Technique de résolution de proformes enchâssées

Claude Laï et Robert Pasero

Laboratoire d'Informatique de Marseille
CNRS & Université de la Méditerranée
Case 901, 163 Avenue de Luminy,
F-13288 Marseille Cedex 9 FRANCE
{lai, pasero}@lim.univ-mrs.fr

Résumé

Nous présentons une technique de résolution de proformes enchâssées à l'aide des métastructures Prolog. Nous montrons tout d'abord un exemple d'utilisation de ces métastructures pour contrôler l'appartenance d'un élément à un domaine. Une plus grande utilité est ensuite démontrée dans la résolution de contraintes contextuelles dynamiques, qui sont particulières dans le sens où elles interviennent en fonction des contraintes déjà existantes sur les éléments considérés. Une application utile de ces contraintes est d'éviter les redondances dans la recherche des possibilités de référents pour un discours considéré, notamment dans le cas de proformes enchâssées.

1. Introduction

De récents travaux de recherche (Pasero & Sabatier, 1997; Garde & Laï, 1998) montrent l'apport considérable de la programmation par contraintes dans le domaine du Traitement Automatique du Langage Naturel (TALN). Certains des problèmes relatifs à ce domaine pourraient être abordés avec des méthodes algorithmiques classiques, mais une manière plus naturelle de raisonner est de ne pas se soucier du moment où les contraintes devront être vérifiées. Il s'agit donc de les poser au plus tôt et de laisser le système les résoudre «au mieux» au fil d'une exécution. La plupart de ces contraintes peuvent être résolues par des méthodes usuelles de Programmation par Contraintes, mais, comme nous allons le voir, certaines d'entre elles peuvent être bien particulières : c'est par exemple le cas lorsque l'on s'intéresse à la résolution des proformes (anaphores pronominales) (Dale, 1989; Corblin, 1995) dans le cadre d'un système de composition guidée (Mouret & Rolbert, 1998), et que l'on désire éviter une redondance dans les solutions. Pour résoudre ces contraintes, nous avons utilisé le langage Prolog (Giannesini *et al.*, 1985), langage particulièrement adapté à notre domaine d'application (Pereira & Shieber, 1987). Notre méthode fait intervenir la notion de métastructure (ou variable attribuée). Dans une première section, nous allons rappeler cette notion et donner une configuration simple où les métastructures peuvent s'avérer utiles. Nous décrirons ensuite le cas des contraintes contextuelles dynamiques, et une application particulière dans le domaine du TALN concernant la résolution des proformes enchâssées.

2. Les métastructures

2.1. Description

Les métastructures (Neumerkel, 1990) représentent un sous-ensemble de termes Prolog pour lesquels l'unification est traitée de manière particulière. En effet, alors que pour les termes «ordinaires» le principe de l'unification est bien déterminé, l'unification des métastructures est contrôlée par des règles spécifiées par l'utilisateur. De manière concrète, un sous-ensemble des variables Prolog est défini d'une manière telle qu'à chaque élément de cet ensemble sont associés un nom de règle (prédicat), ainsi qu'un terme quelconque : ce terme est appelé *attribut*, d'où le nom de *variable attribuée* ou encore *méta-variable*. Lors d'une unification de l'une de ces variables avec un terme Prolog quelconque (y compris une métastructure), la règle (définie par l'utilisateur) attachée à cette variable est appelée avec en paramètre l'attribut de cette variable. Dans cette règle, on a la faculté de changer l'attribut de la variable.

De par ce mécanisme de contrôle, les métastructures peuvent permettre à l'utilisateur d'implémenter efficacement plusieurs extensions de Prolog comme les extensions fonctionnelles, les contraintes (Holzbaur, 1990) ou encore des primitives méta-logiques.

2.2. Exemple simple d'utilisation : contrôle d'appartenance à un domaine

2.2.1. Description

Imaginons que l'on veuille écrire un programme en Prolog qui manipule des formules logiques. La manière la plus naturelle de représenter une variable logique est alors de la représenter par une variable Prolog. Comme cela est discuté dans (Giannesini *et al.*, 1985), cela constitue un abus dans le sens où après une telle construction, toute variable logique pourra être substituée par un terme Prolog quelconque. Le moyen de maîtriser cela est de représenter une variable logique par une métastructure, le prédicat attribué étant un prédicat de vérification d'appartenance de cette variable logique à un domaine bien déterminé. Ainsi, sur un essai de substitution d'une variable logique par un terme, ce prédicat de vérification sera activé et la formule parfois invalidée.

De façon plus générale, pour certains objets, on peut être amené à devoir définir leur domaine d'appartenance (fini ou infini). Par exemple dans une grammaire, les arguments qui représentent les traits doivent prendre leurs valeurs dans des ensembles (finis) bien déterminés. Ces ensembles peuvent être réduits lors d'une analyse de phrase. D'un point de vue logique, ces arguments peuvent être vus comme des méta-variables auxquelles sont attachés des domaines : leur domaine de définition. La relation entre la méta-variable X et son domaine D sera à tout moment : $X \in D$. Le maintien de la validité de cette relation devra se réaliser par des actions effectuées lors des unifications. Notamment, une réduction d'ensembles aura pour conséquence un remplacement du terme attribué à la métastructure par un terme représentant l'intersection des ensembles.

2.2.2. Programmation en Prolog

Les prédicats suivants illustrent la programmation de la vérification de l'appartenance de la variable X à un domaine. Cette variable est en fait une métastructure qui est créée à l'initialisation du programme (règle $init/1$) par un appel au prédicat prédéfini $newMetastructure/3$. Le second argument de ce prédicat est le terme attribué à la métastructure, représentant en l'occurrence son domaine d'appartenance. Le troisième est le nom d'un prédicat d'arité 3 ($verifyInto$)

devant être défini par l'utilisateur, et qui sera automatiquement appelé sur des essais d'unification de la métastructure. En l'occurrence, ce prédicat vérifie que ces essais d'unification se font bien sur des termes appartenant au domaine de la métastructure. Dans ce prédicat utilisateur :

- Y est le terme avec lequel on essaie d'unifier X ;
- $Domaine$ est le domaine attaché à X (attribut de X, en l'occurrence la liste de valeurs *[masculin, féminin, singulier, pluriel]*) ;
- le prédicat prédéfini $member/2$ est un prédicat classique vérifiant l'appartenance d'un élément à une liste ;
- le prédicat prédéfini $metaUnify/2$ réalise l'unification Prolog classique. En effet, l'appel au prédicat standard '$= /2$' en lieu et place de $metaUnify/2$ ne ferait que rappeler le prédicat utilisateur $verifyInto/3$ (boucle infinie).

```
init(X) :-
      newMetastructure(X,
           [masculin, féminin, singulier, pluriel],
           verifyInto).

verifyInto(X,Y,Domaine) :-
      member(X,Domaine),
      metaUnify(X,Y).
```

Un appel pourrait alors être de la forme :

```
?- init(X), init(Y), goal(X,Y).
```

où le but *goal(X,Y)* réaliserait certaines opérations sur les éléments X et Y.

3. Contraintes contextuelles dynamiques

La notion de contrainte contextuelle dynamique utilisée ici fait référence à des contraintes qui seront posées en cours d'exécution, donc dynamiques, par opposition à des contraintes statiques qui sont écrites explicitement par le programmeur et seront prises en compte lors de toute exécution. Elles sont contextuelles dans le sens où elles dépendent d'une exécution et seront posées (ou non) en fonction des contraintes courantes.

3.1. Illustration du problème

Soient E_1 et E_2 deux ensembles de constantes tels que $E_1 \cap E_2 \neq \emptyset$. Soient X et Y deux variables telles que $Y \in E_2$, et $X \in E_1 \cup \{\sigma(Y)\}$, où $\sigma(Y)$ représente une valeur quelconque parmi celles que peut prendre la variable Y. Pour fixer les idées, nous allons prendre $E_1 = \{1, 2, 3\}$ et $E_2 = \{2, 3, 4\}$. Nous avons donc $X \in \{1, 2, 3\} \cup \{\sigma(Y)\}$ et $Y \in \{2, 3, 4\}$. Si nous faisons la liste exhaustive des couples ordonnés $(\sigma(X), \sigma(Y))$, obtenus en énumérant toutes les

valeurs de E_1 et E_2, nous obtenons alors :
$(1,2),(2,2),(3,2),(2,2),(1,3),(2,3),(3,3),(3,3),(1,4),(2,4),(3,4),(4,4)$.

Nous remarquons que les couples $(2,2)$ et $(3,3)$ apparaissent deux fois dans cette liste. Ce que nous désirons en fait est une liste non redondante de ces couples. C'est à dire que sachant que les couples $(\sigma(X),\sigma(Y))$ avec $\sigma(X)=\sigma(Y)$ seront forcément dans la liste résultat de par le fait que $\sigma(Y)$ appartient au domaine de X, nous ne voulons pas qu'ils le soient d'une autre manière.

Donc, en supposant que la variable X soit instanciée avant Y, si l'on veut éviter les redondances dans la liste résultat, juste après l'instanciation de X, deux cas sont à distinguer :

1. si la contrainte $X=Y$ fait partie des contraintes courantes[1], on continue normalement la résolution de Y ;
2. sinon, on pose la contrainte $X\neq Y$ avant de continuer la résolution.

Il est clair que la contrainte $X\neq Y$ ne peut être posée au début de la recherche, car elle interdirait tous les couples $(\sigma(X),\sigma(Y))$ tels que $\sigma(X)=\sigma(Y)$. C'est bien dynamiquement après l'unification de X, et en fonction du contexte (contraintes déjà existantes) que cette contrainte devra être posée. En fait, elle ne devra être ajoutée à l'ensemble courant de contraintes que dans le cas où la contrainte réciproque n'appartient pas déjà à cet ensemble.

3.2. Expression de la contrainte

L'exemple ci-dessus illustre le fait que nous voudrions poser une contrainte $\neq_{dyn}(X_1,X_2)$, qui étant données deux variables X_1 et X_2, signifie que dès que l'on unifiera la variable X_1 avec un terme, deux cas seront à envisager :

- si la contrainte $X_1=X_2$ appartient à l'ensemble courant de contraintes, on continue la résolution ;
- sinon, on pose la contrainte $X_1\neq X_2$ avant de continuer la résolution.

4. Application au TALN

4.1. Expression du problème

Nous allons visualiser le problème étudié sur un cas particulier relatif au TALN : la résolution des proformes enchâssées. Considérons la phrase suivante :

La femme qui regarde le miroir qu'elle tient vient.

Dans cette phrase, nous pouvons remarquer deux enchâssements inhérents à des propositions relatives. Nous pouvons dénombrer trois proformes :

- la femme qui regarde le miroir qu'elle tient ;

1. On différencie la propriété $X=Y$ et la contrainte $X=Y$. Par exemple, l'ensemble de contraintes $\{X=a,Y=a\}$ ne contient pas la contrainte $X=Y$, par contre, si les contraintes de cet ensemble sont vérifiées, la propriété $X=Y$ est alors vraie.

– le miroir qu'elle tient ;

– elle.

Nous ne nous occuperons pas de la seconde proforme (*le miroir qu'elle tient*) qui n'a pas de relation avec une autre proforme dans la phrase. Pour la proforme *elle*, il y a deux solutions possibles :

– soit on dit qu'il s'agit de *la femme qui regarde le miroir qu'elle tient*, quelle que soit l'instantiation ultérieure de cette femme ;

– soit on dit que c'est un autre individu identifié dans le contexte (par exemple *Marie* si *Marie* est apparue précédemment dans le discours), mais alors la résolution de *la femme qui regarde le miroir qu'elle tient* ne devra faire intervenir que des individus distincts de *Marie*. En effet, ne pas considérer cette contrainte génèrerait des redondances dans les solutions et ne traduirait pas l'exclusivité que l'on exprime naturellement.

4.2. Aspect algorithmique

Dans la phrase précédente (*La femme qui regarde le miroir qu'elle tient vient.*), nous considérons donc les deux proformes : *la femme qui regarde le miroir qu'elle tient*, proforme que nous noterons X, et *elle*, proforme que nous noterons Y. La résolution des proformes enchâssées s'effectue à partir de la plus imbriquée (Y). La contrainte que l'on désire poser signifie donc que juste après une opération d'identification de Y deux cas seront à envisager :

– si la contrainte $X = Y$ appartient à l'ensemble courant de contraintes, on continue normalement la résolution ;

– sinon, on pose la contrainte $X \neq Y$ et on continue la résolution.

Il s'agit donc exactement de la contrainte $\neq_{dyn} (X, Y)$ décrite en section 3.2.

4.3. Résolution en Prolog

4.3.1. Description

Chaque proforme est représentée par une métastructure. Dans le cas de proformes enchâssées, on établit un lien entre ces proformes par l'intermédiaire des attributs associés à ces métastructures. Ce lien serait bi-directionnel si l'ordre de résolution était inconnu. Dans notre cas, il est donc mono-directionnel (puisque nous résolvons comme indiqué en 4.2). Si nous reprenons l'exemple de notre phrase, à la métastructure Y qui représente le pronom *elle*, on va associer l'attribut *enchâssé_dans(X)*, X étant une métastructure représentant *la femme*.

Lorsqu'une contrainte d'égalité $Y = Z$ est posée sur Y, le prédicat utilisateur est alors appelé. Dans ce prédicat, on récupère l'attribut *enchâssé_dans(X)* de la métastructure Y et l'on considère deux cas :

– si *enchâssé_dans(Z)* est l'attribut (Z est donc la même variable que X), on pose l'égalité $Y = Z$ et on retire *enchâssé_dans(Z)* de l'ensemble des attributs de Y (Y n'aura donc plus d'attribut). En effet, ce cas représente celui où l'on est en train de poser la contrainte $Y = X$;

– sinon, on ajoute la contrainte d'inégalité entre Z et X (*enchâssé_dans(X)* étant l'attribut de Y), et on pose l'égalité $Y = Z$. Dans le cas où la contrainte $Y = X$ appartient déjà à l'ensemble courant de contraintes, grâce à la suppression évoquée ci-dessus, *enchâssé_dans(X)* ne sera plus l'attribut de Y. L'omission de cette suppression aurait conduit à un échec de cette opération : en effet, l'attribut de Y aurait été *enchâssé_dans(Y)* et on aurait alors posé l'inégalité $Y \neq Z$ avant de poser l'égalité $Y = Z$, ce qui aurait été évidemment insoluble.

Par opposition à l'exemple donné en section 2.2 où d'autres techniques peuvent être employées, le cas des contraintes contextuelles dynamiques peut difficilement se traiter d'une autre manière que par l'utilisation des métastructures.

4.3.2. Extrait de programme

L'extrait de programme Prolog suivant illustre la description qui précède. Il décrit le prédicat utilisateur $dynamicDif/3$, qui est appelé sur un essai d'unification d'une métastructure. Dans ce prédicat :

– on suppose que la relation entre la métastructure Y et son attribut (liste réduite à l'élément *enchâssé_dans(X)*) a été auparavant mise en place, par exemple au moyen de l'appel à un prédicat tel que *newMetastructure(Y, [enchâssé_dans(X)], dynamicDif)* ;

– Y représente la métastructure ;

– Z le terme avec lequel on veut unifier Y ;

– *[enchâssé_dans(X)]* l'attribut de Y (liste à un seul élément) ;

– le prédicat $enleverAttribut(Y)$ se contente de supprimer un élément à la liste représentant l'attribut de Y, donc de réduire cet attribut à la liste vide ([]) ;

– le prédicat prédéfini $metaUnify/2$ est le même que celui décrit en section 2.2.2 ;

– le prédicat prédéfini $dif(X, Z)$ pose la contrainte $X \neq Z$.

```
dynamicDif(Y,Z,[enchâssé_dans(X)]) :-
      Z == X,
      !,
      metaUnify(Y,Z),
      enleverAttribut(Y)
      .

dynamicDif(Y,Z,[enchâssé_dans(X)]) :-
      dif(X,Z),
      metaUnify(Y,Z),
      enleverAttribut(Y)
      .
```

Dans la première règle, on veut donc unifier la métastructure Y avec Z qui est l'élément X contenu dans l'élément unique de la liste représentant l'attribut de Y (Z == X).

Dans la seconde règle, on veut donc unifier Y avec un terme Z qui n'est pas X. La contrainte $dif(X, Z)$ étant mise en place, l'élément *enchâssé_dans(X)* n'est plus utile dans la liste représentant l'attribut de Y.

5. Conclusion

Cette méthode de résolution a été implémentée dans ILLICO (Pasero & Sabatier, 1994; Milhaud, 1994; Pasero & Sabatier, 1997) qui est un système d'analyse et de synthèse de phrases. En particulier, la synthèse peut être utilisée pour faire de la composition guidée. La grammaire d'ILLICO représente un noyau très significatif de la langue française (Pasero & Sabatier, 1996). Cette implémentation a permis une résolution très performante des proformes en évitant les redondances dans les solutions. Une perspective à moyen terme serait la généralisation de cette technique en spécifiant des prédicats de haut niveau dans les langages de Programmation par Contraintes. Ces prédicats permettront alors de poser naturellement des contraintes du type dont nous venons de parler, et donneront aux langages une nouvelle puissance d'expression.

Références

CORBLIN F. (1995). *Les formes de reprise dans le discours. Anaphores et chaînes de référence*. Rennes: Presses Universitaires de Rennes.

DALE R. (1989). Cooking up referring expressions. In *27th Annual Meeting of the association for Computational Linguistics*, Vancouver BC.

GARDE C. & LAÏ C. (1998). Utilisation des contraintes pour la suppression d'impasses dans les grammaires. *Revue d'Intelligence Artificielle*, **12**(4), 453–465.

GIANNESINI F., KANOUI H., PASERO R. & CANEGHEM M. V. (1985). *PROLOG*. InterEditions.

HOLZBAUR C. (1990). *Metastructures as a Basis for the Implementation of CLP Techniques*. Rapport interne, Austrian Research Institute for Artificial Intelligence, Vienna.

MILHAUD G. (1994). *Un environnement pour la composition de phrases assistée*. PhD thesis, Université de la Méditerranée, Marseille.

MOURET P. & ROLBERT M. (1998). Dealing with distinguishing descriptions in a guided composition system. In *Coling-ACL*, Montreal.

NEUMERKEL U. (1990). Extensible unification by metastructures. In *Meta 90*, p. 352–363, Leuven.

PASERO R. & SABATIER P. (1994). Composition de phrases assistée : Principes, outils et applications. In *Conférence TALN*, p. 51–74.

PASERO R. & SABATIER P. (1996). *GNF : Une grammaire noyau du français*. Rapport interne, Laboratoire d'Informatique de Marseille.

PASERO R. & SABATIER P. (1997). *Concurrent Processing for Sentences Analysis, Synthesis and Guided Composition*. Rapport interne, Laboratoire d'Informatique de Marseille. A paraître dans Natural Language Understanding and Computational Logic, G.P. Lopes, S. Mandanhar and W. Nutt (Eds), Lecture Notes in Computer Science, Springer, 1999.

PEREIRA F. & SHIEBER S. (1987). Prolog and natural language analysis. In *CSLI Lecture Notes #10*.