Correction du Partiel CMP1-TYLA

EPITA – Promo 2008 – **Aucun document autorisé**

Avril 2006 (1h30)

Le sujet et sa correction ont été écrits par Akim Demaille.

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante.

1 Incontournables

Il n'est pas admissible d'échouer sur une des questions suivantes : **chacune induit une pénalité sur la note finale**.

1. Quelle est le type (de Chomsky) du langage engendré par $S \to aX \quad S \to b \quad X \to Sc$

   Correction: Ce langage est $a^n b^n$ bien connu pour être hors-contexte, et non rationnel.

2. Qu'est-ce que Lex ?

   Correction: Un générateur de scanners.

3. Qu'est-ce que Yacc ?

   Correction: Un générateur de parsers.

2 Typologie des Langages

1. Apparier chaque auteur avec son langage :

   a. Alan Kay
   b. Andrew Appel
   c. Bjarne Stroustrup
   d. Denis Ritchie
   e. Jean Ichbiah
   f. John Backus
   g. John McCarthy
   h. Kristen Nygaard
   i. Niklaus Wirth
   j. Ole-Johan Dahl

   1. Ada 83
   2. C
   3. C++
   4. FORTRAN
   5. Lisp
   6. Pascal
   7. Simula
   8. Smalltalk
   9. Tiger

Correction:

Who	What
Alan Kay	Smalltalk
Andrew Appel	Tiger
Bjarne Stroustrup	C++
Denis Ritchie	C
Jean Ichbiah	Ada 83
John Backus	FORTRAN, FP
John McCarthy	Lisp
Kristen Nygaard	Simula, Beta
Niklaus Wirth	Algol-W, Euler, Pascal, Modula-2, Oberon
Ole-Johan Dahl	Simula, Beta

2. Quel langage/compilateur a montré au monde les bénéfices des langages de haut-niveau compilés ?

   Correction: Fortran. Euh... FORTRAN, pardon.

3. À quel langage doit-on `if then else` ?

   Correction: Algol.

4. À quel langage doit-on l'orienté objet ?

   Correction: Simula.

5. Que sont les multiméthodes ?

   Correction: Sélection à l'exécution de la bonne fonction selon le type dynamique de n'importe quel argument, et non pas seulement l'argument 0 (`this`). C'est un support à l'exécution de ce que la surcharge permet à la compilation.

6. En C++, comment simule-t-on les multiméthodes ?

   Correction: Les VISITORS.

7. En Eiffel, lorsqu'une méthode est redéfinie, qu'advient-il de ses préconditions ?

   (a) elles ne sont pas modifiables
   (b) elles peuvent être affaiblies
   (c) elles peuvent être renforcées
   (d) elles sont librement modifiables

   Correction: Pas besoin de connaître Eiffel pour répondre ! Bien sûr, si une méthode prétend se substituer à une autre, il lui faut accepter toutes les acceptations de son prédécesseur. Par conséquent les préconditions peuvent être affaiblies.

8. En Eiffel, lorsqu'une méthode est redéfinie, qu'advient-il de ses postconditions ?

   (a) elles ne sont pas modifiables
   (b) elles peuvent être affaiblies
   (c) elles peuvent être renforcées
   (d) elles sont librement modifiables

   Correction: Pour les mêmes raisons, les postconditions peuvent être plus strictes.

3 Construction des Compilateurs

1. Quel est l'avantage de l'algorithme GLR sur LALR ?

   Correction: Toutes les grammaires hors-contexte sont acceptées, ce qui permet de ne pas avoir à contourner les conflits dus aux limitations de LALR(1).

2. Pourquoi est-ce aussi un désavantage dangereux ?

Correction: Mais on se retrouve à accepter des conflits sans nécessairement les comprendre. Autant LALR(1) est pénible, mais quand ça compile, il ne reste plus de doute (autre, bien sûr, que des bugs dans la grammaire), autant en GLR on peut découvrir tardivement des ambiguïtés pas prévues.

3. Que signifie AST en anglais et en français ?

   Correction: Abstract syntax tree, arbre de syntaxe abstraite. Il s'agit surtout de vérifier le féminin en français : c'est bien sûr la syntaxe qui est abstrait, pas l'arbre (bien concret lui).

4. Donner la syntaxe abstraite (soit sous la forme d'une grammaire, soit sous la forme d'une hiérarchie orientée objet typée) de la grammaire suivante.

```
<term> ::= <term> ( <term> )       -- Application
         | λ <var> . <term>        -- Abstraction
         | <var>                   -- Variable
<var>  ::= <identifier>
```

   On utilise -- pour introduire des commentaires qui bien sûr n'appartiennent pas à la grammaire.

   Correction:

```
<term> ::= Application ( <term>,  <term> )
         | Abstraction ( <var>,  <term>)
         | Variable ( <var> )
<var>  ::= NamedVariable ( <identifier> )
```

   ou encore

```
<term> ::= Application ( <term>,  <term> )
         | Abstraction ( <identifier>,  <term>)
         | Variable ( <identifier> )
```

   qui est moins bon : moins typé.
   En terme de hiérarchie, ce serait quelque chose comme

```
/Term/
   Application (Term, Term)
   Abstraction (Var,  Term)
   Variable (Var)
Var (Symbol)
```

5. Qu'est-ce qu'une table de symboles ?

   Correction: Un tableau associatif utilisé pour conserver des informations sur les identificateur. Il fournit donc des fonctionalités gérer pour les portées ; typiquement `scope_begin` et `scope_end`.

6. Pourquoi un langage comme le C inclut les "prédéclarations", et pas un autre comme Java ?

   Correction: Les prédéclarations permettent de faire une simple passe sur l'AST, alors que les langages comme Java et Tiger nécessitent plusieurs passes, ce qui implique d'avoir à conserver une grande part du programme (typiquement sous la forme d'un AST), quelque chose qui est bien trop coûteux au temps de C, et de parfaitement admissible à celui de Java.

7. Étant donné que le programme Tiger suivant est valide, expliquer quelle politique de gestion mémoire le compilateur doit utiliser pour les types.

```
let
  var box :=
    let
      type box = { val: string }
    in
      box { val = "42\n" }
    end
in
  print (box.val)
end
```

Correction: Très manifestement le compilateur aura besoin d'information à propos du type `box` après être sorti de la portée qui le définit. Par conséquent les descriptions des types ne peuvent pas être désaloué simplement au sortir de la portée qui les définit. Une solution simple est de désalouer à la fin de la passe de typage.

8. Soient deux fractions $\frac{a}{b}$, $\frac{c}{d}$, à quelle surprise s'expose-t-on si l'on programme $\frac{a}{b} \leq \frac{c}{d}$ comme `(a/b) <= (c/d)`.

   Correction: Les flottants ne sont pas nos amis, et il est possible que `(1/3) < (1/3)` s'évalue à vrai.

9. À quoi est dû ce comportement, et comment l'éviter ?

   Correction: Dans certaines architectures, les flottants ne sont pas stockés avec la même précision en mémoire et dans le processeur arithmétique. Autant éviter les flottants : `ad <= bc`.