Compte Rendu - Chapitre 2
Analyse syntaxique descendante

Graignic Guillaume
Olivier Cadoret

I/ Nouvelle grammaire LL(1)

<Fichier> --> <Expr> "/000"
<Expr> --> <Termb> <SuiteExpr>
<SuiteExpr> --> "ou" <Termb> <SuiteExpr> | ε
<Termb> --> <Facteurb> <SuiteTermb>
<SuiteTermb> --> "et" <Facteurb> <SuiteTermb> | ε
<Facteurb> --> <Relation> | "(" <Expr> ")" |
"si" <Expr> "alors" <Expr> "sinon" <Expr> "fsi"
<Relation> --> <Ident> <Op> <Ident>
<Op> --> "=" | "<>" | "<" | ">" | ">=" | "<="

II/ Preuve de la propriété LL(1)

Montrons que la grammaire est LL(1)
Il y a trois non-terminaux avec alternants
Pour <SuiteExpr> --> "ou" <Termb> <SuiteExpr> | ε :
premier("ou" <Termb> <SuiteExpr>) = {"ou"} et premier(ε) = ∅ donc
premier("ou" <Termb> <SuiteExpr>) ⋂ premier(ε) = ∅
null(<SuiteExpr>) avec premier(<SuiteExpr>) = {"ou"} et
suivant(<SuiteExpr>) = {<Termb>} toujours ≠ {"ou"}
premier(<SuiteExpr>) ⋂ suivant(<SuiteExpr>) = ∅
Il existe un unique alternant ε tq null(ε)
Pour <SuiteTermb> --> "et" <Facteurb> <SuiteTermb> | ε :
premier("et" <Facteurb> <SuiteTermb>) = {"et"} et premier(ε) = ∅ donc
premier(("et" <Facteurb> <SuiteTermb>) ⋂ premier(ε) = ∅
null(<SuiteTermb>) avec premier(<SuiteTermb>) = {"et"} et
suivant(<SuiteTermb>) = {<Facteurb>} toujours ≠ {"et"}
premier(<SuiteTermb>) ⋂ suivant(<SuiteTermb>) = ∅
Il existe un unique alternant ε tq null(ε)
Pour <Facteurb> --> <Relation> | "(" <Expr> ")" |
"si" <Expr> "alors" <Expr> "sinon" <Expr> "fsi"
premier(<Relation>) = <Ident> qui commence toujours par une lettre,
premier("(" <Expr> ")") = {"("} et
premier("si" <Expr> "alors" <Expr> "sinon" <Expr> "fsi") = {"si"} donc
premier(<Relation>) ⋂ premier("(" <Expr> ")") ⋂ premier("si" <Expr> "alors" <Expr> "sinon" <Expr> "fsi") = ∅ et non null(<Facteurb>)
D'après le corollaire la grammaire est donc LL(1)

III/ Réponses aux questions

Question 2.1
Les commentaires peuvent être insérer à n'importe quel endroit du code et donc, ne relève pas de l'analyse syntaxique. C'est pour cela qu'ils sont ignorés lors de l'analyse lexicale.

Question 2.2
Le fait de construire un arbre et de tester la correspondance des types de chaque unité lexicale remplace la pile. Avec cette implémentation, on est aussi garanti d'avoir un bon parenthèsage.

Question 2.3
On utilise une grammaire LL(1) de manière à avoir une seule dérivation de règle possible lorsque l'on se place sur la première unité lexicale de celle-ci. Cela rend l'implémentation du compilateur beaucoup plus simple car il suffit de regarder la première unité lexicale de la liste fournit par l'analyseur lexical pour savoir comment dériver la règle courante.

Question 2.4
Un arbre abstrait est beaucoup plus simple et agréable à lire.
En effet, dans un arbre concret, toutes les différentes dérivations apparaissent (même les dérivations en epsilon) ce qui rend celui-ci très pénible à lire, contrairement à un arbre abstrait qui ne fait apparaitre que les informations intéressantes.

IV/ Code et jeux de tests commentés