

Construção de compiladores

Prof. Daniel Lucrédio

Departamento de Computação - UFSCar

1º semestre / 2015

Aula 7

Estrutura de um compilador

- Duas partes: análise e síntese

Quebrar o programa em partes
Impor uma estrutura gramatical
Criar uma representação intermediária
Detectar e reportar erros (sintáticos e semânticos)
Criar a tabela de símbolos

(front-end)

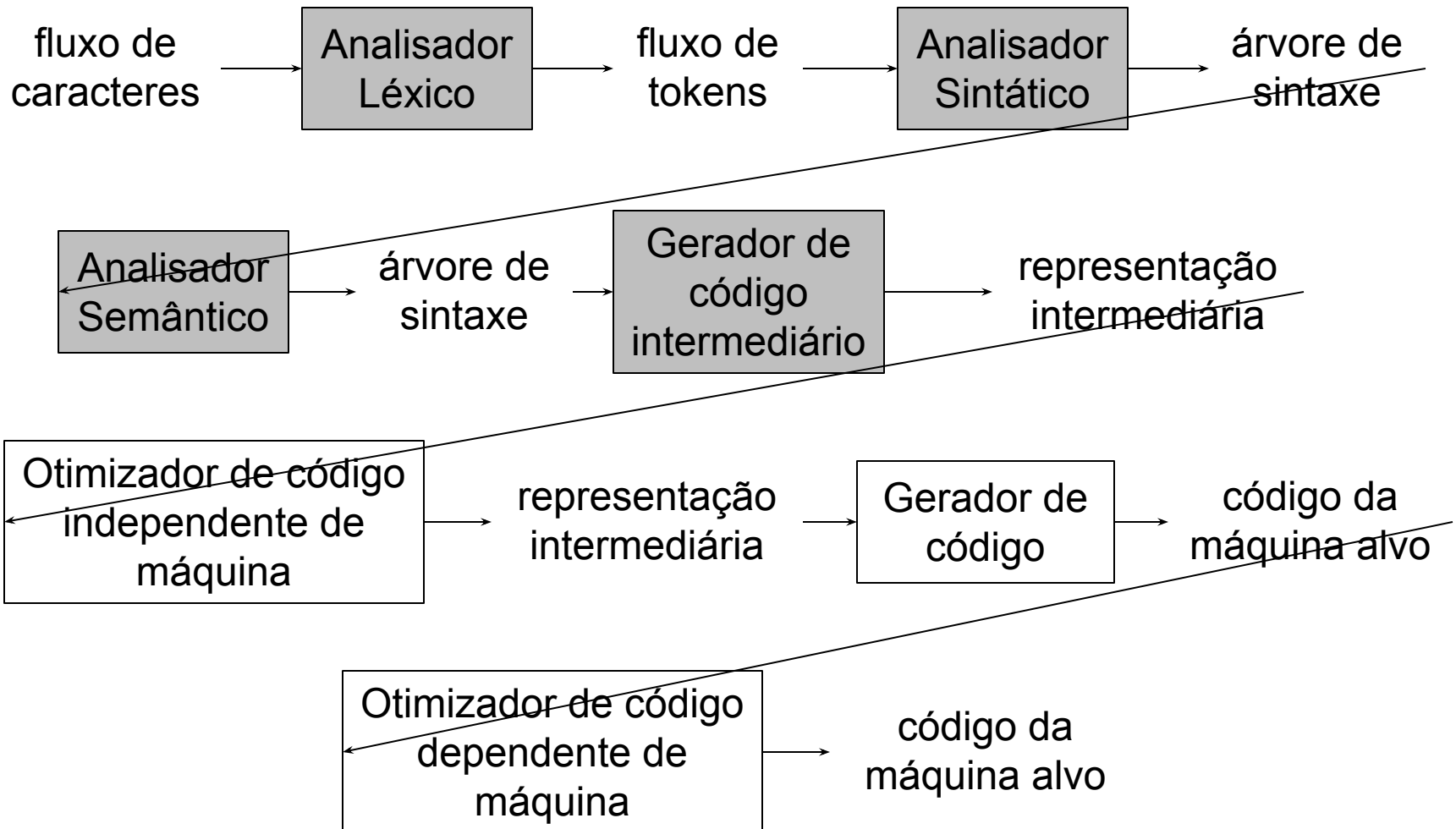
Construir o programa objeto
Com base na representação intermediária
E na tabela de símbolos

(back-end)

Fases de um compilador

front-end


back-end



Fases de um compilador

- Análise léxica (scanning)
 - Lê o fluxo de caracteres e os agrupa em sequências significativas
 - Chamadas **lexemas**
 - Para cada lexema, produz um token

<nome-token, valor-atributo>

- 
- Identifica o tipo do token
 - Símbolo abstrato, usado durante a análise sintática

- Aponta para a tabela de símbolos (quando o token tem valor)
- Necessária para análise semântica e geração de código

Fases de um compilador

- Análise sintática (parsing)
 - Usa os tokens produzidos pelo analisador léxico
 - Somente o primeiro “componente”
 - (ou seja, despreza os aspectos não-livres-de-contexto)
 - Produz uma árvore de análise sintática
 - Representa a estrutura gramatical do fluxo de tokens
 - As fases seguintes utilizam a estrutura gramatical para realizar outras análises e gerar o programa objeto

Fases de um compilador

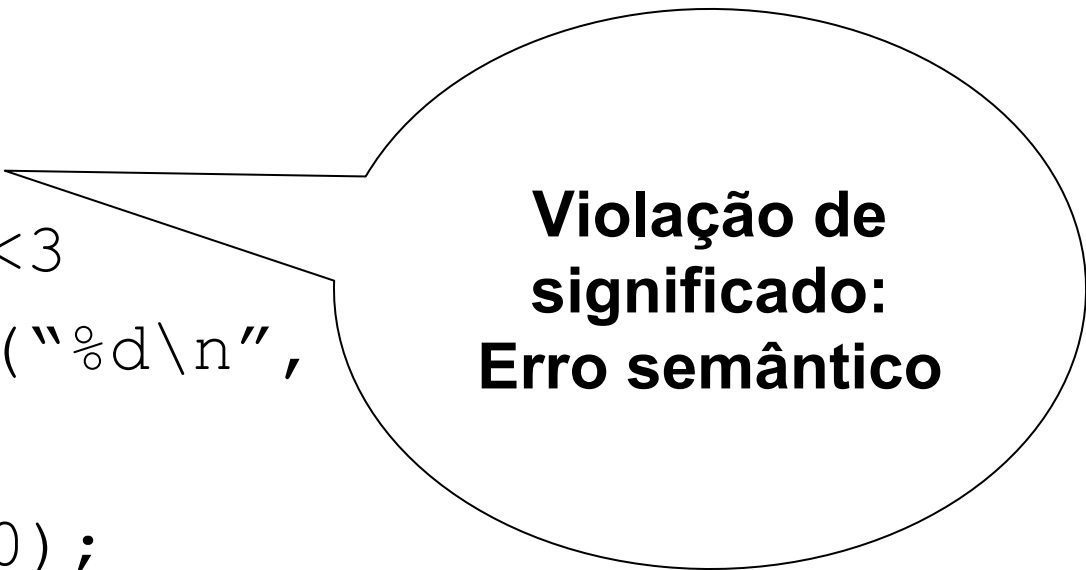
- Análise semântica
 - Checa a consistência com a definição da linguagem
 - Coleta informações sobre tipos e armazena na árvore de sintaxe ou na tabela de símbolos
 - Checagem de tipos / coerção (adequação dos tipos)
- É aqui que aparece a “sensibilidade ao contexto”

Manipulação de erros

```
int main()  
{  
    int i, a[1000000000000000];  
    float j@;  
  
    i = "1";  
    while (i<3  
        printf("%d\n", i);  
    k = i;  
    return (0);  
}
```

Manipulação de erros

```
int main()  
{  
    int i, a[1000000000000000];  
    float j@;  
  
    i = "1";  
    while (i<3  
        printf("%d\n",  
        k = i;  
    return (0);  
}
```



**Violação de
significado:
Erro semântico**

Manipulação de erros

```
int main()  
{  
    int i, a[1000000000000000];  
    float j@;  
  
    i = "1";  
    while (i<3  
        printf("%d\n"  
    k = i;  
    return (0);  
}
```

**Violação de
identificadores
conhecidos:
Erro contextual
("semântico")**

Antes

- Vamos fazer uma breve demonstração de um analisador semântico feito “à mão”
- Demonstração

Problemas

- E se eu precisar de outras análises semânticas?
 - Exs:
 - Detectar métodos com “return” faltando
 - Detectar código inalcançável
 - Considerar diferentes escopos
 - Etc...
- A implementação fica complicada
- Além disso
 - Normalmente utilizamos geradores de analisadores
 - Yacc / ANTLR
 - Não temos controle direto sobre os procedimentos
 - Normalmente trabalhamos com a gramática
 - Em analisadores bottom-up é ainda pior o acesso ao código!!

Análise semântica dirigida pela sintaxe

- Surge a necessidade de um formalismo
 - Que nos permite expressar a análise semântica de forma acoplada à sintaxe
 - Assim como a (E)BNF permite gerar código de análise sintática
 - Esse formalismo permitiria gerar código de análise semântica
- Porém, a análise semântica é muito diversificada
 - Temos que fazer coisas como:
 - Checar fluxo de controle em busca de código inalcançável
 - Calcular tipos de expressões (ex: $\frac{1}{2}$ = real)
 - Verificar se variáveis foram declaradas ou não, seu escopo, etc...
- Em geral a semântica de uma linguagem de programação não é formalmente especificada
 - O projetista do compilador tem que analisar e extrair a semântica

Análise semântica dirigida pela sintaxe

- Não existe um único modelo capaz de cobrir todos os casos
 - Assim, análise semântica é normalmente feita através de código comum
 - Ou seja, código que faz o que o projetista quiser
 - Porém, ainda assim é necessário algum controle
 - Considerando-se as principais ações semânticas
 - Principais tarefas feitas durante a análise semântica
- Formalismo: Semântica Dirigida pela Sintaxe
 - Definições Dirigidas pela Sintaxe (DDS)
 - Esquemas de Tradução Dirigida pela Sintaxe (TDS)
 - Caso especial: gramática de atributos

Semântica Dirigida pela Sintaxe

- Conteúdo semântico é inserido na gramática
 - De forma que o analisador sintático (normalmente gerado) irá conter ações “extras”
 - Essas ações farão as verificações semânticas
 - Checagem de tipos
 - Declaração de variáveis, etc
- Tem uma maneira “certa” de inserir este conteúdo
 - Dependendo das ações
 - É isso que estudaremos nesta parte da disciplina

Semântica Dirigida pela Sintaxe

Semântica Dirigida pela Sintaxe

- Existem duas formas (ou notações) distintas
 - Definição Dirigida pela Sintaxe

PRODUÇÃO	REGRA SEMÂNTICA
$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel "+"$ \hookrightarrow concatenação

- Esquema de Tradução Dirigida pela Sintaxe

$E \rightarrow E_1 + T$	$\{ \text{print } "+" \}$	$ T$
$T \rightarrow id$	$\{ \text{print } id.lexema \}$	

Semântica Dirigida pela Sintaxe

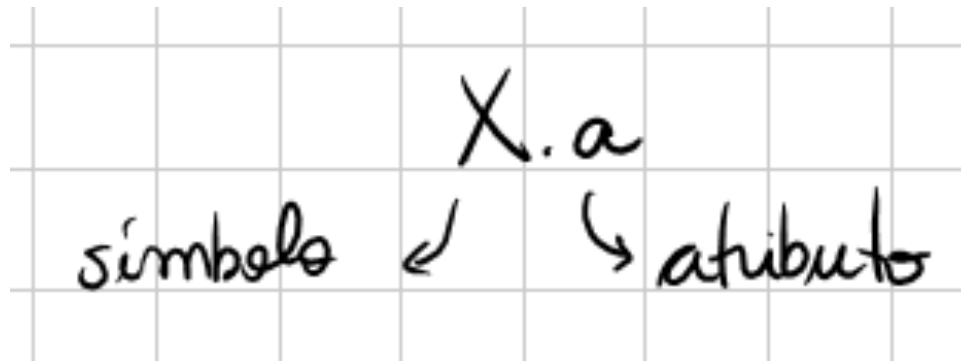
mais legíveis ← DEFINIÇÕES DIRIGIDAS PELA SINTAXE × ESQUEMAS DE TRADUÇÃO → mais eficientes

Definições Dirigidas pela Sintaxe

- Uma DDS =
 - Gramática + Atributos + Regras
- Atributos
 - Associados a símbolos gramaticais
 - Terminais e não-terminais
- Regras
 - Associadas às produções

Atributos

- Armazenam valores associados a um símbolo gramatical
 - Podem ser de diferentes tipos (veremos depois)



Regras semânticas

- Fazem diversas coisas
 - Calculam valores de atributos
 - Interagem com a tabela de símbolos
 - Imprimem valores (para geração de código, por exemplo)

PRODUÇÃO

$E \rightarrow T \times F$

$F \rightarrow (E)$

REGRA SEMÂNTICA

$E.val = T.val \times F.val$

$F.val = E.val$

Definições Dirigidas pela Sintaxe

- Se as regras semânticas APENAS realizam cálculo de atributos
 - Sem nenhum efeito colateral
 - Ex: interagir com a tabela de símbolos, imprimir valores
- A DDS é conhecida como
 - Gramática de atributos

Gramática de atributos

- Gramática de atributos
 - Princípio da semântica dirigida pela sintaxe
- Dada a coleção de atributos a_1, \dots, a_k
 - Para cada regra gramatical $X_0 \rightarrow X_1 X_2 \dots X_n$, onde X_0 é um não-terminal e os outros X_i são símbolos arbitrários
 - Os valores dos atributos $X_i.a_j$ de cada símbolo gramatical X_i são relacionados aos valores dos atributos dos outros símbolos na regra
- Se o mesmo símbolo X_i aparecer mais de uma vez na regra gramatical, cada ocorrência deve ser diferenciada

Gramática de atributos

- Exemplo
 - $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo} \mid \text{termo}$
 - $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$
 - $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} + \text{termo.val}$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} - \text{termo.val}$
$\text{exp} \rightarrow \text{termo}$	$\text{exp.val} = \text{termo.val}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{val} = \text{termo}_2.\text{val} * \text{fator.val}$
$\text{termo} \rightarrow \text{fator}$	$\text{termo.val} = \text{fator.val}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator.val} = \text{exp.val}$
$\text{fator} \rightarrow \text{num}$	$\text{fator.val} = \text{num.val}$

Gramática de atributos

- Exemplo
 - $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo} \mid \text{termo}$
 - $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$
 - $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

Regras gramaticais	Regras de atributos
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} + \text{termo.val}$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} - \text{termo.val}$
$\text{exp} \rightarrow \text{termo}$	$\text{exp.val} = \text{termo.val}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{val} = \text{termo}_2.\text{val} * \text{fator.val}$
$\text{termo} \rightarrow \text{fator}$	$\text{termo.val} = \text{fator.val}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator.val} = \text{exp.val}$
$\text{fator} \rightarrow \text{num}$	$\text{fator.val} = \text{num.val}$

Não aparece à esquerda. Vai ser computado antes (na análise léxica)

Gramática de

- Exemplo

- $\text{exp} \rightarrow \text{exp} + \text{termo}$
- $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{termo} / \text{fator}$
- $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

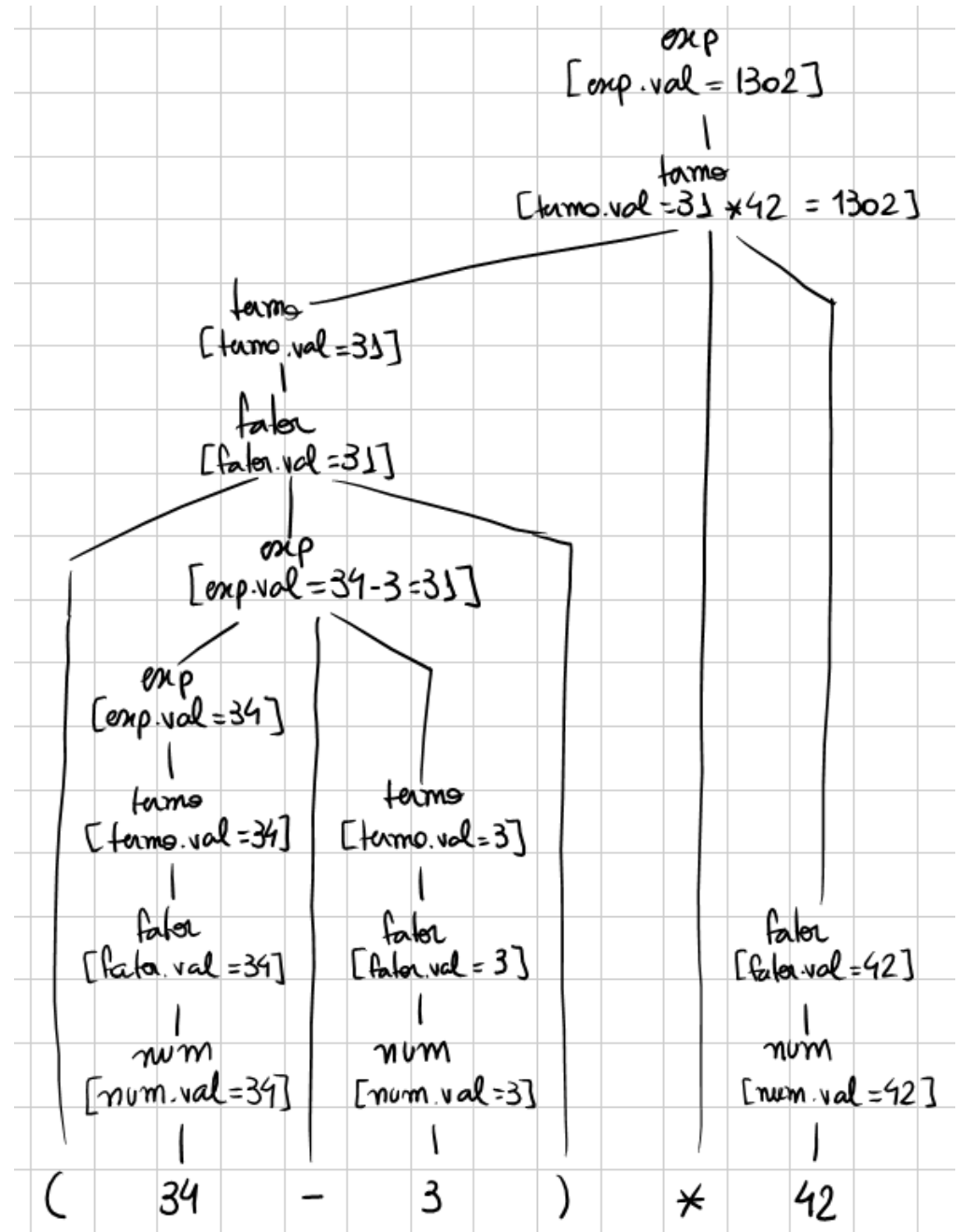
Como seria a árvore de análise sintática com o cálculo dos atributos, para a cadeia **(34-3)*42** ?

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} + \text{termo.val}$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} - \text{termo.val}$
$\text{exp} \rightarrow \text{termo}$	$\text{exp.val} = \text{termo.val}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{val} = \text{termo}_2.\text{val} * \text{fator.val}$
$\text{termo} \rightarrow \text{fator}$	$\text{termo.val} = \text{fator.val}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator.val} = \text{exp.val}$
$\text{fator} \rightarrow \text{num}$	$\text{fator.val} = \text{num.val}$

Gramática de atributos

$(34-3)*42$

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} + \text{termo.val}$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{val} = \text{exp}_2.\text{val} - \text{termo.val}$
$\text{exp} \rightarrow \text{termo}$	$\text{exp.val} = \text{termo.val}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{val} = \text{termo}_2.\text{val} * \text{fator.val}$
$\text{termo} \rightarrow \text{fator}$	$\text{termo.val} = \text{fator.val}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator.val} = \text{exp.val}$
$\text{fator} \rightarrow \text{num}$	$\text{fator.val} = \text{num.val}$



Gramática de atributos

- Na verdade, esse exemplo não é comum
 - Cálculo de expressões é normalmente feito em tempo de execução
 - Mas pode ser feito em tempo de compilação para constantes (produzindo código mais eficiente)
- Mas as regras semânticas podem ser usadas para diversas tarefas
 - Veremos outra a seguir

Gramática de atributos

- Outro exemplo
 - $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo} \mid \text{termo}$
 - $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$
 - $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(+, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(-, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp} \rightarrow \text{termo}$	$\text{exp}.\text{árvore} = \text{termo}.\text{árvore}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{árvore} = \text{mkOpNode}(*, \text{termo}_2.\text{árvore}, \text{fator}.\text{árvore})$
$\text{termo} \rightarrow \text{fator}$	$\text{termo}.\text{árvore} = \text{fator}.\text{árvore}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{árvore} = \text{exp}.\text{árvore}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{árvore} = \text{mkNumNode}(\text{num}.\text{lexval})$

Gramática de atributos

- Outro exemplo
 - $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo}$
 - $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$
 - $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

Aqui os atributos são (sub)árvores.

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(+, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(-, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp} \rightarrow \text{termo}$	$\text{exp}.\text{árvore} = \text{termo}.\text{árvore}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{árvore} = \text{mkOpNode}(*, \text{termo}_2.\text{árvore}, \text{fator}.\text{árvore})$
$\text{termo} \rightarrow \text{fator}$	$\text{termo}.\text{árvore} = \text{fator}.\text{árvore}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{árvore} = \text{exp}.\text{árvore}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{árvore} = \text{mkNumNode}(\text{num}.\text{lexval})$

As regras semânticas montam uma árvore de sintaxe abstrata!

mkOpNode
cria um nó
do tipo Op

1º
argumento é
o tipo do
operador

- Outro exemplo
- $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo} \mid \text{termo}$
- $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$
- $\text{fator} \rightarrow (\text{exp}) \mid \text{num}$

Regras gramaticais	Regras semânticas
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(+, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}(-, \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp} \rightarrow \text{termo}$	$\text{exp}.\text{árvore} = \text{termo}.\text{árvore}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{árvore} = \text{mkOpNode}(*, \text{termo}_2.\text{árvore}, \text{fator}.\text{árvore})$
$\text{termo} \rightarrow \text{fator}$	$\text{termo}.\text{árvore} = \text{fator}.\text{árvore}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{árvore} = \text{exp}.\text{árvore}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{árvore} = \text{mkNumNode}(\text{num})$

2º
argumento é
o primeiro
operando

3º
argumento é
o segundo
operando

Gramática de atributos

- Outro exemplo
 - $\text{exp} \rightarrow \text{exp} + \text{termo} \mid \text{exp} - \text{termo} \mid \text{termo}$
 - $\text{termo} \rightarrow \text{termo} * \text{fator} \mid \text{fator}$

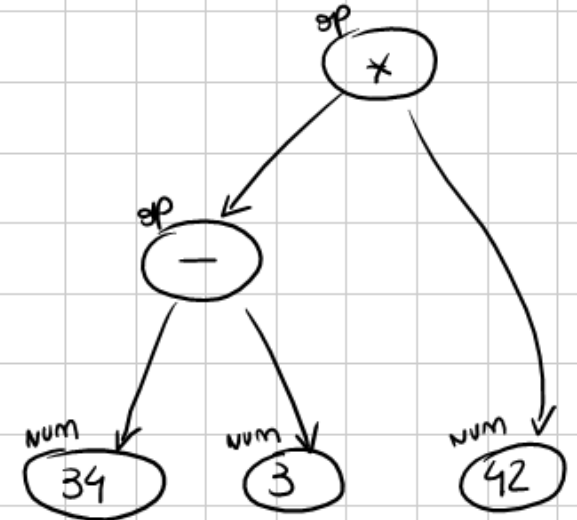
mkNumNode
cria um nó do
tipo Num

O único argumento
é o valor (léxico)
daquele número

Regras	
$\text{exp}_1 \rightarrow \text{exp}_2 + \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}('+', \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp}_1 \rightarrow \text{exp}_2 - \text{termo}$	$\text{exp}_1.\text{árvore} = \text{mkOpNode}('-', \text{exp}_2.\text{árvore}, \text{termo}.\text{árvore})$
$\text{exp} \rightarrow \text{termo}$	$\text{exp}.\text{árvore} = \text{termo}.\text{árvore}$
$\text{termo}_1 \rightarrow \text{termo}_2 * \text{fator}$	$\text{termo}_1.\text{árvore} = \text{mkOpNode}('*', \text{termo}_2.\text{árvore}, \text{fator}.\text{árvore})$
$\text{termo} \rightarrow \text{fator}$	$\text{termo}.\text{árvore} = \text{fator}.\text{árvore}$
$\text{fator} \rightarrow (\text{exp})$	$\text{fator}.\text{árvore} = \text{exp}.\text{árvore}$
$\text{fator} \rightarrow \text{num}$	$\text{fator}.\text{árvore} = \text{mkNumNode}(\text{num}.\text{lexval})$

Gramática de atributos

(34-3)*42



Regras gramaticais	Regras semânticas
$exp_1 \rightarrow exp_2 + termo$	$exp_1.\acute{a}rvore = mkOpNode(+, exp_2.\acute{a}rvore, termo.\acute{a}rvore)$
$exp_1 \rightarrow exp_2 - termo$	$exp_1.\acute{a}rvore = mkOpNode(-, exp_2.\acute{a}rvore, termo.\acute{a}rvore)$
$exp \rightarrow termo$	$exp.\acute{a}rvore = termo.\acute{a}rvore$
$termo_1 \rightarrow termo_2 * fator$	$termo_1.\acute{a}rvore = mkOpNode(*, termo_2.\acute{a}rvore, fator.\acute{a}rvore)$
$termo \rightarrow fator$	$termo.\acute{a}rvore = fator.\acute{a}rvore$
$fator \rightarrow (exp)$	$fator.\acute{a}rvore = exp.\acute{a}rvore$
$fator \rightarrow num$	$fator.\acute{a}rvore = mkNumNode(num.lexval)$

Gramática de atributos

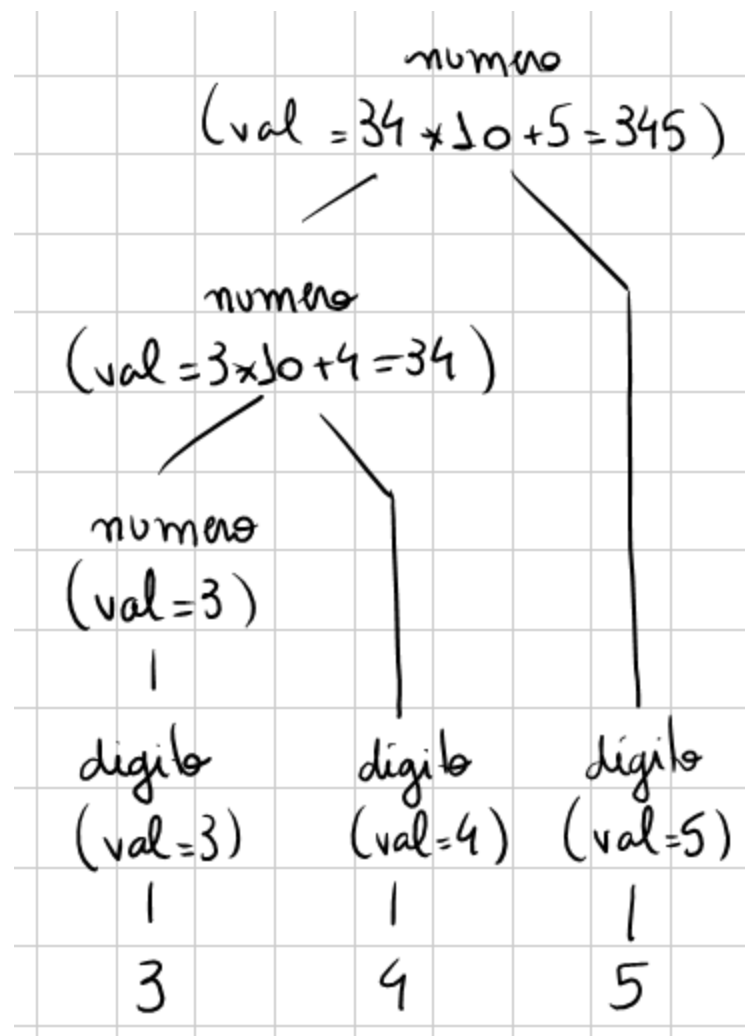
- Exercício
 - Dada a gramática a seguir para números sem sinal
 - Escreva a gramática de atributos correspondente
 - $\text{número} \rightarrow \text{número} \text{ dígito} \mid \text{dígito}$
 - $\text{dígito} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 - Obs: o objetivo é calcular o valor de um número

Regras gramaticais	Regras semânticas
$\text{número}_1 \rightarrow \text{número}_2 \text{ dígito}$	$\text{número}_1.\text{val} = \text{número}_2.\text{val} * 10 + \text{dígito}.\text{val}$
$\text{número} \rightarrow \text{dígito}$	$\text{número}.\text{val} = \text{dígito}.\text{val}$
$\text{dígito} \rightarrow 0$	$\text{dígito}.\text{val} = 0$
$\text{dígito} \rightarrow 1$	$\text{dígito}.\text{val} = 1$
...	...
$\text{dígito} \rightarrow 9$	$\text{dígito}.\text{val} = 9$

Gramática de atributos

- Exercício
 - Faça a análise semântica para a cadeia 345
 - Mostre os valores sendo calculados na árvore de análise sintática

Regras gramaticais	Regras semânticas
número ₁ → número ₂ dígito	número ₁ .val = número ₂ .val * 10 + dígito.val
número → dígito	número.val = dígito.val
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
...	...
dígito → 9	dígito.val = 9



Gramática de atributos

- Nem todo símbolo gramatical tem atributos
- Pode haver manipulação de mais de um atributo em uma mesma regra e para um mesmo símbolo
- Diferentes tipos de atributo
 - Atributo sintetizado
 - Atributo herdado
- Veremos com um exemplo

Gramática de atributos

- Exercício
 - Dada a gramática para números binários ou decimais, indicados pelos sufixos b ou d, respectivamente
 - Escreva a gramática de atributos que dê o valor decimal correspondente
- DICA – use 2 atributos: valor (val) e base
 - número \rightarrow num sufixo
 - sufixo \rightarrow b | d
 - num \rightarrow num dígito | dígito
 - dígito \rightarrow 0|1|2|3|4|5|6|7|8|9

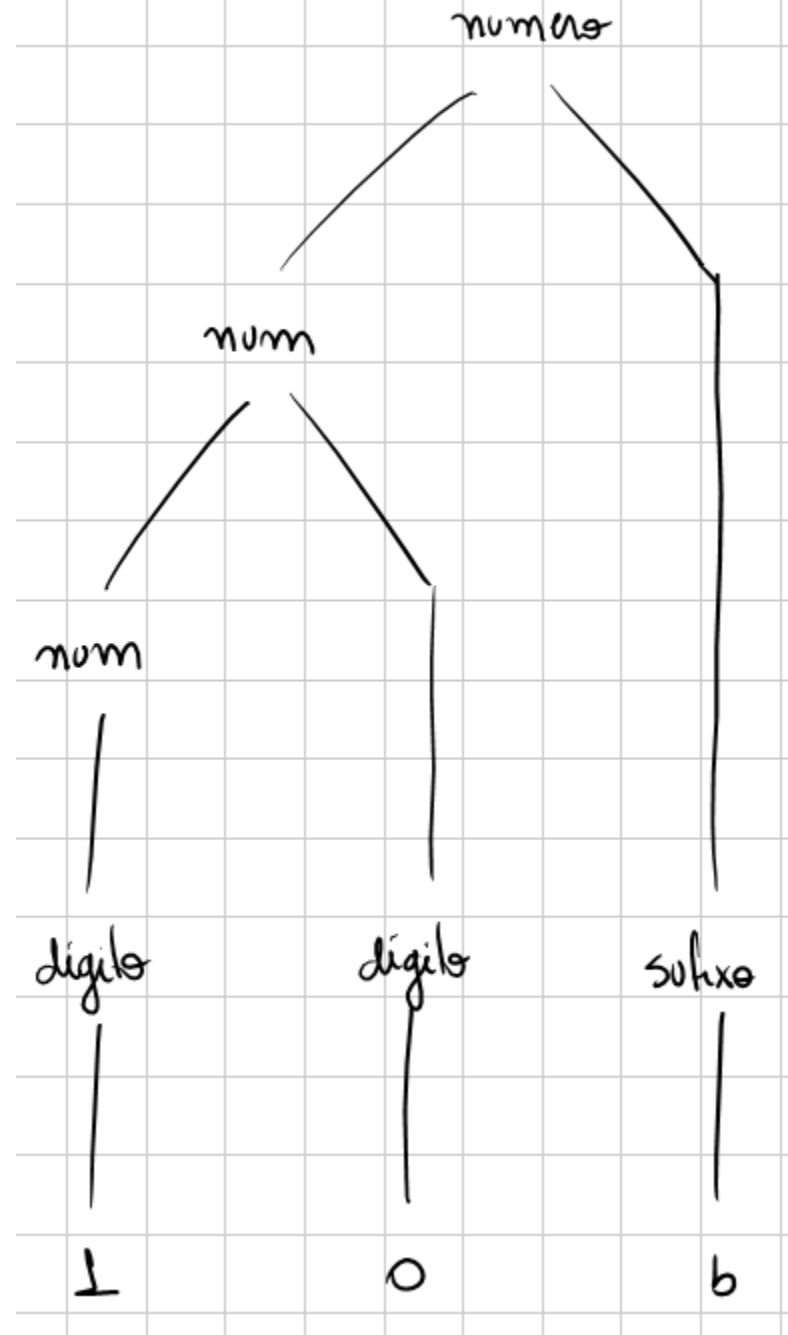
Resposta

Regras gramaticais	Regras semânticas
número \rightarrow num sufixo	número.val = num.val num.base = sufixo.base
sufixo \rightarrow b	sufixo.base = 2
sufixo \rightarrow d	sufixo.base = 10
num ₁ \rightarrow num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num \rightarrow dígito	num.val = dígito.val dígito.base = num.base
dígito \rightarrow 0	dígito.val = 0
dígito \rightarrow 1	dígito.val = 1
dígito \rightarrow 2	dígito.val = if dígito.base=2 then erro else 2
...	...

Gramática de atributos

10b

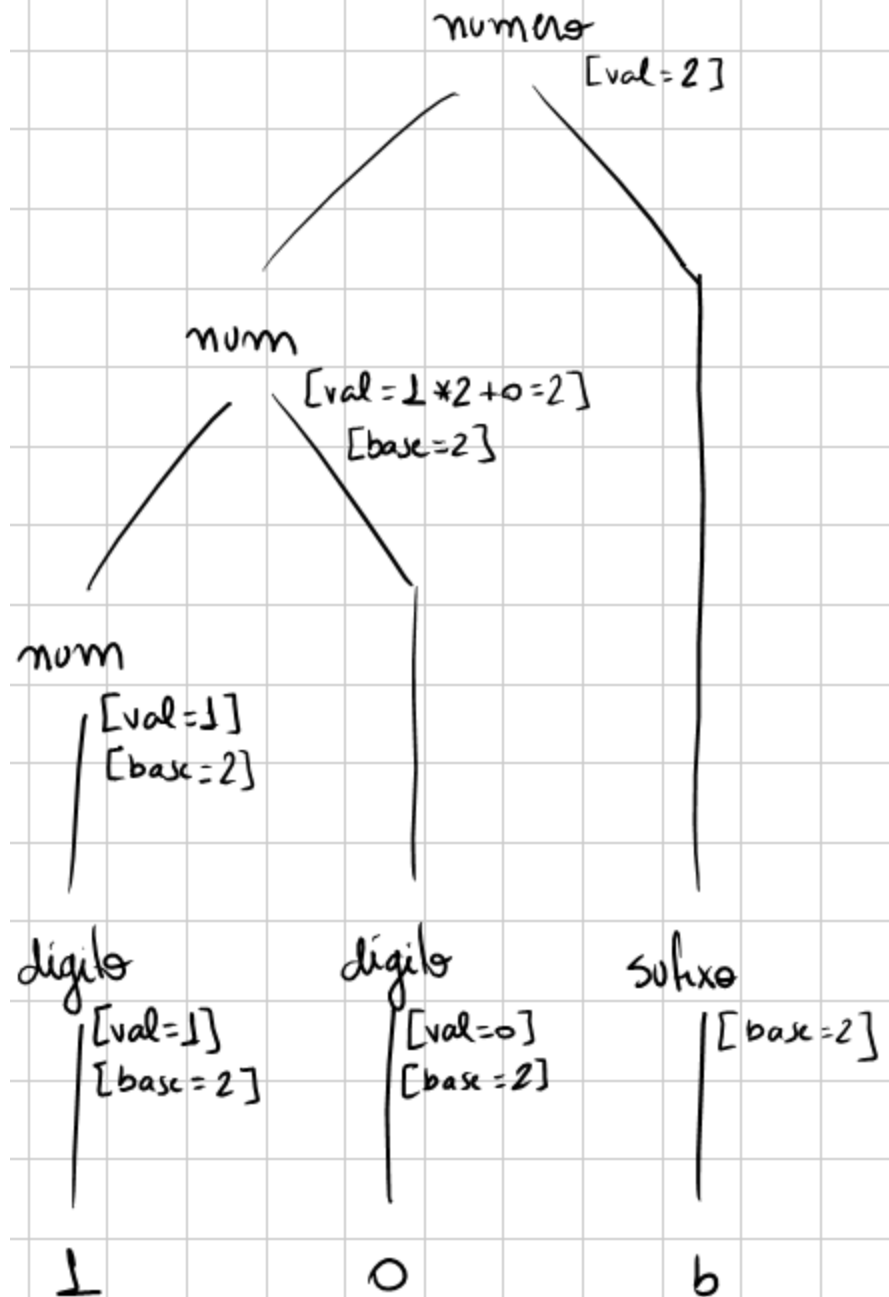
Regras gramaticais	Regras semânticas
número \rightarrow num sufixo	número.val = num.val num.base = sufixo.base
sufixo \rightarrow b	sufixo.base = 2
sufixo \rightarrow d	sufixo.base = 10
num ₁ \rightarrow num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num \rightarrow dígito	num.val = dígito.val dígito.base = num.base
dígito \rightarrow 0	dígito.val = 0
dígito \rightarrow 1	dígito.val = 1
dígito \rightarrow 2	dígito.val = if dígito.base=2 then erro else 2
...	...



Gramática de atributos

10b

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...

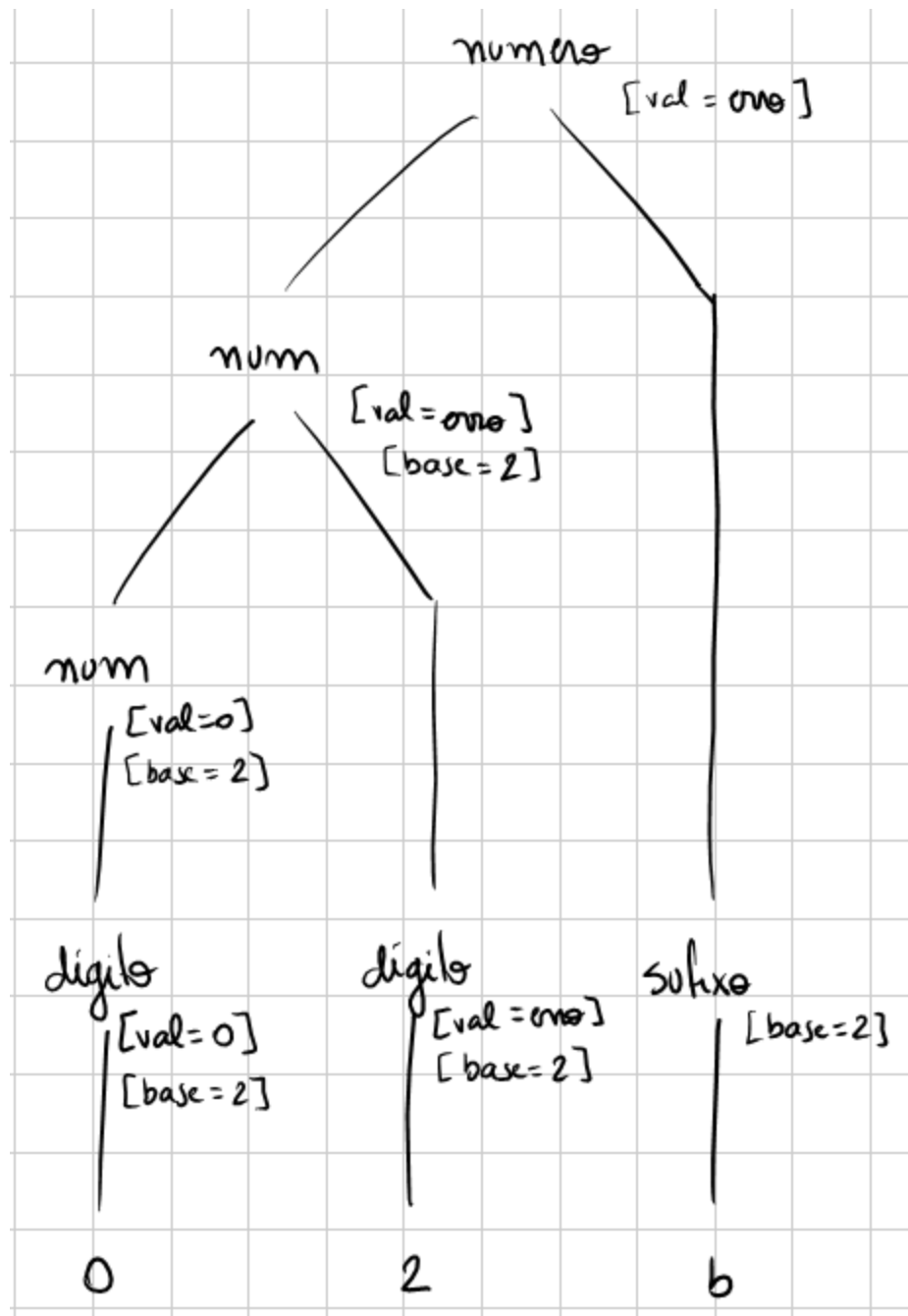


Gramática de atributos

A sintaxe
permite essa
cadeia, mas a
semântica não!

02b

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...

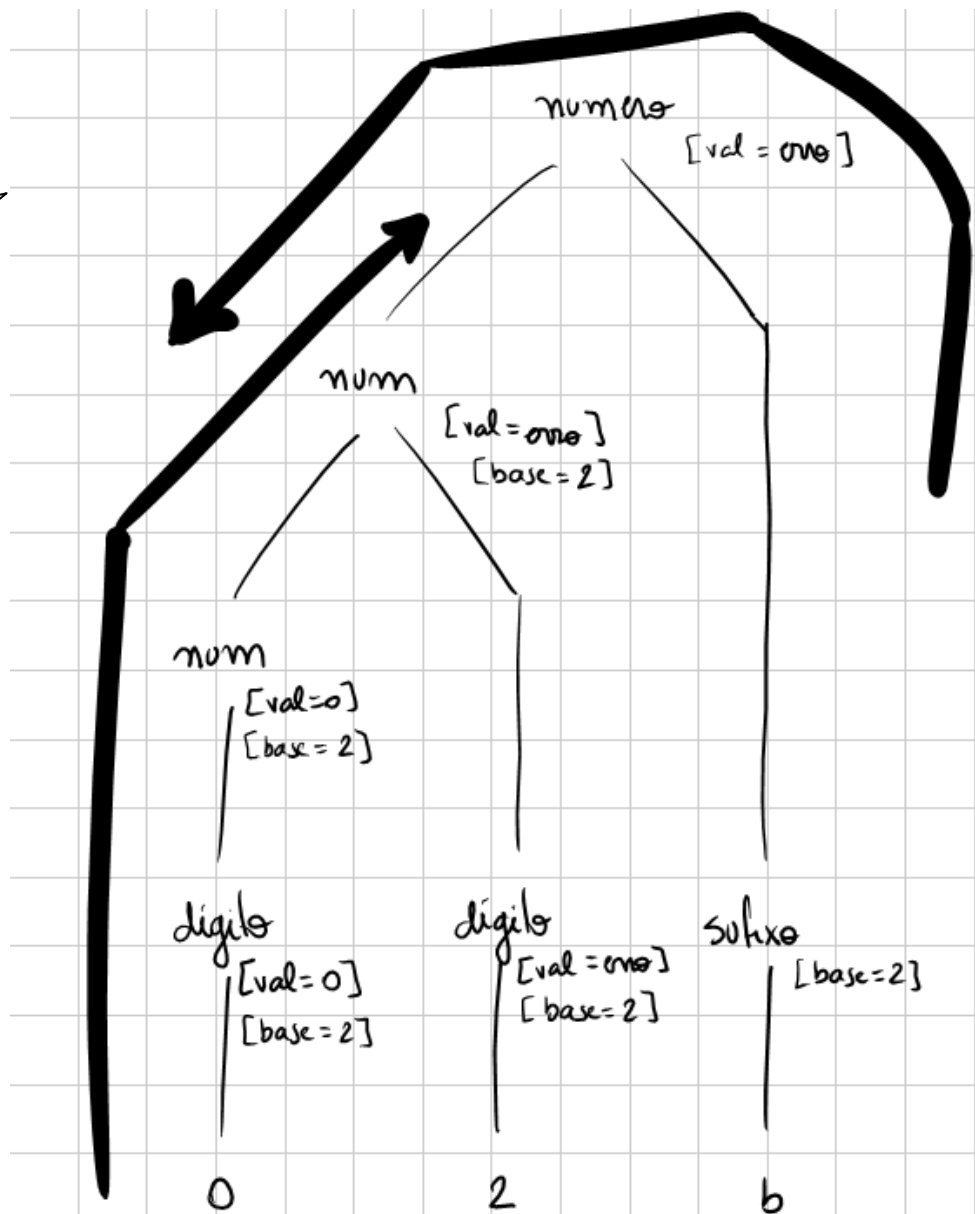


Gramática de atributos

Alguns valores sobem (val)
Outros valores descem (base)

02b

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...



Gramática de atributos

- Atributo sintetizado
 - Atributo que “sobe”
 - Para um não-terminal A , em um nó N da árvore de análise sintática, é definido por uma regra semântica associada com a produção em N (A deve ser a cabeça dessa produção)
 - O atributo é definido somente com base nos atributos de N e seus filhos

Gramática de atributos

- Atributos sintetizados





Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...

Gramática de atributos

- Atributo herdado
 - Atributo que “desce”
 - Para um não-terminal B, em um nó N da árvore de análise sintática, é definido por uma regra semântica associada com a produção no pai de N (B deve estar no corpo dessa produção)
 - O atributo é definido somente com base nos atributos do pai de N, N, e os irmãos de N

Gramática de atributos

- Atributos herdados

Regras gramaticais	Regras semânticas
número \rightarrow num sufixo	número.val = num.val num.base = sufixo.base 
sufixo \rightarrow b	sufixo.base = 2
sufixo \rightarrow d	sufixo.base = 10
num ₁ \rightarrow num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base  dígito.base = num ₁ .base 
num \rightarrow dígito	num.val = dígito.val dígito.base = num.base 
dígito \rightarrow 0	dígito.val = 0
dígito \rightarrow 1	dígito.val = 1
dígito \rightarrow 2	dígito.val = if dígito.base=2 then erro else 2
...	...

Gramática de atributos

- Terminais podem ter atributos sintetizados
 - Mas nunca herdados
 - Valores são sempre atribuídos pelo analisador léxico
 - E nunca por regras semânticas
- Uma gramática de atributos (ou uma DDS) que só possui atributos sintetizados é chamada S-atribuída

Ordem de avaliação

- Em que ordem avaliar atributos?
- Se uma gramática é S-atribuída
 - Qualquer ordem bottom-up resolve!
 - Mas nem sempre é possível criar gramáticas S-atribuídas
 - Em particular: sempre que a estrutura de uma árvore de análise sintática não corresponde à estrutura da árvore de sintaxe abstrata
 - Serão necessários atributos herdados!

Ordem de avaliação

- Exemplo de gramática S-atribuída:

Produções	Regras semânticas
$E_1 \rightarrow E_2 + E_3$	$E_1.val = E_2.val + E_3.val$
$E_1 \rightarrow E_2 - E_3$	$E_1.val = E_2.val - E_3.val$
$E_1 \rightarrow E_2 * E_3$	$E_1.val = E_2.val * E_3.val$
$E \rightarrow F$	$E.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \text{num}$	$F.val = \text{num.val}$

Ordem de avaliação

- Mas uma gramática equivalente não-ambígua:

Produções	Regras semânticas
$E \rightarrow T E'$	$E'.her = T.val$ $E.val = E'.sint$
$E' \rightarrow + T E'_1$	$E'_1.her = T.val + E'.her$ $E'.sint = E'_1.sint$
$E' \rightarrow - T E'_1$	$E'_1.her = E'.her - T.val$ $E'.sint = E'_1.sint$
$E' \rightarrow \varepsilon$	$E'.sint = E'.her$
$T \rightarrow F T'$	$T'.her = F.val$ $T.val = T'.sint$
$T' \rightarrow * F T'_1$	$T'_1.her = F.val * T'.her$ $T'.sint = T'_1.sint$
$T' \rightarrow \varepsilon$	$T'.sint = T'.her$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow num$	$F.val = num.lexval$

Ordem de avaliação

- Neste último exemplo, é impossível calcular os valores somente com atributos sintetizados
 - Motivo é a diferença entre a sintaxe abstrata e a sintaxe concreta
- Resumindo
 - Algumas gramáticas serão fatalmente não-S-atribuídas
- Problema:
 - Nestes casos, não existe uma única ordem fixa!
 - É necessário analisar caso a caso para determinar a ordem

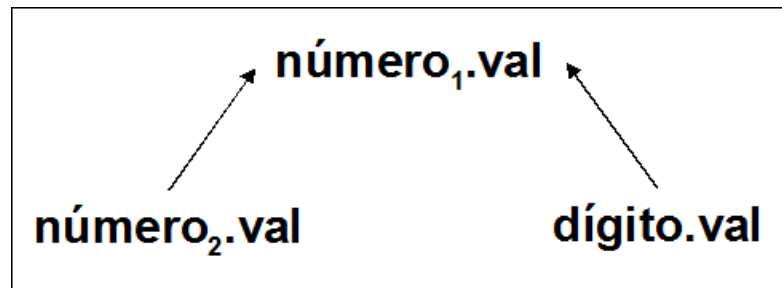
Grafos de dependência

- Especificam a ordem de cálculo dos atributos de cada regra gramatical em uma árvore sintática
 - Indicam as dependências entre atributos
 - Um grafo associado a cada regra gramatical
 - Cada grafo tem um nó rotulado para cada atributo $X_i.a_j$ de cada símbolo na regra gramatical e para cada equação
 - $X_i.a_j = f(\dots, X_m.a_k, \dots)$
 - existe um arco direcionado partindo de cada nó $X_m.a_k$ à direita para o nó $X_i.a_j$
- Para uma cadeia da linguagem, tem-se um grafo composto por todos os subgrafos

Grafos de dependência

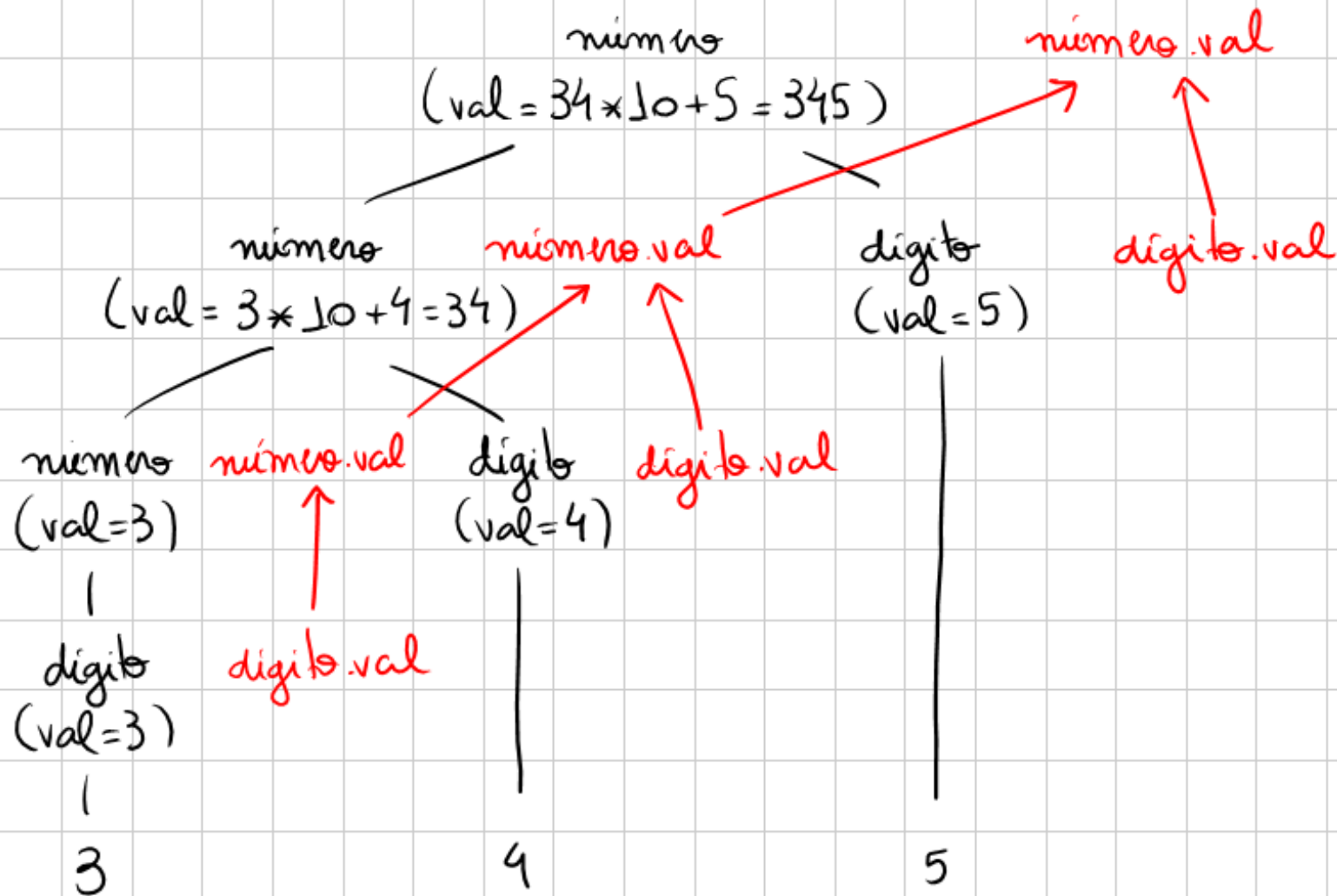
- Exemplo
 - número \rightarrow número dígito | dígito
 - dígito \rightarrow 0|1|2|3|4|5|6|7|8|9

Regras gramaticais	Regras semânticas
número ₁ \rightarrow número ₂ dígito	número ₁ .val = número ₂ .val * 10 + dígito.val
número \rightarrow dígito	número.val = dígito.val
dígito \rightarrow 0	dígito.val = 0
dígito \rightarrow 1	dígito.val = 1
...	...
dígito \rightarrow 9	dígito.val = 9



Grafos de dependência

- Ex: cadeia 345

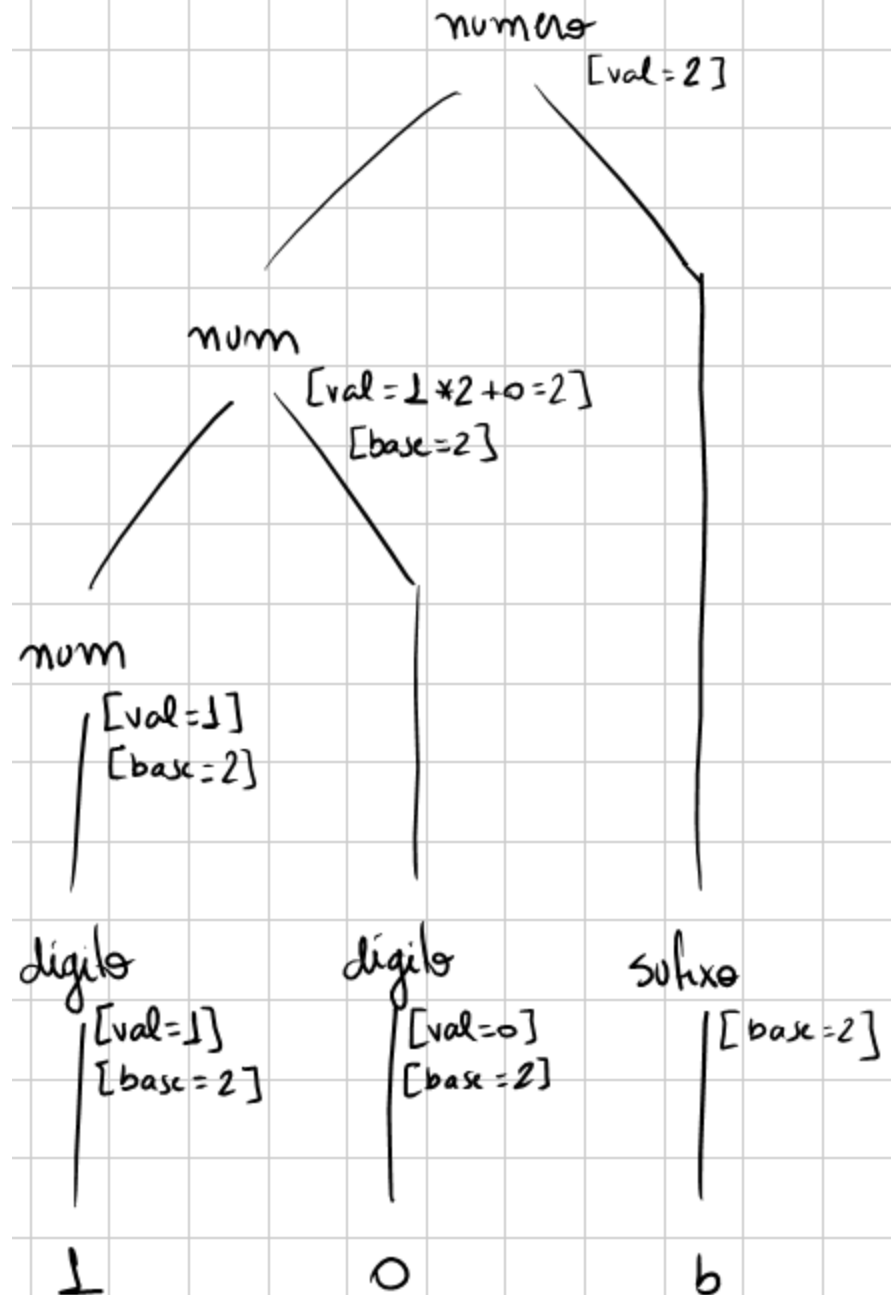


Grafos de dependência

Exercício: construa o grafo de dependência para a cadeia

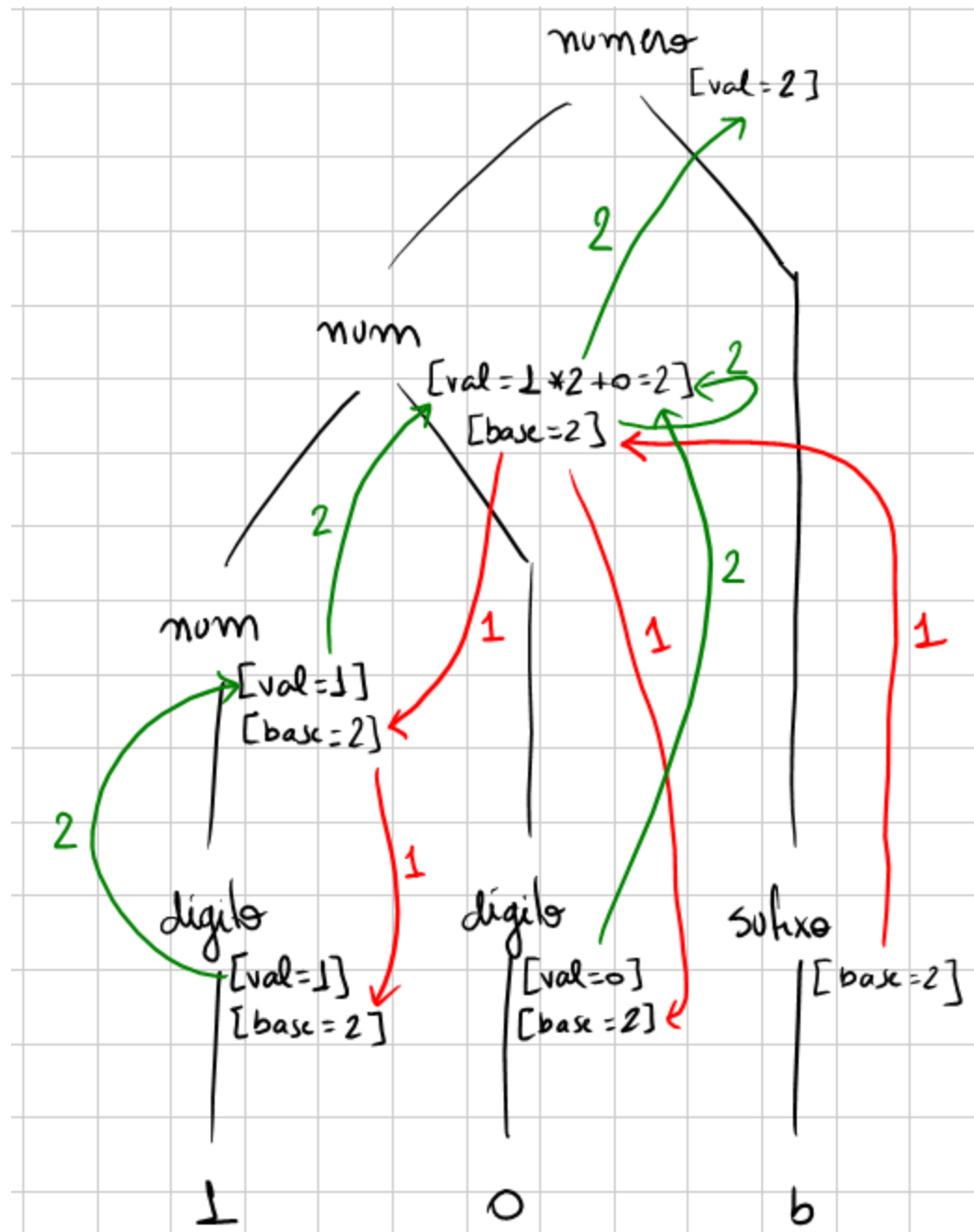
10b

Regras gramaticais	Regras semânticas
número \rightarrow num sufixo	número.val = num.val num.base = sufixo.base
sufixo \rightarrow b	sufixo.base = 2
sufixo \rightarrow d	sufixo.base = 10
num ₁ \rightarrow num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num \rightarrow dígito	num.val = dígito.val dígito.base = num.base
dígito \rightarrow 0	dígito.val = 0
dígito \rightarrow 1	dígito.val = 1
dígito \rightarrow 2	dígito.val = if dígito.base=2 then erro else 2
...	...



Grafos de dependência

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val 2 num.base = sufixo.base 1
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val 2 num ₂ .base = num ₁ .base 1 dígito.base = num ₁ .base 1
num → dígito	num.val = dígito.val 2 dígito.base = num.base 1
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...

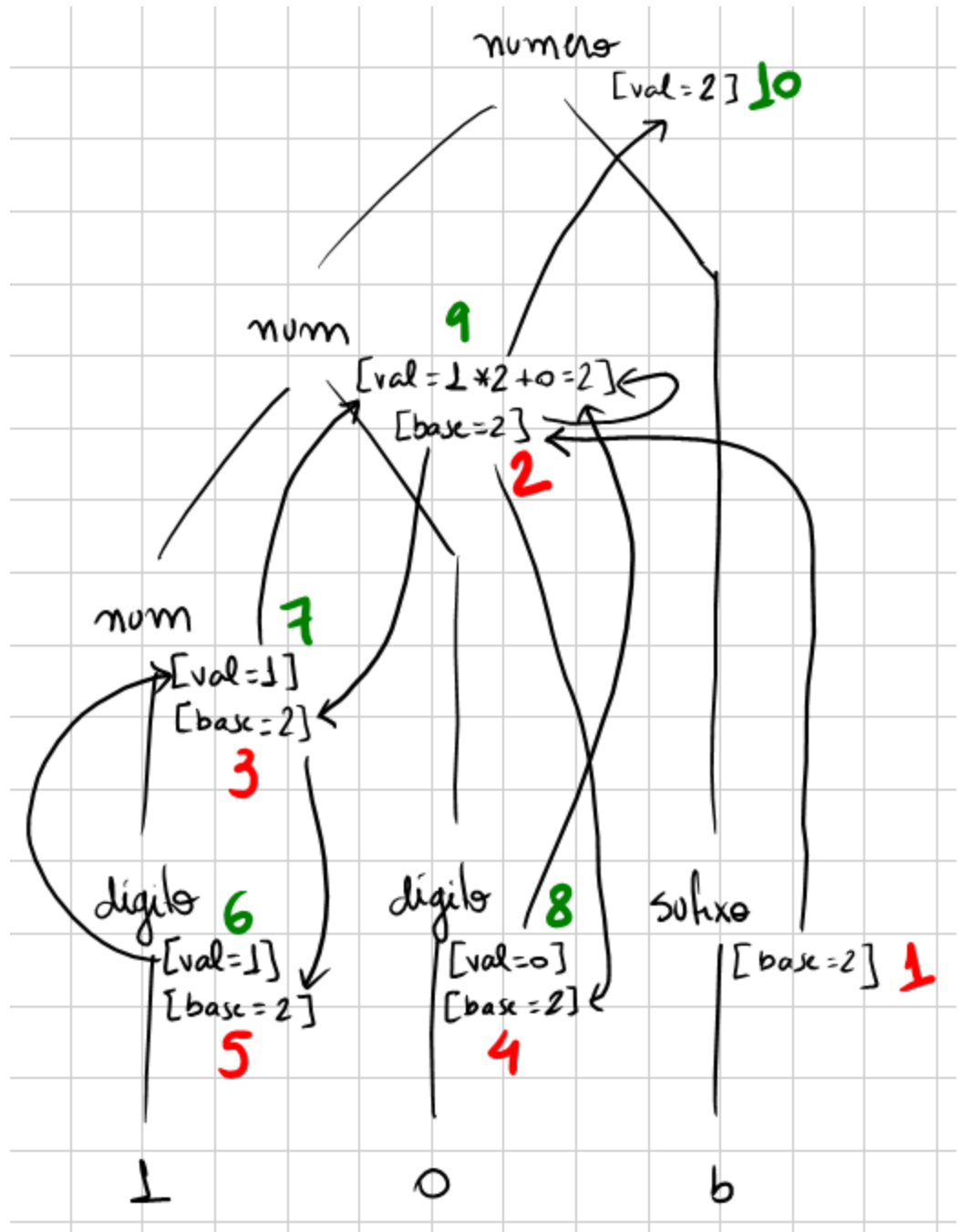


Grafos de dependência

- Os grafos de dependência podem ajudar na determinação da ordem
 - Problema da ordenação topológica
 - Ordem na qual os atributos devem ser calculados

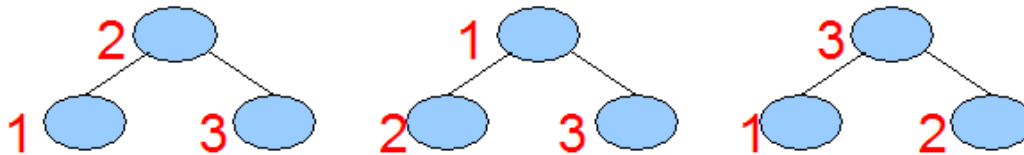
Ordem topológica

- Se existir algum ciclo
 - Não existe uma ordem
- Mesmo se não existir ciclo
 - Problema intratável
 - Pode até ser aceitável, já que a compilação acontece esporadicamente
 - Mas o algoritmo é muito complexo



Avaliação de atributos

- Na prática
 - O projetista analisa a gramática + grafo de dependência
 - Define regras de forma ad hoc, conforme as possibilidades
 - Em geral, não é muito complicado de se fazer
 - Para cada regra gramatical define-se o percurso realizado no trecho correspondente na árvore sintática
 - em-ordem, pré-ordem, pós-ordem ou arbitrário



- Diferentes percursos para diferentes tipos de atributos
- Herdados: em-ordem e pré-ordem
- Sintetizados: pós-ordem

Avaliação de atributos

- Na prática...
- Questões de implementação
 - Opcionalmente, os valores de atributos podem ser associados a parâmetros ou valores de retorno de sub-rotinas de cálculo de atributos (ao invés de serem armazenados nos nós de uma árvore sintática)
 - Interessante para a situação em que muitos atributos são usados apenas temporariamente ou como suporte para cálculo de outros atributos
 - Normalmente, atributos herdados são passados via parâmetros e atributos sintetizados via valor de retorno

Avaliação de atributos

Regras gramaticais	Regras semânticas
número \rightarrow num sufixo	número.val = num.val num.base = sufixo.base
sufixo \rightarrow b	sufixo.base = 2
sufixo \rightarrow d	sufixo.base = 10
num ₁ \rightarrow num ₂ dígito	num ₁ .val = if dígito.val= erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num \rightarrow dígito	num.val = dígito.val dígito.base = num.base
dígito \rightarrow 0	dígito.val = 0
...	...
dígito \rightarrow 9	dígito.val = if dígito.base=2 then erro else 9

```

function AvalComBase(T: nó_árvore; base: inteiro): inteiro;
var temp, temp2: inteiro;
begin
  case nó de T of
    número:
      temp:=AvalComBase(filho à direita de T,0);
      return AvalComBase(filho à esquerda de T,temp);
    num:
      temp:=AvalComBase(filho à esquerda de T,base);
      if filho à direita de T não é NIL then
        temp2:=AvalComBase(filho à direita de T,base);
        if temp<>erro and temp2<>erro then
          return base*temp+temp2
        else return erro;
      else return temp;
    sufixo:
      if filho de T=b then return 2
      else return 10;
    dígito:
      if base=2 and lexval(filho de T) > 1 then return erro
      else return lexval(filho de T);

  end;
end;

```

AvalComBase (no1,0) {

numno:

temp =

temp = 2

return

return 2

}

AvalComBase (no6,0) {

sufixo: return 2

AvalComBase (no2,2) {

num:

temp =

temp = 1

temp2 =

temp2 = 0

return

$1 \times 2 + 0$

= 2

AvalComBase (no3,2) {

num:

temp =

temp = 1

return 1

AvalComBase (no4,2) {

digito: return 1

AvalComBase (no5,2) {

digito: return 0

no1

numno

no2

num

no3

num

no4

digito

no5

digito

no6

sufixo

1

0

6

```

function AvalComBase(T: nó_árvore; base: inteiro): inteiro;
var temp, temp2: inteiro;
begin
  case nó de T of
    número:
      temp := AvalComBase(filho à direita de T, 0);
      return base * temp;
    filho à esquerda:
      num:
        temp := AvalComBase(filho à esquerda de T, 0);
        if filho à direita de T não é NIL then
          temp2 := AvalComBase(filho à direita de T, base);
          if temp <> erro and temp2 <> erro then
            return base * temp + temp2
          else return erro;
        else return temp;
    sufixo:
      if filho de T = b then return 2
      else return 10;
    dígito:
      if base = 2 and lexval(filho de T) > 1 then return erro
      else return lexval(filho de T);
  end;
end;

```

Atributo
herdado:
base

Atributo
sintetizado:
val

```

function AvalComBase(T: nó árvore binária, base: inteiro): inteiro;
var temp, temp2: inteiro;
begin
  case nó de T of
    número:
      temp:=AvalComBase(filho à direita de T,0);
      return AvalComBase(filho à esquerda de T,temp);
    num:
      temp:=AvalComBase(filho à esquerda de T,base);
      if filho à direita de T não é NIL then
        temp2:=AvalComBase(filho à direita de T,base);
        if temp<>erro and temp2<>erro then
          return base*temp+temp2
        else return erro;
      else return temp;
    sufixo:
      if filho de T=b then return
      else return 10;
    dígito:
      if base=2 and lexval(filho de T) > 1 then return erro
      else return lexval(filho de T);

  end;
end;

```

base é computado
em pré-ordem, logo
no início

val é
calculado em
pós-ordem

função que retorna
o valor lido pelo
analisador léxico

Avaliação de atributos

- Porém, uma opção melhor
 - O projetista constrói um esquema de tradução dirigida pela sintaxe

DDS (ou gramática de atributos)

PRODUÇÃO	REGRA SEMÂNTICA
$E \rightarrow E_1 + T$	$E.code = E_1.code \parallel T.code \parallel "+"$ \hookrightarrow concatenação

TDS

$E \rightarrow E_1 + T$	$\{ \text{print } "+" \}$	$ T$
$T \rightarrow id$	$\{ \text{print } id.lexema \}$	

Esquemas de TDS

- Um esquema de TDS é uma gramática livre de contexto com fragmentos de programa embutidos nos corpos das produções
 - Em qualquer lugar
- Vantagens: podem ser utilizados diretamente em geradores de analisadores
 - Yacc
 - ANTLR
- Na verdade, na prática quase não se constrói compiladores à mão
 - Portanto a opção anterior é pouco usada
 - Esquemas de TDS são normalmente a única opção

Convertendo uma DDS para um esquema TDS

- Se a DDS é S-atribuída
 - Basta adicionar as ações no final das produções
 - Demonstração

Convertendo uma DDS para um esquema TDS

- Mas se uma DDS não é S-atribuída
 - É preciso cuidado!
 - Analisadores LR e LL (os mais comuns) lêem a entrada da esquerda para a direita
 - Atributos sintetizados – OK
 - Os filhos podem ser processados em qualquer ordem
 - Atributos herdados – PROBLEMA
 - Implica em não existir dependências que apontem da direita para a esquerda (p. ex., val não pode ser computado até que o sufixo, que está no final da cadeia, seja conhecido)

Gramáticas L-atribuídas

- Restringem o uso de atributos herdados para permitir que as ações semânticas possam ser executadas durante a análise sintática em uma única passada
- Definição informal:
 - Entre os atributos associados com o corpo de uma produção, os arcos de um grafo de dependência vão da esquerda para a direita, mas nunca da direita para a esquerda

Gramáticas L-atribuídas

- Definição formal:
 - Suponha que exista uma produção $A \rightarrow X_1 X_2 \dots X_n$, e existe um atributo herdado X_i a calculado por uma regra semântica associada a essa produção. A regra só pode usar:
 - (a) atributos herdados associados com a cabeça A
 - (b) atributos herdados ou sintetizados associados com as ocorrências dos símbolos X_1, X_2, \dots, X_{i-1} (à esquerda de X_i)
 - (c) atributos herdados ou sintetizados associados com a própria ocorrência de X_i , mas somente se não houverem ciclos no grafo de dependência formado pelos atributos de X_i

Gramáticas L-atribuídas

- Exemplo

Produções	Regras semânticas
$E \rightarrow T E'$	$E'.her = T.val$ $E.val = E'.sint$
$E' \rightarrow + T E'_1$	$E'_1.her = T.val + E'.her$ $E'.sint = E'_1.sint$
$E' \rightarrow - T E'_1$	$E'_1.her = E'.her - T.val$ $E'.sint = E'_1.sint$
$E' \rightarrow \varepsilon$	$E'.sint = E'.her$
$T \rightarrow F T'$	$T'.her = F.val$ $T.val = T'.sint$
$T' \rightarrow * F T'_1$	$T'_1.her = F.val * T'.her$ $T'.sint = T'_1.sint$
$T' \rightarrow \varepsilon$	$T'.sint = T'.her$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow num$	$F.val = num.lexval$

Gramáticas L-atribuídas

- Contra-exemplo

PRODUÇÃO

$A \rightarrow BC$

REGRAS SEMÂNTICAS

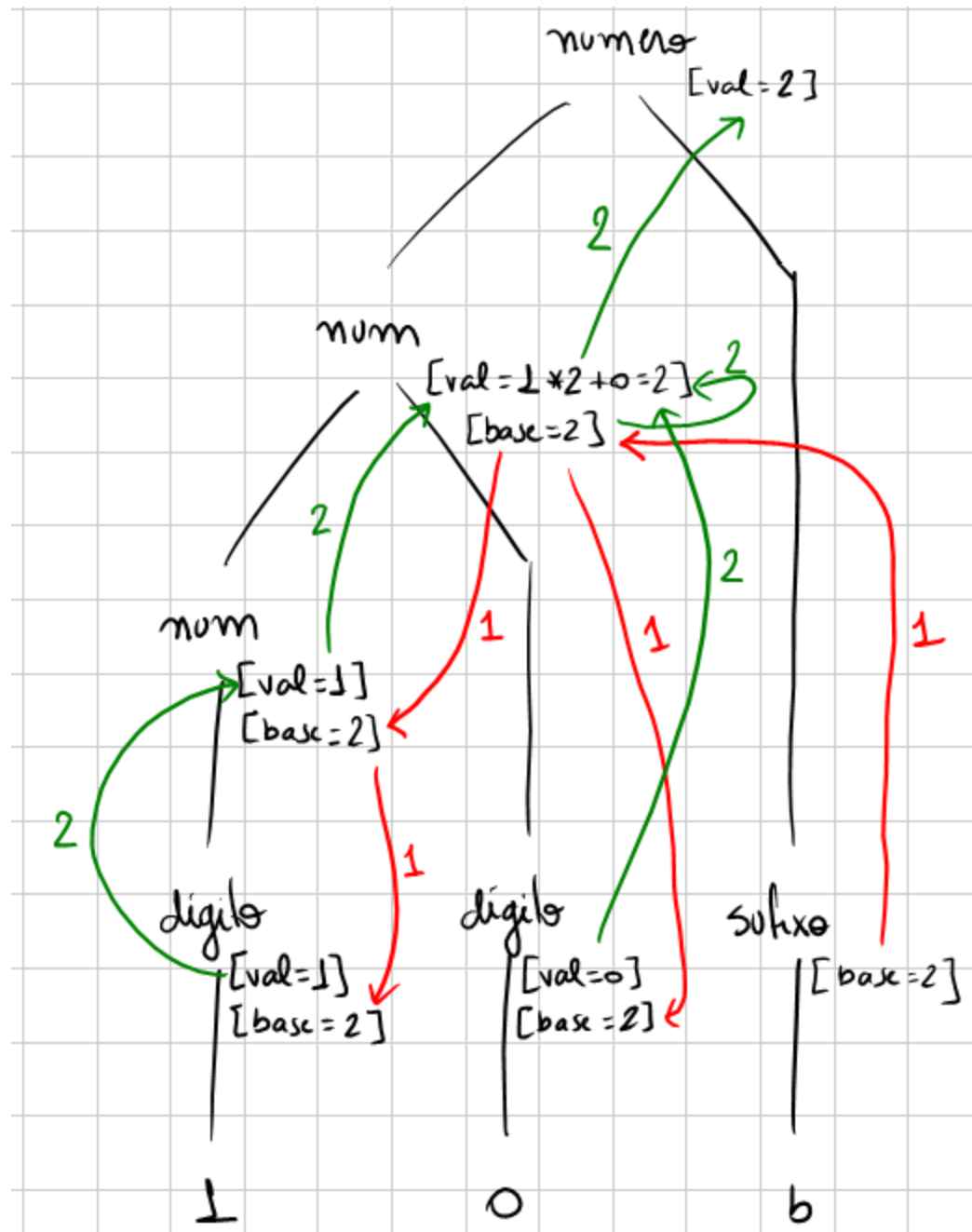
$A.s = B.b$

$B.i = C.c + A.s$

↪ ou qualquer $f(C.c, A.s)$

Outro contra-exemplo

Regras gramaticais	Regras semânticas
número → num sufixo	número.val = num.val num.base = sufixo.base
sufixo → b	sufixo.base = 2
sufixo → d	sufixo.base = 10
num ₁ → num ₂ dígito	num ₁ .val = if dígito.val = erro or num ₂ .val=erro then erro else num ₂ .val * num ₁ .base + dígito.val num ₂ .base = num ₁ .base dígito.base = num ₁ .base
num → dígito	num.val = dígito.val dígito.base = num.base
dígito → 0	dígito.val = 0
dígito → 1	dígito.val = 1
dígito → 2	dígito.val = if dígito.base=2 then erro else 2
...	...



Convertendo uma DDS L-atribuída para um esquema TDS

1. Incorpore a ação que avalia os atributos herdados para um não terminal A imediatamente antes dessa ocorrência de A no corpo da produção
 - Se mais de um atributo herdado de A dependem um do outro (mas sem formar um ciclo), ordene de forma apropriada
2. Coloque as ações que calculam atributos sintetizados da cabeça no final do seu corpo

Convertendo uma DDS L-atribuída para um esquema TDS

- Demonstração

DDS não-L-atribuída

- Se não for L-atribuída, ainda assim é possível fazer a análise semântica
 - Compilação em duas passadas
 - A primeira faz a construção da árvore
 - A segunda faz a análise
- Geradores de parsers (como o ANTLR) podem ajudar na primeira passada
 - Mas a segunda normalmente fica por conta do desenvolvedor
 - Demonstração

Fim