Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques

Pierre Deransart, Mireille Ducassé, Gérard Ferrand

Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques

Pierre Deransart, Mireille Ducassé et Gérard Ferrand

INRIA Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex
IRISA/INSA Campus universitaire de Beaulieu, 35042 Rennes Cedex
LIFO, Université d'Orléans, BP 6759, 45067 Orléans Cedex 2
Pierre.Deransart@inria.fr, Ducasse@irisa.fr, Gerard.Ferrand@univ-orleans.fr

Résumé

Dans cet article on étudie une présentation originale du modèle des boîtes de Byrd basée sur la notion de sémantique observationnelle. Cette approche permet de rendre compte de la sémantique des traceurs Prolog indépendamment d'une implantation particulière.

Le schéma explicatif obtenu est une présentation formelle épurée d'une trace considérée en général comme plutôt obscure et difficile à utiliser. Il peut constituer une approche simple et pédagogique tant pour l'enseignement (par sa forme épurée) que pour les implantations de traceurs Prolog dont il constitue une forme de spécification.

Ceci, en fait, n'est qu'un exemple pour illustrer une problématique générale relative aux traceurs et aux processus observants qui ne connaissent du processus observé que sa trace. La question est alors de pouvoir reconstituer par l'analyse de la trace l'essentiel du processus observé, et si possible, sans perte d'information.

Notre approche met en évidence les qualités du modèle des boîtes qui en ont fait son succès, mais aussi ses inconvénients et ses limites.

Abstract

This article specifies an observational semantics and gives an original presentation of the Byrd's box model. The approach accounts for the semantics of Prolog tracers independently of a particular implementation.

Traces are, in general, considered as rather obscure and difficult to use. The proposed formal presentation of a trace constitutes a simple and pedagogical approach for teaching Prolog or for implementing Prolog tracers. It constitutes a form of declarative specification for the tracers.

As a matter of fact, the presented semantics is only one example to illustrate general problems relating to tracers and observing processes. Observing processes know, from observed processes, only their traces. The issue is then to be able to reconstitute by the sole analysis of the trace the main part of the observed process, and if possible, without any loss of information.

Our approach highlights qualities of the box model which made its success, but also its drawbacks and limits.

1 Introduction

Ce papier présente un modèle de trace Prolog (souvent appelé "modèle des boîtes de Byrd") d'une manière originale, basée sur la notion de sémantique observationnelle (SO). Cette sémantique a été introduite dans [12] afin de rendre compte de la sémantique de traceurs indépendamment de la sémantique du processus tracé.

Ce n'est pas l'objet de ce papier d'étudier cette sémantique. Notre objectif est de l'illustrer ici avec un exemple simple mais non trivial. Le résultat est une sémantique originale de la trace Prolog telle qu'usuellement implantée, sans tenir compte d'une implantation particulière, ni décrire la totalité du processus de résolution. Une telle sémantique constitue également une forme de spécification formelle de traceur Prolog et permet d'en comprendre facilement quelques propriétés essentielles.

"Comprendre une trace" c'est d'une certaine manière tenter de retrouver le fonctionnement du processus tracé à partir d'un état initial connu et d'une suite d'événements de trace. Cette démarche suppose une connaissance suffisante mais non nécessairement complète du modèle de fonctionnement du processus, et de savoir relier les événements de trace à ce modèle. C'est cette démarche que nous voulons capturer avec la

notion de sémantique observationnelle, et les fonctions d'extraction de trace et de reconstruction du modèle original à partir de la trace, c'est à dire d'"adéquation" de la trace au modèle observé.

Le "modèle des boîtes" a été introduit pour la première fois par Lawrence Byrd en 1980 [3] dans le but d'aider les utilisateurs du "nouveau" langage Prolog (il fait alors référence aux implantations d'Edinburgh [13] et de Marseille[16]) à maîtriser la lecture opérationnelle du déroulement du programme. Dès les débuts, en effet, les utilisateurs se sont plaints des difficultés de compréhension du contrôle liées au non déterminisme des solutions. Même si par la suite d'autres modèles ont été adoptés avec des stratégies bien plus complexes[1], les quatre "ports" introduits par Byrd (**Call**, **Exit**, **Redo** et **Fail**), associés aux quatre coins d'une boîte et manipulables dans une sorte d'algèbre de poupées russes, sont restés célèbres et se trouvent dans toutes les traces des systèmes Prolog encore existants.

Le modèle des boîtes de Byrd fascine par sa simplicité apparente. Toujours et souvent cité mais rarement bien expliqué, le modèle des boîtes garde l'aura des premiers essais réussis. C'est sans doute pour cela qu'il reste l'objet de publications ponctuelles mais régulières depuis 1980, comme [1] (1984), [17] (1993), [10] (2000), [11] (2003). Pour autant, il reste souvent difficile à "expliquer" parce que ses diverses définitions sont soit trop informelles, soit noyées dans une formalisation complète de la sémantique de Prolog.

Dans cet article nous proposons une description formelle d'une variante du modèle initial défini informellement par Byrd en 1980. L'originalité de cette description réside dans le fait que, bien que contenant les ingrédients du modèle original et limitée aux éléments de contrôle dont elle veut rendre compte (c'est à dire, sans ou en tous cas, le moins possible, faire référence aux mécanismes de choix de clauses et d'unification propres à la résolution Prolog), elle est formellement complète.

Après une introduction aux traces et leur sémantique observationnelle (sections 2 et 3), nous présentons la SO qui spécifie le modèle des boîtes (section 4) et l'extraction de la trace (section 5). Enfin nous donnons le modèle de reconstruction (section 6) établissant ainsi une grille de lecture possible de la trace, basée sur la SO.

Notre approche met en évidence les qualités du modèle des boîtes qui en ont fait son succès, mais aussi ses défauts principaux (section 7). Elle montre aussi l'intérêt potentiel de l'approche observationnelle. On trouvera également cette étude plus détaillée dans le rapport complet de mêmes titre et auteurs [6].

[1]Le modèle de Byrd se limite à la stratégie standard de parcours-construction d'arbre.

2 Introduction aux traces

Nous donnons ici un aperçu rapide du contexte de cette étude. Pour plus de détails sur les motivations on pourra se reporter à [5] et [12].

D'une manière générale, on veut s'intéresser à l'observation de processus dynamiques à partir des traces qu'ils laissent ou qu'on leur fait produire[2].

On peut toujours considérer qu'entre un observateur et un phénomène observé il y a un objet que nous appellerons "trace". La trace est l'empreinte reconnaissable laissée par un processus et donc "lisible" par d'autres processus. Le phénomène observé sera considéré ici comme un processus fermé (ceci concernant toutes les données et fonctions qu'il manipule) dont on ne connaît que la trace. La trace est une suite d'événements représentant l'évolution d'un état "abstrait" qui contient tout ce que l'on peut ou veut connaître de ce processus. Celle-ci peut être formalisée par un modèle de transition d'états, c'est à dire par une suite d'états et une fonction de transition formalisant le passage d'un état à un autre. Cette sémantique sera appelée *sémantique observationnelle* car elle représente ce que l'on est susceptible de connaître ou de décrire du processus, vu de l'"extérieur".

La SO se caractérise par le fait que chaque transition donne lieu à un événement de trace. Si une trace peut être infinie, les différents types d'actions (ou d'ensemble d'actions) du processus observé réalisant les transitions sont supposées en nombre fini. Ce sont les *ports* de la trace. A chaque port peut correspondre plusieurs transitions. On considèrera ici que la SO est spécifiée par son ensemble fini de transitions nommées, noté R.

Pour formaliser cette approche on introduit la notion de trace intégrale virtuelle.

Definition 1 (Trace intégrale virtuelle) *Une trace intégrale virtuelle est une suite d'évènements de trace qui sont de la forme* $e_t : (t, a_t, S_{t+1}),\ t \geq 0$ *où :*

- e_t *: est l'***identificateur** *unique de l'évènement.*
- t *: est le* **chrono***, temps de la trace. C'est un entier incrémenté d'une unité à chaque évènement.*

[2]Il faut bien distinguer ce qui relève de ce que nous appelons ici "trace" et ce qui relève d'outils l'analyse de processus ("monitoring", présentation particulière de la trace ou "jolies" impressions, visualisation, analyse de performance, débogage, ...) qui tous d'une manière ou d'une autre, directement ou indirectement, de l'intérieur ou indépendemment, en mode synchrone ou asynchrone, utilisent ce que nous appelons une "trace virtuelle". Cette étude n'est pas concernée par la nature ni la forme de ces processus observants.

- $S_{t+1} = p_{1,t+1}, ..., p_{n,t+1}$ *: sont des valeurs des* **paramètres** $p_1, ..., p_n$ *résultant de l'action accomplie au temps* t.
- a_t *: un identificateur d'***action** *caractérisant le type des actions réalisées pour effectuer la transition de l'état* S_t *à* S_{t+1}*, aussi appelé* port.

Une trace est produite à partir d'un état initial noté S_0 *et peut être infinie. Une suite finie d'événements de trace* $e_t e_{t-1} \dots e_0$ *de taille* $t+1, t \geq 0$ *sera dénotée* ${e_t}^+$ *(*${e_t}^*$ *si la suite vide est incluse). La suite vide sera dénotée* ϵ *et on conviendra que* ${e_0}^+ = e_0$ *et* ${e_0}^* = \epsilon$*. Une portion finie non vide de trace sera dénotée* $< S_0, e_t^+ >$.

La trace intégrale virtuelle représente ce que l'on souhaite ou ce qu'il est possible d'observer d'un processus donné. Comme l'état courant (virtuel) du processus est intégralement décrit dans cette trace, on ne peut espérer, ni la produire, ni la communiquer efficacement. En pratique on effectuera une sorte de compression, et on s'assurera que le processus observant puisse la "décompresser". La trace effectivement diffusée sera extraite de la trace virtuelle et communiquée sous forme de *trace actuelle*.

Definition 2 (Trace actuelle, schéma de trace) *Une trace actuelle est une suite d'évènements de trace de la forme* e_t : (t, a_t, A_t) *dérivés de la transition* $< S_t, S_{t+1} >$ *par la fonction*[3] $\mathcal{E}$*, dite* fonction d'extraction*, où chaque* e_t *est obtenu par* $e_t = \mathcal{E}(S_t, S_{t+1})$.

Si $A_t = S_{t+1}$*, la trace actuelle est la trace intégrale virtuelle.*

A_t *dénote une suite finie de valeurs d'attributs. La fonction d'extraction est une famille de fonctions définies pour chaque règle de transition* r *de la SO. Soit :* $\mathcal{E} = \{\mathcal{E}_r | r \in R\}$ *telle que* $\forall < r, S, S' > \in SO,\ e_r = \mathcal{E}_r(S, S')$.

La description de la famille de fonctions $\mathcal{E}_r$ *constitue un* schéma de trace.

La question se pose maintenant de l'utilité d'une trace, c'est à dire la possibilité de reconstruire une suite d'états, éventuellement partiels, à partir d'une trace produite, sans le recours direct à la SO, mais qui corresponde, pas à pas, aux transitions de la SO qui ont produit cette trace. C'est ce que tente de capturer la notion d'adéquation.

La trace est alors vue comme une suite d'informations qui permet de reproduire une suite d'états actuels dont chacun est la restriction, notée S/Q, à une partie Q de l'état virtuel S correspondant.

[3]la trace actuelle est définie par $\mathcal{E}$ qui sera sous-entendue par la suite.

Definition 3 (Trace adéquate) *Etant donnés un état actuel* Q *restriction de* S *à un sous ensemble de ses paramètres et une SO définie sur* S *par un ensemble fini de transitions* R.

Une trace actuelle $T_w =< Q_0, w_t^* >$ *telle que* $Q_0 = S_0/Q$ *est* adéquate *pour* Q *par rapport à la trace virtuelle intégrale* $T_v =< S_0, v_t^* >$ *s'il existe une fonction* $\mathcal{F}$ *telle que*

$$\forall t \geq 0,\ \mathcal{F}(w_t^*, Q_0) = Q_{t+1} \quad et$$
$$\forall i \in [0..t-1],\ Q_i = S_i/Q \ \wedge \exists r \in R, \quad tel\ que\ w_i = \mathcal{E}_r(S_i, S_{i+1}).$$

L'adéquation stipule qu'à toute suite d'états, engendrée par une trace actuelle, il correspond une suite de transitions de la SO qui a engendré cette trace et dont la suite des états restreints est la même[4].

Il peut être essentiel que la trace actuelle soit adéquate par rapport à Q, garantissant ainsi la transmission et compréhension complètes des états partiels que l'on peut retrouver alors par la seule lecture de la trace.

On indique ici des conditions pour prouver l'adéquation d'une trace actuelle qui utilisent des couples d'événement de trace. En effet un événement de trace actuelle, produit par une transition $< S, S' >$ en appliquant une règle r, peut ne pas comporter suffisamment d'attributs pour restituer les paramètres souhaités de l'état S' (ceux qui se trouvent dans l'événement correspondant de la trace virtuelle intégrale). Il est donc parfois nécessaire de recourir à deux événements de trace pour pouvoir reconstruire les paramètres souhaités de l'état courant obtenu. La fonction de reconstruction utilise donc deux événements de trace. Elle sera décrite par une famille de *fonctions locales de reconstruction* d'états restreints Q $\mathcal{C} = \{\mathcal{C}_r | r \in R\}$ telle que $\forall < r, S, S' > \in SO,\ Q' = \mathcal{C}_r(e, e', Q)$. La description des fonctions locales de reconstruction constitue un *schéma de reconstruction*.

Par ailleurs il est nécessaire de pouvoir associer à un événement de trace la transition, donc la règle de la SO, qui l'a produit. Pour ce faire on utilise également une famille de conditions $Cond_r(e, e')$ qui, étant donné un couple d'événements de trace, identifient sans ambiguité la règle r utilisée donc la transition qui a produit le premier évènement e.

Proposition 1 (Condition d'adéquation) *Etant donnés une SO définie avec un ensemble de règles* R*, un schéma de trace* $\mathcal{E}$ *et un schéma de reconstruction* $\mathcal{C}$ *pour un sous-ensemble de paramètres* Q*. Si les deux propriétés suivantes sont satisfaites pour chaque règle* $r \in R$ *:*

[4]Le problème à la limite, dans le cas d'une trace finie, lié à la nature d'un éventuel état terminal, sera omis ici.

$\forall\, e,\, e',\, r', S,\, S',\, S'',$
$\mathcal{E}_r(S, S') = e \ \wedge\ \mathcal{E}_{r'}(S', S'') = e'$
(1) seule $Cond_r(e, e')$ *est vraie, i.e.* $Cond_r(e, e') \bigwedge_{s \neq r} \neg Cond_s(e, e')$.
(2) $\mathcal{C}_r(e, e', S/Q) = S'/Q$.
alors la trace actuelle $T_w =< S_0/Q, w_t^* >$*, définie par le schéma de trace* $\mathcal{E}$*, est adéquate pour* Q *par rapport à la trace virtuelle intégrale* $T_v =< S_0, v_t^* >$.

3 Sémantique Observationnelle et fonctions associées

La Sémantique Observationnelle (SO) se distingue d'une sémantique opérationnelle par le fait que son objet est avant tout la description d'un flot de données éventuellement infini sans faire explicitement référence à un processus particulier ni une sémantique concrète particulière.

La Sémantique Observationnelle (SO) rend compte de toutes les traces virtuelles possibles, c'est à dire de toutes les suites d'états décrits par un ensemble fini de paramètres, et définies par un état initial, une fonction de transition d'états, et telle qu'à chaque transition un élément de trace puisse être produit. Une SO est donc définie par un domaine d'états et une fonction de transition d'états.

Dans la SO la fonction de transition est décrite par un ensemble fini de règles nommées. L'application d'une règle produit un évènement de trace. Une règle a quatre composants.

- Un identificateur de règle (nom).
- Un numérateur comportant des conditions sur l'état antérieur et des calculs de transition.
- Un dénominateur comportant la description de l'état obtenu (ce qui reste invariant peut être omis).
- Des conditions externes (entre accolades) ou propriétés portant sur des éléments non décrits par des paramètres, mais intervenant dans le choix des règles ou les valeurs des paramètres.

Noter que la distinction entre les éléments figurants au numérateur et dans les accolades est arbitraire. Toutefois, toute expression contenant des éléments externes sera dans les accolades.

Chaque règle de transition de la SO sera présentée formellement par un triplet[5] $< nom, S, S' >$ où par abus de notation on dénotera S les conditions portant sur un état courant S_t et auquel la transition peut alors s'appliquer, et par S' l'état résultant de la transition (dont l'instance est alors S_{t+1}), mais simplement décrit ici par les calculs des nouvelles valeurs de paramètres.

[5]Pour la raison indiquée plus haut on omettra ici la partie externe.

On y ajoutera également d'éventuels facteurs externes décrits par des *conditions externes*. Une règle sera donc présentée de la manière suivante.

$$\textsf{Nom}\ \frac{Conds\, caract.\, l'etat\, courant}{Calcul\, des\, nouv.\, parametres}\ \{Conds\, externes\}$$

Pour décrire la SO, on utilisera deux types de fonctions : celles relatives aux objets décrits et leur évolution dans la trace virtuelle et celles relatives à des évènements ou objets non décrits dans cette trace, mais susceptibles de se produire dans les processus observés et d'y être interprétées. Les fonctions de la première catégorie sont dites "utilitaires", celles de la seconde "externes". Elles concernent des paramètres non pris en compte dans la trace virtuelle. Enfin on distinguera également les fonctions exclusivement utilisées pour le calcul des attributs lors de l'extraction de la trace, dites "auxiliaires d'extraction" et celles exclusivement utilisées pour la reconstruction dites "auxiliaires de reconstruction".

La fonction d'extraction $\mathcal{E}$ sera décrite par le même type de règles, mais leur dénominateur comportera exclusivement l'évènement de trace actuelle correspondant, c'est à dire le port et les attributs. Il y a un seul évènement de trace par règle . L'ensemble des règles qui décrivent la fonction d'extraction constituent un *schéma de trace*.

Chaque règle a la forme

$$\textsf{Nom}\ \frac{Calcul\ des\ attributs}{< Evenement\ de\ trace >}\ \{Cond.\ externes\}$$

La description de la reconstruction utilisera une fonction locale de reconstruction $\mathcal{C} = \{\mathcal{C}_r | r \in R\}$. Elle sera décrite avec le même type de règles.

$$\textsf{Nom}\ \frac{Cond\ d'identification}{Calculs\ de\, reconstruction}\ \{Evnmts\ de\ trace\}$$

La trace est cette fois considérée comme une information "externe" et se situe en position de composant externe (dans les accolades, où il y a au plus deux événements de trace). Le numérateur de la règle contient la condition permettant d'identifier la règle de la SO qui s'applique (condition de "compréhensibilité). Le dénominateur contient les calculs de reconstruction (calcul des paramètres de l' état virtuel restreint à partir de la trace). L'ensemble des règles de reconstruction constitue un *schéma de reconstruction*.

Noter que les trois ensembles de règles (SO, schémas de trace et de reconstruction) sont en bijection deux à deux.

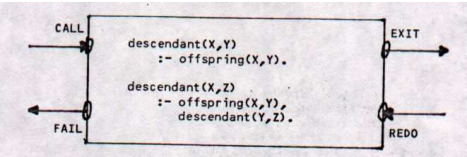

FIG. 1 – Modèle de boîte tel que dessiné par Byrd

4 Une sémantique Observationnelle du modèle des boîtes

Dans ses articles [3] [2], Byrd illustre son modèle à l'aide de deux schémas : une boîte avec les quatre fameux ports (voir figure 1) et un "arbre et/ou", structure très répandue alors qui combine les représentations d'arbre de preuve et d'arbre de recherche. Il n'utilise ni la notion d'arbre de preuve partiel ni celle d'arbre de recherche (arbre SLD) encore peu connues, le rapport de Clark [4] venant à peine de paraître.

Byrd fustige, néanmoins, les implanteurs qui, lors du retour arrière (ce qui se traduit dans la trace par un évènement de port **Redo**) vont directement au point de reprise et n'expriment pas dans la trace tout le cheminement inverse. Byrd estime que ceci est de nature à perdre l'utilisateur et qu'il est préférable de défaire pas à pas ce qui a été explicitement fait lors des recours successifs aux clauses pour résoudre des buts.

Même si nous voulons rester le plus proche possible de ce modèle, nous ne suivrons cependant pas ce point de vue et adopterons celui des implanteurs, plus répandu actuellement, et qui nous semble tout aussi facile à comprendre à partir du moment où tout ce qui est utile est formalisé. En effet, le modèle des boîtes oblige à suivre les appels de clauses à travers un système de boîtes encastrées. Il est alors facile de comprendre qu'à partir du moment où chaque boîte a un identificateur unique, l'accès à un point de choix profondément enfoui dans les profondeurs de l'empilement peut se faire aussi clairement en sautant directement sur la bonne boîte qu'en descendant l'escalier résultant de l'empilement, ou en faisant le cheminement inverse. On évitera ainsi de détailler explicitement la manière d'accéder à la bonne boîte.

Même si, au final, nous ne décrivons pas exactement le modèle initialement défini par Byrd, nous estimons que nous en gardons les éléments historiquement essentiels, à savoir le parcours-construction d'arbre et les boîtes dans lesquelles les clauses, ou un sous-ensemble, sont stockées. Nous qualifierons notre approche de *modèle des boîtes simplifié*.

L'empilement des boîtes et son évolution seront donc décrits par un parcours-construction d'arbre dont chaque nœud correspond à une boîte. La stratégie de parcours correspond à la stratégie de Prolog standard (ISO Prolog [7]), celle d'un parcours-construction descendant gauche droite. Chaque nœud nouveau, ou boîte, reçoit un numéro qui est incrémenté de 1 à chaque création.

Chaque nœud est étiqueté avec une prédication et un paquet de clauses. Chaque boîte est donc la racine d'un sous-arbre réalisant ainsi un jeu de boîtes encastrées.

Dans la mesure du possible nous utilisons le vocabulaire ISO-Prolog [7].

Paramètres de la trace virtuelle

L'état courant comporte 9 paramètres :

$$\{T, u, n, num, pred, claus, first, ct, flr\}.$$

1. T : T est un arbre étiqueté avec un numéro de création, une prédication et un sous-ensemble de clauses du programme P. Il est décrit ici par ses fonctions de construction-reconstruction et parcours (cf plus bas) et étiquetage. Aucune représentation particulière n'est requise. Nous utiliserons cependant dans les exemples une notation "à la Dewey". Chaque nœud est représenté par une suite de nombres entiers et dénotés ϵ, 1, 11, 12, 112, ... L'ordre lexicographique est le suivant : u, v, w sont des mots, $ui < uiv (v \neq \epsilon)$, et $uiv < ujw$ si $i < j$, ϵ est la suite vide.
2. $u \in T$: u est le nœud courant dans T (boîte visitée).
3. $n \in \mathcal{N}$: n est un entier positif associé à chaque nœud dans T par la fonction num (ci-dessous). C'est le numéro du dernier nœud créé.
4. $num : T \rightarrow \mathcal{N}$. Abbrev. : $nu(v)$. $nu(v)$ est le numéro (entier positif) associé au nœud v dans T.
5. $pred : T \rightarrow \mathcal{H}$. Abbrev. : $pd(v)$. $pd(v)$ est la prédication associée au nœud v dans T. C'est un élément de l'ensemble d'atomes non clos $\mathcal{H}$ (base de Herbrand non close).
6. $claus : T \rightarrow 2^P$. Abbrev. : $cl(v)$. $cl(v)$ est une liste de clauses de P (même ordre que dans P) contribuant à la définition du prédicat $pred(v)$ associée au nœud v dans T. [] est la liste vide. Selon les clauses de $cl(v)$, on peut obtenir différents modèles. On ne met ici dans la boîte v que les clauses dont la tête est unifiable avec la prédication $pred(v)$. Si la boîte est vide, la predication $pred(v)$ ne peut être résolue et le nœud sera en échec (cf. *failure*). Cette liste de clauses est définie par un ordre externe lorsque la predication est appelée (voir *claus_pred_init* dans les fonctions externes) et mise à jour chaque fois que le

nœud est visité (voir *update_claus_and_pred* dans les fonctions utilitaires).

7. $first : T \rightarrow Bool$. Abbrev. $fst(v)$. $fst(v)$ est vrai ssi v est un nœud de T qui n'a pas encore été visité (c'est une feuille).
8. $ct \in Bool$: ct est l'indicateur de construction achevée (complète) de T : $true$ ssi le nœud courant devient ϵ (retour à la racine) lors d'une remontée dans l'arbre (en succès ou échec).
9. $flr \in Bool$: flr est l'indicateur d'état d'échec du sous-arbre ($true$ si en échec, $false$ sinon, ce qui n'est pas synonyme de succès).

Etat initial :

$\{\{\epsilon\}, \epsilon, 1, \{(\epsilon, 1)\}, \{(\epsilon, goal)\}, \{(\epsilon, list\_of\_goal\_claus)\}, \{(\epsilon, true)\}, false, false\ \}$

On écrira éventuellement T pour $true$ et F pour $false$ pour abbréger dans les formules s'il n'y a pas de confusion possible. Le modèle est basé sur un parcours-construction d'arbres de preuve partiels, construits puis reconstruits après des retours arrières. Les nœuds ne sont construits que juste avant d'être visités. La relation avec le modèle des boîtes de Byrd est fondée sur l'idée que chaque nœud est une boîte qui contient les clauses susceptibles de donner des développements alternatifs. Si la boîte est vide au moment de sa création, le nœud sera en échec. Chaque visite d'un nœud (boîte) donne lieu à un événement de trace.

Fonctions utilitaires (manipulation sur les objets décrits) : les fonctions sont présentées dans l'ordre des objets qu'elles concernent.

- $parent : T \rightarrow T$. Abbrev. : $pt(v)$. $pt(v)$ est l'ancêtre direct de v dans T. Pour simplifier le modèle, on suppose que $pt(\epsilon) = \epsilon$.
- $leaf : T \rightarrow Bool$. Abbrev. : $lf(v)$. $lf(v)$ est vraie ssi v est une feuille dans T.
- $may\_have\_new\_brother : T \rightarrow Bool$. Abbrev. : $mhnb(v)$. $mhnb(v)$ est vrai ssi $pred(v)$ n'est pas la dernière prédication dans le corps de la clause courante, elle-même la première clause dans la boîte du nœud parent de v dans T. La racine (ϵ) n'a pas de frère.
- $create\_child : T \rightarrow T$. Abbrev. : $crc(v)$. $crc(v)$ est le nouvel enfant de v dans T.
- $create\_new\_brother : T \rightarrow T$. Abbrev. : $crnb(v)$. $crnb(v)$ est le nouveau frère de v dans T. Defini si v différent de ϵ.
- $has\_a\_choice\_point : T \rightarrow Bool$. Abbrev. : $hcp(v)$. $hcp(v)$ est vrai ssi il existe un point de choix w dans le sous-arbre de racine v dans T ($claus(w)$ contient au moins une clause).
- $greatest\_choice\_point : T \rightarrow T$. Abbrev. : $gcp(v)$. $w = gcp(v)$ est le plus grand point de choix dans le sous-arbre de racine v (dans T, $claus(w)$ contient au moins une clause) selon l'ordre lexicographique des nœuds dans T .
- $fact : T \rightarrow Bool$. Abbrev. : $ft(v)$. $ft(v)$ est vrai ssi la première clause dans $claus(v)$ est un fait.
- $update\_number : F, T \rightarrow F$. Abbrev. : $upn(nu, v)$. $upn(nu, v)$ met à jour la fonction num en supprimant toutes les références aux nœuds déconstruits de T jusqu'au nœud v (conservé),
- $update\_claus\_and\_pred : F, T, \mathcal{H} \rightarrow F$. Abbrev. : $upcp(pd, v, p)$**,** $upcp(pd, v)$ **ou** $upcp(cl, v)$. (F ensemble de fonctions) : $upcp(claus, v)$, $upcp(pred, v)$ (2 arguments) ou $upcp(pred, v, p)$ (3 arguments) : met à jour les fonctions $claus$ et $pred$ en supprimant toutes les références aux nœuds déconstruits de T jusqu'au nœud v (conservé), et mettant également à jour, si cela est requis par la fonction externe *pred_update*, la valeur de $pred(v)$ avec la paire (v, p) ainsi que les valeurs de la fonction $claus$ au nœud v en enlevant la dernière clause utilisé.

Fonctions externes :

Elles correspondent aux actions non décrites dans la trace virtuelle mais qui l'influencent effectivement, en particulier tous les aspects de la résolution liés à l'unification et qui sont omis dans cette SO.

- $success : T \rightarrow Bool$. Abbrev. : $scs(v)$. $scs(v)$ est vrai ssi v est une feuille et la prédication courante a été unifiée avec succès avec la tête de la clause utilisée dans cette boîte.
- $failure : T \rightarrow Bool$. Abbrev. : $flr(v)$. $flr(v)$ est vrai ssi v est une feuille et aucune clause du programme ne s'unifie avec la prédication courante (dans ce modèle la boîte ne contient alors aucune clause).
- $claus\_pred\_init : T \rightarrow (pred, list\_of\_clauses)$. Abbrev. : $cpini(v)$. $(c, p) = cpini(v)$ met à jour 1- la fonction $claus$ avec la paire (v, c) où c est la liste des clauses dont la tête est unifiable avec la prédication $pred(v)$ et qui sont donc utilisables pour essayer différentes alternatives pour la résolution (si la liste est vide il n'y a pas de solution), et 2- la fonction $pred$ avec la paire (v, p) où p est la prédication à associer au nœud v. On notera $c\_cpini(v)$ et $p\_cpini(v)$ les arguments respectifs (clauses et prédication) résultants de $cpini(v)$.
- $pred\_update : T \rightarrow \mathcal{H}$. Abbrev. : $pud(v)$. $pud(v)$ est la nouvelle valeur de la prédication attachée au nœud v de T, suite à une unification réussie.

Noter que $\forall u, flr(u) \Rightarrow flr = true$ (voir règle Tree failed)

La sémantique observationnelle est décrite figure 2, chaque règle est commentée dans ce qui suit.

$$\textsf{Leaf reached}\ \frac{fst(u) \wedge lf(u) \wedge \neg ct \wedge ft(u)}{cl' \leftarrow upcp(cl,u),\quad fst'(u) \leftarrow F,\quad flr' \leftarrow F}\ \{\}$$

$$\textsf{Lf rcd \& go down}\ \frac{fst(u) \wedge lf(u) \wedge \neg ct \wedge \neg ft(u),\quad v \leftarrow crc(u)}{\begin{array}{c}T' \leftarrow T \cup \{v\},\quad u' \leftarrow v,\quad n' \leftarrow n+1,\quad nu' \leftarrow nu \cup \{(v,n')\},\quad pd' \leftarrow pd \cup \{(v,p)\},\\ cl' \leftarrow upcp(cl,u) \cup \{(v,c)\},\quad fst'(u) \leftarrow F,\quad fst' \leftarrow fst' \cup \{(v,T)\},\quad flr' \leftarrow F\end{array}}\ \{scs(u),\quad (c,p) = cpini(v)\}$$

$$\textsf{Tree success}\ \frac{\neg fst(u) \wedge \neg mhnb(u) \wedge \neg ct \wedge \neg flr,\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad pd' \leftarrow upcp(pd,u,p),\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T)}\ \{scs(u),\quad p = pud(u)\}$$

$$\textsf{Tree suc \& go right}\ \frac{\neg fst(u) \wedge mhnb(u) \wedge \neg ct \wedge \neg flr,\quad v \leftarrow crnb(u)}{\begin{array}{c}T' \leftarrow T \cup \{v\},\quad u' \leftarrow v,\quad n' = n+1,\quad nu' \leftarrow nu \cup \{(v,n')\},\\ pd' \leftarrow upcp(pd,u,p') \cup \{(v,p)\},\quad cl' \leftarrow cl \cup \{(v,c)\},\quad fst' \leftarrow fst \cup \{(v,T)\}\end{array}}\ \{scs(u),\quad p' = pud(u),\quad (c,p) = cpini(v)\}$$

$$\textsf{Tree failed}\ \frac{\neg fst(u) \wedge \neg ct \wedge \neg hcp(u),\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T),\quad flr' \leftarrow T}\ \{flr(u)\ \vee\ flr\}$$

$$\textsf{Backtrack}\ \frac{\neg fst(u) \wedge hcp(u) \wedge ft(v) \wedge (flr\ \vee\ ct),\quad v \leftarrow gcp(u)}{T' \leftarrow T - \{y | y > v\},\quad u' \leftarrow v,\quad cl' \leftarrow upcp(cl,v),\quad ct \Rightarrow (ct' \leftarrow F),\quad flr' \leftarrow F}\ \{\}$$

$$\textsf{Bkt \& gd}\ \frac{v \leftarrow gcp(u),\quad \neg fst(u) \wedge hcp(u) \wedge \neg ft(v) \wedge (flr\ \vee\ ct),\quad w \leftarrow crc(v)}{\begin{array}{c}T' \leftarrow T - \{y | y > v\} \cup \{w\},\quad u' \leftarrow w,\quad n' = n+1,\quad nu' \leftarrow upn(nu,v) \cup \{(w,n')\},\quad flr' \leftarrow F,\\ pd' \leftarrow upcp(pd,v) \cup \{(w,p)\},\quad cl' \leftarrow upcp(cl,v) \cup \{(w,c)\},\quad fst' \leftarrow fst \cup \{(w,T)\},\quad ct' \Rightarrow (ct \leftarrow F)\end{array}}\ \{scs(v),\quad (c,p) = cpini(w)\}$$

FIG. 2 – Semantique Observationnelle de la résolution Prolog (trace intégrale virtuelle)

- Leaf reached : Le nœud courant est une feuille et la prédication appelée doit être résolue par un fait. Ce nœud restera donc une feuille. Le point de choix est mis à jour (une clause de moins dans la boîte).
- Lf rcd & go down : Le nœud courant est une feuille mais la prédication associée est résolvable avec une clause dont la tête a été unifiée avec succès et dont le corps n'est pas vide. Ce nœud va être développé. Un nouveau nœud est créé dont la boîte v est remplie avec les clauses utiles (susceptibles de réussir) et une prédication appellante est associée. Le point de choix est mis à jour.
- Tree success : sortie en succès de la dernière prédication d'un corps de clause. $pred(u)$ est mis à jour (ce n'est pas nécessairement le même que lors de l'appel). Remontée en succès dans l'arbre sans création de nouvelle branche.
- Tree suc & go right : sortie en succès avec création d'une nouvelle branche "sœur" (nouvelle feuille v, cas du traitement d'une clause avec plus d'une prédication dans le corps). La boîte v est remplie avec les clauses utiles (susceptibles de réussir) et une prédication appelante est associée.
- Tree failed : remontée dans l'arbre en échec tant qu'il n'y a pas de point de choix dans le sous-arbre.
- Backtrack : reprise suite à succès ou échec, s'il y a un point de choix dans le sous-arbre ouvrant une possibilité de solution ou de nouvelle solution si on est à la racine. Comme discuté au début de cette section, dans ce modèle, on ne refait pas tous les "redo" en suivant le chemin jusqu'au point de reprise, comme dans le modèle original de Byrd.
- Bkt & go down : reprise suite à succès ou échec, s'il y a un point de choix dans le sous-arbre ouvrant une possibilité de solution ou une nouvelle solution si on est à la racine. Comme précédemment, mais avec création d'un descendant comme dans le cas de Lf rcd & go down.

Quel que soit l'état, une seule règle peut s'appliquer tant qu'un arbre complet n'a pas été construit ; aucune règle ne s'applique si l'arbre construit est complet et qu'il n'y a plus de point de choix.

5 Extraction de la trace actuelle

Chaque application d'une règle de la SO donne lieu à l'extraction d'un évènement de trace dont le chrono est incrémenté d'une unité à chaque fois. Pour l'extraction on a besoin d'une fonction auxiliaire.

Fonction auxiliaire d'extraction.

- $lpath : T \rightarrow \mathcal{N}$. Abbrev. : $lp(v)$. Byrd l'appelle la profondeur de récursion. $lp(v)$ est le nombre de nœuds sur le chemin de la racine au nœud v. C'est donc la longueur du chemin de la racine au nœud $+1$. $lp(\epsilon) = 1$.

Pour rendre compte des éléments propres à la trace de Byrd seulement (évolution de l'arbre ou des boîtes et étiquettes) on prendra pour état virtuel restreint l'ensemble des 4 paramètres suivants :

$$Q = \{T, u, num, pred\} \subseteq S.$$

La trace actuelle a 3 attributs et chaque événement a la forme

```
t    r    l    port    p
```

où

- `t` est le chrono.

– `r` est le numéro de création du nœud u concerné par l'évènement de trace, soit $nu(u)$.
– `l` est la profondeur dans l'arbre T du nœud concerné, soit $lp(u)$.
– `port` est l'identificateur d'action ayant produit l'évènement de trace (**Call, Exit, Fail** ou **Redo**).
– `p` est la prédication associée au nœud concerné, soit $pd(u)$.

L'exemple 1 ci-dessous présente un programme et la trace extraite correspondant au but `:-goal`.

```
c1: goal:-p(X),eq(X,b).
c2: p(a).
c3: p(b).
c4: eq(X,X).

:- goal.

chrono nu(u) lp(u) port    pd(u)  Etat virt.
  1     1     1    Call    goal      S2
  2     2     2    Call    p(X)      S3
  3     2     2    Exit    p(a)      S4
  4     3     2    Call    eq(a,b)   S5
  5     3     2    Fail    eq(a,b)   S6
  6     2     2    Redo    p(a)      S7
...
```

La fonction d'extraction $\mathcal{E}$ est décrite dans la figure 3.

$$\text{Leaf reached}\ \frac{}{< nu(u)\ \ lp(u)\ \ \textbf{Call}\ \ pd(u) >}\ \{\}$$

$$\text{Lf rcd \& go down}\ \frac{}{< nu(u)\ \ lp(u)\ \ \textbf{Call}\ \ pd(u) >}\ \{\}$$

$$\text{Tr suc}\ \frac{}{< nu(u)\ \ lp(u)\ \ \textbf{Exit}\ \ p >}\ \{p = pud(u)\}$$

$$\text{Ts \& gr}\ \frac{}{< nu(u)\ \ lp(u)\ \ \textbf{Exit}\ \ p >}\ \{p = pud(u)\}$$

$$\text{Tree failed}\ \frac{}{< nu(u)\ \ lp(u)\ \ \textbf{Fail}\ \ pd(u) >}\ \{\}$$

$$\text{Backtrack}\ \frac{v \leftarrow gcp(u)}{< nu(v)\ \ lp(v)\ \ \textbf{Redo}\ \ pd(v) >}\ \{\}$$

$$\text{Bkt \& go down}\ \frac{v \leftarrow gcp(u)}{< nu(v)\ \ lp(v)\ \ \textbf{Redo}\ \ pd(v) >}\ \{\}$$

FIG. 3 – Schéma de trace (fonction d'extraction)

Afin de faciliter la lecture, toutes les informations non nécessaires à l'extraction sont omises. En fait un évènement de trace est extrait lors de chaque transition de la SO, donc chaque règle peut se lire aussi avec l'ensemble des paramètres de l'état virtuel. Ainsi par exemple pour la règle Tree failed, le description complète de l'extraction $\mathcal{E}_{\text{Tree failed}}$ est :

$$\text{T fd}\ \frac{\neg fst(u) \wedge \neg ct \wedge \neg hcp(u),\quad v \leftarrow pt(u)}{\frac{u' \leftarrow v,\quad (u=\epsilon) \Rightarrow (ct' \leftarrow true),\quad flr' \leftarrow true}{<nu(u)\ \ lp(u)\ \ \textbf{Fail}\ \ pd(u)>}}\ \{flr(u) \vee flr\}$$

On peut y observer clairement la remontée "directe" dans l'arbre, suite à un échec (extraction d'évènement de port **Fail**), jusqu'à ce qu'un point de choix puisse se trouver dans le sous-arbre, ou jusquà la racine de l'arbre sinon.

6 Reconstruction d'une trace virtuelle restreinte

On décrit maintenant la fonction locale de reconstruction $\mathcal{C}$ de la trace virtuelle restreinte, à partir d'un état actuel et de la trace actuelle, ainsi que l'adéquation de la trace actuelle pour cet état relativement à la trace virtuelle.

Fonction auxiliaire de reconstruction :
Pour reconstruire l'état courant partiel, une fonction auxiliaire seulement est nécessaire, à savoir la fonction inverse de num, notée $node$.
– $node : \mathcal{N} \rightarrow T$. Abbrev. : $nd(v)$. Fonction inverse de num. $v = nd(n)$ est le nœud de T dont le rang de création est n (tel que $nu(v) = n$). Par définition $nd(nu(v)) = v$ et $nu(nd(n)) = n$.

Le schéma de reconstruction est donné dans la figure 4 par la famille $\{\mathcal{C}_r | r \in R\}$.

Chaque règle comporte en numérateur la condition d'identification de la règle à partir de la trace, au dénominateur les calculs du nouvel état virtuel restreint à partir des événements de trace qui figurent entre accolades et, éventuellement, des paramètres de l'état virtuel restreint courant.

Ces règles permettent en particulier de reconstruire pas à pas un arbre (ou de manière équivalente les boîtes encastrées), son parcours-construction-reconstruction, ainsi que les fonctions num et $pred$. L'état virtuel restreint comporte donc 4 paramètres, à savoir $Q = \{T, u, num, pred\}$.

Cette trace exige de lire deux évènements de trace successifs pour pouvoir être comprise.

Il faut aussi remarquer qu'à partir du moment où la trace est adéquate, et que l'on peut reconstituer ainsi le "fonctionnement" de la SO à partir de la trace, on peut rendre explicite une telle lecture de la trace en incluant dans les règles de reconstruction tous les paramètres de la trace virtuelle. A titre d'exemple, voici ce que donne la règle de reconstruction Lf rcd & go down avec tous les paramètres (figure 5).

Cette règle indique que si après un évènement de port **Call**, les numéros de boîtes croissent avec l'évènement de trace suivant ($r' > r$), alors c'est la règle Lf rcd & go down qui s'applique. Les conditions s'appliquant à l'état courant (sous-dénominateur du numérateur) sont alors vérifiées, un nœud v a été créé, descendant

$$\textsf{Leaf reached}\ \frac{r' = r}{}\ \{< r\ \ l\ \ \textbf{Call}\ \ p > ;\ < r' >\}$$

$$\textsf{Lf rcd \& go down}\ \frac{r' > r}{u' \leftarrow crc(nd(r)),\ \ T' \leftarrow T \cup \{u'\},\ \ nu'(u') \leftarrow r',\ \ pd'(u') \leftarrow p'}\ \{< r\ \ l\ \ \textbf{Call}\ \ p > ;\ < r' p' >\}$$

$$\textsf{Tree success}\ \frac{r' < r \vee u = \epsilon}{u' \leftarrow pt(u),\ \ pd'(u) \leftarrow p}\ \{< r\ \ l\ \ \textbf{Exit}\ \ p > ;\ < r' >\}$$

$$\textsf{Ts \& gr}\ \frac{r' > r \wedge u \neq \epsilon}{u' \leftarrow crnb(u),\ \ T' \leftarrow T \cup \{u'\},\ \ nu'(u') \leftarrow r',\ \ pd'(u) \leftarrow p,\ \ pd'(u') \leftarrow p'}\ \{< r\ \ l\ \ \textbf{Exit}\ \ p > ;\ < r'\ \ p' >\}$$

$$\textsf{Tree failed}\ \frac{}{u' \leftarrow pt(u)}\ \{< r\ \ l\ \ \textbf{Fail}\ \ p >\}$$

$$\textsf{Backtrack}\ \frac{r' = r}{u' \leftarrow nd(r),\ \ T' \leftarrow T - \{y | y > u'\}}\ \{< r\ \ l\ \ \textbf{Redo}\ \ p > ;\ < r' >\}$$

$$\textsf{Bkt \& go down}\ \frac{r' > r}{\frac{v \leftarrow nd(r),\ \ T' \leftarrow T - \{y | y > v\} \cup \{u'\},\ \ u' \leftarrow crc(v),}{nu' \leftarrow upn(nu, v) \cup \{(u', r')\},\ \ pd' \leftarrow upcp(pd, v) \cup \{(u', p')\}}}\ \{< r\ \ l\ \ \textbf{Redo}\ \ p > ;\ < r'\ \ p' >\}$$

FIG. 4 – Reconstruction de la trace virtuelle restreinte (modèle des boîtes simplifié) à partir de la trace actuelle

$$\textsf{Lfr \& gd}\ \frac{\frac{r' > r}{v \leftarrow crc(u),\ \ fst(u) \wedge lf(u) \wedge \neg ft(u) \wedge \neg ct}}{\frac{T' \leftarrow T \cup \{v\},\ \ u' \leftarrow v,\ \ n' = n+1,\ \ nu' \leftarrow nu \cup \{(v, n')\},}{pd' \leftarrow pd \cup \{(v, p')\},\ \ fst(u) \leftarrow false,\ \ fst' \leftarrow fst \cup \{(v, true)\},\ \ flr' \leftarrow false}}\ \{< r\ \ l\ \ \textbf{Call}\ \ p > ;\ < r'\ \ p' >\}$$

FIG. 5 – Exemple de règle de reconstruction complète avec conditions pour la règle Lf rcd & go down (noter que les conditions de la règle de transition utilisée sont toujours vérifiées)

du nœud courant u, et etiqueté avec la prédication donnée dans l'événement de trace suivant p'. On sait également que l'arbre courant T n'est pas complet et qu'il n'est pas en échec.

Sur l'exemple 1 de la section précédente, cette règle est utilisée pour passer des états S_1 à S_2 (figure 6).

La preuve complète de l'adéquation du schéma de reconstruction pour Q relativement à la SO comporte trois parties : lemmes établissant quelques propriétés générales de la SO (enchaînement des règles et des ports, voir figure 7) ; vérification de l'exclusivité des conditions associées à chaque règle du schéma de reconstruction ; enfin, pour chaque règle de R, vérification que le sous-état reconstruit est bien le même que l'état virtuel restreint à Q correspondant.

A titre d'exemple les étapes de preuve sont illustrées ci-dessous pour la règle Lf rcd & go down, dont la figure 8 montre l'état virtuel résultant. L'état virtuel restreint résultant est alors :

$S'/Q = \{T \cup \{u'\}, u' = crc(u), nu'(u') = n', pd'(u') = p\_cpini(u')\}$

L'événements de trace extrait, conformément au schéma de trace pour cette règle , est :

$\mathcal{E}_{\textsf{Lfrcd\&godown}}(S, S') =< nu(u)\ \ lp(u)\ \ Call\ \ pd(u) >$

Il peut être suivi d'un événement de trace e' qui contient les deux attributs suivants (on ne précise pas les ports possibles, mais le diagramme de la figure 7

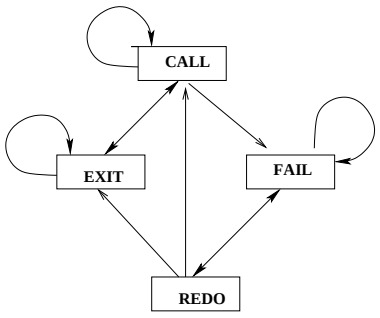

FIG. 7 – Algèbre des ports dans une trace de Byrd (modèle simplifié)

montre que seuls des événements avec des ports **Exit** ou **Call** sont possibles) : $nd(u') = n'$ et $pd'(u') = p\_cpini(u')$, soit :

$\mathcal{E}_s(S', S'') =< n'...p\_cpini(u') >$ avec $u' = crc(u)$.

On utilise alors la règle correspondante du schéma de reconstruction, instancié avec les événements de trace e et e'.

$$\textsf{Lr\&gd}\ \frac{Cond_{\textsf{Lfrcd\&godown}}(e, e')}{\frac{u' = crc(nd(nu(u))),\ \ T' = T \cup \{u'\},\ \ nu'(u') = n',}{pd'(u') = p\_cpini(u')}}\ \{e\ ;\ e'\}$$

On vérifie que la condition discriminant la règle Lf rcd & go down est bien vérifiée :
$Cond_{\textsf{Lfrcd\&godown}}(e, e') = (nu'(crc(u)) > nu(u))$, soit

$$\textsf{Lf rcd \& go down}\ \frac{\frac{2>1}{1{=}crc(\epsilon),\quad fst(\epsilon)\wedge lf(\epsilon)\wedge\neg ft(\epsilon)\wedge\neg ct}}{\begin{array}{c}T'{=}\{\epsilon,1\},\quad u'{=}1,\quad n'{=}2,\quad nu'{=}\{(\epsilon,1),(1,2)\},\\ pd'{=}\{(\epsilon,goal),(1,p(X))\},\quad fst'{=}\{(\epsilon,false),(1,true)\},\quad flr'{=}false\end{array}}\ \{<1\ 1\ \textbf{Call}\ goal>\ ;\ <2\ldots p(X)>\}$$

FIG. 6 – Lecture de la transition S_1 à S_2 avec les évènements de trace 1 et 2 (exemple 1)

$$\textsf{Lf rcd \& go down}\ \frac{fst(u)\wedge lf(u)\wedge\neg ct\wedge\neg ft(u)}{\begin{array}{c}T'{=}T\cup\{u'\},\quad u'{=}crc(u),\quad n'{=}n{+}1,\quad nu'(u'){=}n',\quad pd'(u'){=}p\_cpini(u'),\\ cl'(u){=}upcp(u),\quad cl'(u'){=}c\_cpini(u'),\quad fst'(u){=}false,\quad fst'(u'){=}true\},\quad flr'{=}false\end{array}}\ \{scs(u)\}$$

FIG. 8 – Règle de transition Lf rcd & go down et état S' obtenu

$n' > n$. En effet tout nouveau nœud créé l'est avec un numéro supérieur à tous ceux déjà existants.

Enfin l'état Q' reconstruit à l'aide de la règle du schéma de reconstruction est bien identique à l'état virtuel restreint à Q.

$Q' = \{T \cup \{u'\}, u' = crc(u), nu'(u') = n', pd'(u') = p\_cpini(u')\} = S'/Q$

Il résulte de l'adéquation du schéma de reconstruction que la lecture de la trace peut se faire en utilisant une partie quelconque de l'état courant virtuel (une fois identifiée la règle qui s'applique), ce qui en simplifie considérablement la compréhension.

Ainsi on peut "voir" sur les règles de la figure 4 comment l'arbre de preuve partiel évolue, ou comment se fait le parcours dans les boîtes. Par exemple, il est assez clair qu'une succession de **Exit** jusqu'à la racine de l'arbre (boîte $r = 1$) va permettre d'obtenir un arbre de preuve complet dont toutes les prédications associées aux nœuds auront été mises à jour conformément à la sémantique attendue de la résolution (non décrite ici) ; c'est à dire que l'on aura obtenu une preuve du but associé à la racine en utilisant toutes les instances de clauses correspondant aux prédications associées à chaque nœud et leurs descendants[6] et dont les prédications correspondantes sont associées aux événements de trace de port *exit*.

Ainsi l'examen exclusif de tous les événements de trace de port *exit* permet de reconstituer les arbres de preuve obtenus partiels ou complets.

7 Conclusion sur le modèle des boîtes

Nos premières observations porteront sur la compréhension de la trace que donnent les règles de la figure 4. Celles-ci peuvent se comprendre en effet sans avoir recours à la SO complète, mais en se limitant à un état restreint (dénoté Q). Tout ce qui est nécessaire y est formalisé, le recours à la SO n'étant utile que pour aller plus avant dans la compréhension. Les règles en donnent le squelette dynamique (parcours-construction-reconstruction d'arbre) et leurs conditions optionnelles associées (toujours valides pour la reconstruction d'une trace actuelle produite avec la SO) donnent l'interprétation immédiate des attributs de la trace.

Cette approche met aussi immédiatement en évidence les difficultés d'interprétation d'un tel modèle. Il est naturel que l'interprétation de la trace nécessite d'appréhender l'ensemble de la trace depuis le début (pour avoir une idée de l'état de la preuve) mais nous retiendrons deux remarques. Premièrement la lecture d'un événement "en avant" est nécessaire, ce qui est en soi un facteur de difficulté. Ceci pourrait être évité si une information sur la clause utilisée figurait dans un attribut, les facteurs discriminant les règles portant sur la forme des clauses utilisées (fait ou présence de plus d'une prédication dans le corps). La représentation avec des boîtes avait essentiellement pour objectif de "contenir" les clauses potentiellement utiles. On ne les retrouve plus dans la trace, ce qui retire au modèle une partie de son intérêt en le limitant en fait à la seule description du parcours-construction-reconstruction d'un arbre de preuve.

En deuxième remarque on observera a contrario que la trace contient un attribut inutile. La profondeur (attribut `1`) ne contribue finalement pas à la compréhension de la trace et la surcharge inutilement. En fait, la profondeur pourrait contribuer à la compréhension de l'arbre de preuve partiel en la combinant avec un codage adéquate des nœuds. Ce choix est fait par exemple dans la trace de GNU-Prolog [8] où les nœuds sont codés, non par leur ordre de création, mais par leur rang dans l'arbre. La combinaison des deux attributs permet alors un repérage direct dans l'arbre T du nœud courant. Ce choix constitue bien une amélioration de la trace originale[7].

Les quelques articles cités dans l'introduction traduisent la recherche permanente d'amélioration de la

[6] Dans ce modèle, si des clauses différentes peuvent avoir des instances identiques, on ne saura pas nécessairement quelle clause a été effectivement utilisée.

[7] Beaucoup de travaux introduisent des visualisations de la trace avec indentation et utilisent l'attribut *lpath* pour ce faire. Cela montre que cet attibut a une utilité pratique, mais il n'est pas utile à la reconstruction.

compréhension du contrôle et aussi de l'unification. Ainsi [1] (1984) et [14] (1985) proposent des améliorations de la trace de Byrd avec un nombre d'événements plus réduit apportant ainsi une vision plus synthétique de l'arbre parcouru, et ils proposent également de nouveaux ports concernant l'unification et le choix des clauses. [17] (1993) introduit explicitement une algèbre de boîtes avec graphiques à l'appui, mais ce modèle qui veut saisir tous les aspects de la résolution reste assez complexe. [10] (2000) propose une sémantique de trace fondée sur une sémantique dénotationnelle de Prolog. L'inconvénient principal est que la compréhension de la trace passe par une bonne compréhension d'un modèle complet de Prolog, synthétique mais nécessitant une certaine familiarité avec les continuations. L'article [11] (2003) relève d'une démarche analogue, mais celle-ci s'appuie directement sur les célèbres ports dont les enchaînements possibles constituent son squelette. Le résultat est également que la compréhension de la trace passe par l'assimilation d'une sémantique relativement complexe de Prolog qui s'apparente plus à une sémantique basée sur les "magic sets" qu'à une explication directe de la trace.

Ces études montrent que l'on a beaucoup cherché à améliorer les moyens de comprendre la résolution. Au fil du temps les travaux se sont concentrés sur des méthodes d'analyse et de visualisation de plus en plus complexes (par exemple [9] pour l'analyse des traces Prolog) pour des formes de résolution elles aussi de plus en plus complexes comme la résolution de CSP [15]. Il n'en reste pas moins cependant que la trace de Byrd reste la base des traceurs pour les systèmes de résolution et ses fameux ports inspirent encore, de temps en temps, les chercheurs.

Dans cet exemple on a traité une instance particulière du modèle des boîtes. Il serait intéressant, et ce sera notre prochaine étape, d'obtenir un modèle plus générique susceptible d'engendrer potentiellement diverses implantations connues de ce modèle. Cela semble possible avec l'approche présentée ici (voir [6]).

8 Conclusion générale

Le point essentiel de ce papier est l'illustration d'une approche originale pour donner une sémantique à des traces d'exécution. L'exemple utilisé ici a essentiellement un caractère anecdotique, même si, in fine, le résultat est sans doute une formalisation complète parmi les plus simples (car restreinte aux seuls éléments nécessaires à sa compréhension) que l'on ait pu formuler juqu'à présent d'un modèle des boîtes de Byrd.

La notion de trace virtuelle a pour but de capturer l'idée du "bon" niveau d'observation d'un processus physique. Même pour un programme, le bon niveau d'observation n'est pas évident. Quels sont les éléments significatifs ou utiles à observer ? Toute exécution d'un programme met en œuvre une série de couches de logiciels jusque dans les composants matériels. Certaines erreurs peuvent même provenir d'interférences de particules énergétiques avec des composants électroniques. Le "bon" niveau d'observation ne peut donc être défini de manière absolue. Toute trace virtuelle ne peut être dite intégrale que si l'on se fixe une limite a priori quant à la granularité du phénomène observé (mais penser que l'on puisse atteindre un niveau "ultime" de description relèverait d'une approche excessivement réductionniste). Dans le cas d'un langage de programmation le niveau d'observation sera usuellement défini par le langage lui-même, ne serait-ce que pour des raisons évidentes de capacité de compréhension du processus (celui que l'auteur du programme est seul à même d'appréhender).

Le point important ici est que le niveau d'observabilité est en fait arbitraire et qu'en aucun cas le niveau choisi ne peut être considéré comme ultime. Il est donc normal que, pour un niveau d'observation donné, on soit obligé de tenir compte dans la description, aussi précise soit-elle, d'éléments externes à celle-ci. C'est pourquoi la SO constitue un modèle à la fois indépendant d'un processus particulier observé (c'est en ce sens qu'elle est "générique"), mais également comportant des références à des aspects non formellement décrits associables aux processus que l'on souhaite observer.

Références

[1] Patrice Boizumault. Deux Modèles de Trace pour le Langage Prolog. In M. Dincbas and S. Bourgault, editors, *Actes du Quatrième Séminaire de Programmation en Logique*, CNET-Lannion (France), May 1984.

[2] L. Byrd. Prolog debugging facilities. Technical Report D.A.I. paper No 19, University of Edinburgh, July 1980.

[3] L. Byrd. Understanding the control flow of Prolog programs. In S.-A. Tarnlund, editor, *Logic Programming Workshop*, Debrecen, Hungary, 1980.

[4] K.L. Clark. Predicate Logic as a Computational Formalism. Technical Report 79/59, Imperial College, London, December 1979.

[5] P. Deransart. On using Tracer Driver for External Dynamic Process Observation. In Vanhoof W. and S. Muñoz-Hernandez, editors, *Proceedings of the 16th Workshop on Logic-based Methods in Programming Envi-*

ronments (WLPE'06), a pre-conference workshop of ICLP'06, Seattle, USA, August 2006. `http ://arxiv.org/abs/cs/0701148`.

[6] P. Deransart, M. Ducassé, and G. Ferrand. Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques. Technical report, INRIA, mai 2007. http ://hal.inria.fr.

[7] P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog, The Standard ; Reference Manual*. Springer Verlag, April 1996.

[8] D. Diaz. GNU-Prolog, a free Prolog compiler with constraint solving over finite domains, 2003. `http ://gprolog.sourceforge.net/`, Distributed under the GNU license.

[9] M. Ducassé. Opium : an extendable trace analyzer for Prolog. *The Journal of Logic Programming,* special issue on Synthesis, Transformation and Analysis of Logic Programs, 39 :177–223, 1999.

[10] E. Jahier, M. Ducassé, and O. Ridoux. Specifying Prolog trace models with a continuation semantics. In K.-K. Lau, editor, *Proc. of LOgic-based Program Synthesis and TRansformation*, London, July 2000. Springer-Verlag, LNCS 2042.

[11] Marija Kulàs. Pure Prolog Execution in 21 Rules. In Arnaud Lallouet, editor, *Proc. of the 5th Workshop on Rule-Based Constraint Reasoning and Programming (RCoRP'03)*, Kinsale, September 2003. Repository arXiv :cs :PL/0310020 v1.

[12] Ludovic Langevine, Pierre Deransart, and Mireille Ducassé. A generic trace schema for the portability of cp(fd) debugging tools. In K.R. Apt, F. Fages, F. Rossi, P. Szeredi, and Jozsef Vancza, editors, *Recent Advances in Constraints, 2003*, number 3010 in LNAI. Springer Verlag, May 2004.

[13] Luis Moniz-Pereira, Fernando Pereira, and D.H.D. Warren. User's Guide to DECsystem-10 Prolog, 1978. University of Edinburgh.

[14] Masayuki Numao and Tetsunosuka Fujisaki. Visual Debuggger for Prolog. In *Proceedings of the Second Conference on Artificial Intelligence Applications*, Miami, December 1985.

[15] OADymPPaC. Tools for dynamic analysis and debugging of constraint programs, 2004. French RNTL project (2001-2004) `http ://contraintes.inria.fr/OADymPPaC`.

[16] P. Roussel. Prolog : Manuel de Référence et d'Utilisation, 1975. Université d'Aix-Marseille II.

[17] G. Toberman and C. Berckstein. What's in a Trace : The Box Model revisited. In P. Fritzson, editor, *Proceedings of the First Workshop on Automated and Algorithmic Debugging (AADEGUG'93)*, number 749 in LNCS, Linkoeping, Sweden, May 1993.