

Capítulo IV

Histórico de revisões:

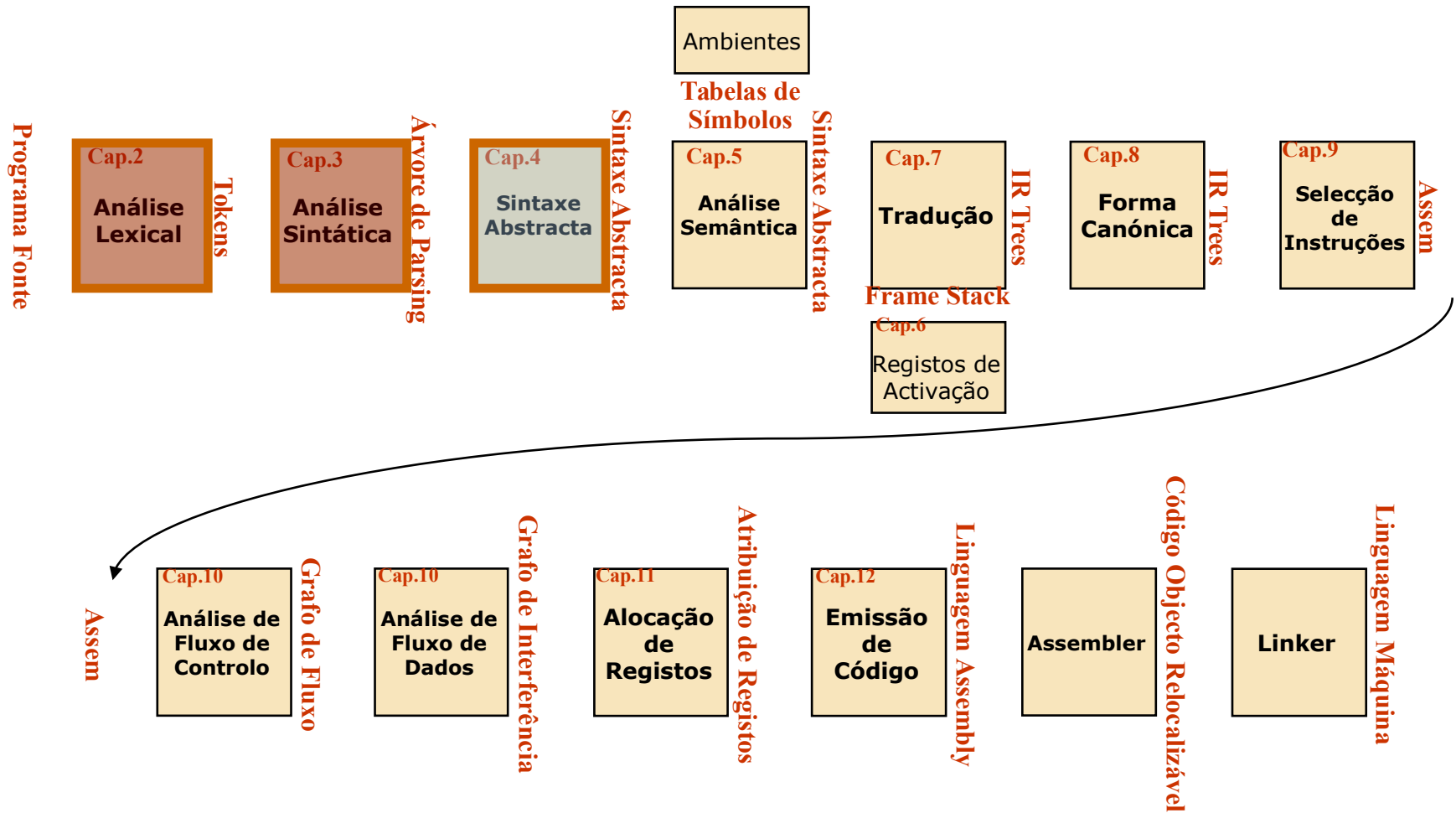
2010-2014 – Carlos M. Fonseca

2008-2010 – Luís Macedo

Anos anteriores – F. Amílcar Cardoso

Onde estamos...?

2



Tradução Orientada pela Sintaxe

3

- Não basta reconhecer se uma frase pertence a uma linguagem: é preciso perceber o seu significado
- A Tradução Orientada pela Sintaxe permite associar informação (valor) a uma linguagem através da associação de atributos aos símbolos da gramática
- Duas abordagens:
 - Regras Semânticas - Definição Orientada pela Sintaxe (“Syntax-Directed Definition” - SDD)
 - Acções Semânticas - Esquemas de Tradução Orientada pela Sintaxe (“Syntax-Directed Translation Schemes” - SDT)

SDD-Regras Semânticas e Atributos

4

- Para cada produção da gramática cria-se um conjunto de regras que vão permitir a determinação do valor semântico
- Notação:
 - Seja X um nó da árvore de derivação e a um atributo de X
 - $X.a$ representa o valor do atributo a de X

PRODUCTION

$E \rightarrow E_1 + T$

SEMANTIC RULE

$E.code = E_1.code \parallel T.code \parallel '+'$

SDD -Atributos

5

□ Atributos:

- Sintetizados (“synthetized”):
 - Definidos por uma regra semântica da produção no nó da árvore de derivação com base nos atributos no próprio nó e nos filhos
- Herdados (“inherited”):
 - Definidos por uma regra semântica da produção no nó-pai da árvore de derivação com base nos atributos no próprio nó, nos pais e irmãos

SDD -Tipos

6

- 2 tipos de SDDs:
 - “S-attributed SDD”: envolve somente atributos sintetizados
 - “L-attributed SDD”

Exemplo: conversão infix / postfix

7

$(9 - 5) + 2 \rightarrow 9\ 5 - 2 +$

Produções

$E \rightarrow E + T$

$E \rightarrow E - T$

$E \rightarrow T$

$T \rightarrow 0$

$T \rightarrow 1$

...

$T \rightarrow 9$

Regras Semânticas

$E.n := E1.n \bullet T.n \bullet '+'$

$E.n := E1.n \bullet T.n \bullet '-'$

$E.n := T.n$

$T.n := '0'$

$T.n := '1'$

...

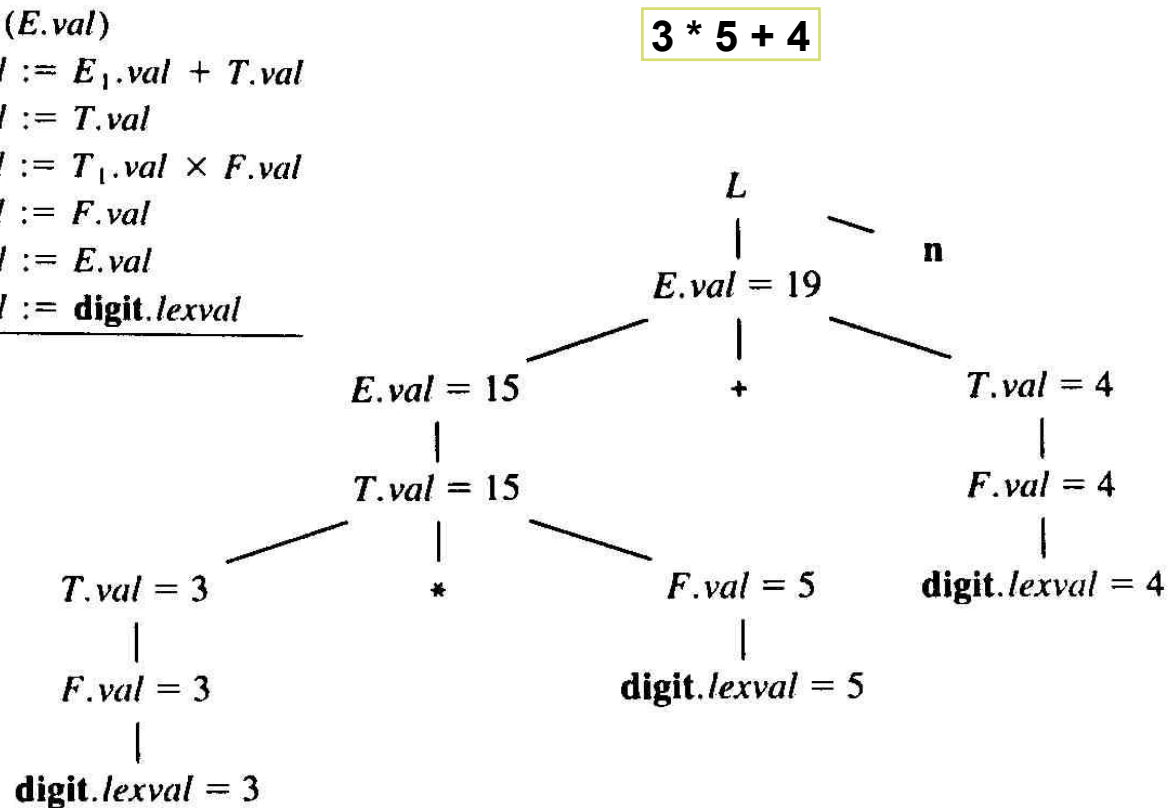
$T.n := '9'$

SDD – Avaliação:

Exemplo: Calculadora Simples

8

PRODUCTION	SEMANTIC RULES
$L \rightarrow E \text{ n}$	$\text{print}(E.val)$
$E \rightarrow E_1 + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T_1 * F$	$T.val := T_1.val \times F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow \text{digit}$	$F.val := \text{digit.lexval}$



SDD – Ordem de Avaliação

9

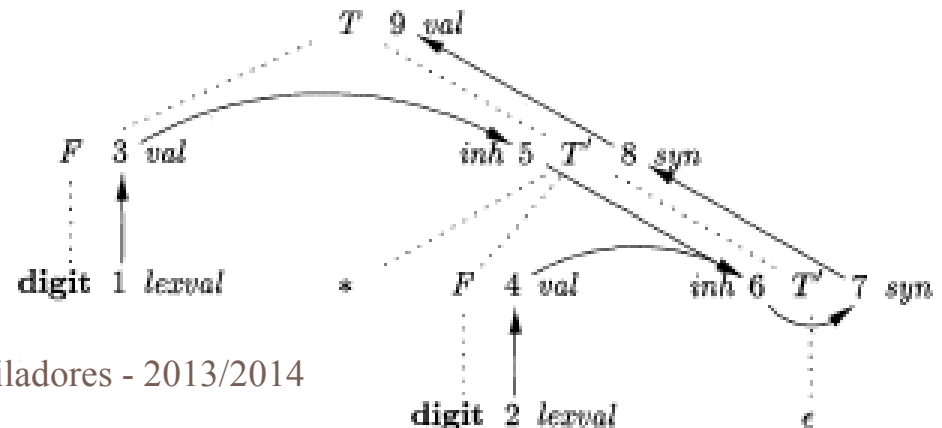
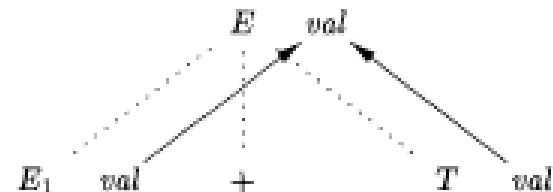
- Grafos de dependência: grafo para determinar a ordem de avaliação dos atributos numa árvore de derivação
- Exemplo:

PRODUCTION

$E \rightarrow E_1 + T$

SEMANTIC RULE

$E.val = E_1.val + T.val$



SDT – Acções Semânticas

10

- Fragmentos de programa (acções semânticas) embebidos nos lados direitos das produções
- Diferença para SDD: a ordem de avaliação está explicitamente especificada; pode não ser construída a árvore de derivação

Dois exemplos:

Produção	Regra Semântica	Produção Anotada
$E \rightarrow E + T$	$E.n := E1.n \bullet T.n \bullet '+'$	$E \rightarrow E + T \{ \text{print} ('+') \}$
Produção	Regra Semântica	Produção Anotada
$R \rightarrow + T R$	$R.n := T.n \bullet '+' \bullet R1.n$	$R \rightarrow + T \{ \text{print} ('+') \} R$

Exemplo: conversão infix / postfix

11

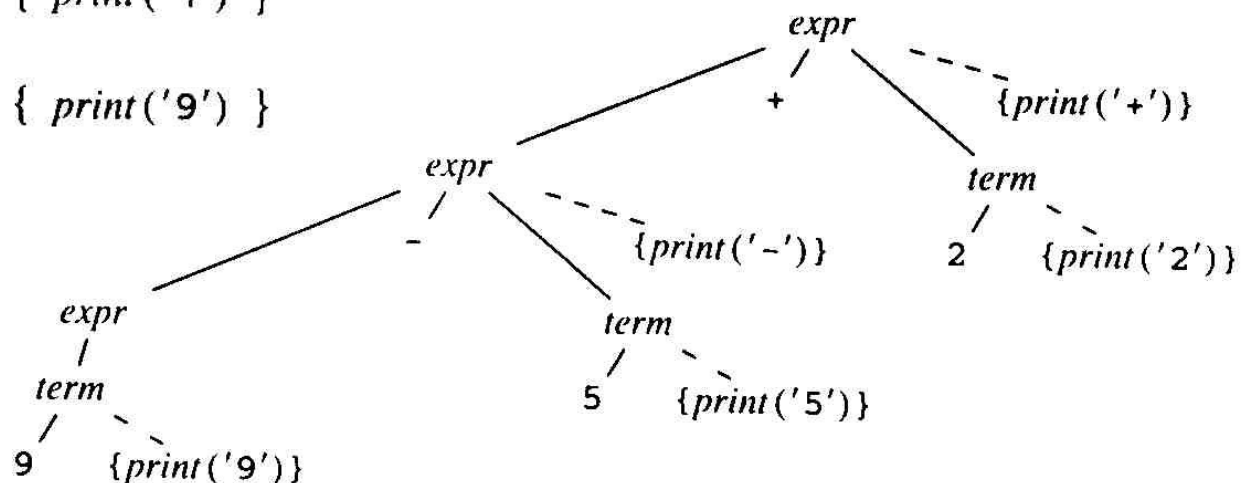
$(9 - 5) + 2 \rightarrow 9\ 5 - 2 +$

Produções	Regras Semânticas	Produções Anotadas
$E \rightarrow E + T$	$E.n := E1.n \bullet T.n \bullet '+'$	$E \rightarrow E + T \{\text{print}(' + ')\}$
$E \rightarrow E - T$	$E.n := E1.n \bullet T.n \bullet '-'$	$E \rightarrow E - T \{\text{print}(' - ')\}$
$E \rightarrow T$	$E.n := T.n$	$E \rightarrow T$
$T \rightarrow 0$	$T.n := '0'$	$T \rightarrow 0 \{\text{print}('0')\}$
$T \rightarrow 1$	$T.n := '1'$	$T \rightarrow 1 \{\text{print}('1')\}$
...
$T \rightarrow 9$	$T.n := '9'$	$T \rightarrow 9 \{\text{print}('9')\}$

Exemplo: conversão infix / postfix

12

$expr \rightarrow expr + term$ { $print(' +')$ }
 $expr \rightarrow expr - term$ { $print(' -')$ }
 $expr \rightarrow term$
 $term \rightarrow 0$ { $print(' 0')$ }
 $term \rightarrow 1$ { $print(' 1')$ }
...
 $term \rightarrow 9$ { $print(' 9')$ }

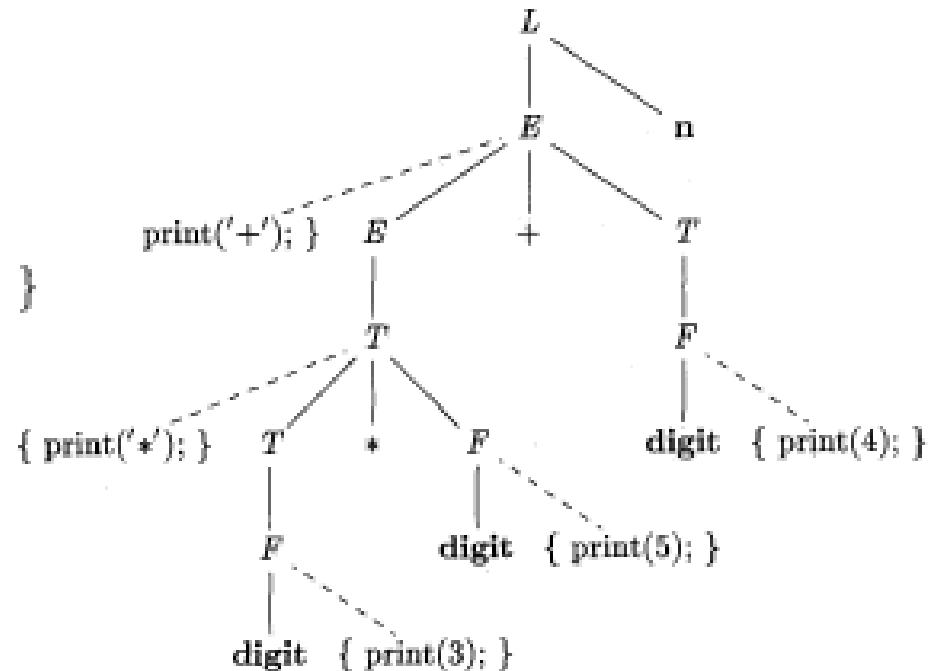


Resultado (travessia em profundidade):
(9 - 5) + 2 → 9 5 - 2 +

Exemplo: conversão infix / prefix

13

- 1) $L \rightarrow E \ n$
- 2) $E \rightarrow \{ \text{print}(' + '); \} \ E_1 + T$
- 3) $E \rightarrow T$
- 4) $T \rightarrow \{ \text{print}(' * '); \} \ T_1 * F$
- 5) $T \rightarrow F$
- 6) $F \rightarrow (E)$
- 7) $F \rightarrow \text{digit} \ \{ \text{print}(\text{digit.lexval}); \}$



Resultado (travessia em profundidade):
 $3 * 5 + 4 \rightarrow + * 3 5 4$

Syntax-Directed Translation - Aplicações

14

- Construção da Árvore de Sintaxe Abstracta (“(Abstract) Syntax Trees” – AST)

- Pode escrever-se um compilador só à custa de Acções Semânticas
 - Problema:
 - questões sintáticas e semânticas tratadas conjuntamente
 - modularidade comprometida
 - compilador tem que atravessar programa pela mesma ordem que o parser o fez

AST

16

- Compiladores de duas passagens:
 - mais modulares
 - evitam necessidade de declarações forward
 - necessitam de muita memória
- Solução possível:
 - Basta que o parser construa uma estrutura de dados que possa mais tarde ser atravessada por outros módulos

- Árvore de Sintaxe Concreto (árvore de derivação – “parse trees”):
 - uma folha por cada token
 - um nó interno por cada regra gramatical reduzida
 - problemas:
 - árvore demasiadamente dependente da gramática
 - sinais de pontuação e certas palavras reservadas tornam-se desnecessárias após análise sintática
 - símbolos não-terminais acrescentados para eliminar ambiguidades e recursividade à esquerda, ou para factorização, não deviam interferir na análise semântica

AST

18

□ Árvores de Derivação \neq AST

- interface "limpa" entre parser e restantes módulos
- estrutura frásica do programa fonte
- parsing resolvido

$E \rightarrow T E'$	$T \rightarrow F T'$	$F \rightarrow \text{id}$
$E' \rightarrow + T E'$	$T' \rightarrow * F T'$	$F \rightarrow \text{num}$
$E' \rightarrow - T E'$	$T' \rightarrow / F T'$	$F \rightarrow (E)$
$E' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$	



$E \rightarrow E + E$
$E \rightarrow E - E$
$E \rightarrow E * E$
$E \rightarrow E / E$
$E \rightarrow \text{id}$
$E \rightarrow \text{num}$

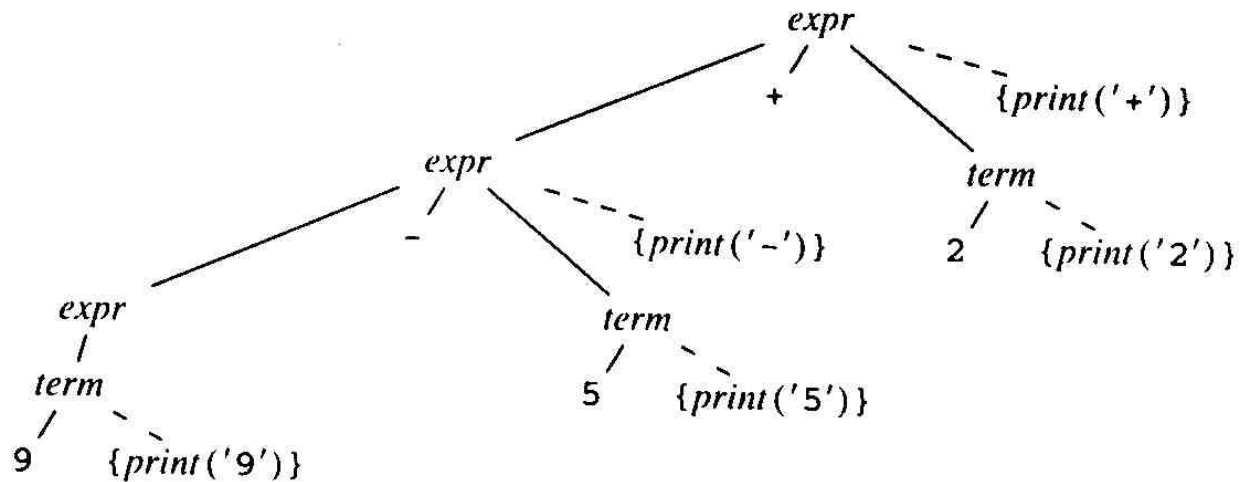
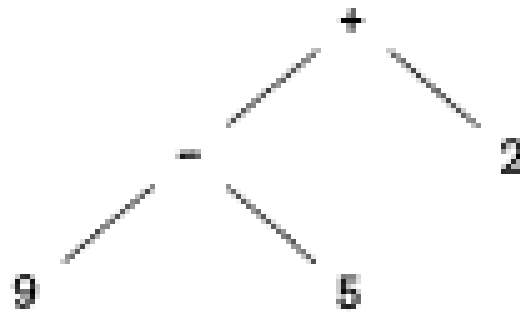
Péssima para parsing.
Porquê?

■ Construção da Árvore:

- Regras Semânticas
- Acções Semânticas

AST

19



AST

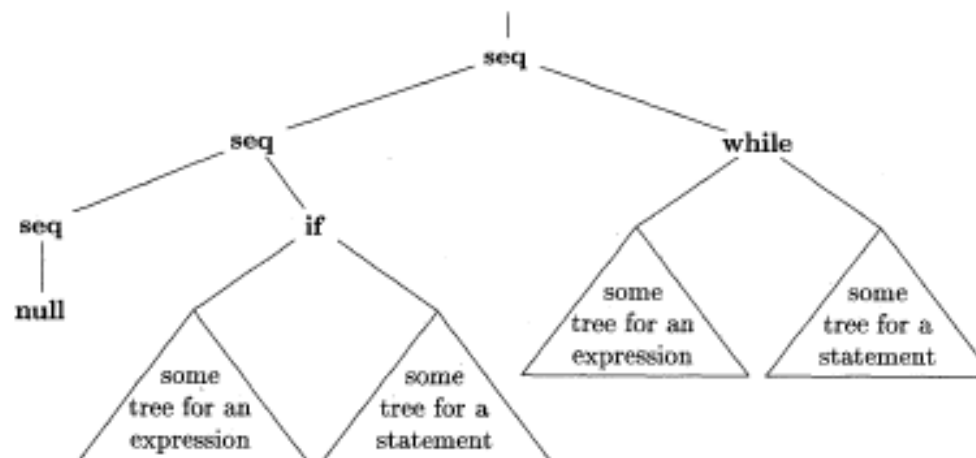
20



$term \rightarrow term_1 * factor \quad \{ term.n = \text{new Op}('* ', term_1.n, factor.n); \}$

$stmt \rightarrow \text{if} (expr) stmt_1 \quad \{ stmt.n = \text{new If}(expr.n, stmt_1.n); \}$

$stmts \rightarrow stmts_1 stmt \quad \{ stmts.n = \text{new Seq}(stmts_1.n, stmt.n); \}$



AST

21

```
program → block          { return block.n; }

block → '{' stmts '}'     { block.n = stmts.n; }

stmts → stmts1 stmt      { stmts.n = new Seq(stmts1.n, stmt.n); }
      | ε                  { stmts.n = null; }

stmt → expr ;             { stmt.n = new Eval(expr.n); }
      | if ( expr ) stmt1 { stmt.n = new If(expr.n, stmt1.n); }
      | while ( expr ) stmt1 { stmt.n = new While(expr.n, stmt1.n); }
      | do stmt1 while ( expr ); { stmt.n = new Do(stmt1.n, expr.n); }
      | block              { stmt.n = block.n; }

expr → rel = expr1       { expr.n = new Assign('=', rel.n, expr1.n); }
      | rel                { expr.n = rel.n; }

rel → rel1 < add          { rel.n = new Rel('<', rel1.n, add.n); }
      | rel1 <= add        { rel.n = new Rel('<=', rel1.n, add.n); }
      | add                 { rel.n = add.n; }

add → add1 + term         { add.n = new Op('+', add1.n, term.n); }
      | term                { add.n = term.n; }

term → term1 * factor      { term.n = new Op('*', term1.n, factor.n); }
      | factor              { term.n = factor.n; }

factor → ( expr )          { factor.n = expr.n; }
        | num              { factor.n = new Num(num.value); }
```

Construção da AST

22

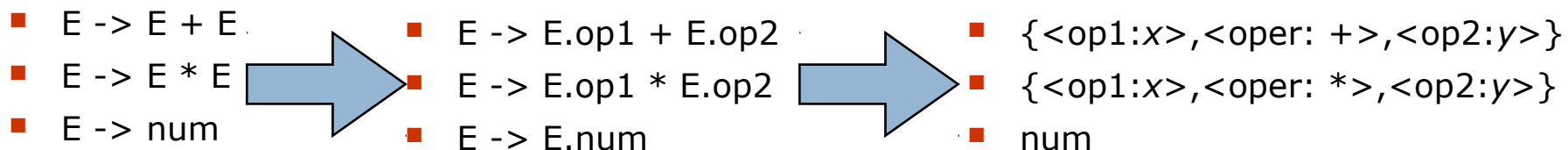
- Existem normas para construir as AST
- Começamos por utilizar uma representação - a Especificação de Sintaxe Abstracta
- Formada pelos componentes essenciais da linguagem
- Permite a derivação das Estruturas Gerais de Dados

Especificação de Sintaxe Abstracta

23

□ Objectos

- Objecto elementar - corresponde à folha da árvore (e.g. símbolos terminais)
- Objecto estruturado - conjunto de um ou mais pares da forma **<s , a>**
 - **s** é o selector (e.g. pode ser visto como “atributo”)
 - **a** é um objecto (elementar ou estruturado)



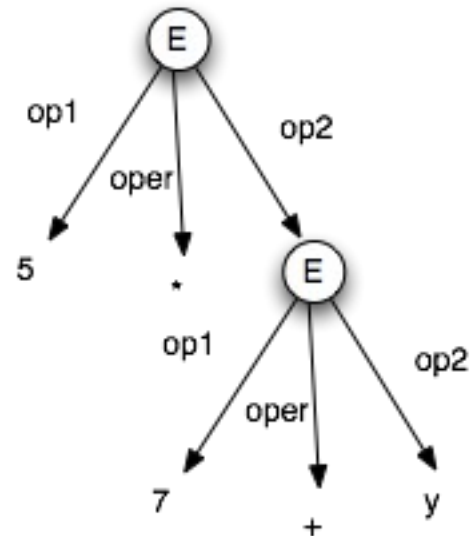
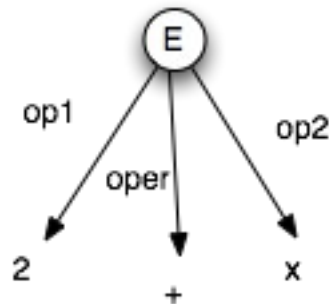
Especificação de Sintaxe Abstracta

24

Exemplos

objecto 2={<op1:5>, <oper:*>,<op2:{<op1: 7>, <oper: +>, <op2: y>}>}

objecto 1={<op1:2>, <oper:+>,<op2:x>}



□ Selectores: op1, op2, oper

□ Objectos elementares: 2, 7, x, y, +, *

Especificação de Sintaxe Abstracta

25

□ Predicados

- Um predicado é uma função que toma um objecto estruturado e retorna o objecto elementar T (true) ou F (false)
- Por convenção, cada predicado começa com o prefixo “is_”
- Se o tipo de objectos que o predicado toma como argumento for uma lista, também se junta o sufixo “_list”
- Um predicado pode consistir na disjunção de outros predicados

Especificação de Sintaxe Abstracta

26

□ Exemplos

- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow \text{num}$

- $\text{is_E} \rightarrow \text{is_operation } V \text{ is_number}$
- $\text{is_operation} \rightarrow \{ \langle \text{op1: is_E} \rangle, \langle \text{oper: is_oper} \rangle, \langle \text{op2: is_E} \rangle \}$
- $\text{is_oper} \rightarrow + \vee *$
- $\text{is_number} \rightarrow \langle \text{val: num} \rangle$

Especificação de Sintaxe Abstracta

27

□ Método

- Pegar em cada regra da gramática
 - Se tem a configuração de uma lista (recursividade: $A \rightarrow \alpha A \beta$), criar um predicado “is ... list” -> juntar esse predicado isolado também no final!
 - Senão, se há uma disjunção criar um predicado para cada sub-produção
 - Senão, criar um objecto estruturado com todos os elementos que compõem a regra
- Deixar de fora tudo o que fôr “açucar sintactico”. Por exemplo, sinais de pontuação, tokens sem valor

Especificação de Sintaxe Abstracta

28

□ Exemplos

- Gramática:

```
statement:  statement expression
           |  expression

expression: infix_expression
           |  unary_expression
           |  NUMBER

infix_expression: expression oper expression

unary_expression: '-' expression

oper: '-' | '+'
```

- Sintaxe Abstracta:

```
is_statement → is_expression_list
is_expression → is_infix_expression ∨ is_unary_expression ∨ is_NUMBER
is_infix_expression → (<exp1: is_expression><oper:is_oper><exp2:is_expression>)
is_unary_expression → (<exp: is_expression>)
is_oper → is_PLUS ∨ is_MINUS
is_expression_list
```

Derivação da Estrutura de Dados da AST

29

- A partir de uma Especificação da Sintaxe Abstracta, é possível definir directamente as estruturas de dados que compõem a AST
- Esta árvore vai ser usada pelo Compilador durante os módulos de Análise Semântica e Tradução
- Cada nó da árvore corresponderá a um objecto estruturado. Em C, deverá ser representado por uma struct. Em Java, poderá ser com uma classe.

Derivação da Estrutura de Dados da AST em C

30


□ Metodologia:

1. Para um objecto especificado por um predicado P_i definido por pares individualizados ($\langle s_m: a_m \rangle, \langle s_n: a_n \rangle, \dots$), a estrutura de dados será a seguinte:

```
typedef struct _a1 {  
    type_  $a_m$   $s_m$ ;  
    type_  $a_n$   $s_n$ ;  
}  $P_i$ ;
```

● Exemplo:

$is_infix_expression \rightarrow (\langle exp1: is_expression \rangle \langle oper: is_oper \rangle \langle exp2: is_expression \rangle)$



```
typedef struct _a4 {  
    is_expression *exp1;  
    is_oper oper;  
    is_expression *exp2;  
} is_infix_expression;
```

Derivação da Estrutura de Dados da AST em C


31

□ Metodologia:

1. A um objecto do tipo lista P_d_list , deverá corresponder uma lista ligada do tipo:

```
typedef struct _a2 {  
    Pd *v ;    /* Elemento de uma lista de  
    objectos Pd */  
    struct _a2 *next;  
} Pd_list;
```

- Exemplo: *is_expression_list*



```
typedef struct _a2 {  
    is_expression *expr;  
    struct _a2 *next;  
}is_expression_list;
```

Derivação da Estrutura de Dados da AST em C

32

□ Metodologia:

1. Para objectos especificados por um predicado P_d definido por uma disjunção de predicados $P_1 \vee P_2 \vee \dots \vee P_n$, o seguinte par de estruturas de dados constitui a definição de dados:

```
typedef enum {d_P1, d_P2, ...d_Pn} disc_Pd
```

```
typedef struct _a1 {  
    disc_Pd disc_d;  
    union{  
        p1 u_P1;  
        ⋮  
        pn u_Pn;  
    } data_Pd;  
} Pd;
```


Derivação da Estrutura de Dados da AST em C

33

□ Metodologia:

1. Para objectos especificados por um predicado P_d definido por uma disjunção de predicados $P_1 \vee P_2 \vee \dots \vee P_n$, o seguinte par de estruturas de dados constitui a definição de dados.

● Exemplo:

is_expression \rightarrow *is_infix_expression* \vee *is_unary_expression* \vee *is_NUMBER*



```
typedef enum {d_infix_exp, d_unary_exp, d_number} disc_expression;

typedef struct _a3 {
    disc_expression disc_d;
    union{
        is_infix_expression *u_infix_exp;
        is_unary_expression *u_unary_exp;
        int number;
    }data_expression;
}is_expression;
```

Implementação da construção da AST

34

- Utilizando as estruturas de dados dadas, podemos implementar a construção da AST
- As “acções semânticas” do compilador vão permitir criar os nós da AST à medida que é feito o parsing
- Assim, fazemos uma passagem para construir a AST, que fica na memória...
- A próxima passagem será feita directamente na AST!

Compiladores de 2 passagens

35

□ Vantagem:

- não precisam de fazer análise sintática e análise semântica de uma vez só
- primeira passagem produz AST
- análise semântica trabalha sobre a AST

□ Problema:

- a AST não tem referências ao código original
- como mostrar ao programador a localização (linha no código fonte) de erros semânticos?

□ Solução:

- referenciar cada nó com a posição do código correspondente

Resumo / Bibliografia

36

- Sintaxe abstracta [Dragão, ch5;Appel,ch4;Crespo, ch6-Sec6.3...; Crespo, ch7,pp243-262;Crespo, ch1, Sec 2.8-2.9, pp67-78]
 - Pre-requisitos:
 - Gramáticas Atributivas [Crespo, pp67...]
 - Árvores sintaxe decorada [Crespo,pp 218...]
 - Gramática Tradutora
 - SDD [Dragão, ch5, ch2 (Sec 2.3)]
 - Tipos de atributos
 - Tipos de SDD
 - Avaliação de SDD
 - SDT [Dragão2006, ch5, sec. 2.5)]
 - Aplicações da Syntax-Directed Translation [Dragão2006,sec.5.3]
 - AST [Appel, ch4; Dragão2006,secs. 2.5, 2.8.1, 2.8.2]
 - Especificação e Implementação da construção da AST [Crespo, ch1,secs. 2.8-2.9; Crespo ch7,pp.243-262; Ficha6]

