UFR d'Informatique
L3 Informatique, Math-Info et Japonais-Info

Grammaires et analyse syntaxique
Année 2024-2025

TP n°9

Suite du TP8

Grammaire pour des expressions booléennes (suite)

Exercice 1 *On souhaite améliorer la fonction* `eval` *de la dernière question du TP précédent, pour tenir compte de la valeur des variables. Pour cela, la fonction* `eval` *doit prendre comme argument supplémentaire un environnement, de type* `(string * bool) list`*, qui associe à chaque nom de variable sa valeur booléenne. Modifier la fonction* `eval` *en conséquence. Elle devrait avoir le type suivant :*

```
eval : (string * bool) list -> Ast.expression -> bool
```

On pourra utiliser la fonction `List.assoc` *pour manipuler l'environnement. Si une variable non declarée est utilisée dans une expression (comme* `x \/ true`*), cela declenche une erreur de* `List.assoc`*. Tester votre fonction sur le programme suivant (quel est le résultat attendu ?) :*

```
let x = false in (let x = true and y = false in x \/ y) /\ x
```

Exercice 2 *Si vous êtes curieux : vous pouvez regarder l'automate fourni par menhir :*

```
menhir --dump parser.mly.
```

Par défaut, menhir, ne fait pas forcément du LR(1), mais on peut lui demander l'automate LR(1) :

```
menhir --dump --canonical parser.mly
```

Si vous avez déjà déclaré les opérateurs avec leur précédences et leur associativités, les conflits des états seront résolus en choisissant une action plutôt qu'une autre.

Exercice 3 *On souhaite ajouter la négation qu'on notera* $\sim e$*. Modifier la grammaire et les différents fichiers en conséquence. Il faudra utiliser la directive* `%nonassoc NOT` *placée de manière judicieuse pour avoir la bonne précédence.*

Exercice 4 *On va ajouter la possibilité d'avoir des expressions de la forme*

```
if e then e else e
if e then e
```

Notez que la deuxième syntaxe est invalide en OCaml *pour des expressions booléennes. Ici, on décidera par convention que si* `e1` *est évalué à false, le* `if e1 then e2` *sera évalué à false.*