Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
GLO-66811

Danny Dubé
Automne 2008
Version: 18 novembre

Travail pratique #4
Analyse statique de types

Problème

Vous devez implanter la 3ème analyse de types vue en classe. Il s'agit de l'analyse qui tient compte des trois types de données et qui traite les formes `if` et `pair?` soigneusement.

Comme l'intérêt de ce travail consiste plus en l'implantation de l'analyse qu'en l'implantation d'un *front-end* de compilateur[1], les programmes à analyser vous sont fournis sous une forme pré-digérée plutôt que sous leur aspect normal. Les résultats attendus de votre analyseur sont le contenu des ensembles α_l (pour les expressions) et α_x (pour les variables).

Format d'entrée des programmes

Par souci de simplicité, les expressions du programme à analyser sont décrites une à une, chacune sur une ligne. La description d'une expression est une liste qui contient tout d'abord le mot-clé correspondant à cette expression, l'étiquette de l'expression et, le cas échéant, divers constituants supplémentaires de l'expression. Comme il y a 10 sortes d'expressions dans notre mini-langage, il y a 10 mots-clés différents: `false`, `ref`, `lambda`, `call`, `if`, `mu`, `cons`, `car`, `cdr` et `pair?`. Une étiquette est un simple numéro: 1, 2, ..., l_{max}, où l_{max} est l'étiquette ayant le plus grand numéro. Un constituant est soit l'étiquette d'une autre expression, soit le numéro d'une variable. Les variables sont aussi représentées par des numéros (au lieu de noms). Les variables ont les numéros suivants: $l_{max} + 1$, $l_{max} + 2$, ... Les constituants supplémentaires de chacune des sortes d'expressions sont:

- `false`: aucun constituant supplémentaire;
- `ref`: le numéro de la variable;
- `lambda`: le numéro de la variable et l'étiquette du corps;
- `call`: les étiquettes des deux sous-expressions;
- `if`: les étiquettes des trois sous-expressions;
- `mu`: le numéro de la variable et l'étiquette de la sous-expression;
- `cons`: les étiquettes des deux sous-expressions;

[1] Partie du compilateur qui effectue la lecture, l'analyse lexicale et l'analyse syntaxique des programmes afin de produire un arbre de syntaxe du programme.

- `car`: l'étiquette de la sous-expression;
- `cdr`: l'étiquette de la sous-expression;
- `pair?`: l'étiquette de la sous-expression.

Afin de rendre le chargement du programme plus simple encore, la toute première ligne du format d'entrée contient deux nombres: le nombre total d'expressions et le nombre total de variables. Ainsi, il vous est possible de préparer les structures de données servant à contenir le programme avant de le charger. Aussi, en sachant le nombre d'expressions, vous pouvez lire le programme à l'aide d'une boucle sans même avoir à vous préoccuper de la fin de fichier.

Notez que les expressions du programme sont énumérées dans un ordre correspondant à un parcours en pré-ordre à travers le programme. C'est-à-dire que pour décrire une partie du programme, on décrit d'abord l'expression qui est à sa tête, ensuite on décrit l'éventuelle première sous-expression, ensuite on décrit l'éventuelle deuxième sous-expression et enfin on décrit l'éventuelle troisième sous-expression. L'étiquetage des expressions est fait à l'aide d'un parcours en pré-ordre aussi, comme cela a toujours été le cas dans nos exemples en classe. Ceci fait en sorte que les étiquettes des expressions dans le format d'entrée sont ordonnées de la plus petite à la plus grande (à partir du numéro 1). D'ailleurs, à cause de l'ordre d'énumération des expressions et des étiquettes, le format d'entrée aurait pu complètement omettre les étiquettes des expressions. Elles sont indiquées tout de même afin de simplifier la lecture.

Votre programme doit fournir la fonction `analyze` qui reçoit deux noms de fichiers sous forme de chaînes de caractères: le fichier contenant le programme et le fichier devant recevoir les résultats d'analyse. Vous n'avez pas à écrire de code servant à vérifier la validité du programme en entrée. Les programmes utilisés pour la correction seront toujours valides.

À titre d'exemple, considérons le programme suivant, lequel est présenté dans les notes de cours:

$$
\begin{array}{l}
({}_1(\lambda_2 \mathrm{f}.\ (\mathrm{if}_3\ \#f_4 \\
\qquad\qquad ({}_5\mathrm{f}_6\ \#f_7) \\
\qquad\qquad ({}_8\mathrm{f}_9\ (\mathrm{cons}_{10}\ \#f_{11}\ \#f_{12})))) \\
\quad (\lambda_{13}\mathrm{x}.\ (\mathrm{car}_{14}\ \mathrm{x}_{15})))
\end{array}
$$

Celui-ci a l'aspect suivant lorsqu'il est décrit selon le format d'entrée décrit plus haut:

```
15 2
(call 1 2 13)
(lambda 2 16 3)
(if 3 4 5 8)
(false 4)
(call 5 6 7)
(ref 6 16)
(false 7)
(call 8 9 10)
```

```
(ref 9 16)
(cons 10 11 12)
(false 11)
(false 12)
(lambda 13 17 14)
(car 14 15)
(ref 15 17)
```

Format de sortie des résultats

Lorsque votre programme a complété l'analyse de types du programme reçu, il doit imprimer les résultats d'analyse. Notez que votre programme doit produire les résultats d'analyse correspondant à *la plus petite solution possible* aux contraintes.

Les résultats imprimés doivent d'abord indiquer la valeur des ensembles α_l et ensuite le contenu des ensembles α_{x}. Les ensembles α_l et leur contenu doivent être énumérées dans l'ordre, un ensemble par ligne. Similairement pour les ensembles α_{x}, selon l'ordre de numérotation des variables.

Les résultats pour α_l (ou α_{x}) doivent être indiqués comme suit: `(`⟨no⟩ ⟨abs⟩ ...`)`, où ⟨no⟩ est le numéro de l'expression ou de la variable et ⟨abs⟩ est une valeur abstraite. Il peut y avoir aucune, une ou plusieurs valeurs abstraites. Une donnée abstraite est soit le booléen faux, `(false)`, une fonction provenant d'une λ-expression e_l, `(lambda` l`)`, ou une paire provenant d'une `cons`-expression e_l, `(cons` l`)`.

Les résultats d'analyse produits pour le programme utilisé dans l'exemple de la section précédente devraient avoir l'air de:

```
(1 (false))
(2 (lambda 2))
(3 (false))
(4 (false))
(5 (false))
(6 (lambda 13))
(7 (false))
(8 (false))
(9 (lambda 13))
(10 (cons 10))
(11 (false))
(12 (false))
(13 (lambda 13))
(14 (false))
(15 (false) (cons 10))
(16 (lambda 13))
(17 (false) (cons 10))
```

L'ordre des éléments dans les ensembles n'est pas important.

Choix d'implantation et rapport

Vous pouvez choisir la façon d'implanter votre programme à votre guise. Il y a de nombreux choix d'implantation à faire. Par exemple: le fait que les contraintes entres les variables soient générées explicitement ou non; l'implantation des ensembles de données abstraites; le stockage du programme à analyser; etc.

Tous vos choix doivent être bien expliqués dans un rapport à remettre indépendamment.

Bien que j'aie mentionné en classe que ce genre d'analyse s'exécute en temps $O(n^3)$, vous n'êtes pas forcé de faire une telle implantation. Vous pouvez vous contenter de faire une implantation moins efficace. Cependant, j'accorderai jusqu'à 10% de bonus pour une implantation qui a une complexité de $O(n^3)$. Ce doit être *clairement* expliqué dans le rapport que vous avez réalisé une telle implantation, comment vous l'avez faite et le tout doit être facilement vérifiable dans votre code. Le bonus ne pourra pas faire dépasser la note globale du travail pratique au-dessus de 100%.

Remise des travaux

Vous devez remettre le devoir au plus tard le 12 décembre **via l'intranet**. Vous devez remettre votre implantation de l'analyseur ainsi qu'un rapport décrivant votre implantation. Le travail doit être fait **individuellement**. Des discussions verbales sont acceptables entre camarades mais pas des échanges de bouts de code.