Partiel de Compilation

Promo 2002 - Septembre 2000

Dans cette épreuve, vous avez le droit d'être intelligent(e)s et de prendre des initiatives.

1 Le coin de Backus et Naur

Donnez des grammaires EBNF *et* BNF pour les langages suivants.

1. Les logins standard de l'EPITA.
2. Les nombres hexadécimaux à virgule fixe (sans exposant). Par exemple `CafeDeca.42`.
3. Nombres hexadécimaux à virgule flottante (avec exposant). Donner des exemples en plus des grammaires.
4. L'arithmétique des nombres entiers décimaux avec les 4 opérations usuelles, plus la puissance `**`, et avec les parenthèses.

On se donnera le droit d'utiliser - comme dans `[ 'a' - 'z' ]`. On sera rigoureux dans la différenciation entre terminaux et non terminaux.

2 Analyse de listes

Le client dit qu'une *liste* c'est un truc avec des *machins* qui sont séparés par des deux-points. Il ne veut pas dire ce que sont les *machins*, il s'en charge, mais en tout cas, il peut ne pas y en avoir, et l'ordre compte.

1. Proposez trois grammaires `bison` qui expriment ses besoins sans engendrer de conflits.
2. Choisissez-en une, et montrez à votre client comment en quelques lignes de Bison vous lui fabriquez une liste STL de *machins* (de la classe `Machin`).
3. Oui, mais le client il n'a pas confiance dans les trucs gratuits, et c'est pas fiable et niah niah niah. Proposez une implémentation C++ par analyse descendante récursive (LL). Est-ce la même grammaire que vous implémentez ?
4. Dérécursivez votre implémentation, il aime pas ça non plus.
5. Le client vient seulement de se souvenir qu'en fait ses fichiers contiennent des listes séparées par des retour à la ligne. Complétez la grammaire, et votre implémentation LL de façon à construire la liste de listes.

Vous prendrez soin de masquer les détails de `eat` et autres `token_check`, mais vous ne cacherez pas au client les noms de non terminaux.

3 Grammaires

Pour chacune des grammaires suivantes, exprimez son langage en français (c'est l'ensemble des mots qui...), exprimez son langage en mathématiques (par exemple $L(G) = \{a^n | n$ est multiple de 42$\}$), donnez la (les) classe(s) auxquelles elles appartiennent, et suggérez des outils/technologies pour vérifier si un mot appartient à leur langage.

1. `S -> Sa  S -> a`
2. `S -> AB  A -> Aa  B -> bB`
   `A -> a  B ->`
3. `S -> ABA  A -> Aa  B -> bBb`
   `A -> a  B ->`
4. `S -> ASA  S -> B  BA -> BB  A -> a  B -> b`

4 Tiger : syntaxe de `let`

Pourquoi diable y a-t-il un `end` pour terminer un `let` ? Pourquoi pas simplement

```
let var a := 6
    var b := 7
in
   a * b
```

Il y a au moins deux réponses intelligentes possibles.

5 Tiger : syntaxe du `where`

Le client voudrait pouvoir écrire :

```
fact (10)
  where function fact (i : int) : int =
          if i > 0 then i * fact (i - 1) else 1
```

avec un sens qu'il trouve évident.

1. Proposez la règle la plus simple qui reconnaisse le cas ci-dessus.
2. Si on l'implémente, `bison` détecte de nombreux conflits. En l'opposant à d'autres constructions simples d'expressions (binaires, structure de contrôle etc.), montrez que la "spécification" donnée par le client est incomplète.
3. Complétez la spécification. À l'aide d'exemples, justifiez votre choix, et critiquez ce que vous rejetez.
4. Si on lit dans les pensées du client, il est manifeste que l'exemple avec juste `fact` ne pousse pas la structure `where` jusqu'au bout. Donnez un exemple simple d'utilisation de `where` qui soit un peu plus poussé.
5. Quelle règle ajoutez-vous pour implémenter ceci dans la grammaire de Tiger ?
6. Donnez-lui une règle de récupération sur erreur.
7. Une fois la règle prête et acceptée par le client, comment allez-vous vous y prendre pour implémenter dans votre compilateur Tiger le support complet de cette extension ? Combien de temps ça va prendre ?
8. Tout ce temps-là plus tard, un soir, au bar, vous expliquez à vos potes(ses) informaticien(ne)s ce que vous avez été obligé(e) de faire pour le client. Videz votre sac sur cette extension, i.e., dites ce que vous en pensez. Ils comprennent ce que vous dites, n'hésitez pas sur les détails.