Proposition de format concret pour des traces générées par des solveurs de contraintes Réalisation RNTL OADYMPPAC 2.2.2.1

Romuald Debruyune Jean-Daniel Fekete Narendra Jussien
Mohammad Ghoniem
École des Mines de Nantes

20 novembre 2001

Résumé

Ce document décrit un format concret permettant de représenter les traces produites par un solveur de contraintes. Les traces sont codées au format XML en utilisant un type de document (DTD) spécifique.

Ce format permet des adaptations aux divers types de solveurs. Ces adaptations prennent en compte les spécificités des solveurs ainsi que le niveau de détails requis par les programmes exploitant la trace.

Finalement, nous décrivons un mécanisme permettant de configurer la richesse de la trace afin de ne pas surcharger le solveur et d'alléger la trace si l'application qui l'utilise n'a pas besoin de tous les détails.

Table des matières

Contributions

Ce document a été rédigé avec la collaboration de :

Cosytec S.A. Abderrahmane. Aggoun, Raphaël Martin

ILOG S.A. Thomas Baudel

INRIA, Rocquencourt Pierre Deransart, Ludovic Langevine, Francois Fages

INSA, Rennes Mireille Ducassé

LIFO, Orléans Alexandre Tessier, AbdelAli Ed-Dbali, Willy Lesaint.

Chapitre 1

Introduction

L'un des objectifs du projet OADymPPaC est d'expérimenter la réalisation d'outils de visualisation facilement adaptables sur différentes plateformes de solveurs de contraintes. Afin de parvenir à un tel objectif, différentes couches de logiciels sont introduites, illustrées sur la figure 1. Le solveur produit une trace[1], ainsi qu'il est défini dans les délivrables D1.2.1, (mécanismes géneriques d'extraction de trace) et D1.2.2.1, (langage d'analyse de traces), élaborées en parallèle. La trace produite concerne de très grandes quantités d'évènements qu'il n'est pas question de transmettre intégralement aux outils potentiels. Seule une partie de cette trace est communiquée, grâce à un filtre (boîte « filtrage »). La partie pertinente à transmettre afin que les outils des partenaires puissent l'utiliser sera déterminée ultérieurement.

A l'autre bout de la chaîne se trouvent les outils de visualisation et de mise au point qui utilisent seulement une partie des informations transmises. Chaque outil réalise lui-même son propre filtrage (boîte « Adap. Visu ») et produit les informations sous la forme attendue.

Cette approche permet une conception relativement indépendante du solveur des outils de débogage et permet de prévoir de faire fonctionner plusieurs outils simultanément sur une plateforme donnée. Le flot de données transmis entre le module de filtrage et celui d'adaptation de l'outil de visualisation est codé au format XML.

Ce document décrit ce format qui doit permettre de représenter toutes les traces qu'il est possible de produire par un solveur de contraintes.

Ce format permet des adaptations aux divers types de solveurs. Cette adaptation prend en compte les spécificités des solveurs ainsi que le niveau de détails requis par les programmes exploitant la trace. Il peut être produit directement à partir des spécifications contenues dans ce document.

Ce document ne décrit pas la syntaxe ni la sémantique de XML. Plusieurs ouvrages peuvent être consultés pour cela, ainsi que les spécifications de XML, disponibles à l'URL suivant :

`http://www.w3.org/TR/REC-xml`.

La trace n'est pas destinée à terminer sur un fichier texte mais à être envoyée à une application d'analyse ou de visualisation. Un mécanisme de connexion réseau devra être défini pour faciliter cette connexion avec des mécanismes standard de type réseau

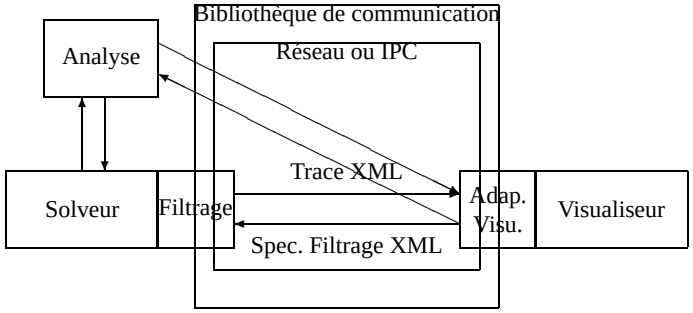

FIG. 1.1 – Modèle architectural de la communication entre solveur et visualiseur

ou communication inter processus, comme le décrit le diagramme de la figure 1.1. Un autre document décrira ces mécanismes quand ils auront été expérimentés et validés par les partenaires du projet OADYMPPAC.

Finalement, les techniques de filtrage de la trace dans les modules d'adaptation des outils de visualisation seront décrites dans un document ultérieur.

1.1 Concepts nouveaux

Le modèle de trace introduit des événements primitifs du moteur de résolution de contraintes. Outre les événements liés à la trace proprement dite, nous avons pensé utile d'ajouter dans la trace concrète des informations de plus haut niveau utiles pour annoter cette trace de manière relativement précise et si possible formalisable. Le format de la trace a besoin de tous les événements du modèle plus trois que nous décrivons ici : l'*annotation*, l'*étape* et le *contexte*.

1.1.1 Annotations

Dans la majorité des applications traitant des problèmes complexes d'ordonnancement ou de planification, on travaille sur des entités de plus haut niveau comme des tâches définies par des attributs comme : le début de la tâche dans le temps, sur quelle machine la tâche sera affectée, la durée d'occupation de la machine, etc. Pour prendre en compte par exemple, la notion de tâche, il est nécessaire d'étendre la trace se basant uniquement sur l'observation des variables. Pour ce faire, nous introduisons la notion d'annotations. Une annotation peut faire intervenir une ou plusieurs variables. Une annotation peut être vue comme un groupe (organisé) de variables définissant une entité du problème à traiter. Une *annotation* est caractérisée par :

- Un identifiant,
- Un type,
- Un libellé,
- Une liste d'identificateurs de variables ou d'autres annotations.

Une méta contrainte, par exemple une contrainte sémantique, sur des tâches en disjonction peut être vue comme une contrainte où les éléments impliqués ne sont plus simplement des variables domaines mais des objets sémantiques représentés par des termes de type *annotation*.

Dans une application faisant intervenir des contraintes sémantiques, quand le domaine d'une des variables d'une tâche est modifiée, l'utilisateur est plus intéressé par la trace de la tâche dans son ensemble que par la modification des domaines des variables individuelles. L'idée est d'enrichir le modèle de trace défini jusqu'à présent en introduisant des entités de type *annotation*. Ainsi la trace est généralisée à des termes de haut niveau et non plus simplement aux seules variables du problème.

Exemple

un tâche est un terme de la forme [Début, Durée, Machine]. La trace générée comportera le terme et les numéros des variables.

Tâche : [Début : :[5..100], Durée : : [4], Machine : : [1,5]] [1, 3]

[1, 3] signifie : 1 est le rang de variable début dans le groupe, 3 est le rang de variable machine dans le groupe. Ces deux variables ont été modifiées pendant le réveil d'une contrainte globale (sémantique).

Voici un programme pour illustrer cette idee

```
top:-
% General Parameters
    End = 100,  NbM1 = 1, NbM2 = 1,
% DEFINING VARIABLES
    [S11,S12,S21,S22,S23,S31,S41] :: 0..End,        % Starts of tasks
    [E1,E2,E3,E4] :: 0..End,                        % Ends of jobs
    [D11,D12,D21,D22,D23,D31,D41] = [5,7,3,5,4,5,7],% Duration of tasks
    RM1 :: 1..NbM1, RM2 :: 1..NbM2,                 % Limit of Machine use
% SETTING UP CONSTRAINTS
    % contraintes syntaxiques : contraintes de précédences
    S12 #>= S11+D11,                                % T12 after T11
    S22 #>= S21+D21,                                % T22 after T21
    S23 #>= S22+D22,                                % T23 after T22
    % Defining Ends of Jobs
    E1 #= S12+D12,                          % End of Job 1
    E2 #= S23+D23,                          % End of Job 2
    E3 #= S31+D31,                          % End of Job 3
    E4 #= S41+D41,                          % End of Job 4
   % contraintes sémantiques : tâches disjonctives par machines
    % définitions des tâches : annotations
    T1 = [S11, D11],
    T2 = [S22, D22],
    T3 = [S41, D41],
    disjunctive([T1, T2, T3]),

    T4 = [S12, D12],
    T5 = [S21, D21],
    T6 = [S23, D23],
```

```
    T7 = [S31, D31],
    disjunctive([T4,T5,T6,T7]),

% LABELING
   % Finding the optimal solution
    min_max(labeling([T1, T2, T3, T4,T5,T6,T7],
                     1,
                     smallest,
                     assign),
            [E1,E2,E3,E4]). % Minimise the max of ends of jobs


assign([S, D]) :-
       indomain(S).
```

La trace produite pour ce programme au niveau de la propagation des contraintes sémantiques de disjonctions sera de la forme :

```
<new-variable vname="S11"><range from="0" to="100"/></new-variable>
...
<new-annotation id="T1" type="task" refs="S11 D11"/>
<new-annotation id="T2" type="task" refs="S22 D22"/>
<new-annotation id="T3" type="task" refs="S41 D41"/>
<new-annotation id="M1" type="machine" refs="T1 T2 T3"/>

<new-annotation id="T4" type="task" refs="S12 D12"/>
<new-annotation id="T5" type="task" refs="S21 D21"/>
<new-annotation id="T6" type="task" refs="S23 D23"/>
<new-annotation id="T7" type="task" refs="S31 D31"/>
<new-annotation id="M2" type="machine" refs="T4 T5 T6 T7"/>

<reduce ...>
  <update vname="S11" type="min"><range from="10" to="50"/></update>
</reduce>
<reduce ...>
  <update vname="D12" type="min"><range from=20" to="30"/></update>
</reduce>
```

Si la visualisation reconnait le type de l'annotation, l'utilisateur devrait voir non plus des évènements sur des variables et des états de domaines mais une réelle représentation d'un planning par exemple sous forme d'un diagramme de Gantt comme celui de la figure 1.2.

Cette exemple illustre l'utilité des objets sémantiques *annotation* qui doivent être impérativement définis pour obtenir une représentation de la trace suffisamment agrégée pour être réellement utile au débogage des solveurs (notamment d'applications industrielles complexes).

1.1.2 Étape

La plupart des utilisations de solveurs s'inscrivent dans un système de plus grande ampleur organisé en différentes étapes qui peuvent être elles-mêmes imbriquées. On

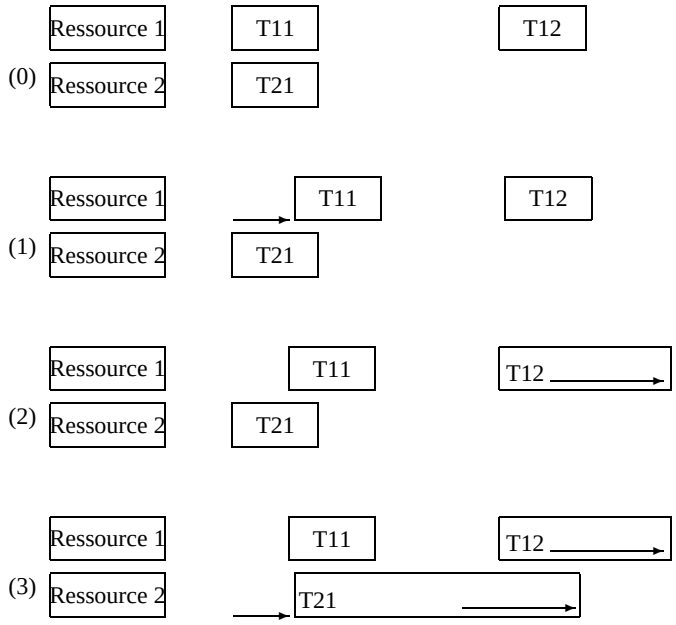

FIG. 1.2 – Diagramme de Gantt

peut alors souhaiter faire ressortir dans la trace la portée d'une procédure, l'exécution d'un prédicat en CLP, l'usage d'un algorithme particulier ou des phases particulières de l'activité d'un solveur par exemple. Les étapes telles que la pose initiale de contraintes et l'énumération sont des exemples remarquables d'étapes que l'on peut souhaiter signaler.

Afin de permettre la délimitation des étapes, cinq évènements sont définis : `<stage>`, `<start-stage>`, `<suspend-stage>`, `<resume-stage>` et `<end-stage>`. Pour le second type, l'attribut `activities` peut être associé à tous les événements.

Nous avons défini plusieurs types d'éléments XML car nous pensons que les étapes des solveurs peuvent être entrelacées tandis que les éléments XML ne doivent pas se chevaucher. Chaque étape doit être déclarée avant d'être utilisée. L'événement `<new-stage>` permet cette déclaration ainsi qu'une description en langage naturel du sens de l'étape.

Ce concept d'étape est nouveau et nous avons besoin d'expériences pour le décrire plus précisément et être sûr que les mécanismes proposés sont suffisants.

1.1.3 Contexte

Les contraintes manipulées par le solveur peuvent être issues d'un système de plus haut niveau. Le contexte permet de maintenir la forme originelle dans laquelle l'utilisateur les comprendra. Ce contexte est différent de la forme externe d'une contrainte qui peut être exprimée dans le langage de contraintes natif tandis que le contexte référencera le langage de plus haut niveau.

Chapitre 2

Format de la trace

Un document de trace suit les conventions lexicales et syntaxiques de XML et de la DTD nommée `oadymppac.dtd` dont l'emplacement publique sera `http://contraintes.inria.fr/OADymPPaC/oadymppac.dtd`. Il commence par l'élément XML `<oadymppac>` suivi d'une suite d'événements représentés pas des éléments XML.

Pour chaque élément, nous décrivons sa fonction, ses attributs, son contenu et sa déclaration XML. La DTD est conçue pour utiliser les mécanismes XML autant que possible. Par exemple, les noms des contraintes sont des identificateurs XML de manière à ce qu'un processeur XML validant puisse vérifier l'existence et l'unicité de ces noms. Le pouvoir d'expression des DTD XML étant très limité, nous avons aussi défini des entités (des macros) pour exprimer les types des attributs. Ces types ne peuvent être vérifiés par un processeur standard mais sont lisibles dans la DTD. XML dispose maintenant de Schémas qui décrivent plus précisément les types des éléments et des attributs. Ce format XML a donc vocation a être traduit en Schéma.

2.1 Règles lexicales

Une trace générée par un solveur devient un document XML valide [4]. Il suit donc les conventions lexicales XML qui ne sont pas toujours les mêmes que les conventions lexicales des solveurs de contraintes. Par exemple, la forme externe d'une contrainte Prolog peut être `X#<Y` qui utilise le caractère `<` réservé en XML. La convention XML est alors d'utiliser une *entité* standard XML pour que la forme externe devienne `X#&lt;Y`. Pour simplifier ce genre de traitement, une interface de programmation en C faisant cette traduction sera disponible.

Une seconde source de problèmes peut venir des identificateurs uniques utilisés pour nommer et référencer les variables et les contraintes. En XML, ces identificateurs doivent commencer par un caractère alphabétique ou le signe souligné (_) et sont limités en longueur à 31 caractères.

Pour le reste, XML est généralement plus libéral que les solveurs de contraintes actuels : il autorise des caractères étendus Unicode [3] et ne définit que cinq caractères

réservés : &, <, >, " et ' nommés respectivement &, <, >, " et '.

Enfin, pour les caractères accentués, XML utilise un codage sur 7 bits nommé UTF-8 [5] qui équivaut à l'US-ASCII lorsqu'aucun accent n'est utilisé. Dans le cas contraire, des séquences de 2, 3, 4 ou 5 caractères peuvent être nécessaires pour coder l'ensemble du registre de caractères Unicode. La trace décrite ici est codée en UTF-8 et les solveurs désirant placer des caractères accentués ou internationaux dans la trace doivent suivre cette norme.

2.2 Événements

Les événements actuellement définis sont :

new-variable déclare une nouvelle variable ;

new-constraint déclare une nouvelle contrainte ;

new-stage déclare une nouvelle étape ;

new-annotation déclare une nouvelle annotation ;

reduce signale une réduction de domaine ;

restore signale la restauration d'un domaine ;

wake-up signale le réveil d'une contrainte ;

suspend signale la mise en attente d'une contrainte ;

true signale qu'une contrainte est vérifiée ;

reject signale qu'une contrainte n'est pas vérifiée ;

select-constraint indique la contrainte sélectionnée pour propagation ;

select-update indique l'update sélectionné pour propagation ;

activate signale qu'une contrainte est active ;

tell met en place un point de choix ;

deactivate signale qu'une contrainte n'est plus active ;

solution signale que le state est solution du système de contraintes ;

stage encadre une nouvelle instance d'étape ;

start-stage signale le début d'une nouvelle étape ;

suspend-stage signale qu'une étape en cours est suspendue ;

resume-stage signale qu'une étape suspendue est redémarrée ;

end-stage signale la fin d'une étape en cours.

2.3 Attributs généraux

Chaque élément peut contenir des attributs et un contenu. Il existe des attributs génériques et d'autres spécifiques. Trois macros expressions (des entités paramètres dans le jargon XML) sont utilisées pour rendre la DTD plus lisible : %constraintAttributes;, %eventAttributes; et %commonAttributes;.

Les attributs pouvant être spécifiés par %constraintAttributes; sont :

cname (opt) référant unique de la contrainte ayant produit cet événement ;

externalrep (opt) représentation externe de la contrainte ;

internalrep (opt) représentation interne de la contrainte.

Les attributs pouvant être spécifiés par %eventAttributes; sont :

n (opt) numéro de l'événement ;

time (opt) temps au moment où l'événement a été émis, en millisecondes depuis le début du programme ;

depth (opt) nombre d'événements "tell" non terminés par un "restore" qui précèdent dans la trace ;

context (opt) description du contexte dans lequel l'événement s'est produit (optionnel) ;

Finalement, %commonAttributes; est juste la concaténation des deux précédents.

Pour rendre la DTD plus claire, l'entité %integer; est utilisée pour déclarer des attributs devoir avoir une valeur entière.

2.3.1 Déclaration XML des attributs généraux constraintAttributes, eventAttributes, commonAttributes et integer

```
<!ENTITY % integer "CDATA" >

<!ENTITY % constraintAttributes
  "cname IDREF #REQUIRED
   externalrep CDATA #IMPLIED
   internalrep CDATA #IMPLIED" >

<!ENTITY % eventAttributes
  "depth %integer; #IMPLIED
   n %integer; #IMPLIED
   time %integer; #IMPLIED
   context CDATA #IMPLIED
   activities IDREFS #IMPLIED" >

<!ENTITY % commonAttributes
  "%eventAttributes;
   %constraintAttributes;" >
```

2.4 new-variable

Lorsqu'une nouvelle variable est référencée par une contrainte, elle doit préalablement être déclarée par un <new-variable>. Cette déclaration permet à des analyseurs de la trace de connaître les variables utilisées par un solveur sans avoir besoin de connaître la syntaxe du langage de programmation. Il s'agit donc d'un événement principalement syntaxique mais qui introduit un référant unique d'une variable, référant ensuite utilisé par chaque détails de la trace en relation avec la variable.

Un élément <new-variable> introduit donc une nouvelle variable. Les attributs requis, en plus de ceux déclarés par %eventAttributes;, sont :

vname (req) le référant unique de la variable, sous la forme d'un identificateur XML ;

type (req implicite) le type de la variable qui peut être int pour un entier (par défaut), rat pour un rationnel, real pour un réel, enum pour une énumération de valeurs symboliques ou string pour des chaînes de caractères. *Seules les valeurs entières sont correctement gérées actuellement.*

externalrep (opt) la représentation externe de la variable.

internalrep (opt) la représentation interne de la variable.

Le contenu de l'élément est une liste de valeurs ou d'intervalles.

2.4.1 Eléments associés : values et range

Les valeurs peuvent être décrites en extension ou par intervalle. Tous les éléments décrivant des valeurs utilisent l'entité %valueList; qui autorise une union de valeurs en extensions et d'intervalles.

2.4.2 Déclaration XML des éléments new-variable, values et range

```
<!ENTITY % valueList "( values | range )*" >

<!ELEMENT new-variable ( values | range )* >
<!ATTLIST new-variable
  %eventAttributes;
  vname ID #REQUIRED
  externalrep CDATA #IMPLIED
  internalrep CDATA #IMPLIED >

<!ELEMENT values (#PCDATA) >

<!ELEMENT range EMPTY>
<!ATTLIST range
  from CDATA #REQUIRED
  to CDATA #REQUIRED >
```

2.4.3 Exemple

```
<new-variable n="1" vname="x">
 <values>1 2 3</values>
</new-variable>
```

qui peut aussi s'écrire

```
<new-variable n="1" vname="x">
 <range from='1' to='3'/>
</new-variable>
```

2.5 new-constraint

Lorsqu'une nouvelle contrainte est ajoutée au système de contraintes géré par le solveur, elle est déclarée dans la trace avec l'événement `<new-constraint>`. Cette déclaration est destinée à des analyseurs de la trace pour connaître les contraintes utilisées par le solveur sans avoir besoin de connaître la syntaxe du langage de programmation.

Un élément `<new-constraint>` introduit donc une nouvelle contrainte. Les attributs requis, en plus de ceux déclarés par %eventAttributes;, sont :

`cname` (req) le référant unique de la contrainte, sous la forme d'un identificateur XML ;

`orig` (opt) spécifie si la contrainte a été ajoutée par un programme utilisateur ou par le système. Les valeurs autorisées sont user et system.

`externalrep` (opt) la représentation externe de la contrainte ;

`internalrep` (opt) la représentation interne de la contrainte.

Le contenu de l'élément est une liste de conditions d'activations de la contrainte décrite par l'élément `<update>` en section 2.7.

2.5.1 Déclaration XML de l'élément new-constraint

Les attributs n'utilisent pas l'entity %constraintAttributes; mais sont spécifiés en extension car l'attribut `cname` est requis et de type ID.

```
<!ELEMENT new-constraint (update)* >
<!ATTLIST new-constraint
  %eventAttributes;
  cname ID #REQUIRED
  orig ( user | system ) #IMPLIED
  externalrep CDATA #IMPLIED
  internalrep CDATA #IMPLIED >
```

2.5.2 Exemple

```
<new-constraint n="7" cname="c2" externalrep="X > Y">
  <update vname="X" type="max" />
  <update vname="Y" type="min" />
</new-constraint>
```

2.6 new-annotation

L'élément `<new-annotation>` désigne des objets sémantiques de natures diverses. Une annotation peut par exemple désigner les différentes phases du solveur ou des allocations de ressources etc. On peut imaginer un grand nombre d'annotations possibles dont il serait intéressant à terme d'extraire un sous-ensemble standard. La

reconnaissance par les outils de visualisation de certaines annotations permettra alors d'afficher des objets de haut niveau (e.g. des tâches, des machines etc.) et construire des représentations ad hoc.

2.6.1 Déclaration XML de l'élément new-annotation

```
<!ELEMENT new-annotation (#PCDATA) >
<!ATTLIST new-annotation
  id ID #REQUIRED
  type ID #IMPLIED
  name CDATA #IMPLIED
  refs IDREFS #IMPLIED >
```

2.6.2 Exemple

```
<new-annotation type="task" id="T1" name="Tache 1" refs="S11 D11"/>
<new-annotation type="machine" id="M1" name="Machine 1" refs="T1 T2 T3"/>
```

2.7 Les éléments update et cause

L'élément `<update>` décrit une des variables modifiée par un événement. L'élément `<cause>` décrit une des variables ayant provoqué un événement. Les attributs sont :

`cname` (opt) le référant unique de la contrainte ;

`vname` (req) le référant unique de la variable ;

`type` (opt) la raison de l'activation, parmi les mots réservés suivants : ground, any, min, max, minmax, empty, nothing et val.

Le contenu de l'élément est une liste de valeurs, décrites de façon isolée ou par intervalle.

2.7.1 Déclaration XML des éléments update et cause

```
<!ELEMENT update %valueList; >
<!ATTLIST update
  cname IDREF #IMPLIED
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax
        | empty | val | nothing) #IMPLIED >

<!ELEMENT cause %valueList; >
<!ATTLIST cause
  cname IDREF #IMPLIED
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax
        | empty | val) #IMPLIED >
```

2.8 state

L'élément `<state>` décrit une partie de l'état interne du solveur de contraintes. Les attributs requis sont :

`precision` (opt) indique la précision avec laquelle le state est décrit. Les valeurs autorisées sont : empty (aucune précision n'est donnée), changed (seules les changements par rapport au `<state>` précédent sont donnés) et full (tous les détails sont donnés).

L'élément `<state>` peut contenir les éléments suivants dans l'ordre :

`<alist>` l'ensemble des contraintes actives ;

`<slist>` l'ensemble des contraintes suspendues ;

`<qlist>` la file de propagations ;

`<tlist>` l'ensemble des contraintes résolues ;

`<rlist>` l'ensemble des contraintes rejetées ;

`<ulist>` l'ensemble des événements `<update>` actifs (utilisés par des solveurs maintenant une file d'événements plutôt qu'une file de contraintes).

`<vlist>` l'ensemble des variables et de leur domaine ;

`<misc>` des informations specifiques aux solveurs, sous une forme textuelle.

2.8.1 Déclaration XML des éléments state, alist, slist, qlist, tlist, rlist, ulist, vlist, misc et vardomain

L'élément `<vardomain>` décrit le domaine d'une variable avant l'exécution de l'événement. Il n'est pas nécessaire de faire apparaître dans la trace le domaine de toutes les variables, celles modifiées par l'événement ou, au pire, référencées par la contrainte ayant déclenché l'événement, suffisent.

```
<!ELEMENT state (alist?, slist?, qlist?, tlist?, rlist?, vlist?,
  ulist?, (misc)* ) >
<!ATTLIST state
  precision (empty | changed | full) #IMPLIED >

<!ELEMENT alist (constraint)* >
<!ELEMENT slist (constraint)* >
<!ELEMENT qlist (constraint)* >
<!ELEMENT tlist (constraint)* >
<!ELEMENT rlist (constraint)* >
<!ELEMENT vlist (vardomain)* >
<!ELEMENT ulist (update)* >

<!ELEMENT vardomain %valueList; >
<!ATTLIST vardomain
  vname IDREF #REQUIRED >
```

2.9 reduce

L'élément <reduce> signale la réduction du domaine de variables lors de l'activation d'une contrainte. Les attributs requis sont les attributs généraux.

Le contenu de l'élément est un <state> optionnel, une liste de <update> spécifiant les domaines retirés aux variable, et une explication sur les raisons ayant abouti à cette réduction.

2.9.1 Eléments associés : explanation et constraint

L'élément <explanation> est une liste d'éléments <cause> suivi d'une liste d'éléments <constraint> expliquant pourquoi la réduction a été effectuée.

L'élément <constraint> référence simplement une contrainte.

2.9.2 Déclaration XML des éléments reduce, explanation et constraint

```
<!ELEMENT reduce ( state?, (update)*, explanation? ) >
<!ATTLIST reduce
  %commonAttributes; >

<!ELEMENT explanation ((cause)*, (constraint)*) >

<!ELEMENT constraint EMPTY>
<!ATTLIST constraint
  cname IDREF #REQUIRED >
```

2.10 restore

L'élément <restore> signale qu'une ou plusieurs valeurs ont été replacée dans le domaine de variables. C'est le mécanisme inverse de <reduce> qui est utilisé explicitement par des solveurs sans backtracking. Les attributs requis sont les attributs généraux. Cet événement est aussi généré par les solveurs avec backtracking. Dans ce cas, l'attribut depth doit être utilisé pour indiquer le retour vers le point de choix précédent. Les programmes d'analyse de la trace sont alors obligés de maintenir eux-même l'état des domaines avec le <tell>.

Le contenu de l'élément est le même que celui de <reduce> décrit en 2.9.

2.10.1 Déclaration XML de l'élément restore

```
<!ELEMENT restore ( state?, (update)*, cause? ) >
<!ATTLIST restore
  %commonAttributes; >
```

2.11 wake-up

L'élément <wake-up> signale le réveil d'une contrainte parce qu'une de ses conditions d'activation a été rencontrée. Les attributs requis sont les attributs généraux.

Le contenu de l'élément est un <state> optionnel et une liste de <cause>.

L'élément <cause> est une liste d'éléments <update> expliquant pourquoi la contrainte été réveillée.

2.11.1 Déclaration XML de l'élément wake-up

```
<!ELEMENT wake-up ((vardomain)*, state?, cause) >
<!ATTLIST wake-up
  %commonAttributes; >
```

2.12 suspend

L'élément <suspend> signale la mise en attente d'une contrainte. Les attributs requis sont les attributs généraux. Le contenu de l'élément est un <state> optionnel.

2.12.1 Déclaration XML de l'élément suspend

```
<!ELEMENT suspend (state?) >
<!ATTLIST suspend
  %commonAttributes; >
```

2.13 true

L'élément <true> signale qu'une contrainte est vérifiée Les attributs requis sont les attributs généraux. Le contenu de l'élément est un <state> optionnel.

2.13.1 Déclaration XML de l'élément true

```
<!ELEMENT true (state?) >
<!ATTLIST true
  %commonAttributes; >
```

2.14 reject

L'élément <reject> signale qu'une contrainte n'est pas vérifiée. Les attributs requis sont les attributs généraux. Le contenu de l'élément est un <state> optionnel et une <explanation>.

2.14.1 Déclaration XML de l'élément reject

```
<!ELEMENT reject ( state?, explanation?) >
<!ATTLIST reject
  %commonAttributes; >
```

2.15 select-constraint

L'élément <select-constraint> indique la contrainte sélectionnée pour propagation. Les attributs requis sont les attributs généraux. Le contenu de l'élément est un <state> optionnel.

2.15.1 Déclaration XML de l'élément select-constraint

```
<!ELEMENT select-constraint ( state?) >
<!ATTLIST select-constraint
  %commonAttributes; >
```

2.16 select-update

L'élément <select-update> indique l'update sélectionnée pour déclencher les contrainte. Les attributs requis sont les mêmes que ceux d'update plus les attributs de %eventAttributes;, à savoir :

vname (req) le référant unique de la variable ;

type (opt) la raison de l'activation, parmi les mots réservés suivants : ground, any, min, max, minmax, empty, nothing et val.

Le contenu de l'élément est un <state> optionnel et une liste de valeurs.

2.16.1 Déclaration XML de l'élément select-update

```
<!ELEMENT select-update ( state? %valueList; ) >
<!ATTLIST select-update
  %eventAttributes;
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax |
  empty | val | nothing) #REQUIRED >
```

2.17 activate

L'élément <activate> signale qu'une contrainte devient active. Les attributs requis sont les attributs généraux. Le contenu de l'élément est un <state> optionnel et une <explanation>.

2.17.1 Déclaration XML de l'élément activate

```
<!ELEMENT activate (state?, explanation?) >
<!ATTLIST activate
  %commonAttributes; >
```

2.18 tell

L'élément `<tell>` met en place un point de choix. Les attributs requis sont les attributs généraux dont l'attribut `depth` n'est pas optionnel et indique le niveau d'imbrication des points de choix. Le contenu de l'élément est un `<state>` optionnel. La gestion du backtrack dépend de la valeur de l'attribut `depth`. Un backtrack est signalé par l'utilisation de l'événement `<restore>` avec une valeur de `<depth>` égale à celle du `<tell>` précédent (on pourrait revenir à plusieurs niveaux en arrière mais ce n'est pas l'usage). Les solveurs sans backtrack n'utilisent pas `<tell>` et l'attribut `depth` a toujours la valeur 0.

2.18.1 Déclaration XML de l'élément tell

```
<!ELEMENT tell (state?) >
<!ATTLIST tell
   depth %integer; #REQUIRED
   n %integer; #IMPLIED
   time %integer; #IMPLIED
   context CDATA #IMPLIED
   activities IDREFS #IMPLIED
   %constraintAttributes; >
```

2.19 deactivate

L'élément `<deactivate>` signale qu'une contrainte n'est plus active. Les attributs requis sont les attributs généraux. Le contenu de l'élément est un `<state>` optionnel.

2.19.1 Déclaration XML de l'élément deactivate

```
<!ELEMENT deactivate (state?) >
<!ATTLIST deactivate
  %commonAttributes; >
```

2.20 solution

L'élément `<solution>` signale que le state est solution du système de contraintes. Les attributs requis sont les attributs généraux et une valeur traduisant la

«qualité» de la solution. Concrètement, cette valeur peut représenter un coût ou la valeur de la fonction objectif correspondant à la solution ; cette information pouvant servir pour discriminer visuellement entre les différentes solutions trouvées. Le contenu de l'élément est un `<state>` optionnel.

2.20.1 Déclaration XML de l'élément solution

```
<!ELEMENT solution (state?) >
<!ATTLIST solution
  val %integer; #IMPLIED
  %commonAttributes; >
```

2.21 new-stage

L'élément `<new-stage>` déclare un nouveau type d'étape. En principe, tous les éléments de ce type apparaissent en début de session et sont utilisés au cours de la session pour annoter la trace de la liaison entre les événements de bas niveau du solveur et les objets de plus haut niveau que veulent voir les utilisateurs du solveur. Une étape peut être temporelle, comme le début du labelling ou le chargement d'un fichier externe.

Les attributs requis sont un identificateur unique qui servira à référencer l'étape par la suite.

Le contenu de l'élément est un commentaire textuel indiquant la signification de l'étape.

2.21.1 Déclaration XML de l'élément new-stage

```
<!ELEMENT new-stage (#PCDATA) >
<!ATTLIST new-stage
  id ID #REQUIRED >
```

2.22 stage

L'élément `<stage>` encadre une nouvelle instante d'étape. Les attributs requis sont :

`sname` (req) le nom de l'étape, tel que déclare par un `<new-stage>` ;

`refs` (opt) les identifiants des entités référencées par cette étape ;

`detail` (opt) une indication sur les détails de cette étape particulière. Par exemple, si l'étape est le chargement d'un fichier externe, le détail peut contenir le nom du fichier.

2.22.1 Déclaration XML de l'élément stage et des attributs utilisables dans chaque étape

```
<!ENTITY % StageAttributes
  "sname IDREF #REQUIRED
   refs IDREFS #IMPLIED
   detail CDATA #IMPLIED" >

<!ELEMENT stage %Toplevel; >
<!ATTLIST stage
  %StageAttributes; >
```

2.22.2 Exemple

La trace qui suit définit une étape "chargement de fichier" et l'utilise sur le fichier init.pl :

```
<new-stage id="load">File loading</new-stage>
...
<stage name="load" detail="init.pl">
  <new-variable n="1" vname="x">
    <values>1 2 3</values>
  </new-variable>
  <new-variable n="2" vname="y">
    <values>1 2 3</values>
  </new-variable>
  <new-constraint n="4"  cname="c1" variables="x y" externalrep="X ## Y">
    <update vname="x" type="ground"/>
    <update vname="y" type="ground"/>
  </new-constraint>
</stage>
```

2.23 start/suspend/resume/end-stage

L'élément `<start-stage>` signale le début d'une nouvelle étape qui peut être suspendue, reprise ou terminée. En utilisant `<start-stage>` plutôt que `<stage>`, il est possible d'entrelacer les étapes, ce qu'XML ne permet pas lorsqu'on utilise des éléments inclus dans d'autres éléments qui doivent suivre la règle d'imbrication stricte.

2.23.1 Déclaration XML des éléments start-stage suspend-stage resume-stage end-stage

```
<!ELEMENT start-stage EMPTY >
<!ATTLIST start-stage
  %StageAttributes; >

<!ELEMENT suspend-stage EMPTY >
```

```
<!ATTLIST suspend-stage
  %StageAttributes; >

<!ELEMENT resume-stage EMPTY >
<!ATTLIST resume-stage
  %StageAttributes; >

<!ELEMENT end-stage EMPTY >
<!ATTLIST end-stage
  %StageAttributes; >
```

Chapitre 3

Codage et compression

La trace produite par un solveur n'est en principe pas destinée à être stockée dans un fichier XML mais à être envoyée à la volée à un programme qui l'exploitera (un analyseur ou un programme de visualisation). Un format XML textuel a l'avantage d'être compréhensible par un humain et d'être manipulable à l'aide de bibliothèques très nombreuses et de très bonne qualité. En revanche, XML est un format coûteux en terme de place mémoire et éventuellement de vitesse d'analyse lexicale.

Pour pallier à ces deux défauts, il est possible d'utiliser le format wbxml[2] (Wap Binary XML) qui code le format XML sous forme d'une suite de marqueurs binaires. Ce codage augmente sensiblement la bande passante en réduisant le nombre d'octets nécessaire à transmettre la même information. En outre, wbxml est plus rapide à analyser car il évite totalement la phase de décodage lexical de XML. Dans le cas où la DTD est fixée, la génération du format wbxml peut aussi être plus rapide que celui de XML car les préfixes fréquents peuvent être transmis sous une forme compressée. Ainsi, tous les noms d'éléments et d'attributs peuvent être envoyés à l'aide d'un ou deux octets.

L'utilisation de wbxml nécessite une bibliothèque de codage/décodage dans les solveurs. Si le besoin s'en fait sentir, une bibliothèque sera développée pour éviter à chaque partenaire de refaire le travail.

Annexe A

DTD OADYMPPAC

```
<!--
  Copyright (c) 2001
  Ecole des Mines de Nantes, INRIA, IRISA, Universite d'Orleans, Cosytec, Ilog

  DTD describing traces for constraint based programs.
  Project OADYMPPAC
  http://contraintes.inria.fr/OADymPPaC/

-->
<!ENTITY % Toplevel
  "(new-variable | new-constraint | reduce | restore | wake-up |
    suspend | true | reject | activate | select-constraint |
    select-update | tell | deactivate | solution | new-annotation |
    new-stage | start-stage | end-stage |
    suspend-stage | resume-stage | stage )*" >

<!ELEMENT oadymppac %Toplevel; >
<!ATTLIST oadymppac
   xmlns CDATA "http://contraintes.inria.fr/OADymPPaC">

<!ENTITY % integer "CDATA" >

<!ENTITY % constraintAttributes
  "cname IDREF #IMPLIED
   externalrep CDATA #IMPLIED
   internalrep CDATA #IMPLIED" >

<!ENTITY % eventAttributes
  "depth %integer; #IMPLIED
   n %integer; #IMPLIED
   time %integer; #IMPLIED
   context CDATA #IMPLIED
   activities IDREFS #IMPLIED" >
```

```
<!ENTITY % commonAttributes
  "%eventAttributes;
   %constraintAttributes;" >

<!ENTITY % valueList "( values | range )*" >

<!ELEMENT new-variable %valueList; >
<!ATTLIST new-variable
  %eventAttributes;
  vname ID #REQUIRED
  type (int | rat | real | enum | string ) "int"
  externalrep CDATA #IMPLIED
  internalrep CDATA #IMPLIED >

<!ELEMENT new-constraint (update)* >
<!ATTLIST new-constraint
  %eventAttributes;
  cname ID #REQUIRED
  orig ( user | system ) #IMPLIED
  externalrep CDATA #IMPLIED
  internalrep CDATA #IMPLIED >

<!ELEMENT reduce ( state?, (update)*, explanation? ) >
<!ATTLIST reduce
  %commonAttributes; >

<!ELEMENT restore ( state?, (update)* ) >
<!ATTLIST restore
  %commonAttributes; >

<!ELEMENT wake-up (state?, (cause)* ) >
<!ATTLIST wake-up
  %commonAttributes; >

<!ELEMENT suspend (state?) >
<!ATTLIST suspend
  %commonAttributes; >

<!ELEMENT true (state?) >
<!ATTLIST true
  %commonAttributes; >

<!ELEMENT reject ( state?, explanation?) >
<!ATTLIST reject
  %commonAttributes; >

<!ELEMENT select-constraint ( state?) >
<!ATTLIST select-constraint
  %commonAttributes; >
```

```
<!ELEMENT select-update ( state? , %valueList; ) >
<!ATTLIST select-update
  %eventAttributes;
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax |
  empty | val | nothing) #REQUIRED >

<!ELEMENT activate (state?, explanation?) >
<!ATTLIST activate
  %commonAttributes; >

<!ELEMENT tell (state?) >
<!ATTLIST tell
   depth %integer; #REQUIRED
   n %integer; #IMPLIED
   time %integer; #IMPLIED
   context CDATA #IMPLIED
   activities IDREFS #IMPLIED
   %constraintAttributes; >

<!ELEMENT deactivate (state?) >
<!ATTLIST deactivate
  %commonAttributes; >

<!ELEMENT solution (state?) >
<!ATTLIST solution
  val %integer; #IMPLIED
  %commonAttributes; >

<!-- Annotations may replace stages -->
<!ELEMENT new-annotation (#PCDATA) >
<!ATTLIST new-annotation
  id ID #REQUIRED
  type IDREF #IMPLIED
  name CDATA #IMPLIED
  refs IDREFS #IMPLIED >

<!ELEMENT new-stage (#PCDATA) >
<!ATTLIST new-stage
  id ID #REQUIRED
  sname CDATA #IMPLIED
  refs IDREFS #IMPLIED >
<!-- such as "labelling", "init" -->

<!ENTITY % StageAttributes
  "sname IDREF #REQUIRED
   refs IDREFS #IMPLIED
   detail CDATA #IMPLIED" >

<!ELEMENT stage %Toplevel; >
```

```
<!ATTLIST stage
  %StageAttributes; >

<!ELEMENT start-stage EMPTY >
<!ATTLIST start-stage
  %StageAttributes; >

<!ELEMENT suspend-stage EMPTY >
<!ATTLIST suspend-stage
  %StageAttributes; >

<!ELEMENT resume-stage EMPTY >
<!ATTLIST resume-stage
  %StageAttributes; >

<!ELEMENT end-stage EMPTY >
<!ATTLIST end-stage
  %StageAttributes; >

<!ELEMENT values (#PCDATA) >

<!ELEMENT range EMPTY>
<!ATTLIST range
  from CDATA #REQUIRED
  to CDATA #REQUIRED >

<!ELEMENT vardomain %valueList; >
<!ATTLIST vardomain
  vname IDREF #REQUIRED >

<!ELEMENT state (alist?, slist?, qlist?, tlist?, rlist?, ulist?, vlist?, (misc)*) >
<!ATTLIST state
  precision (changed | full) #IMPLIED >

<!ELEMENT alist (constraint)* >
<!ELEMENT slist (constraint)* >
<!ELEMENT qlist (constraint)* >
<!ELEMENT tlist (constraint)* >
<!ELEMENT rlist (constraint)* >
<!ELEMENT ulist (update)* >
<!ELEMENT vlist (vardomain)* >
<!ELEMENT misc (#PCDATA) >
<!ATTLIST misc
  type CDATA #IMPLIED >

<!ELEMENT constraint EMPTY>
<!ATTLIST constraint
  cname IDREF #REQUIRED >

<!ELEMENT update %valueList; >
```

```
<!ATTLIST update
  cname IDREF #IMPLIED
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax | empty | val | nothing) #IMPLIED >

<!ELEMENT cause %valueList; >
<!ATTLIST cause
  cname IDREF #IMPLIED
  vname IDREF #REQUIRED
  type (ground | any | min | max | minmax | empty | val) #IMPLIED >


<!ELEMENT explanation ((cause)*, (constraint)*) >
```

Annexe B

Exemple de trace issue de Prolog

```
<!DOCTYPE oadymppac SYSTEM "oadymppac.dtd">
<oadymppac>
  <new-variable n="1" vname="x"><values>1 2 3</values></new-variable>
  <new-variable n="2" vname="y"><values>1 2 3</values></new-variable>
  <new-variable n="3" vname="z"><values>1 2 3</values></new-variable>
  <new-constraint n="4"  cname="c1" externalrep="X ## Y">
    <update vname="x" type="ground"/>
    <update vname="y" type="ground"/>
  </new-constraint>
  <tell n="5" depth="1" cname="c1">
  </tell>
  <suspend n="6" cname="c1">
    <state>
      <alist>
<constraint cname="c1"/>
      </alist>
    </state>
  </suspend>
  <new-constraint n="7" cname="c2" externalrep="X #>= Y">
    <update vname="x" type="max"/>
    <update vname="y" type="min"/>
  </new-constraint>
  <tell n="8" depth="2" cname="c2">
    <state>
      <alist>
<constraint cname="c2"/>
      </alist>
      <slist>
<constraint cname="c1"/>
      </slist>
    </state>
```

```
  </tell>
  <suspend n="9" cname="c2">
    <state>
      <slist>
<constraint cname="c1"/>
      </slist>
    </state>
  </suspend>
  <new-constraint n="10" cname="c3" externalrep="Y #> Z">
    <update vname="y" type="max"/>
    <update vname="z" type="min"/>
  </new-constraint>
  <tell n="11" depth="3" cname="c3">
    <state>
      <slist>
<constraint cname="c1"/>
<constraint cname="c2"/>
      </slist>
    </state>
  </tell>
  <reduce n="12" cname="c3">
    <state>
      <alist>
<constraint cname="c3"/>
      </alist>
      <slist>
<constraint cname="c1"/>
<constraint cname="c2"/>
      </slist>
    </state>
    <update vname="y" type="min"><values>1</values></update>
    <update vname="y" type="any"/>
  </reduce>
  <wake-up n="13" cname="c2">
    <state>
      <alist>
<constraint cname="c3"/>
      </alist>
      <slist>
<constraint cname="c1"/>
      </slist>
      <qlist>
<constraint cname="c2"/>
      </qlist>
    </state>
    <cause vname="y" type="min"/>
  </wake-up>
  <reduce n="14" cname="c3">
    <state>
      <alist>
```

```
<constraint cname="c3"/>
      </alist>
      <slist>
<constraint cname="c1"/>
      </slist>
      <qlist>
<constraint cname="c2"/>
      </qlist>
    </state>
    <update vname="z" type="max"><values>3</values></update>
    <update vname="z" type="any"/>
  </reduce>
  <suspend n="15" cname="c3">
    <state>
      <alist>
<constraint cname="c3"/>
      </alist>
      <slist>
<constraint cname="c1"/>
      </slist>
      <qlist>
<constraint cname="c2"/>
      </qlist>
    </state>
  </suspend>
  <select-constraint n="16" cname="c2">
    <state>
      <slist>
<constraint cname="c1"/>
<constraint cname="c3"/>
      </slist>
      <qlist>
<constraint cname="c2"/>
      </qlist>
    </state>
  </select-constraint>
  <reduce n="17" cname="c2">
    <update vname="x" type="min"><values>1</values></update>
    <update vname="x" type="any"/>
  </reduce>
  <suspend n="18" cname="c2">
    <state>
      <alist>
<constraint cname="c2"/>
      </alist>
      <slist>
<constraint cname="c1"/>
<constraint cname="c3"/>
      </slist>
    </state>
```

```
  </suspend>
  <new-constraint n="19" cname="cx2" externalrep="X#=2"/>
  <tell n="20" depth="4" cname="cx2">
    <state>
      <slist>
<constraint cname="c1"/>
<constraint cname="c2"/>
<constraint cname="c3"/>
      </slist>
    </state>
  </tell>
  <reduce n="21" cname="cx2">
    <state>
      <alist>
<constraint cname="cx2"/>
      </alist>
      <slist>
<constraint cname="c1"/>
<constraint cname="c2"/>
<constraint cname="c3"/>
      </slist>
    </state>
    <update vname="x" type="max"><values>3</values></update>
    <update vname="x" type="any"/>
    <update vname="x" type="ground"/>
  </reduce>
  <wake-up n="22" cname="c2">
    <state>
      <slist>
<constraint cname="c1"/>
<constraint cname="c2"/>
<constraint cname="c3"/>
<constraint cname="cx2"/>
      </slist>
    </state>
    <cause vname="x" type="max"/>
  </wake-up>
  <wake-up n="23" cname="c1">
    <state>
      <slist>
<constraint cname="c1"/>
<constraint cname="c3"/>
<constraint cname="cx2"/>
      </slist>
      <qlist>
<constraint cname="c2"/>
      </qlist>
    </state>
    <cause vname="x" type="ground"/>
  </wake-up>
```

```
  <true n="24" cname="cx2">
    <state>
      <slist>
<constraint cname="c3"/>
<constraint cname="cx2"/>
      </slist>
      <qlist>
<constraint cname="c1"/>
<constraint cname="c2"/>
      </qlist>
    </state>
  </true>
  <select-constraint n="25" cname="c2">
    <state>
      <slist>
<constraint cname="c3"/>
      </slist>
      <qlist>
<constraint cname="c1"/>
<constraint cname="c2"/>
      </qlist>
      <tlist>
<constraint cname="cx2"/>
      </tlist>
    </state>
  </select-constraint>
  <reduce n="26" cname="c2">
    <state>
      <alist>
<constraint cname="c2"/>
      </alist>
      <slist>
<constraint cname="c3"/>
      </slist>
      <qlist>
<constraint cname="c1"/>
      </qlist>
      <tlist>
<constraint cname="cx2"/>
      </tlist>
    </state>
    <update vname="y" type="max"><values>3</values></update>
    <update vname="y" type="any"/>
    <update vname="y" type="ground"/>
  </reduce>

</oadymppac>
```

Annexe C

Exemple de trace issue de Choco

```
<!DOCTYPE oadymppac SYSTEM "oadymppac.dtd">
<oadymppac>
  <new-stage id="init">Initialization Stage</new-stage>
  <new-stage id="narrowing">Narrowing Stage</new-stage>
  <new-stage id="labeling">Labeling Stage</new-stage>

  <start-stage sname="init"/>
  <new-constraint n="0" cname="c-control" externalrep="control"/>
  <new-variable n="1" vname="X">
    <values> 1 2 3 4</values>
  </new-variable>
  <new-variable n="2" vname="Y">
     <values> 1 2 3 4</values>
</new-variable>
<new-variable n="3" vname="Z">
     <values> 1 2 3 4</values>
</new-variable>
<new-constraint n="4" cname="c1" externalrep="X !== Y + 0">
    <update vname="X" type="ground" />
    <update vname="Y" type="ground" />
</new-constraint>
<activate n="5" cname="c1" />
<suspend n="6" cname="c1" />
<new-constraint n="7" cname="c2" externalrep="X >= Y + 0">
    <update vname="X" type="max" />
    <update vname="Y" type="min" />
</new-constraint>
<suspend-stage sname="init"/>
<start-stage sname="narrowing"/>
<activate n="8" cname="c2" />
<suspend n="9" cname="c2" />
<reduce n="10" cname="c2" >
     <update vname="X" type="nothing" />
</reduce>
```

```
<reduce n="11" cname="c2" >
      <update vname="Y" type="nothing" />
</reduce>
<suspend-stage sname="narrowing"/>
<resume-stage sname="init"/>
<new-constraint n="12" cname="c3" externalrep="Y >= Z + 1">
    <update vname="Y" type="max" />
    <update vname="Z" type="min" />
</new-constraint>
<suspend-stage sname="init"/>
<resume-stage sname="narrowing"/>
<activate n="13" cname="c3" />
<suspend n="14" cname="c3" />
<end-stage sname="init"/>
<resume-stage sname="narrowing"/>
<reduce n="15" cname="c3" >
 <update vname="Y" type="min"><values>1</values></update>
</reduce>
<reduce n="16" cname="c3" >
 <update vname="Z" type="max"><values>4</values></update>
</reduce>
<select-update n="17" vname="Y" type="min" />
<select-constraint n="18" cname="c1" />
<wake-up n="19" cname="c1" >
  <cause vname="Y" type="min" />
</wake-up>
<suspend n="20" cname="c1" />
<select-constraint n="21" cname="c2" />
<wake-up n="22" cname="c2" >
  <cause vname="Y" type="min" />
</wake-up>
<reduce n="23" cname="c2" >
 <update vname="X" type="min"><values>1</values></update>
</reduce>
<suspend n="24" cname="c2" />
<select-update n="25" vname="Z" type="max" />
<select-update n="26" vname="X" type="min" />
<select-constraint n="27" cname="c1" />
<wake-up n="28" cname="c1" >
  <cause vname="X" type="min" />
</wake-up>
<suspend n="29" cname="c1" />
<end-stage sname="narrowing"/>
<start-stage sname="labeling"/>
<new-constraint n="30" cname="cX2" externalrep="X = 2">
    <update vname="X"/>
</new-constraint>
<activate n="31" cname="cX2"/>
<tell n="32" depth="1" cname="cX2" />
<reduce n="33" cname="cX2">
```

```
 <update vname="X" type="ground"><values>3 4</values></update>
</reduce>
<select-update n="34" vname="X" type="ground" />
<select-constraint n="35" cname="c1" />
<wake-up n="36" cname="c1" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="37" cname="c1" >
  <update vname="Y" type="min"><values>2</values></update>
</reduce>
<suspend n="38" cname="c1" />
<select-constraint n="39" cname="c2" />
<wake-up n="40" cname="c2" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="41" cname="c2" >
  <update vname="Y"><values>3 4</values></update>
</reduce>
<reject n="42" cname="c2"/>
<deactivate n="43" cname="cX2" />
<restore n="44" depth="0" cname="cX2" />
<reduce n="45" cname="c-control">
 <update vname="X" type="min"><values> 2</values></update>
</reduce>
<select-update n="46" vname="X" type="min" />
<select-constraint n="47" cname="c1" />
<wake-up n="48" cname="c1" >
  <cause vname="X" type="min" />
</wake-up>
<suspend n="49" cname="c1" />
<select-constraint n="50" cname="c2" />
<wake-up n="51" cname="c2" >
  <cause vname="X" type="min" />
</wake-up>
<suspend n="52" cname="c2" />
<new-constraint n="53" cname="cX3" externalrep="X = 3">
  <update vname="X"/>
</new-constraint>
<activate n="54" cname="cX3"/>
<tell n="55" depth="1" cname="cX3" />
<reduce n="56" cname="cX3">
     <update vname="X" type="ground"><values>  4</values></update>
</reduce>
<select-update n="57" vname="X" type="ground" />
<select-constraint n="58" cname="c1" />
<wake-up n="59" cname="c1" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="60" cname="c1" >
  <update vname="Y" type="val"> <values>3</values></update>
```

```
</reduce>
<suspend n="61" cname="c1" />
<select-constraint n="62" cname="c2" />
<wake-up n="63" cname="c2" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="64" cname="c2" >
  <update vname="Y" type="ground"> <values> 4</values></update>
  </reduce>
<suspend n="65" cname="c2" />
<select-update n="66" vname="Y" type="ground" />
<select-constraint n="67" cname="c3" />
<wake-up n="68" cname="c3" >
  <cause vname="Y" type="ground" />
</wake-up>
<reduce n="69" cname="c3" >
  <update vname="Z" type="ground"> <values> 2 3</values></update>
</reduce>
<suspend n="70" cname="c3" />
<select-update n="71" vname="Z" type="ground" />
<solution/>
<deactivate n="72" cname="cX3" />
<restore n="73" depth="0" cname="cX3" />
<reduce n="74" cname="c-control">
  <update vname="X" type="ground"> <values> 3</values></update>
</reduce>
<select-update n="75" vname="X" type="ground" />
<select-constraint n="76" cname="c1" />
<wake-up n="77" cname="c1" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="78" cname="c1" >
  <update vname="Y" type="max"> <values> 4</values></update>
</reduce>
<suspend n="79" cname="c1" />
<select-constraint n="80" cname="c2" />
<wake-up n="81" cname="c2" >
  <cause vname="X" type="ground" />
</wake-up>
<reduce n="82" cname="c2" >
      <update vname="Y" type="nothing" />
</reduce>
<suspend n="83" cname="c2" />
<select-update n="84" vname="Y" type="max" />
<select-constraint n="85" cname="c3" />
<wake-up n="86" cname="c3" >
  <cause vname="Y" type="max" />
</wake-up>
<reduce n="87" cname="c3" >
  <update vname="Z" type="max"> <values> 3</values></update>
```

```
</reduce>
<suspend n="88" cname="c3" />
<select-update n="89" vname="Z" type="max" />
<new-constraint n="90" cname="cX4" externalrep="X = 4">
    <update vname="X"/>
</new-constraint>
<activate n="91" cname="cX4"/>
<tell n="92" depth="1" cname="cX4" />
<new-constraint n="93" cname="cY2" externalrep="Y = 2">
    <update vname="Y"/>
</new-constraint>
<activate n="94" cname="cY2"/>
<tell n="95" depth="2" cname="cY2" />
<reduce n="96" cname="cY2">
  <update vname="Y" type="ground"> <values>  3</values></update>
</reduce>
<select-update n="97" vname="Y" type="ground" />
<select-constraint n="98" cname="c3" />
<wake-up n="99" cname="c3" >
  <cause vname="Y" type="ground" />
</wake-up>
<reduce n="100" cname="c3" >
  <update vname="Z" type="ground"> <values> 2</values></update>
  </reduce>
<suspend n="101" cname="c3" />
<select-update n="102" vname="Z" type="ground" />
<solution/>
<deactivate n="103" cname="cY2" />
<restore n="104" depth="1" cname="cY2" />
<reduce n="105" cname="c-control">
  <update vname="Y" type="ground"> <values> 2</values></update>
</reduce>
<select-update n="106" vname="Y" type="ground" />
<select-constraint n="107" cname="c3" />
<wake-up n="108" cname="c3" >
  <cause vname="Y" type="ground" />
</wake-up>
<reduce n="109" cname="c3" >
      <update vname="Z" type="nothing" />
</reduce>
<suspend n="110" cname="c3" />
<new-constraint n="111" cname="cY3" externalrep="Y = 3">
    <update vname="Y"/>
</new-constraint>
<activate n="112" cname="cY3"/>
<tell n="113" depth="2" cname="cY3" />
<new-constraint n="114" cname="cZ1" externalrep="Z = 1">
    <update vname="Z"/>
</new-constraint>
<activate n="115" cname="cZ1"/>
```

```
<tell n="116" depth="3" cname="cZ1" />
<reduce n="117" cname="cZ1">
  <update vname="Z" type="ground"> <values>  2</values></update>
</reduce>
<select-update n="118" vname="Z" type="ground" />
<solution/>
<deactivate n="119" cname="cZ1" />
<restore n="120" depth="2" cname="cZ1" />
<reduce n="121" cname="c-control">
  <update vname="Z" type="ground"> <values> 1</values></update>
</reduce>
<select-update n="122" vname="Z" type="ground" />
<new-constraint n="123" cname="cZ2" externalrep="Z = 2">
    <update vname="Z"/>
</new-constraint>
<activate n="124" cname="cZ2"/>
<tell n="125" depth="3" cname="cZ2" />
<solution/>
<deactivate n="126" cname="cZ2" />
<restore n="127" depth="2" cname="cZ2" />
<reduce n="128" cname="c-control">
    <update vname="Z"><values>2</values></update>
</reduce>
<reject n="129" cname="c-control"/>
<deactivate n="130" cname="cY3" />
<restore n="131" depth="1" cname="cY3" />
<reduce n="132" cname="c-control">
    <update vname="Y"><values>3</values></update>
</reduce>
<reject n="133" cname="c-control"/>
<deactivate n="134" cname="cX4" />
<restore n="135" depth="0" cname="cX4" />
<reduce n="136" cname="c-control">
    <update vname="X"><values>4</values></update>
</reduce>
<reject n="137" cname="c-control"/>
<end-stage sname="labeling"/>
</oadymppac>
```

Bibliographie

[1] Ludovic Langevine, Pierre Deransart, Mireille Ducasse, and Erwan Jahier. Prototyping clp(fd) tracers : a trace model and an experimental validation environment. In *WLPE'01 post-conférence workshop ICLP'2001, Cyprus*, Novembre 2001.

[2] Bruce Martin and Bashar Jano. Wap binary xml content format, w3c note 24 june 1999. Technical report, W3 Consortium, 1999. http ://www.w3.org/TR/wbxml.

[3] The Unicode Consortium. *The Unicode Standard, Version 3.0*. Addison-Wesley, Reading, MA, USA, 2000. Includes CD-ROM.

[4] Jean Paoli Tim Bray and C. M. Sperberg-McQueen eds. Extensible markup language (xml) 1.0. Technical report, W3 Consortium, 1998. http ://www.w3.org/TR/REC-xml.

[5] F. Yergeau. RFC 2279 : UTF-8, a transformation format of ISO 10646, January 1998.

Index