Compilation de la sémantique naturelle vers ML

Catherine Dubois, Richard Gayraud

LaMI, CNRS EP738, Université d'Évry Val d'Essonne
Bd. des Coquibus – F-91025 Évry Cedex FRANCE

`dubois@lami.univ-evry.fr`

Résumé

Nous présentons une méthode permettant de compiler des spécifications écrites en sémantique naturelle vers ML. Des permutations de prémisses permettent d'aboutir, dans certains cas, à une exécution purement fonctionnelle et peuvent réduire le non-déterminisme. Le schéma de compilation s'appuie sur des traits caractéristiques de ML, comme par exemple le filtrage, les exceptions et les fermetures.

1. Introduction

La sémantique naturelle [2, 7] est un formalisme destiné à spécifier la sémantique des langages de programmation. Une spécification décrit une ou plusieurs relations entre des arbres de syntaxe abstraite, et des valeurs sémantiques (évaluations, types, environnements). Par exemple, une relation sémantique de typage relie un environnement de type, une expression à typer et le type de cette expression. Ces relations, appelées aussi prédicats, sont définies par des règles d'inférence.

Les deux règles suivantes définissent la relation d'évaluation pour un petit langage d'expressions contenant uniquement des identificateurs et des additions. La formule $\rho \vdash_{eval} e : v$ se lit : "la valeur de e relativement au contexte ρ est v".

$$\frac{}{\rho \vdash_{eval} id(str) : \mathbf{find}(\rho, str)}$$

$$\frac{\rho \vdash_{eval} e_1 : v_1 \quad \rho \vdash_{eval} e_2 : v_2}{\rho \vdash_{eval} plus(e_1, e_2) : \mathbf{add}(v_1, v_2)}$$

Dans cet exemple, les fonctions **find** et **add** sont considérées comme des objets de base manipulant respectivement les contextes et les entiers.

On peut voir les règles d'inférence d'une spécification comme un programme logique. La sémantique naturelle devient alors un formalisme exécutable.

Le formalisme Typol, outil majeur de l'environnement CENTAUR, en fut la première implémentation. Il est basé sur une compilation assez naturelle vers Prolog, où chaque relation est transformée en un prédicat Prolog, et chaque règle d'inférence est traduite en une clause. La compilation vers un langage de programmation logique permet de mettre en oeuvre l'unification, le retour-arrière, et les instanciations diverses des variables. Par exemple, un système de typage peut servir à prouver un type, mais aussi à l'inférer, si la variable désignant le type est laissée libre au début de l'exécution. L'utilisation de Prolog se répercute malheureusement sur les performances, réduisant ainsi le champ d'applications de Typol à la réalisation de prototypes relativement peu efficaces.

Des implémentations plus récentes tentent d'y remédier. Minotaur [1] fait un rapprochement entre la sémantique naturelle et les grammaires attribuées, en utilisant le système FNC-2 comme cible pour la compilation de Typol. RML [9, 10] est une compilation directe vers le langage C, basée sur de nombreuses optimisations des différents niveaux de code intermédiaire.

Dans les deux cas, la possibilité d'utiliser la même spécification pour résoudre différents problèmes, selon l'instanciation initiale des variables, est perdue. Les termes instanciés et libres lors de l'utilisation de chaque règle sont distingués syntaxiquement.

Nous proposons une approche de compilation vers un langage fonctionnel à la ML, dont les idées principales sont les suivantes :

- Pour chaque règle d'inférence, un algorithme détermine un ordre d'exécution des prémisses afin d'éliminer les variables logiques au profit de variables "fonctionnelles". Ces dernières sont directement implémentables en ML. Elles ne supportent pas d'instanciation partielle ni d'unification. Ce même algorithme permet de préserver la faculté des relations d'être appelées selon différentes instanciations. Cette phase est appelée "orientation des relations" dans la suite.
- Les portions de spécification déterministes sont compilées selon un modèle plus efficace que celles qui nécessitent l'utilisation du retour-arrière.
- Des structures de contrôle propres à ML sont mises à profit :

le filtrage réalise la discrimination des règles à appliquer, et les expressions fonctionnelles permettent de représenter une forme de "continuations" propice à la compilation du retour-arrière.

Notre objectif dépasse en fait celui d'obtenir une compilation efficace de la sémantique naturelle en ML. Notre motivation à plus long terme est de pouvoir intégrer un module de sémantique naturelle à d'autres modules écrits en ML.

La section 4.3 donne une présentation informelle de l'algorithme d'orientation. La section 3 présente notre implémentation, baptisée Nsc, et son schéma de compilation vers Caml [8]. Il est pour l'instant limité aux variables fonctionnelles, mais il pourra être étendu pour accueillir des variables logiques. Les aspects théoriques sont abordés à la section 4. L'algorithme d'orientation y est formalisé et prouvé. Enfin, la section 5 compare sommairement les performances de Nsc et de Typol.

2. Orientation des relations

L'ensemble de notre approche repose principalement sur l'idée de permuter les prémisses des règles d'inférence afin d'aboutir à une exécution purement fonctionnelle de la spécification. Nous commençons par introduire les notions de direction et de direction cohérente.

2.1. Directions cohérentes

Une direction est un ensemble d'indices désignant des termes dans une formule. Elle est associée à une relation, mais cette dernière peut admettre plusieurs directions. Considérons par exemple les deux règles définissant une boucle *while* dans un langage impératif :

$$\frac{E \vdash_{eval} c : true \quad E \vdash_{exec} s : E' \quad E' \vdash_{exec} while(c, s) : E''}{E \vdash_{exec} while(c, s) : E''}$$

$$\frac{E \vdash_{eval} c : false}{E \vdash_{exec} while(c, s) : E}$$

La direction $\{1, 2\}$ sera dite cohérente pour la relation $\vdash_{exec}$ si il existe au moins un séquencement des prémisses permettant d'utiliser $\vdash_{exec}$ avec les termes 1 et 2 instanciés (ici l'environnement initial et l'arbre de syntaxe abstraite) ; la relation instanciera le terme de rang 3 (l'environnement final). La permutation choisie pour les prémisses doit permettre de n'utiliser que des variables libres ou complètement instanciées, c'est-à-dire déboucher sur une exécution fonctionnelle de la règle. Cette propriété de cohérence dépend des directions déjà connues pour les prémisses. Si on suppose que $\{1,2\}$ est cohérente pour $\vdash_{eval}$, alors la direction $\{1,2\}$ est cohérente pour la relation $\vdash_{exec}$.

L'algorithme d'inférence de ces directions cohérentes travaille sur l'ensemble des relations de la spécification, propageant les directions depuis les relations de base, aussi appelés relations importées, vers la relation principale, c'est-à-dire celle qui sera appelée depuis l'extérieur.

Le mécanisme permettant de décider si une direction est cohérente pour une règle, en connaissant les directions cohérentes pour les prémisses, est détaillé et formalisé dans la section 4.3. Intuitivement, pour une permutation donnée, chaque prémisse ne peut être appelée que lorsque les variables correspondant à l'une de ses directions cohérentes sont instanciées. Cette prémisse instancie alors d'autres variables, autorisant ainsi l'appel de nouvelles prémisses.

La principale difficulté apparaît lorsqu'il s'agit de traiter des relations récursives comme $\vdash_{exec}$. Du fait des appels récursifs, nous avons besoin des directions cohérentes pour $\vdash_{exec}$ (en tant que prémisse) afin de calculer les directions cohérentes pour $\vdash_{exec}$ (au dénominateur).

Cette situation amène à étudier toutes les hypothèses possibles de directions pour les prémisses concernées. Celles que l'on retrouve au dénominateur sont alors considérées comme vraies. C'est le cas de la direction $\{1,2\}$: sous l'hypothèse que $\{1,2\}$ est cohérente au numérateur, on infère qu'il est possible d'appeler $\vdash_{exec}$ avec la direction $\{1,2\}$. On dit alors simplement que $\{1,2\}$ est cohérente pour $\vdash_{exec}$.

En pratique, l'inférence de directions se fait sur un arbre représentant les appels des relations entre elles, comme sur la figure 1. Un noeud *et* représente une règle d'inférence, et a pour fils les relations des différentes prémisses. Un noeud *ou* rassemble les différentes règles d'une relation sémantique. Dans ce cas, les directions obtenues sont calculées par union des directions des fils : si une relation peut travailler avec une direction d, elle pourra a fortiori travailler avec une direction plus large, incluant d. Il en résulte que si deux règles possèdent chacune une direction distincte, alors on en retient l'union, qui est la plus petite direction cohérente pour les deux.

Parmi les nombreuses possibilités, on peut dégager ici deux intuitions différentes de la cohérence :

- Une première possibilité consisterait à traiter les hypothèses conjointement. Par exemple, au niveau du noeud while(2), on pourrait inférer les directions obtenues si le premier appel récursif se faisait avec la direction $\{2,3\}$ (h6), et le deuxième avec la direction $\{1,2\}$ (h3). Malheureusement, cette approche

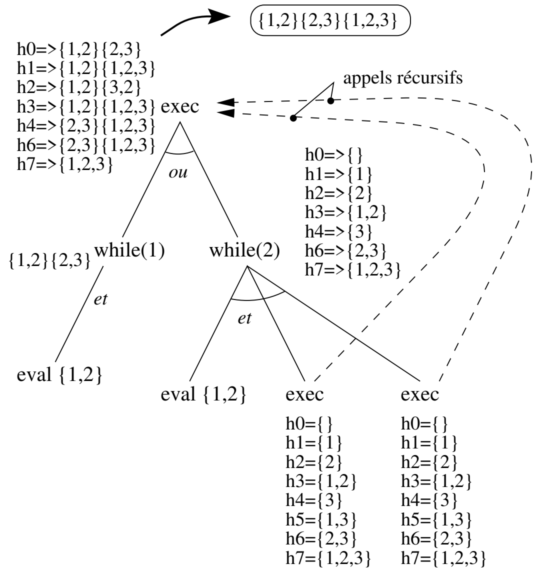


Figure 1 : Inférence de directions pour une relation récursive

engendrerait, à chaque noeud, un nombre d'hypothèses croissant de façon exponentielle.

- Un choix plus raisonnable, et suffisant en pratique, consiste à considérer les hypothèses séparément. C'est le cas sur la figure 1. On impose alors que tous les appels récursifs d'une relation se fassent selon la même direction, ce qui est généralement le cas. Les directions ainsi inférées sont dites "fortement cohérentes". La notion de cohérence forte est utilisée dans la suite et dans l'implémentation. L'algorithme d'inférence des directions fortement cohérentes est exponentiel, mais uniquement en fonction du nombre de termes de la relation et non plus du nombre d'appels récursifs. Cette restriction le rend utilisable : en effet, en sémantique naturelle, le nombre d'arguments est en général assez restreint.

En résumé, l'inférence des directions fortement cohérentes se fait de bas en haut de l'arbre des appels des relations sémantiques, en explorant séparément toutes les hypothèses possibles pour les relations récursives. On remarquera, sur la figure 1, la présence de la direction $\{2, 3\}$, qui permet de restituer l'environnement initial, à partir de l'environnement final et de l'arbre de syntaxe abstraite. Cette information est normale dans le cadre de cet exemple qui ne permet pas d'effets de bord. L'utilisation de la direction $\{2, 3\}$ pour *exec* provoquerait l'évaluation des prémisses de la droite vers la gauche.

Ces directions peuvent être considérées comme une sorte de typage des relations sémantiques, intégrant une forme de polymorphisme : dans une spécification, une relation peut être utilisée avec plusieurs directions. La contrainte de cohérence forte est semblable à celle du *let rec* de ML : une fonction récursive peut être appelée avec différents types, mais tous les appels récursifs sont typés de la même façon.

Notons finalement que notre inférence de direction est différente de l'inférence de mode de [3]. L'analyse de [3] est plus fine dans le sens où elle possède de nombreux modes et autorise l'instanciation partielle. En revanche, elle ne permute pas les prémisses, afin de conserver la sémantique gauche-droite de Prolog.

2.2. Réduction du non-déterminisme

Nous avons vu que la recherche des directions cohérentes, ou fortement cohérentes, conduit à explorer différentes permutations des prémisses de chaque règle. Il arrive couramment que plusieurs permutations aboutissent à une même direction. Lors de la génération de code, il est alors intéressant de choisir la meilleure, au sens d'un critère réduisant le non-déterminisme. En effet, la position d'une prémisse dans la permutation modifie le nombre de variables instanciées lors de l'appel, et par conséquent la direction d'appel.

En sémantique naturelle, le non-déterminisme apparaît lorsque deux règles sont applicables pour les termes connus. Il faut alors utiliser la première règle, puis mémoriser cette information de choix afin de pouvoir effectuer un retour-arrière si nécessaire.

Il est possible de prévoir statiquement où ces points de choix peuvent avoir lieu, ainsi que le nombre de tels conflits entre deux règles, pour une direction donnée. Il suffit pour cela d'unifier les termes des dénominateurs se correspondant entre les deux règles. Si tous les termes désignés par une direction d s'unifient deux à deux, alors il existe un jeu de données d'entrée qui peut provoquer un point de choix entre les règles concernées.

La section 4.3.1 formalise une analyse globale ascendante, appelée complexité, qui permet de connaître, pour une relation, les directions les plus déterministes, et donc les ordonnancements préférables des prémisses.

3. Compilation vers Caml

3.1. Principales caractéristiques de Nsc

Le langage Nsc permet de mélanger du Caml (directement incorporé dans le code cible), et des règles d'inférence définissant les relations sémantiques. Les structures de données manipulées par les règles peuvent être définies dans la partie Caml. En particulier, la syntaxe abstraite s'écrit à l'aide de types concrets.

Notre implémentation bénéficie de tout l'acquis de Caml : l'utilisateur peut utiliser Camllex et Camlyacc pour le "front-end" de son outil, le module de sémantique naturelle travaille alors directement sur l'arbre résultant de cette première étape.

En sémantique naturelle, l'ordre des prémisses n'a pas d'importance. Cette caractéristique est d'autant plus significative en Nsc, car le compilateur effectue des permutations pour la recherche de directions. Il appartient donc à l'utilisateur d'être prudent vis-à-vis des effets de bord éventuellement présents dans les fonctions Caml importées.

3.2. Schéma de compilation

La compilation vers Caml s'effectue en 2 passes principales :

1. Un premier traitement tente d'inférer toutes les directions fortement cohérentes pour chaque relation définie dans la spécification. L'utilisateur peut alors choisir d'utiliser une relation, avec une direction inférée.

2. En fonction des directions calculées à la première étape, et des ordonnancements des prémisses associés, un générateur de code produit une fonction Caml pour chaque relation et chaque direction utile au choix de l'utilisateur.

La phase de génération de code utilise un modèle d'exécution différent selon la complexité du couple (relation, direction). Ainsi, les relations purement déterministes pour une direction donnée sont compilées sous la forme d'un simple `match` Caml , comme le montre l'exemple suivant :

$$\frac{}{E \vdash Constant(v) : v}$$

$$\frac{}{E \vdash Var(i) : search(E, i)}$$

$$\frac{E \vdash e_1 : v_1 \quad E \vdash e_2 : v_2}{E \vdash Plus(e_1, e_2) : add\_int(v_1, v_2)}$$

$$\frac{E \vdash e_1 : v_1 \quad E \vdash e_2 : v_2}{E \vdash Minus(e_1, e_2) : sub\_int(v_1, v_2)}$$

$$\longrightarrow$$

```
...
match (nsc_var_1,nsc_var_2) with
    (e,Constant(v)) ->
      (v,nsc_retry)
  | (e,Var(i)) ->
      let v = search e i in
      (v,nsc_retry)
  | (e,PLUS(ex1,ex2)) ->
      let (v1,nsc_newretry)=
        nscuser_eval3((e,ex1),0) in
      let (v2,nsc_newretry)=
        nscuser_eval3((e,ex2),0) in
      ((add_int v1 v2),nsc_retry)
  | (e,MINUS(ex1,ex2)) ->
      let (v1,nsc_newretry)=
        nscuser_eval3((e,ex1),0) in
      let (v2,nsc_newretry)=
        nscuser_eval3((e,ex2),0) in
      ((sub_int v1 v2),nsc_retry)
  | _ -> raise N_sc_fail
...
```

Dans le cas où la complexité n'est pas nulle, alors il est nécessaire de gérer le retour-arrière potentiel dans le code produit.

Le mécanisme mis en oeuvre utilise les exceptions pour déclencher le retour-arrière. La sauvegarde d'état lors des points de choix est réalisée en utilisant des fermetures Caml, que nous appelons des "fonctions de rappel". Chaque résultat d'une relation non déterministe est accompagné d'une telle fonction. Lorsqu'elle est appliquée à l'argument `()`, elle poursuit la recherche pour fournir une nouvelle solution, également accompagnée d'une fonction de rappel. Nous montrons comment ces fonctions sont générées et composées entre elles sur un exemple : supposons qu'une relation q soit définie par deux règles aux dénominateurs identiques, et que p soit définie par deux appels à q. Le schéma suivant montre l'arbre de recherche lors de l'exécution. Le premier appel de q utilise la première règle. Le trajet en pointillés dénote le deuxième choix possible, c'est-à-dire la deuxième règle qui aurait pu être appelée. Il en résulte un couple composé d'une solution y_1, renvoyée par la première règle, et d'une fonction de rappel f_1, capable de poursuivre la recherche, si nécessaire, dans la deuxième.

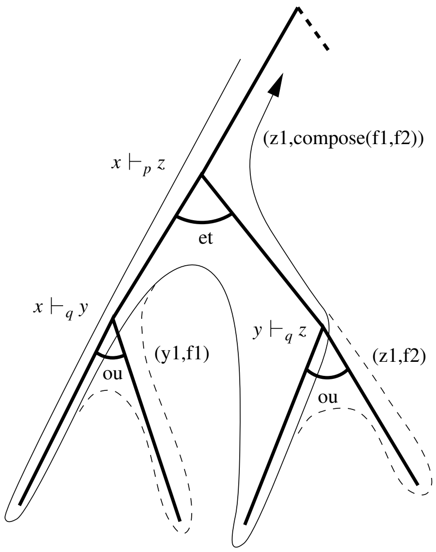
Le deuxième appel de q renvoie pour sa part un couple (z_1, f_2). Il ne peut pas être utilisé comme solution finale de l'appel $x \vdash_p z$. En effet, si le résultat z_1 est remis en cause plus tard, l'espace des solutions à explorer n'est pas caractérisé uniquement par f_2, mais par f_2 et f_1 : pour chaque résultat y_n de f_1, il existe plusieurs z_m.

Le compilateur génère donc, pour $x \vdash_p z$, du code produisant dynamiquement une fonction de rappel ici désignée par compose(f_1, f_2), qui est capable d'explorer toutes les solutions de z pour y_1, puis de calculer y_2, avant de chercher d'autres solutions pour z, et ainsi de suite.

$$\frac{\dots}{x \vdash_q y}$$

$$\frac{\dots}{x \vdash_q y} \qquad \rightarrow$$

$$\frac{x \vdash_q y \quad y \vdash_q z}{x \vdash_p z}$$



4. Aspects formels

Cette section constitue la partie théorique de notre approche. Nous y définissons la sémantique naturelle, les directions cohérentes et fortement cohérentes, ainsi que la complexité.

Nous y donnons ensuite une formalisation de l'algorithme d'inférence des directions fortement cohérentes, puis nous y énonçons les théorèmes de correction et de complétude, dont les preuves ont été omises par souci de concision. Ces dernières figurent dans [5].

4.1. Spécifications

Nous formalisons ici la sémantique naturelle de façon adaptée à notre schéma de compilation. On se donne, selon les définitions usuelles :

- Une signature $\Sigma = (S, F)$, où S est un ensemble de noms de sortes et F est un ensemble de symboles de fonctions, munis chacun d'une arité sur S de la forme $s_1 \times s_2 \times \cdots \times s_n \rightarrow s$.
- $V = \amalg_{s \in S} V_s$, un ensemble de symboles de variables, indexé par les sortes.

- $T(\Sigma, V)$, l'ensemble, défini inductivement, des termes sur Σ et V.
- N, un ensemble de noms de relations sémantiques.

Une **formule** est un couple $(nom, (t_1, \cdots, t_n))$, où $nom \in N$, et où les t_i sont des termes. Le **type** de cette formule est le couple $(nom, s_1 \cdots s_n)$, où s_i est la sorte du terme t_i.

Une **règle d'inférence** est de la forme $\frac{f_1 \cdots f_n}{f_0}$ où les f_i sont des formules. Le **type d'une règle** est le type de son dénominateur.

Soit $t = (nom, s_1, \cdots, s_n)$ un type (de formule, de règle) on appelle **direction sur le type** t un ensemble d'indices $d \subset [1, n]$.

Nous définissons un ordre partiel sur les directions : d **permet** d' si et seulement si $d \subseteq d'$.

L'ordre partiel $\subseteq$ caractérise le fait que si un prédicat peut travailler avec un ensemble de variables instanciées lors de l'appel, il pourra a fortiori travailler avec un ensemble plus grand.

Les relations sémantiques sont séparées en deux catégories :

- Une relation sémantique définie est un ensemble de règles d'inférence qui définissent une propriété entre les termes en s'appuyant sur d'autres relations.
- Une relation sémantique importée est un élément de base qui est considéré comme une boîte noire. C'est par exemple une fonction ML. Elle est munie d'une direction qui indique son mode de fonctionnement.

Une **relation sémantique définie** est un couple $p = (nom, R)$, où $nom \in N$ et $R = \{r_1, \cdots, r_n\}$ est un ensemble de règles de même type. Ce dernier, (nom, ts), est aussi le **type de la relation sémantique** p.

Une **relation sémantique importée** est un triplet (nom, t, dir), où $nom \in N$, t un type, et dir une direction sur t.

Une spécification est un ensemble de relations $\{p_1, \cdots, p_n\}$.

4.2. Contextes d'orientation

Nous avons vu qu'il est difficile de définir la cohérence d'une direction dans une spécification, car cette notion fait appel à une analyse globale. Nous pouvons par contre caractériser la cohérence d'un ensemble de directions pour une spécification : c'est ce que nous appellerons un contexte d'orientation.

Plus formellement, **un contexte d'orientation** C est une fonction qui associe à chaque relation p_i d'une spécification $Sp = \{p_1, \cdots, p_n\}$ un ensemble $C(p_i)$ de directions sur le type de p_i, et telle que si p_i est une relation importée (nom, t, dir), alors $C(p_i) = \{dir\}$.

Le contexte vide (pour toute relation) est dénoté par C_{vide}.
On dit que le contexte C est simple si :

$$\forall p \in Sp, Card(C(p)) \leq 1.$$

Les opérateurs ci-dessous sont utilisés pour construire ou modifier les contextes :
L'union des contextes C_1 et C_2, notée $C_1 \cup C_2$ est définie par :

$$\forall p \in Sp, (C_1 \cup C_2)\,(p) = C_1(p) \cup C_2(p).$$

Si D est un ensemble de directions sur p, le contexte modifié $C \oplus (p, D)$ est défini par :

$$\forall p' \in Sp, (C \oplus (p, D))(p') = \begin{cases} D \cup C(p') \; si \; p' = p, \\ C(p') \; sinon, \end{cases}$$

Le contexte modifié $C + (p, d)$ est défini par :

$$\forall p' \in Sp, (C + (p, D))(p') = \begin{cases} \{d\} \; si \; p' = p, \\ C(p') \; sinon, \end{cases}$$

Le contexte C_1 surchargé par le contexte C_2, noté $C_1 \dagger C_2$ est défini par :

$$\forall p \in Sp, (C_1 \dagger C_2)(p) = \begin{cases} C_1(p) \; si \; C_2(p) = \emptyset, \\ C_2(p) \; sinon. \end{cases}$$

4.3. Directions cohérentes

Les notations suivantes sont utilisées pour définir les directions cohérentes.

- **Sym(d,f)** est l'ensemble des symboles de variables de la formule **f**, qui apparaissent dans les termes désignés par la direction **d**. Par exemple : $Sym(\{1,2\}, E \vdash_{eval} plus(a,b) : v) = \{E, a, b\}$
- **Dir(sy,f)** désigne la direction obtenue à partir de la formule **f**, en considérant les symboles instanciés de **sy** (ensemble de symboles). Par exemple : $Dir(\{E, a, b\}, E \vdash_{eval} plus(a,b) : v) = \{1, 2\}$

- **Symboles(f)** est l'ensemble des symboles de variables de la formule **f**.
- **Rel(f)** désigne le prédicat utilisé par la formule **f**.

Nous introduisons dans la suite les notions de cohérence successivement pour les règles, les relations et les contextes d'orientation.

Une direction est cohérente pour une règle d'inférence s'il est possible d'ordonner les prémisses de manière à respecter l'instanciation des variables imposée par la direction, en utilisant uniquement des directions du contexte pour les appels de ces prémisses.

La définition repose sur des ensembles de symboles instanciés : sy_0 contient les symboles instanciés initialement, lors de l'appel de la règle, alors que sy_m contient les symboles instanciés après l'appel du m-ième prédicat de la règle.

Ainsi, on dit que d est **une direction cohérente pour la règle** $r = \frac{f_1 \cdots f_n}{f_0}$ dans le contexte C, si il existe une permutation $(f_{i_1}, \cdots, f_{i_n})$ du numérateur, et une suite d'ensembles de symboles $(sy_1, \cdots, sy_n)$ tels que :

1. $sy_0 = Sym(d, f_0)$
2. $\forall m_{/1 \leq m \leq n},\ sy_m = sy_{m-1} \cup Symboles(f_{i_m})$
3. $\forall m_{/1 \leq m \leq n},\ d_m = Dir(sy_{m-1}, f_{i_m}) \in C(Rel(f_{i_m}))$.
4. $Symboles(f_0) \subset sy_n$.

On dit aussi que d est une direction cohérente pour r dans C, selon l'**orientation** $((f_{i_1}, \cdots, f_{i_n}), (d_1, \cdots, d_n))$.

Une direction d est dite cohérente pour une relation sémantique définie p dans le contexte C si elle est cohérente pour toutes les règles de p dans le contexte C.
Si p est une relation importée (nom, t, d) alors d est cohérente pour p dans C si $d \in C(p)$.

Enfin, un contexte d'orientation C **est cohérent** pour une spécification $Sp = \{p_1, \cdots, p_n\}$, si $\forall p_i \in Sp$, $\forall d \in C(p_i)$, d est une direction cohérente pour la relation p_i, dans le contexte C.

4.3.1. Mesure du non-déterminisme potentiel

Nous définissons ici deux types de complexité qui reflètent le nombre de points de choix non déterministes, pour une relation sémantique p munie d'une direction d. La détection statique des points de choix se fait par unification des couples de termes désignés par la direction d dans les dénominateurs des règles.

La formule suivante permet de compter, pour une relation p et une direction d données, le nombre de conflits entre deux règles. On note $Cl_{p/d}$ ce nombre et on l'appelle la **complexité locale** de p selon d.
Si p est de la forme : $\left(nom, \frac{\cdots}{(nom,\{t_{1,1},\cdots,t_{1,n}\})}, \cdots, \frac{\cdots}{(nom,\{t_{k,1},\cdots,t_{k,n}\})}\right)$

$$Cl_{p/d} = \sum_{\substack{f_2 \in [1,k] \\ f_1 \in [1,k] \\ f_2 < f_1}} \prod_{s \in d} U(t_{f_1,s}, t_{f_2,s})$$

avec $U(t_1, t_2) = 1$ si t_1 et t_2 s'unifient, 0 sinon.

En général, on s'intéresse au temps total d'exécution d'un appel de relation, c'est-à-dire le temps utilisé par la relation elle-même et par ses appels à d'autres relations. C'est pourquoi, pour choisir entre plusieurs ordonnancements des prémisses de chaque règle, nous nous basons sur une notion plus globale de la complexité, que nous appelons complexité totale. Cette nouvelle définition tient compte de la complexité locale et, récursivement, de la complexité totale des prémisses de chaque règle.

La **complexité totale** $Ct_{p/d}(Sp)$ de p selon d dans Sp est définie par :

- Si p est une relation sémantique importée, $Ct_{p/d}(Sp) = 0$,
- Si p est une relation sémantique définie, alors pour chaque règle r_l de p, on note $((f_{l,1}, \cdots, f_{l,q})(d_{l,1}, \cdots, d_{l,q}))$ une orientation o_l telle que d est cohérente pour r_l dans C selon o_l.
  On a alors $Ct_{p/d}(Sp) = \mathcal{K} + Cl_{p/d}$ avec

$$\mathcal{K} = \underset{l}{Max} \sum_{i=1}^{q} Ct_{Rel(f_{l,i})/d_{l,i}}(Sp)$$
$$\text{sous l'hypothèse } Ct_{p/d}(Sp) = 0.$$

4.3.2. Restriction de la cohérence

Nous avons vu à la section 2.1 que l'algorithme d'inférence des directions cohérentes combinant toutes les hypothèses entre elles conduit à une explosion exponentielle du nombre d'hypothèses de chaque noeud. L'algorithme plus restrictif qui traite les hypothèses séparément utilise la cohérence forte, qui est définie ici.

On commence par définir la fonction rm qui ne garde que les directions cohérentes d'un contexte:

$$\forall p \in Sp, rm(C)(p) = \{d \in C(p) / \text{d est cohérente pour } p \text{ dans } C\}.$$

On note $\overline{C}$ un point fixe de la fonction rm . Le contexte $\overline{C}$ existe car il n'y a qu'un nombre fini de directions pour chaque relation ; $\overline{C}$ peut être égal à C_{vide}.

On dit qu'une direction d est **fortement cohérente** pour une relation p dans le contexte C, si

- $d \in \overline{C + (p, d)}(p)$,
- et si $\forall p' \neq p \ \forall d' \in \overline{C + (p, d)}(p')$, d' est fortement cohérente pour p' dans $\overline{C + (p, d)}$ (ainsi toutes les directions utilisées par d sont fortement cohérentes).

Remarque : la présence de $C + (p, d)$ dans la définition de la cohérence forte assure que toute utilisation de la relation p se fait avec la direction d.

Par conséquent, un contexte C est dit fortement cohérent si :
$\forall p \in Sp, \forall d \in C(p)$, d est fortement cohérente pour p dans C.

Un contexte C_1 est cohérent (resp. fortement) dans un contexte C_2 si :
$\forall p \in Sp, \forall d \in C_1(p)$, d est cohérente (resp. fortement) pour p dans C_2.

4.4. Formalisation de l'algorithme d'inférence des directions

Cette section donne une approche formelle de l'algorithme d'inférence des directions fortement cohérentes.

On note $\mathcal{A}$ l'algorithme récursif d'inférence des directions. Il prend comme paramètres une relation et un contexte simple, appelé contexte d'hypothèse, et retourne un arbre décoré par des directions. L'arbre décoré par l'ensemble de directions D et dont les sous-arbres sont $a_1, a_2 \ldots a_n$ est noté dans la suite $(D, (a_1, a_2 \ldots a_n))$.

L'algorithme $\mathcal{A}$ n'est pas déterministe, car le traitement séparé des hypothèses pour les relations récursives est représenté par le choix arbitraire d'une direction d. La figure 2 montre ce mécanisme. Pendant le développement de l'arbre, le premier noeud rencontré pour une relation est marqué par d arbitraire, qui est ajoutée au contexte d'hypothèse. Elle est ainsi propagée jusqu'aux appels récursifs. Les directions inferées sont alors conditionnées par ce choix ; elles ne seront validées qu'au niveau de ce même noeud, si elles contiennent au moins une direction permettant d.

Une exécution de $\mathcal{A}$ sur une spécification correspond donc à l'un des scénarios possibles, caractérisé par le choix d'une seule hypothèse pour chaque relation récursive. Pour obtenir toutes les directions fortement

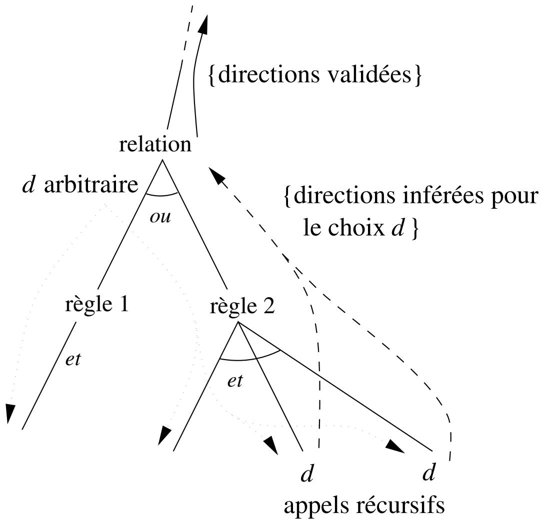


Figure 2 : Choix arbitraire et propagation d'une direction d

cohérentes, il faut considérer l'ensemble, fini, des solutions possibles de $\mathcal{A}$.

Cette formalisation influe sur les énoncés de correction et de complétude : la correction prouve que toutes les solutions de $\mathcal{A}$ sont cohérentes, alors que la complétude établit que pour une direction d fortement cohérente, il existe au moins un ensemble de choix arbitraires tels que $\mathcal{A}$ retourne une solution permettant d.

Soit $p \in Sp$ une relation et C_{hyp} un contexte simple. On définit $\mathcal{A}(p, C_{hyp})$ de la façon suivante :

- Si p est une relation importée (nom, t, d), alors

$$\mathcal{A}(p, C_{hyp}) = (\{d\}, ())$$

- Sinon, p est une relation définie dont les règles d'inférence sont $r_1, \cdots, r_m$.

  1. Si $C_{hyp}(p) \neq \emptyset$, (le contexte d'hypothèse contient une direction pour p, nous traitons donc une feuille de l'arbre et/ou) alors :

$$\mathcal{A}(p, C_{hyp}) = (C_{hyp}(p), ())$$

2. Sinon, soit d une direction choisie arbitrairement, et soit :

$$\begin{array}{rcl} (D_1, a_1) & = & \mathcal{A}(r_1, C_{hyp} + (p, d)) \\ & \vdots & \\ (D_m, a_m) & = & \mathcal{A}(r_m, C_{hyp} + (p, d)) \end{array}$$

   Soit :

$$D' = \{d'/\forall i \in \{1, \cdots, n\}, \exists d_i \in D_i/d_i \subseteq d'\}$$

   l'ensemble des directions permises par au moins une direction de chaque D_i. On note D l'ensemble défini par :

$$D = \begin{cases} D', \ si \ \exists d' \in D'/d' \subseteq d, \\ \emptyset, \ sinon. \end{cases}$$

   Le calcul de D formalise le fait que l'on ne conserve les directions inférées que si l'hypothèse d est valide, c'est-à-dire si l'on retrouve dans D' une direction d' qui permet d.
   $\mathcal{A}(p, C_{hyp})$ est alors défini par :

$$\mathcal{A}(p, C_{hyp}) = (D, (a_1, \cdots, a_m))$$

Remarques :

- Le contexte d'hypothèse C_{hyp} est vide lors du premier appel, et rempli progressivement par les directions accumulées lors des choix arbitraires. Il a deux utilités :

  1. Il empêche la construction d'un arbre infini pour les relations sémantiques récursives, grâce à une condition d'arrêt sur l'existence d'une direction de p dans C_{hyp} (cas *1.*).
  2. Etant marqué d'une direction arbitraire pour chaque développement d'un noeud, il garantit que toutes les feuilles d'un sous-arbre correspondant à une relation récursive seront marquées de la même direction (cohérence forte, voir figure 2).

- En pratique, le calcul de D' est différent. Il est implémenté de manière à conserver, avec le moins de directions possibles (c'est-à-dire en utilisant des directions les plus générales possibles, au sens de l'ordre partiel $\subseteq$), l'ensemble des directions qui sont cohérentes pour toutes les règles à la fois.

Soit r une règle d'inférence, et C_{hyp} un contexte simple. On définit maintenant $\mathcal{A}(r, C_{hyp})$, de la façon suivante :

- Si r est un axiome, alors :

$$\mathcal{A}(r, C_{hyp}) = (D, ())$$

où D est l'ensemble des directions cohérentes pour r dans C_{vide}.

- Sinon, $r = \frac{f_1, \cdots, f_m}{f_0}$
Soit :

$$\begin{aligned} (D_1, a_1) &= \mathcal{A}(rel(f_1), C_{hyp}) \\ &\vdots \\ (D_m, a_m) &= \mathcal{A}(rel(f_m), C_{hyp}) \end{aligned}$$

On définit alors C le contexte, défini sur les relations apparaissant dans les prémisses de la règle r, tel que $C(rel(f_i)) = D_i$.

Soit enfin D l'ensemble des directions cohérentes pour r dans C On pose alors :

$$\mathcal{A}(r, C_{hyp}) = (D, (a_1, \cdots, a_m))$$

L'arbre décoré d'une relation p selon un contexte vide contient les directions inférées pour toutes les relations utilisées par p, mais sur différents noeuds. Par ailleurs, nous avons vu que certains de ces noeuds peuvent contenir des directions invalides, car elles sont conditionnées par le contexte d'hypothèse. Le contexte inféré d'un arbre, défini maintenant, rassemble dans un contexte toutes les directions inférées et valides pour toutes les relations concernées.

Soit $\mathcal{A}(x, C_{hyp}) = a = (D, (a_1, \ldots a_k)$ un arbre décoré pour x (relation ou règle). Le **contexte inféré** de a, noté $CI(a)$, est défini par :

- Si a est l'arbre d'une règle r :

$$CI(a) = \begin{cases} C_{vide}, \text{ si } r \text{ est un axiome,} \\ \bigcup_i CI(a_i), \ sinon. \end{cases}$$

- Si a est l'arbre d'une relation p alors :

$$CI(a) = \begin{cases} C_{vide} \text{ si D est vide ou si } C_{hyp}(p) \neq \emptyset, \\ \left(\bigcup_i CI(a_i)\right) \oplus (p, D) \text{ sinon} \end{cases}$$

Finalement, on définit l'algorithme déterministe $\mathcal{A}_d$, qui retient toutes les solutions de $\mathcal{A}$ pour une relation donnée.

$$\mathcal{A}_d(p) = \bigcup_h CI(\mathcal{A}_h(p, C_{vide}))$$

$\mathcal{A}_h(p, C_{vide})$ dénote ici le développement pour le choix de la direction h pour p.

4.5. Correction et complétude

La correction de $\mathcal{A}$ consiste à prouver que les directions inférées par l'algorithme sont effectivement cohérentes.

Théorème 1 *Soit Sp une spécification, p une relation de Sp, et C_{hyp} un contexte simple. Alors le contexte $CI(\mathcal{A}(p, C_{hyp}))$ est cohérent dans le contexte:*

$$CI(\mathcal{A}(p, C_{hyp})) \cup C_{hyp}.$$

Ce théorème établit la correction de $\mathcal{A}$. En effet, l'appel initial de l'algorithme est fait avec $C_{hyp} = C_{vide}$, donc si $CI(\mathcal{A}(p, C_{hyp}))$ est cohérent dans lui-même, il est cohérent.

Les directions *fortement* cohérentes ont été définies pour caractériser la différence entre les directions que l'algorithme est capable de calculer et les autres. Il est donc nécessaire de prouver que toutes les directions fortement cohérentes sont inférées par $\mathcal{A}$.

Théorème 2 *Soit Sp une spécification, p une relation de Sp, C un contexte, C_{hyp} un contexte simple, et d une direction de $(C \dagger C_{hyp})(p)$. Si $C \dagger C_{hyp}$ est fortement cohérent, alors il existe un arbre décoré $a = \mathcal{A}(p, C_{hyp})$ tel que :*

$$\exists d' \in (CI(a))(p) / d' \subseteq d.$$

Ce théorème prouve la complétude de l'algorithme. En effet, si C est fortement cohérent, et si $d \in C(p)$, il suffit de prendre $C_{hyp} = C_{vide}$, pour réunir les conditions du théorème. Nous connaissons alors l'existence de $a = \mathcal{A}(p, C_{hyp})$, dont le contexte inféré contient une direction qui permet d. Comme dans le cas de la correction, cet argument s'appuie sur le fait que le premier appel de l'algorithme est fait avec C_{vide}.

5. Quelques comparaisons de performances

La figure 3 montre des comparaisons effectuées entre Typol et Nsc. La version de Typol utilisée est celle qui accompagne CENTAUR 2.0. Le code Prolog y est compilé avec Eclipse. Pour Nsc, le code Caml est compilé avec le compilateur natif de Objective Caml. Les tests ont été effectués sur une machine Sun Sparc 10/51, sous SunOS 4.1.3.

Le premier graphe fait apparaître le temps d'exécution pour un nombre variable d'itérations dans une boucle d'un langage impératif. Il met en évidence les performances des appels récursifs, et de la gestion du retour-arrière.

Le deuxième mesure les temps d'accès pour un grand nombre de variables dans un environnement. Dans le cas de Nsc, cette opération est effectuée par un parcours de liste directement écrit en Caml.

Enfin, la dernière courbe concerne l'évaluation déterministe d'une expression mathématique, en fonction du nombre de termes mis en jeu.

6. Unification et sémantique naturelle

La principale limitation actuelle de Nsc est l'absence de variables logiques et d'unification, interdisant l'implémentation de systèmes de types comme celui de Damas-Milner [2].

Le formalisme Typol traite aisément ces aspects logiques puisque ce sont des traits caractéristiques de son langage cible. Le compilateur RML incorpore également la notion de variables logiques. En revanche, l'approche proposée par Attali et Parigot [1] ne supporte pas toutes les formes d'unification.

Il est possible d'inclure les variables logiques et l'unification dans notre modèle d'exécution, cependant il est difficile d'inférer où ces variables logiques doivent être introduites. En effet, l'inférence de directions procède de bas en haut, et ce n'est qu'à la fin du processus que l'on sait si la direction souhaitée par l'utilisateur est fortement cohérente. Si ce n'est pas le cas, nous n'avons pas d'information permettant de désigner quelles variables devraient être considérées comme logiques pour inférer la direction voulue.

Une solution simple consisterait à indiquer explicitement dans la spécification les variables susceptibles de recevoir une instanciation partielle. Durant le calcul des directions, on pourrait alors considérer les termes correspondant à la fois comme des données et des résultats. On pourrait aussi envisager des analyses plus fines permettant d'approcher l'état d'instanciation des variables comme dans [3].

D'autre part, il n'est pas rare que la spécification requiert une procédure d'unification spécifique (par exemple dans certains travaux autour du typage). Une dernière perspective intéressante serait alors de rendre la compilation de la sémantique naturelle indépendante de l'algorithme d'unification.

7. Conclusion

Dans cet article, nous montrons comment réordonner les prémisses des règles de sémantique naturelle de manière à garantir que toutes les variables sont complètement instanciées lors de leur utilisation. Nous basons notre approche sur la notion de direction et proposons une méthode générale permettant d'utiliser les règles d'inférence avec différents *modes*, même si les spécifieurs écrivent souvent les prémisses d'une règle dans le *bon ordre*.

Nous avons réalisé un prototype de notre compilateur en Ocaml. Bien que les performances soient encourageantes comparativement à Typol, notre implémentation n'utilise que l'inférence de directions comme optimisation. Il est possible d'y intégrer de nombreuses autres techniques décrites dans [10]. En particulier, les méthodes de factorisation des prémisses permettent d'éliminer complètement le non-déterminisme artificiel contenu dans une relation du type *while*.

Références

[1] Attali I., Parigot D. "Integrating Natural Semantics and attribute grammars : the Minotaur system", INRIA research report 2339, 1994, ftp : ftp.inria.fr.

[2] Clément D., Despeyroux J., Despeyroux T., Kahn G. "A simple applicative language : Mini-ML", Proc. of the ACM conference on lisp and functional programming, 1986.

[3] Debray S. K. "Static inference of modes and data dependencies in logic programs", in ACM TOPLAS, Vol 11, No 3, 1989.

[4] Despeyroux T. "Typol : a formalism to implement Natural Semantics", INRIA research report 94, 1988.

[5] Richard Gayraud, "Compilation de la sémantique naturelle en ML", rapport de DEA, Université d'Evry, 1996.

[6] Jacobs I. "The CENTAUR 1.2 Manual", INRIA, 1992.

[7] Kahn G. "Natural Semantics", Proc. of Symp. on theorical aspects of computer science, Passau, Germany, LNCS 247,1987.

[8] Weis P., Leroy X. "Le langage Caml", InterEditions, Paris, 1993.

[9] Petterson M. "A compiler for Natural Semantics", Proc. of CC'96, Linköping, 1996.

[10] Petterson M. "Compiling Natural Semantics", Phd Thesis, Linköping university, 1995, ftp : ftp.ida.liu.se.

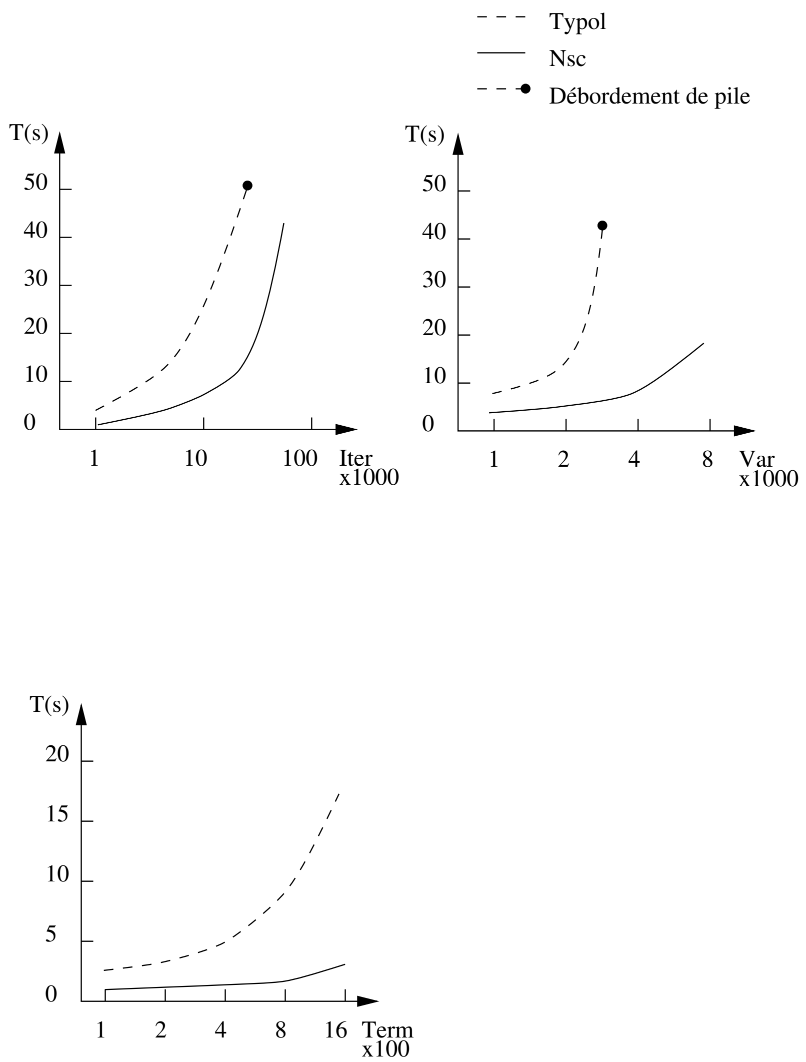


Figure 3 : Comparaison de performances entre Centaur et Nsc.