Votre nom :

Correction du Partiel CMP
Construction des Compilateurs

EPITA – Promo 2008 – **Tous documents autorisés**

Juin 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.

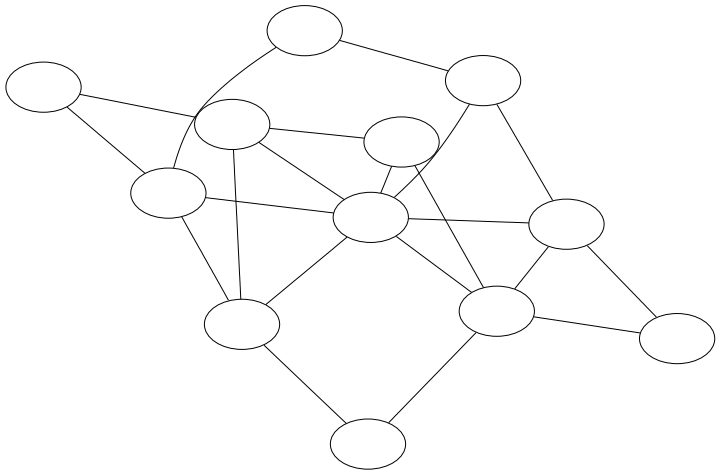
1. Le langage engendré par $S \to aX \quad S \to b \quad X \to Sc$ est rationnel. vrai/faux ?

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

2. Un sous-langage d'un langage rationnel (i.e., un sous-ensemble) est rationnel. vrai/faux ?

   Correction: N'importe quel langage L sur un alphabet Σ vérifie $L \subset \Sigma^\star$, donc bien sûr que c'est faux.

2 Construction des Compilateurs : Allocation des Registres

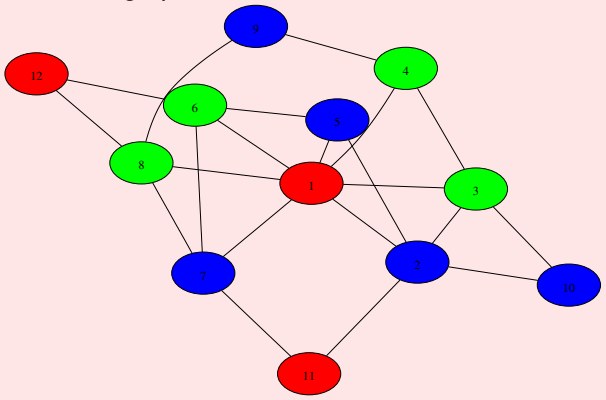


Colorer ce graphe d'exclusion mutuelle en 3 registres : rouge, vert et bleu.

Rendre le sujet en ayant annoté chaque noeud d'un V, R, ou B.

Écrire votre nom en haut.

Correction: Malheureusement j'ai perdu la figure donnée en partiel, qui est isomorphe à celle-ci, mais différente. La démarche est la même. Pour information un ordre de reconstruction du graphe est donné.



3 Construction des Compilateurs : Support de `goto`

On considère l'ajout d'une fonctionalité au compilateur Tiger : `goto`, à l'instar de l'instruction homonyme dans le C et bien d'autres langages. Les questions sont posées dans l'ordre des stades de compilation. Certains modules tardifs nécessitent une collaboration de modules plus en amont : il est plus sain de réfléchir globalement à toutes les questions, puis de répondre dans l'ordre. Si une étape ne nécessite aucune modification, simplement le dire, et ne pas tomber verbeusement dans le piège tendu par la question.

1. Pertinence Sachant que cette instruction est décriée par la plupart des auteurs, quelle peut être la pertinence de l'introduction de `goto` dans le langage supporté par le compilateur ?

> **Correction:** Il y a bien quelques situation où `goto` est bien pratique, comme par exemple la gestion des erreurs, les sorties de plusieurs boucles, etc. Mais de loin le plus gros avantage est de donner plus de pouvoir d'expression au langage pour pouvoir désucrer des instructions de plus haut niveau.

2. Deux-points, Syntaxe Un `goto` nécessite une étiquette désignant l'endroit où il doit brancher. La question de la syntaxe de cette étiquette se pose. Il est courant d'utiliser '`:`' pour poser une étiquette :

```
(start: print ("Hello world\n");
    goto start)
```

Autrement dit, en terme de grammaire :

```
<exp> ::= <id> ":" <exp>
```

À quels problèmes de priorité et d'associativité faut-il s'attendre ? Comment les corriger ?

> **Correction:** Si on fait ça, on a bien sûr des problèmes du genre '`a: 1 + 2`' doit-il être lu '`(a:1)+2`' ou '`a:(1+2)`'. C'est déjà le problème que l'on a avec l'affectation par exemple : '`a := 1+2`'. Le plus raisonnable, c'est de donner à '`:`' une très faible priorité.

3. Deux-points, AST Si l'on garde cette syntaxe, quel nœud d'AST faudrait-il rajouter ? En quoi est-il susceptible de nous poser des problèmes ?

> **Correction:** Si l'on écrit la règle de grammaire, on voit que l'on doit rendre compte de deux non-terminaux :
>
> ```
> exp: "identifier" ":" exp { $$ = LabelExp (@$, $1, $3); }
> ```
>
> Une telle écriture compliquera le parcours des arbres. Par ailleurs, peut-on vraiment dire que l'étiquette est une propriété de l''`exp`' elle-même ? Il semble plus sain qu'elle soit une instruction à part entière dans l'AST, mais alors, la syntaxe nous pose problème, puisqu'ici il faudrait retourner l'étiquette et l'`exp` indépendamment. Dans certains cas, ça nécessiterait même que le parser crée une '`SeqExp`' là où l'utilisateur n'en a pas.

4. 'label', Syntaxe Alternativement, on considère la syntaxe `label <id>`. Définir la grammaire en Bison, avec les éventuelles priorités/associativités.

> **Correction:**
>
> ```
> exp: "label" ID;
> ```
>
> "C'est LL(1)", vu que d'une part '`label`' et '`goto`' sont inutilisés ailleurs, et d'autre part que '`ID`' est un terminal, on n'a aucune concurrence avec une quelconque autre règle.

5. 'label', AST Définir les deux classes d'AST utilisées pour stocker '`label`' et '`goto`'.

Correction: Donnons les signatures, c'est bien suffisant :

```
/Ast/           (Location loc)

  /Exp/         ()
    LabelExp    (Symbol id)
    GotoExp     (Symbol id)
```

6. Espace de noms Si en Tiger (EPITA) il est possible d'utiliser `toto` pour désigner simultanément une variable, une fonction et un type, est-il possible qu'il désigne aussi une étiquette ?

Correction: Bien sûr. En fait on peut écrire la grammaire comme suit, sans conflit.

```
exp: "label" labelid |  "goto" labelid;
labelid: id;
```

Ce qui montre bien que la syntaxe sépare ces "espaces de noms".

7. Correction Proposer des règles définissant les usages corrects d'un identifiant d'étiquette.

Correction: Autoriser des '`goto`' d'une fonction à une autre est bien trop dangereux et violerait toutes les conventions d'appel. Par ailleurs, même au sein d'une fonction certains goto sont beaucoup trop "étranges"

```
(
  let var foo := 42
  in  label return; foo end;
  goto return
)
```

Restreindre un '`goto`' à la portée courante '`let .. in .. end`' est simple et raisonnable.
Les précautions d'usage s'imposent :

– quand on rentre dans une fonction, les '`goto`' courants sont oubliés (comme pour le break) ;
– idem pour une déclaration de variables ('`var foo := goto foo`').

7. Liaison Est-ce que le `BindVisitor` a une rôle à jouer ? Si oui, lequel ?

Correction: Il faudra bien sûr lier les '`GotoExp`' à leur '`LabelExp`'. Si les portées sont alignées sur le '`LetExp`', alors le '`BindVisitor`' sera relativement simple à mettre en œuvre.
Noter une grosse différence pour les labels : on a très envie de pouvoir sauter en avant, et par conséquent il se peut que les déclarations suivent les utilisations. C'est nouveau pour le '`BindVisitor`'.

8. Typage Quelles sont les règles de typage pour '`label`' et '`goto`' ?

Correction: Les deux sont typés '`Void`'.

9. Désucrage '`for`' On rappelle qu'une boucle '`for`' :

```
for i := l to u do b
```

se désucre ainsi :

```
let
  var _l := l'  /* l désucré. */
  var _u := u'  /* u désucré. */
  var i := _l
in
  if i <= _u then
```

```
  while 1 do
   (
    b';    /* b désucré. */
    if i = _u then
     break;
    i := i + 1
   )
end
```

Proposer une version plus légère. Quelle difficulté pose l'introduction de '`label`' ?

Correction:

```
let
 var _l := l'  /* l désucré. */
 var _u := u'  /* u désucré. */
 var i := _l
in
 if i > _u then goto endloop;
 label forever;
 b';    /* b désucré. */
 if i = _u then
  goto endloop;
 i := i + 1
 goto forever;
 label endloop
end
```

Il ne faut pas que les étiquettes introduites soient utilisées pas l'utilisateur.

10. Désucrage Deux autres structures de contrôles (liées) deviennent du sucre syntaxique de '`label`'/'`goto`'. Les citer, et les désucrer.

Correction: Après le '`for`', c'est bien sûr '`while`' et '`break`'.

```
while c do b
```

devient

```
label lbegin;
if c' then (b'; goto lbegin);
label lend;
```

ou bien (il est possible que la suite du compilo préférerait) :

```
label lbegin;
if !c' then goto lend;
b';
goto lbegin;
label lend;
```

et '`break`' devient '`goto end`'.

11. Désucrage En définitive, comment désucrer une boucle '`for`' ?

Correction: Au choix : directement, on en en faisant une boucle '`while`' comme ci-dessus, puis en la désucrant.

12. Code intermédiaire Quelle modification apporter au langage intermédiaire pour supporter '`label`'/'`goto`' ?

Correction: Rien, on a déjà '`label`' et '`jump`'

13. Traduction Comment traduire '`label`'/'`goto`' en langage intermédiaire ?

Correction: C'est trivial, il y a une correspondance directe avec Tree. Cependant, serait intéressant d'essayer de découvrir aussi les '`cjump`' cachés dans les '`if... goto`' comme dans les exemples de désucrage de '`while`'.

14. Canonisation Comment modifier la traduction HIR vers LIR ?

Correction: Pas de changement.

15. Sélection des Instructions Comment modifier la traduction LIR vers assembleur MIPS ?

Correction: Pas de changement.

16. Graphe d'Interférence Comment modifier la génération des graphes de flot de contrôles, de vivacité, d'exclusion mutuelle ?

Correction: Pas de changement.

17. Allocation des Registres Comment modifier l'allocation des registres ?

Correction: Pas de changement.

18. Subsidiaire Quels bénéfices apportés par '`label`'/'`goto`' n'ont pas été abordés dans ce sujet ?

Correction: Une idée ?