et explique la terminologie utilisée. Cependant, ces systèmes asynchrones peuvent simuler sans difficulté d'autres moyens de communication tels des canaux point à point. Un autre avantage de ce modèle est de considérer les actions de l'environnement comme n'importe quelle autre action du système réactif. Nous nous donnons également une *spécification*, c'est à dire une propriété définissant les comportements attendus du système. Le problème de la synthèse de contrôleur distribué consiste alors à calculer un contrôleur *distribué* sur le même ensemble de processus (les actions du contrôleur observent les états locaux de certains processus). Avec ces informations, le contrôleur doit permettre ou interdire les actions *contrôlables* du système de manière à ce que le système dans son ensemble se conduise correctement vis à vis de la spécification.

Il a été montré que ce problème est décidable et a une solution optimale dans le cas séquentiel [47], c'est à dire quand il n'y a qu'un unique processus. Pour les systèmes distribués, cependant, les résultat sont plus partiels. Plusieurs modèles ont déjà été introduits jusqu'à présent (fussent-ils présentés dans la terminologie de la théorie du contrôle ou des jeux). Pour des processus synchrones, communicant par l'intermédiaire de "buffers", le problème est indécidable [45] pour des spécifications exprimées en LTL sauf pour un nombre très réduit d'architectures. Des travaux récents [36, 33] étendent ces résultats, par exemple pour des spécifications locales (des spécifications ne parlant que des actions d'un unique processus). L'approche de [39] unifie [45, 36, 33]. Dans tous les modèles, une raison majeure menant à l'indécidabilité est la possibilité pour la spécification d'exprimer des propriétés liées à l'observation d'une linéarisation particulière des

actions des processus sans prendre en compte les possibilités d'actions concurrentes. Un autre modèle de système distribué a été étudié en [37] et il a été montré que l'existence d'un contrôleur spécifique est décidable pour des langages de spécifications ne pouvant pas distinguer deux linéarisations différentes de la même exécution concurrente.

Les systèmes que nous avons introduits englobent ceux de [45, 36, 33, 39] et de [37]. Dans [37], les transitions globales du système sont obtenues en synchronisant les actions locales des processus, et les transitions de l'environnement sont locales. Dans notre modèle, en revanche, une transition d'un action synchronisée dépend également de l'état de tous les processus impliqués de telle manière que les fonctions de transitions des actions ne sont plus nécessairement un produit cartésien de fonctions de transitions locales. De plus, les actions de l'environnement peuvent être définies de manière globale. Le systèmes de [39] dans lesquels les actions de l'environnement sont globales et les transitions du système purement locales (les communications se faisant par l'intermédiaire de l'environnement) peuvent également être modélisés naturellement dans notre cadre. Une autre différence est que [37, 39] considèrent des contrôleurs à mémoire *locale*, c'est à dire que le contrôleur ne peut se baser que sur les événements passés qui se sont déroulés sur son processus pour décider de la prochaine action. Notre cadre considère des contrôleurs à mémoire *causale* : un contrôleur peut avoir accès aux informations calculées par d'autres processus au cours de l'exécution. L'existence d'un contrôleur distribué dans le cadre de [37, 39] implique l'existence d'un contrôleur distribué dans notre cadre. Comme la proposition réciproque n'est pas vraie, on ne peut pas étendre les résultats d'indécidabilité de [37, 39] à notre cadre.

14.2 Jeux

Notre objectif principal est de modéliser le problème de la synthèse de contrôleur distribué par des *jeux*. Les jeux séquentiels à deux joueurs fournissent un outil naturel et largement utilisé pour modéliser les systèmes réactifs [42, 54, 64, 58]. Le joueur 0 représente le système et le joueur 1 représente l'environnement. Les règles du jeu décrivent les interactions possibles entre eux et la condition de gain pour le joueur 0 exprime la spécification que le système doit remplir. Ainsi, décider si le joueur 0 a une stratégie gagnante correspond à décider si le système peut être contrôlé afin de vérifier la spécification et calculer une stratégie gagnante pour le joueur 0 revient à résoudre le problème de la synthèse de contrôleur.

Les jeux distribués que nous introduisons correspondent aux systèmes asynchrones et fournissent un cadre naturel pour étudier le problème de synthèse de contrôleur distribué. Dans ces jeux, deux équipes jouent l'une contre l'autre. Les joueurs de l'équipe 0 peuvent être vus comme les actions contrôlables d'un système distribué qui coopèrent afin de vérifier la spécification quelle que soit la manière dont l'environnement (les joueurs de l'équipe 1) se conduit. Tous les joueurs utilisent un ensemble de variables partagées pour transmettre de l'information. Le jeu n'est pas un jeu tour par tour : dans chaque position du jeu, plusieurs joueurs de l'équipe 0 et de l'équipe 1 peuvent être en mesure de jouer en même temps. Ainsi, le jeu est asynchrone, contrairement au cadre de [39] où a chaque étape les joueurs jouent de manière synchrone. Si on considère que les

dépendances entre les actions sont fixées (c'est à dire qu'elles ne dépendent pas du contexte), une partie est une trace de Mazurkiewicz.

Nous définissons également la notion de stratégie distribuée pour une équipe sur un tel jeu. De manière informelle, une stratégie est distribuée, si un coup prévu par la stratégie pour un joueur dépend uniquement de la vue causale de ce joueur. Dans ce contexte, les jeux ne sont pas déterminés, c'est à dire qu'il existe des jeux dans lesquels ni l'équipe 0, ni l'équipe 1 n'ont de stratégie gagnante.

Chapitre 15

Définitions

Un système distribué constitué de processus asynchrones interagissant les uns avec les autres et avec l'environnement peut être vu comme un simple modèle asynchrone dont les actions sont partagées en deux groupes : un groupe d'actions contrôlables (les actions du système) et un groupe d'actions incontrôlables (les actions de l'environnement). Dans le cadre des jeux, on modélise les actions par des joueurs qui sont séparés en deux équipes : l'équipe Σ_0 (les actions du système) et l'équipe Σ_1 (les actions de l'environnement). Une exécution du système en interaction avec un environnement correspond alors à une partie, une propriété de l'exécution correspond à une condition de gain et un contrôleur distribué correspond à une stratégie distribuée gagnante pour l'équipe 0.

Soit $(\Sigma, \mathcal{P}, R, W)$ une architecture (voir section 2.4). Un *jeu distribué* sur $(\Sigma, \mathcal{P}, R, W)$ est un n-uplet $G = (\Sigma_0, \Sigma_1, (Q_i)_{i\in\mathcal{P}}, (T_a)_{a\in\Sigma}, q^0, \mathcal{W})$ où Σ_0 et Σ_1 sont les joueurs de l'équipe 0 et 1 respectivement et nous avons $\Sigma = \Sigma_0 \uplus \Sigma_1$. Quel que soit le processus i dans $\mathcal{P}$, Q_i est l'ensemble des états locaux du processus i. Quel que soit le joueur a dans Σ, $T_a \subseteq Q_{R(a)} \times Q_{W(a)}$ est la description des coups locaux du joueur a. La *position initiale* du jeu est donnée par $q^0 \in Q = \prod_{i\in\mathcal{P}} Q_i$. Enfin $\mathcal{W}$ décrit la condition de gain de G.

15.1 Sémantique des jeux distribués

La manière la plus simple de décrire la sémantique d'un jeu distribué est par l'intermédiaire des jeux séquentiels. Le jeu séquentiel associé à G a pour position les éléments de Q. Sa position initiale est q^0. Il y a un coup de $p \in Q$ à $q \in Q$ par la lettre a (que l'on notera $(p \xrightarrow{a} q)$ si $(p_{R(a)}, q_{W(a)})$ est un élément de T_a et que $q_{\mathcal{P}\setminus W(a)} = p_{\mathcal{P}\setminus W(a)}$. Une partie séquentielle est alors une suite $q^0 \xrightarrow{a_1} q^1 \xrightarrow{a_2} q^2 \cdots$. On peut remarquer que pour une position $q \in Q$ de ce jeu séquentiel, plusieurs joueurs de l'équipe 0 et de l'équipe 1 peuvent avoir la possibilité de jouer. De ce fait, le graphe de ce jeu séquentiel ne correspond pas au graphe d'un jeu (séquentiel) conventionnel dans lequel chaque position est soit une position du joueur (ou de l'équipe) 0, soit une position du joueur (ou de l'équipe) 1.

Nous considérons l'alphabet $\Sigma' = \{(a, p) \mid a \in \Sigma \text{ et } p \in Q_{W(a)}\}$ avec la relation de dépendance $\mathrm{D}' = \{((a, p), (b, q)) \mid R(a) \cap W(b) \neq \emptyset\}$. La condition de gain est un ensemble de mot appartenant à $\widetilde{\Sigma}^\infty$ qui est clos par l'équivalence

usuelle sur les traces (voir section 2.3 et [17, 18]). Nous associons à la partie séquentielle $\pi = q^0 \xrightarrow{a_1} q^1 \xrightarrow{a_2} q^2 \cdots$ de G, le mot $w = (a_1, q^1_{W(a_1)})(a_2, q^2_{W(a_2)}) \cdots$ sur l'alphabet Σ'. On remarque que le mot w représente de manière équivalente la partie π et l'équipe 0 gagne si et seulement si w est dans $\mathcal{W}$.

Une sémantique plus naturelle de ces jeux distribués peut être exprimée par l'intermédiaire des traces.

On considère alors une partie distribuée comme une trace finie ou infinie sur l'alphabet de dépendance (Σ', D'). Cette trace $t = (V, \leq, \ell, \sigma)$ doit vérifiér la condition suivante : pour tout $a \in \Sigma$, et pour tout $x \in \ell^{-1}(a)$, on a $((q^0 * \mathrm{lw}(\Downarrow x))_{R(a)}, \sigma(x)) \in T_a$. La condition de gain $\mathcal{W}$ est alors un sous ensemble de $\mathbb{R}(\Sigma', \mathrm{D}')$ et l'équipe 0 gagne si et seulement si $t \in \mathcal{W}$. Les deux définitions sont équivalentes. En effet, si nous considérons une linéarisation $w = (a_1, q_1), (a_2, q_2) \cdots (a_n, q_n) \cdots$ de t, alors la partie $\pi = q_0 \xrightarrow{a_1} q_0 * \mathrm{lw}(a_1) \xrightarrow{a_2} q_0 * \mathrm{lw}(a_1 a_2) \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_0 * \mathrm{lw}(a_1 \cdots a_n) \cdots$ est une partie du jeu séquentiel associé à G. De plus w est alors le mot de Σ'^{∞} associé à la partie π.

Par exemple, si nous considérons le jeu G sur l'architecture $(\mathcal{P}, \Sigma, R, W)$ avec $\mathcal{P} = \{1, 2\}$, $\Sigma_0 = \{a\}$, $\Sigma_1 = \{b\}$ avec $R(a) = W(a) = \{1\}$ et $R(b) = W(b) = \{2\}$. On définit ensuite $Q_a = \{\bullet, 0, 1\}$, $Q_b = \{\bullet, 0, 1\}$, $q^0 = (0, 0)$, $\delta_a(0) = \{0, 1\}$, $\delta_b(0) = \{0, 1\}$ et nous choisissons une condition de gain quelconque.

Si nous considérons ensuite la partie suivante sous forme de la trace t suivante :

$$(b, 0)$$
$$(a, 0)$$

Nous pouvons lui associer deux parties séquentielles : $\pi_1 = (\bullet, \bullet) \xrightarrow{a} (0, \bullet) \xrightarrow{b} (0, 0)$ et $\pi_2 = (\bullet, \bullet) \xrightarrow{b} (\bullet, 0) \xrightarrow{a} (0, 0)$. On remarque alors que les mots w_1 (associé à π_1) et w_2 (associé à π_2) sont exactement les deux linéarisations de la trace t. Comme $\mathcal{W}$ est un ensemble clos pas l'équivalence usuelle sur les traces, t est dans $\mathcal{W}$ si et seulement si w_1 et w_2 sont dans $\mathcal{W}$.

Ces deux définitions sont donc équivalentes, mais nous utiliserons la seconde, car elle est plus appropriée aux jeux distribués et permet de définir naturellement la notion de stratégie distribuée.

Dans les jeux séquentiels, on ne considère souvent que les parties infinies. Nous allons également considérer les parties finies, car cela peut être parfois plus commode.

15.2 Les stratégies distribuées

L'étude de ces jeux distribués nécessitent que nous définissions la notion de *stratégie distribuée* pour une équipe. Une stratégie est une fonction qui définit pour chaque état du jeu le comportement d'une équipe. La stratégie est dite *gagnante* si une équipe qui la suit gagne toutes parties jouées quel que soit le comportement de l'équipe adverse. Plus formellement :

15.2.1 Stratégie à mémoire totale

Définition 11 (Stratégie distribuée causale).
Soit $G = (\Sigma_0, \Sigma_1, (Q_i)_{i \in \mathcal{P}}, (T_a)_{a \in \Sigma}, q^0, \mathcal{W})$ un jeu distribuée, une stratégie dis-

tribuée causale pour l'équipe 0 *sur* G *est une fonction* f *de* $\mathbb{M}(\Sigma', \mathrm{D}') \times \Sigma_0 \to \bigcup_{a\in\Sigma} Q_{W(a)} \uplus \{stop\}$ *qui à une partie* r *de* G *et à un élément* a *de* Σ_0*, associe un élément* q *de* $Q_{W(a)}$ *tel que, si* q *est différent de stop, alors* $r \cdot (a, q)$ *est une partie de* G*. De plus, pour tout* $r \in \mathbb{M}(\Sigma', \mathrm{D}')$*,* $f(r, a) = f(\partial_{R(a)} r, a)$*.*

Une autre caractérisation des stratégies distribuées causales est la suivante :

Lemme 31 *Soit* $G = (\Sigma_0, \Sigma_1, (Q_i)_{i\in\mathcal{P}}, (T_a)_{a\in\Sigma}, q^0, \mathcal{W})$ *un jeu distribuée, soit* f *une fonction de* $\mathbb{M}(\Sigma', \mathrm{D}') \times \Sigma_0 \to \bigcup_{a\in\Sigma} Q_{W(a)} \uplus \{stop\}$ *qui à une partie* r *de* G *et à un élément* a *de* Σ_0*, associe un élément* q *de* $Q_{W(a)}$ *tel que, si* q *est différent de stop, alors* $r \cdot (a, q)$ *est une partie de* G*. La fonction* f *est une stratégie distribuée causale si et seulement si pour tout* r *dans* $\mathbb{M}(\Sigma', \mathrm{D}')$*, pour tous* $c, d \in \Sigma$ *tels que* $c \,\mathrm{I}\, d$*, si* $f(r, d) = q$ *alors* $f(r(c, p), d) = q$ *pour tout état* p*.*

Preuve Pour montrer ce lemme, il suffit de montrer l'équivalence des deux propositions suivantes :

1. Pour tout $r \in \mathbb{M}(\Sigma', \mathrm{D}')$, pour tout $a \in \Sigma$, $f(r, a) = f(\partial_{R(a)} r, a)$.
2. Pour tout $r \in \mathbb{M}(\Sigma', \mathrm{D}')$, pour tous $c, d \in \Sigma$ tels que $c \,\mathrm{I}\, d$, si $f(r, d) = q$ alors $f(r(c, p), d) = q$ pour tout état p.

La proposition 1 entraîne la proposition 2 car $\partial_{R(d)} r(c, p) = \partial_{R(d)} r$ car $c \,\mathrm{I}\, d$, donc $f(r, d) = f(r(c, p), d) = f(\partial_{R(d)} r, d)$.

Nous montrons maintenant que la proposition 2 entraîne la proposition 1.

Soit $r \in \mathbb{M}(\Sigma', \mathrm{D}')$, r s'écrit $\partial_{R(a)} rs$ avec la propriété que pour tout sommet x dans s, $\ell(x) \,\mathrm{I}\, a$.

Nous allons donc montrer la propriété par induction sur la taille de s. Si $s = \varepsilon$, le résultat est vrai.

Sinon $s = s_0(c, q)$ avec $c \,\mathrm{I}\, a$. D'après la propriété 2, nous en déduisons que $f(r, a) = f(\partial_{R(a)} rs_0, a)$ puis par induction nous en déduisons que $f(\partial_{R(a)} rs_0, a) = f(\partial_{R(a)} r, a)$, donc finalement $f(r, a) = f(\partial_{R(a)} r, a)$. □

Étant donné un jeu G et une stratégie distribuée f pour l'équipe 0 sur le jeu G, nous dirons qu'une partie r de G est jouée selon la stratégie f (r est une *f-partie*) si pour tout préfixe $s(a, q)$ de r, $a \in \Sigma_0 \Rightarrow q = f(s, a)$. Ceci montre que chaque coup de l'équipe 0 dans la partie r a bien été effectué en accord avec la fonction f.

Étant donné un jeu G, et, de plus, une stratégie distribuée f pour l'équipe 0 sur le jeu G, nous dirons qu'une f-partie r est *f-maximale* si pour tout $a \in \Sigma_0$ tel que $\partial_{R(a)} r$ est finie, $f(r, a) = \mathrm{stop}$. Ceci signifie que l'équipe 0 a décidé que la partie était finie : aucun des joueurs de l'équipe 0 qui a la possibilité de jouer ne veut continuer à jouer. Ceci ne signifie nullement qu'aucun coup n'est possible. Si un joueur de l'équipe 1 joue, la stratégie peut alors tout à fait permettre à l'équipe 0 de continuer la partie. Ceci se rapproche du jeu de Go, ou un joueur peut décider de ne pas jouer (donc de laisser son adversaire rejouer), ce qui ne l'empêchera pas de rejouer une fois que son adversaire aura joué un nouveau coup.

Étant donné un sous-ensemble A de Σ, nous dirons qu'une f-partie r est *f_A-maximale* si pour tout $a \in A \cap \Sigma_0$ tel que $\partial_{R(a)} r$ est finie, $f(r, a) = \mathrm{stop}$.

Étant donné un jeu G et une stratégie distribuée f pour l'équipe 0 sur le jeu G, nous dirons que f est une *stratégie gagnante* si pour toute f-partie f-maximale r de G, r est une partie gagnante. Ceci signifie que toute partie jouée

selon la stratégie f et sur laquelle l'équipe 0 ne souhaite plus jouer est une partie gagnante du jeu G.

15.2.2 Stratégie à mémoire finie

Pour définir les stratégies à mémoire finie, nous commençons par définir ce que nous appellerons une *mémoire distribuée*.

Pour ce faire nous avons besoin d'utiliser des fonction asynchrones. Cette notion a été définie en [12]. Ce sont exactement les fonctions calculables par des automates asynchrones.

Définition 12 (Fonction asynchrone).
Une fonction $\mu : \mathbb{M}(\Sigma, \mathrm{D}) \to V$ où V est un ensemble fini est une fonction asynchrone si et seulement si il existe deux fonctions $f : V \times \Sigma \to V$ et $g : V \times 2^\Sigma \times V \times 2^\Sigma \to V$ telles que :

- ▷ *Pour tout $r \in \mathbb{M}(\Sigma, \mathrm{D})$, pour tout $A, B \subseteq \Sigma$, on a : $\mu(\partial_A(r) \vee \partial_B(r)) = g(\mu(\partial_A(r)), A, \mu(\partial_B(r)), B)$.*
- ▷ *Pour tout $r \in \mathbb{M}(\Sigma, \mathrm{D})$, pour tout $a \in \Sigma$, on a : $\mu(\partial_a(r)a) = f(\mu(\partial_a(r)), a)$.*

Définition 13 (Mémoire distribuée (ou causale)).
Une fonction $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ où V est un ensemble fini est une mémoire distribuée si pour tout $m \in V$, $\mu^{-1}(m)$ est un langage reconnaissable.

Cette définition s'explique à la lumière de la définition et du lemme suivant :

Définition 14 (Abstraction d'une fonction asynchrone).
Une fonction $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ est l'abstraction d'une fonction asynchrone s'il existe une fonction asynchrone $\nu : \mathbb{M}(\Sigma', \mathrm{D}') \to T$ dans un ensemble fini et une fonction $\alpha : T \to V$ telles que $\alpha \circ \nu = \mu$.

Nous avons alors :

Lemme 32 *Une fonction $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ est l'abstraction d'une fonction asynchrone si et seulement si μ est une mémoire distribuée.*

Les fonctions asynchrones (introduites dans [12]) sont exactement les fonctions calculables par des automates asynchrones.

Les jeux distribués qui nous intéressent sont joués sur des automates asynchrones. Nous déduisons donc du lemme 32 que nous pouvons enrichir le jeu de façon à calculer en parallèle de l'état du jeu, la valeur d'une mémoire distribuée, et ce en utilisant la même architecture, donc les mêmes moyens de communication. Il s'agit de plus de la plus grande famille de langage que nous pouvons ainsi calculer. Ceci explique pourquoi nous nous intéressons à ces fonctions.

Preuve de 32 Soit μ une mémoire distribuée D'après [12], quel que soit m dans V, la fonction caractéristique du langage $\mu^{-1}(m)$ s'obtient à partir d'une fonction asynchrone $\nu_m : \mathbb{M}(\Sigma', \mathrm{D}') \to T_m$ et d'une abstraction $\alpha_m : T_m \to \{0, 1\}$, telles que quel que soit r dans $\mathbb{M}(\Sigma', \mathrm{D}')$, $\alpha_m \circ \nu_m(r) = 1$ si et seulement si $\mu(r) = m$. Dans ces conditions, la fonction $\nu : \mathbb{M}(\Sigma', \mathrm{D}') \to \prod_{m \in V} \nu_m$ est une fonction asynchrone. Pour tout r dans $\mathbb{M}(\Sigma', \mathrm{D}')$, il existe un unique m dans V, tel que $\alpha_m(\nu(r)_m) = 1$. Nous définissons alors $\alpha : \nu(\mathbb{M}(\Sigma', \mathrm{D}')) \to V$ qui à $\nu(r)$ associe ce m. Dans ces conditions $\mu = \alpha \circ \nu$. Donc μ est l'abstraction d'une fonction asynchrone.

Soit maintenant μ, l'abstraction d'une fonction asynchrone ν. Dans ces conditions, il existe une fonction calculable α telle que $\mu = \alpha \circ \nu$. Dans ces conditions, $\mu^{-1}(m) = \bigcup_{n \in \alpha^{-1}(m)} \nu^{-1}(n)$. D'après [12], nous savons que $\nu^{-1}(n)$ est un langage reconnaissable, donc $\mu^{-1}(m)$ est une union finie de langages reconnaissables, donc c'est un langage reconnaissable. Donc μ est une mémoire distribuée. □

Maintenant que nous avons défini une mémoire distribuée, nous allons définir la notion de stratégie à mémoire μ.

Définition 15 (Stratégie à mémoire μ).
Soit G un jeu distribué, et f une stratégie pour l'équipe 0 sur le jeu G. Soit μ une mémoire distribuée, f est une stratégie distribuée à mémoire μ si et seulement si, pour toutes f-parties r, r' pour tout a dans Σ_0, $\mu(\partial_{R(a)} r) = \mu(\partial_{R(a)} r') \Rightarrow f(r, a) = f(r', a)$.

Cette définition assure que si deux positions du jeu atteignent le même état mémoire, la stratégie f doit jouer de la même manière sur les deux parties. De plus, comme cette mémoire est calculable sur la même architecture que le jeu lui même, si une stratégie nous est donnée, on peut construire un contrôleur à mémoire finie qui assure que toutes les exécutions du système sont des f-parties du jeu G. Si f est alors une stratégie gagnante, ce contrôleur assure que toutes les exécutions du système sont correctes.

Il existe en fait de nombreuses caractérisations d'une mémoire distribuée. En utilisant l'extension du théorème de Büchi sur les traces [52], nous obtenons le corollaire suivant :

Corollaire 33 *Une fonction $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ est une mémoire distribuée si et seulement si pour tout m dans V, il existe une formule φ_m de $\mathrm{MSO}'_\Sigma(\leq)$ telle que $\mathcal{L}(\varphi_m) = \mu^{-1}(m)$.*

De même si nous reprenons la définition d'une mémoire distribuée, nous obtenons le résultat suivant :

Proposition 34 *Soit $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ où V est un ensemble fini. Si pour toutes traces r, s, t de $\mathbb{M}(\Sigma', \mathrm{D}')$, $\mu(r) = \mu(s) \Rightarrow \mu(r \cdot t) = \mu(s \cdot t)$, alors μ est une mémoire distribuée.*

Preuve Pour chaque élément m de V, on construit un automate $\mathcal{A}_m = (V, \Sigma', \delta, \mu(\varepsilon), \{m\})$ où δ est défini de la manière suivante : $\delta(v \in V, (a, q)) = v'$ s'il existe une trace $r \in \mathbb{M}(\Sigma', \mathrm{D}')$ telle que $\mu(r) = v$ et $\mu(r(a, q)) = v'$. Cette fonction est bien définie car l'hypothèse sur μ assure que δ ne dépend pas du choix de r. Cet automate reconnaît le langage $\mu^{-1}(m)$ ce qui prouve bien que c'est un langage reconnaissable. □

Enfin si nous revenons à la définition d'une fonction asynchrone, nous obtenons la proposition suivante :

Proposition 35 *Soit μ' l'abstraction d'une fonction asynchrone et soit μ une fonction telle que :*

1. *Pour toute trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$, pour tout sous ensemble C, D de Σ tels que $r = \partial_C r \cup \partial_D r$, $\mu(r)$ ne dépend que de $\mu(\partial_C r)$, de $\mu(\partial_D r)$ et de $\mu'(r)$.*

2. *Pour toute trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$, pour tout a de Σ tel que $r = \partial_a r$, $\mu(r \cdot a)$ ne dépend que de $\mu(r)$, de a, et de $\mu'(r \cdot a)$)*

alors μ est l'abstraction d'une fonction asynchrone.

Preuve D'après [12], μ est une fonction asynchrone si et seulement si :

1. Pour toute trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$, pour tout sous ensemble C, D de Σ tels que $r = \partial_C r \cup \partial_D r$, $\mu(r)$ ne dépend que de $\mu(\partial_C r)$ et $\mu(\partial_C r)$.
2. Pour toute trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$, pour tout a de Σ tel que $r = \partial_a r$, $\mu(r \cdot a)$ ne dépend que de $\mu(r)$ et de a

Comme μ' est l'abstraction d'une fonction asynchrone, il existe ν' une fonction asynchrone et α' telles que $\mu' = \alpha' \circ \nu'$. Connaissant α', on peut calculer μ à l'aide de ν'. Nous montrons alors que $\mu \times \nu'$ est une fonction asynchrone. Considérons donc une trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$ et deux sous ensemble C, D de Σ tels que $r = \partial_C r \cup \partial_D r$. Connaissant $\mu \times \nu'(\partial_C r)$ et $\mu \times \nu'(\partial_D r)$, nous connaissons $\nu'(\partial_C r)$ et $\nu'(\partial_D r)$. Nous connaissons donc $\nu'(r)$ et donc $\mu'(r)$. Maintenant $\mu(r)$ ne dépend que de $\mu(\partial_C r)$, de $\mu(\partial_C r)$ et de $\mu'(r)$, donc nous connaissons $\mu(r)$. La preuve est semblable pour le second cas, nous en déduisons donc que $\mu \times \nu'$ est une fonction asynchrone.

On pose ensuite α comme la projection sur la première coordonnée, et on trouve $\mu = \alpha \circ (\mu \times \nu')$ ce qui prouve que μ est bien l'abstraction d'une fonction asynchrone. □

Ces différents résultats montrent que la notion de mémoire distribuée que nous avons choisie est naturelle et englobe un nombre important de modèles de calcul.

15.2.3 Stratégie sans mémoire

Nous allons maintenant définir la notion de stratégie sans mémoire.

Définition 16 (Stratégie sans mémoire).
Une stratégie distribuée f est dite sans mémoire si pour toutes traces r, r' dans $\mathbb{M}(\Sigma', \mathrm{D}')$, pour toute action a dans Σ_0, on a : $(q^0 * \mathrm{lw}(r))_{R(a)} = (q^0 * \mathrm{lw}(r'))_{R(a)} \Rightarrow f(r, a) = f(r', a)$. Dans ces conditions, on peut considérer f comme une collection de fonctions $(f_a)_{a \in \Sigma}$ avec pour tout a, $f_a : Q_{R(a)} \to Q_{W(a)}$.

De par la définition que nous avons choisie pour les mémoire distribuées, nous ne pouvons pas nous contenter de dire qu'une stratégie sans mémoire est une stratégie de mémoire $\mu : \mathbb{M}(\Sigma', \mathrm{D}') \to V$ avec $|V| = 1$. En effet, une telle stratégie ne peut même pas lire l'état courant d'une partie. Or l'état courant d'une partie r tel que peut le lire l'action a est exactement $(q^0 * \mathrm{lw}(r))_{R(a)}$ Ceci explique la définition ci-dessus.

15.3 Détermination des jeux distribués

Les jeux distribués ne sont pas déterminés. Il existe des jeux dans lesquels ni l'équipe 0, ni l'équipe 1 n'ont de stratégies gagnantes distribuées, même avec une mémoire totale. Considérons deux exemples :

L'alphabet de dépendance est $\Sigma_0 = \{a\}$, et $\Sigma_1 = \{b\}$. L'ensemble des processus est $\{1, 2\}$ et on a plus $R(a) = W(a) = \{1\}$ et $R(b) = W(b) = \{2\}$. Les ensembles d'états locaux sont $Q_0 = Q_1 = \{1\}$. Les transitions sont $T_a = T_b = \{(1, 1)\}$. L'état initial est $q^0 = (1, 1)$ et la condition de gain est $\mathcal{W} = \mathbb{M}(\Sigma', \mathrm{D}') \uplus \{(a, 1)^\omega (b, 1)^\omega\}$. Dans ces conditions, le jeu est complètement symétrique et la condition de gain également, donc si une équipe a une stratégie distribuée gagnante, il en est de même pour l'autre équipe. De façon plus précise, considérons une stratégie distribuée f pour l'action a. Cette stratégie ne peut dépendre que du nombre de a déjà joués dans la partie, et ne peut prendre que deux valeurs : 1 ou stop. Supposons que quel que soit le nombre de a, cette stratégie propose toujours 1, alors elle est perdante, car la stratégie qui consiste pour l'équipe 1 à ne pas jouer conduit à la partie $(a, 1)^\omega$ qui n'est pas dans $\mathcal{W}$. Maintenant, s'il existe un n, tel $f((a, 1)^n) = \{\text{stop}\}$, alors on considère le plus petit n vérifiant cette propriété, et la partie $(a, 1)^n (b, 1)^\omega$ est jouée selon f, mais elle n'est pas gagnante. Ceci prouve que f n'est pas une stratégie gagnante. Maintenant par symétrie, il n'existe pas non plus de stratégie distribuée gagnante pour l'équipe 1.

Le second exemple est sur le même alphabet de dépendance. On considère les ensembles d'états locaux $Q_0 = Q_1 = \{\bullet, 0, 1\}$. Les transitions sont $T_a = T_b = \{(\bullet, 0), (\bullet, 1)\}$. La condition de gain est $\{(0, 0), (1, 1), (0, \bullet), (1, \bullet)\}$. Supposons qu'il existe une stratégie gagnante distribuée sur ce jeu pour l'équipe 0. Celle-ci n'a qu'un seul choix à faire pour l'état $\bullet$. Si elle choisit stop, alors elle est perdante. Si elle choisit 0, alors la stratégie qui consiste pour l'équipe 1 à choisir 1 est gagnante, et si elle choisit 1, la stratégie qui consiste pour l'équipe 1 à choisir 0 est gagnante. Encore une fois le problème est quasiment symétrique. Nous avons à faire ici à un jeu du type "pierre, papier, ciseaux" et nous savons bien qu'il n'y a pas de stratégie gagnante. Tout au plus on peut assurer une probabilité de gagner en utilisant des stratégies probabilistes.

Cette non détermination des jeux est très similaire au problème que nous avons rencontré avec les automates alternants cellulaires et le fait que leurs complémentaires n'étaient pas leurs duaux. Cependant, pour le problème de la synthèse de contrôleur, cette non détermination ne nous pose pas de problème. Nous cherchons en effet un stratégie distribuée gagnante pour les actions contrôlables (l'équipe 0) qui permet d'assurer les bons comportements du système. Nous ne sommes pas intéressés par une stratégie gagnante *distribuée* pour les actions incontrôlables : en effet les actions incontrôlables sont gouvernées par l'environnement et nous n'avons aucune raison de nous intéresser uniquement aux environnements distribués.

15.4 Les conditions de gains

Nous avons vu que nous pouvons décrire la condition de gain comme un ensemble de $\mathbb{R}(\Sigma', \mathrm{D}')$. Cependant si nous voulons obtenir des résultats de décidabilité pour le problème de la synthèse de contrôleur, nous ne pouvons pas choisir un sous ensemble quelconque. En effet, dès que l'on se permet de prendre un ensemble rationnel pour $\mathcal{W}$, il est indécidable de savoir si l'équipe 0 a une stratégie gagnante distribuée dans le jeu G. En effet, sur $\mathbb{M}(\Sigma, \mathrm{D}) = \mathbb{M}(A, \mathrm{D}) \times \mathbb{M}(B, \mathrm{D})$, déterminer si un langage rationnel de trace $\mathcal{L}$ est égal à $\mathbb{M}(\Sigma, \mathrm{D})$ est indécidable [17]. Pour un tel langage, nous construisons un jeu sur deux processus dans

lequel l'équipe 0 a une stratégie gagnante distribuée (sans mémoire) si et seulement si $\mathcal{L} = \mathbb{M}(\Sigma, \mathrm{D})$. Nous prenons $\Sigma_0 = \emptyset$, $\Sigma_1 = A \uplus B$, $R(a) = W(a) = 1$ pour tout a dans A et $R(b) = W(b) = 1$ pour tout b dans B. Enfin nous posons $|Q| = 1$ afin de pouvoir identifier Σ et Σ' et nous prenons $\mathcal{W} = \mathcal{L} \cup (\mathbb{R}(\Sigma, \mathrm{D}) \setminus \mathbb{M}(\Sigma, \mathrm{D}))$. Toute trace de $\mathbb{R}(\Sigma, \mathrm{D})$ est alors une partie de G, et l'équipe 1 a toute liberté du choix de la partie (l'équipe 0 n'a aucun coup). Dans ces conditions, l'équipe 1 gagne dès qu'il existe une trace qui n'appartienne pas à $\mathcal{W}$, donc dès que $\mathcal{L}$ est différent de $\mathbb{M}(\Sigma, \mathrm{D})$, donc l'équipe 0 a une stratégie gagnante si et seulement si $\mathcal{L} = \mathbb{M}(\Sigma, \mathrm{D})$, ce qui prouve que le problème est indécidable.

Dans ces conditions, nous nous intéresserons à partir de maintenant aux conditions de gain reconnaissables.

15.5 Comparaison avec d'autres jeux distribués

Dans [39] un jeu distribué est un n-uplet $G = \langle P, E, T, \mathcal{W}_s, q^0 \rangle$ construit à partir de n jeux locaux $G_1, \cdots, G_n$ où $G_i = \langle P_i, E_i, T_i, q_i^0 \rangle$ avec $T_i \subseteq (P_i \times E_i)$. Les positions de l'environnement sont : $E = \prod_i E_i$. Les positions des joueurs sont : $P = \prod_i (P_i \cup E_i) \setminus E$. Les transitions des joueurs sont définis par un produit cartésien : $T_p = (\prod_i (T_i \cup \Delta_i)) \cap (P \times E)$ où $\Delta_i = \{(x_i, x_i) \mid x_i \in E_i\}$ est l'identité. Les transitions de l'environnement sont données comme un sous-ensemble T_e de $E \times P$ et finalement $T = T_p \uplus T_e$. Une partie de G commence en une position q^0 appartenant à E. Les coups de l'environnement et des joueurs alternent. Toute partie infinie est un élément de $(E \cdot P)^\omega$ et la condition d'acceptation est $\mathcal{W}_s \subseteq (E \cdot P)^\omega$.

On peut décrire de manière naturelle ces jeux à l'aide des jeux distribués que nous avons introduits. Nous montrons comment nous pouvons associer à un jeu G de [39] un de nos jeux distribués $\overline{G}$ tel que nous pouvons associer à toute partie de G une partie de $\overline{G}$ et à toute partie gagnante de G une partie gagnante de $\overline{G}$. Nous procédons ainsi : à un jeu G, nous associons le jeu $\overline{G} = (\Sigma_0, \Sigma_1, (T_a)_{a \in \Sigma}, q^0, \mathcal{W})$ de la manière suivante : l'ensemble des processus est $\mathcal{P} = \{1, \cdots, n\}$ et les états locaux sont $Q_i = E_i \uplus P_i$ pour tout processus i dans $\mathcal{P}$. L'équipe 0 est définie par $\Sigma_0 = \{1, \cdots, n\}$ avec $R(i) = W(i) = \{i\}$ pour tout i dans Σ_0. Les transitions du joueur i sont tout simplement T_i. L'équipe 1 se compose d'un unique joueur e (l'environnement) avec $R(e) = W(e) = \mathcal{P}$. Ses transitions sont T_e.

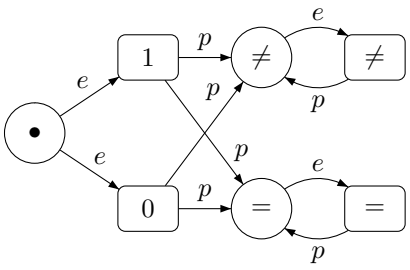
Pour définir la condition de gain, nous associons à chaque partie infinie $w = e^0 x^1 e^1 \cdots \in (E \cdot P)^\omega$ de G avec $e^0 = q^0$, une partie distribuée $\mathrm{trace}(w) \in \mathbb{R}(\Sigma', \mathrm{D}')$ qui est la borne supérieure de la suite croissante $(t_i)_{i \geq 0}$ définie inductivement par $t_0 = \varepsilon$ et $t_{n+1} = t_n \cdot (e, x^{n+1}) \cdot \prod_{i | x_i^{n+1} \in P_i} (i, e_i^{n+1})$. Puis la condition de gain est $\mathcal{W} = \mathrm{trace}(\mathcal{W}_s) \cup \{t \in \mathbb{M}(\Sigma', \mathrm{D}') \mid q^0 * \mathrm{lw}(t) \in E\}$. L'ensemble $\mathrm{trace}(\mathcal{W}_s)$ prend en compte les parties infinies et s'assurent qu'elles correspondent à des parties gagnantes du jeu G. L'ensemble $\{t \in \mathbb{M}(\Sigma', \mathrm{D}') \mid q^0 * \mathrm{lw}(t) \in E\}$ prend en compte les parties finies. Il nous faut nous assurer que les joueurs et l'environnement veulent jouer des parties infinies. Nous faisons donc gagner les joueurs quand l'environnement refuse de jouer (c'est le cas pour toutes les parties finies où l'état final est un état de l'environnement) et nous faisons gagner l'environnement quand ce sont les joueurs qui refusent de jouer.

Les jeux distribués de [39] sont donc un cas particulier de nos jeux distribués dans lesquels les joueurs de l'équipe 0 sont complètement locaux et l'environ-

nement consiste en un seul joueur global. On remarque que dans le jeu G, le partage d'information entre les joueurs ne peut passer qu'à travers les coups de l'environnement. L'environnement peut donc décider quelle information doit passer entre les joueurs.

Cependant, la différence cruciale entre les jeux de [39] et ceux présentés ici concerne la définition des stratégies. Dans [39], une stratégie f est un n-uplet de fonctions $f_i : (P_iE_i)^*P_i \to E_i$. A chaque début de partie $P_0E_0P_1E_1 \cdots P_{n-1}E_{n-1}P_n$ on associe pour chaque joueur, sa vue. Pour le joueur i, il s'agit de la fonction view_i. Cette fonction associe au début de partie $w = P_0E_0P_1E_1 \cdots P_{n-1}E_{n-1}P_n$ l'élément de $(P_iE_i)^*P_i$ obtenu en prenant la projection $\Pi_i(w)$ de w sur la coordonnée i et en l'abstrayant par la congruence générée par $p_i^3 = p_i$ pour $p_i \in E_i$. Une partie $w = P_0E_0P_1E_1 \cdots P_nE_n \cdots$ est alors jouée selon la stratégie f, si pour tout n, $E_n = \prod_{k\in[\![1..n]\!]} f_i(\text{view}_i(P_0E_0 \cdots P_n))$. De ce fait, un coup du joueur i ne dépend que de sa vue locale qui est l'historique des actions du processus i. Du fait de la fonction view_i il ne peut même pas prendre en compte le nombre de coups auxquels il n'a pas participé. Dans notre modèle la stratégie du joueur i est basée sur la vue causale de i. Comme les actions de l'environnement sont globales, la vue causale est très proche de l'état global du jeu, il ne manque que les actions immédiatement concurrentes. À partir de là, s'il existe une stratégie gagnante pour l'équipe 0 dans le jeu G, cette même stratégie est une stratégie gagnante pour l'équipe 0 dans le jeu $\overline{G}$. Cependant la réciproque est fausse. On peut trouver des jeux G non déterminés alors qu'il existe une stratégie gagnante pour l'équipe 0 dans le jeu $\overline{G}$.

Considérons par exemple $G = (G_1, G_2)$ avec $G_1 = G_2 =$



On appelle T_{ip} les transitions du joueur dans le jeu i et T_{ie} les transitions de l'environnement dans le jeu i. Les transitions de l'environnement sont alors $T_{ie} \times T_{ie}$. L'état initial est $(\bullet, \bullet)$. Les parties gagnantes sont $(\bullet, \bullet)(0,0)(=,=)^\omega$, $(\bullet, \bullet)(1,1)(=,=)^\omega$, $(\bullet, \bullet)(0,1)(\neq,\neq)^\omega$ et $(\bullet, \bullet)(1,0)(\neq,\neq)^\omega$.

Les joueurs gagnent donc la partie si au second coup, ils se mettent d'accord (jouent la même chose) et qu'ils détectent si l'environnement a joué un coup identique dans les deux jeux ou pas. Une stratégie des joueurs consiste donc en une paire de stratégies qui en fonction du coup de l'environnement (0 ou 1) doit décider du coup du joueur (= ou $\neq$). Quelle que soit la paire de stratégies des joueurs, il existe un coup de l'environnement qui fera que cette stratégie est perdante. En effet supposons que le premier joueur joue = si on lui présente un 0, alors si l'environnement commence par $(\bullet, \bullet) \to (0,1)$ la partie sera perdante pour les joueurs. Si le premier joueur joue $\neq$ quand on lui présente un 0, alors si l'environnement commence par $(\bullet, \bullet) \to (0,0)$ la partie sera de nouveau perdante pour les joueurs. Donc les joueurs n'ont pas de stratégie gagnante. Cependant, quel que soit le premier coup de l'environnement, il existe un coup des joueurs

leur permettant de gagner la partie. Nous en déduisons donc que ce jeu n'est pas déterminé.

Si nous considérons maintenant le jeu $\overline{G}$, chaque joueur verra les deux coups de l'environnement. Il pourra donc décider avec une stratégie sans mémoire si les deux coups sont identiques ou non. Dans ces conditions, l'équipe 0 a donc une stratégie gagnante distribuée.

Cependant, si un jeu G de [39] est déterminé, alors il existe une stratégie gagnante distribuée pour l'équipe 0 dans $\overline{G}$ si et seulement si les joueurs ont une stratégie gagnante dans G. En effet, si G est déterminé, soit les joueurs ont une stratégie gagnante et dans ce cas l'équipe 0 à une stratégie gagnante dans $\overline{G}$, soit l'environnement a une stratégie gagnante, et cette stratégie est également gagnante pour l'équipe 1 dans $\overline{G}$.

Si nous voulons obtenir une équivalence dans tous les cas entre les jeux G et $\overline{G}$, il nous faut limiter la mémoire utilisée par nos stratégies. Pour ce faire nous introduisons la notion de *mémoire locale*. Soit t une trace de $\mathbb{M}(\Sigma', \mathrm{D}')$, nous appellerons la i-projection de t (noté $\Pi_i(t)$) le morphisme de $\mathbb{M}(\Sigma', \mathrm{D}')$ dans Q_i^* défini par $\Pi_i((x,q)) = q_i$ si $x = e$ ou $x = i$ et $\Pi_i((x,q)) = \varepsilon$ sinon. Comme le joueur 0 ne peut connaître que les coups auxquels il participe, nous devons abstraire cette projection pour effacer le compte des coups de l'environnement. Pour ce faire nous utilisons la congruence sur Q_i^* générée par $p_i^3 = p_i$ pour $p_i \in E_i$ et nous écrivons $w_\equiv$ pour la classe d'équivalence de w dans Q_i^*. Nous dirons alors qu'une stratégie distribuée f est locale si pour tous t, t' dans $\mathbb{M}(\Sigma, \mathrm{D}')$, pour tout i dans $\mathcal{P}$, $\Pi_i(t)_\equiv = \Pi_i(t')_\equiv \Rightarrow f(t,i) = f(t',i)$. Nous obtenons alors la proposition suivante :

Proposition 36 *Les joueurs ont une stratégie gagnante dans G si et seulement si l'équipe 0 a une stratégie gagnante distribuée locale dans $\overline{G}$.*

Preuve Soit $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ une stratégie gagnante pour les joueurs dans le jeu G, nous construisons f de la manière suivante :

Pour tout $i \in \Sigma_0$, pour tout $t \in \mathbb{M}(\Sigma', \mathrm{D}')$, $f(t,i) = \sigma_i(\Pi_i(\partial_i t)_\equiv)$.

Par définition, cette stratégie est locale et distribuée. Nous devons montrer que cette stratégie est gagnante pour l'équipe 0.

Considérons une trace t de $\mathbb{R}(\Sigma', \mathrm{D}')$ une f-partie f-maximale du jeu $\overline{G}$.

Si la partie t est une partie finie, alors les joueurs ne peuvent pas jouer sur la partie t. En effet, la stratégie f ne propose jamais aux joueurs de s'arrêter. Dans ces conditions, nous en déduisons que $q^0 * \mathrm{lw}(t)$ est un élément de E, donc t est une partie gagnante.

Maintenant, si t est une partie infinie, nous allons tout d'abord construire une partie de G.

La partie t peut s'écrire de la manière suivante : $t = (e, q^1)s^1(e, q^2)s^2 \cdots$, où s^m est une trace non vide contenant au plus un coup du joueur i. Nous pouvons alors construire une partie w de G de la manière suivante :

$w = q^0 q^1 q'^1 q^2 q'^2 \cdots$

où nous avons $q^m = (x_1^m, \ldots, x_n^m)$ et $q'^m = (x_1'^m, \cdots, x_n'^m)$ avec $x_i'^m = x_i^m$ si s^m ne contient pas de coup du joueur i et $x_i'^m = e_i$ si (i, e_i) est dans s^m.

Par construction de $\overline{G}$, w est une partie de G. Nous prouvons maintenant que w est jouée selon la stratégie σ. Pour se faire, il suffit de montrer que pour tout m, $q^m \to q'^m$ est un coup joué selon la stratégie σ. C'est à dire que pour

tout i tel que $x_i \neq x'_i$, $x'_i = \sigma_i(\text{view}_i(q^0q^1q'^1 \dots q'^{m-1}q^m))$. Cependant, nous avons :

$$\begin{aligned} x'_i &= f((e,q^1)s^1(e,q^2)s^2 \cdots (e,q^m),i) \\ &= \sigma_i(\Pi_i((e,q^1)s^1(e,q^2)s^2 \cdots (e,q^m))_{\equiv}) \end{aligned}$$

et par construction :

$\Pi_i((e,q^1)s^1(e,q^2)s^2 \cdots (e,q^m))_{\equiv} = \text{view}_i(q^0q^1q'^1 \dots q'^{m-1}q^m)$

Nous en déduisons que w est une σ-partie. Donc c'est une partie gagnante, donc w appartient à W_s. Mais nous avons $t = \text{trace}(w)$, nous en déduisons donc que t est dans $\mathcal{W}$, donc la partie t est une partie gagnante pour l'équipe 0.

Soit f une stratégie gagnante distribuée locale de $\overline{G}$ pour l'équipe 0. Nous construisons $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ de la manière suivante :

$\sigma_i(view_i(q^0q^1q'^1q^2q'^2 \cdots q^m)) = f(\text{trace}(q^0q^1q'^1q^2q'^2 \cdots q^m), i)$.

Nous devons montrer que σ_i est bien définie. Pour cela nous devons prouver que si :

$\text{view}_i(q^0q^1q'^1q^2q'^2 \cdots q^m) = \text{view}_i(r^0r^1r'^1r^2r'^2 \cdots r^{m'})$

alors nous avons :

$f(\text{trace}(q^0q^1q'^1q^2q'^2 \cdots q^m), i) = f(\text{trace}(r^0r^1r'^1r^2r'^2 \cdots r^{m'}), i)$

Mais comme $\Pi_i(\text{trace}(q^0q^1q'^1q^2q'^2 \cdots q^m))_{\equiv} = \text{view}_i(q^0q^1q'^1q^2q'^2 \cdots q^m)$ et que f est une stratégie locale, le résultat est vrai.

Nous considérons maintenant une σ-partie w de G. Nous devons montrer que w est une partie gagnante pour les joueurs. Nous montrons tout d'abord que $\text{trace}(w)$ est une partie jouée selon la stratégie f. Pour ce faire, il suffit de montrer que pour tout préfixe de $\text{trace}(w)$ de la forme $t_1(e,q)(i,q')$, nous avons $f(t_1(e,q),i) = q'$. Cependant $f(t_1(e,q),i) = \sigma_i(\Pi_i(t_1(e,q))_{\equiv})$ et par définition de w et $\text{trace}(w)$, $\sigma_i(\Pi_i(t_1(e,q))_{\equiv}) = q'$.

Nous devons maintenant montrer que $w \in \mathcal{W}_s$, mais $\text{trace}(w) \in \mathcal{W}$, et trace est une fonction injective sur son ensemble de définition. En effet, nous pouvons reconstruire w à partir de $\text{trace}(w)$ comme nous l'avons fait dans la première partie de cette preuve, donc $\text{trace}(w) \in \text{trace}(\mathcal{W}_s)$, et donc $w \in \mathcal{W}_s$. Ceci prouve que w est une partie gagnante pour les joueurs.

Finalement, nous avons montré que les deux jeux sont équivalents.

□

Nous allons cependant prouver dans le chapitre 17 que pour nos jeux distribués, pour certains alphabets, il est possible de décider si l'équipe 0 a une stratégie gagnante.

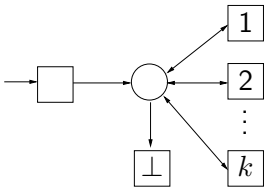
Ce résultat nous pousse à regarder les résultats de [39]. En effet, nous avons vu que nous pouvions simuler les jeux de [39] par nos jeux distribués. Cependant décider de l'existence d'une stratégie distribuée pour les joueurs dans ces jeux est indécidable. Pourtant l'architecture que nous utilisons pour les simuler est une architecture série parallèle sur laquelle nous pouvons décider de l'existence d'une stratégie gagnante sur nos jeux. La différence est donc bien dans la mémoire.

La preuve de l'indécidabilité des jeux de [39] peut se montrer ([6]) à l'aide du problème de la correspondance de Post. On se donne donc un entier m et deux suites de mots sur l'alphabet $\{a,b\}$ $(u_i)_{1\leq i\leq m}$ et $(v_i)_{1\leq i\leq m}$ et on cherche à savoir s'il existe une suite finie $i_1, \dots, i_n$ d'entiers dans l'intervalle $[\![1,n]\!]$ telle que $u_{i_1}u_{i_2}\cdots u_{i_n} = v_{i_1}v_{i_2}\cdots v_{i_n}$. Ce problème est indécidable.

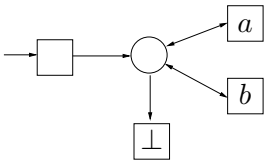
On considère alors le jeu composé des trois arènes suivantes :

Pendant une partie, les joueurs 1 et 2 ne reçoivent jamais d'informations de l'environnement. Le joueur 1 doit générer une suite d'indice et s'arrêter quand

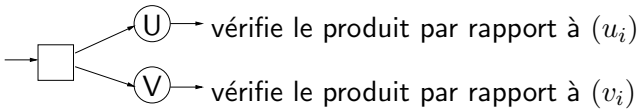
Joueur 1 : génère les indices



Joueur 2 : génère les lettres



Joueur 3 : reçoit la suite des indices et des lettres par l'environnement et vérifie que le produit est correct par rapport à (u_i) ou (v_i).



il le souhaite en choisissant le symbole $\perp$. Le joueur 2 doit générer un mot et s'arrêter quand il le souhaite en choisissant le symbole $\perp$. Au préalable l'environnement aura choisi dans l'arène numéro 3 s'il souhaite effectuer une vérification par rapport à la suite (u_i) ou à la suite (v_i) et ensuite au fur et à mesure que les joueurs 1 et 2 choisissent les indices et les lettres, il fournira les informations au joueur 3 qui est un joueur qui n'a aucune décision à prendre : il se contente de vérifier que les joueurs 1 et 2 choisissent bien de manière cohérente le mot et les indices. Pour gagner, les joueurs doivent donc choisir le mot et les indices de manière indépendante du choix de l'environnement, donc une stratégie gagnante de ce jeu est une solution du problème de Post, et on en déduit bien qu'il existe une stratégie gagnante pour les joueurs si et seulement si le problème de Post à une solution. Ce problème est donc également indécidable.

Cependant, si nous regardons maintenant la traduction de ce jeu dans notre cadre, la mémoire causale permet aux joueurs 1 et 2 de connaître le choix de l'environnement sur la troisième arène. Ils peuvent donc jouer différemment si l'environnement a choisi d'effectuer sa vérification sur la suite (u_i) ou sur la suite (v_i). Il suffit donc au joueur 1 de toujours jouer 1 puis $\perp$ et au joueur 2 de jouer le mot u_1 si le joueur 3 est en train de vérifier le calcul par rapport aux (u_i) et de jouer v_1 si le joueur 3 est en train de vérifier le calcul par rapport aux (v_i). Cette stratégie est donc une stratégie gagnante pour l'équipe 0.

Cet exemple nous montre que la mémoire est le point clé qui rend nos jeux distribués décidables.

Dans le problème de contrôle distribué présenté en [37] les coups de l'environnement sont locaux et les transitions d'une action contrôlable sont définies par le produit cartésien de fonctions de transitions locales. De nouveau nos jeux distribués peuvent modéliser de façon directe ces systèmes. Comme des stratégies locales sont utilisées en [37] la proposition 36 est encore valable.

Chapitre 16

Les stratégies sans mémoire

16.1 Intérêt des stratégies sans mémoire

La proposition suivante nous permet d'étudier les stratégies sans mémoire à la place des stratégies dont la mémoire nous est donnée.

Proposition 37 *Soit G un jeu distribué et soit μ une mémoire distribuée sur G. Nous pouvons alors construire un nouveau jeu distribué G^μ tel qu'il existe une stratégie distribuée gagnante à mémoire μ sur G si et seulement s'il existe une stratégie distribuée sans mémoire sur G^μ. De plus, si G est un jeu fini, alors G^μ l'est également.*

Preuve Nous commençons par construire G^μ. Le jeu G est : $G = (\Sigma_0, \Sigma_1, (Q_i)_{i\in\mathcal{P}}, (T_a)_{a\in\Sigma}, q^0, \mathcal{W})$. La mémoire μ s'écrit $\alpha \circ \nu$ où ν est une fonction asynchrone. En particulier, s'il existe une stratégie gagnante à mémoire μ dans G, il existe une stratégie gagnante à mémoire ν dans G. Dans la suite, nous pouvons donc supposer que μ est une fonction asynchrone. On pose $M = \mu(\mathbb{M}(\Sigma', \mathrm{D}'))$, $m_0 = \prod_{i\in\mathcal{P}} \mu(\varepsilon)$ et on pose alors $G^\mu = (\Sigma_0, \Sigma_1, (Q_i \times M)_{i\in\mathcal{P}}, (T^\mu_a)_{a\in\Sigma}, (q^0 \times m_0), \mathcal{W}^\mu)$.

Il nous faut définir $(T^\mu_a)_{a\in\Sigma}$ et $\mathcal{W}^\mu$. Tout d'abord une trace t appartient à $\mathcal{W}^\mu$, si la trace obtenue en ne conservant que la première composante des états locaux de t est dans $\mathcal{W}$.

Nous introduisons alors la notation $m \cdot (a, q)$. Soit $(a, q) \in \Sigma'$, et $m = (m_i)_{i\in R(a)} \in M^{R(a)}$. S'il existe une trace $t = \partial_{R(a)} t \in \mathbb{M}(\Sigma', \mathrm{D}')$ telle que, pour tout i dans $R(a)$, $\mu(\partial_i t) = m_i$, nous posons $m \cdot (a, q) = \mu(t(a, q))^{W(a)}$. S'il n'existe pas de tel t, alors $m \cdot (a, q)$ est indéfini. Ceci défini bien $m \cdot (a, q)$ car μ est une fonction asynchrone, donc $m \cdot (a, q)$ ne dépend pas du choix particulier de t dès que $\mu(\partial_i t) = m_i$ quel que soit i dans $R(a)$.

Nous avons maintenant $((p, m), (p', m')) \in T^\mu_a$ si et seulement si $(p, p') \in T_a$ et $m' = m \cdot (a, p')$.

Nous allons maintenant montrer que l'existence d'une stratégie gagnante distribuée f dans G implique l'existence d'une stratégie gagnante distribuée sans mémoire f' dans G^μ. Pour décrire une telle stratégie, il nous suffit, pour tout a dans Σ_0, de restreindre la fonction de transition T^μ_a tel que pour tout (p, m), il existe au plus un élément $((p, m), (p', m'))$ dans T^μ_a. Pour ce faire, on considère une partie t de G telle que $t = \partial_{R(a)} t$, $q^0 * \mathrm{lw}(t)_{R(a)} = p$, $\mu(t) = m$. Si

un tel t n'existe pas, alors il n'existe pas d'élément du type $((p,m),(p',m'))$ dans T_a^μ. Sinon la valeur $f(t,a)$ ne dépend pas du choix particulier de t, car f est une stratégie à mémoire μ. Si $f(t,a) \neq \text{stop}$, nous imposons $((p,m),(p',m')) \in T_a^\mu$ avec $p' = f(t,a)$ et $m' = m \cdot (a,p')$. Ceci définit bien une stratégie distribuée sans mémoire, il reste à montrer que c'est une stratégie gagnante. Pour montrer que ce résultat est vrai, nous considérons une f'-partie f'-maximale t' de G'. Nous considérons alors la trace t de $\mathbb{R}(\Sigma', \mathrm{D}')$ obtenue en ne conservant que la première composante des états locaux de t'. Il suffit alors de montrer que t est une f-partie f-maximale de G. Ce résultat est vrai par construction, car quelque soit le sommet x de t', nous avons $\sigma(x) = (p,m)$ avec $m = \mu(\downarrow x)$ et les coups sur t' sont alors joués en fonction des coups sur t.

Supposons maintenant l'existence d'une stratégie distribuée sans mémoire gagnante sur G^μ. Cette stratégie nous donne une stratégie à mémoire μ sur G. En effet considérons t, nous devons déterminer la valeur de $f(t,a)$. Soit $p = (q^0 * \mathrm{lw}(t))_{R(a)}$ et $m = \mu(t)_{R(a)}$ Si la stratégie sans mémoire sur G^μ nous donne un coup $(p,m) \xrightarrow{a} (p',m')$, on pose $f(t,a) = p'$, sinon on pose $f(t,a) = \text{stop}$. Tout d'abord, cette stratégie est bien à mémoire μ, car quels que soient $t, t' \in \mathbb{M}(\Sigma', \mathrm{D}')$, quel que soit $a \in \Sigma_0$, $\mu(\partial_{R(a)} t) = \mu(\partial_{R(a)} t') \Rightarrow f(t,a) = f(t',a)$. Ensuite cette stratégie est gagnante car si on considère une f-partie f-maximale t, nous pouvons construire une f'-partie t', car tous les coups des joueurs dans t nous donnent un coup des joueurs dans t' par construction de f. Cette partie est alors f'-maximale, donc gagnante. Par définition de $\mathcal{W}^\mu$, nous en déduisons que t est dans $\mathcal{W}$, donc t est une partie gagnante. □

16.2 Jeu global

De façon à pouvoir utiliser les résultats usuels de théorie des jeux, nous souhaitons définir un jeu classique à deux joueurs $\widetilde{G} = (Z,T)$ tel que l'équipe 0 a une stratégie distribuée gagnante sans mémoire dans le jeu distribué G si et seulement si le joueur 0 a une stratégie gagnante sans mémoire sur le jeu $\widetilde{G}$.

Les positions du jeu global $\widetilde{G}$ sont $Z = Z_0 \cup Z_1$ où $Z_0 = Q \times \Sigma_0$ sont les positions du joueur 0 et $Z_1 = Q \times (\Sigma_1 \cup \{0,1,2\})$ sont les positions du joueur 1. La position initiale est $(q^0, 0) \in Z_1$. Sur une position (q,a), la première composante décrit l'état global actuel de la partie et la seconde composante est utilisée pour déterminer le joueur qui doit jouer et quelle action doit être exécutée. L'ensemble T des transitions est le sous ensemble de $Z \times Z$ suivant :

- ▷ $(p,b) \to (p,a)$ avec $b \in \{0,1,2\}$ et $a \in \Sigma$. Ce coup signifie que le joueur 1 décide que le prochain coup devra être un *a-coup* (un coup joué avec la lettre a). Dans ce jeu global, c'est le joueur 1 qui décide quelles actions auront lieu et dans quel ordre elles devront être jouées. Cela lui permet de tester toutes les linéarisations d'une partie distribuée.
- ▷ $(p,a) \to (q,1)$ avec $a \in \Sigma$, $(p_{R(a)}, q_{W(a)}) \in T_a$ et $q_{\mathcal{P} \setminus W(a)} = p_{\mathcal{P} \setminus W(a)}$. C'est un a-coup exécuté par le joueur 0 ou le joueur 1 selon que a est dans Σ_0 ou Σ_1.
- ▷ $(p,a) \to (p,2)$ avec a dans Σ_0. Le joueur 0 *refuse* de faire un a-coup.
- ▷ $(q,b) \to (q^0,0)$ avec $b \in \{0,1,2\}$. Ce sont des coups de *remise à zéro*. Ils sont utilisés par le joueur 1 pour montrer que le joueur 0 ne joue pas selon une stratégie *distribuée*.

On remarque que le joueur 1 peut effectuer plusieurs coups de suite.

Une partie globale est une suite finie ou infinie $z = z_0 z_1 z_2 \cdots \in Z^\infty$ débutant par la position initiale $z_0 = (q^0, 0)$ et telle que $z_n \to z_{n+1}$ est un coup quel que soit $n \in \mathbb{N}$. Soit $z = z_0 z_1 z_2 \cdots \in Z^\infty$ une partie globale et soit $z_n = (q^n, a_n) \in Z$ pour tout n dans $\mathbb{N}$. On définit par induction la suite $(t_n)_{n\geq 0} \in \mathbb{M}(\Sigma', \mathrm{D}')$ associée à z. Si $z_n = (q^0, 0)$, alors $t_n = \varepsilon$ même si n est supérieur à 0. Si $a_{n+1} \in \Sigma \cup \{2\}$, alors $t_{n+1} = t_n$. Enfin si $a_n = a \in \Sigma$ et $a_{n+1} = 1$, alors $t_{n+1} = t_n(a, q^{n+1}_{W(a)})$. Nous prouvons par induction que t_n est une partie distribuée et que $q^0 * \mathrm{lw}(t_n) = q^n$ quel que soit n dans $\mathbb{N}$. Le seul cas non trivial est pour $a_n = a \in \Sigma$ et $a_{n+1} = 1$. Par induction t_n est une partie distribuée et $q^0 * \mathrm{lw}(t_n) = q^n$. Nous avons $(q^n_{R(a)}, q^{n+1}_{W(a)}) \in T_a$ et $q^n_{R(a)} = (q^0 * \mathrm{lw}(\partial_{R(a)} t_n))_{R(a)}$. Dans ces conditions, t_{n+1} est une partie distribuée, et en utilisant $q^{n+1}_{\mathcal{P}\setminus W(a)} = q^n_{\mathcal{P}\setminus W(a)}$ nous obtenons $q^0 * \mathrm{lw}(t_{n+1}) = q^{n+1}$.

La partie globale z est *consistante* si pour tout j, k dans $\mathbb{N}$, avec $a_j = a_k = a \in \Sigma_0$, et $q^j_{R(a)} = q^k_{R(a)}$, nous avons $a_{j+1} = a_{k+1}$ et $q^{j+1}_{W(a)} = q^{k+1}_{W(a)}$. Cela signifie que lorsque les joueurs voient les mêmes états locaux, ils jouent de la même façon. Une partie est donc consistante si les joueurs jouent avec une stratégie sans mémoire.

La partie globale z est *juste* si $\{n \in \mathbb{N} \mid a_n = 0\}$ est un ensemble fini et que pour tout $a \in \Sigma_0$, soit l'ensemble $\{n \in \mathbb{N} \mid a_n = a\}$ est infini, soit $\partial_{R(a)} t(z)$ est infinie. Une partie est juste quand l'environnement ne joue pas une infinité de remise à zéro et quand il n'empêche pas les joueurs de jouer une action qui reste toujours possible.

Si z est à la fois consistante et juste, nous définissons $N(z) = \max\{n \in \mathbb{N} \mid a_n = 0\}$. Dans ces conditions, la suite $(t_n)_{n\geq N(z)}$ est une suite croissante et admet donc une borne supérieure $t(z)$ qui est une partie distribuée de G.

La condition de gain $\widetilde{\mathcal{W}}$ de $\widetilde{G}$ ne considère que les parties infinies $z \in Z^\omega$. Si z n'est pas consistante, alors la joueur 0 perd la partie car ceci prouve qu'il n'imite pas une stratégie *distribuée* sans mémoire. Si z n'est pas juste, alors le joueur 1 perd la partie. Finalement, si z est à la fois consistante et juste, alors le joueur 0 gagne la partie si et seulement si $t(z)$ est dans $\mathcal{W}$.

Une *stratégie* (globale) (S) pour le joueur 0 dans $\widetilde{G}$ est une fonction $g : Z^* Z_0 \to Z_1$ telle que $g(z(p,a)) = (q,b)$ implique que le coup $(p,a) \to (q,b)$ est un coup de $\widetilde{G}$. Une partie globale $z = z_0 z_1 z_2 \cdots \in Z^\infty$ est *jouée selon la stratégie* g (g-partie) si chaque coup du joueur 0 est jouée selon la stratégie g : $z_k \in Z_0$ implique $z_{k+1} = g(z_0 \cdots z_k)$. Si le joueur 0 gagne toutes les g-parties, alors g est une stratégie gagnante pour le joueur 0. Une stratégie g est *sans mémoire*, si pour tout x, x' dans Z^* et tout y dans Z_0, nous avons $g(xy) = g(x'y)$.

Nous pouvons maintenant énoncer le principal résultat de cette section :

Théorème 38 *Pour un jeu distribué G, les propositions suivantes sont équivalentes :*

1. *Il existe une stratégie distribuée sans mémoire gagnante pour l'équipe* 0 *dans le jeu distribué G.*
2. *Il existe une stratégie sans mémoire gagnante pour le joueur* 0 *dans le jeu global $\widetilde{G}$.*
3. *Il existe une stratégie gagnante pour le joueur* 0 *dans le jeu globale $\widetilde{G}$.*

La preuve de ce théorème est divisée entre la propositions 39 ($3 \Rightarrow 2$), la proposition 40 ($1 \Rightarrow 2$) et la proposition 42 ($2 \Rightarrow 1$). L'implication ($2 \Rightarrow 3$) est évidente.

Proposition 39 *Si le joueur* 0 *a une stratégie gagnante dans* $\widetilde{G} = (Z, T)$*, il a une stratégie gagnante sans mémoire dans ce jeu.*

Preuve L'argument qui va nous permettre de prouver cette proposition est le fait que toute partie gagnante pour les joueurs doit être consistante. Cependant, ceci ne signifie pas que toute stratégie gagnante est sans mémoire.

Considérons l'alphabet de dépendance $\Sigma = \{a, b\}$ avec $R(a) = W(a) = \{a\}$ et $R(b) = W(b) = \{b\}$. Nous considérons alors $\Sigma_0 = \{a\}$, $\Sigma_1 = \{b\}$ et le jeu $G = (\Sigma_0, \Sigma_1, (Q_i)_{i\in\mathcal{P}}, (T_a)_{a\in\Sigma}, q^0, \mathcal{W})$ avec $Q_a = \{0, 1\}$, $Q_b = \{0, 1\}$, $T_a = Q_a \times Q_a$, $T_b = Q_b \times Q_b$, $q^0 = \{0, 0\}$ et $\mathcal{W} = \mathbb{R}(\Sigma', \mathrm{D}')$. Dans ces conditions, la stratégie g du jeu $\widetilde{G}$ défini ainsi : $g((p_0, a_0)\cdots(p_n, a_n)) = (p_n, 0)$ si $a_0 = b$ et $g((p_0, a_0)\cdots(p_n, a_n)) = (p_n, 1)$ sinon est une stratégie gagnante car toutes les parties sont consistantes. En revanche cette stratégie n'est pas sans mémoire. Pour construire une stratégie gagnante, nous allons profiter des coups de remise à zéro pour jouer une partie de g qui contiendra tous les coups possibles pour le joueur 0. À partir de cette partie nous pourrons alors construire une stratégie sans mémoire et nous vérifierons qu'elle est bien gagnante.

Nous écrirons $u \rightarrow_1 w$ si (u, w) est un élément de $T \cap (Z_1 \times Z)$, c'est à dire un coup du joueur 1. Nous définissons $\xrightarrow{*}_1$ comme la clôture réflexive et transitive de $\rightarrow_1$. Nous définissons la 1-clôture d'un graphe $(Y, S) \subseteq \widetilde{G}$ par $\mathsf{cl}_1(Y, S) = (Y_1, S_1)$ avec $Y_1 = \{v \in Z \mid \exists v' \in Y,\ v' \xrightarrow{*}_1 v\}$ et $S_1 = S \cup \{(v, w) \in Y_1 \times Y_1 \mid v \rightarrow_1 w\}$.

Soit g une stratégie gagnante pour le joueur 0 dans $\widetilde{G} = (Z, T)$. Nous construisons par induction une suite croissante x^n de g-parties de $\widetilde{G}$ et de graphes $\widetilde{G}_n = (Y_n, S_n) \subseteq \widetilde{G}$. Nous posons $x^0 = (q^0, 0)$ et $\widetilde{G}_0 = \mathsf{cl}_1\big(\{(q^0, 0)\}, \emptyset\big)$. Supposons que x^n et $\widetilde{G}_n = \mathsf{cl}_1(\widetilde{G}_n)$ ont déjà été construits et satisfont les invariants suivants :

(*i*) Toute position de Y_n est accessible depuis $(q^0, 0)$ dans $\widetilde{G}_n$.

(*ii*) Tous les coups du joueur 0 dans $\widetilde{G}_n$ apparaissent dans x^n, c'est à dire si $(y, y') \in S_n$ et $y \in Z_0$, alors nous avons $x^n = x'yy'x''$ pour certains $x', x'' \in Z^*$.

(*iii*) x^n est une g-partie et un chemin dans $\widetilde{G}_n$ finissant en $(q^0, 0)$.

Si $(\{z\} \times Z) \cap S_n \neq \emptyset$ pour toutes les positions $z \in Y_n \cap Z_0$ du joueur 0, alors la construction s'arrête et nous posons $\overline{G} = (Y, S) = \widetilde{G}_n$ et $x = x^n$. Dans le cas contraire, nous choisissons z dans $Y_n \cap Z_0$ tel que $(\{z\} \times Z) \cap S_n = \emptyset$ et nous choisissons par (i) un chemin $(q^0, 0)yz$ dans $\widetilde{G}_n$. Nous posons $x^{n+1} = x^n yz \cdot g(x^n yz) \cdot (q^0, 0)$ et $\widetilde{G}_{n+1} = \mathsf{cl}_1(Y_n \cup \{g(x^n yz)\}, S_n \cup \{(z, g(x^n yz))\})$.

Nous vérifions maintenant les invariants. Il est clair que la condition (i) est vérifiée. La seule nouvelle transition du joueur 0 dans $\widetilde{G}_{n+1}$ est $(z, g(x^n yz))$, or cette transition fait partie de x^{n+1}, donc la condition (ii) est également vérifiée. Par définition, x^{n+1} est un chemin de $\widetilde{G}_{n+1}$ finissant en $(q^0, 0)$. Pour montrer que $x^{n+1} = x^n yz \cdot g(x^n yz) \cdot (q^0, 0)$ est une g-partie, il suffit de montrer que les coups de y sont joués selon la stratégie g. Supposons donc que $y = y'vwy''$ avec $v \in Z_0$, $w \in Z_1$ et supposons que $x^n y'$ est une g-partie. Dans ces conditions, $x^n y'v \cdot$

$g(x^n y'v)$ est une g-partie et elle peut être étendue en une g-partie infinie. Comme g est une stratégie gagnante, la g-partie $x^n y'v \cdot g(x^n y'v)$ doit être consistante. D'après la propriété (ii) nous pouvons écrire $x^n = x'vwx''$. Nous en déduisons que $g(x^n y'v) = w$ par consistance. Donc le coup (v, w) dans y est joué selon g. Nous en déduisons donc que x^{n+1} est une g-partie, donc que la propriété (iii) est vérifiée.

On remarque que pour tout y dans $Y \cap Z_0$, il existe un unique $g'(y) \in Y$ tel que $(y, g'(y)) \in S$. L'existence est une conséquence de la condition d'arrêt de la construction. L'unicité utilise la propriété (ii), en effet chaque coup du joueur 0 dans $\overline{G}$ apparaît dans x qui est consistante car c'est une g-partie. On étend alors g' en une stratégie sans mémoire de la manière suivante : Pour x dans Z^* et $y = (p, a) \in Z_0$, nous posons $g'(xy) = g'(y)$ si y est un élément de Y et $g'(x, y) = (p, 2)$ dans le cas contraire.

Nous prouvons maintenant que g' est une stratégie gagnante sans mémoire pour le joueur 0 dans $\widetilde{G}$. Soit z un élément de Z^ω une g'-partie. Comme g' est *sans mémoire*, z est consistante. En utilisant la définition de g' et comme $\mathsf{cl}_1(\overline{G}) = \overline{G}$, nous en déduisons que z est un chemin de $\overline{G}$. Nous montrons que xz est une g-partie. Supposons que ce ne soit pas le cas, et considérons (v, w) avec v dans Z_0 comme le premier coup de xz qui n'est pas joué selon g. Comme x est une g-partie, ce coup est forcément un coup joué dans z. Donc $z = z'vwz''$ et $xz'v$ est une g-partie. Comme z est un chemin de $\overline{G}$, nous avons $(v, w) \in S$. D'après la propriété (ii) nous pouvons écrire $x = x'vwx''$. Comme $x'vwx''z'v$ est une g-partie, son extension par $g(x'vwx''z'v)$ doit être consistante. Donc $g(x'vwx''z'v) = w$ ce qui est une contradiction. Nous en déduisons donc que xz est une g-partie et donc une partie gagnante pour le joueur 0 car g est une stratégie gagnante. Si xz n'est pas juste, alors z n'est pas juste et donc gagnante pour le joueur 0. Si xz est juste, alors nous avons $t(xz) = t(z)$ car x se termine par $(q^0, 0)$. Nous en déduisons donc que $t(z) = t(xz) \in \mathcal{W}$, donc z est une partie gagnante.

□

Proposition 40 *Soit f une stratégie distribuée gagnante sans mémoire pour l'équipe 0 dans le jeu G. Pour (p, a) dans Z_0, nous définissons $g((p, a)) = (p, 2)$ si $f(p_{R(a)}, a) = stop$ et $g((p, a)) = (q, 1)$ avec $q_{\mathcal{P}\setminus W(a)} = p_{\mathcal{P}\setminus W(a)}$ et $f(p_{R(a)}, a) = q_{W(a)}$ dans le cas contraire. Alors g est une stratégie gagnante sans mémoire pour le joueur 0 dans le jeu global $\widetilde{G}$.*

Preuve Considérons une g-partie $z = z_0 z_1 z_2 \cdots \in Z^\omega$. Nous alors tout d'abord montrer que z est consistante. Soit $j, k \geq 0$ avec $a_j = a_k = a \in \Sigma_0$ et $q^j_{R(a)} = q^k_{R(a)}$. Nous avons alors $f(q^j_{R(a)}, a) = f(q^k_{R(a)}, a)$. Comme z est une g-partie, nous en déduisons par définissions de g que soit $a_{j+1} = a_{k+1} = 2$ soit $a_{j+1} = a_{k+1} = 1$ et $q^{j+1}_{W(a)} = q^{k+1}_{W(a)}$.

Si la partie n'est pas juste, c'est une partie gagnante pour le joueur 0. Nous supposons donc à partir de maintenant que la partie z est juste. Nous montrons que $t(z)$ est une f-partie f-maximale. Comme f est une stratégie distribuée gagnante, nous en déduisons que $t(z)$ appartient alors à $\mathcal{W}$ et le joueur 0 gagne donc la partie globale z ce qui termine la preuve.

Soit x un sommet de $t(z)$ avec $a = \ell(x) \in \Sigma_0$. Soit $j > N(z)$ l'entier minimal tel que $\downarrow x \leq t_{j+1}$. De façon nécessaire, nous avons $a_j = a$, $\partial_{R(a)} t_j = \Downarrow x$,

$a_{j+1} = 1$ et $\sigma(x) = q^{j+1}_{W(a)}$. Nous en déduisons que $(q^0 * \mathrm{lw}(\Downarrow x))_{R(a)} = (q^0 * \mathrm{lw}(t_j))_{R(a)} = q^j_{R(a)}$. Comme z est une g-partie, nous avons $f(q^j_{R(a)}, a) = q^{j+1}_{W(a)}$. Nous en déduisons donc que $\sigma(x) \in f((q^0 * \mathrm{lw}(\Downarrow x))_{R(a)}, a)$ et $t(z)$ est donc une f-partie. Il reste à vérifier que $t(z)$ est f-maximale. Considérons a dans Σ_0 tel que $\partial_{R(a)}t(z)$ est finie. Comme z est juste, il existe $k > N(z)$ avec $a_k = a$ et $\partial_{R(a)}t_k = \partial_{R(a)}t(z)$. Comme $t_{k+1} \leq t(z)$, on en déduit que $a_{k+1} = 2$ (le joueur 0 a refusé le a-coup). Nous avons alors $(q^0 * \mathrm{lw}(\partial_{R(a)}t(z)))_{R(a)} = (q^0 * \mathrm{lw}(\partial_{R(a)}t_k))_{R(a)} = q^k_{R(a)}$. Comme z est une g-partie, nous en déduisons que $f((q^0 * \mathrm{lw}(\partial_{R(a)}t(z)))_{R(a)}, a) = f(q^k_{R(a)}, a) = \mathrm{stop}$. □

Pour prouver l'implication $(2 \Rightarrow 1)$ du théorème 38, nous utilisons les coups de remise à zéro.

Lemme 41 *Soit g une stratégie gagnante sans mémoire pour le joueur G dans le jeu global $\widetilde{G}$. Soit $(p^1, a) \in Z_0$ et $(p^2, a) \in Z_0$ deux positions accessibles par des g-parties telles que $p^1_{R(a)} = p^2_{R(a)}$. Alors, $g((p^1, a)) = (p^1, 2)$ si et seulement si $g((p^2, a)) = (p^2, a)$ et si $g((p^1, a)) = (q^1, 1)$ et $g((p^2, a)) = (q^2, 1)$, alors $q^1_{W(a)} = q^2_{W(a)}$.*

Preuve Soit $x = x_0x_1x_2\cdots \in Z^\omega$ une g-partie et soit $k > 0$ tel que $x_k = (p^1, a)$. Soit $y = y_0y_1y_2\cdots \in Z^\omega$ une g-partie et soit $n > 0$ tel que $y_n = (p^2, a)$. Dans ces conditions $z = x_0x_1x_2\cdots x_kx_{k+1}y_0y_1y_2\cdots \in Z^\omega$ est également une g-partie (le coup $x_{k+1} \to y_0$ est un coup de remise à zéro). Comme g est une stratégie gagnante sans mémoire, z doit être consistante. On en déduit donc que $x_{k+1} = (p^1, 2)$ si et seulement si $y_{n+1} = (p^2, 2)$ et sinon $x_{k+1} = (q^1, 1)$ si et seulement si $y_{n+1} = (q^2, 1)$ avec $q^1_{W(a)} = q^2_{W(a)}$. De plus z étant une g-partie, on a $x_{k+1} = g(x_k)$ et $y_{n+1} = g(y_n)$. □

Proposition 42 *Soit g une stratégie gagnante sans mémoire pour le joueur 0 dans le jeu global $\widetilde{G}$. Pour toute position $(p, a) \in Z_0$ accessible par une g-partie, on définit $f(p_{R(a)}, a) = \mathit{stop}$ si $g((p, a)) = (p, 2)$ et $f(p, a) = q_{W(a)}$ si $g((p, a)) = (q, 1)$. Alors f est une stratégie distribuée gagnante sans mémoire pour l'équipe 0 dans le jeu distribué G.*

Preuve Soit t une f-partie f-maximale, et soit $(a_1, p^1)(a_2, p^2)\cdots$ une linéarisation de t. Pour tout $n \geq 0$, nous définissons $t[n] = (a_1, p^1)\cdots(a_n, p^n)$ et $q^n = q^0 * \mathrm{lw}(t[n])$. Nous définissons alors la suite $(z_n)_{n\geq 0}$ par $z_0 = (q^0, 0)$ et pour $n \geq 0$, $z_{2n+1} = (q^n, a_{n+1})$ et $z_{2n+2} = (q^{n+1}, 1)$. Nous montrons que $z = z_0z_1z_2\cdots \in Z^\infty$ est une g-partie. Soit $n \geq 0$. Nous devons montrer que $z_{2n+1} \to z_{2n+2}$ est un coup joué selon g. Nous avons $q^n_{R(a)} = (q^0 * \mathrm{lw}(\partial_{R(a)}t[n]))_{R(a)}$ et comme t est une f-partie, nous avons $p^{n+1} = f((q^0 * \mathrm{lw}(\partial_{R(a)}t[n]))_{R(a)}, a) = f(q^n_{R(a)}, a)$. Par définition de f, nous avons $g(q^n, a) = (q, 1)$ avec $q_{W(a)} = p^{n+1}$ et $q_{\mathcal{P}\setminus W(a)} = q^n_{\mathcal{P}\setminus W(a)}$. Comme nous avons également $q^{n+1}_{W(a)} = (q^0 * \mathrm{lw}(t[n+1]))_{W(a)} = p^{n+1}$ et $q^{n+1}_{\mathcal{P}\setminus W(a)} = (q^0 * \mathrm{lw}(t[n+1]))_{\mathcal{P}\setminus W(a)} = (q^0 * \mathrm{lw}(t[n]))_{\mathcal{P}\setminus W(a)} = q^n_{\mathcal{P}\setminus W(a)}$, nous en déduisons que $q^{n+1} = q$ et $(q^n, a) \to (q^{n+1}, 1)$ est un coup joué selon la stratégie g.

La partie z n'est pas forcément juste, nous allons donc la transformer afin d'obtenir une partie juste. Soit A la projection sur Σ_0 de $\mathrm{Alph}_\infty(t)$ et $B = \{b_1, \ldots, b_k\}$ l'ensemble des lettres de Σ_0 tel que pour tout $b \in B$, $\partial_{R(b)}t$ est une trace finie. Soit $N > 0$ tel que $\mathrm{Alph}(t[N]^{-1}t) = \mathrm{Alph}_\infty(t)$. Nous avons $A \times B \subseteq \mathrm{I}$, donc $\partial_{R(b)}t = \partial_{R(b)}t[n]$ pour tout $n \geq N$ et pour tout b dans B. Pour $n \geq N$, posons $y_n = (q^n, b_1)(q^n, 2)\cdots(q^n, b_k)(q^n, 2)$. Pour tout b dans B, comme t est f-maximale, nous avons $f(q^n_{R(b)}, b) = f((q^0 * \mathrm{lw}(\partial_{R(b)}t))_{R(b)}, b) = \mathrm{stop}$. Nous en déduisons, en utilisant la définition de f, que $g(q^n, b) = (q^n, 2)$ quel que soit b dans B ce qui montre que tous les coups du joueur 0 dans y_n sont joués selon la stratégie g. Donc la partie $x = z_0 z_1 \cdots z_{N-1} z_N y_n z_{N+1} y_{N+1} \cdots$ est une g-partie. De plus cette partie est juste et infinie. Comme g est une stratégie gagnante sans mémoire pour le joueur 0, on en déduit que x est une partie gagnante du jeu $\widetilde{G}$, donc $t = t(z) = t(x)$ appartient à $\mathcal{W}$, donc t est une partie gagnante du jeu G. □

Nous allons maintenant montrer comment utiliser le théorème 38 pour décider si l'équipe 0 a une stratégie distribuée gagnante sans mémoire. Nous rappelons que nous notons $\mathrm{Lin}(t)$ l'ensemble des linéarisations de la trace t (voir section 2.3). Pour déterminer si l'équipe 0 a une stratégie distribuée gagnante sans mémoire dans le jeu G avec $\mathcal{W}$ un ensemble reconnaissable, nous pouvons énumérer toutes les stratégies sans mémoire de l'équipe 0 et vérifier si l'une d'entres elles est gagnante. Si nous considérons une stratégie sans mémoire f pour l'équipe 0, le jeu nous donne un automate asynchrone qui calcule toutes les parties jouées selon la stratégie f. L'ensemble de ces parties est donc un ensemble reconnaissable et tester si f est une stratégie gagnante revient à tester l'inclusion de langages reconnaissables de traces. Cependant le théorème 38 nous permet d'utiliser une méthode plus efficace. Le principe est de construire le jeu global $\widetilde{G}$, de le transformer en un jeu de parité et d'utiliser ensuite les algorithmes connus sur ce jeu.

La condition de gain $\widetilde{W}$ pour le joueur 0 sur le jeu global $\widetilde{G}$ peut être définie par $\widetilde{W} = \widetilde{W}_\mathrm{c} \cap (\widetilde{W}_\mathrm{g} \cup \widetilde{W}_\mathrm{nf})$ où $\widetilde{W}_\mathrm{g} = \{z \in Z^\omega \mid t(z) \in \mathcal{W}\}$, $\widetilde{W}_\mathrm{c} = \{z \in Z^\omega \mid z \text{ est consistante}\}$ et $\widetilde{W}_\mathrm{nf} = \{z \in Z^\omega \mid z \text{ n'est pas juste}\}$. Pour montrer qu'il existe un automate à parité qui reconnaît $\widetilde{W}$, il suffit de montrer que les langages $\widetilde{W}_\mathrm{g}$, $\widetilde{W}_\mathrm{c}$ et $\widetilde{W}_\mathrm{nf}$ sont reconnaissables.

- ▷ Nous montrons tout d'abord que le langage $\widetilde{W}_\mathrm{g}$ est reconnaissable. Pour ceci, nous pouvons ne pas considérer les coups de remise à zéro. En effet, nous pouvons remettre l'automate à zéro pour chaque coup de remise à zéro et comme les parties contenant une infinité de remises à zéro seront ensuite éliminées, nous pouvons ne pas les considérer. Il ne reste plus que les parties qui ne contiennent qu'un nombre fini de remises à zéro et dans ces conditions, le chemin précédent la dernière remise à zéro n'a pas d'importance. Ensuite en appariant les coups du jeu par deux nous obtenons facilement une correspondance entre deux coups consécutifs du jeu et une lettre de $\Sigma \uplus \{\varepsilon\}$. Nous obtenons donc facilement un automate reconnaissant les mots de $\widetilde{W}_\mathrm{g}$.
- ▷ Pour construire une automate reconnaissant toutes les parties consistantes (donc $\widetilde{W}_\mathrm{c}$) il suffit de conserver toutes les transitions $(p_{R(a)}, q_{W(a)})$ effectuées par le joueur 0 ainsi que les transitions refusées par le joueur 0. Cet automate va dans un état puit si une transition inconsistante est détecté.

- ▷ Pour $\widetilde{W}_{\mathrm{nf}}$, on peut construire un automate qui pour chaque lettre a de Σ_0 vérifie qu'il existe une infinité de coups de la forme (p, a) ou une infinité de suites de deux coups de la forme $(p, b)(q, 1)$ avec $a \,\mathrm{D}\, b$. De même on peut écrire un automate qui vérifie qu'il n'y a pas une infinité de coups de remise à zéro.

Finalement le langage $\widetilde{W}$ est donc reconnaissable. Nous pouvons construire à partir des automates pour $\widetilde{W}_{\mathrm{c}}$, $\widetilde{W}_{\mathrm{g}}$ et $\widetilde{W}_{\mathrm{nf}}$ un automate à parité pour $\widetilde{W}$. Nous obtenons donc que $\widetilde{G}$ est un jeu de parité et nous pouvons utiliser les résultats classiques sur ces jeux.

Enfin en utilisant la proposition 37 nous pouvons utiliser ce résultat pour trouver une stratégie gagnante à mémoire μ donnée.

Chapitre 17

Les conditions de gains reconnaissables

Nous allons maintenant montrer que dans des conditions particulières (lorsque l'architecture est une architecture série paralléle), nous pouvons décider si l'équipe 0 a une stratégie gagnante dans un jeu G dont la condition de gain est reconnaissable.

Pour se faire nous allons utiliser la reconnaissance des langages par morphisme.

Étant donné un morphisme de semi-groupe $\varphi : \Sigma^+ \to T$, nous noterons $\sim_\varphi \subseteq \Sigma^\infty \times \Sigma^\infty$ la relation définie par $u \sim_\varphi v$ si et seulement s'il existe deux factorisations en mots finis $u = u_0u_1\cdots$ et $v = v_0v_1\cdots$ avec le même nombre (possiblement infini) de facteurs, telles que $\varphi(u_i) = \varphi(v_i)$ pour tout i. Nous noterons $\approx_\varphi$ la clôture transitive de $\sim_\varphi$.

Nous dirons qu'un langage $L \subseteq \Sigma^\infty$ est reconnu par le morphisme $\varphi : \Sigma^+ \to T$, si T est fini et L est saturé par $\approx_\varphi$. L'ensemble des langages reconnaissables par morphisme est l'ensemble des langages reconnaissables.

Si maintenant φ est un morphisme de semi-groupe dont l'ensemble de départ est $\mathbb{M}(\Sigma, \mathrm{D}) \setminus \{\varepsilon\}$, la même définition nous permet de caractériser les langages reconnaissables de $\mathbb{M}(\Sigma, \mathrm{D})$.

Pour montrer qu'il est décidable de savoir si dans un jeu G sur une architecture série parallèle l'équipe 0 a une stratégie gagnante, nous allons montrer que pour un jeu donné, il existe une borne M calculable telle que si l'équipe 0 a une stratégie gagnante sur G, elle a une stratégie gagnante dont la taille de la mémoire est inférieure à M.

Cette propriété nous donne alors un algorithme pour vérifier si l'équipe 0 a une stratégie gagnante distribuée sur G. Il suffit d'énumérer toutes les mémoires de taille inférieure à M, et pour chacune d'entre elles, vérifier s'il existe une stratégie gagnante distribuée pour l'équipe 0 utilisant cette mémoire. Si c'est le cas, le problème est résolu, et si ce n'est pas le cas, nous savons qu'il n'existe pas de stratégie gagnante distribuée pour l'équipe 0.

Pour montrer cette propriété nous allons procéder par induction structurelle sur l'alphabet. Nous allons alors supposer qu'il existe une stratégie gagnante distribuée pour l'équipe 0 dans le jeu G. Nous allons alors la modifier afin d'en construire une nouvelle dont nous pourrons borner la taille de la mémoire.

Pour ce faire nous allons nous baser sur l'idée suivante. Si Σ est un alphabet série parallèle tel que $\Sigma = A \uplus B$ avec $A \times B \subseteq \mathrm{D}$ (Σ est le produit série des alphabets A et B), alors toute t trace sur l'alphabet Σ se décompose de façon unique (à symétie entre A et B près) en $t = t_1^A \cdot t_1^B \cdot t_2^A \cdots$ avec t_i^A des traces non vides sur l'alphabet A et t_i^B des traces non vides sur l'alphabet B. De plus, pour tout a dans A, pour tout i, $\partial_{R(a)} t_i^B = t_i^B$. On en déduit donc que les premières actions de chaque facteur ont tous les facteurs précédents dans leur passé. On va donc considérer les changements d'alphabet comme des points de coupure. En effet, à l'un de ces points de coupure, toutes stratégie est distribuée, car elle peut considérer l'intégralité de la partie pour choisir quel coup jouer. À ces endroits particuliers, nous allons donc définir une notion d'équivalence par rapport à la condition de gain. Deux traces seront équivalentes si elles peuvent être continuées de la même manière afin que si l'une est gagnante, l'autre l'est également.

En utilisant cette propriété nous allons donc uniformiser la stratégie, puis entre deux points de coupure, nous serons sur un alphabet plus petit et nous utiliserons l'induction pour trouver une stratégie à mémoire connue. Enfin nous reconstruirons une stratégie gagnante et à mémoire connue en utilisant toutes ces stratégies.

Le résultat principal de ce chapitre est alors le suivant :

Soit $(\Sigma, \mathcal{P}, R, W)$ une architecture série parallèle. Soient $G = (\Sigma_0, \Sigma_1, (Q_i)_{i\in\mathcal{P}}, (T_a)_{a\in\Sigma}, q^0, \mathcal{W})$ un jeu distribué sur l'architecture $(\Sigma, \mathcal{P}, R, W)$ avec $\Sigma = \Sigma_0 \uplus \Sigma_1$ et $\mathcal{W}$ un ensemble reconnaissable de $\mathbb{R}(\Sigma', \mathrm{D}')$. Nous avons alors le théorème suivant :

Théorème 43 *Il existe une fonction* Memory Size Upper Bound MSUB : $\Sigma \to \mathbb{N}$ *telle que s'il existe une stratégie distribuée gagnante f pour l'équipe* 0 *dans le jeu G, alors il existe une stratégie distribuée gagnante f' pour l'équipe* 0 *dans le jeu G, de mémoire de taille inférieure à* $\mathrm{MSUB}(\Sigma)$.

Pour prouver ce résultat, nous allons procéder par induction sur la structure de l'alphabet, et nous allons avoir besoin, pour des raisons techniques, d'une hypothèse d'induction un peu plus forte.

On commence par introduire $\psi : \mathbb{M}(\Sigma', \mathrm{D}') \to N$ un morphisme à valeur dans un semi-groupe N reconnaissant le langage $\mathcal{W}$ tel que, de plus, pour tout r, s dans $\mathbb{M}(\Sigma', \mathrm{D}')$, $\psi(r) = \psi(s) \Rightarrow \mathrm{Alph}(r) = \mathrm{Alph}(s)$.

La fonction $\mathrm{lw} : \mathbb{M}(\Sigma', \mathrm{D}') \to (Q_i \uplus \{\bullet\})_{i\in\mathcal{P}}$ (voir section 2.4) est également un morphisme, donc nous introduisons le morphisme $\mu_0 : \mathbb{M}(\Sigma', \mathrm{D}') \to (Q_i \uplus \{\bullet\})_{i\in\mathcal{P}} \times N$ qui à la trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$ associe $(\mathrm{lw}(r), \psi(r))$.

Ce morphisme reconnaît également le langage $\mathcal{W}$ car deux traces qui sont équivalentes pour μ_0 le sont également pour ψ, mais il a quelques propriétés supplémentaires. Tout d'abord deux traces qui sont équivalentes pour μ_0 ont le même alphabet. En particulier si l'une d'entre elles ne contient que des lettres de A ou de B, il en est de même pour la seconde. Enfin deux traces qui sont équivalentes pour μ_0 prennent la même valeur pour lw, donc elles ont le même état global. Ceci signifie que si on peut continuer l'une des deux traces d'une certaine façon en respectant les règles du jeu, on peut continuer de la même façon l'autre trace en respectant également les règles du jeu. Ces différentes propriétés font que μ_0 est un candidat intéressant pour une mémoire permettant à l'équipe 0 de déterminer sa stratégie.

Soit f une stratégie distribuée sur le jeu G, nous introduisons alors la fonction $\text{stop}_f : \mathbb{R}(\Sigma', \mathrm{D}') \to 2^{\Sigma_0}$ qui associe à une f-partie r l'ensemble des éléments a de Σ_0 tels que $\partial_{R(a)} r$ est fini et $f(r, a) = \text{stop}$.

Nous introduisons alors la définition suivante :

Définition 17 (Compatibilité).
Si f et f' sont deux stratégies distribuées, on dira que f' est f-compatible*, si et seulement si pour toute f'-partie finie r', il existe une f-partie finie r telle que $\mu_0(r) = \mu_0(r')$ et $\text{stop}_{f'}(r') = \text{stop}_f(r)$.*

Nous allons alors montrer le lemme suivante qui implique immédiatement le théorème 43.

Lemme 44 *Il existe une fonction* $\text{MSUB} : \Sigma \to \mathbb{N}$ *telle que s'il existe une stratégie distribuée gagnante f pour l'équipe* 0 *dans le jeu G, alors il existe une stratégie distribuée gagnante f' pour l'équipe* 0 *dans le jeu G, de mémoire μ' de taille inférieure à* $\text{MSUB}(\Sigma)$ *et f-compatible et vérifiant les conditions de la proposition 34.*

Nous montrons donc ce lemme par induction sur l'architecture $(\Sigma, \mathcal{P}, R, W)$. La base de l'induction est traitée dans la section 17.1, le cas parallèle dans la section 17.2 et enfin le cas série dans la section 17.3.

17.1 Base de l'induction

Nous supposons que Σ est composée d'une unique lettre a. Ce système est séquentiel et de nombreux résultats classiques sont donc applicables. En particulier le théorème 43 est vérifié dans ce cadre. Cependant les étapes suivantes de l'induction nécessitent l'hypothèse de compatibilité et nous avons donc besoin de refaire une preuve afin de montrer que nous pouvons assurer cette propriété.

Nous pouvons déjà séparé la preuve en deux cas :
Si $\Sigma = \Sigma_1$, alors l'équipe 0 a une stratégie vide, donc sans mémoire et le lemme 44 est vérifiée.

Le seul cas intéressant se pose donc quand $\Sigma = \Sigma_0$, et alors il existe une unique f-partie f-maximale r. Nous transformons alors tout préfixe fini t de r en une autre f-partie finie s vérifiant les propriétés suivantes :

1. Pour tout $s' \leq s$, il existe $t' \leq t$ tel que $\mu_0(s') = \mu_0(t')$
2. $\mu_0(s) = \mu_0(t)$
3. Pour tout $s', s'' \leq s$, $\mu_0(s') = \mu_0(s'') \Rightarrow s' = s''$.

Pour montrer ce résultat nous introduisons la fonction partielle model : $\mathbb{R}(\Sigma', \mathrm{D}') \to \mathbb{R}(\Sigma', \mathrm{D}')$ définie par $\text{model}(u) = \max\{t' \leq t \mid \mu_0(t') = \mu_0(u)\}$.

Nous construisons alors la partie s par induction de la manière suivante :

On pose $s_0 = \varepsilon$ puis
- ▷ Si $\text{model}(s_i) = t$ la construction est terminée, et $s = s_i$.
- ▷ Sinon, soit (a, q) tel que $\text{model}(s_i)(a, q) \leq t$, on pose $s_{i+1} = s_i(a, q)$. Ceci nous assure que s_{i+1} est une partie de G car $\text{lw}(s_i) = \text{lw}(\text{model}(s_i))$.

Puisque μ_0 est un morphisme, $\mu_0(s_{i+1}) = \mu_0(\text{model}(s_i)(a,q))$. De plus, $\text{model}(s_i)(a,q)$ est un préfixe de t, donc $\text{model}(s_{i+1})$ est bien défini. De plus $\text{model}(s_i) < \text{model}(s_{i+1}) \leq t$, donc cette construction termine et nous assure que l'on construit bien une partie s, finie, vérifiant les conditions 1, 2. Il nous reste à montrer la condition 3. Ce résultat est vrai car si on a deux préfixes s_1 et s_2 de s avec $s_1 < s_2$, alors $\text{model}(s_1) < \text{model}(s_2)$, donc par définition de la fonction model, $\mu(s_1) \neq \mu(s_2)$.

Maintenant que nous avons cette outil nous allons enfin pouvoir montrer le lemme 44. Nous allons considérer deux cas distincts selon que r est une partie finie ou infinie :

1. Si r est une partie finie, nous appliquons la construction définie ci-dessus à $t = r$. Dans ces conditions, s est une partie de G. On peut donc lui associer une stratégie (distribuée) g. Dans ces conditions g est une stratégie gagnante car $\mu_0(s) = \mu_0(r)$, donc r étant un élément de $\mathcal{W}$, s est également un élément de $\mathcal{W}$. D'après la condition 3, g est une stratégie à mémoire μ_0 dont la taille est inférieure à $|Q^\bullet| \times |N|$.

   De plus, la stratégie g est f-compatible. En effet, soit s une g-partie, alors $s' = \text{model}(s)$ est une f-partie qui vérifie les conditions de la compatibilité.

2. Si r est une partie infinie, en utilisant le théorème de Ramsey (voir [42]), on peut décomposer r en $r_0 r_1 r_2 \cdots$ tel que $\mu_0(r_0) = \mu_0(r_0 r_1)$ et pour tout $i \geq 1$, $\mu_0(r_i) = \mu_0(r_1)$.

   Dans ces conditions, la partie $r_0 r_1^\omega$ est également une partie de G et elle est gagnante car elle est ψ-équivalente à r. Maintenant en utilisant la construction définie ci-dessus à $t = r_0$ et à $t = r_1$, nous obtenons deux traces s_0 et s_1 telles que $s_0 s_1^\omega$ est une partie de G qui est de plus gagnante car $s_0 s_1^\omega$ est ψ-équivalente à $r_0 r_1^\omega$ donc à r. On peut encore une fois associer à cette partie une stratégie (distribuée) g.

   Maintenant on associe à chaque préfixe fini s' de $s_0 s_1^\omega$ un élément $\mu'(s')$ dans $Q^\bullet \times N \times \{0,1\}$ défini ainsi :

   ▷ Si s' est un préfixe de s_0, on lui associe $(\mu_0(s'), 0)$.
   ▷ Si $s' = s_0 s_1^n s_1'$ avec $s_1' < s_1$, on lui associe $(\mu_0(s_1'), 1)$.

   Par construction, si $\mu'(s') = \mu'(s'')$, alors $s'^{-1}s = s''^{-1}s$.

   En effet, si $\mu'(s') = (m, 0)$, alors $s' = s''$ et le résultat est vrai. Maintenant si $\mu'(s') = (m, 1)$, alors s' s'écrit $s_0 s_1^k s_1'$ avec $s_1' < s_1$ et s'' s'écrit $s_0 s_1^l s_1'$, donc $s'^{-1}s = s''^{-1}s$.

   Nous en déduisons donc que g est une stratégie distribuée de mémoire μ' dont la taille est inférieure à $2 * |Q^\bullet| * |N|$ et qui vérifie les conditions de la propriété 34.

   Il nous reste à montrer que g est f-compatible.

   On considère un préfixe fini de $s_0 s_1^\omega$, si c'est un préfixe s_0' de s_0, alors il existe un préfixe r_0' de r_0 tel que $\mu_0(s_0') = \mu_0(r_0')$ et $\text{stop}_g(s_0') = \emptyset = \text{stop}_f(r_0')$ et on a bien la compatibilité. Sinon il s'écrit $s_0 s_1^n s_1'$ avec $\varepsilon < s_1' \leq s_1$, alors il existe un préfixe r_1' de r_1 tel que $\mu_0(s_1') = \mu_0(r_1')$. Mais dans ces conditions $\mu_0(r_0) = \mu_0(s_0 s_i^n)$, donc $\mu_0(r_0 r_1') = \mu_0(s_0 s_i^n s_1')$ et $\text{stop}_g(r_0 r_1') = \emptyset = \text{stop}_f(s_0 s_i^n s_1')$ et on a encore la compatibilité.

Finalement, on a donc montré le lemme 44 quand Σ ne contient qu'une seule lettre et nous avons vu que nous pouvons prendre $\text{MSUB}(\{a\}) = 2 * |Q^\bullet| * |N|$.

17.2 Cas parallèle

Nous allons maintenant considérer le cas où l'architecture est composée de deux ensembles de processus complètement indépendants.

On travaille donc sur l'architecture $(\Sigma, \mathcal{P}, R, W)$ avec $\Sigma = A \uplus B$ et $A \times B \cap \mathrm{D} = \emptyset$.

Considérons r, un partie de G, on peut décomposer r en r_A et r_B où $\mathrm{Alph}(r_A) \subseteq A$, $\mathrm{Alph}(r_B) \subseteq B$ et $r = r_A \cdot r_B = r_B \cdot r_A$.

Lemme 45 *Soit f une stratégie distribuée te soient r_1 et r_2, deux f-parties, alors $r_{1A} r_{2B}$ est une f-partie.*

Preuve Tout d'abord, comme A et B sont deux alphabets indépendants, r_{1A} est un préfixe de r_1 et r_{2B} est un préfixe de r_2, donc r_{1A} et r_{2B} sont deux f-parties. De plus, elles sont indépendantes.

Il suffit donc de remarquer que la réunion de 2 f-parties indépendantes est encore une f-parties. □

Soit P l'ensemble des f-parties, f-maximale. On définit P_A comme la projection de P sur A' et P_B la projection de P sur B'. On déduit du lemme précédent que $P = P_A \times P_B$ et que P_A est un ensemble de parties jouées sur l'alphabet A et P_B un ensemble de parties jouées sur l'alphabet B.

On peut ensuite considérer deux jeux $G_A = (\Sigma_0 \cap A, \Sigma_1 \cap A, (Q_i)_{i \in \mathcal{P}}, (T_a)_{a \in A}, q^0, [P_A]_{\approx_\psi})$ et $G_B = (\Sigma_0 \cap B, \Sigma_1 \cap B, (Q_i)_{i \in \mathcal{P}}, (T_b)_{b \in B}, q^0, [P_B]_{\approx_\psi})$.

Ce sont deux jeux distribués dont la condition de gain est reconnaissable. De plus f est une stratégie gagnante pour l'équipe 0 pour chacun de ces deux jeux.

On en déduit donc qu'il existe deux stratégies f'_A et f'_B, distribuées, f-compatible, de mémoire respective μ'_A et μ'_B, bornées respectivement par $\mathrm{MSUB}(A)$ et $\mathrm{MSUB}(B)$, et gagnantes respectivement pour les jeux G_A et G_B.

On défini alors la stratégie f' de la manière suivante :

▷ $f'(r, c) = f'_A(\partial_{R(c)} r, c)$ si $c \in A$

▷ $f'(r, c) = f'_B(\partial_{R(c)} r, c)$ si $c \in B$

Sa mémoire est $\mu'(r) = (\mu'_A(\partial_A r), \mu'_B(\partial_B r))$. Cette fonction est bien une mémoire distribuée. En effet, si on revient à la définition 13, il suffit de considérer un élément m et de montrer que ${\mu'}^{-1}(m)$ est un langage reconnaissable. Mais m s'écrit (m_A, m_B) et on a : ${\mu'}^{-1}(m) = {\mu'}_A^{-1}(m_A) \times {\mu'}_B^{-1}(m_B)$. Or ${\mu'}_A^{-1}(m_A)$ et ${\mu'}_B^{-1}(m_B)$ sont des langages reconnaissables car μ'_A et μ'_B sont des mémoires distribuées, donc ${\mu'}_A^{-1}(m_A) \times {\mu'}_B^{-1}(m_B)$ est également un langage reconnaissable, donc μ est bien une mémoire distribuée.

Ceci définit bien une stratégie distribuée. Sa mémoire est bornée par $\mathrm{MSUB}(A) \cdot \mathrm{MSUB}(B)$. De plus comme μ'_A et μ'_B vérifient les conditions de la proposition 34, μ' elle-même vérifie encore ces conditions.

Il nous reste à montrer que la stratégie f' est f-compatible et gagnante.

On considère r' une f'-partie. On peut écrire $r' = r'_A r'_B$ ou r'_A est une f'_A-partie et r'_B est une f'_B-partie.

Par f-compatibilité, il existe une f-partie r_A et une f-partie r_B telles que $\mu_0(r_A) = \mu_0(r'_A)$ et $\mu_0(r_B) = \mu_0(r'_B)$. De plus $\mu_0(r') = \mu_0(r'_A) \cdot \mu_0(r'_B) = \mu_0(r_A) \cdot \mu_0(r_B) = \mu_0(r)$ où $r = r_A r_B$. De plus $\mathrm{stop}_{f'}(r'_A r'_B) = \mathrm{stop}_{f'_A}(r'_A) \uplus$

$\text{stop}_{f'_B}(r'_B) = (\text{stop}_f(r_A) \cap A) \uplus (\text{stop}_f(r_B) \cap B) = \text{stop}_f(r_A r_B)$ donc f' est bien f-compatible.

Considérons maintenant une f'-partie f'-maximale $r' = r'_A \cdot r'_B$.

Tout d'abord r'_A est une f'_A-partie f'_A-maximale, donc une partie gagnante de G_A, donc il existe une f-partie $r_A \in P_A$ telle que $r'_A \approx_\psi r_A$. De même il existe une f-partie $r_B \in P_B$ telle que $r'_B \approx_\psi r_B$. Alors, puisque $A \times B \subseteq \text{I}$, on obtient $r' = r'_A \cdot r'_B \approx_\psi r_A \cdot r_B$. Or $r_A \cdot r_B$ est une partie de P, donc une partie gagnante de G, donc r' est également une partie gagnante de G ce qui prouve que f' est une stratégie gagnante.

Nous avons donc vérifié le lemme 44 et nous avons vu que lorsque Σ est composé de deux ensembles de processus indépendants, nous pouvons prendre $\text{MSUB}(\Sigma) = \text{MSUB}(A) \cdot \text{MSUB}(B)$.

17.3 Cas série

Nous allons maintenant considérer le cas où l'architecture est composée de deux ensembles de processus complètement dépendants.

On travaille donc sur l'architecture $(\Sigma, \mathcal{P}, R, W)$ avec $\Sigma = A \uplus B$ et $A \times B \subseteq \text{D}$.

Pour pouvoir montrer le lemme 44 dans ce cadre nous allons avoir besoin de montrer une propriété algébrique :

17.3.1 Une propriété algébrique

Soit $L \subseteq \Sigma^\infty$ reconnu par le morphisme $\varphi : \Sigma^+ \to T$. Pour x dans T, nous définissons :

$$
\begin{aligned}
S(x) &= \{y \in T \mid xy = x\} \\
S_\top(x) &= \{y \in S(x) \mid \varphi^{-1}(x)(\varphi^{-1}(y))^\omega \cap L \neq \emptyset\} \\
S_\bot(x) &= S(x) \setminus S_\top(x)
\end{aligned}
$$

L'ensemble $S(x)$ est l'ensemble des stabilisateurs à droite de x.

Nous commençons par montrer que nous pouvons écrire $S_\top(x) = \{y \in S(x) \mid \varphi^{-1}(x)(\varphi^{-1}(y))^\omega \subseteq L\}$. En effet, considérons $u \in \varphi^{-1}(x)(\varphi^{-1}(y))^\omega$, alors u s'écrit $u_0 v_0 v_1 \cdots v_n \cdots$ avec $\varphi(u_0) = x$ et pour tout i, $\varphi(v_i) = y$. De plus $\varphi^{-1}(x)(\varphi^{-1}(y))^\omega \cap L \neq \emptyset$ donc il existe $u' = u'_0 v'_0 v'_1 \cdots v'_n \cdots \in L$ tel que $\varphi(u'_0) = x$ et pour tout i, $\varphi(v'_i) = y$. Dans ces conditions $u \approx_\varphi u'$, donc u est un élément de L car u' est un élément de L. Les deux définitions sont donc équivalentes.

Nous avons alors le lemme suivant :

Lemme 46 *Tout langage régulier L est reconnu par un morphisme $\varphi : \Sigma^+ \to T$ tel que pour tout $x \in T$, $S_\top(x)$ et $S_\bot(x)$ sont des semi-groupes.*

Preuve Pour prouver ce lemme, nous allons partir d'un langage régulier L et construire le morphisme reconnaissant L et vérifiant les propriétés du lemmes 46.

Nous considérons donc $\mathcal{A} = (\Sigma, Q, \delta, F, \gamma, q^0)$ un automate à parité déterministe et complet reconnaissant le langage L (voir [35]). L'ensemble Q est

l'ensemble des états, $\delta : Q \times \Sigma \to Q$ est la fonction de transition, $F \subseteq Q$ est l'ensemble des états finaux, $\gamma : Q \times \Sigma \to \mathbb{N}$ est la fonction d'évaluation des arrêtes et q^0 est l'état initial. Nous étendons la fonction γ aux chemins : si $u \in \Sigma^\infty$, $\gamma(p,u)$ est la suite des couleurs vues le long du chemin partant de p et étiqueté par u. Comme $\mathcal{A}$ est un automate complet et déterministe, γ est bien défini sur $Q \times \Sigma^\infty$.

Un mot u est accepté par $\mathcal{A}$ s'il est fini et $\delta(q^0,u) \in F$, ou alors s'il est infini et $\liminf \gamma(q^0,u)$ est un nombre paire.

Nous munissons alors l'ensemble des matrices $\mathcal{M} = (\mathbb{N} \cup \{-\infty\})^{Q\times Q}$ du produit défini par $(M{\cdot}N)_{p,q} = \max_{i\in Q} \min(M_{p,i}, N_{i,q})$. Ceci défini alors un semi-groupe car l'opération produit est bien associative du fait de la distributivité de l'opération min sur l'opération max.

Pour u un mot de Σ^+, nous définissons alors $\varphi(u) \in \mathcal{M}$ par

$$\varphi(u)_{p,q} = \begin{cases} -\infty & \text{si } \delta(p,u) \neq q \\ \min \gamma(p,u) & \text{sinon} \end{cases}$$

On vérifie facilement que $\varphi(uv) = \varphi(u)\varphi(v)$. En effet, considérons $\varphi(uv)_{p,q}$. L'automate $\mathcal{A}$ étant déterministe complet, en partant de l'état p et en lisant le mot uv nous obtenons un chemin $p \xrightarrow{u} p' \xrightarrow{v} p''$. Nous considérons alors deux cas :

- ▷ Si $p'' \neq q$, alors $\varphi(uv)_{p,q} = -\infty$ et quelque soit i, soit $\varphi(u)_{p,i}$, soit $\varphi(v)_{i,q} = -\infty$ sinon nous aurions un chemin $p \xrightarrow{u} i \xrightarrow{v} q$, donc $(\varphi(u)\varphi(v))_{p,q} = -\infty$.
- ▷ Si $p'' = q$, alors $\varphi(uv)_{p,q} = \min \gamma(p,uv) = \min(\min \gamma(p,u), \min \gamma(p',v))$. Or comme pour tout $i \neq p'$, $\varphi(u)_{p,i} = \varphi(v)_{i,q} = -\infty$, on obtient bien $(\varphi(u)\varphi(v))_{p,q} = \varphi(uv)_{p,q}$.

Nous en déduisons donc que φ est un morphisme et donc $T = \varphi(\Sigma^+)$ est un semi-groupe finie de $\mathcal{M}$.

Nous allons montrer que φ est le morphisme que nous recherchons.

Nous commençons par montrer que $\varphi : \Sigma^+ \to T$ reconnaît L. Soient u et v deux mots de Σ^∞, nous considérons deux factorisations en mots finis, non vides avec le même nombre de facteur (possiblement infini) $u = u_0u_1\cdots$ et $v = v_0v_1\cdots$, telles que $\varphi(u_i) = \varphi(v_i)$ pour tout i.

Si u et v sont tous les deux des mots finis, alors $\varphi(u) = \varphi(v)$. Dans ces conditions, l'unique état tel que $\varphi(u)_{q^0,q} \neq -\infty$ est égal à l'unique état tel que $\varphi(v)_{q^0,q} \neq -\infty$, donc $\delta(q^0,u) = \delta(q^0,v)$. Nous en déduisons donc que $u \in L$ si et seulement si $v \in L$.

Si u et v sont tous les deux infinis, soit $q^{i+1} = \delta(q^0, u_0\cdots u_i)$. Comme $\varphi(u_j) = \varphi(v_j)$ pour tout j, nous avons $q^{i+1} = \delta(q^0, v_0\cdots v_i)$. Comme $\liminf \gamma(q^0,u) = \liminf(\varphi(u_i)_{q^i,q^{i+1}})$, nous obtenons que $\liminf \gamma(q^0,u) = \liminf \gamma(q^0,v)$, donc de nouveau $u \in L$ si et seulement si $v \in L$.

Nous en déduisons finalement que φ reconnaît bien le langage L.

Nous allons maintenant montrer que quel que soit x dans T, $S_\top(x)$ est un semi-groupe. Comme $S(x)$ est un semi-groupe, il suffit de montrer que si $y,z \in S_\top(x)$, alors $\varphi^{-1}(x)(\varphi^{-1}(yz))^\omega \cap L \neq \emptyset$. Comme le morphisme φ est surjectif, nous pouvons choisir $u,v,w \in \Sigma^+$ tels que $\varphi(u) = x$, $\varphi(v) = y$ et $\varphi(w) = z$. Dans ces conditions, nous avons $u(vw)^\omega \in \varphi^{-1}(x)(\varphi^{-1}(yz))^\omega$. Comme $\varphi(u) = \varphi(uv) = \varphi(uvw)$, les mots u, uv et uvw agissent de manière identique sur les états et en particulier $\delta(q^0,u) = \delta(q^0,uv) = \delta(q^0,uvw)$. Nommons q cet état. Par définition de φ, nous avons $\gamma(uv^\omega) = \varphi(v)_{q,q} = y_{q,q}$, qui est paire car

$uv^\omega \in L$. De même $z_{q,q} = \varphi(w)_{q,q}$ est paire. Dans ces conditions $\gamma(u(vw)^\omega) = \min(y_{q,q}, z_{q,q})$ est paire, donc $u(vw)^\omega \in L$.

La preuve pour $S_\perp(m)$ est très semblable. Il suffit de montrer que si $y, z \in S_\perp(x)$, alors $\varphi^{-1}(x)(\varphi^{-1}(yz))^\omega \cap L = \emptyset$. Nous considérons donc trois mots quelconques $u, v, w \in \Sigma^+$ tels que $\varphi(u) = x$, $\varphi(v) = y$ et $\varphi(w) = z$. Il suffit donc de montrer que $u(vw)^\omega \notin L$. Comme $\varphi(u) = \varphi(uv) = \varphi(uvw)$, les mots u, uv et uvw agissent de manière identique sur les états et en particulier $\delta(q^0, u) = \delta(q^0, uv) = \delta(q^0, uvw)$. Nommons q cet état. Par définition de φ, nous avons $\gamma(uv^\omega) = \varphi(v)_{q,q} = y_{q,q}$, qui est impaire car $uv^\omega \notin L$. De même $z_{q,q} = \varphi(w)_{q,q}$ est impaire. Dans ces conditions $\gamma(u(vw)^\omega) = \min(y_{q,q}, z_{q,q})$ est impaire, donc $u(vw)^\omega \in L$.

Nous avons donc trouvé un semi-groupe et un morphisme vérifiant les propriétés recherchées. □

Il nous faut remarquer que cette propriété est vraie sur les langages de mots, mais faux sur les langages de traces. Nous ne pouvons pas trouver de morphisme $\varphi : \mathbb{M}(\Sigma, \mathrm{D}) \to T$ qui vérifie la même propriété.

Tout d'abord, la construction décrite dans la preuve ne s'étend pas aux langages de traces. En effet, si nous voulons étendre cette preuve, il nous faudrait partir d'un automate diamant. Or considérons l'alphabet de dépendance $\Sigma = \{a, b\}$ avec $a \,\mathrm{I}\, b$ et le langage $L = \{w \in \mathbb{M}(\Sigma, \mathrm{D}) \mid \mathrm{Alph}_\infty(w) \neq \{a, b\}\}$. Considérons $\mathcal{A}$ un automate à parité déterministe, complet et diamant qui reconnaîtrait tous les mots de ce langages. Considérons le chemin a^ω à partir de q^0. Il existe forcément un état q^1 répété le long de ce chemin, et donc deux nombre n_1 et n_2 tels que l'on ait le chemin $q^0 \xrightarrow{a^{n_1}} q^1 \xrightarrow{a^{n_2}} q^1$. Maintenant considérons le mot $a^{n_1}b^\omega$. Il existe alors forcément un état q^2 et deux nombres m_1 et m_2 tels que l'on aie le chemin $q^0 \xrightarrow{a^{n_1}} q^1 \xrightarrow{b^{m_1}} q^2 \xrightarrow{b^{m_2}} q^2$. De plus $a^{n_1}b^{m_1}b^\omega$ est un mot de L, donc $\gamma(q^2, b^{m_2})$ est pair. Comme $\mathcal{A}$ est diamant, et que nous avons le chemin $q^1 \xrightarrow{a^{n_2}} q^1 \xrightarrow{b^{m_1}} q^2$, nous avons également le chemin $q^1 \xrightarrow{b^{m_1}} q^2 \xrightarrow{a^{n_2}} q^2$. De plus, le mot $a^{n_1}b^{m_1}a^\omega$ est un mot de L, donc $\gamma(q^2, a^{n_2})$ est pair. Maintenant nous pouvons considérer le mot $a^{n_1}b^{m_1}(a^{n_2}b^{m_2})^\omega$. Par construction $\gamma(q^0, a^{n_1}b^{m_1}(a^{n_2}b^{m_2})^\omega) = \min(\gamma(q^2, b^{m_2}), \gamma(q^2, a^{n_2}))$, donc pair. Nous en déduisons que $a^\omega b^\omega$ est accepté par l'automate $\mathcal{A}$, donc il n'existe pas d'automate à parité déterministe, complet et diamant qui reconnaît L.

En fait le problème n'est pas lié à la technique de preuve, mais bien au résultat lui même. Considérons un morphisme φ reconnaissant le même langage L que ci-dessus. Il existe un entier n tel que $e_1 = \varphi(a)^n$ et $e_2 = \varphi(b)^n$ soient idempotents (voir [42]). De plus $e_1e_2 = e_2e_1 = e_1e_2e_2 = e_2e_1e_1$. On en déduit donc que e_1 et e_2 appartiennent à $S(e_1e_2)$ et donc à $S_\top(e_1e_2)$ car $a^nb^na^\omega$ et $a^nb^nb^\omega$ sont dans L. En revanche $e_1e_2 \notin S_\top(e_1e_2)$ car $a^nb^n(a^nb^n)^\omega$ n'est pas un élément de L.

Ceci montre qu'en général nous ne pouvons pas trouver de morphisme de $\mathbb{M}(\Sigma', \mathrm{D}')$ vérifiant le lemme 46. Ceci explique que nous n'avons pas pu choisir pour le morphisme ψ (le morphisme alphabétique reconnaissant le langage $\mathcal{W}$ défini à la page 114) un morphisme vérifiant ces propriétés alors que ceci nous aurait aidé à traiter le cas série.

17.3.2 Uniformisation des stratégies

Pour traiter le cas série, nous allons commencer par uniformiser les stratégies au niveau des changements de sous-alphabet. Pour cela nous allons définir une mémoire μ, et nous allons montrer que nous pouvons modifier la stratégie f en une stratégie g qui sera distribuée et gagnante et qui aura de plus la propriété d'être uniforme. Cela signifie que si deux g-parties r et r' atteignent le même état de la mémoire que nous aurons définie, et que rs est une g-partie qui change de sous-alphabet entre r et s alors $r's$ est également une g-partie. La stratégie g n'est pas une stratégie à mémoire μ, mais elle s'en approche car la connaissance de μ est suffisante pour connaître la valeur de g dans certains cas particuliers.

Pour définir la mémoire μ nous commençons par introduire la fonction Change : $\mathbb{R}(\Sigma', \mathrm{D}') \to \{A, B, \Sigma\}$ définie par :

$$\mathrm{Change}(r) = \begin{cases} \Sigma & \text{si } r = \varepsilon \\ A & \text{si } \max(r) \subseteq B \\ B & \text{si } \max(r) \subseteq A \end{cases}$$

Nous introduisons ensuite la décomposition par bloc d'une trace r de $\mathbb{R}(\Sigma', D')$ comme étant la décomposition $r = r_0 \cdot r_1 \cdots$ avec pour tout i, $\mathrm{Alph}(r_i) \subseteq A$ ou $\mathrm{Alph}(r_i) \subseteq B$, $\mathrm{Alph}(r_{i+1}) \subseteq \mathrm{Change}(r_i)$ et $r_i \neq \varepsilon$.

On appelle Δ l'ensemble des traces de $\mathbb{R}(\Sigma', D')$ dont l'alphabet infini est soit vide ($\mathbb{M}(\Sigma', D')$), soit contient au moins une lettre de A et une lettre de B.

On définit alors $\widetilde{\psi} : \Delta \to N^\infty$ par $\widetilde{\psi}(r) = \{\psi(r_0)\psi(r_1)\cdots \mid r_0 \cdot r_1 \cdots\}$ est la décomposition de $r \in \mathcal{W} \cap \Delta\}$ où ψ est le morphisme alphabétique reconnaissant le langage $\mathcal{W}$ défini à la page 114.

On introduit alors le morphisme d'évaluation ν de N^+ dans N défini par $\nu(n_1 n_2 \cdots n_k) = n_1 \cdot n_2 \cdot \ldots \cdot n_k$.

On considère le langage $\widetilde{\psi}(\mathcal{W} \cap \Delta)$ et on nomme $K = [\widetilde{\psi}(\mathcal{W} \cap \Delta)]_{\approx_\nu}$.

Remarque 3 *Le langage K est un langage reconnaissable de mots, et $\mathcal{W} \cap \Delta = \widetilde{\psi}^{-1}(K)$.*

Preuve Ce résultat est énoncé et prouvé dans le lemme 11 de [16]. □

On considère alors $\varphi : N \to M$ un morphisme reconnaissant K et vérifiant les propriétés du lemme 46.

Maintenant que nous avons introduit ces notions nous pouvons définir la mémoire μ par rapport à laquelle nous allons uniformiser la stratégie f.

Nous introduisons donc la mémoire suivante :

$\mu : \mathbb{M}(\Sigma', D') \to Q^\bullet \times N \times N \times M$ par $\mu(\varepsilon) = (\mathrm{lw}(\varepsilon), \psi(\varepsilon), 1_N, 1_M)$ et pour $r \neq \varepsilon$, $\mu(r) = (\mathrm{lw}(r), \psi(r), \psi(r_i), \varphi \circ \widetilde{\psi}(r_0 r_1 \cdots r_{i-1}))$ où $r = r_0 r_1 \cdots r_i$ est la décomposition par bloc de r.

Nous commençons par remarquer que les deux premières composantes de μ forment μ_0.

Ensuite par définition de φ et $\widetilde{\psi}$, nous avons $\varphi \circ \widetilde{\psi}(r) = \varphi \circ \widetilde{\psi}(r_0 r_1 \cdots r_{i-1}) \cdot \varphi(\psi(r_i))$, donc $\mu(r) = \mu(s)$ entraîne $\varphi \circ \widetilde{\psi}(r) = \varphi \circ \widetilde{\psi}(s)$.

Nous montrons que μ est une mémoire distribuée. Pour ce faire, il suffit de constater que μ vérifie les conditions de la proposition 34, c'est à dire que

pour tout $r, s \in \mathbb{M}(\Sigma', \mathrm{D}')$ tels que $\mu(r) = \mu(s)$, pour tout $c \in \Sigma$ et pour tout $q \in Q_{W(c)}$, on a $\mu(r(c,q)) = \mu(s(c,q))$.

Si $c \notin \mathrm{Change}(r)$, alors $\mu(r(c,q)) = (\mathrm{lw}(r) * \mathrm{lw}((c,q)), \psi(r) \cdot \psi((c,q)), \psi(r_2) \cdot \psi((c,q)), \varphi \circ \widetilde{\psi}(r_1)) = \mu(s(c,q))$.

Si $c \in \mathrm{Change}(r)$, alors $\mu(r(c,q)) = (\mathrm{lw}(r) * \mathrm{lw}((c,q)), \psi(r) \cdot \psi((c,q)), \psi((c,q)), \varphi \circ \widetilde{\psi}(r_1) \cdot \varphi(\psi(r_2))) = \mu(s(c,q))$.

Nous avons donc montré que μ vérifie les conditions de la proposition 34. La fonction μ est donc une mémoire distribuée.

Nous allons maintenant introduire une notion d'équivalence qui nous permettra de définir de manière formelle la notion d'uniformité.

Nous rappelons que la fonction stop_f est la fonction de $\mathbb{R}(\Sigma', \mathrm{D}') \to 2^{\Sigma_0}$ qui associe à une f-partie r l'ensemble des éléments a de Σ_0 tels que $\partial_{R(a)} r$ est fini et $f(r, a) = \mathrm{stop}$.

Si f et g sont deux stratégies distribuées, et r (respectivement s) est une f-partie (respectivement une g-partie) finie, nous dirons que (r, f) est équivalent à (s, g) noté $(r, f) \equiv (s, g)$ si

1. $\mu(r) = \mu(s)$.
2. $\mathrm{stop}_f(r) \setminus \mathrm{Change}(r) = \mathrm{stop}_g(s) \setminus \mathrm{Change}(s)$.

Cette équivalence entraîne quelques conséquences qui nous seront utiles.

Remarque 4 *La relation d'équivalence $\equiv$ est d'index fini et on a la propriété : $(r, f) \equiv (\varepsilon, g) \Leftrightarrow r = \varepsilon$. On appellera alors $\rho(\varepsilon)$ la classe d'équivalence de (ε, f) car elle ne dépend pas de f.*

Remarque 5 *Comme ψ assure que si deux traces sont équivalentes, elles ont le même alphabet, et que μ contient la valeur de ψ sur le dernier bloc, nous savons que si $\mu(r) = \mu(s) \neq \mu(\varepsilon)$, alors* Change$(r)$ = Change(s). *Si $\mu(r) = \mu(s) = \mu(\varepsilon)$, alors $r = s = \varepsilon$, donc* Change(r) *est encore égale à* Change(s). *Nous en déduisons que si deux traces sont équivalentes, elles ont le même* Change.

Preuve Pour montrer que la relation d'équivalence est d'index fini, il suffit de remarquer que μ et stop ne peuvent prendre qu'un nombre fini de valeurs.

De plus, dès que t est différent de ε, $\mathrm{lw}(t) \neq \mathrm{lw}(\varepsilon)$ et comme $\mathrm{stop}_f(\varepsilon) \setminus \mathrm{Change}(\varepsilon) = \emptyset$ quel que soit f, on en déduit donc la seconde propriété. □

Nous introduisons maintenant une notation supplémentaire qui spécialisera cette équivalence quand les deux stratégies impliquées seront identiques. Nous noterons alors $r \equiv_f r'$ à la place de $(r, f) \equiv (r', f)$.

La notion intuitive d'uniformité que nous avons décrite plus haut s'intéresse au suffixe d'une partie qui change de bloc. De manière formelle nous définissons $\mathrm{From}_f(r)$ comme l'ensemble des traces s de $\mathbb{R}(\Sigma', \mathrm{D}')$ telles que s fait changer de bloc à r (i.e. $\emptyset \neq \min(s) \subseteq \mathrm{Change}(r)$ et en particulier s est non vide), et $r \cdot s$ est une f-partie.

Nous pouvons maintenant définir de manière formelle la notion d'uniformité pour une stratégie.

Définition 18 (Stratégie uniforme).
On dira qu'une stratégie distribuée f est uniforme, si $r \equiv_f r' \Rightarrow \mathrm{From}_f(r) = \mathrm{From}_f(r')$.

Nous allons donc maintenant pouvoir énoncer formellement le résultat que nous recherchons.

Lemme 47 *Soit f une stratégie distribuée gagnante, il existe une stratégie gagnante distribuée g, f-compatible et uniforme.*

Pour montrer ce résultat, nous allons procéder par induction sur le nombre de classe d'équivalence de $\equiv_f$. Pour ce faire, nous introduisons les notions suivantes :

On définit $\rho_f = \{r \mid (r, f) \in \rho\}$ qui est une classe d'équivalence de $\equiv_f$.

Définition 19 (ρ-uniformité).
On dira qu'une stratégie distribuée f est ρ-uniforme, où ρ est une classe d'équivalence de $\equiv$ si et seulement si $r, r' \in \rho_f \Rightarrow \mathrm{From}_f(r) = \mathrm{From}_f(r')$.

Remarque 6 *Si quelle que soit la classe d'équivalence ρ de $\equiv$, f est ρ-uniforme, alors f est uniforme.*

Pour prouver le lemme 47, on va donc considérer une classe d'équivalence ρ de $\equiv$ et on va montrer le lemme suivant :

Lemme 48 *Soit f une stratégie distribuée gagnante et soit ρ une classe d'équivalence de $\equiv$, il existe une stratégie gagnante distribuée g, f-compatible ρ-uniforme, et telle que pour toute classe d'équivalence ρ' de $\equiv$, si f est ρ'-uniforme, alors g l'est également.*

Ce lemme suffit bien à prouver le lemme 47, car il suffit de partir de f, puis de l'uniformiser successivement par rapport à chaque classe d'équivalence de $\equiv$, et nous finissons par trouver une stratégie g qui est ρ-uniforme quelle que soit la classe d'équivalence ρ de $\equiv$.

Nous en sommes donc rendus à montrer le lemme 48.

Pour montrer ce lemme, nous allons considérer que f n'est pas ρ-uniforme, et nous allons construire g. En effet, si f est déjà ρ-uniforme, $g = f$ convient.

Dans le cas contraire, il existe une f-partie r dans ρ_f qui vérifie la propriété suivante :

$$\text{Quel que soit } s \in \mathrm{From}_f(r),\ r \cdot s \in \rho_f \Rightarrow \varphi \circ \widetilde{\psi}(s) \in S_\top(\varphi \circ \widetilde{\psi}(r)) \qquad (17.1)$$

Supposons en effet qu'une telle f-partie n'existe pas. Comme f n'est pas ρ-uniforme, ρ_f n'est pas vide, et pour toute trace r dans ρ_f, il existe une trace s dans $\mathrm{From}_f(r)$, telle que $r \cdot s \in \rho_f$ et $\varphi \circ \widetilde{\psi}(s) \in S_\perp(\varphi \circ \widetilde{\psi}(r))$.

On remarque que $\widetilde{\psi}(rs) = \widetilde{\psi}(r)\widetilde{\psi}(s)$ et donc $\varphi \circ \widetilde{\psi}(rs) = \varphi \circ \widetilde{\psi}(r)$.

On construit donc une trace infinie $t = r \cdot s_0 \cdot s_1 \cdots s_k \cdots$, telle que quel que que soit i, $\varphi \circ \widetilde{\psi}(s_i) \in S_\perp(\varphi \circ \widetilde{\psi}(r)))$, dans ces conditions, d'après le théorème de Ramsey on peut décomposer t en $uv_0v_1 \cdots$ où $u = rs_0 \ldots s_{j_0}$ et $v_i = s_{1+j_i} \ldots s_{j_{i+1}}$, $x = \varphi \circ \widetilde{\psi}(u) = \varphi \circ \widetilde{\psi}(r) = \varphi \circ \widetilde{\psi}(u) \cdot \varphi \circ \widetilde{\psi}(v_0)$ et $y = \varphi \circ \widetilde{\psi}(v_i) = \varphi \circ \widetilde{\psi}(v_0)$ pour tout i. D'après le lemme 46, on sait que $S_\perp(\varphi \circ \widetilde{\psi}(r)) = S_\perp(x)$ est un semigroupe, donc $y \in S_\perp(x)$ et donc $\widetilde{\psi}(t) = \widetilde{\psi}(u)\widetilde{\psi}(v_0)\widetilde{\psi}(v_1) \cdots \in \varphi^{-1}(x)(\varphi^{-1}(y))^\omega$ et n'est donc pas un mot de K. Finalement, on en déduit que t n'est pas une trace de $\widetilde{\psi}^{-1}(K) = \mathcal{W}$ ce qui en contradiction avec le fait que f soit une stratégie

gagnante car t est une f-partie et elle est f-maximale car elle est infinie et change infiniment d'alphabet.

On en déduit donc qu'il existe bien un r vérifiant la propriété énoncée ci-dessus. Nous allons donc considérer une telle f-partie r dans la suite de la preuve. Le but va alors être de construire une nouvelle stratégie g qui agira comme f sauf sur les traces qui prennent la même valeur que r pour μ. Sur ces traces, g agira comme f aurait agit sur r. La trace r a été choisie de telle façon a assurer que g restera une stratégie gagnante.

Nous définissons par induction une stratégie sur les mots g sur $\Sigma'^* \times \Sigma$ et une fonction partielle simul : $\Sigma'^* \to \mathbb{M}(\Sigma', \mathrm{D}')$ de la manière suivante :

- ▷ $\mathrm{simul}(\varepsilon) = \varepsilon$
- ▷ Si $s' = \mathrm{simul}(s)$ et $c \in \Sigma$:
  - ◇ Si $c \in \mathrm{Change}(s')$ et $s' \equiv_f r$ alors $g(s,c) = f(r,c) = p$ et si $p \neq \mathrm{stop}$ alors $\mathrm{simul}(s(c,p)) = r(c,p)$
  - ◇ sinon $g(s,c) = f(s',c) = p$ et si $p \neq \mathrm{stop}$ alors $\mathrm{simul}(s(c,p)) = s'(c,p)$

On remarque que si $s' = \mathrm{simul}(s)$, alors s' est une f-partie et s est une g-partie.

La fonction g ainsi définie est la stratégie que nous recherchons. Cependant, comme elle a été définie sur des mots, il nous faut tout d'abord vérifier que g est bien définie sur des traces. C'est à dire qu'elle ne prend pas deux valeurs différentes sur deux linéarisations de la même trace.

Pour montrer ce résultat nous commençons par montrer un lemme intermédiaire :

Lemme 49 *Nous posons* $s' = \mathrm{simul}(s)$ *et soient* $c, d \in \Sigma$ *avec* $c \,\mathrm{I}\, d$. *Soit* $p = g(s,c)$ *et* $q = g(s,d)$.

1. *Si* $p \neq$ *stop, alors* $g(s(c,p),d) = q$
2. *Si* $p, q \neq$ *stop, alors* $\mathrm{simul}(s(c,p)(d,q)) = \mathrm{simul}(s(d,q)(c,p))$

Preuve Nous commençons par remarquer que $c \in \mathrm{Change}(s')$ si et seulement si $d \in \mathrm{Change}(s')$, car sinon l'une des deux actions est dans A et la seconde dans B et $A \times B$ est inclus dans D.

Commençons par supposer $c \in \mathrm{Change}(s')$ et $s' \equiv_f r$:

1. Si $p \neq \mathrm{stop}$, alors $\mathrm{simul}(s(c,p)) = r(c,p)$ et $f(r,d) = q$. Remarquons que $d \notin \mathrm{Change}(r(c,p)$ donc $g(s(c,p),d) = f(r(c,p),d) = f(r,d) = q$ car f est une stratégie distribuée.
2. Si $p, q \neq \mathrm{stop}$, alors $\mathrm{simul}(s(c,p)) = r(c,p)$ et $\mathrm{simul}(s(d,q)) = r(d,q)$. Comme $d \notin \mathrm{Change}(r(c,p))$ nous obtenons $\mathrm{simul}(s(c,p)(d,q)) = r(c,p)(d,q)$. De la même manière $\mathrm{simul}(s(d,q)(c,p)) = r(d,q)(c,p)$. Les deux traces sont égales car c et d sont indépendants.

Maintenant si $c \notin \mathrm{Change}(s')$ ou bien $s' \not\equiv_f r$:

1. Si $p \neq \mathrm{stop}$, alors $\mathrm{simul}(s(c,p)) = s'(c,p)$ et $q = f(s',d)$. Comme $d \notin \mathrm{Change}(s'(c,p))$ nous obtenons $g(s(c,p),d) = f(s'(c,p),d) = f(s',d) = q$ comme f est une stratégie distribuée.
2. Si $p, q \neq \mathrm{stop}$, alors $\mathrm{simul}(s(c,p)) = s'(c,p)$ et $\mathrm{simul}(s(d,q)) = s'(d,q)$. Nous avons $d \notin \mathrm{Change}(s'(c,p)$ donc $\mathrm{simul}(s(c,p)(d,q)) = s'(c,p)(d,q)$. De la même manière $\mathrm{simul}(s(d,q)(c,p)) = s'(d,q)(c,p)$. Les deux traces sont égales car c et d sont indépendants.

□

Nous pouvons maintenant montrer que g est bien définie sur les traces (et en fait que simul est également définie sur les traces) :

Lemme 50 *Soit $u, v \in \Sigma'^*$ deux linéarisations de la même trace, alors u est une g-partie si et seulement si v est une g-partie et dans ce cas* $\text{simul}(u) = \text{simul}(v)$ *et pour tout c dans Σ_0, $g(u,c) = g(v,c)$.*

Preuve Sans perte de généralité, nous pouvons écrire $u = s(c,p)(d,q)t$ et $v = s(d,q)(c,p)t$ avec $c \,\mathrm{I}\, d$.

Si u est une g-partie, nous avons $p = g(s,c) \neq \text{stop}$ et $q \neq \text{stop}$. D'après le lemme 49, nous obtenons $q = g(s(c,p),d) = g(s,d)$ donc $s(d,q)$ est une g-partie. Encore une fois d'après le lemme 49 nous obtenons $p = g(s(d,q),c)$ et $s(d,q)(c,p)$ est une g-partie. De plus nous obtenons également $\text{simul}(s(c,p)(d,q)) = \text{simul}(s(d,q)(c,p))$. Puis finalement, par induction sur $|t|$ nous obtenons aisément que v est une g-partie et que $\text{simul}(u) = \text{simul}(v)$.

De plus $g(u,c)$ ne dépend que de $\text{simul}(u)$ et de c, donc $g(u,c) = g(v,c)$. □

Finalement nous avons montré que g et simul sont bien définies sur les *traces*.

Nous montrons maintenant que g est une stratégie distribuée. Ce résultat est une conséquence immédiate du lemme 49. En effet, considérons $s(c,p)$ une g-partie et $d \in \Sigma_0$ tel que $c \,\mathrm{I}\, d$, alors le lemme 49 nous assure que $g(s,d) = g(s(c,p),d)$ ce qui montre que g est une stratégie distribuée d'après le lemme 31.

Nous montrons maintenant que g est f-compatible. Pour ce faire, nous avons besoin de montrer le lemme suivant :

Lemme 51 *Soit s une g-partie et $s' = \text{simul}(s)$. Nous avons alors $(s,g) \equiv (s',f)$.*

Preuve Nous montrons ce résultat par induction sur la taille de s.

- ▷ Si $s = \varepsilon$ le résultat est vérifié.
- ▷ Soit s une g-partie, posons $s' = \text{simul}(s)$ et soit $c \in \Sigma$.
  - ◇ Si $c \in \text{Change}(s')$ et $s' \equiv_f r$ alors $g(s,c) = f(r,c) = p$ et si $p \neq \text{stop}$, alors $\text{simul}(s(c,p)) = r(c,p)$. Par induction, nous avons $(s,g) \equiv (s',f) \equiv (r,f)$, donc :
    - ⋆ $\mu(s) = \mu(r)$ et nous obtenons $\mu(s(c,p)) = \mu(r(c,p))$
    - ⋆ Considérons $d \notin \text{Change}(s(c,p)) = \text{Change}(r(c,p))$, alors nous avons $g(s(c,p),d) = f(r(c,p),d)$, donc $d \in \text{stop}_g(s(c,p))$ si et seulement si $d \in \text{stop}_f(r(c,p))$.

    Nous en déduisons $(s(c,p),g) \equiv (r(c,p),f)$.
  - ◇ Dans le cas contraire, si $c \notin \text{Change}(s')$ ou $s' \not\equiv_f r$ alors $g(s,c) = f(s',c) = p$ et si $p \neq \text{stop}$, $\text{simul}(s(c,p)) = s'(c,p)$. Nous avons $(s,g) \equiv (s',f)$ par induction et comme ci-dessus, nous montrons $(s(c,p),g) \equiv (s'(c,p),f)$.

□

Nous pouvons maintenant montrer que g est f-compatible. C'est un corollaire du lemme précédent. En effet considérons une g-partie s. Nous avons deux cas à envisager :

1. Si $(s,g) \not\equiv (r,f)$, alors nous considérons la f-partie $s' = \text{simul}(s)$. D'après le lemme précédent, $(s,g) \equiv (s',f)$, donc par définition de $\equiv$ nous savons que $\mu_0(s) = \mu_0(s')$ et par définition de g, pour tout $a \in \Sigma_0$, $g(s,a) = f(\text{simul}(s),a) = f(s',a)$, donc $\text{stop}_g(s) = \text{stop}_f(s')$.
2. Si $(s,g) \equiv (r,f)$, alors nous considérons la f-partie r. Par définition de $\equiv$, nous savons que $\mu_0(s) = \mu_0(r)$. De plus $\text{stop}_f(r) \setminus \text{Change}(r) = \text{stop}_g(s) \setminus \text{Change}(s)$ et $\text{Change}(r) = \text{Change}(s)$ (cf. remarque 5). Il nous reste alors à montrer que $\text{stop}_f(r) \cap \text{Change}(r) = \text{stop}_g(s) \cap \text{Change}(s)$, mais ce résultat est vrai par définition de g.

Nous avons donc maintenant montré que g est f-compatible.

Les lemmes 52 et 53 vont montrer que g est ρ-uniforme et que quelle que soit la classe d'équivalence ρ' de $\equiv$, si f est ρ'-uniforme, alors g l'est encore, mais tout d'abord nous allons avoir besoin d'un lemme technique :

Nous pouvons maintenant énoncer et prouver les lemmes liés à l'uniformité de g.

Lemme 52 *La stratégie g est ρ-uniforme.*

Preuve Soit s et t deux g-parties avec (s,g) et (t,g) des éléments de ρ. Soit $v \in \text{From}_g(s)$ que nous écrirons $v = (c,p)u$. Soient $s' = \text{simul}(s)$ et $t' = \text{simul}(t)$. D'après le lemme 51 nous avons $(s',f) \equiv (s,g) \equiv (r,f)$ car $(r,f) \in \rho$. De ce fait, $s' \equiv_f r$ et $c \in \text{Change}(s) = \text{Change}(s')$. Nous obtenons alors $p = g(s,c) = f(r,c)$ et $\text{simul}(s(c,p)) = r(c,p)$. Maintenant nous obtenons $(t',f) \equiv (t,g) \equiv (r,f)$ et $c \in \text{Change}(s) = \text{Change}(t) = \text{Change}(t')$. Donc $g(t,c) = f(r,c) = p$, $t(c,p)$ est une g-partie et $\text{simul}(t(c,p)) = r(c,p)$.

Par induction sur $|u|$, nous obtenons que $t(c,p)u$ est une g-partie et $\text{simul}(t(c,p)u) = \text{simul}(s(c,p)u)$. En effet, si $u = w(d,p)$ et $\text{simul}(t(c,p)w) = \text{simul}(s(c,p)w)$ alors $q = g(s(c,p)w,d) = g(t(c,p)w,d) \neq \text{stop}$ car sv est une g-partie et $\text{simul}(t(c,p)w(d,q)) = \text{simul}(s(c,p)w(d,q))$.

Nous en déduisons donc que $v \in \text{From}_g(t)$. □

Lemme 53 *Soit ρ' une classe d'équivalence de $\equiv$ tel que $\rho' \neq \rho$. Si f est ρ'-uniforme alors g est également ρ'-uniforme.*

Preuve Soit s et t deux g-parties avec (s,g) et (t,g) des éléments de ρ'. Soit v dans $\text{From}_g(s)$. Soit $s' = \text{simul}(s)$ et $t' = \text{simul}(t)$. Nous montrons par induction sur $|v|$ que tv est une g-partie et que soit $\text{simul}(sv) = s'v \equiv_f \text{simul}(tv) = t'v$, soit $\text{simul}(sv) = \text{simul}(tv)$.

- ▷ Le résultat est vrai pour $v = \varepsilon$ même si $\varepsilon \notin \text{From}_g(s)$ car d'après le lemme 51 $(s',f) \equiv (s,g) \equiv (t,g) \equiv (t',f)$.
- ▷ Supposons maintenant que $v = u(c,p)$, que tu est une g-partie et :
  - ◇ Si $\text{simul}(su) = \text{simul}(tu)$, alors $g(tu,c) = g(su,c) = p \neq \text{stop}$, donc tv est une g-partie et $\text{simul}(tv) = \text{simul}(sv)$.
  - ◇ Sinon nous avons $\text{simul}(su) = s'u \equiv_f \text{simul}(tu) = t'u$
    - ⋆ Si $c \in \text{Change}(s'u)$ et $s'u \equiv_f r$ alors $c \in \text{Change}(t'u) = \text{Change}(s'u)$ et $t'u \equiv_f r$. Donc $g(tu,c) = f(r,c) = g(su,c) = p \neq \text{stop}$ et $\text{simul}(tv) = r(c,p) = \text{simul}(sv)$. Puisque $g(tu,c) \neq \text{stop}$, tv est bien une g-partie.

⋆ Dans le cas contraire, si $c \notin \text{Change}(s'u)$ ou $s'u \not\equiv_f r$, nous avons $g(su,c) = f(s'u,c) = p \neq \text{stop}$ et $\text{simul}(sv) = s'v$ est une f-partie. Puisque f est ρ'-compatible et $s', t' \in \rho'_f$, nous en déduisons que $t'v$ est aussi une f-partie et que $\text{stop}_f(s'v) = \text{stop}_f(t'v)$. Donc $g(tu,c) = f(t'u,c) = p$ et tv est une g-partie et $\text{simul}(tv) = t'v$. Finalement, $s'u \equiv t'u$ implique que $\mu(s'v) = \mu(t'v)$ puisque μ est un morphisme. On a donc $s'v \equiv t'v$.

□

Jusqu'à présent nous avons montré que g est une stratégie distribuée, f-compatible et qui vérifie les propriétés d'uniformité recherchées. Il ne nous reste plus qu'à montrer que g est une stratégie gagnante.

Pour ce faire, nous allons considérer une g-partie s, g-maximale et nous allons montrer qu'elle est gagnante. C'est à dire, nous allons montrer que s est dans $\mathcal{W}$.

Nous considérons tout d'abord la décomposition par bloc de s : $s = s_0 s_1 \cdots$.

Nous appellons $\mathcal{I}$ l'ensemble suivant : $\mathcal{I} = \{i \mid (s_0 s_1 \cdots s_i, g) \equiv (r, f)\}$.

Nous nommons $i_1 < i_2 < \cdots$ les éléments de $\mathcal{I}$.

Enfin nous nommons $t_j = s_{1+i_j} \cdots s_{i_{j+1}}$ avec par convention $i_0 = -1$.

Si $\mathcal{I}$ est un ensemble infini, nous avons $s = t_0 t_1 t_2 \cdots$, dans le cas contraire, nous avons $s = t_0 \cdots t_{i-1} \overline{s}$.

Nous commençons par montrer le lemme technique suivants :

Lemme 54 $\text{simul}(t_0) = t_0$ *et pour* $j \geq 1$, $\text{simul}(t_0 \cdots t_j) = r t_j$.

Preuve

▷ Par induction sur $u \leq t_0$ nous montrons que $\text{simul}(u) = u$.
  ◇ Pour $|u| = 0$ le résultat est vérifié car $u = \varepsilon = \text{simul}(\varepsilon)$.
  ◇ Si $u(c,p) \leq t_0$ et $\text{simul}(u) = u$, soit $c \notin \text{Change}(u)$ soit $u \not\equiv_f r$ par définition de $\mathcal{I}$. Donc $g(u,c) = f(u,c) = p$ et $\text{simul}(u(c,p)) = u(c,p)$.

▷ Pour $j \geq 1$, par induction sur $\varepsilon < u \leq t_j$ nous montrons que $\text{simul}(t_0 t_1 \cdots t_{j-1} u) = ru$. Soit $v = t_0 t_1 \cdots t_{j-1}$ et $v' = \text{simul}(v)$.
  ◇ Si $u = (c,p)$ alors $c \in \text{Change}(v) = \text{Change}(v')$ et $(v', f) \equiv (v, g) \equiv (r, f)$, donc $p = f(r,c)$ et $\text{simul}(v(c,p)) = r(c,p)$.
  ◇ Si $u = w(c,p)$ avec $\varepsilon < w$ et $u \leq t_j$, nous savons que $\text{simul}(vw) = rw$. Soit $c \notin \text{Change}(vw) = \text{Change}(rw)$, soit $rw \not\equiv_f r$ par définition de $\mathcal{I}$. Donc $p = f(rw,c)$ et $\text{simul}(vu) = ru$.

□

Nous déduisons de ce lemme que $\varphi \circ \widetilde{\psi}(t_0) = \varphi \circ \widetilde{\psi}(r)$ et que pour tout $j \geq 1$, $\varphi \circ \widetilde{\psi}(t_j) \in S_\top(\varphi \circ \widetilde{\psi}(r))$.

En effet, $(t_0 t_1 \cdots t_j, g) \equiv (r, f)$ donc $\varphi \circ \widetilde{\psi}(t_0 t_1 \cdots t_j) = \varphi \circ \widetilde{\psi}(r)$ pour tout j.

Nous en déduisons que $\varphi \circ \widetilde{\psi}(t_0) = r$ et $\varphi \circ \widetilde{\psi}(t_j) \in S(\varphi \circ \widetilde{\psi}(r))$ pour tout $j \geq 1$. D'après les lemmes 51 et 54 $(rt_j, f) \equiv (t_0 t_1 \cdots t_j, g) \equiv (r, f)$, donc nous obtenons $\varphi \circ \widetilde{\psi}(t_j) \in S_\top(\varphi \circ \widetilde{\psi}(r))$ par choix de r.

Nous pouvons maintenant montrer que g est une stratégie gagnante. Nous allons considérer 3 cas.

▷ Si $\mathcal{I}$ est un ensemble infini.
  Soit $x = \varphi \circ \widetilde{\psi}(r)$. En utilisant le théorème de Ramsey, nous trouvons une suite infini $j_1 < j_2 < \cdots$ et un élément y de $S(x)$ tels que pour tout $k > 1$

$\varphi \circ \widetilde{\psi}(t_{j_{k-1}+1} \cdots t_{j_k}) = y$. Comme $S_\top(x)$ est un semi-groupe par choix de φ, nous obtenons également que y est un élément de $S_\top(x)$ par choix de r (voir équation 17.1). Maintenant $\varphi \circ \widetilde{\psi}((t_0 t_1 \cdots t_{j_1}) = \varphi \circ \widetilde{\psi}(r) = x$, donc $\widetilde{\psi}(s) \in \varphi^{-1}(x)\varphi^{-1}(y)^\omega \subseteq K$, et donc $s \in \widetilde{\psi}^{-1}(K) = \mathcal{W}$. Finalement nous obtenons bien que s est une partie gagnante.

- ▷ Si $\mathcal{I}$ est l'ensemble vide.
  Pour tout $u \leq s$, tel que u est fini, nous avons simul$(u) = u$ (voir la preuve du lemme 54). Nous en déduisons que s est une f-partie.
  Nous montrons alors que s est f-maximale. Considérons $c \in \Sigma_0$ avec $c \,\mathrm{I}\, \mathrm{Alph}_\infty(s)$. Nous pouvons écrire $s = uv$ avec u fini, Alph(min(v)) $\subseteq$ Change(u) et $c \,\mathrm{I}\, v$.
  Le cas $c \in$ Change(u) et $u \equiv_f r$ est impossible : en effet comme $c \,\mathrm{I}\, v$, si $c \in$ Change(u), alors $u = s_0 s_1 \cdots s_i$ pour un certain i et $u \equiv_f r$ entraînerait $\mathcal{I} \neq \emptyset$.
  Nous en déduisons donc que $f(u, c) = g(u, c) =$ stop car s est g-maximale. Nous en déduisons donc que s est une f-partie f-maximale, donc une partie gagnante.
- ▷ Si $0 < |\mathcal{I}| < \infty$.
  Nous écrivons $s = uv$ avec $u = t_0 t_1 \cdots t_{|\mathcal{I}|-1}$ et $v = \overline{s}$.
  - ◇ Si $v \neq \epsilon$, on considère c dans Σ_0 avec $c \,\mathrm{I}\, \mathrm{Alph}_\infty(s) = \mathrm{Alph}_\infty(v)$. Nous écrivons $v = v_1 v_2$ avec $v_1 \neq \epsilon$ fini et $c \,\mathrm{I}\, v_2$. Nous avons simul$(uv_1) = rv_1$ (voir la preuve du lemme 54). Le cas $rv_1 \equiv_f r$ est impossible car u se termine par le facteur $t_{max(\mathcal{I})}$. Nous en déduisons que $f(rv_1, c) = g(uv_1, c) =$ stop car $s = uv_1 v_2$ est g-maximale. Nous en déduisons que la f-partie rv est f-maximale, donc gagnante. Maintenant $\psi(r) = \psi(u)$, donc $uv \approx_\psi rv$ et comme rv est un élément de $\mathcal{W}$, on en déduit que uv est également un élément de $\mathcal{W}$. Nous en déduisons donc que s est une partie gagnante.
  - ◇ Si $v = \varepsilon$, alors s est une partie finie. et $(s, g) \equiv (r, f)$. Considérons c un élément de Σ_0.
    - ⋆ Si c est un élément de Change(simul(s)), comme simul$(s) \equiv_f r$ nous obtenons $f(r, c) = g(s, c) =$ stop puisque s est g-maximale.
    - ⋆ Si c n'est pas un élément de Change(simul(s)) = Change(s), alors $c \in \mathrm{stop}_g(s) \setminus \mathrm{Change}(s)$, mais $(s, g) \equiv (r, f)$, donc $\mathrm{stop}_g(s) \setminus \mathrm{Change}(s) = \mathrm{stop}_f(r) \setminus \mathrm{Change}(r)$, donc $c \in \mathrm{stop}_f(r)$, et donc $f(r, c) =$ stop.

    Nous en déduisons donc que r est une partie f-maximale, donc r est une partie gagnante donc un partie de $\mathcal{W}$. Mais $\psi(r) = \psi(s)$, donc s est également une partie de $\mathcal{W}$, donc finalement s est une partie gagnante.

Nous avons donc finalement montré que toute g-partie g-maximale est une partie gagnante. Ceci prouve donc que g est bien une stratégie gagnante ce qui termine de montrer que g vérifie bien toutes les propriétés du lemme 48.

Ceci nous permet donc de finir de montrer le lemme 47. De plus comme la f-compatibilité est une propriété transitive (si f'' est f'-compatible et f' est f-compatible, f'' est alors f-compatible), nous allons pouvoir considérer dans le reste de la preuve que nous sommes partis d'une stratégie f uniforme.

17.3.3 Borner la mémoire dans le cas série

Pour construire une stratégie dont nous connaîtrons une borne pour la mémoire, nous allons associer à chaque classe d'équivalence de $\equiv$ une stratégie sur un des sous-alphabets A ou B. Ensuite nous allons recoller ces stratégies et montrer que nous obtenons une stratégie distribuée gagnante dont nous pouvons calculer la mémoire à partir de la mémoire des différentes stratégies obtenues par induction. Ceci nous fournira donc une borne supérieure sur la mémoire nécessaire à une stratégie gagnante distribuée.

Nous considérons donc que la stratégie f est une stratégie gagnante distribuée et uniforme.

Nous considérons alors une classe d'équivalence ρ de $\equiv$ telle que ρ_f n'est pas vide.

Nous considérons enfin un élément r de ρ_f, et un alphabet $C \in \{A, B\}$ tel que $C \subseteq \text{Change}(\rho)$. Notons que $\text{Change}(r)$ ne dépend pas du choix de r dans ρ_f (cf. remarque 5), donc nous pouvons définir $\text{Change}(\rho) = \text{Change}(r)$. De même que l'on a étendu Σ en Σ', on étend C en $C' \subseteq \Sigma'$.

Nous définissons alors un nouveau jeu sur l'alphabet C :

Définition 20 (Jeux partiels).

$$G_{C,\rho,f} = (\Sigma_0 \cap C, \Sigma_1 \cap C, (Q_i)_{i\in\mathcal{P}}, (T_c)_{c\in C}, q^0 * \text{lw}(r), \mathcal{W}_{C,\rho,f})$$

avec $\mathcal{W}_{C,\rho,f} = [\{t \mid t \in (\varepsilon \cup \text{From}_f(r)) \cap \mathbb{R}(C', \text{D}')$ *et* $r \cdot t$ *est* f_C*-maximale*$\}]_{\approx_\psi}$

Il nous faut tout d'abord vérifier que ces jeux sont bien définis. C'est à dire que $G_{C,\rho,f}$ ne dépend pas du choix de r. Mais ce résultat est facile car $G_{C,\rho,f}$ ne dépend que de $q^0 * \text{lw}(r)$ et de $\text{From}_f(r)$ et comme f est uniforme, ces deux valeurs ne dépendent que de ρ et non pas du choix particulier de r.

Pour pouvoir utiliser l'induction sur ces jeux, nous avons également besoin de vérifier que $\mathcal{W}_{C,\rho,f}$ est un ensemble reconnaissable. Mais comme $\mathcal{W}_{C,\rho,f}$ est défini comme la saturation d'un ensemble par le morphisme ψ, c'est par définition un ensemble reconnaissable.

Nous définissons alors sur chacun de ces jeux une stratégie et nous montrons qu'elle est gagnante.

Définition 21 (Stratégies partielles).
Pour s *une trace de* $\mathbb{M}(C', D')$ *et* c *un élément de* $C \cap \Sigma_0$*, nous définissons* $g_{C,\rho,f}$ *par :*

$$g_{C,\rho,f}(s, c) = f(rs, c)$$

La première chose que nous avons à montrer est encore une fois que $g_{C,\rho,f}$ est bien défini, c'est à dire que $g_{C,\rho,f}$ ne dépend pas du choix particulier de r dans ρ. Comme la stratégie f est uniforme, pour tout r et r' dans ρ_f, $\text{From}_f(r) = \text{From}_f(r')$. Dans ces conditions, pour tout s dans $\mathbb{M}(C', D')$ et pour tout c dans $C \cap \Sigma_0$ nous avons :

$$\begin{array}{lllll} f(rs,c) = \text{stop} & \Leftrightarrow & \forall q,\ s(c,q) \notin \text{From}_f(r) = \text{From}_f(r') & \Leftrightarrow & f(r's,c) = \text{stop} \\ f(rs,c) = q & \Leftrightarrow & s(c,q) \in \text{From}_f(r) = \text{From}_f(r') & \Leftrightarrow & f(r's,c) = q \end{array}$$

Ceci prouve bien que $g_{C,\rho,f}$ ne dépend pas du choix particulier de r dans ρ.

Nous montrons maintenant que $g_{C,\rho,f}$ est une stratégie distribuée gagnante de $G_{C,\rho,f}$.

Tout d'abord, il est facile de vérifier que $g_{C,\rho,f}$ est bien une stratégie distribuée, car f l'est.

On considère s une $g_{C,\rho,f}$-partie $g_{C,\rho,f}$-maximale. Dans ces conditions rs est une f-partie f_C-maximale par définition de $g_{C,\rho,f}$. Nous en déduisons donc que s est un élément de $\mathcal{W}_{C,\rho,f}$ par définition de la condition de gain. Ceci prouve que s est une partie gagnante de $G_{C,\rho,f}$ et donc finalement $g_{C,\rho,f}$ est une stratégie gagnante pour l'équipe 0.

Nous allons maintenant utiliser l'hypothèse d'induction pour pouvoir considérer un ensemble de stratégies sur les sous-alphabets A et B.

En effet, pour tout ρ, pour tout $C \in \{A, B\}$ tel que $C \subseteq \text{Change}(\rho)$, il existe une stratégie gagnante distribuée $g'_{C,\rho,f}$, $g_{C,\rho,f}$-compatible pour l'équipe 0 sur le jeu $G_{C,\rho,f}$ à mémoire $\mu_{C,\rho,f}$ de taille au plus $\text{MSUB}(C)$ vérifiant les hypothèse de la proposition 34.

Maintenant que nous avons cet ensemble de stratégies, il ne nous reste plus qu'à les combiner pour obtenir une stratégie f' vérifiant les propriétés du lemme 44.

Nous allons construire la stratégie f' de la manière suivante :

- $f'(\varepsilon, a) = g'_{A,\rho(\varepsilon),f}(\varepsilon, a)$ pour $a \in \Sigma_0 \cap A$.
- $f'(\varepsilon, b) = g'_{B,\rho(\varepsilon),f}(\varepsilon, b)$ pour $b \in \Sigma_0 \cap B$.

Soit $s = s_1 \cdot s_2$ avec $s_2 \neq \varepsilon$ et $\text{Alph}(s_2) \subseteq C \subseteq \text{Change}(s_1)$ avec $C \in \{A, B\}$. Soit ρ la classe d'équivalence de (s_1, f').

- Si $c \in C \cap \Sigma_0$: $f'(s, c) = g'_{C,\rho,f}(s_2, c)$
- Si $c \in \Sigma_0 \setminus C$, soit ρ' la classe d'équivalence de (s, f') : Nous définissons alors $f'(s, c) = g'_{\Sigma\setminus C,\rho',f}(\varepsilon, c)$ si ρ'_f n'est pas vide et $f'(s, c) = \text{stop}$ sinon.

La définition de f' ainsi énoncée pose un problème. En effet, nous avons besoin de connaître la classe d'équivalence de (s_1, f'), puis dans le dernier cas, la classe d'équivalence de (s, f'). Ceci ne pose en fait pas de problème, car pour connaître la classe de (s_1, f'), nous n'avons pas besoin de connaître la fonction f', mais seulement la valeur de $\mu(s_1)$ et de $\text{stop}'_f(s_1) \setminus \text{Change}(s_1)$. La valeur de $\mu(s_1)$ ne dépend pas de f' et la valeur de $\text{stop}'_f(s_1) \setminus \text{Change}(s_1)$ a été calculé précédemment par induction. Pour la classe de (s, f'), encore une fois $\mu(s)$ ne dépend pas de f' et $\text{stop}'_f(s) \setminus \text{Change}(s) = \text{stop}'_f(s) \cap C = \text{stop}_{g'_{C,\rho,f}}(s_2)$. Finalement ρ' ne dépend pas non plus de f'.

De plus, le lemme 55 que nous énoncerons dans la suite montrera que le cas ρ'_f n'est pas possible.

Nous allons alors montrer qu'il s'agit d'une stratégie distribuée gagnante à mémoire μ' défini de la manière suivante :

$$\begin{aligned} \mu'(\varepsilon) &= (\mu(\varepsilon), \#, \#) \\ \mu'(s) &= (\mu(s), \rho, \mu_{C,\rho,f}(s_2)) \end{aligned}$$

Tout d'abord nous montrons que la fonction qui à une trace r de $\mathbb{M}(\Sigma', \mathrm{D}')$ associe $\mu'(r)$ est une mémoire distribuée. Pour ce faire, il suffit de vérifier que que μ' vérifie la proposition 34.

Soit r et s deux traces de $\mathbb{M}(\Sigma', D')$ telles que $\mu'(r) = \mu'(s)$. Si If $r = s = \varepsilon$, le résultat est vérifié. Sinon ni r ni s ne sont égales à ε.

Nous pouvons alors écrire $r = r_1 \cdot r_2$ avec $r_2 \neq \varepsilon$ et $\mathrm{Alph}(r_2) \subseteq C \subseteq \mathrm{Change}(r_1)$ avec $C \in \{A, B\}$. Nous écrivons de la même manière $s = s_1 \cdot s_2$ Notons que $\mathrm{Change}(r) = \mathrm{Change}(s)$ puisque $\mu(r) = \mu(s)$ (cf. remarque 5).

Nous montrons que pour tout $(c, q) \in \Sigma'$, $\mu'(r(c, q)) = \mu'(s(c, q))$.

Tout d'abord nous avons montré que μ vérifiait déjà cette propriété.

Maintenant si $c \notin \mathrm{Change}(r) = \mathrm{Change}(s)$, $\mu'(r(c, q)) = (\mu(r(c, q)), \rho, \mu_{C,\rho,f}(r_2(c, q)))$. Comme $\mu_{C,\rho,f}$ est une fonction asynchrone, et $\mu_{C,\rho,f}(r_2) = \mu_{C,\rho,f}(s_2)$, nous avons $\mu_{C,\rho,f}(r_2(c, q)) = \mu_{C,\rho,f}(s_2(c, q))$, donc $\mu'(r(c, q)) = \mu'(s(c, q))$.

Nous considérons maintenant le cas où $c \in \mathrm{Change}(r)$.

Nous avons alors $\mu'(r) = \mu'(s)$, donc $(r_1, f') \in \rho$, $(s_1, f') \in \rho$, $\psi(s_2) = \psi(r_2)$ et $\mu_{C,\rho,f}(r_2) = \mu_{C,\rho,f}(s_2)$.

Ces dernières égalités nous assurent que $\mathrm{stop}_{f'}(r) \setminus \mathrm{Change}(r) = \mathrm{stop}_{f'}(s) \setminus \mathrm{Change}(s)$. En effet $\mathrm{stop}_{f'}(r) \setminus \mathrm{Change}(r) = \mathrm{stop}_{f'}(r) \cap C = \mathrm{stop}_{g'_{C,\rho,f}}(r_2) = \mathrm{stop}_{g'_{C,\rho,f}}(s_2)$ car $\mu_{C,\rho,f}(r_2) = \mu_{C,\rho,f}(s_2)$, donc $\mathrm{stop}_{f'}(r) \setminus \mathrm{Change}(r) = \mathrm{stop}_{f'}(s) \setminus \mathrm{Change}(s)$.

Nous en déduisons donc donc $r \equiv_{f'} s$. Soit ρ' la classe d'équivalence de (r, f'), nous en déduisons que $\mu'(r(c, q)) = (\mu(r(c, q)), \rho', \mu_{C,\rho',f}((c, q))) = \mu'(s(c, q))$.

Finalement μ' vérifie donc les hypothèses de la proposition 34, donc μ' est une mémoire distribuée.

Nous allons maintenant montrer que nous pouvons calculer $f'(r, c)$ en connaissant c et la valeur de $\mu'(r)$. Ceci prouvera que f' est une stratégie distribuée à mémoire μ'.

Si $\mu'(r) = \mu'(\varepsilon)$, alors $r = \epsilon$, et nous avons les deux cas suivants :

1. Si $c \in A \cap \Sigma_0$, alors $f'(r, c) = g'_{A,\rho(\varepsilon),f}(\varepsilon, c)$
2. Si $c \in B \cap \Sigma_0$, alors $f'(r, c) = g'_{B,\rho(\varepsilon),f}(\varepsilon, c)$

Supposons maintenant que $r \neq \varepsilon$. Remarquons que connaissant $\mu'(r)$, nous connaissons $\mathrm{Change}(r)$. On pose $r = r_1 \cdot r_2$ avec $r_2 \neq \varepsilon$ et $\mathrm{Alph}(r_2) \subseteq C \subseteq \mathrm{Change}(r_1)$ avec $C \in \{A, B\}$. Dans ces conditions, nous savons que $\mu'(r) = (\mu(r), \rho, \mu_{C,\rho,f}(r_2))$ où ρ est la classe de (r_1, f'). Nous avons les deux cas suivants.

1. Si $c \notin \mathrm{Change}(r)$, alors $f'(r, c) = g'_{C,\rho,f}(r_2, c)$. Nous pouvons calculer cette valeur car $g'_{C,\rho,f}$ est une stratégie à mémoire $\mu_{C,\rho,f}$ et nous connaissons la valeur $\mu_{C,\rho,f}(r_2)$.
2. Si $c \in \mathrm{Change}(r)$, nous pouvons calculer la classe d'équivalence ρ' de (r, f') :
   - ▷ Nous connaissons $\mu(r)$.
   - ▷ $\mathrm{stop}_{f'}(r) \setminus \mathrm{Change}(r) = \mathrm{stop}_{f'}(r) \cap C = \mathrm{stop}_{g'_{C,\rho,f}}(r_2)$ que nous pouvons calculer en utilisant $\mu_{C,\rho,f}(r_2)$.

   Puis, finalement $f'(r, c) = g'_{\mathrm{Change}(r),\rho',f}(\varepsilon, c)$.

Finalement nous avons donc montré que f' était une stratégie distribuée à mémoire μ'.

Il nous reste à montrer que f' est f-compatible et gagnante. Pour ce faire nous avons besoin du lemme suivant qui relie les f'-parties aux f-parties.

Lemme 55 *Soit s une f'-partie et $s_0 s_1 \cdots$ sa factorisation par blocs. Alors, si s est finie, ou que sa factorisation est infinie, il existe une f-partie r dont la factorisation par blocs est $r_0 r_1 \cdots$ et pour tout i, $(s_0 s_1 \cdots s_i, f') \equiv (r_0 r_1 \cdots r_i, f)$, $\mu_0(r_i) = \mu_0(s_i)$ et $stop_{f'}(s_0 s_1 \cdots s_i) = stop_f(r_0 r_1 \cdots r_i)$.*

Nous remarquons que ce lemme montre que quelle que soit la classe d'équivalence ρ tel que $\rho_{f'}$ est non vide, ρ_f est également non vide. Donc dans la définition de f', ρ'_f n'est jamais vide.

Preuve Nous prouvons ce résultat par induction sur le nombre de facteurs de la factorisation par blocs de s.

- ▷ Si $s = \varepsilon$, alors $r = \varepsilon$ est solution. La seule propriété qu'il nous faut montrer est $\mathrm{stop}_f(\varepsilon) = \mathrm{stop}_{f'}(\varepsilon)$, mais $\mathrm{stop}_{f'}(\varepsilon) = \mathrm{stop}_{g'_{A,\rho(\varepsilon),f}}(\varepsilon) \uplus \mathrm{stop}_{g'_{B,\rho(\varepsilon),f}}(\varepsilon)$. Par $g_{A,\rho(\varepsilon),f}$-compatibilité de $g'_{A,\rho(\varepsilon),f}$, nous obtenons $\mathrm{stop}_{g'_{A,\rho(\varepsilon),f}}(\varepsilon) = \mathrm{stop}_{g_{A,\rho(\varepsilon),f}}(\varepsilon) = \mathrm{stop}_f(\varepsilon) \cap A$ et $\mathrm{stop}_{g'_{B,\rho(\varepsilon),f}} = \mathrm{stop}_f(\varepsilon) \cap B$. Au final $\mathrm{stop}_{f'}(\varepsilon) = \mathrm{stop}_f(\varepsilon)$.
- ▷ Si $s = s_0 \cdot s_1 \cdots s_i$, par induction, il existe $r_0 r_1 \cdots r_{i-1}$ qui vérifient les conditions. Posons ρ la classe d'équivalence de $(s_0 s_1 \cdots s_{i-1}, f')$. Par définition de f', s_i est une $g'_{C,\rho,f}$-partie où C est un élément de $\{A, B\}$ tel que $\mathrm{Alph}(s_i) \subseteq C$. Par compatibilité, comme s_i est une trace finie, il existe une $g_{C,\rho,f}$-partie r_i telle que $\mu_0(r_i) = \mu_0(s_i)$. Par définition de $g_{C,\rho,f}$, nous obtenons que $r_0 r_1 \cdots r_{i-1} r_i$ est une f-partie car $(r_0 r_1 \cdots r_{i-1}, f)$ appartient à la classe d'équivalence ρ.
  Maintenant, $\mu(r_0 \cdots r_{i-1}) = \mu(s_0 \cdots s_{i-1})$, donc $\varphi \circ \widetilde{\psi}(r_0 \cdots r_{i-1}) = \varphi \circ \widetilde{\psi}(s_0 \cdots s_{i-1})$. De plus $\psi(r_i) = \psi(s_i)$ et μ_0 et ψ sont des morphismes, donc par définition de μ nous avons $\mu(r_0 \cdots r_i) = \mu(s_0 \cdots s_i)$.
  Il reste à montrer que $\mathrm{stop}_{f'}(s_0 s_1 \cdots s_i) = \mathrm{stop}_f(r_0 r_1 \cdots r_i)$ ce qui revient à montrer que pour tout c dans Σ_0, $f(r_0 \cdots r_i, c) = \mathrm{stop} \Leftrightarrow f'(s_0 \cdots s_i) = \mathrm{stop}$.
  Si c est un élément de C, par définition de $g_{C,\rho,f}$, $f(r_0 \cdots r_i, c) = \mathrm{stop} \Leftrightarrow g_{C,\rho,f}(r_i, c) = \mathrm{stop}$. Par $g_{C,\rho,f}$-compatibilité de $g'_{C,\rho,f}$, $g_{C,\rho,f}(r_i, c) = \mathrm{stop} \Leftrightarrow g'_{C,\rho,f}(s_i, c) = \mathrm{stop}$, enfin par définition de f', $g'_{C,\rho,f}(s_i, c) = \mathrm{stop} \Leftrightarrow f'(s_0 \cdots s_i, c) = \mathrm{stop}$. Ceci prouve de plus que $(s_0 s_1 \cdots s_i, f') \equiv (r_0 r_1 \cdots r_i, f)$.
  Maintenant, si $c \notin C$, posons ρ' la classe d'équivalence de $(s_0 s_1 \cdots s_i, f')$. Nous avons $f'(s_0 s_1 \cdots s_i, c) = \mathrm{stop} \Leftrightarrow g'_{\Sigma\setminus C,\rho',f}(\varepsilon, c) = \mathrm{stop}$. Ceci est équivalent, par $g_{\Sigma\setminus C,\rho',f}$-compatibilité de $g'_{\Sigma\setminus C,\rho',f}$ à $g_{\Sigma\setminus C,\rho',f}(\varepsilon, c) = \mathrm{stop}$ ce qui est équivalent par définition de $g_{\Sigma\setminus C,\rho',f}$ à $f(r_0 r_1 \cdots r_i, c) = \mathrm{stop}$ puisque $(r_0 r_1 \cdots r_i, f)$ est dans la classe ρ'.
  Ceci termine l'induction et montre le résultat.

□

Ce lemme prouve, de plus, que f' est f-compatible.

Il ne nous reste donc plus qu'à montrer que f' est une stratégie gagnante. Pour ce faire nous considérons s une f'-partie f'-maximale, et nous allons montrer qu'il s'agit d'une partie gagnante. Nous posons $s = s_0 s_1 \cdots s_i \cdots$ la décomposition par blocs de s.

Nous séparons alors trois cas :

- ▷ Si la factorisation par bloc de s est infinie, nous posons $r = r_0 \cdots r_i \cdots$ la f-partie construite dans le lemme 55. Cette factorisation est infinie, donc r est f-maximale. La stratégie f est une stratégie gagnante de G, donc r est une partie gagnante dans G. Mais comme par construction, $r \approx_\psi s$, alors s est également une partie gagnante de G.
- ▷ Si s est une partie finie, sa factorisation est également finie. Soit $r = r_0 \cdots r_i$ la f-partie construite dans le lemme 55. La factorisation de r

est finie. Nous avons $\psi(r) = \psi(s)$ et comme s est f'-maximale, r est f-maximale. De ce fait, r est une partie gagnante car f est une stratégie gagnante, ce qui prouve que s est également une partie gagnante.

- ▷ Si s est une partie infinie, mais avec une factorisation finie $s_0 \cdots s_i$, considérons ρ la classe d'équivalence de $(s_0 \cdots s_{i-1}, f')$. soit $r_0 \cdots r_{i-1}$ la f-partie construire dans le lemmee 55. Dans ces conditions s_i est une $g'_{C,\rho,f}$-partie. Comme s est f'-maximale, s_i est $g'_{C,\rho,f}$-maximale, donc gagnante dans $G_{C,\rho,f}$. Par définition de $\mathcal{W}_{C,\rho,f}$, il existe r_i tel que $s_i \approx_\psi r_i$ et $r = r_0 \cdots r_{i-1} r_i$ est f-maximale donc gagnante. Mais dans ces conditions, $r \approx_\psi s$, donc s est également une partie gagnante.

Finalement nous avons donc trouvé une stratégie distribuée f' qui est gagnante et f-compatible. De plus sa mémoire est μ'. Nous pouvons calculer la taille de μ' à l'aide de la taille de μ, du nombre de classe d'équivalence de $\equiv$ et des mémoires des différentes stratégies partielles. La taille de μ est $|Q^\bullet| \times |N|^2 \times |M|$. Connaissant N, on peut borner la taille de M, donc la taille de μ est une constante S_μ par rapport à Σ. Le nombre de classe d'équivalence de $\equiv$ est égal à $S_\mu \times 2^\Sigma$. Enfin, on peut borner la taille des mémoires des différentes stratégies partielles par $\max(\mathrm{MSUB}(A), \mathrm{MSUB}(B))$.

Nous en déduisons finalement :

$$\mathrm{MSUB}(\Sigma) = S_\mu^2 \times 2^\Sigma \times \max(\mathrm{MSUB}(A), \mathrm{MSUB}(B))$$

17.4 Décider de l'existence d'une stratégie distribuée

Nous savons maintenant que pour tout jeu G sur une architecture série parallèle (Σ, D), s'il existe une stratégie distribuée gagnante pour l'équipe 0 dans G, il en existe une de mémoire inférieure à $\mathrm{MSUB}(\Sigma)$. Cette borne est calculable et connue.

Pour décider s'il existe une stratégie distribuée gagnante pour l'équipe 0 dans G, il suffit donc de considérer toutes les stratégies de mémoire $\mathrm{MSUB}(\Sigma)$. Étant donné une mémoire nous avons vu au chapitre 16 que nous pouvions décider s'il existait une stratégie gagnante pour l'équipe 0 utilisant cette mémoire. Nous pouvons donc vérifier si une de ces mémoires permet aux joueurs de gagner. Si c'est le cas, nous avons prouvé l'existence d'une stratégie gagnante et nous l'avons même calculée. Dans le cas contraire, nous savons qu'il ne peut pas exister de stratégie gagnante distribuée pour l'équipe 0 dans le jeu G.

Cinquième partie

Conclusion

Chapitre 18

Bilan

Au cours de nos travaux, nous nous sommes intéressés à différents aspects des problèmes de spécification et modélisations des systèmes distribués.

Nous nous sommes tout d'abord intéressés au problème de spécification, en particulier à la spécification à l'aide de logiques temporelles. Nous sommes partis de la constatation que les logiques locales ont, dans la très grande majorité des cas, un problème de satisfaisabilité dans PSpace alors que les logiques globales sont souvent indécidables ou ont un problème de satisfaisabilité de forte complexité. Nous avons alors montré que ISTL$^\diamond$, qui est une logique globale dont la complexité est relativement faible, pouvait en fait se réduire à une logique locale. Ce résultat explique la faible complexité de ISTL$^\diamond$. Nous avons alors étudié une nouvelle logique globale : la logique des filtres. Cette logique est proche de ISTL$^\diamond$, mais un peu plus expressive. Nous espérions trouver une complexité pour le problème de satisfaisabilité analogue à celle de ISTL$^\diamond$. Malheureusement, le seul algorithme que nous avons trouvé pour la satisfaisabilité de cette logique est non élémentaire.

Dans l'espoir de trouver un meilleur algorithme pour la satisfaisabilité de la logique des filtres, nous avons recherché à étendre le concept d'alternance aux automates asynchrones. Nous avons donc introduit les automates asynchrones cellulaires alternants. Le principal intérêt des automates alternants sur les mots est la facilité avec laquelle on peut les complémenter. Malheureusement, nous nous sommes rendus compte que la procédure ne s'étendait pas aux automates asynchrones cellulaires alternants. Nous nous sommes alors intéressés au pouvoir d'expression de ces automates afin de vérifier s'il pouvait être plus important que celui des automates asynchrones cellulaires. Nous avons alors montré que ce n'était pas le cas sur les alphabets séries parallèles.

Enfin nous nous sommes intéressés à la synthèse de contrôleurs. Ce problème se réduisant à un problème de théorie des jeux, nous avons introduit un nouveau type de jeu. Ce sont des jeux distribués dans lesquels deux équipes de joueurs s'affrontent. A un moment donné plusieurs joueurs de plusieurs équipes peuvent avoir la possibilité de jouer. Nous avons alors défini la notion de stratégie distribuée : il s'agit de stratégies qui ne peuvent pas prendre en compte des événements se déroulant simultanément. Pour se faire, nous avons dû définir différentes notions de mémoires. En particulier nous avons envisagé le cas des stratégies sans mémoire dans lesquelles un coup ne peut dépendre que de l'état

du système au moment de le jouer. Nous avons également envisagé le cas de la mémoire causale où chaque coup peut dépendre de l'intégralité de l'historique de la partie. Cette mémoire considère également que le maximum d'informations est échangé entre les joueurs à chaque coup. Nous avons alors montré que ces jeux n'étaient pas déterminés : c'est à dire qu'il existe des jeux pour lesquels aucune des deux équipes n'a de stratégie gagnante distribuée.

Dans le cas des stratégies sans mémoire, nous avons décrit un algorithme permettant de calculer, si elle existe, une stratégie gagnante. Nous avons également montré comment se ramener au cas d'une stratégie sans mémoire quand nous recherchons une stratégie à mémoire finie dont nous connaissons à l'avance la mémoire.

Dans le cas des stratégies à mémoire causale, nous avons montré que nous pouvions décider s'il existait une stratégie gagnante dans le cadre des alphabets séries parallèles. Nous avons en outre montré que s'il y a une stratégie gagnante, cette stratégie utilise une mémoire finie.

Chapitre 19

Travaux futurs

Ces travaux présentent une étude de divers outils permettant la modélisation et la spécification des systèmes distribués. Cette étude est bien loin d'être terminée, mais les résultats que nous avons obtenus permettent de jalonner le terrain.

Tout d'abord dans le domaine de la spécification à l'aide de logiques temporelles, le résultat de [21] nous permet d'utiliser les logiques locales de manière efficace. D'un autre côté les logiques globales semblent très difficile d'usage et quand elles sont utilisables, elles sont souvent assez peu expressives. Ces considérations nous poussent à penser que, dans le cadre de la spécification avec des logiques temporelles, l'utilisation de logiques globales semble peut adéquat. D'autant plus qu'il est tout à fait possible d'introduire des opérateurs pseudo globaux dans une logique locale en utilisant le cadre de [21].

Dans le cadre de la modélisation des systèmes distribués à l'aide d'automates asynchrones cellulaires alternants, le résultat trouvé demande à être généralisé aux alphabets non séries parallèles. Pour ce faire, il faudra cependant trouver une nouvelle technique de preuve car l'induction présentée dans cette thèse est fortement dépendante des propriétés de l'alphabet. De plus, le fait que le langage de l'automate dual n'est pas le langage complémentaire de l'automate de départ rend cet outil difficile à utiliser pour la modélisation. Il n'est pas toujours facile de bien réaliser quel langage représente un automate donné.

C'est dans le cadres des jeux distribués que les perspectives sont les plus vastes.

Tout d'abord, il faudrait étendre le résultat présenté dans ce mémoire aux alphabets quelconques. Encore une fois, la méthode de démonstration devra être adaptée. En effet, sur les alphabets séries parallèles, nous pouvions découper les parties de façon à avoir une vue globale du début de la partie après chaque coupure. Ce type de propriété ne s'étend pas directement sur des alphabets quelconques.

Ensuite, ces jeux étant non déterminés, il existe des jeux pour lesquels ni l'équipe 0, ni l'équipe 1 n'ont de stratégie gagnante. Dans ces conditions, nous sommes dans un cadre très proche des jeux à information incomplète. En effet, la non détermination des jeux distribués est due a fait que, au cours de la partie, les joueurs ne connaissent pas exactement l'état du jeu. Dans ces conditions nous pouvons chercher à réutiliser les techniques et les résultats existants pour les jeux

à informations partielles. En particulier on pourrait introduire des stratégies probabilistes en espérant pouvoir calculer les stratégies offrant la plus grande probabilité de gain.

Bibliographie

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP '89 : Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Sciences*, pages 1–17. Springer-Verlag, 1989.

[2] R. Alur, K. L. McMillan, and D. Peled. Deciding global partial-order properties. In *ICALP '98 : Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Sciences*, pages 41–52. Springer-Verlag, 1989.

[3] R. Alur, D. Peled, and W. Penczek. Model-checking of causality properties. In *LICS '95 : Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 90–100. IEEE Computer Society Press, 1995.

[4] A. Anuchitanukul and Z. Manna. Realizability and synthesis of reactive modules. In *CAV '94 : Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Sciences*, pages 156–169. Springer-Verlag, 1994.

[5] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings of the IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[6] J. Bernet, D. Janin, and I. Walukiewicz. Private communication. 2004.

[7] J. Brzozowski and E. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Computer Science*, 10 :19–35, 1980.

[8] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138 :295–311, 1969.

[9] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1) :114–133, 1981.

[10] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2) :244–263, 1986.

[11] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[12] R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106(2) :159–202, 1993.

[13] L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. The control of synchronous systems. In *CONCUR '00 : Proceedings of the 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Sciences*, pages 458–473. Springer-Verlag, 2000.

[14] S. Demri and P. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1) :84–103, 2002.

[15] V. Diekert and P. Gastin. An expressively complete temporal logic without past tense operators for mazurkiewicz traces. In *CSL '99 : Proceedings of the 13th International Workshop and 8th Annual Conference of the EACSL on Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Sciences*, pages 188–203. Springer-Verlag, 1999.

[16] V. Diekert and P. Gastin. Local temporal logic is expressively complete for cograph dependence alphabets. *Information and Computation*, 195 :30–52, 2004.

[17] V. Diekert and Y. Métivier. *Handbook of Formal languages*, volume 3, chapter Partial Commutations and Traces, pages 457–533. Springer-Verlag, 1997.

[18] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

[19] M. Droste, P. Gastin, and D. Kuske. Asynchronous cellular automata for pomsets. *Theoretical Computer Science*, 247(1) :1–38, 2000.

[20] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *POPL '80 : Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 163–173. ACM Press, 1980.

[21] P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *CONCUR '03 : Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Sciences*, pages 222–236. Springer-Verlag, 2003.

[22] P. Gastin, R. Meyer, and A. Petit. A non-elementary modular decision procedure for LTrL. In *MFCS '98 : Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Sciences*, pages 356–365. Springer-Verlag, 1998.

[23] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV '01 : Proceedings of the 13th Conference on Computer Aided Verification*, number 2102 in Lecture Notes in Computer Sciences, pages 53–65. Springer-Verlag, 2001.

[24] P. Gastin and A. Petit. Asynchronous cellular automata for infinite traces. In *ICALP '92 : Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Sciences*, pages 583–594. Springer-Verlag, 1992.

[25] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV '95 : Proceedings of the 15th IFIP TC6/WG6.1 International Symposium on Protocol Specification, Testing and Verification*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1996.

[26] Y. Gurevich and L. Harrington. Automata, trees and games. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 60–65. ACM Press, 1982.

[27] J. G. Henriksen. An expressive extension of TLC. *International Journal of Foundations of Computer Science*, 13(3) :341–360, 2002.

[28] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News*, 32(1) :60–65, 2001.

[29] H. W. Kamp. *Tense logic and the theory of linear order*. PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[30] S. Katz and D. Peled. Interleaving set temporal logic. In *Proceedings of Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Sciences*, pages 21–43. Springer-Verlag, 1987.

[31] S. Katz and D. Peled. Interleaving set temporal logic. *Theoretical Computer Science*, 75(3) :263–287, 1990.

[32] O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi. Open systems in reactive environments : Control and synthesis. In *CONCUR '00 : Proceedings of the 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Sciences*, pages 92–107. Springer-Verlag, 2000.

[33] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *LICS '01 : Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 389–398. IEEE Computer Society Press, 2001.

[34] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, volume 193 of *Lecture Notes in Computer Sciences*, pages 196–218. Springer-Verlag, 1985.

[35] C. Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS '99 : Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Sciences*, pages 97–109. Springer-Verlag, 1999.

[36] P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *ICALP '01 : Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Sciences*, pages 396–407. Springer-Verlag, 2001.

[37] P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR '02 : Proceedings of the 13th International Conference on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Sciences*, pages 145–160. Springer-Verlag, 2002.

[38] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2) :149–184, 1993.

[39] S. Mohalik and I. Walukiewicz. Distributed games. In *FSTTCS '03 : Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Sciences*. Springer-Verlag, 2003.

[40] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science*, 13(1) :85–108, 1981.

[41] W. Penczek. On undecidability of propositional temporal logics on trace systems. *Information Processing Letters*, 43(3) :147–153, 1992.

[42] J. Pin and D. Perrin. *Infinite words. Automata, Semigroups, Logic and Games*. Elsevier, 2004.

[43] A. Pnueli. The temporal logic of programs. In *FOCS '77 : Proceedings of the 18th Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[44] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP '89 : Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Sciences*, pages 652–671. Springer-Verlag, 1989.

[45] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthetize. In *FOCS '90 : Proceedings of the 31st Symosium on Foundations of Computer Science*, pages 746–757. IEEE Computer Society Press, 1990.

[46] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Sciences*, pages 337–351. Springer-Verlag, 1982.

[47] P. Ramadge and W. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77, pages 81–98, 1989.

[48] M. Sipser. Introduction to the theory of computation. *SIGACT News*, 27(1) :27–29, 1996.

[49] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the Association for Computing Machinery*, 32(3) :733–749, 1985.

[50] P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for mazurkiewicz traces. In *LICS '97 : Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 183–194. IEEE Computer Society Press, 1997.

[51] J. G. Thistle and W. M. Wonham. Supervision of infinite behavior of discrete-event systems. *SIAM Journal On Control and Optimization*, 32(4) :1098–1113, 1994.

[52] W. Thomas. On logical definability of traces languages. In *Proceedings of the workshop of ESPRIT BRA 3166, ASMICS*, pages 172–182, 1990.

[53] W. Thomas. On the synthesis of strategies in infinite games. In *STACS '95 : Proceedings of 12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Sciences*, pages 1–13. Springer-Verlag, 1995.

[54] W. Thomas. Infinite games and verification. In *CAV '02 : Proceedings of the 6th International Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Sciences*, pages 58–64. Springer-Verlag, 2002.

[55] M. Vardi. Alternating automata and program verification. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Sciences*, pages 471–485. Springer-Verlag, 1995.

[56] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *CAV '95 : Proceedings of the 7th International Conference on Computer Aided Verification*, volume 939 of *Lecture Notes in Computer Sciences*, pages 267–278. Springer-Verlag, 1995.

[57] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, volume 1043 of *Lecture Notes in Computer Sciences*, pages 238–266. Springer-Verlag, 1996.

[58] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV '00 : Proceedings of the 12th Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Sciences*, pages 202–215. Springer-Verlag, 2000.

[59] I. Walukiewicz. Difficult configurations - on the complexity of LTrL. In *ICALP '98 : Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Sciences*, pages 140–151, 1998.

[60] P. Wolper. Constructing automata from temporal logic formulas : A tutorial. In *Lectures on Formal Methods and Performance Analysis, First EEF/Euro Summer School on Trends in Computer Science*, volume 2090 of *Lecture Notes in Computer Sciences*, pages 261–277. Springer-Verlag, 2002.

[61] W. Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications*, 21 :99–135, 1987.

[62] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In *Proceedings of the Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Sciences*, pages 278–289. Springer-Verlag, 1989.

[63] W. Zielonka. Asynchronous automata. In *Book of Traces [18]*, pages 175–217. World Scientific, 1995.

[64] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2) :135–183, 1998.

Index