

Compilation de connaissances avec automates à intervalles et applications à la planification

Alexandre Niveau¹, H el ene Fargier², C edric Pralet¹, and G erard Verfaillie¹

¹ ONERA/DCSD Toulouse 2, avenue  douard Belin 31400 Toulouse {alexandre.niveau, cpralet, verfail}@onera.fr

² IRIT-CNRS/RPDMP Universit  Paul Sabatier, Toulouse fargier@irit.fr

R esum  : La planification pour un syst me autonome se heurte   des probl mes de m moire et de puissance de calcul, que l'on peut r esoudre gr ce   la compilation de connaissances (Darwiche, 2001; Cadoli & Donini, 1998; Selman & Kautz, 1996; del Val, 1994). Elle consiste   transformer hors-ligne un probl me vers une forme facilement exploitable en ligne. Dans cet article, nous introduisons de nouvelles structures, bas es sur la notion d'*automate   intervalles* (IA, *interval automaton*), adapt es   la compilation de probl mes faisant intervenir des variables discr tes et/ou continues, et plus sp cialement de politiques de d cision.

Les automates   intervalles peuvent  tre vus comme une g n ralisation des diagrammes de d cision binaires (BDDs, *binary decision diagrams*), dans la mesure o  il s'agit de graphes acycliques orient s, poss dant une seule racine, et dont les n uds sont  tiquet s par des variables ; cependant les automates   intervalles sont des structures non-d terministes, leurs arcs sont  tiquet s par des intervalles (ferm s), et la multiplicit  des n uds n'est pas limit e   deux.

Nous  tudions ici la complexit  des requ tes et transformations classiquement consid r es lors de l'examen d'un nouveau langage de compilation. Nous montrons qu'un sous-langage particulier des automates   intervalles, ceux satisfaisant la propri t  de *convergence* (FIAs, *focusing interval automata*), ont des capacit s th oriques tr s proches de celles des DNNFs ; ils autorisent notamment les principales op rations n cessaires   l'exploitation de politiques de d cision en ligne.

Des r sultats exp rimentaux sont pr sent s de mani re   soutenir ces affirmations.

Mots-cl s : compilation de connaissances, variables continues, diagrammes de d cision, syst mes autonomes

1 Introduction

Un syst me autonome doit  tre capable de prendre des d cisions automatiquement, en fonction de ses observations et objectifs. Effectuer ces t ches compl tement *en ligne*, c'est- -dire seulement avec les capacit s de calcul embarqu es, peut compromettre la *r activit * du syst me. D'un autre c t , la taille limit e de la m moire embarqu e emp che de stocker les d cisions correspondant   toutes les situations possibles.

Une fa on de r esoudre cette contradiction est d'utiliser la *compilation de connaissances*, qui consiste   transformer le probl me de planification hors-ligne de mani re   faciliter sa r solution en ligne. Cette transformation peut consister   le r esoudre int gralement hors-ligne, puis exprimer la politique de d cision obtenue dans un certain *langage-cible de compilation*. La forme compil e doit alors  tre aussi compacte que possible, pour respecter les contraintes dues   la m moire, et aussi facilement exploitable que possible, de mani re   ce que les op rations n cessaires (qui d pendent de ce qu'on veut faire) puissent  tre effectu es efficacement en ligne.

Des langages-cibles de compilation efficaces ont  t  propos s pour des domaines de planification faisant intervenir des variables   domaines bool ens ou  num r s (par exemple les OBDDs (Bryant, 1986), les automates    tats finis (Vempaty, 1992), les DNNFs (Darwiche & Marquis, 2002), etc.). Cependant, de nombreuses applications r elles utilisent des variables continues ou   grand domaine  num r , telles que le temps ou l' nergie. L'objectif de cet article est de d finir de nouveaux langages-cibles permettant de faire de la planification hybride, c'est- -dire faisant intervenir simultan ment variables continues et discr tes. On voudrait en particulier pouvoir repr senter et exploiter en ligne des politiques de d cision.

Tout d'abord, nous définissons formellement les automates à intervalles dans la section 2. Nous étudions ensuite diverses opérations en section 3. La façon dont nous construisons des automates à intervalles est présentée section 4. Enfin, des résultats expérimentaux sont fournis section 5. Les preuves des propositions sont rassemblées à la fin de l'article.

2 Automates à intervalles

2.1 Structure et sémantique

Définition 1 (Automate à intervalles). *Un automate à intervalles (IA, interval automaton) est un couple $\phi = \langle X, \Gamma \rangle$, avec :*

- X (noté $\text{Var}(\phi)$) un ensemble fini et totalement ordonné de variables réelles, dont les domaines sont représentables par une union finie d'intervalles fermés de \mathbb{R} ;
- Γ un graphe acyclique orienté, possédant au plus une racine et au plus une feuille (appelée puits), dont les nœuds internes sont étiquetés par une variable de X ou par le symbole de disjonction \vee (que l'on traitera comme une variable particulière), et dont les arcs sont étiquetés par un intervalle fermé de \mathbb{R} . Les arcs sortants de nœuds \vee peuvent seulement être étiquetés soit par \mathbb{R} , soit par \emptyset .

Cette définition autorise un automate à intervalles à être vide (aucun nœud) ou à contenir un seul nœud (qui est à la fois racine et puits), et assure que tout arc appartient à au moins un chemin de la racine au puits. La figure 1 donne un exemple d'automate à intervalles.

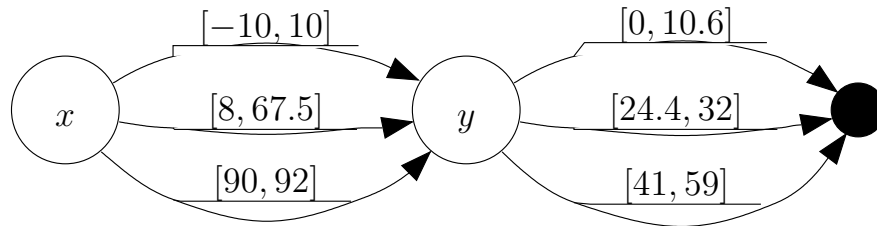


FIGURE 1 – Un exemple d'automate à intervalles. Son ensemble de modèles (voir déf. 2), représenté comme union de boîtes, est $[-10, 10] \times [0, 10.6] \cup [-10, 10] \times [24.4, 32] \cup [-10, 10] \times [41, 59] \cup [8, 67.5] \times [0, 10.6] \cup [8, 67.5] \times [24.4, 32] \cup [8, 67.5] \times [41, 59] \cup [90, 92] \times [0, 10.6] \cup [90, 92] \times [24.4, 32] \cup [90, 92] \times [41, 59]$.

Pour $x \in X$, $\text{Dom}(x) \subseteq \mathbb{R}$ représente le *domaine* de x , qui peut être énuméré ($\text{Dom}(x) = \{1, 3, 56, 4.87\}$) ou continu ($\text{Dom}(x) = [1, 7] \cup [23.4, 28]$). Par convention, $\text{Dom}(\vee) = \mathbb{R}$. On appelle « boîte » un produit cartésien d'intervalles. Pour $Y = \{y_1, \dots, y_k\} \subseteq X$, tel que les y_i sont rangés par ordre croissant, $\text{Dom}(Y)$ représente $\text{Dom}(y_1) \times \dots \times \text{Dom}(y_k)$, et on note \vec{y} une Y -*instanciation* de variables de Y , c'est-à-dire que $\vec{y} \in \text{Dom}(Y)$. Quand $Y \cap X = \emptyset$, $\vec{x} \cdot \vec{y}$ est la concaténation de \vec{x} et \vec{y} . Enfin, on note $\vec{y}(y_i)$ la valeur assignée à y_i dans \vec{y} .

Soit $\phi = \langle X, \Gamma \rangle$ un automate à intervalles, N un nœud et E un arc de Γ . On peut alors définir les éléments suivants :

- $\text{Root}(\phi)$ la racine de Γ et $\text{Sink}(\phi)$ son puits ;
- $|\phi|$ la *taille* de Γ , c'est-à-dire son nombre d'arcs ;
- $\text{Out}_\phi(N)$ (resp. $\text{In}_\phi(N)$) l'ensemble des arcs sortants de (resp. entrants) de N ;
- $\text{Var}_\phi(N)$ la variable étiquetant N (par convention $\text{Var}_\phi(\text{Sink}(\phi)) = \vee$) ;
- $\text{Src}_\phi(E)$ le nœud duquel arrive E et $\text{Dest}(E)$ le nœud vers lequel pointe E ;
- $\text{Itv}_\phi(E)$ l'intervalle étiquetant E ;
- $\text{Var}_\phi(E) = \text{Var}_\phi(\text{Src}(E))$ la variable associée à E .

Quand il n'y a pas d'ambiguïté, nous oublierons les références à ϕ dans les indices des notations.

Un automate à intervalles peut être vu comme une représentation compacte d'une fonction booléenne à variables discrètes ou continues. Cette fonction est la *fonction d'interprétation* de l'automate :

Définition 2 (Sémantique d'un automate à intervalles). *Un automate à intervalles ϕ sur X (c'est-à-dire tel que $\text{Var}(\phi) = X$) représente une fonction de $\text{Dom}(X)$ vers $\{\top, \perp\}$. Cette fonction, appelée sa fonction d'interprétation $I(\phi)$, est définie comme suit : pour toute X -instanciation \vec{x} , $I(\phi)(\vec{x}) = \top$ si et seulement s'il existe un chemin p de la racine au puits de ϕ tel que pour tout arc E sur p , soit $\text{Var}(E) = \vee$ et $\text{Itv}(E) \neq \emptyset$, soit $\vec{x}(\text{Var}(E)) \in \text{Itv}(E)$.*

On dit que \vec{x} est un modèle de ϕ quand $I(\phi)(\vec{x}) = \top$. On note $\text{Mod}(\phi)$ l'ensemble des modèles de ϕ . ϕ est dit être équivalent à un autre IA ψ (ce que l'on note $\phi \equiv \psi$) ssi $\text{Mod}(\phi) = \text{Mod}(\psi)$.

Notons que la fonction d'interprétation de l'automate vide renvoie toujours \perp , puisqu'un IA vide ne contient aucun chemin de la racine au puits. Inversement, la fonction d'interprétation de l'automate à un seul nœud (automate-puits) renvoie toujours \top , étant donné que dans cet IA, le seul chemin depuis la racine jusqu'au puits ne contient aucun arc. Remarquons également le rôle particulier des nœuds \vee : ce sont tout simplement des nœuds de disjonction, qui ne posent de condition sur aucune variable. Nous pouvons à présent introduire d'utiles définitions :

Définition 3 (Satisfiabilité, validité, contexte). *Soit ϕ un automate à intervalles sur X .*

ϕ est dit satisfiable (resp. valide) si et seulement si $\text{Mod}(\phi) \neq \emptyset$ (resp. $\text{Mod}(\phi) = \text{Dom}(X)$).

Une valeur $\omega \in \mathbb{R}$ est satisfaisante pour une variable $y \in X$ dans ϕ si et seulement s'il existe une instanciation $\vec{x} \in \text{Mod}(\phi)$ telle que $\vec{x}(y) = \omega$.

L'ensemble de toutes les valeurs satisfaisantes pour y dans ϕ est appelé le contexte de y dans ϕ , et noté $\text{Ctx}_\phi(y)$.

Nous verrons dans la suite que décider si un IA est satisfiable est un problème difficile. Une des raisons à cela est que les intervalles n'ont pas une structure hiérarchisée : sur un chemin donné, les intervalles associés à une variable donnée peuvent s'élargir après avoir réduit, et inversement. Ils peuvent même être en conflit les uns avec les autres (d'où la difficulté de la requête de satisfiabilité). Nous allons donc considérer des IAs *convergen*ts, c'est-à-dire dans lesquels les intervalles peuvent seulement réduire de la racine au puits.

Définition 4 (Automate à intervalles convergent). *Un arc convergent d'un automate à intervalles ϕ est un arc E tel que tous les arcs E' sur un chemin quelconque de la racine de ϕ à $\text{Src}(E)$ tels que $\text{Var}(E) = \text{Var}(E')$ vérifient $\text{Itv}(E) \subseteq \text{Itv}(E')$.*

*Un automate à intervalles convergent (FIA, focusing interval automaton) est un IA ne contenant que des arcs convergen*ts.

Un exemple de FIA se trouve fig. 2.

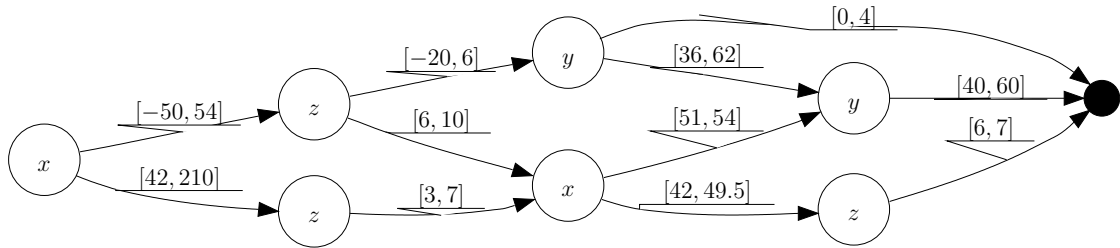


FIGURE 2 – Un exemple d'automate à intervalles convergent. Les domaines des variables sont les suivants : $\text{Dom}(x) = [0, 100]$, $\text{Dom}(y) = [0, 100]$ et $\text{Dom}(z) = \{0, 3, 7, 10\}$.

Comme suggéré fig. 1, la taille d'un IA peut être exponentiellement inférieure à celle de son ensemble de modèles (exprimé sous forme d'union de produits cartésiens, ou « boîtes »). Cela est notamment dû au fait que les IAs peuvent être *réduits* en supprimant les redondances, à la manière des BDDs et NNFs. Avant de présenter cette opération de réduction plus en détail, éclairons les relations entre les IAs et ces autres langages-cibles.

2.2 Relations avec les BDDs et autres langages-cibles

Introduits par Bryant dans (Bryant, 1986), les diagrammes de décision binaires (BDDs, *Binary Decision Diagrams*) sont des graphes acycliques orientés à une seule racine, qui représentent des fonctions booléennes à variables booléennes. Ils ont exactement deux feuilles, étiquetées respectivement \top et \perp ; leurs

nœuds internes sont étiquetés par une variable booléenne et ont exactement deux arcs sortants, également étiquetés respectivement \top et \perp (ou de manière équivalente 1 et 0). Un BDD libre (FBDD, *free BDD*) est un BDD satisfaisant la propriété de lecture unique (*read-once* : tout chemin contient au plus une occurrence de chaque variable). Quand on impose un même ordre sur les variables le long de tous les chemins, on obtient un BDD ordonné (OBDD).

Les automates à intervalles sont, en un sens, une généralisation des BDDs. La fonction d'interprétation de ces derniers est en effet similaire à celle des IAs : pour une instantiation donnée des variables Booléennes, la fonction retourne \top si et seulement s'il existe un chemin de la racine à la feuille étiquetée \top , tel que l'instanciation donnée est cohérente avec tout arc sur le chemin.

Nous voyons donc que, quand on interprète un BDD, il est possible de ne pas tenir compte de la feuille \perp . Si nous retirons cette feuille (et les arcs y pointant, et les nœuds sans fils...), nous obtenons un IA dont les arcs sont étiquetés par $[0, 0]$ ou $[1, 1]$:

Proposition 5 (Correspondance entre IAs et BDDs). *Tout BDD peut être exprimé sous la forme d'un IA équivalent, en temps linéaire en la taille du BDD.*

Cette *traduisibilité linéaire* nous aidera à prouver certaines propositions. On peut utiliser la même technique pour traduire n'importe quel FBDD ou OBDD en FIA :

Proposition 6 (Correspondance entre FIAs et FBDDs). *Tout FBDD (et a fortiori tout OBDD) peut être exprimé sous la forme d'un FIA équivalent, en temps linéaire en la taille du FBDD.*

La principale différence entre la famille des IAs et celle des BDDs est qu'un automate à intervalles n'est pas forcé d'être déterministe (la même solution peut être vérifiée sur plusieurs chemins du graphe), et évidemment que les IAs ne se limitent pas aux variables Booléennes. Les automates de Vempaty (Vempaty, 1992; Amilhastre *et al.*, 1999) supportent également les domaines non-Booléens, mais sont restreints à des domaines finis. Ce sont de plus des structures ordonnées, à la manière des OBDDs, ce qui est également le cas des diagrammes à intervalles (Strehl & Thiele, 1998). Il est intéressant de noter que les FIAs n'ont pas une structure décomposable au sens des DNNFs, mais conservent l'essence de cette propriété : ils sont *linkless* (Murray & Rosenthal, 2003) — dans un FIA, une restriction sur une variable peut se répéter sur un même chemin (en terme de NNFs, des deux côtés d'un nœud ET), mais la seconde restriction ne peut être en conflit avec la première, à l'exception notable des arcs \emptyset , qui sont retirés par l'opération de réduction que nous allons voir maintenant.

2.3 Réduction

Comme pour un BDD, il est possible de réduire la taille d'un IA sans changer sa sémantique, en fusionnant certains arcs et nœuds. Le format de *graphe*, contrairement à celui d'arbre, permet de factoriser les sous-graphes identiques. Les opérations de réduction introduites ci-après sont basées sur la notion de nœuds isomorphes, bégayants et non décisifs, et d'arcs contigus et infranchissables. Certaines de ces notions sont de simples généralisations des définitions introduites dans le contexte des BDDs (Bryant, 1986), tandis que d'autres sont spécifiques aux automates à intervalles.

Définition 7 (Nœuds isomorphes). *Deux nœuds internes N_1, N_2 d'un IA ϕ sont isomorphes si et seulement si*

- $\text{Var}(N_1) = \text{Var}(N_2)$;
- *il existe une bijection σ de $\text{Out}(N_1)$ vers $\text{Out}(N_2)$, telle que $\forall E \in \text{Out}(N_1), \text{Itv}(E) = \text{Itv}(\sigma(E))$ et $\text{Dest}(E) = \text{Dest}(\sigma(E))$.*

Les nœuds isomorphes sont redondants, puisqu'ils représentent la même fonction ; un seul d'entre eux est nécessaire (voir figure 3).

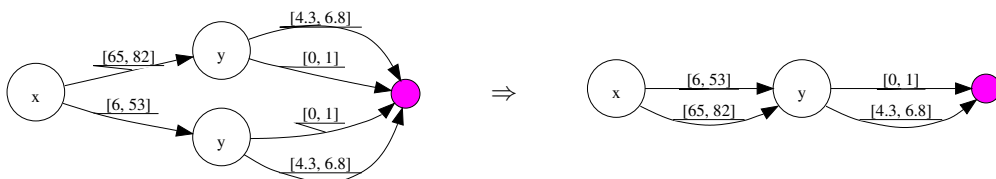


FIGURE 3 – Fusion de nœuds isomorphes.

Définition 8 (Nœud bégayant). *Un nœud non-racine N d'un IA ϕ est bégayant si et seulement si tous les nœuds parents de N sont étiquetés par $\text{Var}(N)$, et soit $|\text{Out}(N)| = 1$, soit $|\text{In}(N)| = 1$.*

Les nœuds bégayants sont inutiles, car l'information qu'ils apportent peut être facilement reportée sur leurs parents (voir figure 4).



FIGURE 4 – Fusion de nœuds bégayants.

Définition 9 (Nœud non décisif). *Un nœud N d'un IA ϕ est non décisif si et seulement si $|\text{Out}(N)| = 1$ et $E \in \text{Out}(N)$ est tel que $\text{Dom}(\text{Var}(E)) \subseteq \text{Itv}(E)$.*

Un nœud non décisif ne restreint pas les solutions correspondant aux chemins auxquels il appartient ; il est toujours franchi « automatiquement » (voir figure 5).



FIGURE 5 – Élimination de nœuds non décisifs.

Définition 10 (Arcs contigus). *Deux arcs E_1 et E_2 d'un IA ϕ sont contigus si et seulement si*

- $\text{Src}(E_1) = \text{Src}(E_2)$;
- $\text{Dest}(E_1) = \text{Dest}(E_2)$;
- *il existe un intervalle $I \subseteq \mathbb{R}$ tel que $I \cap \text{Dom}(\text{Var}(E_1)) = (\text{Itv}(E_1) \cup \text{Itv}(E_2)) \cap \text{Dom}(\text{Var}(E_1))$.*

Les arcs contigus proviennent du même nœud, pointent vers le même nœud, et ne sont pas disjoints (modulo le domaine de leur variable) : on peut les remplacer par un seul arc (voir figure 6). Par exemple, dans le cas d'une variable entière, un couple d'arcs étiquetés respectivement $[0, 3]$ et $[4, 8]$ est équivalent à un arc étiqueté $[0, 8]$.

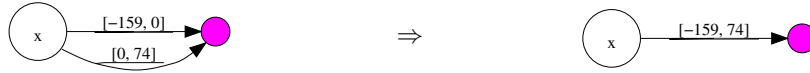


FIGURE 6 – Fusion d'arcs contigus.

Définition 11 (Arc infranchissable). *Un arc E d'un IA ϕ est infranchissable si et seulement si $\text{Itv}(E) \cap \text{Dom}(\text{Var}(E)) = \emptyset$.*

Un arc infranchissable ne sera jamais franchi (!), car aucune valeur dans son intervalle n'est cohérente avec le domaine de sa variable (voir figure 7).



FIGURE 7 – Élimination d'arcs infranchissables (ici $\text{Dom}(x) = \mathbb{R}_+$).

Définition 12 (Automate à intervalles réduit). *Un automate à intervalles ϕ est dit réduit si et seulement si*

- *aucun nœud de ϕ n'est isomorphe à un autre, ni bégayant, ni non décisif ;*
- *aucun arc de ϕ n'est contigu à un autre, ni infranchissable.*

Dans la suite, nous pouvons ne considérer que des IAs réduits, car la réduction peut s'effectuer en temps polynomial en la taille de la structure.

Proposition 13 (Réduction d'un IA). *Il existe un algorithme polynomial qui transforme n'importe quel IA ϕ en un IA réduit équivalent ϕ' tel que $|\phi'| \leq |\phi|$.*

Le premier résultat que nous obtenons sur les FIAs est que ceux-ci ne sont pas plus difficiles à réduire que les IAs : l'algorithme de réduction maintient la propriété de convergence si on l'applique à un FIA.

Proposition 14 (Réduction d'un FIA). *Il existe un algorithme polynomial qui transforme n'importe quel FIA ϕ en un FIA réduit équivalent ϕ' tel que $|\phi'| \leq |\phi|$.*

3 Requêtes et transformations sur les automates à intervalles

Comme dit précédemment, un automate à intervalles représente une fonction d'un certain ensemble de variables vers $\{\perp, \top\}$. Cette section formalise les principales requêtes et transformations qui peuvent être utiles dans le contexte de la planification. Dans un objectif d'exhaustivité, nous avons également introduit les opérations habituellement étudiées lors de l'évaluation des capacités d'un langage-cible de compilation (Darwiche & Marquis, 2002).

3.1 Opérations utiles pour la planification

Compilation de politiques de décision

Dans un contexte de planification, nous désirons tout d'abord représenter par un automate à intervalles une politique de décision δ produite par un algorithme de planification donné. Dans ce cas, δ est une fonction portant sur deux ensembles de variables, l'ensemble S des variables d'état (*State variables*) et l'ensemble D des variables de décision. Pour une S -instanciation \vec{s} et une D -instanciation \vec{d} quelconques, $\delta(\vec{s}, \vec{d}) = \top$ si et seulement si \vec{d} est une décision qu'il convient de prendre dans l'état \vec{s} .

Pour exploiter une politique de décision δ en ligne, deux opérations de base sont nécessaires. Premièrement, à chaque fois que l'on observe un nouvel état \vec{s} , on doit déterminer l'ensemble des décisions que δ propose pour \vec{s} . Cette opération correspond au *conditionnement* de δ par \vec{s} . Une des décisions convenables doit alors être extraite, pour être exécutée. Cette opération correspond à l'*extraction de modèle*. Ces deux opérations seront définies formellement dans la suite.

Certains algorithmes de planification construisent une politique de décision δ de manière incrémentale, jusqu'à ce qu'elle couvre l'ensemble des états atteignables. Ici, la construction incrémentale consiste à ajouter itérativement à δ de nouvelles paires (\vec{s}, \vec{d}) telles que \vec{d} couvre l'état \vec{s} . Si nous voulons procéder ainsi en utilisant les automates à intervalles, l'opération nécessaire à chaque étape est de la forme $\delta := \delta \vee (\vec{s}, \vec{d})$, c'est-à-dire une *disjonction*.

Compilation de relations de transition

Les IAs peuvent également être utilisés pour représenter les relations de base utilisées dans un domaine de planification : celles qui définissent l'ensemble des états initiaux possibles, celui des états buts, et celui des transitions possibles d'un système donné. Considérons l'exemple d'une relation de transition T . Une telle relation porte sur trois ensembles de variables : l'ensemble S des variables représentant l'état (*State*) courant, l'ensemble D des variables représentant la décision prise, et l'ensemble S' des variables représentant l'état résultant après application de la décision. Pour toute $S \cup D \cup S'$ -instanciation $\vec{s}, \vec{d}, \vec{s}'$, $\delta(\vec{s}, \vec{d}, \vec{s}') = \top$ signifie que \vec{s}' est un état successeur possible, quand on applique la décision \vec{d} dans l'état \vec{s} . Plusieurs opérations peuvent être nécessaires pour manipuler efficacement des relations de transition compilées en IAs.

D'une part, dans les approches « en avant », il peut être utile de pouvoir calculer efficacement, pour un état courant \vec{s} et une décision \vec{d} , l'ensemble S' des successeurs possible de \vec{s} , c'est-à-dire les S' -instanciations \vec{s}' telles que $T(\vec{s}, \vec{d}, \vec{s}') = \top$. Cela requiert les opérations de *conditionnement*, pour assigner dans T l'état \vec{s} et la décision \vec{d} , et d'*énumération de modèles* pour obtenir tous les successeurs \vec{s}' possibles. Quand les actions sont déterministes, la *relation* de transition T devient une *fonction* de transition, et l'*extraction de modèle* suffit alors pour obtenir le seul successeur possible \vec{s}' .

D'autre part, dans les approches « en arrière » (Giunchiglia & Traverso, 1999), on cherche des politiques de décision en partant des états buts ; il est alors nécessaires d'évaluer des formules du type $\exists \vec{d}. \left(P(\vec{s}, \vec{d}) \wedge \forall \vec{s}', \left(T(\vec{s}, \vec{d}, \vec{s}') \rightarrow G(\vec{s}') \right) \right)$. Cette dernière formule définit tous les états \vec{s} tels qu'ils existe une décision \vec{d} satisfaisant certaines préconditions données par $P(\vec{s}, \vec{d})$, et tels que tout état successeur

possible \vec{s}' obtenu en effectuant \vec{d} dans l'état \vec{s} satisfait l'état but. De telles formules contiennent plusieurs opérations Booléennes : la conjonction (\wedge), la disjonction et la négation (pour \rightarrow), et les projections existentielle ($\exists \vec{d}$) et universelle ($\forall \vec{s}'$). Quand les actions sont déterministes, la conjonction et la projection existentielle suffisent.

Toutes les opérations intéressantes du point de vue de la planification, ainsi que d'autres opérations standard, sont définies formellement dans la suite.

3.2 Opérations sur les automates à intervalles

Détaillons maintenant les opérations¹ sur lesquelles nous nous focaliserons, et vérifions si elles peuvent être exécutées efficacement sur une forme compilée. Nous introduisons tout d'abord les *requêtes*, c'est-à-dire les opérations qui renvoient certaines informations sur un IA.

Définition 15 (Requêtes). Soit L un sous-langage de IA.

- L satisfait² **CO** (resp. **VA**) ssi il existe un algorithme polynomial associant tout automate ϕ de L à 1 si $I(\phi)$ est satisfiable (resp. valide), et à 0 sinon.
- L satisfait **EQ** ssi il existe un algorithme polynomial associant tout couple d'automates (ϕ, ϕ') de L à 1 si $I(\phi) \equiv I(\phi')$ et à 0 sinon.
- L satisfait **MC** ssi il existe un algorithme polynomial associant tout automate ϕ de L et toute $\text{Var}(\phi)$ -instanciation \vec{x} à 1 si $I(\phi)(\vec{x}) = \top$ et à 0 sinon.
- L satisfait **MX** ssi il existe un algorithme polynomial associant tout automate ϕ de L à un modèle de ϕ s'il en existe, et s'arrête sans rien retourner dans le cas contraire.
- L satisfait **ME** ssi il existe un polynôme $p(\cdot)$ et un algorithme qui renvoie l'ensemble des modèles de tout automate ϕ de L en temps $p(n; m)$, avec n la taille de ϕ et m le nombre minimal de boîtes (c'est-à-dire de produits cartésiens d'intervalles) dont l'union est égale à $\text{Mod}(\phi)$.
- L satisfait **CX** ssi il existe un polynôme $p(\cdot)$ et un algorithme qui renvoie $\text{Ctxt}_\phi(y)$ pour tout ϕ de L et tout $y \in \text{Var}(\phi)$ en temps $p(n; m)$, avec n la taille de ϕ et m le nombre minimal d'intervalles dont l'union est égale à $\text{Ctxt}(y, \phi)$.

Nous allons maintenant définir un certain nombre de *transformations* sur les IAs (c'est-à-dire des opérations dont le résultat est un IA modifié); nous présentons en premier lieu les opérations sémantiques sur lesquelles elles se basent.

Définition 16. Soient I, J les fonctions d'interprétation portant sur $\text{Var}(I), \text{Var}(J)$ d'automates donnés.

- La conjonction (resp. disjonction) de I et J est la fonction $I \wedge J$ (resp. $I \vee J$) sur les variables de $X = \text{Var}(I) \cup \text{Var}(J)$ définie par $(I \wedge J)(\vec{x}) = I(\vec{x}) \wedge J(\vec{x})$ (resp. $(I \vee J)(\vec{x}) = I(\vec{x}) \vee J(\vec{x})$).
- La projection existentielle de I sur $Y \subseteq \text{Var}(I)$ est la fonction $I^{\downarrow Y}$ sur les variables de Y définie par : $I^{\downarrow Y}(\vec{y}) = \top$ ssi il existe une Z -instanciation \vec{z} (avec $Z = \text{Var}(I) \setminus Y$) t.q. $I(\vec{z} \cdot \vec{y}) = \top$. L '« oubli » est l'opération duale : $\text{forget}(I, Y) = I^{\downarrow \text{Var}(I) \setminus Y}$.
- La projection universelle de I sur $Y \subseteq \text{Var}(I)$ est la fonction $I^{\uparrow Y}$ sur les variables de Y définie par : $I^{\uparrow Y}(\vec{y}) = \top$ ssi pour toute Z -instanciation \vec{z} (avec $Z = \text{Var}(I) \setminus Y$), $I(\vec{z} \cdot \vec{y}) = \top$. L '« assurance » est l'opération duale : $\text{ensure}(I, Y) = I^{\uparrow \text{Var}(I) \setminus Y}$.
- Étant donnée une instanciation \vec{y} d'un ensemble quelconque de variables $Y \subseteq \text{Var}(I)$, le conditionnement de I par \vec{y} est la fonction $I_{|\vec{y}}$ sur les variables de $Z = \text{Var}(I) \setminus Y$ définie par : $I_{|\vec{y}}(\vec{z}) = I(\vec{y} \cdot \vec{z})$.

Nous pouvons à présent introduire les transformations :

Définition 17 (Transformations). Soit L un sous-langage de IA.

- L satisfait **CD** ssi il existe un algorithme polynomial associant tout automate ϕ de L et toute instanciation \vec{y} de $Y \subseteq \text{Var}(\phi)$ à un automate ϕ' de L tel que $I(\phi') = I(\phi)_{|\vec{y}}$.
- L satisfait **FO** (resp. **EN**) ssi il existe un algorithme polynomial associant tout automate ϕ de L et tout $Y \subseteq \text{Var}(\phi)$ à un automate ϕ' de L tel que $I(\phi') = \text{forget}(I(\phi), Y)$ (resp. $I(\phi') = \text{ensure}(I(\phi), Y)$).

1. **CO** signifie « Consistency » (satisfiabilité); **VA**, « Validité »; **EQ**, « Équivalence »; **MC**, « Model Checking » (vérification de modèle); **MX**, « Model eXtraction »; **ME**, « Model Enumeration »; **CX**, « Context Extraction »; **CD**, « ConDitionnement »; **FO**, « FOrgetting » (oubli); **EN**, « ENsuring » (enforcement); **SCD**, **SFO**, **SEN**, « Simple **CD**, **FO**, **EN** »; $\wedge C$, $\vee C$, « \wedge , \vee -closure » (conjonction et disjonction); $\wedge BC$, $\vee BC$, « \wedge , \vee -Binary Closure » (conjonction et disjonction binaires); et $\wedge tC$, « Closure under conjunction with a term » (conjonction avec un terme).

2. On dit aussi « supporte ».

L	CO	VA	EQ	MC	MX	CX	ME	CD	SCD	\wedge tC	FO	SFO	EN	SEN	\wedge C	\wedge BC	\vee C	\vee BC
IA	○	○	○	√	○	○	○	√	√	√	○	√	○	√	√	√	√	√
FIA	√	○	○	√	√	√	[√]	√	√	√	√	√	○	○	○	○	√	√

TABLE 1 – Résultats sur la complexité des requêtes et transformations. √ signifie « satisfait », ○ « ne satisfait pas, excepté si P = NP ». [.] signale une conjecture.

- L satisfait SCD (resp. SFO, resp. SEN) ssi il satisfait CD (resp. FO, resp. EN) pour une seule variable (c'est-à-dire que $\text{Card}(Y) = 1$).
- L satisfait \wedge C (resp. \vee C) ssi il existe un algorithme polynomial associant tout ensemble fini d'automates $\Phi = \{\phi_1, \dots, \phi_k\}$ de L à un automate ϕ de L tel que $I(\phi) = I(\phi_1) \wedge \dots \wedge I(\phi_k)$ (resp. $I(\phi) = I(\phi_1) \vee \dots \vee I(\phi_k)$).
- L satisfait \wedge BC (resp. \vee BC) ssi il satisfait \wedge C (resp. \vee C) pour un couple d'automates (c'est-à-dire que $\text{Card}(\Phi) = 2$).
- L satisfait \wedge tC ssi il existe un algorithme polynomial associant tout automate ϕ de L, tout ensemble de variables $\{y_1, \dots, y_k\} \subseteq \text{Var}(\phi)$ et toute suite (A_1, \dots, A_k) d'intervalles fermés, à un automate ϕ' de L tel que $I(\phi') = I(\phi) \wedge f_{y_1, A_1} \wedge \dots \wedge f_{y_k, A_k}$, où $f_{x, A}$ est la fonction définie sur $Y = \{x\}$ par $f_{x, A}(\vec{y}) = \top \Leftrightarrow \vec{y}(x) \in A$.

3.3 Résultats de complexité

Proposition 18. *Les résultats de la table 1 sont prouvés.*

Il apparaît que les performances des automates à intervalles sont faibles en ce qui concerne la plupart des requêtes, et en particulier CO, MX et VA. Imposer la propriété restrictive de convergence rend la plupart des requêtes polynomiales, y compris CO et MX. La raison principale est que tout chemin de la racine au puits d'un FIA réduit est cohérent, car aucun arc n'est en conflit avec un autre sur ce chemin.

C'est pour cette même raison (ajoutée au fait que les nœuds \vee sont autorisés) que FIA supporte CD et FO.

La proposition 18 montre que les FIAs conviennent à la compilation de politiques de décision, ainsi que de tables de transition³ si celles-ci sont utilisées dans le cadre d'une exploitation « en avant ».

4 Construction d'automates à intervalles

Nous avons montré que les FIAs permettent de faire en temps polynomial des opérations utiles à la planification. Penchons-nous à présent sur la construction de ces automates.

Union de boîtes

Il est immédiat de convertir une union de boîtes en FIA. Cela peut se faire en temps polynomial, grâce à la transformation \vee C. Nous pouvons alors facilement compiler en FIA n'importe quelle politique ou table de transition donnée sous forme d'union de boîtes, qu'elle soit discrète (obtenue par exemple grâce à un algorithme renvoyant des DNFs) ou continue (obtenue par exemple grâce à un solveur de contraintes basé sur les intervalles).

Trace de RealPaver

Il est également possible d'adopter un procédé similaire à (Huang & Darwiche, 2005), en utilisant la trace d'un algorithme de recherche comme un moyen pratique pour obtenir la forme compilée. Ce procédé consiste à créer de nouveaux nœuds et arcs aussitôt qu'une solution est trouvée par l'algorithme de recherche, et à les fusionner avec le FIA courant qui retient toutes les solutions déjà trouvées. Nous utiliserons ici cette méthode sur le solveur RealPaver (Granvilliers & Benhamou, 2006), qui fonctionne avec des

3. L'exploitation de tables de transition n'est pour l'instant envisageable que pour des problèmes à actions déterministes, car ME n'est pas encore prouvé. Cependant les politiques de décision peuvent tout à fait être non-déterministes (voir 3.1).

problem	durée réd. (ms)	taille (arcs)	% entrée	% OBDD	CD (ms)	MX (ms)
obsmem2	1102	100	74	66	1	5
obsmem3	2168	197	75	69	4	11
obsmem4	4729	342	75	70	4	11
obsmem5	5657	546	76	70	7	19
obsmem6	9433	820	76	76	11	35
porobot	4035	56	97	36	0	1
forobot	52767	60	99	31	0	3
ring7	92	13	75	71	0	1
ring8	185	13	78	75	0	1
ring9	92	13	80	75	0	1
ring10	82	13	81	75	0	2
drone10	46732	453	95	47	11	23
drone20	947174	763	97	44	30	61
drone30	2850715	944	98	43	21	48
drone40	5721059	944	98	45	15	29
drone10	102262	17061	35	×	850	15
drone20	953961	43260	26	×	13230	156
drone30	1964061	56410	33	×	22957	492

TABLE 2 – Résultats d’application.

intervalles, pour créer un FIA représentant une approximation de l’ensemble des solutions d’un réseau de contraintes.

5 Résultats

Le tableau 2 présente quelques résultats pour un certain nombre de problèmes discrets et continus, sous la forme de politiques ou de tables de transition. Le problem `obsmem` régit les connexions entre l’appareil d’observation et la mémoire de masse d’un satellite. Le problème `robot` met en scène un robot explorant un environnement, et le problème `ring` est un *benchmark* standard de planification non-déterministe. Dans le problème `drone`, un drone doit remplir différents objectifs sur un certain nombre de zones en temps limité.

Dans le tableau 2, les trois dernières instances sont des tables de transition avec des variables continues (ce qui rend impossible la comparaison avec les OBDDs), obtenues en suivant la trace de RealPaver. Toutes les autres sont des politiques de décision discrètes, obtenues en compilant des unions de boîtes. Pour chaque instance, nous indiquons le temps nécessaire à la réduction du FIA compilé, la taille du FIA réduit, le taux de réduction (0% signifiant pas de réduction) par rapport à l’entrée (nombre de boîtes \times nombre de variables), le taux de réduction par rapport à l’OBDD équivalent, et la durée moyenne prise par un conditionnement simple ou une extraction de modèle.

Ces résultats montrent que les FIAs peuvent être avantageusement comparés aux OBDDs du point de vue de la taille, et que la durée des opérations nécessaires à l’exploitation des formes compilées est compatible avec leur utilisation en ligne.

6 Conclusion

Dans cet article, nous avons introduit les automates à intervalles, un nouveau langage-cible de compilation adapté à la représentation de fonctions Booléennes portant sur des variables énumérées et/ou continues. Nous avons identifié une sous-classe de ce langage, celle des automates à intervalles convergents, pour laquelle nous avons prouvé la polynomialité de plusieurs opérations, utiles dans un contexte de planification. Nous avons montré que les gains en matière de taille de la forme compilée étaient significatifs par rapport aux OBDDs, les IAs ayant de plus l’avantage de pouvoir représenter des domaines continus sans avoir

recours à une discrétisation. À l'avenir, nous projetons d'étudier d'autres classes intéressantes des IAs, ainsi que de définir d'autres langages-cibles adaptés à la gestion de domaines de planification.

Références

- [Amilhastre *et al.*, 1999] AMILHASTRE J., P. & VILAREM M.-C. (1999). Fa Minimisation Heuristics for a Class of Finite Languages. In *WIA*, p. 1–12.
- [Bryant, 1986] BRYANT R. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, **35**(8), 677–691.
- [Cadoli & Donini, 1998] CADOLI M. & DONINI F. (1998). A Survey on Knowledge Compilation. *AI Communications*, **10**(3–4), 137–150.
- [Darwiche, 2001] DARWICHE A. (2001). Decomposable Negation Normal Form. *Journal of the ACM*, **48**(4), 608–647.
- [Darwiche & Marquis, 2002] DARWICHE A. & MARQUIS P. (2002). A Knowledge Compilation Map. *Journal of Artificial Intelligence Research (JAIR)*, **17**, 229–264.
- [del Val, 1994] DEL VAL A. (1994). Tractable Databases: How to Make Propositional Unit Resolution Complete Through Compilation. In *Proc. of KR'94*, p. 551–561.
- [Giunchiglia & Traverso, 1999] GIUNCHIGLIA F. & TRAVERSO P. (1999). Planning as Model Checking. In *European Conference on Planning (ECP)*, p. 1–20.
- [Granvilliers & Benhamou, 2006] GRANVILLIERS L. & BENHAMOU F. (2006). Algorithm 852: RealPaver: an Interval Solver Using Constraint Satisfaction Techniques. *ACM Trans. Math. Softw.*, **32**(1), 138–156.
- [Huang & Darwiche, 2005] HUANG J. & DARWICHE A. (2005). DPLL with a Trace: From SAT to Knowledge Compilation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 156–162.
- [Meinel & Theobald, 1998] MEINEL C. & THEOBALD T. (1998). *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer.
- [Murray & Rosenthal, 2003] MURRAY N. V. & ROSENTHAL E. (2003). Tableaux, Path Dissolution, and Decomposable Negation Normal Form for Knowledge Compilation. In *TABLEAUX*, p. 165–180.
- [Selman & Kautz, 1996] SELMAN B. & KAUTZ H. (1996). Knowledge Compilation and Theory Approximation. *Journal of the ACM*, **43**, 193–224.
- [Strehl & Thiele, 1998] STREHL K. & THIELE L. (1998). Symbolic Model Checking of Process Networks Using Interval Diagram Techniques. In *Proc. of the 1998 IEEE/ACM international conference on Computer-aided design*, p. 686–692.
- [Vempaty, 1992] VEMPATY N. R. (1992). Solving Constraint Satisfaction Problems Using Finite State Automata. In *Association for the Advancement of Artificial Intelligence Conference (AAAI)*, p. 453–458.

Preuves

Dans toutes les preuves, pour des raisons de clarté, nous considérerons que pour une fonction d'interprétation I avec $Y = \text{Var}(I)$, un ensemble de variables Z t.q. $Z \cap Y = \emptyset$, une Y -instanciation \vec{y} et une Z -instanciation \vec{z} , $I(\vec{y} . \vec{z})$ est simplement défini comme $I(\vec{y})$.

Soit ϕ un IA sur X et \vec{x} une X -instanciation. Un chemin p de la racine au puits de ϕ est dit *compatible* avec \vec{x} si et seulement si pour tout arc E sur p , soit $\text{Var}(E) = \vee$ et $\text{Itv}(E) \neq \emptyset$, soit $\vec{x}(\text{Var}(E)) \in \text{Itv}(E)$.

Pour un I quelconque, on notera $\vec{x} \models I$ le fait que \vec{x} est un modèle de I , et $\vec{x} \models \phi$ ssi $\vec{x} \models I(\phi)$. On notera également $\vec{x} \models p$ si p est un chemin compatible avec \vec{x} .

Preuve de la proposition 5. Étant donné que les variables Booléennes peuvent être représentées par des variables réelles de domaine $\{0, 1\}$, on peut transformer un BDD en IA en enlevant sa feuille \perp , et en supprimant récursivement tous les arcs ne pointant vers aucun nœud, et tous les nœuds sans arc sortant (excepté la feuille \top). Le graphe obtenu devient alors un IA si on remplace tous les arcs \top par des arcs $[1, 1]$ et tous les arcs \perp par des arcs $[0, 0]$. \square

Preuve de la proposition 6. En utilisant la procédure décrite dans la preuve de la prop. 5 sur un FBDD, l'IA obtenu est convergent. En effet, toute variable ne peut être rencontrée qu'une seule fois sur tout chemin ; il n'y a donc aucun risque qu'un intervalle soit en conflit avec un autre. \square

Algorithm 1 Algorithme de réduction. À tout moment de la procédure, les arcs sans nœud source ou sans nœud destination sont supprimés, ainsi que les nœuds (non-puits) sans arc sortant et les nœuds (non-racine) sans arc entrant.

```

1: repeat
2:   numérotter les nœuds de  $\phi$  de façon à ce que si  $N_i \in \text{Ch}(N_j)$ , alors  $i < j$ 
3:   for  $i$  de 1 au nombre de nœuds dans  $\Gamma(\phi)$  do
4:     if  $N_i$  est bégayant then
5:       for all  $(E_{\text{in}}, E_{\text{out}}) \in \text{In}(N_i) \times \text{Out}(N_i)$  do
6:         ajouter un arc de  $\text{Src}(E_{\text{in}})$  à  $\text{Dest}(E_{\text{out}})$  étiqueté par  $\text{Itv}(E_{\text{in}}) \cap \text{Itv}(E_{\text{out}})$ 
7:       supprimer  $N_i$ 
8:     else
9:       for all  $E \in \text{Out}(N_i)$  do
10:        if  $E$  est infranchissable then
11:          supprimer  $E$ 
12:        else
13:          marquer  $E$ 
14:          for all  $E' \in \text{Out}(N)$  tel que  $E'$  n'est pas marqué do
15:            if  $E$  et  $E'$  sont contigus then
16:              étiqueter  $E$  par  $\text{Itv}(E) \cup \text{Itv}(E')$ 
17:              supprimer  $E'$ 
18:            if  $N_i$  est non décisif then
19:              for all  $E_{\text{in}} \in \text{In}(N_i)$  do
20:                rediriger  $E_{\text{in}}$  vers le fils de  $N_i$ 
21:              supprimer  $N_i$ 
22:            else
23:              for  $j$  de 1 au nombre de nœuds dans  $\Gamma(\phi)$  do
24:                if  $N_i$  et  $N_j$  sont isomorphes then
25:                  for all  $E_{\text{in}} \in \text{In}(N_i)$  do
26:                    rediriger  $E_{\text{in}}$  vers  $N_j$ 
27:                  supprimer  $N_i$ 
28:   until  $\phi$  n'a pas changé durant le processus

```

Preuve de la proposition 13. Appliquons l'algorithme 1 à un IA ϕ .

Pour un nœud quelconque N :

- l'opération de la ligne 4 supprime N s'il est bégayant ;
- l'opération de la ligne 10 assure que N n'a plus d'arc sortant infranchissable ;
- l'opération de la ligne 15, assure que N n'a plus d'arcs sortants contigus ;
- l'opération de la ligne 18 supprime N s'il est non décisif ;
- l'opération de la ligne 24 supprime tous les nœuds isomorphes à N ;

et l'algorithme s'arrête lorsqu'on ne peut plus appliquer d'opération, donc quand l'IA est réduit. De plus, il est simple de vérifier qu'aucune des opérations ne change la sémantique de ϕ . Enfin, chaque opération enlève strictement plus de nœuds qu'elle n'en crée⁴ ; cela prouve que (i) l'algorithme finit par s'arrêter (une fois que l' IA est vide, il ne change plus), et (ii) la taille de l'IA résultant est inférieure à $|\phi|$ (le seul cas où la taille reste inchangée est quand l'IA d'entrée est déjà réduit).

Pour chaque nœud, le calcul est, de manière évidente, polynomial, et la boucle sur les nœuds (ligne 3) ne traite chaque nœud qu'une seule fois ; par conséquent, tout ce qui est à l'intérieur de la boucle principale de répétition (de la ligne 2 à la ligne 27) est effectué en temps polynomial en $|\phi|$.

Comme les propriétés de réductibilité ne sont pas mutuellement indépendantes, le parcours du graphe doit être répété tant que quelque chose a été modifié dans ϕ (ligne 28). Cela n'influe pas sur la polynomialité ; en effet, comme chaque parcours diminue la taille de ϕ (sauf pour le dernier bien sûr), la boucle de parcours ne sera pas répétée plus de $|\phi|$ fois.

À noter qu'il existe évidemment des méthodes beaucoup plus efficaces pour réduire un IA, mais l'objectif

4. La seule opération qui ajoute quelque chose est celle qui concerne les nœuds bégayants, mais rappelons que dans ce cas soit $\text{In}(N_i)$ soit $\text{Out}(N_i)$ ne contient qu'un seul élément.

ici est simplement de montrer la polynomialité de cette opération. \square

Définition 19 (Convergence selon une variable). *Un IA ϕ est dit convergent selon y , avec $y \in \text{Var}(\phi)$ ssi tout arc E dans ϕ t.q. $\text{Var}(E) = y$ est convergent.*

Lemme 20. *L'algorithme 1 maintient la propriété de convergence selon une variable donnée.*

Preuve. Soit ϕ un IA convergent selon $y \in \text{Var}(\phi)$. Supposons que nous sommes à l'étape i de l'algorithme, avec $\text{Var}(N_i) = y$.

- opération « nœud bégayant » : soit $E \in \text{Out}(N_i)$, et E' un arc sur un chemin de $\text{Dest}(E)$ au puits, tel que $\text{Var}(E') = y$. E' est convergent, donc $\text{Itv}(E') \subseteq \text{Itv}(E)$. Ainsi $\text{Itv}(E') \subseteq \text{Itv}(E) \cap \text{Itv}(E_{\text{in}})$ pour tout $E_{\text{in}} \in \text{In}(N_i)$. Par conséquent E' est toujours convergent après cette opération.
- opération « arc infranchissable » : supprimer des arcs n'a aucune influence sur la convergence d'autres arcs dans le graphe.
- opération « arcs contigus » : les deux arcs contigus pointent vers le même nœud, donc tout arc E associé à y dans la descendance de ces deux arcs est tel que $\text{Itv}(E)$ est inclus dans l'intervalle étiquetant chacun des deux arcs ; il est donc inclus dans leur union.
- opération « nœud non décisif » : tout arc descendant du fils de N_i descend aussi de N_i , donc cette opération ne compromet pas leur convergence.
- étant donné que tout arc sortant de tout nœud isomorphe à N_i est convergent (par hypothèse), rediriger tous ses arcs entrants vers N_i est sans conséquence sur la convergence. \square

Lemme 21 (IAs convergents selon toutes leurs variables). *Un IA réduit ϕ qui est convergent selon toutes les variables de $\text{Var}(\phi)$ est convergent.*

Preuve. Soit E un arc dans ϕ . Soit $\text{Var}(E) = y \neq \perp$, auquel cas E est convergent, puisque ϕ est convergent selon y ; soit $\text{Var}(E) = \perp$, auquel cas E est également convergent, puisque tout E' dans ϕ tel que $\text{Var}(E') = \perp$ est étiqueté par \mathbb{R} (ϕ étant réduit). ϕ est donc convergent. \square

Preuve de la proposition 14. Soit ϕ un FIA. ϕ est a fortiori convergent selon toutes ses variables. Le lemme 20 établit que l'IA ϕ' obtenu en appliquant l'algorithme de réduction défini dans la preuve de la prop. 13 est également convergent selon toutes ses variables. Comme ϕ' est réduit, nous utilisons le lemme 21 pour déduire que ϕ' est convergent. Ainsi, notre algorithme de réduction maintient la propriété de convergence, d'où le résultat. \square

Définition 22 (Maillage d'une variable dans un IA). *Soit ϕ un IA et $x \in \text{Var}(\phi)$. Un maillage de x dans ϕ est une partition $\mathcal{M} = \{M_1, \dots, M_n\}$ de \mathbb{R} telle que pour tout arc E de ϕ tel que $\text{Var}(E) = x$, quel que soit $1 \leq i \leq n$, $(M_i \cap \text{Itv}(E) \neq \emptyset) \Rightarrow (M_i \subseteq \text{Itv}(E))$.*

Lemme 23 (Obtention d'un maillage). *On peut obtenir en temps linéaire un maillage d'une variable dans un IA quelconque.*

Preuve. Soit ϕ un IA et $x \in \text{Var}(\phi)$. Il suffit de parcourir le graphe et de récupérer les bornes (finies, inférieures et supérieures) de tous les arcs associés à x ; cette procédure est linéaire. Soit $\mathcal{B} = \{b_1, \dots, b_k\}$ l'ensemble des bornes ainsi obtenues, triées par ordre croissant. Alors

$$\mathcal{M} = \left\{] - \infty, b_1[, \{b_1\},]b_1, b_2[, \{b_2\}, \dots,]b_{i-1}, b_i[, \{b_i\},]b_i, b_{i+1}[, \dots, \{b_k\},]b_k, +\infty[\right\}$$

est un maillage de x dans ϕ . En effet :

- de manière évidente \mathcal{M} est une partition de \mathbb{R} (tous les ensembles sont disjoints et leur union est égale à \mathbb{R}) ;
- soit un arc E de ϕ tel que $\text{Var}(E) = x$, et un entier i tel que $M_i \cap \text{Itv}(E) \neq \emptyset$. Par construction de \mathcal{M} , M_i est soit un singleton, soit un intervalle ouvert des deux côtés. Dans le premier cas il est évident que $M_i \subseteq \text{Itv}(E)$. Dans le second cas, il est impossible que M_i contienne une borne de $\text{Itv}(E)$, par construction ; donc forcément $M_i \subseteq \text{Itv}(E)$. \square

Lemme 24 (Conditionnement sur un élément d'un maillage). *Soit ϕ un IA de fonction d'interprétation I , $x \in \text{Var}(\phi)$, \mathcal{M} un maillage de x dans ϕ , et \vec{x} une $\{x\}$ -instanciation. Notons M l'élément du maillage tel que $\vec{x}(x) \in M$: pour n'importe quelle $\{x\}$ -instanciation \vec{m} telle que $\vec{m}(x) \in M$, on a $I_{|\vec{x}} = I_{|\vec{m}}$.*

Preuve. Soit $Z = \text{Var}(\phi) \setminus \{x\}$, et \vec{z} une Z -instanciation.

(\Rightarrow) Supposons que $I_{|\vec{x}}(\vec{z}) = \top$, alors $I(\vec{x} \cdot \vec{z}) = \top$. Il existe donc un chemin p dans ϕ compatible avec \vec{x} et \vec{z} . Soit une quelconque $\{x\}$ -instanciation \vec{m} telle que $\vec{m}(x) \in M$. p est compatible avec \vec{m} , car pour n'importe quel arc E de p tel que $\text{Var}(E) = x$, on sait que $M \cap \text{Itv}(E) \neq \emptyset$ (puisque $\vec{x}(x) \in M$ et $\vec{x}(x) \in \text{Itv}(E)$) et que donc par définition de \mathcal{M} , $M \subseteq \text{Itv}(E)$. Par conséquent, $I(\vec{m} \cdot \vec{z}) = \top$, donc $I_{|\vec{m}}(\vec{z}) = \top$.

(\Leftarrow) Supposons que $I_{|\vec{x}}(\vec{z}) = \perp$, alors $I(\vec{x} \cdot \vec{z}) = \perp$, donc tout chemin de ϕ compatible avec \vec{z} n'est pas compatible avec \vec{x} . Soit une quelconque $\{x\}$ -instanciation \vec{m} telle que $\vec{m}(x) \in M$. Supposons qu'il existe un chemin p compatible avec $\vec{m} \cdot \vec{z}$: tout arc E de p tel que $\text{Var}(E) = x$ vérifie alors $\vec{m}(x) \in \text{Itv}(E)$. Donc, par définition de \mathcal{M} , $M \subseteq \text{Itv}(E)$; puisqu'on a choisi M tel que $\vec{x}(x) \in M$, $\vec{x}(x) \in \text{Itv}(E)$, ce qui est absurde puisque p est incompatible avec \vec{x} . On en déduit que $I(\vec{m} \cdot \vec{z}) = \perp$ et donc que $I_{|\vec{m}}(\vec{z}) = \perp$. □

Lemme 25 (Forgetting/ensuring). *Soit ϕ un IA de fonction d'interprétation I , $x \in \text{Var}(\phi)$, et $\mathcal{M} = \{M_1, \dots, M_n\}$ un maillage de x dans ϕ . Soit $(\vec{m}_1, \dots, \vec{m}_n)$ une suite de $\{x\}$ -instanciations telles que pour $1 \leq i \leq n$, $\vec{m}_i(x) \in M_i$. On montre que*

- $\text{forget}(I, \{x\}) = \bigvee_{i=1}^n I_{|\vec{m}_i}$;
- $\text{ensure}(I, \{x\}) = \bigwedge_{i=1}^n I_{|\vec{m}_i}$.

Preuve. Soit $Z = \text{Var}(\phi) \setminus \{x\}$, et \vec{z} une Z -instanciation.

— (\Rightarrow) Supposons que $\text{forget}(I, \{x\})(\vec{z}) = \top$. Il existe alors une $\{x\}$ -instanciation \vec{x} telle que $I(\vec{z} \cdot \vec{x}) = \top$, c'est-à-dire que $I_{|\vec{x}}(\vec{z}) = \top$. Soit i l'entier tel que $\vec{x}(x) \in M_i$ (il existe car \mathcal{M} est une partition de \mathbb{R}). Le lemme 24 nous permet d'affirmer que $I_{|\vec{m}_i}(\vec{z}) = \top$. Par conséquent, de manière évidente, $(\bigvee_{i=1}^n I_{|\vec{m}_i})(\vec{z}) = \top$.

(\Leftarrow) Supposons que $\text{forget}(I, \{x\})(\vec{z}) = \perp$. Alors toute $\{x\}$ -instanciation \vec{x} vérifie $I(\vec{z} \cdot \vec{x}) = \perp$, et donc aussi $I_{|\vec{x}}(\vec{z}) = \perp$. Par conséquent $(\bigvee_{i=1}^n I_{|\vec{m}_i})(\vec{z}) = \perp$.

— (\Rightarrow) Supposons que $\text{ensure}(I, \{x\})(\vec{z}) = \top$. Alors toute $\{x\}$ -instanciation \vec{x} vérifie $I(\vec{z} \cdot \vec{x}) = \top$, et donc aussi $I_{|\vec{x}}(\vec{z}) = \top$. Par conséquent $(\bigwedge_{i=1}^n I_{|\vec{m}_i})(\vec{z}) = \top$.

(\Leftarrow) Supposons que $\text{ensure}(I, \{x\})(\vec{z}) = \perp$. Il existe alors une $\{x\}$ -instanciation \vec{x} telle que $I(\vec{z} \cdot \vec{x}) = \perp$, c'est-à-dire que $I_{|\vec{x}}(\vec{z}) = \perp$. Soit i l'entier tel que $\vec{x}(x) \in M_i$ (il existe car \mathcal{M} est une partition de \mathbb{R}). Le lemme 24 nous permet d'affirmer que $I_{|\vec{m}_i}(\vec{z}) = \perp$. Par conséquent, de manière évidente, $(\bigwedge_{i=1}^n I_{|\vec{m}_i})(\vec{z}) = \perp$. □

Définition 26 (Restriction sur une variable). *Pour tout IA ϕ , pour tout $x \in \text{Var}(\phi)$, pour tout intervalle fermé $A \in \mathbb{R}$, la restriction de ϕ à $x \in A$, notée $\phi_{|x \in A}$ est l'IA ayant le même graphe Γ que ϕ , excepté que pour tout arc E tel que $\text{Var}(E) = x$, $\text{Itv}_{\phi_{|x \in A}}(E) = \text{Itv}_{\phi}(E) \cap A$.*

Notons que l'ordre dans lequel les opérations de restriction sont appliquées n'importe pas : $(\phi_{|x_i \in A})_{|x_j \in B} = (\phi_{|x_j \in B})_{|x_i \in A}$. On notera $\phi_{|x_1 \in I_1, \dots, x_k \in I_k}$ la restriction de ϕ à $x_1 \in I_1, \dots, x_k \in I_k$.

Lemme 27 (Propriétés de la restriction).

- Pour tout IA ϕ , tout $x \in \text{Var}(\phi)$, tout intervalle fermé A , $I(\phi_{|x \in A}) = I(\phi) \wedge f_{x,A}$.
- Pour tout IA ϕ , $I(\phi_{|x_1 \in A_1, \dots, x_k \in A_k}) = I(\phi) \wedge f_{x_1, A_1} \wedge \dots \wedge f_{x_k, A_k}$.
- Si ϕ est un FIA, alors $\phi_{|x_1 \in I_1, \dots, x_k \in I_k}$ est aussi un FIA.

Preuve. — Supposons que $\vec{y} \models \phi_{|x \in A}$; il existe alors un chemin ϕ tel que pour tout arc E sur ce chemin, $\vec{y}(\text{Var}(E)) \in \text{Itv}(E)$ quand $\text{Var}(E) \neq x$, et $\vec{y}(x) \in \text{Itv}(E) \cap A$. Ainsi $\vec{y} \models \phi$ et $\vec{y}(x) \in A$: $\vec{y} \models I(\phi) \wedge f_{x,A}$.

Réciproquement, supposons que $\vec{y} \models I(\phi) \wedge f_{x,A}$; alors $\vec{y} \models I(\phi)$, et $\vec{y} \models f_{x,A}$.

Il existe donc dans ϕ un chemin p de la racine au puits tel que pour tout E sur p , $\vec{y}(\text{Var}(E)) \in \text{Itv}(E)$; de plus $\vec{y}(x) \in A$. Par conséquent pour tout E sur p , $\vec{y}(\text{Var}(E)) \in \text{Itv}(E)$ si $\text{Var}(E) \neq x$, et $\vec{y}(\text{Var}(E)) \in \text{Itv}(E) \cap A$ si $\text{Var}(E) = x$.

Puisque ϕ et $\phi|_{x \in A}$ ont les mêmes arcs, et par définition des intervalles étiquetant les arcs de $\phi|_{x \in A}$, on en déduit qu'il existe dans ϕ un chemin p de la racine au puits tel que pour tout E sur p , $\vec{y}(\text{Var}(E)) \in \text{Itv}(E) : \vec{y} \models I(\phi|_{x \in A})$.

Donc $I(\phi|_{x \in A}) = I(\phi) \wedge f_{x,A}$.

- $I(\phi|_{x_1 \in A_1, \dots, x_k \in A_k}) = I(\phi) \wedge f_{x_1 \in A_1} \cdots \wedge f_{x_k \in A_k}$ est vrai car l'ordre dans lequel sont appliquées les opérations de restriction n'importe pas.
- Le dernier point est vrai car $A \subseteq B$ implique $(A \cap C) \subseteq (B \cap C)$: si ϕ est convergent, alors c'est aussi le cas pour $\phi|_{x_i \in A_i}$. En répétant l'opération pour $1 \leq i \leq k$, on obtient que $\phi|_{x_1 \in A_1, \dots, x_k \in A_k}$ est convergent.

□

Preuve de la partie IA de la proposition 18.

CO, VA, EQ, ME.

Tout BDD peut être transformé en IA en temps polynomial (prop. 5) et BDD ne satisfait pas CO (resp. VA, resp. EQ, resp. ME) sauf si $P = NP$.

MX, CX.

Si MX (resp. CX) était polynomial, nous aurions un algorithme polynomial pour décider si un BDD est satisfiable (encore à cause de la prop. 5), alors que BDD ne supporte pas CO.

FO.

De la même manière, étant donné un IA ϕ quelconque, ϕ est satisfiable ssi $\text{forget}(I(\phi), \text{Var}(\phi)) \equiv \top$. Les seuls IAs réduits sans aucune variable sont l'automate vide et l'automate-puits, et tester si un IA est vide se fait en temps constant. Si IA satisfaisait FO, nous aurions un algorithme polynomial pour décider de la satisfiabilité de n'importe quel BDD (encore à cause de la prop. 5).

EN.

Étant donné un IA ϕ quelconque, ϕ est valide ssi $\text{ensure}(I(\phi), \text{Var}(\phi)) \equiv \top$. Encore une fois, les seuls IAs réduits sans aucune variable sont l'automate vide et l'automate-puits. Si IA satisfaisait EN, nous aurions un algorithme polynomial pour décider de la validité de n'importe quel BDD (encore à cause de la prop. 5).

AtC.

Grâce au lemme 27, il nous suffit de conditionner syntaxiquement ϕ , en remplaçant l'étiquette de chaque arc E tel que $\text{Var}(E) = y_i \in \{y_1, \dots, y_k\}$ par $\text{Itv}(E) \cap A_i$.

CD, SCD.

Soit ϕ un IA, $Y \subseteq \text{Var}(\phi)$ et \vec{y} une Y -instanciation. On note $\phi|_{\vec{y}}$ l'IA obtenu en remplaçant, pour tout nœud N tel que $\text{Var}(N) \in Y$, son étiquette par \vee et l'étiquette de chaque arc $E \in \text{Out}(N)$ par \mathbb{R} si $\vec{y}(\text{Var}(E)) \in \text{Itv}(E)$ et par \emptyset sinon.

Par définition du conditionnement, $\vec{z} \in \text{Dom}(\text{Var}(\phi) \setminus \{Y\})$ est un modèle de $I(\phi)|_{\vec{y}}$ ssi $\vec{y} \cdot \vec{z}$ est un modèle de ϕ .

Supposons que $\vec{y} \cdot \vec{z}$ est un modèle de ϕ . Il y a alors dans ϕ un chemin p tel que $\vec{z} \cdot \vec{y} \models p$. Par construction, une copie $p|_{\vec{y}}$ de p existe dans $\phi|_{\vec{y}}$ et $\vec{z} \models p|_{\vec{y}} : \vec{z}$ est un modèle de $\phi|_{\vec{y}}$.

Supposons que $\vec{y} \cdot \vec{z}$ ne soit pas un modèle de ϕ : pour tout chemin p dans ϕ , il y a un arc E sur ce chemin tel que $\vec{y} \cdot \vec{z}(\text{Var}(E)) \notin \text{Itv}(E)$. Rappelons que les chemins de $\phi|_{\vec{y}}$ sont les mêmes que ceux de ϕ , et prenons $p|_{\vec{y}}$ celui correspondant à p . Si $\text{Var}(E) \in Y$, le fait que $\vec{y} \cdot \vec{z}(\text{Var}(E)) \notin \text{Itv}(E)$ nous a conduit à étiqueter l'arc correspondant dans $\phi|_{\vec{y}}$ par \emptyset : \vec{z} ne peut pas être compatible avec ce chemin. Si $\text{Var}(E) \in Y$, $\vec{y} \cdot \vec{z}(\text{Var}(E)) \notin \text{Itv}(E)$ signifie que $\vec{z}(\text{Var}(E)) \notin \text{Itv}(E)$: comme ces arcs ne sont pas modifiés, \vec{z} ne peut être compatible avec ce chemin. Donc \vec{z} n'est compatible avec aucun chemin dans $\phi|_{\vec{y}}$: \vec{z} n'est pas un modèle de $\phi|_{\vec{y}}$.

Par conséquent, $I(\phi|_{\vec{y}}) = I(\phi)|_{\vec{y}}$.

Comme la construction de $\phi|_{\vec{y}}$ est faite en temps polynomial en $|\phi|$, IA satisfait CD (et *a fortiori* SCD).

MC.

Conditionner tout d'abord l'IA par la X -instanciation \vec{x} que l'on veut vérifier, puis réduire : on obtient soit l'IA vide (alors \vec{x} n'est pas un modèle) soit l'IA puits (\vec{x} est alors un modèle).

$\forall C, \forall BC.$

Pour faire la disjonction de k IAs ϕ_1, \dots, ϕ_k , il suffit de créer une nouvelle racine N , étiquetée par \forall , puis d'ajouter, pour chaque ϕ_i , un arc de N à $\text{Root}(\phi_i)$ étiqueté par \mathbb{R} . Fusionner les puits des ϕ_i en un seul.

$\wedge C, \wedge BC.$

Pour faire la conjonction de k IAs ϕ_1, \dots, ϕ_k , il suffit de remplacer le puits de ϕ_i par la racine de ϕ_{i+1} , pour $1 \leq i \leq k-1$. La racine du nouvel IA est celle de ϕ_1 , son puits est celui de ϕ_k .

SFO.

Soit ϕ un IA de fonction d'interprétation I , et $x \in Z = \text{Var}(\phi)$ la variable à oublier. Le lemme 23 permet d'affirmer qu'on peut construire en temps linéaire en $|\phi|$, un maillage $\mathcal{M} = \{M_1, \dots, M_n\}$ de x dans ϕ . Or, d'après le lemme 25, on a $\text{forget}(I, \{x\}) = \bigvee_{i=1}^n I_{|\bar{m}_i}$; pour obtenir un IA dont la fonction d'interprétation est égale à $\text{forget}(I, \{x\})$, il suffit donc de faire n conditionnements simples et $n-1$ disjonctions. Ces deux opérations sont polynomiales (voir au-dessus **SCD** et $\forall C$), et n est linéaire en $|\phi|$, donc **SFO** est faisable en temps polynomial en $|\phi|$.

SEN.

Soit ϕ un IA de fonction d'interprétation I , et $x \in Z = \text{Var}(\phi)$ la variable à assurer. Le lemme 23 permet d'affirmer qu'on peut construire en temps linéaire en $|\phi|$, un maillage $\mathcal{M} = \{M_1, \dots, M_n\}$ de x dans ϕ . Or, d'après le lemme 25, on a $\text{ensure}(I, \{x\}) = \bigwedge_{i=1}^n I_{|\bar{m}_i}$; pour obtenir un IA dont la fonction d'interprétation est égale à $\text{ensure}(I, \{x\})$, il suffit donc de faire n conditionnements simples et $n-1$ conjonctions. Ces deux opérations sont polynomiales (voir au-dessus **SCD** et $\wedge C$), et n est linéaire en $|\phi|$, donc **SEN** est faisable en temps polynomial en $|\phi|$. □

Lemme 28 (Correspondance entre termes, clauses et FIA). *Tout terme (resp. clause) de la logique propositionnelle peut s'exprimer sous la forme d'un FIA en temps polynomial.*

Preuve. Il est immédiat de convertir tout terme (resp. toute clause) en FBDD en temps polynomial ; or la prop. 6 établit que tout FBDD peut être transformé en FIA en temps polynomial. □

Lemme 29 (Correspondance entre DNF et FIA). *Toute formule du langage DNF peut s'exprimer sous la forme d'un FIA en temps polynomial.*

Preuve. Le lemme 28 établit que tout terme peut être transformé en FIA en temps polynomial, et FIA supporte $\forall C$. □

Preuve de la partie FIA de la proposition 18.

$\forall C, \forall BC.$

Le même preuve que pour les IAs est valable pour les FIAs. Si les ϕ_i sont convergents, le FIA résultant est, de manière évidente, également convergent.

VA, EQ.

Toute DNF peut être changée en FIA en temps polynomial (lemme 29). DNF ne satisfait pas **VA** et **EQ** sauf si $P = NP$. Si FIA satisfaisait **VA** (resp. **EQ**), nous aurions un algorithme polynomial pour décider si une DNF est valide (resp. si deux DNFs sont équivalentes).

CO.

Soit ϕ un FIA. Nous pouvons réduire ϕ en temps polynomial (prop. 14). Or, le seul FIA réduit insatisfiable est l'automate vide. En effet, supposons que ϕ a au moins un arc E tel que $\text{Var}(E) \neq \perp$ (un IA réduit non-vide ne peut avoir *que* des nœuds \perp); comme $\text{Itv}(E) \cap \text{Dom}(\text{Var}(E)) \neq \emptyset$, il y a au moins une valeur ω dans $\text{Itv}(E)$ qui est compatible avec le domaine de sa variable. E étant convergent, ω est aussi compatible avec tous les arcs précédents dans ϕ . Puisque c'est le cas pour tous les arcs, ϕ ne peut pas être insatisfiable.

CX.

L'algorithme suivant est polynomial (chaque arc n'est rencontré qu'une seule fois) et calcule le contexte de y dans ϕ :

```

1: réduire  $\phi$ 
2: soit  $C := \emptyset$ 
3: marquer le puits de  $\phi$ 
4: for all nœud  $N$  dans  $\phi$ , ordonnés du puits à la racine do
5:   if  $N$  est marqué then
6:     for all  $E \in \text{In}(N)$  do
7:       if  $\text{Var}(E) = y$  then
8:          $C := C \cup \text{Itv}(E)$ 
9:       else
10:        marquer  $\text{Src}(E)$ 
11: if la racine de  $\phi$  est marquée then
12:    $C := \text{Dom}(y)$ 
13: return  $C$ 

```

L'idée est de trouver la y -frontière du puits (c'est-à-dire l'ensemble des nœuds N étiquetés par y tels qu'il existe un chemin d'un des fils de N au puits ne mentionnant pas y), en faisant remonter une marque, qui signifie qu'aucun nœud y n'a encore été rencontré. Si une marque atteint la racine, il y a au moins un chemin de la racine au puits ne contenant aucun nœud y , le contexte de y dans ϕ est donc $\text{Dom}(y)$ (en effet ϕ est réduit, donc le chemin est trivialement satisfiable, pour les mêmes raisons qu'un FIA réduit non vide est satisfiable). Si ce n'est pas le cas, le contexte de y est alors l'union des intervalles étiquetant les arcs par lesquels la y -frontière atteint le puits.

MX.

Soit ϕ un FIA. S'il est satisfiable (ce qui est vérifiable en temps polynomial), prenons $\vec{x} \in \text{Dom}(\text{Var}(\phi))$. Réduisons tout d'abord ϕ , ce qui se fait en temps polynomial (prop. 14). En partant du puits, on choisit un chemin menant à la racine. Pour tout arc E de ce chemin, on choisit une valeur $\omega \in \text{Itv}(E) \cap \text{Dom}(\text{Var}(E))$ (comme ϕ est réduit, on sait que cet ensemble n'est pas vide), et on l'assigne à \vec{x} : $\vec{x}(\text{Var}(E)) := \omega$, sauf si cette variable a déjà été rencontrée (on sait que la valeur choisie auparavant est aussi compatible avec cet arc, car ϕ est convergent) ou si $\text{Var}(E) = \perp$. Quand on atteint la racine, \vec{x} est un modèle de ϕ (pour toutes les variables non rencontrées, on a assigné dès le début \vec{x} à une valeur compatible avec leur domaine). Cette procédure est faite en temps linéaire en $|\phi|$ (elle est même plus précisément linéaire en la hauteur de ϕ).

MC.

Inféré du fait que IA supporte MC.

$\wedge BC, \wedge C$.

Grâce à la prop. 6, tout OBDD peut être transformé en FIA en temps polynomial. Si FIA supportait $\wedge BC$, on aurait un algorithme polynomial pour décider si la conjonction de deux OBDDs (l'ordre des variables n'étant pas nécessairement le même dans chaque OBDD) est satisfiable, puisque FIA supporte CO; cependant, ce problème est NP-complet, comme cela a été montré dans le lemme 8.14 de (Meinel & Theobald, 1998). On en déduit que FIA ne supporte pas $\wedge BC$, et *a fortiori* ne supporte pas $\wedge C$.

\wedge tC.

On peut utiliser la même preuve que pour les IAs, puisque grâce au lemme 27 on sait que l'opération utilisée ne compromet pas la convergence.

SEN, EN.

Soient ϕ_1 et ϕ_2 deux FIAs. Soit $Z = \text{Var}(\phi_1) \cup \text{Var}(\phi_2)$, et une variable $x \notin Z$ de domaine \mathbb{R} . On construit l'automate ψ en fusionnant les puits de ϕ_1 et ϕ_2 , et en ajoutant un nœud qui sera la racine de ψ , étiqueté par x et ayant deux arcs sortants, l'un étiqueté par \mathbb{R}_- pointant vers la racine de ϕ_1 , et l'autre étiqueté par \mathbb{R}_+ pointant vers la racine de ϕ_2 . Il est évident que ψ est convergent, comme ϕ_1 et ϕ_2 le sont, et que x n'apparaît que dans un seul nœud. Nous allons prouver que $\text{ensure}(\text{I}(\psi), \{x\}) = \text{I}(\phi_1) \wedge \text{I}(\phi_2)$, c'est-à-dire que pour toute Z -instanciation \vec{z} , $(\text{I}(\phi_1) \wedge \text{I}(\phi_2))(\vec{z}) = \top \Leftrightarrow \forall \vec{x} \in \text{Dom}(\{x\}), \text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$.

(\Rightarrow) Soit \vec{z} une Z -instanciation telle que $(\text{I}(\phi_1) \wedge \text{I}(\phi_2))(\vec{z}) = \top$. Soit une $\{x\}$ -instanciation \vec{x} : on a soit $\vec{x}(x) \in \mathbb{R}_-$, auquel cas $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$, puisque $\text{I}(\phi_1)(\vec{z}) = \top$; soit $\vec{x}(x) \in \mathbb{R}_+$, auquel cas $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$ également, puisque cette fois $\text{I}(\phi_2)(\vec{z}) = \top$.

(\Leftarrow) Soit \vec{z} une Z -instanciation telle que $(\text{I}(\phi_1) \wedge \text{I}(\phi_2))(\vec{z}) = \perp$. On a soit $\text{I}(\phi_1)(\vec{z}) = \perp$, soit $\text{I}(\phi_2)(\vec{z}) = \perp$. Dans le premier cas, prenons \vec{x} tel que $\vec{x}(x) = -1$: il n'existe aucun chemin dans ψ compatible avec $\vec{x} \cdot \vec{z}$, puisque la valeur de x nous oblige à choisir l'arc menant à ϕ_1 , dans lequel aucun chemin n'est compatible avec \vec{z} . Symétriquement, dans le second cas, en prenant \vec{x} tel que $\vec{x}(x) = 1$, il n'existe aucun chemin dans ψ compatible avec $\vec{x} \cdot \vec{z}$. Donc il existe toujours un \vec{x} tel que $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \perp$.

On a donc prouvé que $\text{ensure}(\text{I}(\psi), \{x\}) = \text{I}(\phi_1) \wedge \text{I}(\phi_2)$. Il est possible de construire ψ en temps linéaire en la taille de ϕ_1 et ϕ_2 . Donc si FIA supportait SEN, nous aurions un algorithme polynomial permettant de construire un FIA équivalent à la conjonction de deux FIAs quelconques. Or cela est impossible sauf si $P = NP$ (voir \wedge BC). Donc FIA ne supporte pas SEN, et *a fortiori* EN.

CD, SCD.

Soit ϕ un FIA, $Y \subseteq \text{Var}(\phi)$, et \vec{y} une Y -instanciation. Soit $\phi_{|\vec{y}}$ l'IA défini dans la preuve de CD sur les IAs, et $\phi'_{|\vec{y}}$ l'IA obtenu en appliquant l'opération de réduction sur $\phi_{|\vec{y}}$. Nous allons montrer que $\phi'_{|\vec{y}}$ est convergent.

En effet, dans $\phi_{|\vec{y}}$ les seuls arcs à avoir été modifiés sont ceux dont l'arc correspondant dans ϕ est associé à une variable de Y . Ces arcs-là sont tous associés à \vee dans $\phi_{|\vec{y}}$.

En ce qui concerne les autres arcs de $\phi_{|\vec{y}}$, comme ils restent *tous* inchangés, ils sont encore convergents. On déduit du lemme 20 que $\phi'_{|\vec{y}}$ est convergent selon toutes les variables de $\text{Var}(\phi) \setminus Y = \text{Var}(\phi'_{|\vec{y}})$. Le lemme 21 montre alors que $\phi'_{|\vec{y}}$ est convergent. Parce que $\phi'_{|\vec{y}} \equiv \phi_{|\vec{y}}$, que $\text{I}(\phi_{|\vec{y}}) = \text{I}(\phi)_{|\vec{y}}$ et que $\phi'_{|\vec{y}}$ est obtenu en temps polynomial, FIA supporte CD (et par conséquent aussi SCD).

SFO.

Soit ψ un FIA réduit quelconque, tel que $\text{Var}(\psi) \neq \emptyset$, et $x \in \text{Var}(\psi)$. On note $\psi^{\downarrow x}$ le FIA obtenu en changeant tous les nœuds x de ψ par un nœud \vee et tous les arcs sortants de ces nœuds par des arcs \mathbb{R} . Nous allons montrer, par récurrence sur le nombre de nœuds de ψ , que $\text{I}(\psi^{\downarrow x}) = \text{forget}(\text{I}(\psi), \{x\})$.

Pour $n \in \mathbb{N}$, soit $\mathcal{P}(n)$ la proposition suivante : « Pour tout FIA réduit ψ tel que $\text{Var}(\psi) \neq \emptyset$ et ψ contient n nœuds, et tout $x \in \text{Var}(\psi)$, $\text{I}(\psi^{\downarrow x}) = \text{forget}(\text{I}(\psi), \{x\})$ ».

(I) $\mathcal{P}(0)$ et $\mathcal{P}(1)$ sont trivialement vraies : l'oubli de n'importe quelle variable de l'automate vide (resp. l'automate puits) est une fonction renvoyant toujours \perp (resp. \top), et ψ est exactement identique à $\psi^{\downarrow x}$ dans les deux cas.

(II) Soit $n \geq 2$. Supposons que $\mathcal{P}(k)$ est vraie pour tout $k < n$. Nous voulons en déduire que $\mathcal{P}(n)$ est également vraie. Soit ψ un FIA réduit tel que $\text{Var}(\psi) \neq \emptyset$ et ψ contient n nœuds, et $x \in \text{Var}(\psi)$. Soit $\text{Out}(\text{Root}(\psi)) = \{E_1, \dots, E_m\}$, et pour $1 \leq i \leq m$, notons ψ_i le sous-graphe dont la racine est $\text{Dest}(E_i)$. Il est évident que ψ_i contient strictement moins de n nœuds, donc pour tout $1 \leq i \leq m$,

$$\text{I}(\psi_i^{\downarrow x}) = \text{forget}(\psi_i, \{x\}) \quad . \quad (1)$$

Il est également clair que le FIA obtenu en remplaçant dans ψ chaque ψ_i par $\psi_i^{\downarrow x}$, et si $\text{Var}(\text{Root}(\psi)) = x$, en modifiant $\text{Root}(\psi)$ de façon à ce que $\text{Var}(\text{Root}(\psi)) = \perp$ et $\forall E \in \text{Out}(\text{Root}(\psi)), \text{Itv}(E) = \mathbb{R}$, correspond exactement à $\psi^{\downarrow x}$. Nous allons montrer à présent que $\text{I}(\psi^{\downarrow x}) = \text{forget}(\text{I}(\psi), \{x\})$. Soit $Z = \text{Var} \psi \setminus \{x\}$. Nous devons prouver que

$$\forall \vec{z} \in \text{Dom}(Z), \text{I}(\psi^{\downarrow x})(\vec{z}) = \top \quad \Leftrightarrow \quad \exists \vec{x} \in \text{Dom}(\{x\}), \text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top \quad .$$

Soit \vec{z} une Z -instanciation quelconque :

- (\Rightarrow) si $\text{I}(\psi^{\downarrow x})(\vec{z}) = \top$, il y a un chemin dans le graphe qui est compatible avec \vec{z} : soit $E_i^{\downarrow x}$ le premier arc de ce chemin, et $\psi_i^{\downarrow x}$ le sous-graphe vers la racine duquel pointe cet arc. Nous avons $\text{I}(\psi_i^{\downarrow x})(\vec{z}) = \top$, par définition de la sémantique d'un IA. Par conséquent, grâce à l'éq. (1), il existe une $\{x\}$ -instanciation \vec{x} telle que $\text{I}(\psi_i)(\vec{x} \cdot \vec{z}) = \top$. Soit E_i l'arc de ψ venant de $\text{Root}(\psi)$ et pointant vers la racine de ψ_i .
- si $\text{Var}(E_i) \neq x$, par construction de $\psi^{\downarrow x}$, E_i et $E_i^{\downarrow x}$ ont les mêmes variable et intervalle. Comme $E_i^{\downarrow x}$ est compatible avec \vec{z} , E_i l'est aussi, donc $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$.
 - si $\text{Var}(E_i) = x$, on peut supposer sans perte de généralité que $\vec{x}(x) \in \text{Itv}(E_i)$. En effet, $\text{I}(\psi_i)(\vec{x} \cdot \vec{z}) = \top$, donc il y a un chemin dans ψ_i qui est compatible avec \vec{x} : soit il y a un nœud x sur ce chemin, auquel cas $\vec{x}(x) \in \text{Itv}(E_i)$, puisque ψ est convergent ; soit il n'y en a pas, auquel cas n'importe quelle valeur de $\text{Dom}(x)$ peut être choisie (et $\text{Dom}(x) \cap \text{Itv}(E_i) \neq \emptyset$, car ψ est réduit). Donc $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$.
- (\Leftarrow) si $\text{I}(\psi^{\downarrow x})(\vec{z}) = \perp$, considérons un quelconque $E_i^{\downarrow x} \in \text{Out}(\text{Root}(\psi^{\downarrow x}))$, $\psi_i^{\downarrow x}$ le sous-graphe vers la racine duquel il pointe, et E_i l'arc qui lui correspond dans ψ (c'est-à-dire celui qui pointe vers la racine de ψ_i):
- si soit $\text{Var}(E_i^{\downarrow x}) = \perp$ soit $\vec{z}(\text{Var}(E_i^{\downarrow x})) \in \text{Itv}(E_i^{\downarrow x})$, nous savons que $\text{I}(\psi_i^{\downarrow x})(\vec{z}) = \perp$. Alors pour toute $\{x\}$ -instanciation \vec{x} , $\text{I}(\psi_i)(\vec{x} \cdot \vec{z}) = \perp$ (grâce à l'éq. (1)), donc E_i n'appartient à aucun chemin de ψ compatible avec $\vec{x} \cdot \vec{z}$.
 - dans les autres cas, nous avons (i) $\text{Var}(E_i^{\downarrow x}) \neq \perp$ et (ii) $\vec{z}(\text{Var}(E_i^{\downarrow x})) \notin \text{Itv}(E_i^{\downarrow x})$. Par construction de $\psi^{\downarrow x}$, (i) implique que E_i et $E_i^{\downarrow x}$ ont les mêmes variable et intervalle. Nous déduisons de ceci et de (ii) que $\vec{z}(\text{Var}(E_i)) \notin \text{Itv}(E_i)$, donc on ne peut pas trouver de $\{x\}$ -instanciation \vec{x} telle que E_i appartienne à un chemin dans ψ compatible avec $\vec{x} \cdot \vec{z}$.

Nous en concluons qu'il n'existe pas de $\{x\}$ -instanciation \vec{x} telle que $\text{I}(\psi)(\vec{x} \cdot \vec{z}) = \top$.

Les deux points prouvent que $\text{I}(\psi^{\downarrow x}) = \text{forget}(\text{I}(\psi), \{x\})$; par conséquent $\mathcal{P}(n)$ est vraie.

(III) Puisque les étapes d'initialisation (I) et de récurrence (II) ont été prouvées, nous avons montré par récurrence que $\mathcal{P}(n)$ est vraie quel que soit $n \in \mathbb{N}$.

Comme pour n'importe quel FIA réduit ψ , la construction de $\psi^{\downarrow x}$ ne nécessite de rencontrer chaque arc et chaque nœud qu'une seule fois, on peut obtenir, en temps linéaire en la taille de ψ , un FIA dont la fonction d'interprétation est égale à $\text{forget}(\text{I}(\psi), \{x\})$. La réduction étant polynomiale sur les FIAs, cela prouve que FIA supporte SFO.

FO.

L'opération de *single forgetting* peut être effectuée en temps linéaire sur les FIAs réduits, par conséquent l'opération de *forgetting* générale est polynomiale sur les FIAs réduits. La réduction étant polynomiale sur les FIAs, cela prouve que FIA supporte FO. □