Compilation
Année 2006-2007
4 avril

Florent Bouchez
Christoph Lauter

Partiel de Compilation

1 Saoûls comme des polonais

Le but de cet exercice est de donner le code `ocamllex` et `ocamlyacc` pour une calculette en notation polonaise inverse.

La notation polonaise inverse ou bien notation postfixée est une façon d'écrire des expressions avec un parenthésage implicite. Bien qu'inventée très tôt et employée anecdotiquement dans les calculettes des années 70, elle reste vivante dans plusieurs langages comme par exemple le Postscript.

Pour écrire une expression faisant appel à un opérateur $f : D^n \to D^m$, on donne d'abord n expressions qui deviendront les opérandes de l'opérateur f donné ensuite. Son évaluation produira m valeurs qui pourront servir comme arguments au prochain opérateur. Voici quelques exemples avec leur traduction en notation infixée habituelle :

$$\begin{aligned} 4\ 5\ + &\sim 4 + 5 \\ 4\ 5 + \exp &\sim \exp(4 + 5) \\ 4\ 5\ + 6\ 3\ + \times &\sim (4 + 5) \times (6 + 3) \end{aligned}$$

Exercice 1 : Parfaitement à jeûn, je la vois qui bascule[1]

Concentrons-nous d'abord sur l'analyse lexicale, en particulier sur l'analyse de nombres réels écrits en notation scientifique. Un nombre en notation scientifique est composé d'un signe s + ou - optionnel et d'une partie entière n, optionnellement suivi d'un point `.` et d'une partie fractionnelle f auxquels se rajoute optionnellement un facteur de mise à l'échelle. Le facteur de mise à l'échelle se compose de la lettre `e`, d'un signe s' + ou - optionnel et d'un entier E. La valeur associée à un tel nombre est $s \cdot (n + f) \cdot 10^{s' \cdot E}$. À titre d'illustration, voici quelques exemples :

```
4.3
-5.2e-5
+12.07
-0.0e-0
3e10
```

Question 1.1. Montrez que le langage formé par les nombres écrit en notation scientifique est régulier en donnant une expression regulière qui l'engendre.

Question 1.2. Donnez un automate fini qui reconnaît le même langage. Minimisez ensuite votre automate pour qu'il ait le nombre minimal d'états. Servez-vous du procédé vu en cours.

Question 1.3. En compilation, on se sert d'automates finis pour l'analyse lexicale. Quelle est la fonctionalité supplémentaire que doit implémenter un analyseur lexical comme celui produit par `ocamllex` ? Traditionellement elle n'est pas présente dans la représentation théorique d'automates finis. Pensez aux "valeurs renvoyées" par l'automate et au fait qu'il y a reconnaissance d'une suite de lexèmes.

[1] la virgule flottante

Exercice 2 : Analyse syntaxique des expressions

Au premier abord, le langage des expressions en notation polonaise inverse ressemble à une suite finie de constantes et d'opérateurs. Montrez que cette suite forme un langage régulier. Convainquez ensuite le correcteur que le langage des expressions en notation polonaise inverse ne forme pas un langage régulier. Donnez des exemples de suites reconnues par un automate fini qui ne sont pas des expressions bien-formées du langage. Quelle est la structure de données dont vous avez besoin en plus pour ne reconnaître que les expressions bien-formées ? Faudra-t-il sortir les gros moyens comme `ocamlyacc` pour notre calculette ?

Exercice 3 : Germination des graines de calculettes

Remplissez les lignes marquées `A COMPLETER` pour compléter le fichier `ocamllex` de la feuille annexe. La fonction `evaluate` analysera une chaîne de caractères prise en argument, évaluera l'expression en notation polonaise inverse et renverra le ou les résultat(s) du calcul. Si une chaîne de caractère ne représente pas une expression correcte, une exception sera levée.

2 Mmmh ? heu... où suis-je ?

Exercice 4 : Bien gérer ses petites cases

Question 4.1. Pour chacune des variables a, b, c, d, e de la fonction `dummy` suivante, écrite en Pascal, la garderiez vous dans un registre ou en mémoire ? Justifiez vos choix.

```
function dummy(a : integer; var b : integer) :integer;
var c : array[1..3] of integer;
  d, e : integer;
begin
  d := a + b;
  e := stupid(c, b);
  return (e + c[d]);
end;
```

```
function stupid(x : array of integer; var y : integer) : integer;
...
```

Note : les fonctions sont similaires aux procédures, mais ont une valeur de retour dont le type est défini par la première ligne de la déclaration, par un « `: <type> ;` » après les arguments de la fonction et dont la valeur est fixée par l'instruction `return`.

Exercice 5 : Les rechargables sont meilleures pour l'environnement !

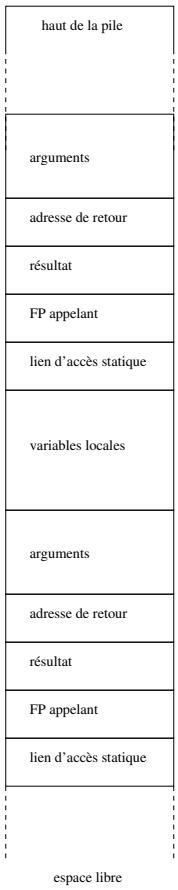
On utilisera la représentation des *cadres* de la pile présentée sur la partie gauche de la figure 1. À titre de rappel, le *FP appelant* désigne le pointeur vers le début du cadre de la fonction appelante, et le *lien d'accès statique* est un pointeur vers le cadre de la fonction lexicalement englobante.

Question 5.1. La partie gauche de la figure 1 représente l'état d'une partie de la pile pendant l'exécution d'un programme. Identifiez sur cette figure un cadre complet (ensemble créé par une unique fonction) et justifiez votre choix.

Question 5.2. Le programme `frames` (partie droite de la figure 1) est exécuté. Dessinez l'état de la pile lors de l'affichage de x (ligne 18 :`write(x)`). En particulier, vous indiquerez vers où pointent les liens « cadre de l'appelant » et « lien d'accès statique ».

Question 5.3. Lors de l'appel à `A` par `C`, le lien d'accès statique de `A` doit pointer vers le cadre du programme principal `frames`. Dans quelle fonction se trouvera le code assembleur qui calculera cette adresse ? Donnez une version possible de ce code.

Note : utilisez une syntaxe « à la C », avec des instructions `load` et `store` (pour charger et sauvegarder en mémoire). Les registres seront notés `r1`, `r2`, ...




```
program frames;

var x : integer;

procedure A(i : integer);
  procedure B(i : integer);
    procedure C(i : integer);
    begin
      x := x + 1;
      A(i-1);                          10
    end;
  begin
    C(i);
  end;
begin
  if (i <= 0)
  then
    write(x)
  else
    B(i);                              20
end;

begin
  A(1);
end
.
```

FIG. 1 – Un bout de pile et un programme

Question 5.4. On suppose que la variable x est stockée dans la pile. Quel serait le code généré pour exécuter l'instruction de la ligne 9 : `x := x + 1` ?

Exercice 6 : Le p'tit commerce de la mémoire

Certains processeurs ont des contraintes très spécifiques. Dans cet exercice, on en considère un qui ne peut charger ou sauvegarder en mémoire que si l'adresse est dans un registre spécifique : le *registre d'adresse*, noté r_a. Par contre, celui-ci possède des version avec auto-incrément (ou auto-décrément) des instructions standard *load* et *store*. On considérera qu'il n'y a que deux registres : r_a et r. Les instructions disponibles sont les suivantes :

Instruction	**Effet**
$r_a \leftarrow val$	charge une valeur dans le registre d'adresse
$r \leftarrow *(r_a)$	charge dans r la valeur mémoire pointée par r_a
$r \leftarrow *(r_a{+}{+})$	idem, puis fait pointer r_a sur l'adresse suivante
$r \leftarrow *(r_a{-}{-})$	idem, puis fait pointer r_a sur l'adresse précédente
$*(r_a) \leftarrow r$	sauvegarde la valeur dans r dans la mémoire à l'adresse r_a
$*(r_a{+}{+}) \leftarrow r$	idem, puis fait pointer r_a sur l'adresse suivante
$*(r_a{-}{-}) \leftarrow r$	idem, puis fait pointer r_a sur l'adresse précédente
$r \leftarrow r + *(r_a)$	incrémente r de la valeur mémoire pointée par r_a
$r \leftarrow r + *(r_a{+}{+})$	idem, puis fait pointer r_a sur l'adresse suivante
$r \leftarrow r + *(r_a{-}{-})$	idem, puis fait pointer r_a sur l'adresse précédente

Exemple : assigner b à a puis charger c dans le registre r :

a, b, c quelconques			a, b, c consécutifs dans la pile		
r_a	$\leftarrow$	$@a$	r_a	$\leftarrow$	$@a$
r	$\leftarrow$	$*(r_a)$	r	$\leftarrow$	$*(r_a\text{- -})$
r_a	$\leftarrow$	$@b$	$*(r_a\text{- -})$	$\leftarrow$	r
$*(r_a)$	$\leftarrow$	r	r	$\leftarrow$	$*(r_a)$
r_a	$\leftarrow$	$@c$			
r	$\leftarrow$	$*(r_a)$			

Pour simplifier, on supposera que chaque opération coûte 1. Pour l'exemple ci-dessus, il est donc possible d'économiser un tiers des opérations par un arrangement judicieux des données.

Le problème général s'énonce alors comme suit : étant donnée une suite d'opérations, placer les variables utilisées dans la pile de manière à minimiser le nombre total d'instructions nécessaires pour réaliser ces opérations.

On utilisera par la suite l'exemple d'opérations suivant :

$$\begin{array}{lcl} c & \leftarrow & a+b \\ f & \leftarrow & d+e \\ a & \leftarrow & a+d \\ c & \leftarrow & d+a \\ b & \leftarrow & d+f+a \end{array}$$

La *trace* des accès aux variables est la liste des accès aux variables dans l'ordre d'exécution du programme ; par exemple $a, b, c, a, \ldots$

Question 6.1. On considère que dans l'opération $x + y$, on accède à x avant y. Quelle est alors la trace des accès aux variables pour l'exemple ci-dessus ?

Soit un placement en mémoire de n variables. Une trace $T = x_1, x_2, \ldots, x_m$ a un coût $c(T) = \sum_{i=1}^{m-1} N(x_i, x_{i+1})$ où $N(x, y)$ vaut 0 si x et y sont voisins en mémoire, 1 sinon.

Question 6.2. On cherche à trouver le placement en mémoire qui minimise $c(T)$. Proposez une modélisation de ce problème à l'aide d'un graphe et trouvez une solution optimale pour l'exemple.

Question (difficile?) 6.3. Trouvez un algorithme polynomial optimal (à prouver) pour ce problème ou prouvez qu'il est NP-complet.

On considère maintenant que le placement en mémoire est fixe. En revanche l'ordre dans lequel sont effectuées les additions d'une opération est à la discrétion du compilateur.

Question 6.4. Montrer que pour un ensemble d'opérations donné, les différentes traces possibles sont exactement de la forme $x_1, Y_1, x_2, Y_2, \cdots Y_{k-1}, x_k$, avec x_i variables fixées et les Y_i sont des séquences de variables où toute permutation est possible.

Montrer que l'on peut se ramener au cas où pour tout i, pour tout j, x_j apparaît au plus une fois dans Y_i sauf x_i et x_{i+1} qui n'y apparaissent pas.

Question 6.5. Trouvez un algorithme polynomial optimal (à prouver) pour ce problème ou prouvez qu'il est NP-complet.

Question (difficile?) 6.6. Que se passe-t'il si on s'autorise à modifier l'ordre des additions dans une opération *avant* de décider le placement en mémoire ? Faites des propositions.