INF404 - Travaux pratiques - Séance 4 bis

Expressions booléennes

Important :

Avant de commencer ce TP il est **nécessaire** d'avoir fini le TP4 (parties 1 et 2) ...

Objectifs

L'objectif de cette séance est de réaliser une "calculette booléenne" sur le modèle de la calculatrice réalisée jusqu'à présent. Ces expressions booléennes pourront êtres intégrés dans le langage de programmation que vous allez définir dans la suite du projet.

Etape 1 : expressions booléennes

On définit la syntaxe d'*expressions booléennes* (*Ebg*), On propose ici de considérer des "vrais" expressions booléennes (comme en Ada) [1], construites à partir des constantes `VRAI` et `FAUX` et des opérateurs logiques `ET`, `OU` et `NON`. Pour faciliter l'analyse syntaxique, on introduira également les deux lexèmes `CROCHO` et `CROCHF` correspondants respectivement aux "crochet ouvert" (`[`) et "crochet fermant" (`]`).

Une première étape consiste donc à choisir une représentation lexicale pour ces constantes et opérateurs et mettre à jour le paquetage ***analyse_lexicale***.

On donne ci-dessous une grammaire G_1 pour les *Ebg*, similaire à celle des *Eag* (en remplaçant `PLUS` par `OU` et `MULT` par `ET`) :

Ebg	$\rightarrow$	Disj Suite_Disj
Suite_Disj	$\rightarrow$	`OU` Disj Suite_Disj
Suite_Disj	$\rightarrow$	ε
Disj	$\rightarrow$	Conj Suite_Conj
Suite_Conj	$\rightarrow$	`ET` Conj Suite_Conj
Suite_Conj	$\rightarrow$	ε
Conj	$\rightarrow$	`CROCHO` Ebg `CROCHF`
Conj	$\rightarrow$	`NON` Ebg
Conj	$\rightarrow$	`VRAI`
Conj	$\rightarrow$	`FAUX`

Ajoutez à votre analyse syntaxique la reconnaissance des *Ebg* (en programmant les procédures récursives `Rec_Ebg`, `Rec_Suite_Disj`, etc.). Complétez ensuite cette analyse avec la construction de l'arbre abstrait correspondant (il faudra enrichir les modules ***Ast***).

1. Une alternative serait d'utiliser les valeur 0 pour "faux" et différent de 0 pour "vrai" comme en C.

Ecrivez un (petit) programme de test pour vérifier que tout fonctionne : analyse d'une *Ebg*, construction de l'arbre abstrait et évaluation (calul de la valeur booléenne résultat).

Etape 2 : expressions booléennes et arithmétiques

Dans ce langage de programmation il sera nécessaire de combiner expressions booléennes et arithmétiques, par exemple pour exprimer des conditions comme `(x > 3) et non (y - 9 <= z)`. Il faut donc tout d'abord étendre l'analyse lexicale pour ajouter les lexèmes correspondants aux opérateurs de comparaison `INF` (`<`), `INF_EGAL` (`<=`), `EGAL` (`==`), etc. Il faut ensuite compléter la grammaire des *Ebg* en ajoutant la règles suivantes :

Conj	→	`PARENTH_OUVRANTE` Eag Relop Eag `PARENTH_FERMANTE`
Relop	→	`INF`
Relop	→	`INF_EGAL`
Relop	→	`EGAL`
Relop	→	...

Intégrez cette extension en modifiant les modules ***analyse_lexicale***, ***analyse syntaxique*** et ***Ast***.