Correction du Partiel de Compilation

Promo 2003 - Septembre 2001

Oui, ça me fait beaucoup rire de penser à ceux d'entre vous qui vont s'arracher les cheveux de ne pas avoir écouté la correction du deuxième partiel (disponible sur le web depuis mai).

1 Mise en ligne (Inlining)

La *mise en ligne* consiste à remplacer un appel de fonction par le corps de la fonction elle-même. Par exemple la mise en ligne de `function add (a: int, b: int) : int = a + b` dans `add (20, 22)` donne bien entendu (?) `20 + 22`. Sans chercher à aller jusqu'au bout, étudions les difficultés pour l'implémentation de la mise en ligne dans *notre* implémentation de `tc`.

1. En considérant par exemple la mise en ligne de `string_is_smaller (a: string, b: string) : int = a < b` et le contrôle de type, à quelle étape de la compilation suggérez-vous d'effectuer la mise en ligne ?
2. Que pensez-vous de la mise en ligne de `add (a: int, b: int) : int = a + b` dans une expression telle que `add (1, 2) * 3` ?
3. La mise en ligne naïve de `double (a: int) : int = a + a` peut conduire à des résultats désagréables voire désastreux. Comme quoi ?
4. Comment faut-il faire alors ?
5. En vous basant sur les questions précédentes, quelles différences faites-vous entre fonctions mises en ligne et macros en C ?
6. Comment faire dans le cas de :

   ```
   let function fact (n: int) : int = if n = 0 then 1 else n * fact (n - 1)
   in fact (5) end
   ```

   Qu'en déduisez-vous ?
7. Considérons :

   ```
   let var delta := 1
       function advance (x : int) : int = x + delta
       function start_from (delta : int) : int = advance (0) + delta
   in
       start_from (delta)
   end
   ```

   (a) À quel problème s'expose-t-on ?

   (b) Sauriez-vous réécrire cet exemple pour éviter ça ?

   (c) Quelle solution proposez-vous pour que l'implémentation de la mise en ligne ne soit pas galère à cause de ça ?

   (d) Comment implémenteriez-vous ceci dans `tc` ? Évidemment je ne veux pas de code ici, mais décrivez correctement votre suggestion, en utilisant le vocabulaire acquis cette année.

2 Références et Tiger

Le but de ce problème est d'étudier l'ajout du concept de *référence* dans le langage Tiger et son implémentation dans notre compilateur. Par *référence* j'entends quelque chose de proche des *pointeurs* en C.

Tiger est un langage décent, et vous m'avez déjà entendu dire que le C est un langage indécent, en particulier à cause de certaines propriétés des pointeurs du C. Alors pourquoi venir pourrir Tiger ? Peut-être parce qu'il y a moyen de le faire plus proprement qu'en C (mais en moins puissant, bien sûr).

Dans ce problème, beaucoup de choses ne sont pas dites, et c'est délibéré : c'est à vous de réfléchir et de tirer les bonnes conclusions des questions précédentes. En particulier, vous serez jugés sur votre respect de l'esprit Tiger dans les questions suivantes : c'est à vous de comprendre quand vous sortez du beau pour tomber dans le crade [1]. Et bien entendu, si on vous demande "est-ce que ça change quelque chose" et que vous répondez seulement "oui", je répondrai seulement "0".

1. Sûreté faiblement typée. En C, quelle est l'opération sur les pointeurs la plus susceptible de produire des pointeurs non nuls invalides ?

> **Correction:** L'affectation combinée avec le cast permet d'aller pointer n'importe où : `int *p = (int *) 42;`. Plus rigoureusement, c'est le cast et le cast lui seul qui est responsable du problème, puisque `(int *) 42` est bel et bien un pointeur invalide (en général).

2. Sûreté typée. Expliquez l'intérêt des opérations arithmétiques sur les pointeurs, puis le danger qu'elles représentent.

> **Correction:** Parcours de tableaux / déplacements non bornés.

3. Opérations. Par opposition aux pointeurs du C, les références en Tiger nous permettrons seulement de référer au contenu d'une (autre) variable. Quelles sont les deux primitives supplémentaires à introduire en Tiger ?

> **Correction:** Prendre une référence sur une variable (`my_ref := reference-to my_var`), et descendre une référence (`contents-of my_ref = my_var`).

4. Exemple. Écrire un programme Tiger qui, *successivement*

1. déclare une variable `nic = 42`
2. déclare une variable `theo`
3. fait pointer `theo` sur `nic`
4. utilise `theo` pour mettre `nic` à 51.

1. Par exemple, par pitié, respectez l'esprit de la syntaxe de Tiger, langage de mots plutôt que langage de symboles obscures.

Correction:

```
let var nic  : int := 42
    var theo : reference-to int := null
in
  theo := reference-to nic
  contents-of theo := 51
end
```

Cet exemple amène à réfléchir sur la nécessité de types “références”. Tiger était proprement typé, il est hors de question d'introduire le type `reference` à la façon de `void *` : il faut introduire le *constructeur de type* `reference-to`.
Par ailleurs, en Tiger il n'est jamais possible de déclarer une variable sans lui donner de valeur, il nous faut donc une valeur pour “référence pas initialisée”. Le premier qui dit “0” sort. On pourrait penser à `nil`, mais ce dernier est le record universel, pas le pointer universel. On ne le veut pas, donc autant introduire un token supplémentaire, disons `null`. Ce pointeur universel a les mêmes propriétés qu'en C : SEGV.

5. Grammaire. Proposez les règles de grammaires à rajouter à Tiger.

Correction:

```
  exp    ::= 'reference-to' lvalue;
  lvalue ::= 'contents-of' lvalue;
  ty     ::= 'reference-to' id;
```

6. Absyn. Présentez sous la forme de diagrammes UML ce que vous allez devoir rajouter pour pouvoir représenter ces opérations dans l'arbre de syntaxe abstraite.

7. Échappements. Est-ce que l'introduction des références nécessite des changements dans le calcul des variables en échappement ?

Correction: Pour une variable, être en échappement signifie ne pas pouvoir être stockée dans un registre. Clairement, si on prend une référence sur une variable, cette dernière doit avoir une adresse. Elle est donc en échappement.

8. Typage. Est-ce que l'introduction des références nécessite des changements dans les règles de typage de Tiger ?

9. Représentation intermédiaire Est-ce que l'introduction des références nécessite des changements dans la représentation intermédiaire ?

Correction: Que pouic, on sait déjà parler d'adresses etc.

10. Reste du compilateur. Est-ce que l'introduction des références change quelque chose en aval de la représentation intermédiaire ?

Correction: Ben vu qu'à la question précédente on a répondu que non, ici non plus.

11. References dans le vent. En dépit des précautions prises (en particulier en éliminant les opérations les plus dangereuses du C), nous venons d'offrir au programmeur le moyen de se tirer dans les pieds en beauté.

Écrire un programme Tiger qui crée une référence vers une cellule morte. Que suggérez-vous pour éviter cette situation ?

Correction:

```
let function dead_ref (a: int) : reference-to int = reference-to a
    var dead_ref = dead_ref (0)
in
  contents-of dead_ref = 42 /* Booom. */
end
```

Pas facile d'éviter ça ! Of course on veut pouvoir prendre des références sur des entités locales, sinon plus rien n'est possible ! Ce qu'on souhaite interdire c'est que des références à des variables locales remontent de leur créateur (mais par exemple on veut pouvoir les passer à des fonctions qu'on appelle).
Euh, là, à froid, je vois pas trop, faudrait réfléchir.