Primeiro Exercício Escolar de Compiladores 2012.1

Prof. Marcelo d'Amorim

Aluno:

**1) (2.0) Associe cada tarefa abaixo a uma parte do front-end: (L) análise léxica, (S) sintática, e (C) contextual (semântica). Use (T) caso a tarefa se aplique a todas as partes e (N) caso não se aplique a nenhuma das partes.**

( C ) Verifica se número e ordem de parâmetros na chamada de uma função corresponde ao declarado na função

( T ) Reporta erros ao usuário

( L ) Lê e escreve na tabela de símbolos

( L ) Usa expressões regulares na sua descrição

( S ) Verifica se o “else” aparece sempre depois de um “if”

( N ) É responsável pela geração de código

( C ) Identifica o tipo de uma expressão

( N ) Realiza otimizações no código

**2) Considere a BNF abaixo usada para descrever a gramática usada nas próximas questões**:

Program ::= Decls Stmts Expr

Decls ::= TypeDecl **;** Decls | TypeDecl

TypeDecl ::= Type **Id**

Type ::= **int | float**

Stmts ::= Stmt **;** Stmts | Stmt

Stmt ::= Assign

Assign ::= **Id** **=** Expr

Expr ::= NumExpr

NumExpr := BinNumExpr | **Number | (** NumExpr **)**

BinNumExpr ::= NumExpr **Aop** NumExpr

Id, Number, e Aop são definidos pelas expressões regulares abaixo:

Id ::= [a-ZA-Z]+

Number ::= [0-9]+

Aop ::= '+' | '-' | '\*' | ‘/'

a) (2.0) Quais característica(s) da gramática dificulta(m) a implementação de um Recursive Descent Parser (RDP)? Identifique e explique.

As recussões à esquerda, pois podem provocar um loop infinito, e ambiguidades, pois podem gerar duas ASTs diferentes, como no caso de se ter a expressão 5 + 3 + 2, que pode ser gerada através da derivação do primeiro NumExpr em BinNumExpr como sendo BinNumExpr gerando Number Aop Number (resultando em 5 + 3) ou do segundo NumExpr em BinNumExpr ( de forma análoga resultando em 3 + 2).

b) (1.0) Estas modificações, por vezes, são trabalhosas e não triviais (questão abaixo sobre isto)! Que recurso outros tipos de parsers (não preditivos como o RDP) usam para reconhecer a gramática acima sem necessitar de modificações?

Podem utilizar regras de precedência para eliminar as ambiguidades.

c) (2.0) Modifique a gramática acima para habilitar geração de recursive-descent parser.

Program ::= Decls Stmts Expr

Decls ::= TypeDecl **;** Decls | TypeDecl

TypeDecl ::= Type **Id**

Type ::= **int | float**

Stmts ::= Stmt **;** Stmts | Stmt

Stmt ::= Assign

Assign ::= **Id** **=** Expr

Expr ::= NumExpr

NumExpr := BinNumExpr | **Number | (** NumExpr **)**

BinNumExpr ::= **Number** **Aop** NumExpr

b) (1.0) Explique diferenças de uma árvore abstrata para uma árvore concreta. Mostre exemplo desta diferenças para uma árvore produzida a partir de um parser da BNF acima.

A árvore abstrata não se preocupa com detalhes intrínsecos às regras, como o uso de parêntesis e ponto e vírgula; já as árvores concretas implementam esses detalhes.

Abstrata:

Program

/ | \

TypeDecl = \*

/ \ / \ / \

int x x 2 3 +

/ \

4 5

Concreta:

Program

/ | \

Decls Stmts Expr

/ | \

TypeDecl Stmt NumExpr

/ \ | \

int x Assign BinNumExpr

/ | \ / | \

id = Expr Number Aop NumExpr

| | | | / | \

x NumExpr 3 \* ( NumExpr )  
 | |

Number BinNumExpr

| / | \

2 Number Aop NumExpr

| | |

4 + Number

|

5

e) (2.0) Qual a relação entre gramática de atributos e visitors. Explique.

A gramática de atributos cria regras para atributos herdados ou sintetizados e os visitors utilizam essas regras na implementação de um sistema de tipo simples.