CMP2 – Construction des compilateurs

EPITA – Promo 2010
Tous documents (notes de cours, polycopiés, livres) autorisés.
Tous dispositifs de calcul automatisé interdits.*

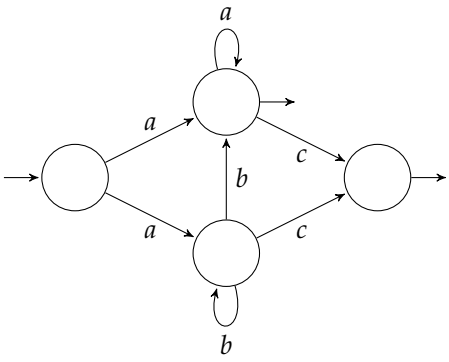
Juin 2008 (1h30)

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**. Répondez sur les feuilles de QCM qui vous sont remises (n'oubliez pas d'y inscrire votre nom ou login).

Considérez l'automate ci-dessous, et répondez aux questions qui suivent.



1. Cet automate est déterministe. A. Vrai/B. Faux ?
2. Un ordinateur peut reconnaître un mot du langage décrit par cet automate directement (c'est-à-dire, en simulant une exécution de cet automate sans modification ou ajustement préalable de celui-ci). A. Vrai/B. Faux ?
3. Cet automate reconnaît le mot "*a*". A. Vrai/B. Faux ?
4. Le langage reconnu par cet automate est dans la classe des langages hors-contexte. A. Vrai/B. Faux ?

*Calculatrice, téléphone portable, ordinateur, *mentat*, etc.

2 Étendons Bison

1. Voici une grammaire simplifiée de la partie “grammaire” que l'on peut trouver dans un fichier d'entrée de Yacc (située entre les deux séparateurs `%%` du fichier), sans les actions (pour ne compliquer l'exercice outre mesure) :

```
grammar −> rules
grammar −> grammar rules

rules −> ID ":" rhses

rhses −> rhs
rhses −> rhses "|" rhs
rhses −> rhses ";"

rhs −> rhs ID       (Typo dans le sujet initial, qui comportait
                    une règle erronée « rhs −> rhs SYMBOL ».)
rhs −> eps
```

   Cette grammaire est-elle LR(1) ? Est-elle LR(k) ? Si oui, pour quel k ? Justifiez votre réponse.

2. Vous avez vu en cours de Théorie des Langages qu'en pratique, Yacc ne peut pas utiliser cette grammaire *directement*[1]. Pour quelle raison ?

3. Quelle modification peut-on introduire pour faire accepter cette grammaire à Yacc ?

4. On décide d'étendre cette grammaire pour pouvoir *nommer* les symboles dans les règles, plutôt que d'y faire référence avec des numéros. Concrètement, cela signifie qu'au lieu d'écrire :

```
exp: "identifier" "[" exp "]" "of" exp
{
 $$ = new Array (@$, new Type (@1, $1), $3, $6)
}
```

   dans un fichier d'entrée de Bison, on veut pouvoir écrire :

```
exp$res : "identifier"$type "[" exp$size "]" "of" exp$init
{
 $res = new Array(@res, new Type(@type, $type), $size, $init)
}
```

   À partir de la grammaire donnée au début de l'exercice, et corrigée à la question 3, proposez une nouvelle grammaire **LR(1)** prenant en compte cette syntaxe étendue (sans les actions).

3 Tigris fluctuat nec mergitur

Dans cet exercice, on se propose de rajouter un nouveau type de données à notre langage de programmation préféré (Tiger) : les nombres à virgule flottante ou flottants (*floats*).

~~Nous allons~~ Vous allez (c'est votre épreuve, après tout) donc passer en revue les différentes parties de notre compilateur afin de déterminer quels sont les aménagements nécessaires.

Rappel : le langage Tiger utilisé dans cet exercice est celui qui est décrit dans le Manuel de Référence du Compilateur Tiger, qui a été utilisé comme référence tout au long du projet du même nom.

[1] Les entrées de Yacc sont donc définies avec une grammaire que Yacc ne sait pas parser, ce qui est *a priori* paradoxal.

[1] Il s'agit de l'option `%glr-parser` (cf. la correction également).

1. **Spécification lexicales.** Que faut-il ajouter aux spécifications *lexicales* du langage pour supporter les flottants ?

2. **Scanner.** Comment étendre le scanner, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`scantiger.ll`' pour supporter ces nouvelles spécifications lexicales ?

3. **Spécification syntaxiques.** Que faut-il ajouter aux spécifications *syntaxiques* du langage pour supporter les flottants ?

4. **Parser.** Comment étendre le parser, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`parsetiger.yy`' pour supporter ces nouvelles spécifications syntaxiques ?

5. **Liaison des noms.** Que faut-il changer dans le `Binder` vis-à-vis des flottants ?

6. **Sémantique des flottants.** Le nouveau type de données flottant nécessite des règles de sémantique précises (manipulation de variables ou littéraux flottants). Au moins deux éléments essentiels sont concernés, qui sont liés à deux types de *polymorphismes* que vous connaissez.

   (a) Citez ces deux polymorphismes.

   (b) Quelles règles décidez-vous d'appliquer concernant les éléments suivants du langage dans lesquels les flottants sont impliqués :

      i. affectation & initialisation des variables locales,
      ii. initialisation des arguments effectifs des fonctions,
      iii. expressions binaires arithmétiques,
      iv. expressions binaires logiques.

   Vous êtes libres des choix de sémantique de cette extension, mais la cohérence de ceux-ci sera bien sûr prise en compte.

   Vous n'êtes pas obligés de calquer la sémantique d'un autre langage, mais il peut-être utile de se rappeler ce que vous connaissez dans d'autres langages au sujet des flottants.

7. **Vérification des types.** Qu'allez-vous changer dans le module `type` du compilateur ? Réfléchissez bien, et n'oubliez rien !

8. **Représentation intermédiaire** Faut-il ajouter/modifier quelque chose dans le langage Tree ?

9. **Traduction (vers HIR).** Y'a-t-il des changements à apporter au visiteur chargé de la traduction vers la représentation intermédiaire de haut niveau ? Justifiez votre réponse.

10. **Canonisation (HIR vers LIR).** La canonisation est-elle affecté par cette extension ? Justifiez votre réponse.

11. **Langage *Assem*.** Nous allons considérer que nous ciblons l'architecture MIPS, comme dans le projet (par défaut). Que doit-on ajouter dans le langage d'assemblage à registres infinis *Assem* ?

12. **Sélection d'instructions (génération de code).** Quels sont les changements à apporter au générateur de code en langage d'assemblage ?

13. **Analyse de vivacité.** L'analyse de vivacité est-elle modifiée par cette extension ? Justifiez votre réponse.

[1] Attention à ne pas confondre IEEE 1394 (http://en.wikipedia.org/wiki/IEEE_1394), une norme de bus série également appelée "Firewire" ou "i.LINK" ; et IEEE 754 (http://en.wikipedia.org/wiki/IEEE_754-1985), une norme de représentation des nombres à virgule flottante !

14. **Allocation de registres.** L'allocation de registres est-elle modifiée par cette extension ? Justifiez votre réponse.

15. **Bibliothèque standard.** Quelles routines pourrait-on souhaiter ajouter à la bibliothèque standard du langage Tiger dans le cadre de cette extension ?

16. **Questions Bonux™.** Après les nombres flottants (qui sont une représentation "approchée" de $\mathbb{R}$), on souhaite ajouter un type de données pour représenter les nombre complexes ($\mathbb{C}$). Dans cette question, on admet que l'on dispose d'un compilateur Tiger avec le support des flottants.

    Cette nouvelle extension peut se traiter de deux façons différentes au moins :

    - de façon *intrusive*, en modifiant le langage lui-même (ce qui implique une modification du compilateur) ;
    - de façon *non intrusive*, sans affecter le langage (en étendant la bibliothèque standard par exemple).

    Les questions suivantes donnent lieu à des points de bonus :

    (a) Quels sont les changements à apporter au compilateur dans la première approche ? (Vous pouvez reprendre le plan des questions 1 à 15 pour répondre.)

    (b) Comment implémenteriez-vous le second cas ? (Les autres langages peuvent donner des idées.)

4 À propos de ce cours

Pour terminer cette épreuve, nous vous invitons à répondre à un petit questionnaire, comme lors de l'épreuve du premier semestre.

Les renseignements ci-dessous ne serons bien entendu pas utilisés pour noter votre copie. Ils ne sont pas anonymes, car nous souhaitons pouvoir confronter réponses et notes. En échange, quelques points seront attribués pour avoir répondu. Merci d'avance.

Sauf indication contraire, vous pouvez cocher plusieurs réponses par question. Répondez sur les feuilles de QCM qui vous sont remises. N'y passez pas plus de dix minutes.

Le cours

5. Quelle a été votre implication dans les cours (THL, CCMP, TYLA) ?

   A Rien.

   B Bachotage récent.

   C Relu les notes entre chaque cours.

   D Fait les annales.

   E Lu d'autres sources.

6. Ce cours

   A Est incompréhensible et j'ai rapidement abandonné.

   B Est difficile à suivre mais j'essaie.

[1] Pour info, C++ aussi (http://www.ddj.com/web-development/184401491), mais son approche serait à ranger dans le second cas, non intrusif (NDC).

[1] Voir http://fr.wikipedia.org/wiki/Klingon_(langue) et http://en.wikipedia.org/wiki/Klingon_language (NDC).

C Est facile à suivre une fois qu'on a compris le truc.

D Est trop élémentaire.

7. Ce cours

   A Ne m'a donné aucune satisfaction.

   B N'a aucun intérêt dans ma formation.

   C Est une agréable curiosité.

   D Est nécessaire mais pas intéressant.

   E Je le recommande.

8. La charge générale du cours (relecture de notes, compréhension, recherches supplémentaires, etc.) est

   A Telle que je n'ai pas pu suivre du tout.

   B Lourde (plusieurs heures par semaine).

   C Supportable (environ une heure de travail par semaine).

   D Légère (quelques minutes par semaine).

Les formateurs

9. L'enseignant

   A N'est pas pédagogue.

   B Parle à des étudiants qui sont au dessus de mon niveau.

   C Me parle.

   D Se répète vraiment trop.

   E Se contente de trop simple et devrait pousser le niveau vers le haut.

10. Les assistants

    A Ne sont pas pédagogues.

    B Parlent à des étudiants qui sont au dessus de mon niveau.

    C M'ont aidé à avancer dans le projet.

    D Ont résolu certains de mes gros problèmes, mais ne m'ont pas expliqué comment ils avaient fait.

    E Pourraient viser plus haut et enseigner des notions supplémentaires.

Le projet Tiger

11. Vous avez contribué au développement du compilateur de votre groupe (une seule réponse attendue) :

    A Presque jamais.

    B Moins que les autres.

    C Équitablement avec vos pairs.

    D Plus que les autres.

    E Pratiquement seul.

12. La charge générale du projet Tiger est

A Telle que je n'ai pas pu suivre du tout.

B Lourde (plusieurs jours de travail par semaine).

C Supportable (plusieurs heures de travail par semaine).

D Légère (une ou deux heures par semaine).

E J'ai été dispensé du projet.

13. Y a-t-il de la triche dans le projet Tiger ? (Une seule réponse attendue.)

A Pas à votre connaissance.

B Vous connaissez un ou deux groupes concernés.

C Quelques groupes.

D Dans la plupart des groupes.

E Dans tous les groupes.

Questions 14-20 (1 pt) Le projet Tiger vous a-t-il bien formé aux sujets suivants ? Répondre selon la grille qui suit. (Une seule réponse attendue par question.)

A Pas du tout

B Trop peu

C Correctement

D Bien

E Très bien

14. Formation au C++
15. Formation à la modélisation orientée objet et aux *design patterns*.
16. Formation à l'anglais technique.
17. Formation à la compréhension du fonctionnement des ordinateurs.
18. Formation à la compréhension du fonctionnement des langages de programmation.
19. Formation au travail collaboratif.
20. Formation aux outils de développement (contrôle de version, systèmes de construction, débogueurs, générateurs de code, etc.

Questions 22-37 (1 pt) Comment furent les étapes du projet (ne pas répondre à celles que vous n'avez pas faites). Répondre selon la grille suivante. (Une seule réponse attendue par question.)

A Trop facile.

B Facile.

C Nickel.

d Difficile.

E Trop difficile.

21. Mini-projet en Tiger (LZW)

22. `TC-0`, Scanner & Parser.
23. `TC-1`, Scanner & Parser en C++, Autotools.
24. `TC-2`, Construction de l'AST.
25. `TC-3`, Liaison des noms.
26. `TC-4`, Typage.
27. Desucrage des constructions objets (transformation Tiger → Panther)
28. `TC-5`, Traduction vers représentation intermédiaire.

29. `TC-6`, Simplification de la représentation intermédiaire.
30. `TC-7`, Sélection des instructions.
31. `TC-8`, Analyse du flot de contrôle.
32. `TC-9`, Allocation de registres.

33. Option `TC-E`, Calcul des échappements.
34. Option `TC-A`, Surcharge des fonctions.
35. Option `TC-D`, Suppression du sucre syntaxique (boucles `for`, comparaisons de chaînes de caractères).
36. Option `TC-B`, Vérification dynamique des bornes de tableaux.
37. Option `TC-I`, Mise en ligne du corps des fonctions.