Une sémantique observationnelle du modèle des boîtes pour la résolution de programmes logiques (version étendue)

Pierre Deransart, Mireille Ducassé, Gérard Ferrand

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

Pierre Deransart[1], Mireille Ducassé[3], and Gérard Ferrand[2]

[1] Inria Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France
Pierre.Deransart@inria.fr
[2] IRISA/INSA, Campus universitaire de Beaulieu, 35042 Rennes Cedex, France
Mireille.Ducasse@irisa.fr
[3] LIFO, BP 6759, 45067 Orléans Cedex 2, France
Gerard.Ferrand@lifo.univ-orleans.fr

Résumé

Dans ce rapport 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 de toute 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.

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

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.

Abstract

This report 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.

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

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.

1 Introduction

Ce rapport présente un modèle de trace Prolog (souvent appelé "modèle des boîtes de Byrd") d'une manière originale, basée sur la notion de sémantique ob-

servationnelle (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 rapport 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'aucune 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, de schéma de trace et de schéma 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 liés au non déterminisme des solutions. Même si, par la suite, d'autres modèles ont été adoptés avec des stratégies bien plus complexes[4], 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 retrouvent dans toutes les traces des systèmes Prolog 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 ce rapport 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

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

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) ; la preuve d'adéquation est en annexe, 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 de l'approche observationnelle.

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[5].

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 donnés 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 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 un domaine d'états et une fonction de transition formalisant le passage d'un état à un autre. Cette sémantique sera appelée *sémantique observationnelle* (SO) 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 ensemble d'actions) du processus observé réalisant les transitions sont supposés en nombre fini. Ce sont les *ports* de la trace. A chaque port peuvent correspondre plusieurs transitions. On considérera ici que la SO est spécifiée par un ensemble fini de règles de transition nommées, noté R.

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

[5] 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.

Definition 1 (Trace intégrale virtuelle).

Une trace intégrale virtuelle est une suite d'événements 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'événement.*
- t *: est le* **chrono***, temps de la trace. C'est un entier incrémenté d'une unité à chaque événement.*
- $S_{t+1} = p_{1,t+1}, ..., p_{n,t+1}$ *: S_{t+1} est l'état courant suivant l'événement de trace au moment t et les $p_{i,t+1}$ sont des valeurs des* **paramètres** *p_i de l'état obtenu, une fois l'action réalisée.*
- 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. Une portion finie non vide de trace sera dénotée $< S_0, e_t^+ >$. Une trace a au moins un événement vide.*

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'événements de trace de la forme $e_t : (t, a_t, A_t),\ t \geq 0$*, dérivés de la transition $< S_t, S_{t+1} >$ par la fonction $\mathcal{E}$, dite* fonction d'extraction*, telle que $e_t = \mathcal{E}(S_t, S_{t+1})$.*

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

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 $\forall < r, S, S' > \in SO, \mathcal{E}_r(S, S') = e_r$, où e_r dénote les calculs des valeurs de chaque attribut.

La description de la famille de fonctions $\mathcal{E}_r$, avec les calculs d'attributs, constitue un schéma de trace.

La trace actuelle est la trace émise par le traceur du processus observé.

La trace intégrale virtuelle est un cas particulier de trace actuelle où les attributs décrivent complètement les états obtenus par la suite des transitions.

La question se pose maintenant de l'utilité d'une trace, c'est à dire la possibilité de reconstruire une suite d'états, éventuellement partielle, 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.

L'idée d'adéquation est donc d'assurer que la suite des états partiels qui est obtenue à partir de cette trace est bien la même que (en fait incluse dans) celle qui a servi à produire la trace. La notion d'adéquation est donc relative à des états limités à un sous-ensemble des paramètres. On note S/Q la restriction d'un état quelconque S aux paramètres Q. Q sera appelé *état actuel courant* et S/Q l'état virtuel restreint aux paramètres de Q, ou, s'il n'y a pas d'ambiguité, *état virtuel courant restreint*.

On supposera donc que la SO est décrite par un ensemble fini de règles qui constituent un "modèle de trace", tel que chaque règle donne lieu à la production d'un événement de trace. La fonction d'extraction est donc constituée d'autant de composants qu'il y a de règles et dénotés $\mathcal{E}_r$ pour chaque règle r. De même on utilisera une *fonction de reconstruction* $\mathcal{C}_r$ décrite par autant de composants qu'il y a de règles et dénotés $\mathcal{C}_r$ pour chaque règle r. La description de $\mathcal{C}$ constitue un *schéma de reconstruction*.

Definition 3 (Trace adéquate).

Etant donnés un état actuel Q restriction de S à un sous ensemble de ses paramètres, une SO définie sur S par un ensemble fini de transitions R et une trace actuelle $T_w =< Q_0, w_t^* >$ telle que $Q_0 = S_0/Q$
T_w 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*

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

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.

L'adéquation pour un sous-état Q donne à une trace une sémantique propre : la lecture de la trace peut être comprise comme l'évolution d'un état restreint. De plus, cette suite d'états est exactement la suite des états restreints correspondants observables sur le processus observé. L'adéquation assure que toute l'information possible est bien dans la trace actuelle modulo le fait que seule une partie de ce qui est observable est communiqué dans la trace.

Si $Q = S$ et T_w est une trace adéquate, alors T_w est une trace intégrale. Une telle trace actuelle, adéquate et intégrale est la garantie que l'observateur est capable de reconstituer toute l'évolution observable du processus et donc de reconstituer l'intégralité des objets observés et leur évolution à partir de la trace observée. Si l'on considère que $\mathcal{E}$ est une forme de fonction de compression, $\mathcal{C}$ peut être vue, dans ce cas, comme une fonction de décompression sans perte de données[6].

[6] On n'utilise pas ici les termes "compression/décompression" afin de se démarquer du cas où le flot de trace (par exemple codé en XML) peut lui-même faire l'objet d'une "compression/décompression" numérique. Ces termes sont donc réservés pour ce cas.

En réalité, et c'est le but de la trace virtuelle, un observateur ne sera intéressé qu'à une partie de la trace virtuelle, c'est à dire qu'à un sous-ensemble Q de ses paramètres. Par contre il est 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'événements de trace. En effet, un événement de trace actuelle, produit par une transition $< r, S, S' >$ 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 éventuellement 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 événement 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 $r \in R$:

$\forall\, e,\, e',\, r',\, S,\, S',\, S'',$
$\mathcal{E}_r(S, S') = e\ \wedge\ \mathcal{E}_{r'}(S', S'') = e'$
(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 toute trace actuelle $T_w =< Q_0, w_t^+ >$ *, définie par le schéma de trace* $\mathcal{E}$ *et telle que* $Q_0 = S_0/Q$*, est* adéquate *pour Q par rapport à la trace intégrale virtuelle* $T_v =< S_0, v_t^+ >$.

Noter que la condition 1 est trivialement satisfaite s'il y a autant de ports de trace distincts que de transitions possibles dans la SO (isomorphisme des règles et des ports). Dans ce cas, la condition d'adéquation se réduit à la condition 2.

Il y a deux manières d'interpréter cette proposition. Elle montre que l'adéquation est d'abord une propriété de correction de trace. Dans ce sens elle signifie que si l'on suit pas à pas l'extraction d'une trace à partir de l'état initial S_0 durant $t + 1$ étapes, l'état obtenu reconstruit à partir de la portion de trace $T_w =< S_0/Q, w_t^* >$ est le même que celui obtenu par application des règles de la SO, restreint à Q, soit S_{t+1}/Q. Ceci correspond à la deuxième condition.

Mais il y a une deuxième lecture liée à la question de compréhension du processus observé à travers la trace et qui correspond aux deux conditions prises ensembles. Connaissant l'état initial et la suite des événements de trace, la proposition assure que l'on peut en déduire une suite d'états dérivés par transitions dans la SO, suite identique dans le sens où elle engendre la même trace.

Ceci met en évidence la distinction qu'il y a lieu de faire entre "correction" de la trace (condition 2) et la capacité de compréhension du processus observé à travers la trace. La seconde lecture est aussi liée à la capacité d'associer une transition ou règle à un pas de trace (condition 1) et ainsi de retrouver des informations sur les états virtuels au delà de leur restriction. L'adéquation assure que la trace extraite représente bien l'évolution possible d'un sous-état virtuel, mais aussi, que si on connaît la SO, on peut également appréhender le fonctionnement du processus observé.

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 à une sémantique concrète particulière.

La Sémantique Observationnelle (SO) rend compte de toutes les traces virtuelle 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 telles 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 événement de trace. Une règle a quatre composants.

- Un identificateur de règle (nom).
- Un numérateur comportant des conditions sur l'état antérieur et des calculs préliminaires.
- Un dénominateur comportant la description de l'état obtenu (ce qui reste invariant peut être omis), par les calculs des nouvelles valeurs des paramètres.
- 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 figurant 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[7] $< nom, S, S' >$ où par abus de notation on dénotera S les conditions

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

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. On y ajoutera également d'éventuels facteurs externes[8] décrits par des *conditions externes*. Une règle sera donc présentée de la manière suivante.

$$\mathsf{Nom}\ \frac{Conditions\ caracterisant\ l'etat\ courant}{Calcul\ des\ nouveaux\ parametres}\ \{Conditions\ externes\}$$

Pour décrire la SO, on utilisera deux types de fonctions : celles relatives aux objets décrits et leur évolution dans la trace virtuelle et celles relative à des événements 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'événement de trace actuelle correspondant, c'est à dire le port et les attributs. Il y a un seul événement de trace par règle . L'ensemble des rgles qui décrivent la fonction d'extraction constitue un *schéma de trace*. Chaque règle du schéma de trace a la forme suivante.

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

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.

$$\mathsf{Nom}\ \frac{Conditions\ d'identification}{Calculs\ de\ reconstruction}\ \{Evenements\ de\ trace\}$$

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.

[8] Ces facteurs sont dits "externes" du point de vue de la SO. Ils ne le sont pas du point de vue du processus observé. Il ne s'agit donc pas d'interaction.

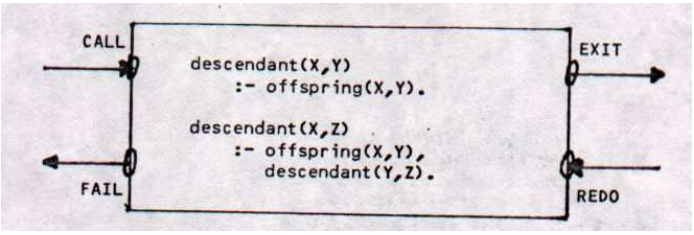


Fig. 1. Modèle de boîte tel que dessiné par Byrd [3]

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 déjà très répandue à cette époque 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 connus, 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 événement 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 strictement le chemin 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. L'approche formalisée ici sera qualifiée 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 [6]), 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 qui se déploie à la manière d'un "treemap", réalisant ainsi un jeu de boîtes encastrées.

Dans la mesure du possible nous utilisons le vocabulaire ISO-Prolog [6].

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 le mot 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. $nu(v)$ est le numéro (entier positif) associé au nœud v dans T.
5. $pred : T \rightarrow \mathcal{H}$. Abbrev. : pd. $pd(v)$ 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. $cl(v)$ 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. $fst(v)$ 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 est redevenu 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 S_0 :

Pour des raisons d'espace typographique, on utilisera parfois T (F) pour $true$ (resp. $false$).

$\{\{\epsilon\}, \epsilon, 1, \{(\epsilon, 1)\}, \{(\epsilon, goal)\}, \{(\epsilon, list\_of\_goal\_claus)\}, \{(\epsilon, T)\}, F, F\}$

Le modèle est basé sur un parcours de construction d'arbres de preuve partiels, construits puis reconstruits après des retours arrières. Les nœuds ne sont construits que juste avant d'être visités pour la première fois. 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 des objets décrits) :

- $parent : T \rightarrow T$. Abbrev. : pt. $pt(v)$ est l'ancêtre direct de v dans T. Pour simplifier le modèle, on suppose que $pt(\epsilon) = \epsilon$.
- $leaf : T \rightarrow Bool$. Abbrev. : lf. $lf(v)$ est vraie ssi v est une feuille dans T.
- $may\_have\_new\_brother : T \rightarrow Bool$. Abbrev. : $mhnb$. $mhnb(v)$ 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. $crc(v)$ est le nouvel enfant de v dans T.
- $create\_new\_brother : T \rightarrow T$. Abbrev. : $crnb$. $crnb(v)$ est le nouveau frère de v dans T. Defini si v différent de ϵ.
- $has\_a\_choice\_point : T \rightarrow Bool$. Abbrev. : hcp. $hcp(v)$ 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. $w = gcp(v)$ 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 . $ft(v)$ est vrai ssi la première clause dans $claus(v)$ est un fait.
- $update\_number : F, T \rightarrow F$. Abbrev. : upn. $upn(nu, v)$ 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. : $upcp$. (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 utilisée.

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. $scs(v)$ 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. $flr(v)$ 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. : $cpini$. $(c,p) = cpini(v)$ 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 . $pud(v)$ est la nouvelle valeur de la prédication attachée au nœud v de T, suite à une unification réussie.

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

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

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

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

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

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

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

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

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

La SO est décrite par les règles de la figure 2. Chaque règle est commentée dans ce qui suit.

- 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 appelante 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.

A l'état initial S_0, seule une des règles Leaf reached ou Lf rcd & go down s'applique. 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. Pour les règles Leaf reached et Lf rcd & go down le port associé est **Call**, pour les règles Tree success et Tree success, **Exit**, pour Tree failed, **Fail**, et pour Backtrack Bkt & go down, le port est **Redo**.

5 Extraction de la trace actuelle

Chaque application d'une règle de la SO donne lieu à l'extraction d'un événement 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 . 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 $+1$. $lp(\epsilon) = 1$.

Comme la trace de Byrd (voir Annexe A), dans sa forme originale[9], se contente de décrire le parcours construction d'arbre dont les nœuds sont étiquetés avec des prédications et de donner, en cas de succès, le squelette final complet décoré avec les étiquettes finales correctes, on obtient ainsi au final les instances de clauses utilisées, sans nécessairement savoir quelle clause a effectivement été utilisée à un nœud donné.

Pour rendre compte des éléments propres à la trace de Byrd seulement (évolution de l'arbre ou des boîtes et étiquettes) 4 paramètres sont suffisants. On prendra donc comme état virtuel restreint les paramètres suivants :

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

Noter que l'on aurait pu ajouter les paramètres ct et flr. Mais cela ne parait pas nécessaire a priori car ct est vrai (sauf au premier événement de trace) ssi le premier ou le deuxième attribut est 1 (en fait ils le sont enesembles) ; et flr devient faux (échec) pour tout événement de trace de port **Fail**. En particulier si on est à la racine, on sait alors si on est en échec (événement de port **Fail** à la racine) ou en succès (événement de port **Exit** à la racine). On sait alors si on a à faire à un arbre en échec ou un arbre de preuve complet (succès).

L'état initial S_0/Q est donc : (voir l'état complet à la section précédente)

$\{\{\epsilon\}, \epsilon, \{(\epsilon, 1)\}, \{(\epsilon, goal)\}\}$

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'événement 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'événement 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.` (u nœud courant)

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

[9] Une description détaillée de la trace de Byrd originale est donnée dans les annexes A et D. La trace actuelle originale correspondante ne dit rien sur l'évolution des clauses elles-mêmes dans les boîtes. Pour cette raison les informations sur les clauses sont omises dans l'état actuel courant.

```
c3: p(b).
c4: eq(X,X).

:- goal.
```

chrono	nu(u)	lp(u)	port	pd(u)	Etat virtuel atteint
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
7	2	2	Exit	p(b)	S8
8	4	2	Call	eq(b,b)	S9
9	4	2	Exit	eq(b,b)	S10
10	1	1	Exit	goal	S11

L'exemple est détaillé dans l'annexe B.

Le schéma de trace est décrit à la figure 3.

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

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

$$\textsf{Tree success}\ \frac{}{< nu(u)\ \ lp(u)\ \ \mathbf{Exit}\ \ p >}\ \{p = pud(u)\}$$

$$\textsf{Tree suc \& go right}\ \frac{}{< nu(u)\ \ lp(u)\ \ \mathbf{Exit}\ \ p >}\ \{p = pud(u)\}$$

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

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

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

Fig. 3. Schéma de trace (fonction d'extraction de la Trace)

Afin de faciliter la lecture, toutes les informations non nécessaires à l'extraction sont omises. En fait, un événement 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 $\mathcal{E}_{\textsf{Tree failed}}$ est :

$$\textsf{Tree failed}\ \frac{\neg fst(u) \wedge \neg ct \wedge \neg hcp(u),\quad v \leftarrow pt(u)}{\frac{u' \leftarrow v,\quad (u=\epsilon) \Rightarrow (ct' \leftarrow true),\quad flr' \leftarrow true}{< nu(u)\ \ lp(u)\ \ \mathbf{Fail}\ \ pd(u) >}}\ \{flr(u)\ \vee\ flr\}$$

On peut y observer clairement la remontée "directe" dans l'arbre, suite à un échec (extraction d'événement 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 de reconstruction $\mathcal{C}$ de la trace virtuelle restreinte, à partir d'un état actuel initial 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. 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 $Q = S/Q = \{T, u, num, pred\}$

Cette trace exige de lire deux événements 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.

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

Cette règle indique que si après un événement de port **Call**, les numéros de boîtes croissent avec l'événement 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 du nœud courant u, et étiqueté avec la prédication donnée dans l'événement de trace suivant p'. On sait également que l'arbre courant T n'est

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

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

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

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

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

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

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

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

pas complet et qu'il n'est pas en échec. Noter également que les conditions de la règle utilisée (sous-dénominateur du numérateur) sont toujours vérifiées.

Sur l'exemple 1 de la section précédente, cette règle est utilisée pour passer des états S_1 à S_2 (voir annexe B pour les détails). Elle donne une lecture de la transition S_1 à S_2 avec les événements de trace de chrono 1 et 2.

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

La preuve complète de l'adéquation du schéma de reconstruction pour Q relativement à la SO est donnée dans l'annexe C. Elle comporte trois parties : lemmes établissant quelques propriétés générales de la SO (enchaînement des règles et des ports, voir figure 5) ; 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 6 montre l'état virtuel résultant. L'état virtuel restreint résultant est alors :

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

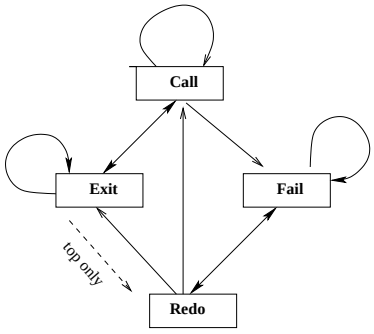


Fig. 5. Algèbre des ports dans une trace de Byrd (modèle simplifié)

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

Fig. 6. Règle de transition Lf rcd & go down et état S' obtenu

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

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

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 8 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, instanciée avec les événements de trace e et e'.

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

On vérifie que la condition discriminant la règle Lf rcd & go down est bien vérifiée :
$Cond_{\mathsf{Lfrcd\&godown}}(e, e') = (nu'(crc(u)) > nu(u))$, soit $n' > n$. 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 (évolution de l'état restreint Q) 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[10] et dont les prédications correspondantes sont associées aux événement 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 re-construction 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. Nous en retiendrons deux. En premier lieu on observera que s'il est normal 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 résolution), il l'est moins que la lecture d'un événement "en avant" soit 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[11] (par exemple la clause choisie avant un événement de port **Call**). 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 grande partie de son intérêt, en le limitant de fait à la seule description d'un parcours d'arbre.

En deuxième remarque on observera a contrario que la trace contient un attribut inutile. La profondeur (attribut `l`) ne contribue finalement pas à la compréhension de la trace et la surcharge inutilement. En fait la profondeur

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

[11] Dans les conditions en effet, les facteurs discriminant les règles utilisées portent sur la nature des clauses.

pourrait contribuer à la compréhension de l'arbre de preuve partiel en la combinant avec un codage adéquat 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[12].

Les quelques articles cités dans l'introduction traduisent la recherche permanente d'améliorations de la compréhension du contrôle et aussi de l'unification. Ainsi [1] (1984) [14] (1985) proposent des améliorations de la trace de Byrd avec un nombre d'événement 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 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 est possible avec l'approche présentée ici. Une première description de différents modèles est faite dans l'annexe D, avec une SO proposée à l'annexe E.

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

8 Conclusion générale

Le point essentiel de ce rapport 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 (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 Environments (WLPE'06), a pre-conference workshop of ICLP'06, Seattle, USA, August 2006. http ://arxiv.org/abs/cs/0701148.

6. P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog, The Standard ; Reference Manual*. Springer Verlag, April 1996.
7. P. Deransart and J. Małuszyński. *A Grammatical View of Logic Programming*. The MIT Press, 1993.
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.

ANNEXE A : Le modèle de Byrd

On décrit ici le modèle de Byrd avec ses représentations possibles en utilisant des boîtes ou des arbres. La figure 7 montre la manière dont les boîtes se combinent donnant une sorte d'algèbre des ports. On indique également la correspondance graphique avec la représentation sous forme d'arbre de la combinaison des boîtes.

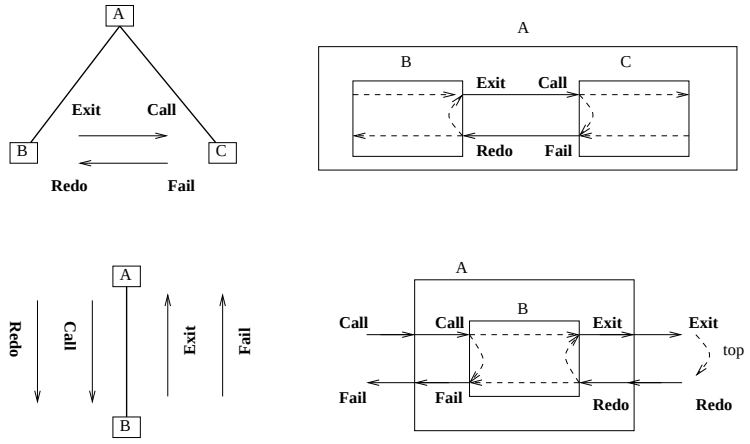


Fig. 7. Correspondance des représentations arbres /boîtes : combinaisons possibles des ports (boîtes adjacentes ou enchâssées)

Il y a deux points de vue possibles : avec boîtes ou avec arbres. Les boîtes en effet sont encastrables les unes dans les autres et peuvent donc être représentées à la manière d'un treemap[13]. Les deux représentations sont isomorphes. La trace de Byrd consiste à tracer systématiquement les passages d'un port à l'autre des boîtes , tout en respectant un ordre de visite fixé a priori et en refaisant dans l'ordre inverse tout le parcours déjà effectué afin de trouver d'autres solutions possibles. Cet ordre est représenté par des flèches dont la succession est limitée à certaines combinaisons ainsi qu'il est indiqué sur la figure.

[13] Un treemap est une manière de représenter un arbre, au départ représenté en deux dimensions verticales (manière adoptée ici), dans un plan horizontal. La racine est un grand rectangle. Celui-ci est subdivisé en autant de parties qu'il y a d'enfants. Chaque enfant est à son tour subdivisé ... etc. Afin de pouvoir mettre en évidence les communications (ports) entre descendants de même niveau, la partition d'une "boîte " est en fait une juxtaposition de sous-boîtes mises dans le même ordre que les nœuds correspondants de l'arbre. Il en est de même pour boîtes encastrées dont aucun contour n'est jointif afin de mettre en évidence les sous-boîtes et les ports liant les descendants.

Comme la représentation par arbres ou par boîtes est isomorphe, les parcours possibles sont également isomorphes. Du point de vue des boîtes cela signifie que l'on ne peut sortir d'une boîte qu'en y étant rentré d'abord et réciproquement et toute traversée gauche droite complète d'une boîte doit (trajet aller) être suivie d'un trajet retour. Seuls les parcours externes sont tracés. L'effet recherché par Byrd était une vision globale immédiate du non déterminisme des solution en utilisant une représentation systématique structurée avec une évolution déterministe.

Du point de vue des arbres, chaque boîte est représentée par un nœud ; les parcours entre boîtes de même niveau correspondent aux parcours entre nœuds voisins de même niveau dans l'arbre ; et les parcours entre descendants correspondent à des parcours entre boîtes encastrées. Le parcours contraint des boîtes correspond dans l'arbre à des parcours de visite descendant gauche-droite mais avec la particularité qu'on ne peut remonter dans l'arbre, suite à un échec, que par la première branche visitée (i.e. la branche la plus à gauche).

Il résulte de cette représentation que les ports ne peuvent se suivre que selon certaines combinaisons. La figure 8 montre les combinaisons de ports possibles qui résultent de cette algèbre.

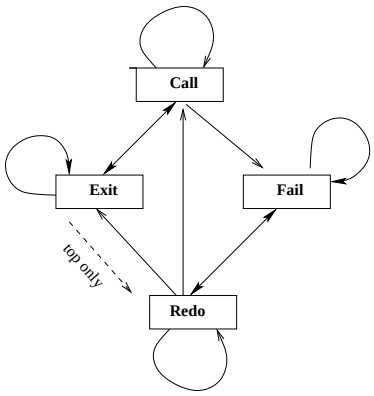


Fig. 8. Algèbre des ports dans une trace de Byrd

La trace de Byrd originale peut être produite en instrumentant le programme avec le méta-interprète suivant proposé par Byrd [3]. Il est intéressant d'observer que malgré la simplicité du méta-programme qui instrumente tout programme afin de le tracer, sa compréhension[14] nécessite une excellente connaissance de

[14] Le méta-interprète utilise les prédicats système suivants : *call*, exécution d'une prédication, *fail*, échec provoqué (suivi d'actions de retour arrière), *;* , dénotant la disjonction. Le prédicat *display* est ici une évocation d'écriture de trace sur une sortie standard. Seuls les ports sont tracés ici. La production de la trace actuelle complète de la section 5 est simple à programmer, mais détruit la clarté de cette

la sémantique opérationnelle des interprètes Prolog. La sémantique d'une telle implantation du traceur est donc loin d'être évidente.

```
:- trace(goal).

goal:- trace(p(X)), trace(eq(X,b)).
p(a).
p(b).
eq(X,X).

trace(Pred) :-  display('Call',Pred),
               (call(Pred) ; (display('Fail',Pred), fail)),
               (display('Exit',Pred) ; (display('Redo',Pred), fail)).
```

présentation du fait de la nécessité d'instrumenter, dans le méta-programme, tous les programmes avec un argument supplémentaire correspondant à la profondeur.

ANNEXE B : Exemple simple

On illustre ici la SO de la section 3, l'extraction et la reconstruction de la trace avec un exemple simple de programme. La trace obtenue est la même pour les différents modèles de traces considérés ici (voir Annexes suivantes D et E).

Exemple 1

```
Programme:

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 virtuel atteint
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
7	2	2	Exit	p(b)	S8
8	4	2	Call	eq(b,b)	S9
9	4	2	Exit	eq(b,b)	S10
10	1	1	Exit	goal	S11

La trace est celle définie dans la section 5, figure 3.

Elle a la forme suivante (le "nœud concerné" correspond au nœud courant vu du point de vue de la trace).

```
t    r    l   port  p
```

où

- `t` est le chrono.
- `r` est le numéro de création du nœud u concerné par l'événement de trace, soit $nu(u)$ (v au lieu de u pour les règle de port **Redo**).
- `l` est la profondeur dans l'arbre T du nœud concerné, soit $lp(u)$ ($lp(\epsilon) = 1$).
- `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 $pd(u)$ [ou $pud(u)$ (événements de port **Exit**)].

L'identificateur d'événement de trace est omis ici, le chrono en tenant lieu.

L'état courant a la forme : $\{T, u, n, nu, pd, cl, fst, ct, flr\}$.

Les paramètres sont :

1. $T : T$ est l'arbre courant.
2. $u \in T : u$ est le nœud courant dans T (boîte visitée).
3. $n \in \mathcal{N} : n$ est le numéro de création du dernier nœud créé.
4. $nu : T \rightarrow \mathcal{N} : nu(u)$ est le numéro de création de u dans T.
5. $pd : T \rightarrow \mathcal{H} : pd(u)$ est la prédication associée au nœud u.
6. $cl : T \rightarrow 2^P : cl(u)$ est une liste de clauses du programme tracé définissant le prédicat de $pred(u)$.
7. $fst : T \rightarrow Bool : fst(u)$ est vrai ssi u n'a pas encore été visité.
8. $ct \in Bool : ct$ vrai ssi le sous-arbre de racine u a été complètement visité.
9. $flr \in Bool : flr$ est vrai si le sous-arbre de racine u est en échec.

On ommet S_0 et la transition S_0 vers S_1 (action "top level") et l'état "initial" considéré est donc létat S_1.

La trace actuelle est adéquate pour l'état $Q = \{T, u, nu, pd\}$. On a besoin de la fonction nu, son inverse nd étant utilisé pour la reconstruction.

L'exemple est donné intégralement : pour chaque transition décrite (de $t = 1$ à 10) à partir de l'état S_t, on donne : la règle de transition, l'état S_{t+1} obtenu, l'événement de trace extrait et l'état Q_{t+1} reconstruit, dont l'identité avec la partie Q de S_{t+1} peut être vérifiée.

Etat "initial" :
$S_1 : \{\{\epsilon\}, \epsilon, 1, \{(\epsilon, 1)\}, \{(\epsilon, goal)\}, \{(\epsilon, [c1])\}, \{(\epsilon, true)\}, false, false\}$

Sa restriction à Q est :
$S_1/Q : \{\{\epsilon\}, \epsilon, \{(\epsilon, 1)\}, \{(\epsilon, goal)\}\}$

De S_1 à S_2

Seule la règle (1) Lf rcd & go down s'applique (première visite du nœud courant). On vient de "rentrer" dans la boîte `goal` et la première clause but (`c1`) n'est pas un fait. Un descendant est construit dont l'étiquette $p(X)$ n'apparaitra que dans l'événement de trace suivant.

(1) Lf rcd & go down $\dfrac{fst(\epsilon) \wedge leaf(\epsilon) \wedge \neg ct \wedge \neg fact(c1), \quad v \leftarrow 1}{\begin{array}{c} T'=\{\epsilon,1\}, \quad u'=1, \quad n'=2, \quad nu'=\{(\epsilon,1),(1,2)\}, \quad pd'=\{(\epsilon,goal)(1,p(X))\}, \\ cl'=\{(\epsilon,[])(1,[c2,c3])\}, \quad fst'=\{(\epsilon,false),(1,true)\}, \quad flr'=false \end{array}}$ {
$scs(\epsilon), \quad ([c2, c3], p(X)) = cpini(1)\}$

$S_2 : \{\{\epsilon, 1\}, 1, 2, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(X))\},$
$\{(\epsilon, []), (1, [c2, c3])\}, \{(\epsilon, false), (1, true)\}, false, false\}$

L'événement de trace extrait est (règle d'extraction Lf rcd & go down de la section 5) :
`< 1 1 1 Call goal >` (suivi de `< 2 2 2 Call p(X) >`).

L'état reconstruit est (règle de reconstruction Lf rcd & go down de la section 6) :
$Q_2 = S_2/Q : \{\{\epsilon, 1\}, 1, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(X))\}\}$

Pour la génération de Q_2, on utilise les informations $< r' \quad p' >$ du second événement de trace.

De S_2 à S_3

Seule la règle (2) Leaf reached s'applique (la première clause utilisée (`c2`) est un fait), on entre dans la boîte `p`.

$$(2)\quad \text{Leaf reached}\ \frac{first(1) \wedge leaf(1) \wedge \neg ct \wedge fact(c2)}{cl' = \{(\epsilon, []), (1, [c3])\},\quad fst' = \{(\epsilon, false), (1, false)\}}\ \{\}$$

$$S_3 : \{\{\epsilon, 1\}, 1, 2, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(X))\},$$
$$\{(\epsilon, []), (1, [c3])\}, \{(\epsilon, false), (1, false)\}, false, false\}$$

L'événement de trace extrait est (règle d'extraction Leaf reached de la section 5) :

`< 2  2  2  Call  p(X) >`

L'état reconstruit est (règle de reconstruction Leaf reached de la section 6) :
$Q_3 = S_3/Q : \{\{\epsilon, 1\}, 1, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(X))\}\}$

mais l'événement de trace laisse l'état courant actuel invariant : $Q_3 = Q_2$.

De S_3 à S_4

La règle (3) Tree suc & go right s'applique car la prédication appelée $p(X)$ est un succès et devient $p(a)$, et peut avoir un frère (existence d'une deuxième prédication dans le corps de la clause `c1` utilisée). Un nœud est créé dont l'étiquette est $eq(a, b)$.

$$(3)\text{Ts \& gr}\ \frac{\neg fst(1) \wedge mhnb(1) = true \wedge \neg ct \wedge \neg flr,\quad v \leftarrow 2}{\begin{array}{c}T'=\{\epsilon,1,2\},\quad u'=2,\quad n'=3,\quad nu'=\{(\epsilon,1),(1,2),(2,3)\},\quad pd'=\{(\epsilon,goal),(1,p(a)),(2,eq(a,b))\},\\ cl'=\{(\epsilon,[]),(1,[c3]),(2,[c4])\},\quad fst'=\{(\epsilon,false),(1,false),(2,true)\}\end{array}}\ \{$$
$$scs(1),\quad p' = p(a),\quad ([c4], eq(a, b)) = cpini(2)\}$$

$$S_4 : \{\{\epsilon, 1, 2\}, 2, 3, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b))\},$$
$$\{(\epsilon, []), (1, [c3]), (2, [c4])\}, \{(\epsilon, false), (1, false), (2, true)\}, false, false\}$$

L'événement de trace extrait est (règle d'extraction Tree suc & go right de la section 5) :

`< 3  2  2  Exit  p(a) >` (suivi de `< 4  3  2  Call  eq(a,b) >`).

L'état actuel reconstruit est (règle de reconstruction Tree suc & go right de la section 6) :
$Q_4 = S_4/Q : \{\{\epsilon, 1, 2\}, 2, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b))\}\}$

mais sa reconstruction nécessite de connaître l'événement de trace suivant.

De S_4 à S_5

La règle (4) Leaf reached est la seule qui s'applique (première visite du nœud).

$$(4)\ \text{Lf rd}\ \frac{first(3) \wedge leaf(3) \wedge \neg ct \wedge fact(3)}{cl' = \{(\epsilon, [])(1, [c3]), (2, [])\}, fst' = \{(\epsilon, false), (1, false), (2, false)\}}\ \{\}$$

$$S_5 : \{\{\epsilon, 1, 2\}, 2, 3, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b))\},$$
$$\{(\epsilon, []), (1, [c3]), (2, [])\}, \{(\epsilon, false), (1, false), (2, false)\}, false, false\}$$

L'événement de trace extrait est (règle d'extraction Leaf reached de la section 5) :

`< 4  3  2  Call  eq(a,b) >` (suivi de `< 5  3  2  Fail  eq(a,b) >`)

L'état actuel reconstruit est (règle de reconstruction Leaf reached de la section 6) est invariant ($Q_5 = Q_4$) :
$Q_5 = S_5/Q : \{\{\epsilon, 1, 2\}, 2, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b))\}\}$

De S_5 à S_6

L'unification de $eq(a, b)$ et $eq(X, X)$ (clause c4) échoue, le nœud est donc en échec ($flr(2) = true$) et seule la règle (5) Tree failed s'applique (en effet ce n'est ni une première visite, ni un succès, ni un échec en cours ($flr = false$), ni une visite de la racine de l'arbre ($ct = false$)).

$$(5)\text{Tr fld}\ \frac{\neg first(2) \wedge \neg ct}{u' = \epsilon,\quad ct' = true,\quad flr' = true}\ \{flr(2)(= true) \vee flr(= false)\}$$

$S_6 : \{\{\epsilon, 1, 2\}, \epsilon, 3, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b)\},$
$\{(\epsilon, []), (1, [c3]), (2, [])\}, \{(\epsilon, false), (1, false), (2, false)\}, true, true\}$

Le nœud courant devient la racine, l'arbre complet en cours est donc un arbre d'échec et la raison de l'échec est la prédication $eq(a, b)$.

L'événement de trace extrait est (règle d'extraction Tree failed de la section 5) :
`< 5 3 2 Fail eq(a,b) >`

L'état reconstruit (règle de reconstruction Tree failed de la section 6) est : (on aurait pu en fait ajouter le paramètre flr dans l'état restreint pour rendre compte de l'état d'échec de l'arbre ; ici seul l'événement de trace porte cette information).
$Q_6 = S_6/Q : \{\{\epsilon, 1, 2\}, \epsilon, \{(\epsilon, 1), (1, 2), (2, 3)\}, \{(\epsilon, goal), (1, p(a)), (2, eq(a, b))\}$

De S_6 à S_7

Du fait que le nœud courant est la racine, seul un retour arrière peut intervenir. Seul le nœud 1 a une clause dans sa boîte (clause c3), et la clause est un fait. Seule la règle (6) Backtrack s'applique (pas de création de nouveau nœud) et le nœud 2 est supprimé). Le nœud courant devient 1.

$$(6)\qquad \text{Bkt}\ \frac{v \leftarrow 1,\quad \neg fst(\epsilon) \wedge hcp(\epsilon) \wedge ft(1) \wedge (flr(= T)\ \vee\ ct(= T))}{\begin{array}{c} T'=\{\epsilon,1\},\quad u'=1,\quad pd'=\{(\epsilon,goal),(1,p(a))\},\quad cl'=\{(\epsilon,[]),(1,[])\}, \\ ct=false,\quad flr'=false \end{array}}\ \{\}$$

$S_7 : \{\{\epsilon, 1\}, 1, 3, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(a))\},$
$\{(\epsilon, []), (1, [])\}, \{(\epsilon, false), (1, false)\}, false, false\}$

L'événement de trace extrait est (règle d'extraction Backtrack de la section 5) :
`< 6 2 2 Redo p(a) >` (suivi de `< 7 2 2 Exit p(b) >`)

L'état reconstruit (règle de reconstruction Backtrack de la section 6) est :
$Q_7 = S_7/Q : \{\{\epsilon, 1\}, 1, \{(\epsilon, 1), (1, 2)\}, \{(\epsilon, goal), (1, p(a))\}$

De S_7 à S_8

L'unification réussit à nouveau, toujours avec la même clause utilisée à la racine. Le frère du nœud courant (nœud 2) est donc recréé, avec numéro de création 4. C'est donc la règle (7) Tree suc & go right qui s'applique. La prédication appelée (toujours "faussement" notée $p(a)$) est un succès et devient $p(b)$. Elle peut avoir un frère (existence d'une deuxième prédication dans le corps de la

clause `c1` utilisée). Le nœud re-créé a pour étiquette $eq(b,b)$ et a une clause dans sa boîte (`c4`).

$$(7)\ \textsf{Ts \& gr}\ \frac{\neg first(1) \wedge mhnb(1) \wedge \neg ct \wedge \neg flr, \quad v \leftarrow 2}{\begin{array}{c}u'{=}2, \quad n'{=}4, \quad nu'{=}\{(\epsilon,1),(1,2),(2,4)\}, \quad pd'{=}\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\},\\ T'{=}\{\epsilon,1,2\}, \quad cl'{=}\{(\epsilon,[]),(1,[]),(2,[c4])\}, \quad fst'{=}\{(\epsilon,false),(1,false),(2,true)\}\end{array}}\ \{ scs(1), \quad p(b) = pud(1), \quad (c4, eq(b,b)) = cpini(2)\}$$

$$S_8 : \{\{\epsilon,1,2\},2,4,\{(\epsilon,1),(1,2),(2,4)\},\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\},\\ \{(\epsilon,[]),(1,[]),(2,[c4])\},\{(\epsilon,false),(1,false),(2,true)\},false,false\}$$

L'événement de trace extrait est (règle d'extraction Tree suc & go right de la section 5) :

`< 7 2 2 Exit p(b) >` (suivi de `< 8 4 2 Call eq(b,b) >`)

L'état reconstruit (règle de reconstruction Tree suc & go right de la section 6) est :

$Q_8 = S_8/Q : \{\{\epsilon,1,2\},2,\{(\epsilon,1),(1,2),(2,4)\},\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\}$

mais sa reconstruction nécessite de connaître l'événement de trace suivant (reconnaissance du fait que le prochain nœud visité sera un nouveau nœud).

De S_8 à S_9

Le nœud courant n'ayant pas encore été visité et la clause utilisée `c4` étant un fait, seule la règle (8) Leaf reached s'applique .

$$(8)\qquad \textsf{Leaf reached}\ \frac{first(2) \wedge leaf(2) \wedge \neg ct \wedge fact(2)}{cl' = \{(\epsilon,[]),(1,[]),(2,[])\}, \quad fst'(2) = false, \quad flr' = false}\ \{\}$$

$$S_9 : \{\{\epsilon,1,2\},2,4,\{(\epsilon,1),(1,2),(2,4)\},\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\},\\ \{(\epsilon,[]),(1,[]),(2,[])\},\{(\epsilon,false),(1,false),(2,false)\},false,false\}$$

L'événement de trace extrait est (règle d'extraction Leaf reached de la section 5) :

`< 8 4 2 Call eq(b,b) >`

L'état reconstruit (règle de reconstruction Leaf reached de la section 6) est :

$Q_9 = S_9/Q : \{\{\epsilon,1,2\},2,\{(\epsilon,1),(1,2),(2,4)\},\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\}$

mais l'événement de trace laisse l'état actuel courant invariant $Q_9 = Q_8$.

De S_9 à S_{10}

L'unification de $eq(b,b)$ et $eq(X,X)$ (clause `c4` du nœud numéro 4) réussit, il n'y a pas de frère potentiel et on n'est pas dans un état d'échec ($flr = false$). Seule la règle (9) Tree success s'applique donc. $pred(2) = eq(b,b)$ reste invariant (fait clos).

$$(9)\qquad \textsf{Tree success}\ \frac{\neg first(2) \wedge \neg mhnb(2) \wedge \neg ct \wedge \neg flr, \quad v \leftarrow \epsilon}{u' = \epsilon, \quad pd' = pd}\ \{scs(2), \quad eq(b,b) = pud(2)\}$$

$$S_{10} : \{\{\epsilon,1,2\},\epsilon,4,\{(\epsilon,1),(1,2),(2,4)\},\{(\epsilon,goal),(1,p(b)),(2,eq(b,b))\},\\ \{(\epsilon,[]),(1,[]),(2,[])\},\{(\epsilon,false),(1,false),(2,false)\},false,false\}$$

L'événement de trace extrait est (règle d'extraction Tree success de la section 5) :

`< 9 4 2 Exit eq(b,b) >` (suivi de `< 10 1 1 Exit goal >`)

L'état reconstruit (règle de reconstruction Tree success de la section 6) est :
$Q_{10} = S_{10}/Q : \{\{\epsilon, 1, 2\}, \epsilon, \{(\epsilon, 1), (1, 2), (2, 4)\}, \{(\epsilon, goal), (1, p(b)), (2, eq(b, b))\}$

De S_{10} à S_{11}

Finalement, la règle (10) Tree success s'applique à nouveau (pas de frère possible). On obtient alors un arbre de preuve complet. La prédication de la racine *goal* est invariante.

$$(10) \qquad \text{Tree success } \frac{\neg first(\epsilon) \wedge \neg mhnb(\epsilon) \wedge \neg ct \wedge \neg flr, \quad v \leftarrow \epsilon}{u' = \epsilon, \quad pd' \leftarrow upcp(pd, u, p), \quad ct' = true} \{scs(\epsilon), goal = pud(\epsilon)\}$$

$S_{11} : \{\{\epsilon, 1, 2\}, \epsilon, 4, \{(\epsilon, 1), (1, 2), (2, 4)\}, \{(\epsilon, \bar{goal}), (1, p(b)), (2, eq(b, b))\},$
$\{(\epsilon, []), (1, []), (2, [])\}, \{(\epsilon, false), (1, false), (2, false)\}, true, false\}$

L'événement de trace extrait est (règle d'extraction Tree success de la section 5) :

`< 10 1 1 Exit goal >`

(suivi d'aucun autre car la racine de l'arbre est à nouveau atteinte et il n'y a plus de point de choix dans l'arbre).

L'état reconstruit (règle de reconstruction Tree success de la section 6) est :
$Q_{11} = S_{11}/Q : \{\{\epsilon, 1, 2\}, \epsilon, \{(\epsilon, 1), (1, 2), (2, 4)\}, \{(\epsilon, goal), (1, p(b)), (2, eq(b, b))\}$

mais l'événement de trace n'engendre aucune modification de l'état actuel courant $Q_{11} = Q_{10}$.

Comme l'arbre T est complet ($ct = true$) et qu'il n'y a plus de point de choix dans l'arbre ($hcp(\epsilon) = false$), aucune règle de retour arrière ne peut s'appliquer. On a obtenu une solution, et aucune règle ne s'applique plus. La trace (virtuelle) se termine donc et on a obtenu un arbre de preuve complet. La figure 9 illustre les arbres (partiel et complet) construits, ainsi que les enchaînements des ports dans la trace de cet exemple.

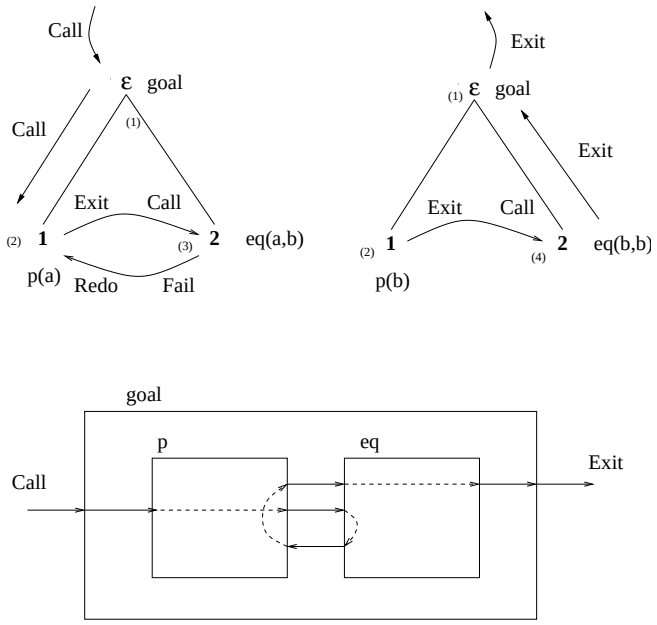


Fig. 9. Arbre de preuve obtenu et parcours dans les boîtes

ANNEXE C : Adéquation de la trace actuelle (Byrd simplifié)

[**Rappel : 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 $r \in R$:

$\forall\, e,\, e',\, r', S,\, S',\, S''$,
$\mathcal{E}_r(S, S') = e \ \wedge\ \mathcal{E}_{r'}(S', S'') = e'$
(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 $T_v =< S_0, v_t^* >$.

La preuve se fait en plusieurs étapes
- Etude de quelques propriété dynamiques de la SO concernant l'enchainement de règles et des ports. Ces lemmes sont illustrés par les schémas de la figures 10.
- Vérification de la condition 1 : exclusivité des conditions d'identification des règles dans le schéma de reconstruction), conséquence directe des lemmes précédents.
- Enfin vérification de la condition 2 : correction du schéma de reconstruction pour le sous-ensemble de paramètres $\{T, u, num, pred\}$.

Démonstration (Preuve des lemmes). On aura besoin des lemmes suivants établis sur la trace virtuelle, illustrés par la figure 10, qui montrent les enchaînements possibles des règles et des ports pour toute trace. Le point d'entrée obligé dans la figure (événement faisant suite à un événement initial non traité ici) est l'état dit "top" ou unévénement de port **Call**.

La figure 10 **a** illustre l'enchaînement des règles dans la SO. Elle inclue dans les états **c2**, **e2** et **r2** l'incrément du compteur de nœuds créés (n), montrant ainsi que seule la traversée de ces nœuds fait croître celui-ci. Cette propriété est utilisé explicitement pour discriminer les règles dans la deuxième partie de la démonstration.

Pour l'enchaînement des ports dans une trace actuelle (figure 10 **b**), les flèches sont pour la plupart évidentes. On observera ici simplement que la possibilité d'avoir un **Redo** après un **Exit** est liée à la condition ct. La possibilité d'avoir des **Call** successifs est liée à l'usage exclusif de la règle Lf rcd & go down. L'absence de transition de **Fail** vers **Call** est lié à la condition fst qui l'empêche. L'absence de transition **Call** vers **Redo** est due à la condition $flr \vee ct$. Enfin les **Redo** ne peuvent boucler sur eux-mêmes.

Démonstration (Condition 1).
L'ensemble des conditions exclusives telles que décrites dans le schéma de reconstruction (figure 4) est rappelé ici (figure 11). Seuls les éléments de trace utiles sont indiqués.

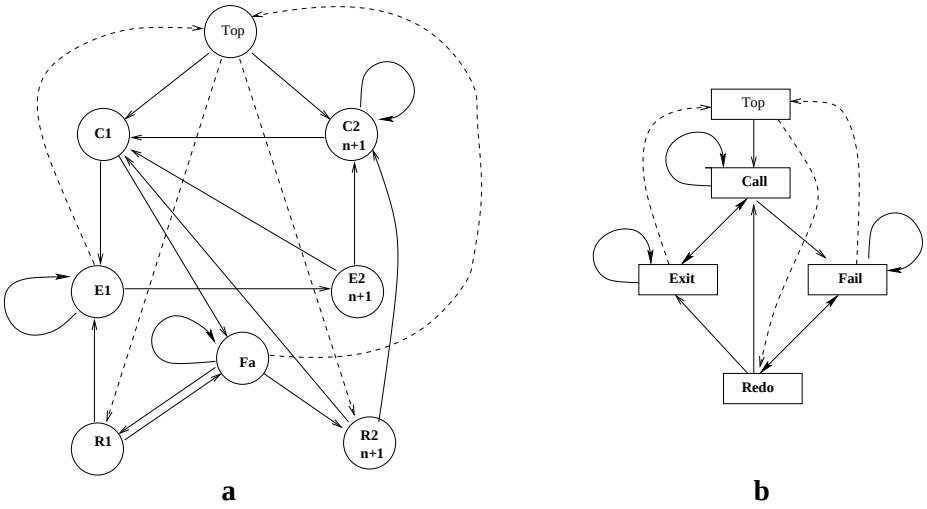


Fig. 10. Enchaînement des règles dans la SO (**a**) et des ports dans toute trace (**b**) (le diagramme **b** est une abstraction du diagramme **a**)

La discrimination des règles se fait en premier à partir du port qui est toujours présent dans les événements de trace (ceci est suffisant pour les événements de port **Fail**), puis sur l'accroissement ou non du compteur de nœuds , avec une particularité pour les événements de port **Exit** où la condition est un peu plus complexe.

Noter que l'on utilise ici une information qui découle de la propriété de correction : le numéro de création du nœud courant $nd(u)$ est toujours égal au premier attribut de l'événement de trace associé r, soit $nd(u) = r$ pour tous les événements de port différent de **Redo**. Dans le cas en effet de la règle Bkt & go down, le nœud tracé ($nd(v)$) n'apparaît ni dans S ni dans S' ; dans le cas de la règle Backtrack, il apparaît seulement dans l'état S'.

Démonstration (Condition 2 : correction).

Pour chaque règle $r \in R$ de la SO (il y en a 7), on indique les éléments suivants.

- Description de la transition correspondant à la règle r en mettant en évidence ce qui relève de l'état "avant" (S) et "après" (S') (tous les paramètres de cet état sont primés).
- On en déduit immédiatement l'état virtuel S'/Q restreint à Q.
- On regarde alors l'événements de trace extrait $e = \mathcal{E}_r(S, S')$ et les suivants possibles $e' = \mathcal{E}_{r'}(S', S'')$, en se limitant aux attributs utilisés dans la règle correspondante du schéma de reconstruction (l'événement correspondant à la règle r' n'est pas décrit complètement, laissant la possibilité d'avoir

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

$$\textsf{Lf rcd \& go down}\ \frac{r'>r}{}\ \{< r\ \ l\ \ \textbf{Call}\ \ p>;\ < r'\ \ p'>\}$$

$$\textsf{Tree success}\ \frac{r'<r\vee nd(r)=\epsilon}{}\ \{< r\ \ l\ \ \textbf{Exit}\ \ p>;\ < r'>\}$$

$$\textsf{Tree suc \& go right}\ \frac{r'>r\wedge nd(r)\neq\epsilon}{}\ \{< r\ \ l\ \ \textbf{Exit}\ \ p>;\ < r'\ \ p'>\}$$

$$\textsf{Tree failed}\ \ ——\ \ \{< r\ \ l\ \ \textbf{Fail}\ \ p>\}$$

$$\textsf{Backtrack}\ \frac{r'=r}{}\ \{< r\ \ l\ \ \textbf{Redo}\ \ p>;\ < r'>\}$$

$$\textsf{Bkt \& go down}\ \frac{r'>r}{}\ \{< r\ \ l\ \ \textbf{Redo}\ \ p>;\ < r'\ \ p'>\}$$

Fig. 11. Conditions exclusives du schéma de reconstruction

en fait différents événements et/ou règles r'). En fait on se limite aux propriétés communes aux attributs des événements suivants possibles.
- On donne la règle correspondante du schéma de reconstruction $\mathcal{C}_r$ décrivant la transition $< S/Q, Q' >$.
- On vérifie que la condition $Cond_r(e, e')$ est bien vérifiée.
- On donne la description de l'état Q' résultants de la règle de reconstruction ($Q' = \mathcal{C}_r(e, e', S/Q)$).
- Enfin et si cela n'est pas évident par construction, on montre l'égalité de S'/Q et Q'.

Règle Leaf reached

$$\textsf{Leaf reached}\ \frac{fst(u)\wedge lf(u)\wedge\neg ct\wedge ft(u)}{\frac{T'=T,\ \ u'=u,\ \ nu'=nu,\ \ pd'=pd,\ \ cl'=upcp(cl,u),}{fst'(u)=F,\ \ flr'=F}}\ \{\}$$

$S' = S$

$\mathcal{E}_{\textsf{Leafreached}}(S, S') =< nu(u)\ \ lp(u)\ \ Call\ \ pd(u) >$
$\mathcal{E}_{r'}(S', S'') =< nu'(u')... >$

$\mathcal{C}_{\textsf{Leafreached}}$ $\qquad$ $\textsf{Leaf reached}\ \frac{Cond_{\textsf{Leafreached}}(e, e')}{}\ \{< ... >\}$

(aucun élément utile)

$Cond_{\textsf{Leafreached}}(e, e') = (nu(u) = nu'(u'))$ (vraie, car $u' = u$ et $nu' = nu$)

La restriction à Q de S ou S' est invariante. De plus la règle de reconstruction ne modifie aucun des paramètres de de S/Q. Il en rsulte que $Q' = Q = S'/Q$.

Règle Lf rcd & go down

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

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

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

$$\textsf{Lfr \& gd}\ \frac{Cond_{\textsf{Lfrcd\&godown}}(e, e')}{\begin{array}{c}u'=crc((nd(nu(u))),\quad T'=T\cup\{u'\},\quad nu'(u')=nu'(u'),\\ pd'(u')=pd'(u')=p\_cpini(u')\end{array}}\ \{e\ \ e'\}$$

$Cond_{\textsf{Lfrcd\&godown}}(e, e') = (nu'(crc(u)) > nu(u))$ en effet tout nouveau nœud créé l'est avec un numéro supérieur à tous ceux déjà existants ($n' > n$).

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

Règle Tree success

$$\textsf{Tr suc}\ \frac{\neg fst(u) \wedge \neg mhnb(u) \wedge \neg ct \wedge \neg flr}{T' = T,\quad u' = pt(u),\quad nu' = nu,\quad pd'(u) = pud(u),\quad (u = \epsilon) \Rightarrow (ct' = T)}\ \{ scs(u),\quad p = pud(u)\}$$

$S'/Q = \{T, u' = pt(u), nu' = nu, pd'(u) = pud(u)\}$

$\mathcal{E}_{\textsf{Treesuccess}}(S, S') =< nu(u)\ \ lp(u)\ \ Exit\ \ pud(u) >$
$\mathcal{E}_{r'}(S', S'') =< nu'(pt(u))... >$

$$\textsf{Tree success}\ \frac{Cond_{\textsf{Treesuccess}}(e, e')}{u' = pt(u),\quad pd'(u) = pud(u)}\ \{< nu(u)\ \ lp(u)\ \ Exit\ \ pud(u) >;\ < nu'(pt(u))... >\}$$

$Cond_{\textsf{Treesuccess}}(e, e') = (nu'(pt(u)) < nu(u) \vee u = \epsilon)$ est vérifiée ; en effet tout nœud inférieur dans l'ordre lexicographique a été créé avec un numéro d'ordre inférieur.

$Q' = \{T, u' = pt(u), nu' = nu, pd'(u') = pud(u)\} = S'/Q.$

Règle Tree suc & go right

$$\textsf{Ts \& gr}\ \frac{\neg fst(u) \wedge mhnb(u) \wedge \neg ct \wedge \neg flr}{\begin{array}{c}u'=crnb(u),\quad T'=T\cup\{u'\},\quad n'=n+1,\quad nu'(u')=n',\\ pd'(u)=pud(u),\quad pd'(u')=p,\quad cl'(u')=c,\quad fst'(u')=true\end{array}}\ \{scs(u),\ (c,p) = cpini(u')\}$$

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

$\mathcal{E}_{\textsf{Treesuc\&goright}}(S, S') =< nu(u)\ \ lp(u)\ \ Exit\ \ pud(u) >$
$\mathcal{E}_{r'}(S', S'') =< nu'(crnb(u)) = n'\ \ p\_cpini(u') >$

$$\textsf{Ts \& gr}\ \frac{Cond_{\textsf{Ts\&gr}}(e, e')}{\begin{array}{c}u'=crnb(u),\quad T'=T\cup\{u'\},\quad nu'(u')=n',\\ pd'(u)=pud(u),\quad pd'(u')=p\_cpini(u')\end{array}}\ \{< nu(u)\ \ lp(u)\ \ \textbf{Exit}\ \ pud(u) >;\ < n'\ \ p\_cpini(u') >\}$$

$Cond_{\mathsf{Treesuc\&goright}}(e,e') = (n' = n+1) > nu(u) \wedge u \neq \epsilon$ est vérifiée ; en effet tout nœud inférieur dans l'ordre lexicographique a été créé avec un numéro d'ordre inférieur, soit $n \geq nu(u)$. De plus, comme u' est frère de u, u ne peut être racine.

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

Règle Tree failed

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

$S'/Q = \{T, u' = pt(u), nu, pd\}$

$\mathcal{E}_{\mathsf{Tree\ failed}}(S, S') =< nu(u)\ \ lp(u)\ \ Fail\ \ pd(u) >$
$\mathcal{E}_{r'}(S', S'') =< nu'(pt(u))... >$

$$\mathsf{Tree\ failed}\ \frac{Cond_{\mathsf{Tree\ failed}}(e,e')}{u' = pt(u)}\ \{< nu(u)\ \ lp(u)\ \ \mathbf{Fail}\ \ pd(u) >\}$$

Il n'y a pas de condition autre que le nom du port.

$Q' = \{T, u' = pt(u), nu, pd\} = S'/Q.$

Règle Backtrack

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

$S'/Q = \{T - \{y|y > u'\}, u' = gcp(u), nu, pd\}$

$\mathcal{E}_{\mathsf{Backtrack}}(S, S') =< nu(v)\ \ lp(v)\ \ Redo\ \ pd(v) >$ avec $v = gcp(u)$.
$\mathcal{E}_{r'}(S', S'') =< nu'(v)... >$ avec $nu' = nu$.

L'événement de trace produit a pour premier attribut $nu'(v)$ qui est égal à $nu(u)$ car les événements suivants sont différents de **Redo**.

$$\mathsf{Backtrack}\ \frac{Cond_{\mathsf{Backtrack}}(e,e')}{u' = nd(nu(gcp(u))) = gcp(u) = v, \quad T' = T - \{y|y > v\}}\ \{ < nu(v)\ \ lp(v)\ \ \mathbf{Redo}\ \ pd(v) > ;\ < nu'(v) >\}$$

$Cond_{\mathsf{Backtrack}}(e,e') = (nu(v) = nu'(v))$ est vérifiée (cf ci-dessus).

$Q' = \{T - \{y|y > v\}, u' = gcp(u), nu, pd\} = S'/Q.$

Règle Bkt & go down

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

$S'/Q = \{T - \{y|y > v\} \cup \{w\}, u' = w, nu' = upn(nu, v) \cup \{(u', n')\}, pd' = upcp(pd, v) \cup \{(u', p\_cpini(w))\}\}$ avec $v = gcp(u)$ et $w = crc(v)$.

$\mathcal{E}_{\mathsf{Bkt\&godown}}(S, S') =< nu(v) \;\; lp(v) \;\; Redo \;\; pd(v) >$
$\mathcal{E}_{r'}(S', S'') =< nu'(w) = n' \;\; pd'(w) = p\_cpini(w) >$

$$\mathsf{Bkt\ \&\ gd}\ \frac{Cond_{\mathsf{Bkt\&godown}}(e, e')}{\frac{v=nd(nu(v))=gcp(u),\quad u'=crc(v),\quad T'=T-\{y|y>v\}\cup\{u'\},}{nu'=upn(nu,v)\cup\{(u',n')\},\quad pd'=upcp(pd,v)\cup\{(u',p\_cpini(u'))\}}}\ \{$$

$< nu(v) \;\; lp(v) \;\; \mathbf{Redo} \;\; pd(v) > ; < nu'(w) = n+1 \;\; pd'(w) = p\_cpini(w) >\}$

$Cond_{\mathsf{Bkt\&godown}}(e, e') = nu'(w) > nu(v)$ est vérifiée ; en effet tout nouveau nœud créé l'est avec un numéro supérieur à tous ceux déjà existants. Le nœud $nd(u)$ peut avoir été supprimé, mais tout $y < w$ dans l'ordre lexicographique des nœuds est tel que $n + 1 > nu(y)$.

$Q' = \{T - \{y|y > v\} \cup \{u'\}, u' = crc(v), nu' = upn(nu, v) \cup \{(u', n')\}, pd' = upcp(pd, v) \cup \{(u', p\_cpini(u'))\}\} = S'/Q$ (avec $v = gcp(u)$).

ANNEXE D : Analyse des différents modèles de traces

On compare ici le modèle présenté ici, dit "Byrd simplifié" (m1), celui de GNU Prolog (m2), et le modèle de Byrd avec l'implantation par méta-interprète (m3).

Sur le même exemple (Annexe B) la trace produite par la SO présentée ici (comme celle de Byrd) et celle de GNU Prolog 1.2.16 copyright (C) 1999-2002 Daniel Diaz [8] sont les mêmes avec une seule différence sur les événements `8` et `9`. Cela résulte du fait que dans GNU `r` est le rang du nœud courant atteint dans l'arbre de preuve (selon l'ordre lexicographique). Le chrono est ajouté. Seuls les événements 8 et 9 diffèrent donc (voir la trace complète plus bas).

```
GNU
8     3    2    Call:   eq(b,b) ?
9     3    2    Exit:   eq(b,b) ?

Byrd
8     4    2    Call    eq(b,b)
9     4    2    Exit    eq(b,b)
```

On observera dans la suite que, si l'on se limite à la suite des ports, les modèles sont inclus les uns dans les autres (`m1` $\subseteq$ `m2` $\subseteq$ `m3`) ; la seule différence portant, entre `m1` et `m2` sur le premier attribut (cf ci-dessus).

Il est intéressant d'observer ici que les trois traces obtenues sur cet exemple simple ont la même suite de ports.

L'exemple 2 suivant, plus sophistiqué, met en évidence de quelle manière elles diffèrent en fait sensiblement. Les SO correspondantes sont décrites dans l'annexe E suivante.

```
goal:-q(_).
q(X):-p1(X),p2(X),eq(X,b).
p1(X) :- p(X).
p(a).
p(b).
p(c).
p2(_).
eq(X,X).
```

Trace obtenue avec le modèle `m1` (Byrd simplifié) :

```
1     1    1    Call    goal
2     2    2    Call    q(_86)
3     3    3    Call    p1(_86)
4     4    4    Call    p(_86)
5     4    4    Exit    p(a)
6     3    3    Exit    p1(a)
```

```
7     5    3    Call    p2(a)
8     5    3    Exit    p2(a)
9     6    3    Call    eq(a,b)
10    6    3    Fail    eq(a,b)
11    4    4    Redo    p(a)
12    4    4    Exit    p(b)
13    3    3    Exit    p1(b)
14    7    3    Call    p2(b)
15    7    3    Exit    p2(b)
16    8    3    Call    eq(b,b)
17    8    3    Exit    eq(b,b)
18    2    2    Exit    q(b)
19    1    1    Exit    goal
20    4    4    Redo    p(b)
21    4    4    Exit    p(c)
22    3    3    Exit    p1(c)
23    9    3    Call    p2(c)
24    9    3    Exit    p2(c)
25    10   3    Call    eq(c,b)
26    10   3    Fail    eq(c,b)
27    2    2    Fail    q(_86)
27    1    1    Fail    goal
yes
```

Trace obtenue avec le modèle `m2` (traceur de GNU-Prolog).

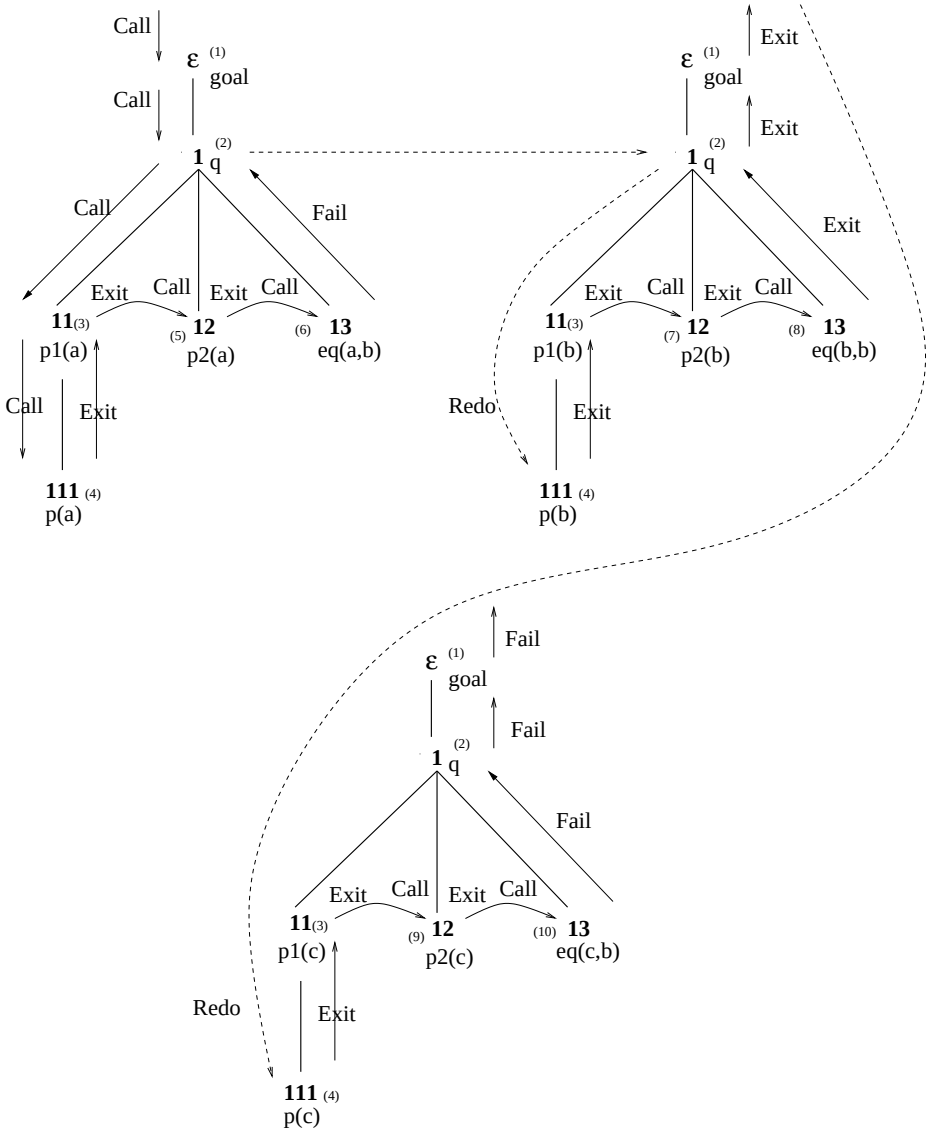
```
1     1    1  Call: goal ?
2     2    2  Call: q(_38) ?
3     3    3  Call: p1(_38) ?
4     4    4  Call: p(_38) ?
5     4    4  Exit: p(a) ?
6     3    3  Exit: p1(a) ?
7     5    3  Call: p2(a) ?
8     5    3  Exit: p2(a) ?
9     6    3  Call: eq(a,b) ?
10    6    3  Fail: eq(a,b) ?
11    3    3  Redo: p1(a) ?
12    4    4  Redo: p(a) ?
13    4    4  Exit: p(b) ?
14    3    3  Exit: p1(b) ?
15    5    3  Call: p2(b) ?
16    5    3  Exit: p2(b) ?
17    6    3  Call: eq(b,b) ?
17    6    3  Exit: eq(b,b) ?
19    2    2  Exit: q(b) ?
20    1    1  Exit: goal ?
```

```
21    1    1  Redo: goal ?
22    2    2  Redo: q(b) ?
23    3    3  Redo: p1(b) ?
24    4    4  Redo: p(b) ?
25    4    4  Exit: p(c) ?
26    3    3  Exit: p1(c) ?
27    5    3  Call: p2(c) ?
28    5    3  Exit: p2(c) ?
29    6    3  Call: eq(c,b) ?
30    6    3  Fail: eq(c,b) ?
31    2    2  Fail: q(_38) ?
32    1    1  Fail: goal ?
```

Trace obtenue avec le modèle `m3` (Byrd, méta-interprète de l'annexe A et instrumentation du code).

```
1     1    1    Call    goal
2     2    2    Call    q(_86)
3     3    3    Call    p1(_86)
4     4    4    Call    p(_86)
5     4    4    Exit    p(a)
6     3    3    Exit    p1(a)
7     5    3    Call    p2(a)
8     5    3    Exit    p2(a)
9     6    3    Call    eq(a,b)
10    6    3    Fail    eq(a,b)
11    5    3    Redo    p2(a)
12    5    3    Fail    p2(a)
13    3    3    Redo    p1(a)
14    4    4    Redo    p(a)
15    4    4    Exit    p(b)
16    3    3    Exit    p1(b)
17    7    3    Call    p2(b)
18    7    3    Exit    p2(b)
19    8    3    Call    eq(b,b)
20    8    3    Exit    eq(b,b)
21    2    2    Exit    q(b)
22    1    1    Exit    goal
23    1    1    Redo    goal
24    2    2    Redo    q(b)
25    8    3    Redo    eq(b,b)
26    8    3    Fail    eq(b,b)
27    7    3    Redo    p2(b)
28    7    3    Fail    p2(b)
29    3    3    Redo    p1(b)
30    4    4    Redo    p(b)
```

```
31    4    4    Exit    p(c)
32    3    3    Exit    p1(c)
33    9    3    Call    p2(c)
34    9    3    Exit    p2(c)
35    10   3    Call    eq(c,b)
36    10   3    Fail    eq(c,b)
37    9    3    Redo    p2(c)
38    9    3    Fail    p2(c)
39    3    3    Redo    p1(c)
40    4    4    Redo    p(c)
41    4    4    Fail    p(_86)
42    3    3    Fail    p1(_86)
43    2    2    Fail    q(_86)
44    1    1    Fail    goal
yes
```



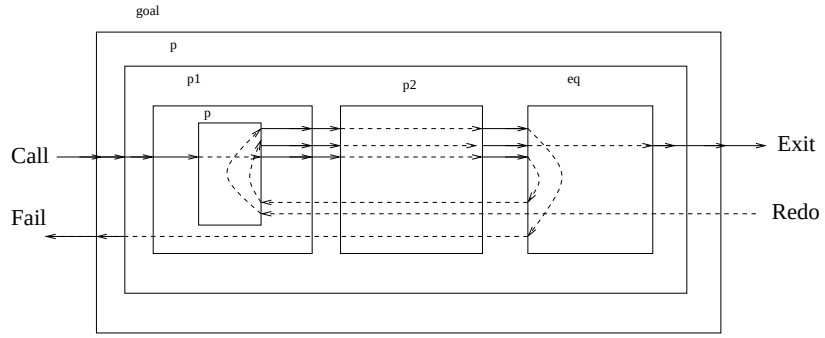


Fig. 12. Illustration de l'exemple 8 avec arbres et boîtes (modèle m1, 28 événements)

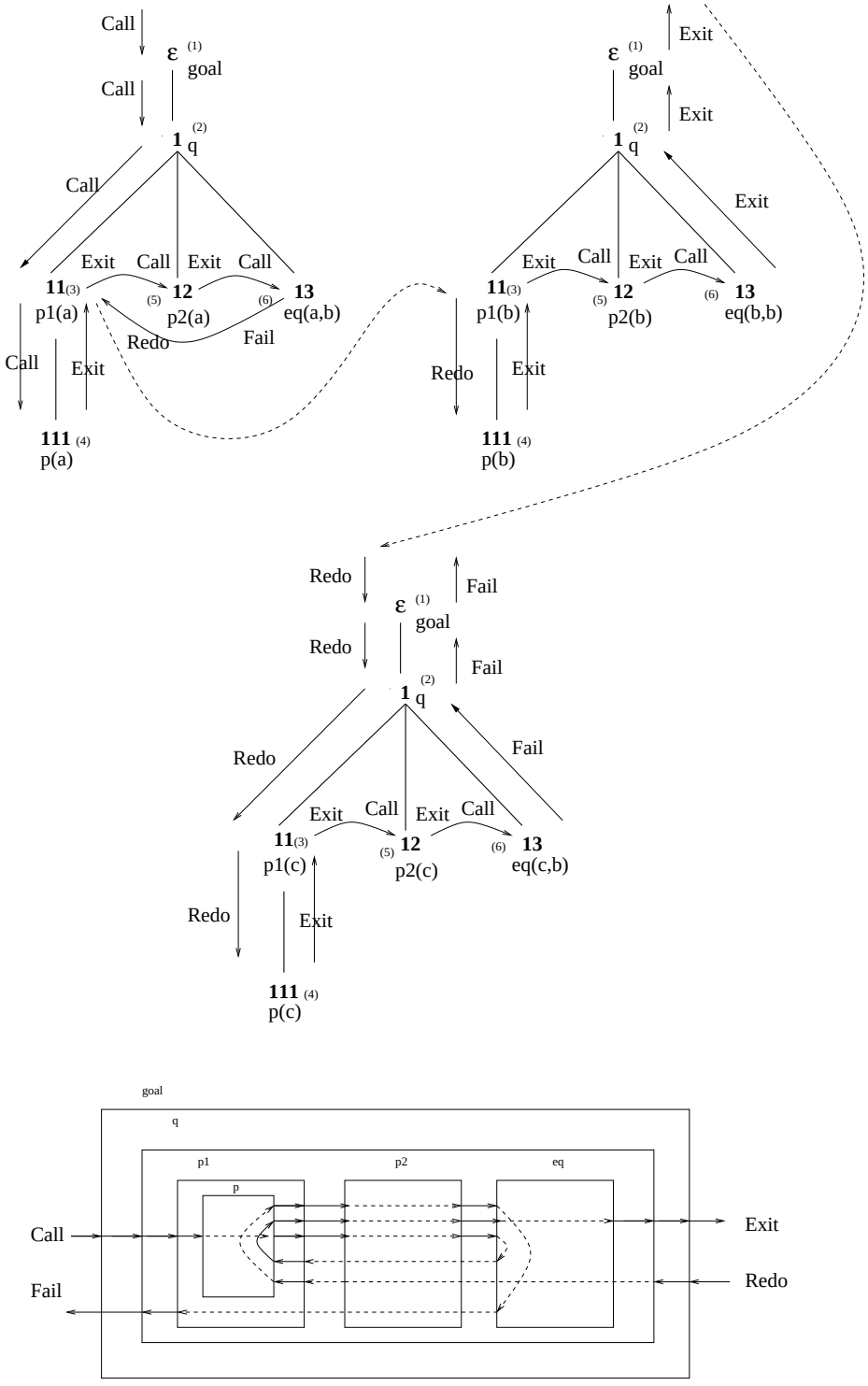


Fig. 13. Illustration de l'exemple 2 avec arbres et boîtes (modèle m2, 32 événements)

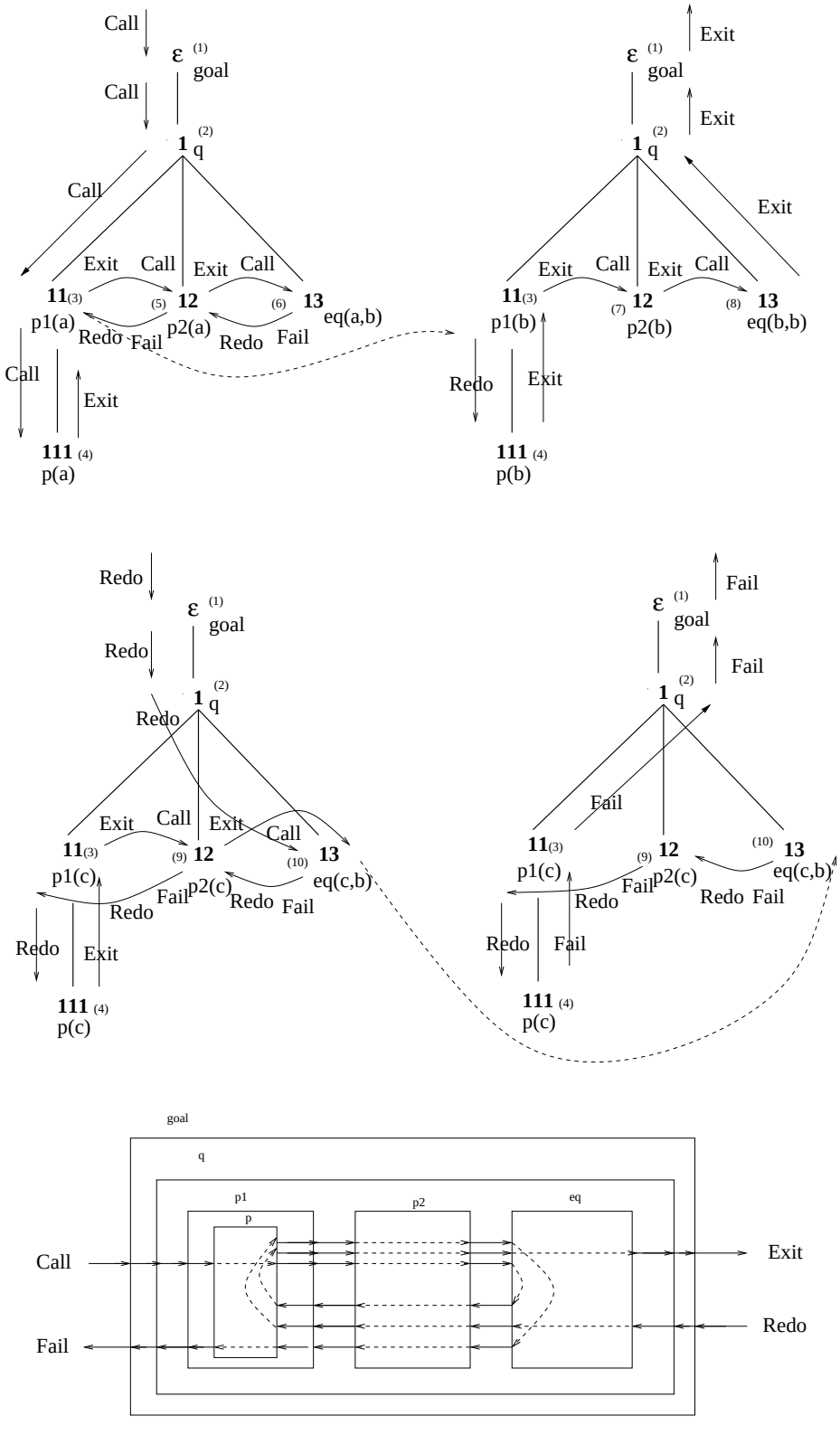


Fig. 14. Illustration de l'exemple 2 avec arbres et boîtes (modèle m3, 44 événements)

ANNEXE E : Sémantique observationnelle pour les trois modèles (m1, m2, m3)

On présente ici une SO qui correspond à une forme de recherche d'arbres de preuve complets, paramétrable par des fonctions de choix (clauses et feuilles). L'objectif est de pouvoir rendre compte avec la même SO de plusieurs formes de traces. Cette SO, plus détaillée que la précédente illustre mieux la distinction entre trace virtuelle et trace actuelle intégrale ou non. En effet pour obtenir une trace correspondant au modèle `m1` par exemple, il y aura lieu de filtrer la trace virtuelle pour en éliminer certains événements [15].

La sémantique opérationnelle de résolution Prolog sous-jacente correspond à la SLDT-résolution telle que présentée dans [7] p. 57. C'est une stratégie descendante de construction d'arbre de preuve partiels qui repose sur deux fonctions de choix : choix d'une feuille incomplète à développer, choix d'une clause parmi celles associables à la feuille. Un nœud auquel il reste des clauses associées constitue un point de choix. La SLDT-résolution construit des "squelettes" d'arbre et les "décore" de manière non déterministe. Le déterminisme est introduit de manière à garder la complétude du schéma de résolution et à pouvoir reproduire les traces souhaitées.

On se limitera donc ici au parcours standard descendant gauche droite ; mais d'autres stratégies peuvent être considérées à condition de garder à celles-ci leur propriété d'être complètes ("full" dans [7]). Le choix des feuilles consistera donc à choisir le première feuille non visitée (dans l'ordre total des nœud d'un arbre) dont on cherchera à développer systématiquement le sous-arbre, et en cas d'échec local ou de succès avec un arbre de preuve complet, on reviendra toujours au dernier point de choix créé.

On pourrait adopter des stratégies plus générales, déterministes et complètes, en particulier en autorisant un ordonnancement dynamique des descendants d'un nœud, ou même en changeant complètement de stratégie (en largeur d'abord par exemple), mais ces stratégies n'ont plus aucun rapport avec le modèle des boîtes ; en particulier l'ordonnancement des ports **Call** ou **Redo** n'a plus grand chose à voir avec la structure de boîtes puisque l'on peut alors "sauter" d'un nœud dans un sous-arbre à n'importe quel nœud d'un autre sous-arbre, donc d'une boîte à n'importe quelle autre.

Les prédicats associés aux nœuds , par la fonction *pred* sont ceux produits par le squelette.

Ce modèle supprime en partie les fonctions externes dans la mesure où succès et échecs peuvent être traités comme des variables d'état global (état du sous-arbre courant).

[15] Le modèle présenté ici suppose que le choix de la trace soit fait de manière externe. Il s'agit donc, dans cette première approche, de donner en fait une forme unifiée à trois SO différentes (une par modèle). La possibilité de présenter les trois modèles dans une SO unique, bien que possible du fait de l'inclusion partielle des traces, est encore à l'étude.

Paramètres de la trace virtuelle

L'état courant a 13 paramètres :

$\{T, u, n, num, pred, chosen\_claus, claus\_list, first, \sigma, ct, flr, scs, bk3\}$.

1. $T : T$ est un arbre étiqueté avec un numéro de création ou re-création, une prédication et un sous-ensemble de clauses du programme P. Il est décrit ici par ses fonctions de construction/reconstruction, 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 (mais ce pourrait être n'importe quel alphabet ordonné) et dénotés ϵ, 1, 11, 12, 112, ... L'ordre lexicographique est le suivant : u, v, w sont des mots, $ui < uiv (v \neq \epsilon)$, and $uiv < ujw$ si $i < j$, ϵ est le mot 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). Il correspond à son ordre de création, re-création ou re-visite ; c'est aussi le numéro de la boîte associée à un nœud. C'est le numéro du dernier nœud créé.
4. $num : T \rightarrow \mathcal{N}$: (abbrev. nu) $nu(v)$ est le numéro (entier positif) associé au nœud v dans T. .
5. $pred : T \rightarrow \mathcal{H}$: (abbrev. pd) $pd(v)$ 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). Cette prédication est invariante et correspond à un élément du dorps de la clause choisie au nœud parent.
6. $chosen\_claus : T \rightarrow P$: (abbrev. cc) $cc(v)$ est la clause choisie pour résoudre la prédication courante. $cc(v)$ est vide si aucune clause ne peut être choisie.
7. $claus\_list : T \rightarrow 2^P$: (abbrev. cl) $cl(v)$ est une liste de clauses de P (même ordre que dans P) susceptible de contribuer à la définition du prédicat de $pd(v)$ associée au nœud v dans T. Selon les clauses sélectionnées dans $cl(v)$, on peut obtenir différentes sémantiques opérationnelles (mais il y a toujours une seule SO). Si la boîte est vide, la predication $pd(v)$ ne peut être résolue et le nœud sera en échec (*failure*). Cette liste de clauses est définie par un ordre externe lorsque la predication est appelée (voir $claus\_init$ dans les fonctions externes).
8. $\sigma : T \rightarrow Subst$: $\sigma(v)$ est la substitution courante, celle qui, appliquée à $pd(v)$, donne la prédication appelée au nœud v. La substitution vide est notée $\emptyset$. La substitution échec est notée $\perp$. La composition de substitutions est notée par justaposition des substitutions, ex $\mu\sigma$.
9. $first : T \rightarrow Bool$: (abbrev. fst) $fst(v)$ est vrai ssi v est un nœud de T qui n'a pas encore été visité.
10. $ct \in Bool$: ct est l'indicateur de construction achevée (arbre complètement construit et visité, retour à la racine) de T : $true$ ssi le nœud courant est redevenu ϵ lors d'une remonté dans l'arbre (en succès ou échec).

11. $flr \in Bool$: flr est l'indicateur d'état du sous-arbre ($true$ si en échec, $false$ sinon, ce qui n'est pas synonyme de succès).
12. $scs \in Bool$: scs est l'indicateur d'état du sous-arbre ($true$ si succès, $false$ sinon, ce qui n'est pas synonyme d'échec).
13. $bk3 \in Bool$: $bk3$ est l'indicateur de retour arrière actif de la trace produite pour le modèle `m3` seulement ($true$ ssi un parcours inverse de l'arbre courant est en cours).

Noter que la SO utilise comme donnée externe la méthode de trace (`m1`, `m2` ou `m3`), qui sont alors des booléens exlcusifs, mais ces éléments n'ont pas été introduits dans l'état afin de ne pas le surcharger.

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) $pt(v)$ est l'ancêtre direct de v dans T. Pour simplifier le modèle , on suppose que $pt(\epsilon) = \epsilon$.
- $leaf : T \rightarrow Bool$: (abbrev. lf) $lf(v)$ est vraie ssi v est une feuille dans T.
- $create\_children : T \rightarrow T^*$: (abbrev. crc) $v = crc(v)$ est la liste des nouveaux enfants de v in T.
- $first\_elem : liste(E) \rightarrow E$: (abbrev. fe) $v = fe(l)$ est le premier élément de la liste l.
- $has\_a\_next\_node : T \rightarrow Bool$: (abbrev. hnn) $hnn(v)$ est vrai si le nœud v a un nœud suivant dans l'arbre T. Faux si v est ϵ.
- $next\_right\_node : T \rightarrow T$: (abbrev. nrn) $w = nrn(v)$ est le nœud suivant v dans T. La racine ϵ ne peut avoir de suivant.
- $has\_a\_choice\_point : T \rightarrow Bool$: (abbrev. hcp) $hcp(v)$ est vrai ssi il existe un point de choix w dans le sous-arbre de racine v dans T ($cl(w)$ contient au moins une clause).
- $youngest\_child\_with\_choice\_point : T \rightarrow T$: (abbrev. $ycwcp$) $w = ycwcp(v)$ est le plus récent point de choix dans le sous-arbre de racine v dans T ($cl(w)$ contient au moins une clause) selon l'ordre lexicographique des nœuds dans T.
- $child\_with\_greates\_choice\_point : T \rightarrow T$: (abbrev. $cwgcp$) $cwgcp(v)$ est le nœud enfant v dans T dont le sous-arbre contient le plus récent point de choix.
- $greatest\_choice\_point : T \rightarrow T$: (abbrev. gcp) $w = gcp(v)$ est le plus grand point de choix dans le sous-arbre de racine v dans T ($cl(w)$ contient au moins une clause) selon l'ordre lexicographique des nœuds dans T.
- $fact : Claus \rightarrow Bool$: (abbrev. ft) $ft(c)$ est vrai ssi c est un fait.
- $update\_number : F_X, T \rightarrow F_X$: (abbrev. upn) $upn(nu, v)$ met à jour la fonction nu en supprimant toutes les références aux nœuds déconstruits de T jusqu'au nœud v (conservé).
- $initialize\_tree\_pred\_and\_first : 2^T, F_{pred}, F_{first}, 2^T \rightarrow 2^T, F_{pred}, F_{first},$: (abbrev. $itpf$) $(T', pd', fst') = itpf((T, pd, fst), U)$ (où $U = cc(v)$ est un ensemble de nouveaux enfants du nœud v) met à jour l'arbre T et les fonctions pd et fst en ajoutant à l'arbre T les nœuds U (T'), associant à

chaque nœud supplémentaire la prédication correspondante résultant de la clause choisie au nœud parent (pd') et en indiquant que les nœuds U n'ont pas encore été visités (fst').
- $update : 2^T, F_{first}, T \rightarrow 2^T, F_{first}$ (abbrev. $updt$) $(T', fst') = updt((T, fst), v)$ où T' est l'arbre T dans lequel on enlève les sous-arbres posterieurs à v sauf leur racine dont l'indicateur de première visite est remis à $true$. Formellement, on supprime les nœuds y tels que $\{y > v \wedge anc(y) > v\}$ et on modifie fst pour les nœuds y tels que $\{y > v \wedge anc(y) < v\}$.

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 description.
- $unify : T \rightarrow Substitution$: (abbrev. $unif$) $unif(v, \mu) : \mu$ est la substitution obtenue par unification de $\sigma(v)pred(v)$ avec la tête de $cc(v)$, la clause choisie. L'échec de l'unification se traduit par un unificateur "échec" $\perp$.
- $claus\_init : T \rightarrow list\_of\_clauses$: (abbrev. $cini$) $lc = cini(v)$ met à jour la fonction $claus$ avec la paire (v, lc) où lc est une liste des clauses renommées, utiles pour résoudre $\sigma pred(v)$ et qui sont donc utilisables pour essayer différentes alternatives pour la résolution (si la liste est vide il n'y a pas de solution). Selon les clauses mises dans la liste, on obtient différents modèles d'exécution.
- $choice : list\_of\_clauses \rightarrow clause : c = choice(lc)$ choisit une clause dans la liste (en Prolog ISO la première de la liste), mais, selon le choix opéré, on obtient différents modèles d'exécution.
- $m1, m2, m3 : Bool$: booléens exclusifs utilisés pour distinguer les modèle de trace souhaité.

Etats initiaux considérés :

$S_1 :$ $\{\{\epsilon\}, \epsilon, 1, \{(\epsilon, 1)\}, \{(\epsilon, goal\}, \{\}, \{(\epsilon, cl_{goal})\}, \{(\epsilon, \emptyset)\}, \{(\epsilon, true)\},$
$false, false, false, false/true\}$

La figure 16 montre l'algèbre des ports avec la SO des 3 modèles .

$$\text{Leaf reached}\ \frac{\neg flr \wedge fst(u) \wedge lf(u) \wedge \neg ct}{\frac{cl'(u)\leftarrow cl,\quad cc(u)\leftarrow\emptyset,\quad n'\leftarrow n+1,\quad num'(u)\leftarrow n',}{fst'(u)\leftarrow F,\quad scs'\leftarrow F,\quad flr'\leftarrow F}}\ \{cl = cini(u)\}$$

$$\text{Claus choice}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge cc(u) = \emptyset \wedge \neg cl(u) = []}{cc'(u) \leftarrow c,\quad cl'(u) \leftarrow cl - c}\ \{c = choice(cl(u))\}$$

$$\text{Fact succeeds}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge \mu \neq \bot \wedge cc(u) \neq \emptyset \wedge ft(cc(u))}{\sigma'(u) \leftarrow \mu\sigma(u),\quad scs' \leftarrow T,\quad flr' \leftarrow F}\ \{unif(u,\mu)\}$$

$$\text{Cl suc}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge \mu \neq \bot \wedge cc(u) \neq \emptyset \wedge \neg ft(cc(u)) \wedge U \leftarrow crc(u)}{itpf((T, pd, fst), U),\quad u' \leftarrow fi(U),\quad \sigma'(u) \leftarrow \mu\sigma(u)}\ \{unif(u,\mu)\}$$

$$\text{Tree success}\ \frac{\neg fst(u) \wedge scs \wedge \neg hnn(u) \wedge \neg ct,\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T)}\ \{\}$$

$$\text{Tree suc \& go right}\ \frac{\neg fst(u) \wedge scs \wedge hnn(u) \wedge \neg ct,\quad v \leftarrow nrn(u)}{u' \leftarrow v}\ \{\}$$

$$\text{Fail no claus}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge cc(u) = \emptyset,\quad cl(u) = [],\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T),\quad flr' \leftarrow T,\quad scs' \leftarrow F,\quad bk3 \leftarrow T}\ \{\}$$

$$\text{Fail not unify 12}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge cc(u) \neq \emptyset \wedge \mu = \bot,\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T),\quad flr' \leftarrow T,\quad scs' \leftarrow F}\ \{m1 \vee m2,\ unif(u,\mu)\}$$

$$\text{Fail not unify 3}\ \frac{\neg fst(u) \wedge lf(u) \wedge \neg ct \wedge cc(u) \neq \emptyset \wedge \mu = \bot}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T),\quad flr' \leftarrow T,\quad scs' \leftarrow F,\quad bk3 \leftarrow T}\ \{m3,\ unif(u,\mu)\}$$

$$\text{Tree failed 12a}\ \frac{\neg fst(u) \wedge \neg lf(u) \wedge flr \wedge \neg ct \wedge \neg hcp(u),\quad v \leftarrow pt(u)}{u' \leftarrow v,\quad (u = \epsilon) \Rightarrow (ct' \leftarrow T)}\ \{m1 \vee m2\}$$

$$\text{Tree failed 2b}\ \frac{\neg fst(u) \wedge \neg lf(u) \wedge flr \wedge \neg ct \wedge hcp(u),\quad v \leftarrow ycwcp(u)}{u' \leftarrow v}\ \{m2\}$$

$$\text{Bkt1}\ \frac{\neg fst(u) \wedge hcp(u) \wedge (flr \vee ct),\quad v \leftarrow gcp(u)}{updt((T, fst), v),\quad u' \leftarrow v,\quad cc(v) \leftarrow \emptyset,\quad ct \Rightarrow (ct' \leftarrow F),\quad scs' \leftarrow F,\quad flr' \leftarrow F}\ \{m1\}$$

$$\text{Bkt2a}\ \frac{\neg fst(u) \wedge hcp(u) \wedge (flr \vee ct),\quad gcp(u) \neq u}{u' \leftarrow cwgcp(u)}\ \{m2\}$$

$$\text{Bkt2b}\ \frac{\neg fst(u) \wedge hcp(u) \wedge (flr \vee ct),\quad u = gcp(u)}{updt((T, fst), u),\quad cc(u) \leftarrow \emptyset,\quad scs' \leftarrow F,\quad flr' \leftarrow F,\quad ct \Rightarrow (ct' \leftarrow F)}\ \{m2\}$$

$$\text{Bkt3a}\ \frac{\neg fst(u) \wedge bk3 \wedge cl(u) \neq [] \wedge (flr \vee ct)}{updt((T, num, fst), u),\quad cc(u) \leftarrow \emptyset,\quad scs' \leftarrow F,\quad flr' \leftarrow F,\quad bk3 \leftarrow F,\quad ct \Rightarrow (ct' \leftarrow F)}\ \{m3\}$$

$$\text{Bkt3b}\ \frac{\neg fst(u) \wedge bk3 \wedge \neg lf(u) \wedge cl(u) = []}{u' \leftarrow rcld(u)}\ \{m3\}$$

$$\text{Bkt3c}\ \frac{\neg fst(u) \wedge bk3 \wedge lf(u) \wedge hpn(u) \wedge cl(u) = []}{u' \leftarrow pn(u)}\ \{m3\}$$

$$\text{Bkt3d}\ \frac{\neg fst(u) \wedge bk3 \wedge lf(u) \wedge \neg hpn(u) \wedge cl(u) = [] \wedge u \neq \epsilon}{u' \leftarrow pt(u)}\ \{m3\}$$

Fig. 15. Sémantique observationnelle d'une résolution SLDT déterministe avec 3 méthodes de retour arrière

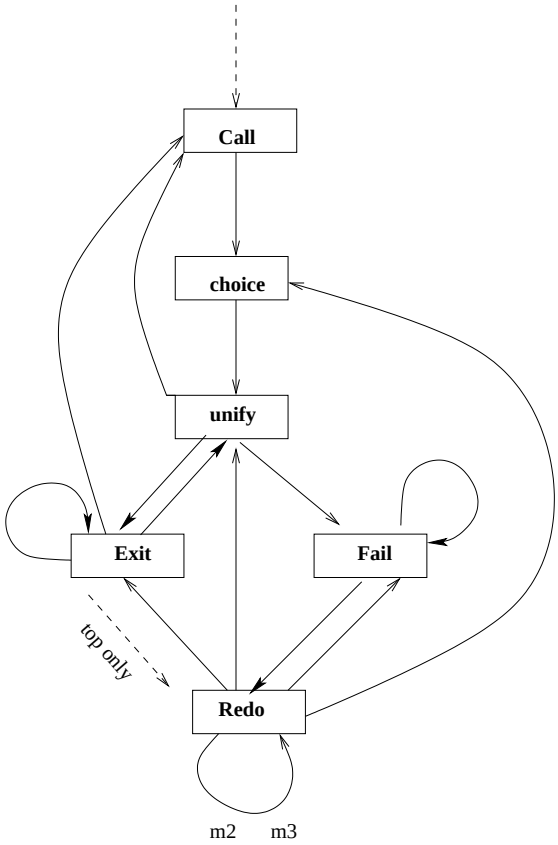


Fig. 16. Algèbre des ports avec la SO des 3 modèles (sauf indication explicite, une transition est possible dans tous les modèles)

ANNEXE F : Preuve de correction de la condition d'adéquation (Section 2, Proposition 1

On prouve que si la propriété suivante est vérifiée :

[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 $r \in R$:

$\forall\, e,\, e',\, r', S,\, S',\, S'',$

$\mathcal{E}_r(S, S') = e \ \wedge\ \mathcal{E}_{r'}(S', S'') = e'$

(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 toute trace actuelle $T_w =< Q_0, w_t^+ >$, définie par le schéma de trace $\mathcal{E}$ et telle que $Q_0 = S_0/Q$, est *adéquate* pour Q par rapport à la trace intégrale virtuelle $T_v =< S_0, v_t^+ >$;

c'est à dire qu'il existe une fonctions $\mathcal{F}$ telle que

$\forall t \geq 0,\ \mathcal{F}(w_t^+, Q_0) = Q_t$ et

$\forall i \in [0..t],\ Q_i = S_i/Q \ \wedge \exists r_i \in R,$ tel que $w_i = \mathcal{E}_{r_i}(S_i, S_{i+1})$.

Cet ennoncé appelle un commentaire afin de bien comprendre ce que signifie l'adéquation. L'existence d'une fonction $\mathcal{F}$ assure que la lecture de la trace permet de suivre l'évolution d'un état restreint aux paramètres de Q, un sous-ensemble de l'état intégral virtuel S. Ceci n'a de sens que si on part d'un état initial Q_0 connu qui ne peut être que l'état intégral virtuel retreint initial S_0/Q (donc $Q_0 = S_0/Q$).

Par ailleurs on notera que la fonction $\mathcal{F}$ ne préjuge pas de l'état S_{t+1} atteint après l'événement w_t. Si la trace est infinie, cela ne pose pas problème. Si par contre la trace est finie et si aucun état défini comme final n'est atteint ou si la trace est brusquement interrompue (ceci suppose des interactions externes au processus observé non étudiées ici), on ne connaît rien en général sur l'état atteint. Ceci n'affaiblit en rien l'idée d'adéquation mais montre seulement que tout "redémarrage" de la trace nécessite de fournir un nouvel état de départ, c'est à dire de fournir à nouveau un état initial.

L'adéquation assure donc deux choses : la première est que chaque événement de trace amène à un état Q toujours égal à l'état intégral virtuel restreint courant S/Q (ce qui est assuré par le fait que $S_i/Q = Q_i$).

L'adéquation assure également, en liaison avec la propriété précédente, qu'à chaque événement w_i de la trace actuelle une règle $< r, S, S' >$ de la SO s'applique de telle manière que $< r_i, S_i, S_{i+1} >$ en est une instance. Ceci assure que la trace peut "faire tourner" la machine définie par la SO de telle manière que la trace actuelle est bien susceptible d'avoir été extraite à partir de celle-ci. Noter que l'adéquation n'exige pas que la SO soit décrite par un automate déterministe ; la même trace actuelle peut donc être produite de différentes manières (l'unicité de la règle r_i qui s'applique n'est pas requise). La condition proposée ici impose

cette unicité, c'est à dire qu'à une trace actuelle correspond un seul fonctionnement possible de l'automate. Il s'agit donc seulement d'une condition suffisante d'adéquation. Son intérêt cependant est d'assurer qu'à toute trace actuelle correspond une lecture unique dans la SO (ceci est en effet trivialement le cas si dans la trace le nom de la règle de transition fait partie des attributs ou si $Cond_r(e, e')$ ne dépend que de e, ce qui se vérifie si les ports sont isomorphes aux noms des règles).

Démonstration. Rappel : w_t^+ représente la suite finie non vide $w_t \dots w_1 w_0$ et w_t^* est w_t^+ ou la suite vide ϵ (la suite vide n'est possible que pour $t \leq 0$). Cette formulation de l'adéquation suppose que les traces actuelles aient au moins deux événements ($w_t^+, t \geq 1$ débute toujours avec les événements w_0 et w_1).

On définit alors la valeur de $\mathcal{F}(w_t^*, Q_0)$ récursivement, à partir de la fonction de reconstruction locale, de la manière suivante :

$\mathcal{F}(\epsilon, Q_0) = Q_0$;
$\mathcal{F}(w_0, Q_0) = Q_0$; (l'état actuel courant obtenu ne peut être connu qu'avec au moins deux événements de trace).
et pour $t \geq 1$, $\mathcal{F}(w_t w_{t-1} w_{t-2}^*, Q_0) =$
si $Cond_r(w_{t-1}, w_t)$ alors $\mathcal{C}_r(w_{t-1}, w_t, \mathcal{F}(w_{t-1}^*, Q_0))$.

La définition de $\mathcal{F}$ est donc la suivante :

$\mathcal{F}(\epsilon, S) = S$;
$\mathcal{F}(e, S) = S$;
$\mathcal{F}(e'eE, S) =$ si $Cond_r(e, e'))$ alors $\mathcal{C}_r(e, e', \mathcal{F}(eE, S))$
avec E suite quelconque d'événements de trace.

Notons que selon la condition 1 cette fonction est déterministe.

Si on considère alors un état initial S_0, un état initial restreint $Q_0 = S_0/Q_0$, et une trace actuelle $T_w =< Q_0, w_t^+ >$, définie par le schéma de trace $\mathcal{E}$, la condition d'adéquation assure que cette trace est adéquate pour Q par rapport à la trace virtuelle intégrale $T_v =< S_0, v_t^+ >$ (rappel :T_v est obtenue par application de fonctions d'extraction $\mathcal{E}_r(S, S')$ telles que les valeurs des attributs de l'événement de trace correspondant sont $A_t = S_{t+1}$).

En effet, par hypothèse, l'état initial restreint est $Q_0 = S_0/Q_0$, et du fait de la condition 2 et de la définition de $\mathcal{F}$, les états restreints successifs calculés avec la fonction $\mathcal{F}$ ainsi définie vérifient bien $\forall i \in [1..t]$, $Q_i = S_i/Q$. A chaque étape également, par la condition 1, une seule condition $Cond_{r_{i-1}}(w_{i-1}, w_i)$ est vérifiée et une seule règle r_{i-1} de reconstruction s'applique aux deux événements w_{i-1} et w_i : $\mathcal{C}_{r_{i-1}}(w_{i-1}, w_i, S_{i-1}/Q)$. Le premier événement w_{i-1} a été produit par application de la fonction d'extraction aux états virtuels S_{i-1} et S_i, $\mathcal{E}_{r_{i-1}}(S_{i-1}, S_i) = w_{i-1}$, et le second événement w_i, par application de la fonction d'extraction aux états virtuels S_i et S_{i+1} pour une certaine règle r_i avec $\mathcal{E}_{r_i}(S_i, S_{i+1}) = w_i$.

Comme les événements successifs de la trace actuelle sont produits par application du schéma de trace $\mathcal{E}$ et que chaque couple d'évènements permet de

reconstruire un état virtuel restreint à Q par la fonction $\mathcal{F}$ déduite du schéma de reconstruction, l'ensemble des propriétés caractérisant l'adéquation sont bien vérifiés.

De plus, à chaque événement de trace actuelle w_i ($i \in [0..t-1]$) correspond une transition $< r_i, S_i, S_{i+1} >$ qui produit l'événement de trace intégrale virtuelle v_i dont les attributs vérifient $A_i/Q = Q_{i+1}$. On observe ainsi que la trace actuelle, bien que ne donnant qu'une vue partielle du processus observé, permet d'en reconstituer le déroulement complet, à condition toutefois d'en connaître la SO.