

### 1. Grammaire étendue des expressions

Soit les expressions arithmétiques faites d'opérateurs habituels (+, ×, −, /), les expressions de comparaison (<, ≤, >, ≥, =, ≠) et les expressions logiques (∨, ∧, ¬).

Les termes atomiques sont les nombres entiers.

Il sera par exemple possible d'écrire

$$(7 \times 6 = 42) \wedge (7 \times 7 = 49)$$

- (a) Écrire une grammaire qui produit les expressions arithmétiques et logiques utilisant seulement les opérateurs binaires en n'utilisant que trois symboles `exprArith`, `exprComp` et `exprLog`.  
Compiler l'analyseur syntaxique et observer les informations données par `Bison` en utilisant les paramètres `--report itemsets` et `-Wcex` pour une version récente de `bison` (pas toujours installée dans les distributions Linux actuelles). Les corrections apportées automatiquement après affichage des avertissements du compilateur de compilateur sont-elles satisfaisantes ?
- (b) Réécrire la même grammaire en utilisant les mots clé `%left`, `%right`, `%nonassoc` ou `%precedence` en tenant compte des propriétés suivantes de l'algèbre :
  - Les multiplications sont prioritaires sur les additions.  $(a + b \times c = a + (b \times c))$
  - Les conjonctions sont prioritaires sur les disjonctions.  $(a \vee b \wedge c = a \vee (b \wedge c))$
  - Tous les opérateurs binaires sont associatifs à gauche.  $(a + b + c = (a + b) + c)$
  - Les opérandes finales des opérations logiques sont des expressions de comparaison ou des constantes logiques.
  - Les opérandes finales des opérations de comparaison sont des expressions arithmétiques ou des nombres.
  - Les opérandes finales des opérations arithmétiques sont des nombres
- (c) Ajouter les opérateurs unaires en remarquant que ces opérateurs sont prioritaires sur les opérateurs binaires et sont associatifs à droite.

#### Remarque :

Pour distinguer l'opérateur « − » binaire de l'opérateur « − » unaire, le tokenizer ne suffit pas (ils s'écrivent pareil). Il faut alors déclarer un token `UMINUS` et l'ajouter à la règle pour exprimer la priorité de ce opérateur comme ceci :

```
exprArith : MINUS exprArith %prec UMINUS;
```

### 2. Sémantique des expressions

Les trois symboles `exprArith`, `exprComp` et `exprLog` seront maintenant typés respectivement comme des objets de type `Integer`, `Boolean` et `Boolean`.

Reprendre cette grammaire afin de calculer la sémantique associée aux expressions arithmétiques et logiques. Afficher enfin le résultat.

### 3. Analyse LR

Soit la grammaire augmentée suivante :

- 1)  $E' \rightarrow E \#$
- 2)  $E \rightarrow E + B$
- 3)  $E \rightarrow E * B$
- 4)  $E \rightarrow B$
- 5)  $B \rightarrow nb$

La construction des ensembles d'items donne ceci :

- $I_0$     $E' \rightarrow \bullet E \#$   
           $E \rightarrow \bullet E + B$   
           $E \rightarrow \bullet E * B$   
           $E \rightarrow \bullet B$   
           $B \rightarrow \bullet nb$
- $I_1$     $E' \rightarrow E \bullet \#$   
           $E \rightarrow E \bullet + B$   
           $E \rightarrow E \bullet * B$
- $I_2$     $E \rightarrow B \bullet$
- $I_3$     $B \rightarrow nb \bullet$
- $I_4$     $E \rightarrow E + \bullet B$   
           $B \rightarrow \bullet nb$
- $I_5$     $E \rightarrow E * \bullet B$   
           $B \rightarrow \bullet nb$
- $I_6$     $E \rightarrow E + B \bullet$
- $I_7$     $E \rightarrow E * B \bullet$

- (a) Construire la table des suivants de chaque symbole
- (b) Construire la table SLR de la grammaire
- (c) Analyser le mot  $nb + nb * nb \#$
- (d) Construire l'arbre d'analyse étant donné l'analyse obtenue
- (e) Que doit-on conclure de la grammaire en examinant cet arbre ?