1EE de Teoria e Implementação de Linguagens Computacionais (Compiladores)

Prof. Marcelo d'Amorim, 15 de Abril de 2011

Aluno: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (2 pontos) Reponda com “V” ou “F” as afirmações abaixo

[ V ] Um parser para uma gramática ambígua pode gerar várias árvores sintáticas.

[ V ] Identificar se uma variável usada em uma expressão foi inicializada requer informação contextual. Esta tarefa é, portanto, responsabilidade do analisador semântico.

[ V ] Parsers para gramáticas ambíguas precisam realizar backtracking. [apenas os preditivo não precisa]

[ V ] Recursão à esquerda precisa ser elminada para construir parser recursivo descente.

[ F ] Comparado com parsers top-down, parsers bottom-up são, em geral, menos eficientes e suportam uma classe menor de gramáticas; porém, são mais simples de se construir e manter.

[ V ] Gramáticas de atributo permitem definição de regras semânticas contextuais. Podem ser usadas, portanto, para descrição de um sistema de tipos simples e de uma gerador de código.

[ V ] O padrão de projeto Visitor é útil para se navegar na estrutura de uma grafo de objetos. Em particular, na estrutura de uma árvore sintática.

[ V ] O front-end de um compilador inclui dois componentes: o lexer e o parser.

2. (4 pontos) Um projetista escreveu um parser recursive-descent para uma linguagem imperativa simples, cuja gramática é caracterizada pela BNF abaixo. Ele precisa adicionar comando de loop “while” e “do-while” à sua gramática. Neste momento, o loop precisa apenas suportar uma condição de parada que checa se um número é maior que o outro. Mostre as mudanças que ele precisa fazer (i) na gramática abaixo e (ii) em seu parser. Nota: não é preciso escrever o parser inteiro; assuma dadas as funções recursivas para os símbolos não-terminais Stmt, Statements, Cmd, e Exp.

Stmt => Statements

Statements => Cmd **;** Statements | Cmd

Cmd => **Id** **:=** Exp | **print** **(** **Id** **)**

Exp => **Num** **+** Exp | **Num**

Mudanças na gramática:

Stmt => Statements Stmt

Statements => Cmd **;** Statements ComposedStmt

| Cmd SingleStmt

Cmd => **Id :=** Exp AssignmentCmd

| **print ( Id )**  PrintCmd

| “while” ( Num > Num ) “do-while” { Statements } WhileCmd

Exp => **Num** + Exp Add

| **Num**  Literal

Mudança no parser (especificamente Cmd):

void Cmd () {

switch (lookahead) {

case (TO\_ID): match (TO\_ID); match (TO\_ASG); Exp(); break;

case (TO\_PRINT): match (TO\_PRINT); match (TO\_OP); match (TO\_ID); match (TO\_CL); break;

case (TO\_WHILE): match (TO\_WHILE); match (TO\_OP); match (TO\_Num); match (TO\_BIGGER); match (TO\_Num); match (TO\_CL); match (TO\_DO\_WHILE); match (TO\_OP2); Statements(); match (TO\_CL2); break

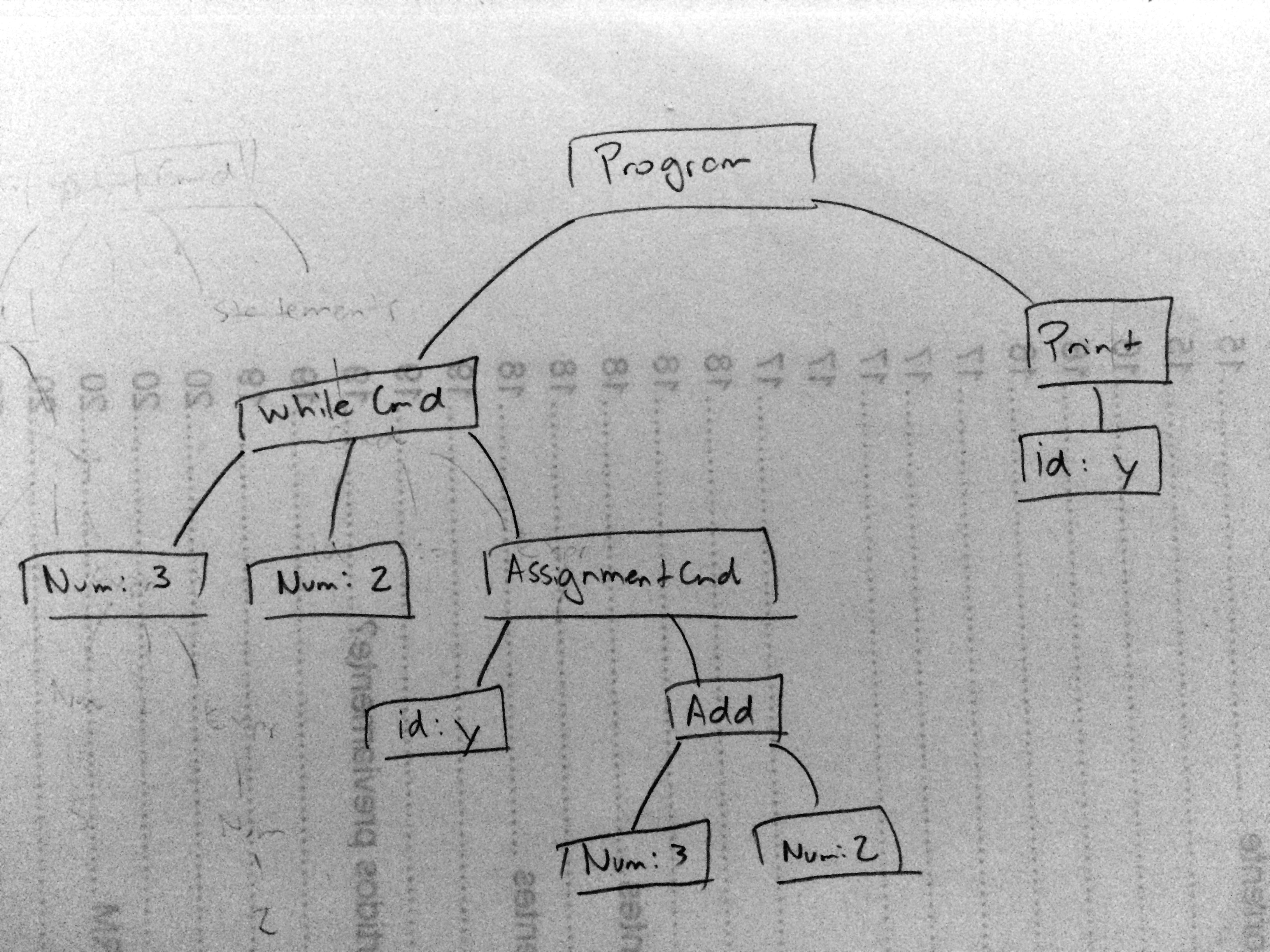
default: syntax\_error ();

}

}

Considerando: ASG representando “:=“, BIGGER representando “>”, OP representando “(“, CL representando “)”, OP2 representando “{“ e CL2 representando “}”.

3. (1 ponto) Derive uma string que pertença a linguagem acima e mostre sua árvore sintática abstrata.

“while” (3 > 2) “do-while” { y = 3 + 2 }; print (y)

4. (2 pontos) O projetista identificou que um erro que programadores de sua linguagem frequentemente cometem é escrever expressões envolvendo variáveis não inicializadas. Mostre, da forma mais detalhada possível, como ajudá-lo a identificar este problema em tempo de compilação.

O problema pode ser identificado através da definição de um atributo sintetizado reach, que armazena um conjunto de variáveis que são usadas e estão potencialmente sem definição. Para cada novo uso é necessário atualizar esse conjunto. Além isso, se for encontrada uma declaração para a variável, ela deve ser eliminada do conjunto. Dessa forma, caso o atributo reach não for vazio no nó raiz, é porque há variável não sendo inicializada, necessitando retornar um erro.

5. (1 ponto) Explique a diferença de atributos sintetizados e herdados em uma gramática de atributos. crie um exemplo que destaque esta diferença e explique como isto é implementado.

Os atributos sintetizados dependem dos nós filhos, enquanto os herdados dependem dos nós pai e irmão.

Exemplo:

D 🡒T L

T 🡒 **int**

T 🡒 **real**

L 🡒L1 **, id**

L 🡒 **id**

Implementação:

L.in = T.type (1)

T.type = integer (2)

T.type = real (2)

L1.in = L.in (3)

addtype(id.entry, L.in) (3)

Neste caso o atributo type é sintetizado, pois T recebe a informação dos filhos int ou real (2).

Já o atributo in é herdado, cada L1gerado vai receber a informação do pai L (3), que por sua vez pai recebeu a informação do irmão T (1).