Polytech Nice Sophia
SI4 / 2021-2022
Paul Franchi / Erick Gallesio

Feuille 4

Yacc / Bison

Utilisation de Yacc sur des grammaires simples

1 Une grammaire simple

Ce premier exercice utilise une grammaire très simple pour démarrer avec **Bison**. Pour cela on utilise la grammaire **G** suivante:

```
S ⟶ Sa      (r1)
S ⟶ x       (r2)
S ⟶ y       (r3)
```

Construire un fichier source **Bison** permettant d'analyser **G**.

1. Dans un premier temps, on se contentera ici d'un analyseur lexical simple "inline" comme celui qui a été donné en cours (i.e. on n'utilisera pas *lex/flex*).

2. Construire les fichiers `'.output'` et `'.dot'`. Ces fichiers permettent de visualiser l'automate produit par **Bison**. Regarder les options `--report`, `-g` pour produire un fichier `'.output'` et un fichier `'.dot'`.

   - Le fichier `'.output'` est un fichier texte.

   - Le fichier `'.dot'` est un fichier qui peut être visualisé avec le programme `xdot` (ou un fichier PDF peut être produit avec la commande `dot`. Par exemple: `dot -Tpdf fich.dot > fich.pdf`).

     Sous GNU/Linux, ces fichiers peuvent aussi visualisés avec l'outil interactif `xdot`.

3. Étendre la grammaire pour autoriser plusieurs analyses (sur le mode une entrée par ligne, terminée par un `'\n'`) pour la même exécution de l'analyseur.

4. Afficher un prompt avant chaque commande. Ce prompt contiendra au moins le numéro de la ligne courante sur le fichier standard d'entrée.

5. En cas d'erreur syntaxique, afficher un message d'erreur avec le numéro de la ligne dans le fichier de source et le "token" en erreur (il est dans `yychar`). Essayez de définir la valeur de la macro `YYERROR_VERBOSE` à 1, pour voir son influence sur le message d'erreur.

6. Implémenter une reprise en cas d'erreur (faire retaper la ligne complète).

7. Utiliser maintenant *flex* pour définir l'analyseur lexical. Votre analyseur devra renvoyer quatre unités lexicales:

   - `TOK_A` quand on rencontre un 'a' (minuscule ou majuscule)
   - `TOK_X` quand on rencontre un 'x' (minuscule ou majuscule)
   - `TOK_Y` quand on rencontre un 'y' (minuscule ou majuscule)
   - la constante `\n` quand on rencontre un fin de ligne.

Par ailleurs, vous utiliserez ici la variable `yylineno` gérée par *flex*. Pour cela, il faudra activer `%option yylineno` dans le source de l'analyseur.

2 Expressions

Utiliser le couple *flex/bison* pour reconnaître des expressions simples sur les entiers (les quatre opérateurs, le moins unaire et les parenthèses, pas de variable).

Un exemple d'utilisation de cet analyseur est donné ci-dessous:

```
[1] 3 - 2 * 5
==> -7
[2] -(2+2)
==> -4
[3] 1+*1
Error: Line 3. Token: '*' syntax error, unexpected '*', expecting NUMBER or '-' or '('
```

L'analyseur à construire doit:

- afficher un prompt
- accepter les entrées multiples (une par ligne, terminée par un ' `\n` ').
- afficher des messages d'erreurs précis
- avoir une reprise d'erreur (faire retaper la ligne complète).
- évaluer l'expression (calcul par les attributs *yacc*).

On écrira plusieurs versions de cet analyseur:

1. la première version sera basée sur la grammaire **ETF** du cours;
2. écrire ensuite une version de type **E+E**. Avant de mettre les règles de priorité,
   - vérifier que la grammaire est ambiguë
   - Essayer d'évaluer `2*5+7` ainsi que `2+5*7` . Qu'obtenez vous et pourquoi?
3. ajouter ensuite les règles de priorité et vérifier que les évaluations précédentes sont correctes.
4. modifier l'analyseur syntaxique pour accepter des nombres réels (ne pas oublier de modifier aussi votre analyseur lexical). Pour cela on utilisera des attributs de type `double` .
5. Regarder l'automate produit pour votre dernière grammaire. Impressionnant, non?

3 Un traducteur ETF vers XML

Reprenez un analyseur pour une grammaire **ETF** simple (les quatre opérateurs sur des entiers et les parenthèses). Dans cet exercice, on va produire du code XML au lieu de d'évaluer la valeur de l'expression.

Par exemple si on entre l'expression `2+3*5*(3-7/10)` , l'analyseur produira la sortie XML suivante:

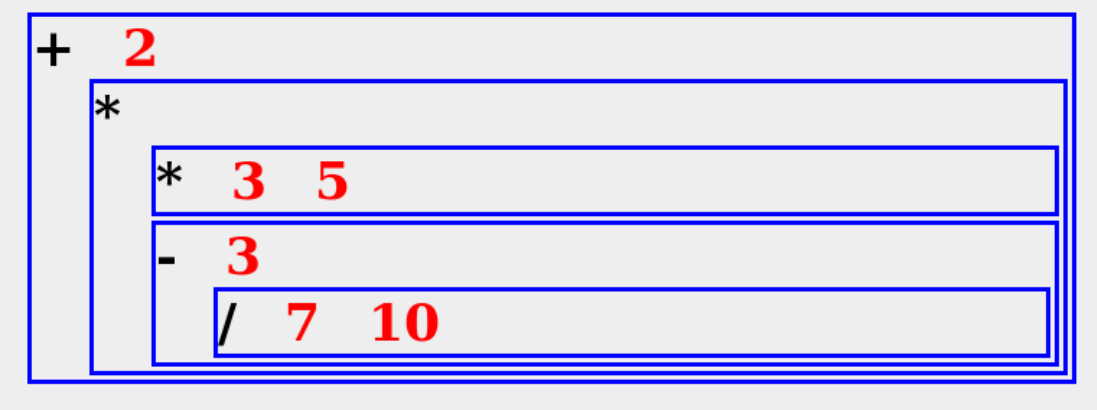
```xml
<?xml version='1.0' encoding='utf-8' ?>
<!-- Generated by bison on Oct 16 2015 at 21:00:17 -->
<?xml-stylesheet href='exo3.css' type='text/css' ?>
<expr>
  <add op='+'>
    <number>2</number>
    <mul op='*'>
      <mul op='*'>
        <number>3</number>
        <number>5</number>
      </mul>
      <add op='-'>
         <number>3</number>
         <mul op='/'>
            <number>7</number>
            <number>10</number>
         </mul>
      </add>
    </mul>
  </add>
</expr>
```

Télechargement:

Le fichier contenant la CSS: exo3.css.

Visualisation:

Le fichier produit sur l'expression précédente peut être lu par un navigateur; on obtient la sortie suivante:



Lorsque vous aurez fini, vous aurez donc construit un compilateur **expressions → XML**