COMPILADORES
TRADUÇÃO DIRIGIDA POR SINTAXE

Roberto S. Bigonha e Mariza A. S. Bigonha

UFMG

9 de agosto de 2011

Regras Semânticas + Produções

- Definições Dirigidas pela Sintaxe São especificações de mais alto nível para as traduções. Elas escondem vários detalhes de implementação e liberam os usuários da especificação explícita da ordem em que a tradução deve acontecer.

- Esquemas de Tradução Indicam a ordem na qual as regras semânticas devem ser avaliadas, portanto permitem que detalhes de implementação sejam mostrados.

- A avaliação das Regras Semânticas pode:
  - gerar código.
  - Salvar informações na Tabela de Símbolos.
  - Emitir mensagens de erro.
  - Efetuar qualquer outra tarefa.

Definição Dirigida por Sintaxe

- **Conceito: gramática + definição de atributos.**

Gramática livre do contexto na qual às produções são associadas ações semânticas.

- Ascendente

$$A \rightarrow XYZ \{ R \}$$

- **Por ocasião da redução de XYZ para A executa-se a ação R.**

- Descendente

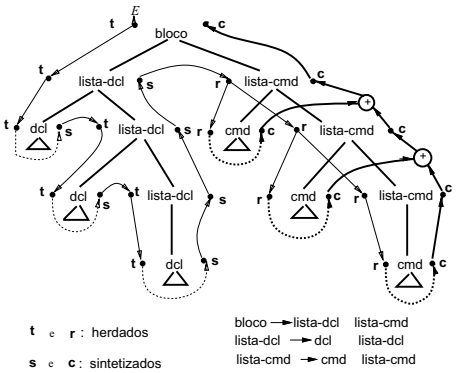
$$A \rightarrow XYZ \{ R \}$$

- **A ação R é executada no momento em que A, X, Y ou Z é expandido dependendo da conveniência.**

Atributos

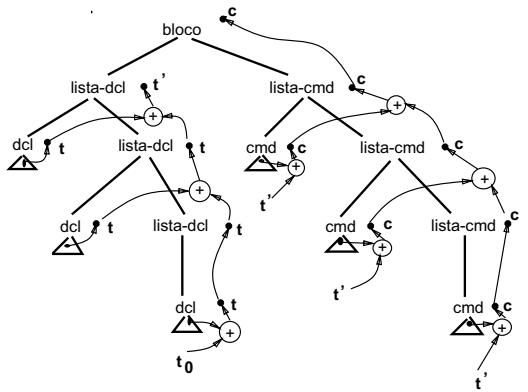
- Sintetizados (synthesized)

- Herdados (inherited)



Atributos

- Compilação usa só "sintetizados" .
- A tabela de símbolos funciona como "herdados" .



Forma das Definições Dirigidas pela Sintaxe

Cada produção $A \rightarrow \alpha$ tem associada a si um conjunto de regras semânticas da forma: $b := f(c_1, c_2, \cdots, c_k)$, onde f é uma função e b pode ser:

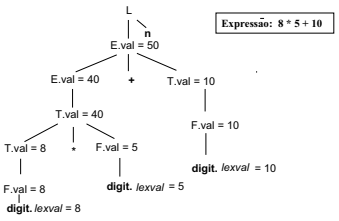
1. um atributo *sintetizado* de A e $(c_1, c_2, \cdots, c_k)$ são atributos pertencentes aos símbolos gramaticais da produção, OU
2. um atributo *herdado* pertencente a um dos símbolos gramaticais do lado direito da produção, e $(c_1, c_2, \cdots, c_k)$ são atributos pertencentes aos símbolos gramaticais da produção.

Em (1) ou (2), dizemos que o atributo b depende dos atributos $c_1, c_2, \cdots, c_k$.

Uma gramática de atributos é uma definição dirigida pela sintaxe, na qual as funções nas regras semânticas não têm efeitos colaterais.

Atributo Sintetizado

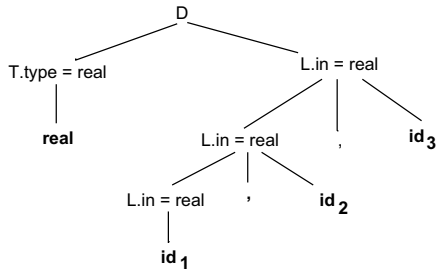
Produções	Rotinas Semânticas
L → E n	print(E.val)
E → E_1 + T	E.val := E_1.val + T.val
E → T	E.val := T.val
T → T_1 * F	T.val := T_1.val x F.val
T → F	T.val := F.val
F → (E)	F.val := E.val
F → digit	F.val := digit.lexval



Atributo Herdado

Produções	Rotinas Semânticas
D → TL	L.in := T.type
T → int	T.type := integer
T → real	T.type := real
L → L_1 , id	L_1.in := L.in
	addtype(id.entry, L.in)
L → id	*addtype*(id.entry, L.in)

- REAL id_1, id_2, id_3

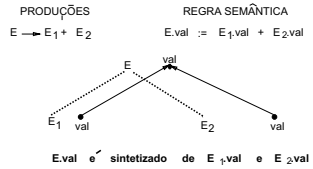


Grafo de Dependência

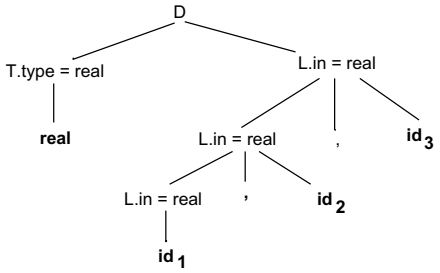
for cada vértice *n* na árvore do reconhecedor do
  for cada atributo *a* do símbolo da gramática em *n* do
    construa o vértice no grafo de dependência para *a*
for cada vértice *n* na árvore do reconhecedor do
  for cada regra semântica $b := f(c_1, c_2, \cdots, c_k)$ associada com a produção usada em *n* do
    for i := 1 to k do construa uma aresta a partir do vértice de c_1 para o vértice de *b*

Suponha **A.a := f(X.x, Y.y)** é uma regra semântica para $A \rightarrow XY$.
A.a := f(X.x, Y.y) define um ***atributo sintetizado*** **A.a** que depende dos atributos **X.x** e **Y.y**.

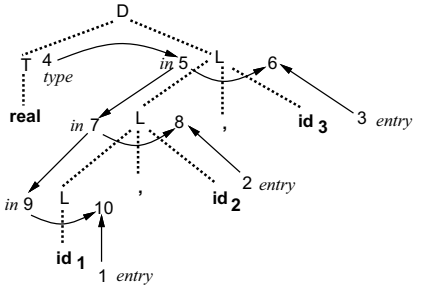
Se esta produção é usada na árvore de derivação, então haverá 3 nodos **A.a**, **X.x** e **Y.y** no grafo de dependência com uma aresta para **A.a** de **X.x** e **Y.y** uma vez que **A.a depende** de **X.x** e **Y.y**.



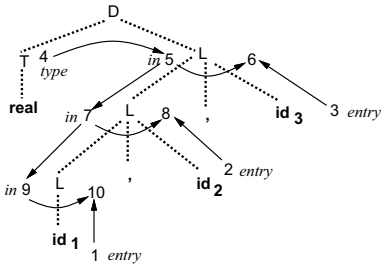
Ordenação Topológica



Vértices marcados por números



Ordem de Avaliação

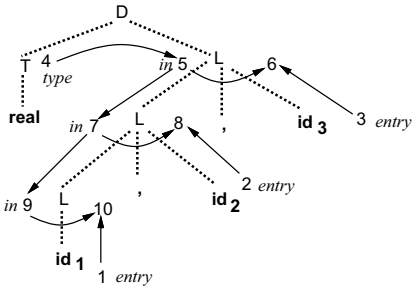


- Ordem: $m_1, m_2, \cdots, m_k$ se $m_i \rightarrow m_j$, **i < j na ordenação.**

Uma ordenação particular pode resultar na necessidade de criação da árvore de "parser" .

Na prática, procura-se uma ordenação de fácil implementação.

· · · Ordem de Avaliação



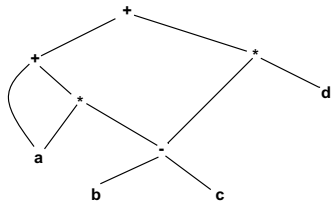
A partir da classificação topológica obtem-se o programa:

a_4 := real
$a_5 := a_4$
addtype (id_3.entry, a_5);
$a_7 := a_5$
addtype (id_2.entry, a_7);
$a_9 := a_7$
addtype (id_1.entry, a_9);

Resultado da avaliação:
armazena real **na TS para cada id.**

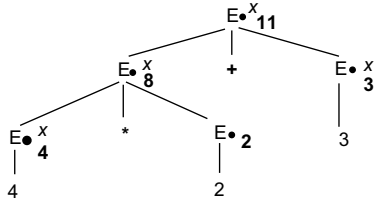
Avaliação BOTTOM-UP de Definição S-atribuida

- Definição S-atribuida: Definições dirigidas por sintaxe com somente atributos sintetizados.
- Pilha Semântica - Exemplo: $A \rightarrow XYZ$



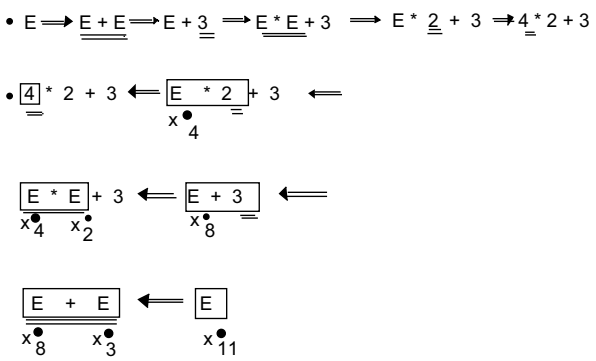
Ações semânticas colocadas nos estados de redução.
Exemplo: Implementação da Calculadora.

Exemplo: Avaliador de Expressões



Produções	Rotinas Semânticas
E $\rightarrow$ digito	{ E.x := valor(digito) }
E $\rightarrow E_1$ + E_2	{ E.x := E_1.x + E_2.x }
E $\rightarrow E_1$ * E_2	{ E.x := E_1.x * E_2.x }

Implementação (reconhecedor bottom-up)



Produções	Rotinas Semânticas
E $\rightarrow$ digito	{ E.x := valor(digito) }
E $\rightarrow E_1$ + E_2	{ E.x := E_1.x + E_2.x }
E $\rightarrow E_1$ * E_2	{ E.x := E_1.x * E_2.x }

Pilha de Atributos

Pilha antes da redução: E $\rightarrow$ E * E

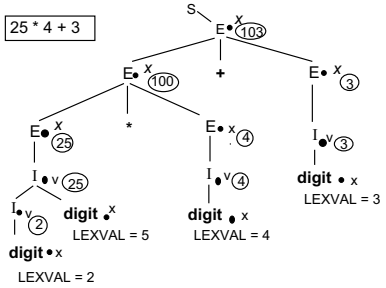
E_1.x $\equiv$ VAL[TOPO $-$ 2]
E_2.x $\equiv$ VAL[TOPO]
E.x $\equiv$ VAL[TOPO $-$ 2]

- (E $\rightarrow E_1$ * E_2) Então E.x := E_1.x * E_2.x significa:
VAL[TOPO $-$ 2] := VAL[TOPO $-$ 2] * VAL[TOPO]

Nota: Após a ação semântica a redução é efetuada.

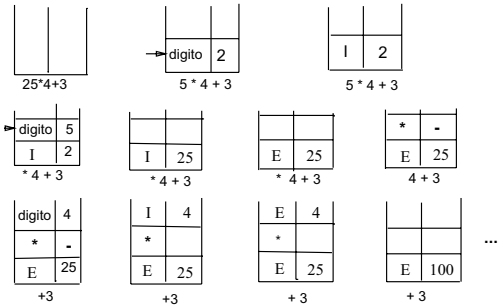
Exemplo

Produções	Rotinas Semânticas
S → E	print(E.x)
E → E_1 + E_2	{ E.x := E_1.x + E_2.x }
E → E_1 * E_2	{ E.x := E_1.x * E_2.x }
E → (E_1)	{ E.x := E_1.x }
E → I	{ E.x := I.v }
I → I digito	{I.v := I_1.v * 10 + digito.x}
I → digito	{ I.v := digito.x }



Implementação

Produções	Rotinas Semânticas
(1) S → E	{ print(VAL[TOPO]) }
(2) E → E + E	{ VAL[TOPO − 2] := VAL[TOPO − 2] + VAL[TOPO]}
(3) E → E * E	{ VAL[TOPO − 2] := VAL[TOPO − 2] * VAL[TOPO]}
(4) E → I	{ VAL[TOPO] := VAL[TOPO] } ≡ *nada*
(5) I → I digito	{VAL[TOPO−1]:=VAL[TOPO−1]*10+VAL[TOPO]}
(6) I → digito	{ VAL[TOPO] := VAL[TOPO] ≡ *nada*
(7) E → (E)	{ VAL[TOPO − 2] := VAL[TOPO − 1] }



Implementação

```
procedure sem(p);
 case  p
       S → E          1: print(VAL[TOPO])
       E → E + E      2: VAL[TOPO−2]:=VAL[TOPO−2]+VAL[TOPO]
       E → E * E      3: VAL[TOPO−2]:=VAL[TOPO−2]*VAL[TOPO]
       E → I          4: nada
       I → I digito   5:VAL[TOPO−1]:=VAL[TOPO−1]*10+VAL[TOPO]
       I → digito     6: nada
       E → (E)        7: VAL[TOPO − 2] := VAL[TOPO − 1]
 end
 procedure scan(tipo,valor);
 tipo          valor
 +             0
 ⋆             0
 )             0
 digito        valor do dígito
 end
```


Análise Sintática e Tradução

```
begin /* LR(1) */
     parsing :=  true; scan(tipo, valor)
     s := estado inicial;  push s;
      while parsing
         case ACTION[s, tipo]
            shift k: push k
                     VAL[TOPO]:= valor;  s:= k; scan(tipo,valor)
            acc: parsing :=  false
            error: parsing := recupera-do-erro
            reduce p:  sem(p); A:= LE da produção p
                       n:=comprimento LD de p; pop n
                       s := GOTO[ESTADO[TOPO],A]; push s
         end
  end  end
```

Definições L-Atribuidas

Uma definição dirigida por sintaxe (DDS) pode ser avaliada por um caminhamento "depth-first" da árvore de reconhecimento.

```
procedure dfvisit(n : node);
begin
      for "cada filho m de n, da esquerda para a direita"
            do begin
                  avalia atributos herdados de m;
                  dfvisit(m)
            end
            avalia atributos sintetizados de n
end
```


· · · Definições L-Atribuidas

Uma definição dirigida por sintaxe é L-atribuida quando atributos herdados de Xz, $1 \leq z \leq n$ do lado direito, em todas as produções do tipo $A \rightarrow X_1, X_2, \cdots, X_n$ dependem somente de:

(1) atributos de X_1 a X_{z-1};
(2) atributos herdados de A.

• **Esquema de Tradução:**
Gramáticas + ações semânticas especificando o processo de tradução.
Exemplo de uma definição dirigida por sintaxe não L-atribuida:

Produções	Rotinas Semânticas
A → L M	L.i := l(A.i)
	M.i := m(L.s)
	A.s := f(M.s)
A → Q R	R.i := r(A.i)
	Q.i := q(R.s)
	A.s := f(Q.s)

· · · Definições L-Atribuidas

- **Observe que:**
  1. **O atributo herdado de um símbolo do lado direito deve ser computado numa ação anterior ao símbolo.**
  2. **Uma ação não pode referenciar um atributo sintetizado de um símbolo a sua direita.**
  3. **Atributo sintetizado do não-terminal a esquerda de uma produção só pode ser computado depois que todos os atributos de que necessita o foram.**

Exemplo:
S → A_1 A_2 { A_1.in := 1; A_2.in := 2 }
A → a { print(A.in) }
A.in não definido na segunda produção.

· · · Definições L-Atribuidas

A.in na segunda produção não estará disponível quando for realizada uma tentativa de imprimir seu valor durante uma travessia em profundidade da árvore gramatical para a cadeia aa.

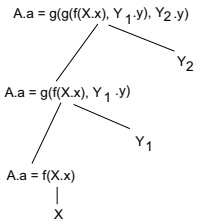
A travessia em profundidade começa por S e visita a subárvore de A_1 e A_2 antes que os valores de A_1.n e A_2.n sejam estabelecidos.

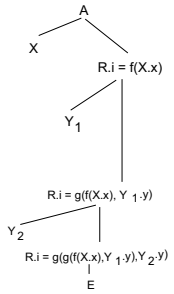
Se a ação definindo os valores de A_1.n e A_2.n estivesse inserida antes dos A's no lado direito de
S → A_1 A_2, ao invés de depois, então A.n estaria definida a cada vez em que imprimir (A.in) fosse executada.

Tradução TOP-DOWN

- **Eliminação de recursividade esquerda**

A → A_1 Y	{	A.a:=g(A_1.a,Y.y)}
A → X	{	A.a := f(X.x) }
	⇓	
A → X	{	R.i := f(X.x) }
R	{	A.a := R.s }
R → Y	{	R_1.i:=g(R.i,Y.y) }
R_1	{	R.s := R_1.s
R → $\mathcal{E}$	{	R.s := R.i }





· · · Tradução TOP-DOWN

- Definição Dirigida pela Sintaxe

E → E_1 + T	E.nptr := mknode ('+', E_1.nptr, T.nptr)
E → E_1 − T	E.nptr := mknode ('−', E_1.nptr, T.nptr)
E → T	E.nptr := T.nptr
T → (E)	T.nptr := E.nptr
T→ id	T.nptr := mkleaf (id, id.entry)
T→ num	T.nptr := mkleaf (num, num.val)

- Esquema de Tradução para E

E → E_1 + T	E.nptr := mknode ('+', E_1.nptr, T.nptr)
E → E_1 − T	E.nptr := mknode ('−', E_1.nptr, T.nptr)
E → T	E.nptr := T.nptr

· · · Tradução TOP-DOWN

E.T. para E após a retirada da Recursividade a Esquerda

E →	T	{ R.i := T.nptr }
	R	{ E.nptr := R.s }
R →	+	
	T	{ R_1.i := *mknode* ('+', R.i, T.nptr) }
	R_1	{ R.s := R_1.s }
R →	−	
	T	{ R_1.i := *mknode* ('−', R.i, T.nptr) }
	R_1	{ R.s := R_1.s }
R →	$\mathcal{E}$	{ R.s := R.i }
T →	(E)	{ T.nptr := E.nptr }
T →	id	{ T.nptr := *mkleaf* (id, id.*entry*) }
T→	num	T.nptr := *mkleaf* (num, num.*val*) }

Uma Implementação de ET para expressão

```
function E: ↑ syntax-tree-node;
function T: ↑ syntax-tree-node;
function R( in: ↑ syntax-tree-node):↑ syntax-tree-node);
var nptr, il, sl, s : ↑ syntax-tree-node; addoplexeme : char;
begin
        if lookahead = addop then begin
                /* produção R → addop T R */
                addoplexeme := lexval;
                match(addop);
                nptr := T; il := mknode(addoplexeme, in, nptr);
                sl := R(il); s := sl
        end else s:= in; /* produção R → ℰ*/
        return s
end;
```

SDDs L-Atribuídas e Análise LL

SDT para a geração direta de código por comandos $while$

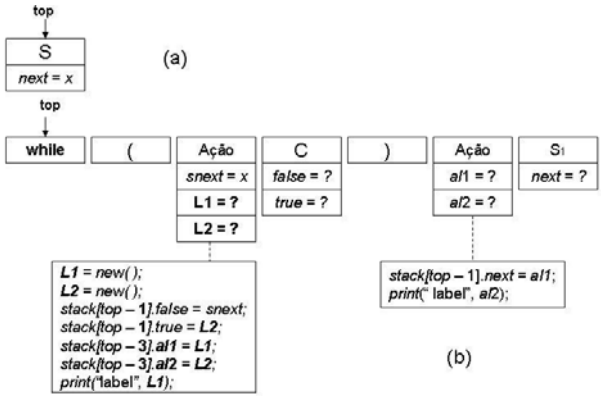
Este STD não possui atributos sintetizados, exceto para atributos fictícios que representam rótulos.

$$
\begin{array}{lll}
S \rightarrow \text{while} (& \{ \textbf{L1=new()}; L2 = new(); C.false = S.next; \\
 & C.true = L2; print(\text{"}label\text{"}, L1); \} \\
C) & \{S_1.next = L1; print(\text{"}label\text{"}, L2); \} \\
S_1 &
\end{array}
$$

SDDs L-Atribuídas e Análise LL

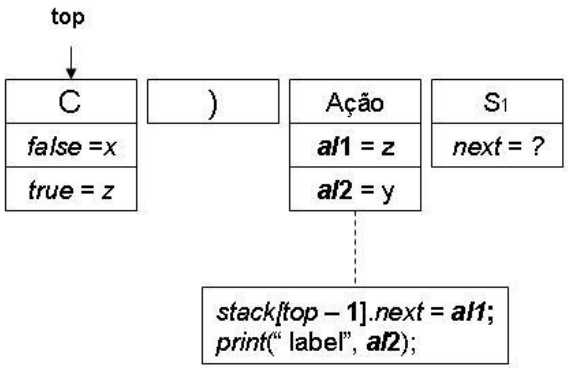
Implementação de atributos herdados durante a análise LL.

Expansão de S de acordo com a produção do comando $while$



SDDs L-Atribuídas e Análise LL

Depois que a ação de C é realizada



· · · SDDs L-Atribuídas e Análise LL

Expansão de S com atributo sintetizado construído na pilha

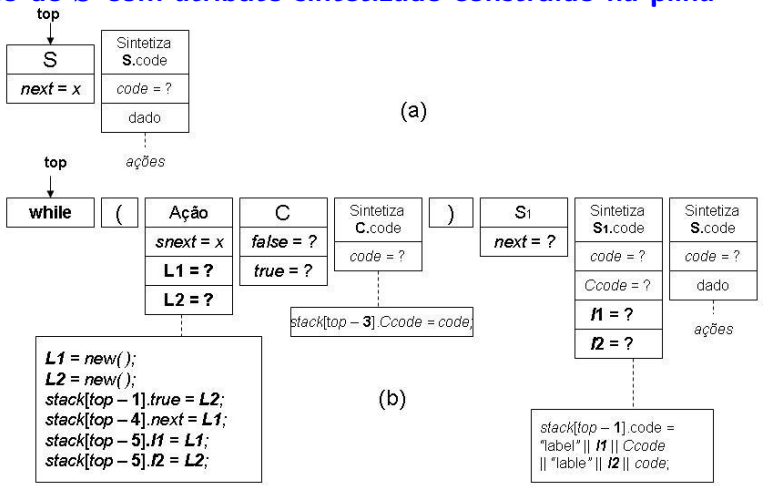
Neste exemplo faz-se uma tradução que produz a saída $S.code$ como atributo sintetizado, em vez de geração direta de código.

Invariante:

"Todo não-terminal que possui código associado a ele deixa esse código, como uma cadeia, no registro-de-sintetizados logo abaixo dele na pilha".

· · · SDDs L-Atribuídas e Análise LL

Expansão de S com atributo sintetizado construído na pilha



Análise Ascendente de SDDs L-atribuídas

É possível fazer uma tradução ascendente sempre que pudermos fazê-la descendente.

Dada uma SDD L-atribuída a uma gramática LL, é possível adaptar a gramática para computar a mesma SDD para a nova gramática durante uma análise LR.

Exemplo: suponha que exista uma produção $A \to B\ C$ em uma gramática LL, e que o atributo herdado $B.i$ seja calculado a partir do atributo herdado A.i por alguma fórmula $B.i = f(A.i)$. Ou seja, o fragmento de um SDT que nos interessa é

$$A \to \{B.i = f(A.i);\}\ B\ C$$

Introduzimos o marcador M com o atributo herdado $M.i$ e o atributo sintetizado $M.s$. O primeiro será uma cópia de $A.i$, e o segundo será $B.i$. O SDT será escrito como

$$\begin{aligned} A &\to M\ B\ C \\ M &\to \{M.i = A.i;\ M.s = f(M.i);\} \end{aligned}$$

· · · Análise Ascendente de SDDs L-atribuídas

Observação:

A regra para M não tem $A.i$ disponível para ela, mas de fato providenciaremos para que todo atributo herdado de um não-terminal como A apareça na pilha imediatamente abaixo de onde a redução para A ocorrerá mais tarde.

Assim, quando reduzimos ϵ para M, encontraremos $A.i$ imediatamente abaixo dela, de onde pode ser lida.

Além disso, o valor de $M.s$, que é deixado na pilha junto com M, é realmente $B.i$ e é encontrado corretamente logo abaixo de onde a redução para B ocorrerá mais tarde.

· · · Análise Ascendente de SDDs L-atribuídas

Vamos transformar o SDT

$$\begin{aligned} S \to \text{while (}\ &\{\ \textbf{L1=new();}\ L2 = new();\ C.false = S.next; \\ &\quad C.true = L2;\ \} \\ C\ )\quad &\{S_1.next = L1;\ \} \\ S_1\quad &\{\ S.code = \text{label}||L1||C.code||\text{label}||L2||S_1.code;\ \} \end{aligned}$$

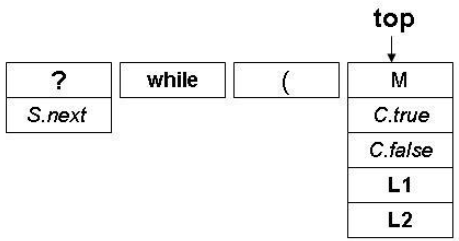
em um SDT que funcione com uma análise LR da gramática revisada.

Introduzimos um marcador M antes de C e um marcador N antes de S_1, de modo que a gramática subjacente se torna:

$$\begin{aligned} S &\to \text{while (}\ M\ C\ \text{)}\ N\ S_1 \\ M &\to \epsilon \\ N &\to \epsilon \end{aligned}$$

Análise Ascendente de SDDs L-atribuídas

Pilha sintática LR após a redução de ϵ para M

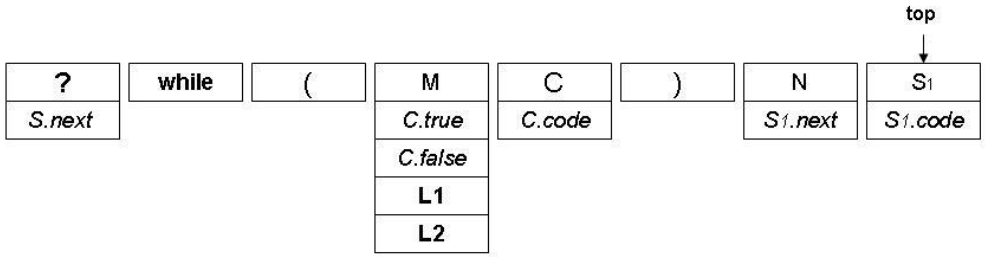


Código executado durante redução de ϵ para M

$$
\begin{aligned}
L1 &= new();\\
L2 &= new();\\
C.true &= L2;\\
C.false &= stack[top-3].next;
\end{aligned}
$$

Análise Ascendente de SDDs L-atribuídas

Pilha imediatamente antes da redução do corpo da produção while para S.

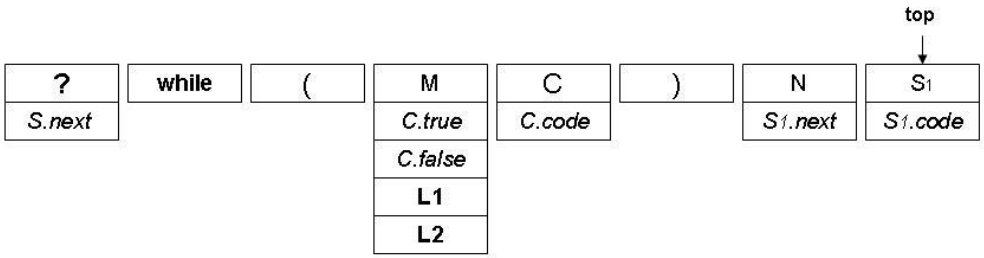


O código que é executado para computar o valor de $S_1.next$ é

$$S_1.next = stack[top-3].L1;$$

Análise Ascendente de SDDs L-atribuídas

O valor de $S_1.code$ é calculado e aparece no registro da pilha para S_1. Esse passo nos leva à condição ilustrada em



Nesse ponto, o analisador sintático reduzirá tudo desde o $while$ até S_1 para S.

Análise Ascendente de SDDs L-atribuídas

O código executado durante essa redução é:

$$
\begin{aligned}
&tempCode = \textbf{label} \parallel stack[top-4].L1 \parallel stack[top-3].code \parallel\\
&\quad \textbf{label} \parallel stack[top-4].L2 \parallel stack[top].code;\\
&top = top - 5;\\
&stack[top].code = tempCode;
\end{aligned}
$$

Observação: não mostramos, em nenhuma parte desta discussão, a manipulação dos estados LR, os quais também devem aparecer na pilha nos campos que preenchemos com símbolos da gramática.

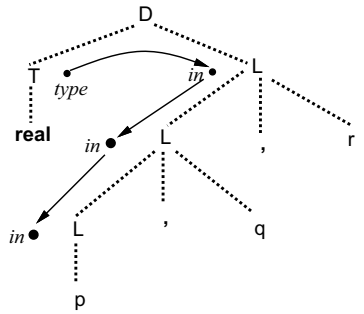
Avaliação BOTTOM-UP: Atributos Herdados

E → TR	E → TR
R → +T {print("+")} R	R → +T M R
| −T {print("−")} R ⇒	| −T N R
| $\mathcal{E}$	| $\mathcal{E}$
T → num {print(num.val)}	T → num {print(num.val) }
	M → $\mathcal{E}$ {print("+") }
	N → $\mathcal{E}$ {print("−") }

As gramáticas nos dois esquemas aceitam exatamente a mesma linguagem e, fazendo a árvore sintática com vértices extras para as ações, podemos mostrar que as ações são efetuadas na mesma ordem.

Note que: **ações no esquema de tradução depois de transformadas terminam produções, então elas podem ser efetuadas imediatamente antes que o lado direito seja reduzido durante a A.S. bottom-up.**

Atributos Herdados na Pilha do Reconhecedor



Produções		Rotinas Semânticas
D →	T L	L.in := T.type
T →	int	T.type := integer
T →	real	T.type := real
L →	L_1 , id	L_1.in := L.in addtype(id.entry, L.in)
L →	id	addtype(id.entry, L.in)

Reconhecimento de real p, q, r

entrada	estados	produções
real p,q,r	−	
p,q,r	real	
p,q,r	T	T→ real
,q,r	Tp	
,q,r	TL	L→ id
q,r	TL,	
,r	TL,q	
,r	TL	L→ L,id
r	TL,	
	TL,r	
	TL	L → L,id
	D	D → TL

Produções	Rotinas Semânticas
D→ TL	
T→ int	val[ntop]:= integer
T→ real	val[ntop]:= real
L→ L, id	addtype(val[top],val[top−3])
L→ id	addtype(val[top],val[top−1])

O valor de T.type é usado no lugar de L.in.

Avaliadores Recursivos - Considerações:

Dada uma árvore de reconhecimento, seus vértices devem ser visitados em qualquer ordem.
Na tradução especificada por definição não L-atribuida:

(1) Os filhos de um vértice para uma produção necessitam ser visitados da esquerda para a direita.
(2) Enquanto os filhos de um vértice para uma outra produção necessitam ser visitados da direita para a esquerda.

O exemplo a seguir ilustra o poder de uso de funções mutuamente recursivas para avaliar os atributos dos vértices na árvore de reconhecimento.

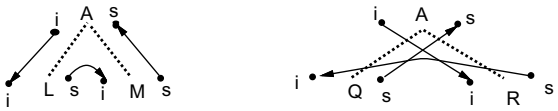
Funções não precisam depender da ordem em que os vértices da árvore foram criados.

Considerações mais importantes para avaliação durante caminhamento na árvore:

(1) atributos herdados em um vértice devem ser computados antes da primeira visita ao vértice;
(2) abributos sintetizados em um vértice devem ser computados antes de deixar o vértice pela última vez.

Avaliadores Recursivos: dada a def. Não L-atribuida:

Produções	Rotinas Semânticas
A → L M	L.i := l(A.i) M.i := m(L.s) A.s := f(M.s)
A → Q R	R.i := r(A.i) Q.i := q(R.s) A.s := f(Q.s)



i ≡ atributo herdado associado a cada não terminal.
s ≡ atributo sintetizado associado a cada não terminal.

Função de "A" da Definição Não L-atribuida

```
function A(n, ai);
begin
    case produção no vértice n of
    "A → LM" : /* da esquerda para a direita */
        li := l(ai); ls := L(child(n,l), li);
        mi := m(ls); ms := M(child(n, 2), mi); return f (ms);
    "A → QR" : /* da direita para a esquerda */
        ri := r(ai); rs := R(child(n,2), ri);
        qi := q(rs);
        qs := Q(child(n, 1), qi); return f (qs);
    default: error
    end end
```

• Note bem: Assumimos que funções para L, M, Q e R podem ser construidas. As variáveis li e ls correspondem a L.i e L.s.

Análise das Definições Dirigidas por Sintaxe

• **Problema:** Um identificador *overloaded* pode ter um conjunto de tipos, como conseqüencia, uma expressão, também, pode ter um conjunto de tipos. Informações sobre o contexto são usadas para selecionar um dos possíveis tipos para cada sub-expressão.

• **Possível solução:** Efetuar um passo *bottom-up* para sintetizar o conjunto dos possíveis tipos seguido de um passo *top-down* para tranformar o conjunto em um único tipo.

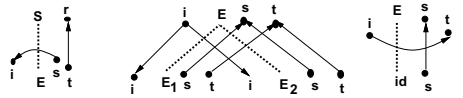
Nas regras semânticas a seguir, atributos sintetizados *s* representam o conjunto de tipos possíveis.

Atributos herdados *i* representam informações sobre o contexto.

O atributo sintetizado *t* representa o código gerado ou o tipo selecionado para uma sub-expressão, este atributo não pode ser avaliado no mesmo passo que *s*.

Análise das Definições Dirigidas por Sintaxe

Produções	Rotinas Semânticas
S → E	E.i := g(E.s) S.r := E.t A.s := f(M.s)
E → E_1 E_2	E.s := fs(E_1.s, E_2.s) E_1.i := fi1(E.i) E_2.i := fi2(E.i) E.t := ft(E_1.t, E_2.t)
E → id	E.s := id.s E.t := h(E.i)



FIM