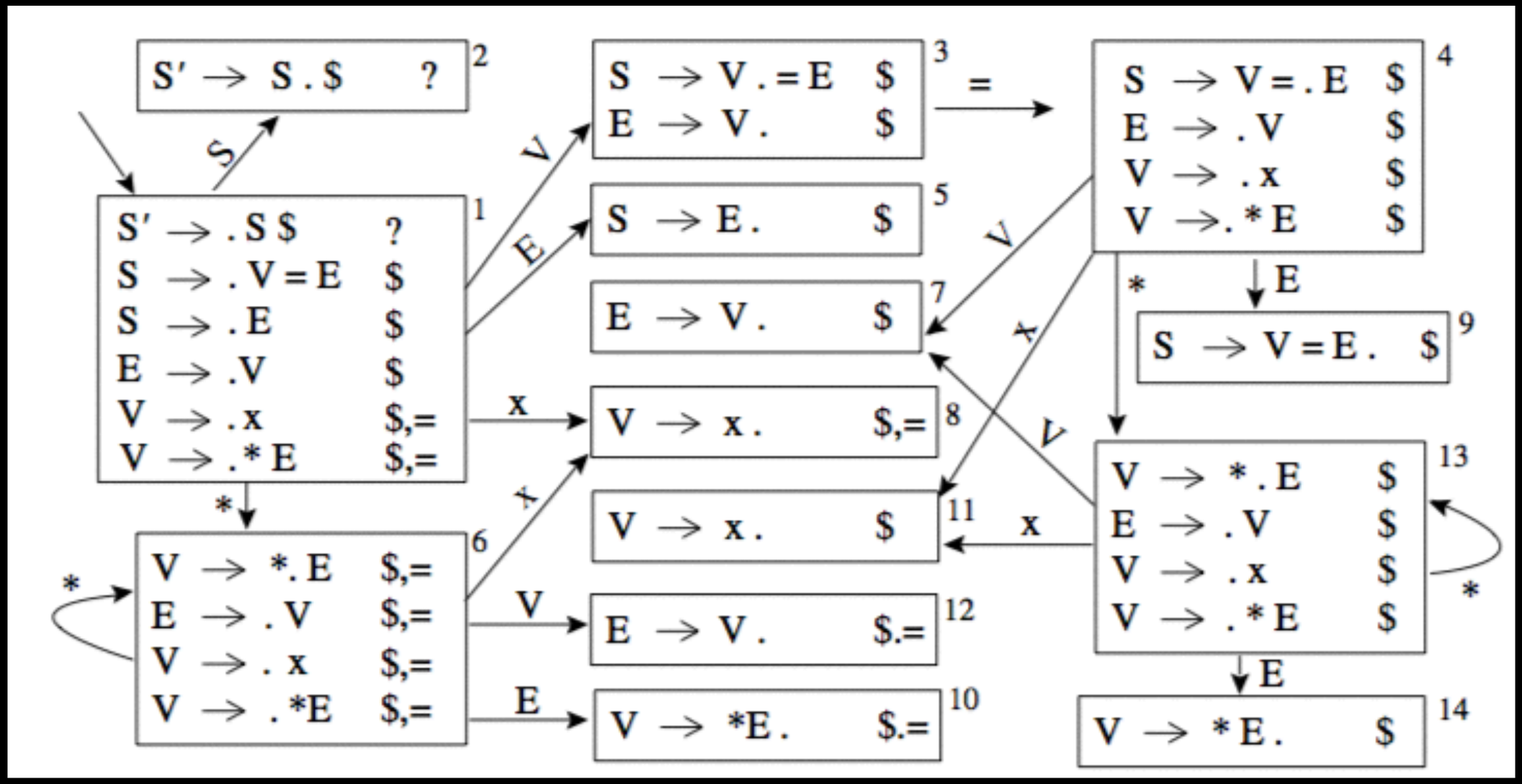
Compiladores
(IF688)

Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

$_0 S' \rightarrow S\$$ $_2 S \rightarrow E$ $_4 V \rightarrow \mathbf{x}$

$_1 S \rightarrow V = E$ $_3 E \rightarrow V$ $_5 V \rightarrow *E$



Adaptado de Andrew Appel
Modern Compiler Implementation in Java

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

Adaptado de Andrew Appel
Modern Compiler Implementation in Java

LALR

- Autômato LR(1) contém muitos estados
- Ideia: combinar estados cujos itens são exatamente iguais, exceto pelo lookahead
- Resultado é um parser LALR(1)
  - lookahead LR

LR(1) vs. LALR(1)

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			

Adaptado de Andrew Appel
Modern Compiler Implementation in Java

Situação atual

- Programas bem-formados **lexicamente**
  - identificadores tem nomes válidos
  - strings são delimitadas apropriadamente
  - nenhum caractere inesperado
- Programas bem-formados **sintaticamente**
  - declarações de classes e métodos com estrutura
  - expressões sintaticamente válidas

Isto significa que o programa é válido?

Considere x…

antes de gerar código…

Perguntas a serem respondidas

- Que tipo de valor é armazenado em x?
- Qual o tamanho de x?
- Se x é uma função, que argumentos recebe? Qual o tipo de valor retornado, se aplicável?
- Por quanto tempo o valor de x deve ser preservado?
- Quem é responsável por alocar espaço para x? (e inicializá-lo?)

Considere o programa a seguir...

```
class Factorial {
  public static void main(String[] a){
    System.out.println(new Fac().calcula(true));
  }
}

class Fac extends A {
  public int ComputeFac(int num){
    if (num < 1)
      num_aux = 1 ;
    else
      num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
  }
}
```

```
class Factorial {
  public static void main(String[] a){
    System.out.println(new Fac().calcula(true));
  }
}

class Fac extends A {
  public int ComputeFac(int num){
    if (num < 1)
      num_aux = 1 ;
    else
      num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
  }
}
```

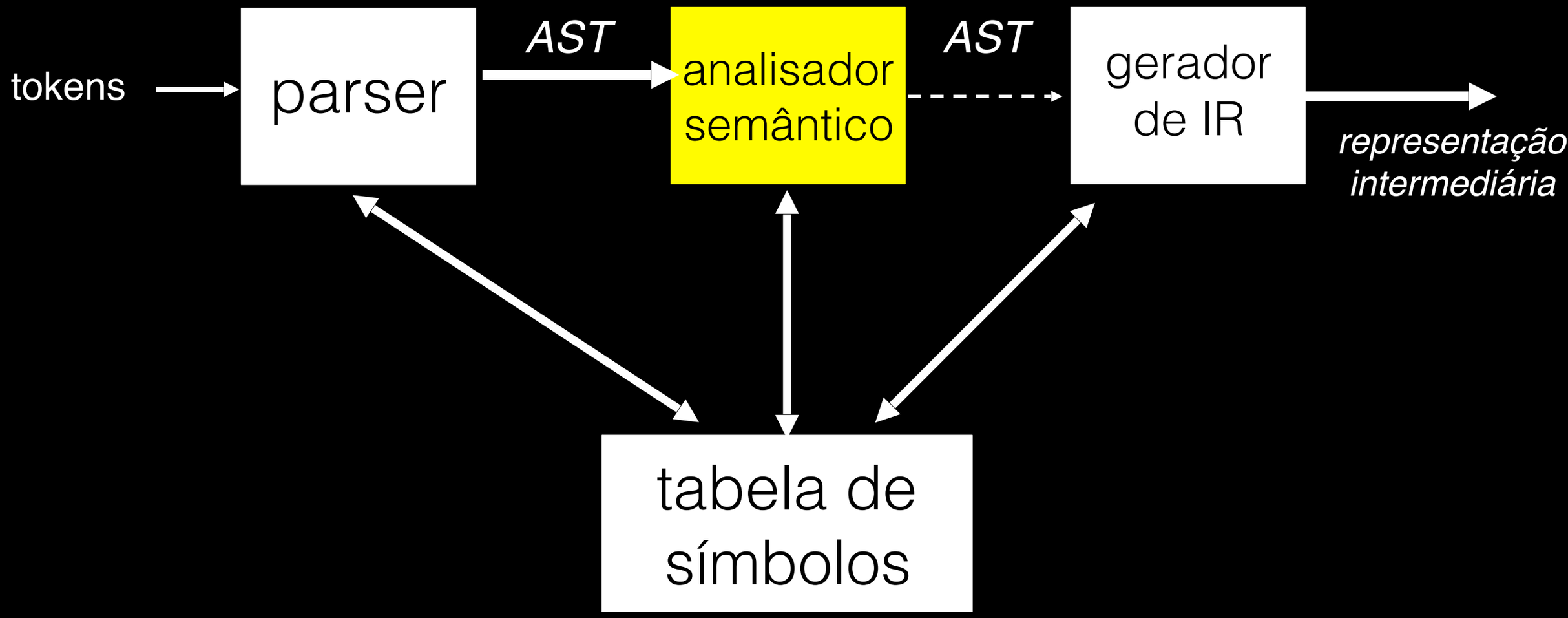
Análise Semântica

- Garantir que o programa de um significado bem definido
- Verificar propriedades que não são capturadas nas fases anteriores
- Uma vez encerrada a análise semântica, consideramos o programa de entrada válido

Propriedades

- Variáveis usadas sem terem sido declaradas
  - definição sem uso (warning)
- Expressões tem o tipo correto
- Indexação em variável que não é array
- Tipos incompatíveis
- Classes herdando de classes inexistentes
- ...

Organização do Compilador

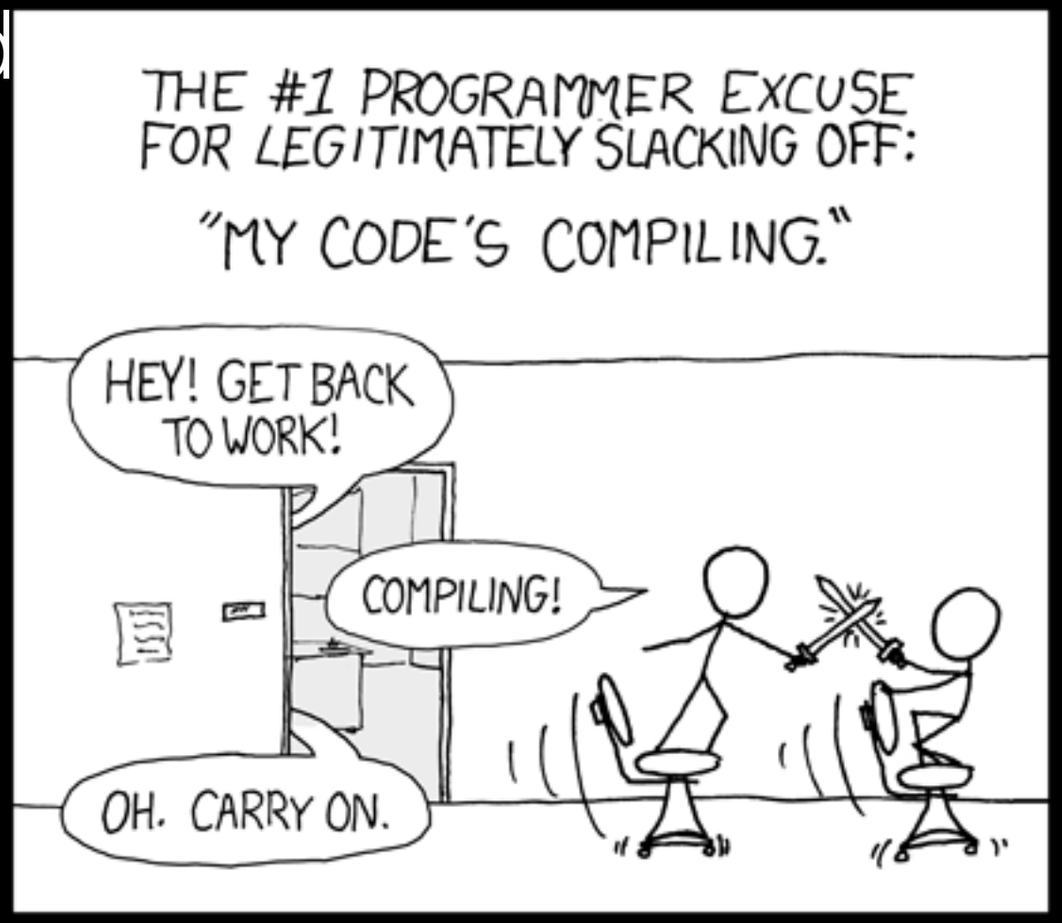


Desafios

- Rejeitar o maior número de programas incorretos
- Aceitar o maior número de programas corretos

Desafios

- Rejeitar o maior número de programas incorretos
- Aceitar o maior número d
- Fazer isto rapidamente



https://xkcd.com/303/

Outros objetivos

- Coletar informação útil sobre o programa para fases subsequentes
  - Determinar quais variáveis são associadas com cada identificador
  - Construir representação interna da hierarquia de herança
  - Contar quantas variáveis estão no escopo em um dado ponto do programa

Então… não era melhor já fazer isto durante a fase de *parsing*?

Limitações de GLC

- Como você previne definições de classes duplicadas?
- Como você diferencia variáveis de um tipo de variáveis de outro tipo?
- Como você garantiria que uma dada classe implementa todos os métodos de uma interface?

Limitações de GLC

- Para a maioria das linguagens de programação, estas limitações podem ser provadas
- lema do bombeamento para linguagens livres de contexto

Implementando

- Gramáticas de Atributos
  - fazer algumas checagens durante parsing
  - tem suas limitações
- Travessia na AST
  - construir a AST e usar funções virtuais e recursão para explorar a árvore
  - padrão de projeto *visitor*

Exemplo de Gramática de Atributos

Produção	Regra Semântica
$D \rightarrow T\ L$	
$T \rightarrow \textbf{int}$	
$T \rightarrow \textbf{real}$	
$L \rightarrow L_1, \textbf{id}$	
$L \rightarrow \textbf{id}$	

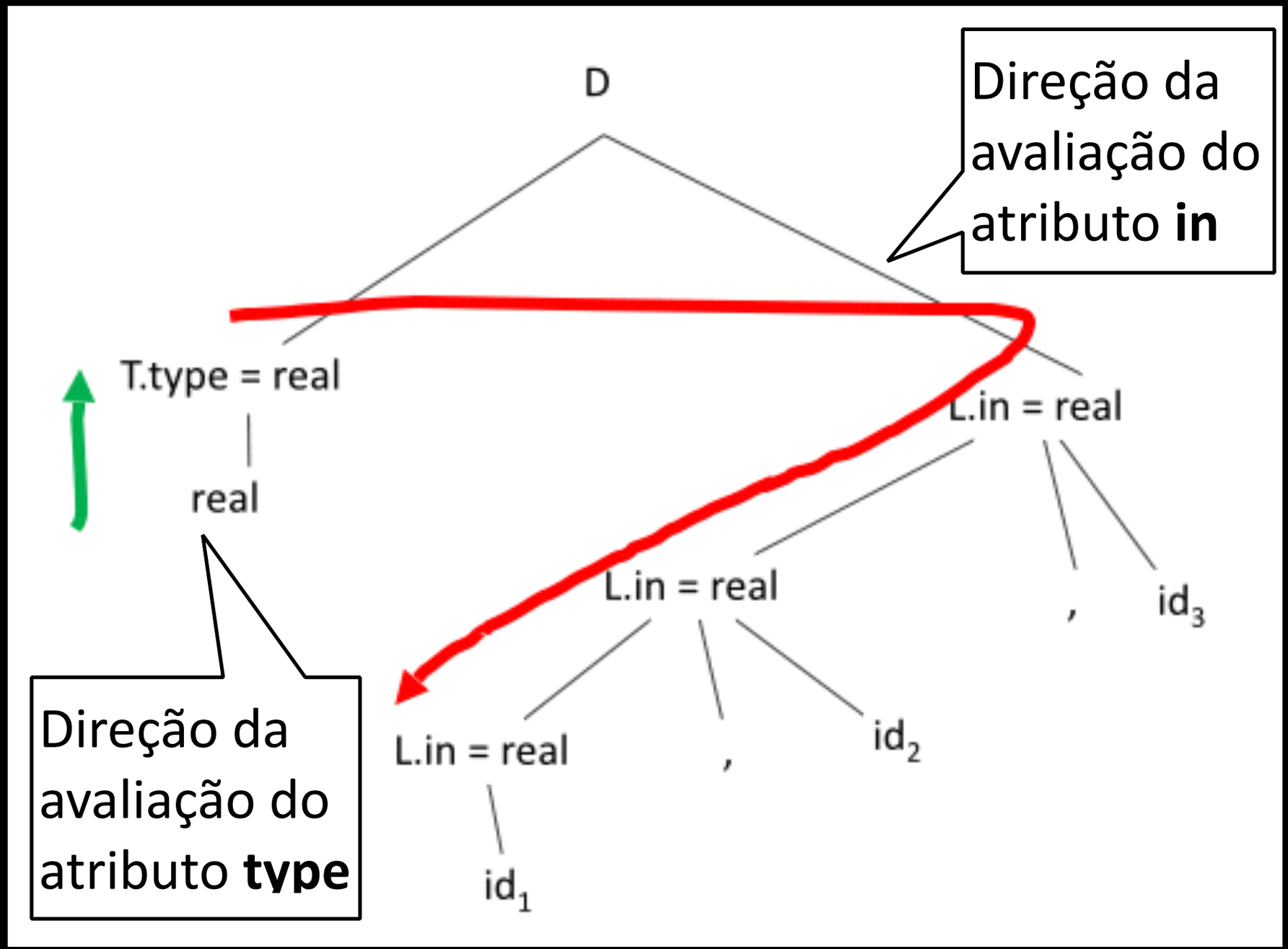
Exemplo de Gramática de Atributos

Produção	Regra Semântica
$D \rightarrow T\ L$	L.in = T.type
$T \rightarrow$ **int**	T.type = integer
$T \rightarrow$ **real**	T.type = real
$L \rightarrow L_1$, **id**	L_1.in = L.in
$L \rightarrow$ **id**	addtype(**id**.entry, L.in)

Atributo type é **sintetizado**
e atributo in é **herdado**

Como fica a árvore de **real id1, id2, id3$**?

real id1, id2, id3$



Construção de ASTs

Árvores Sintáticas Abstratas

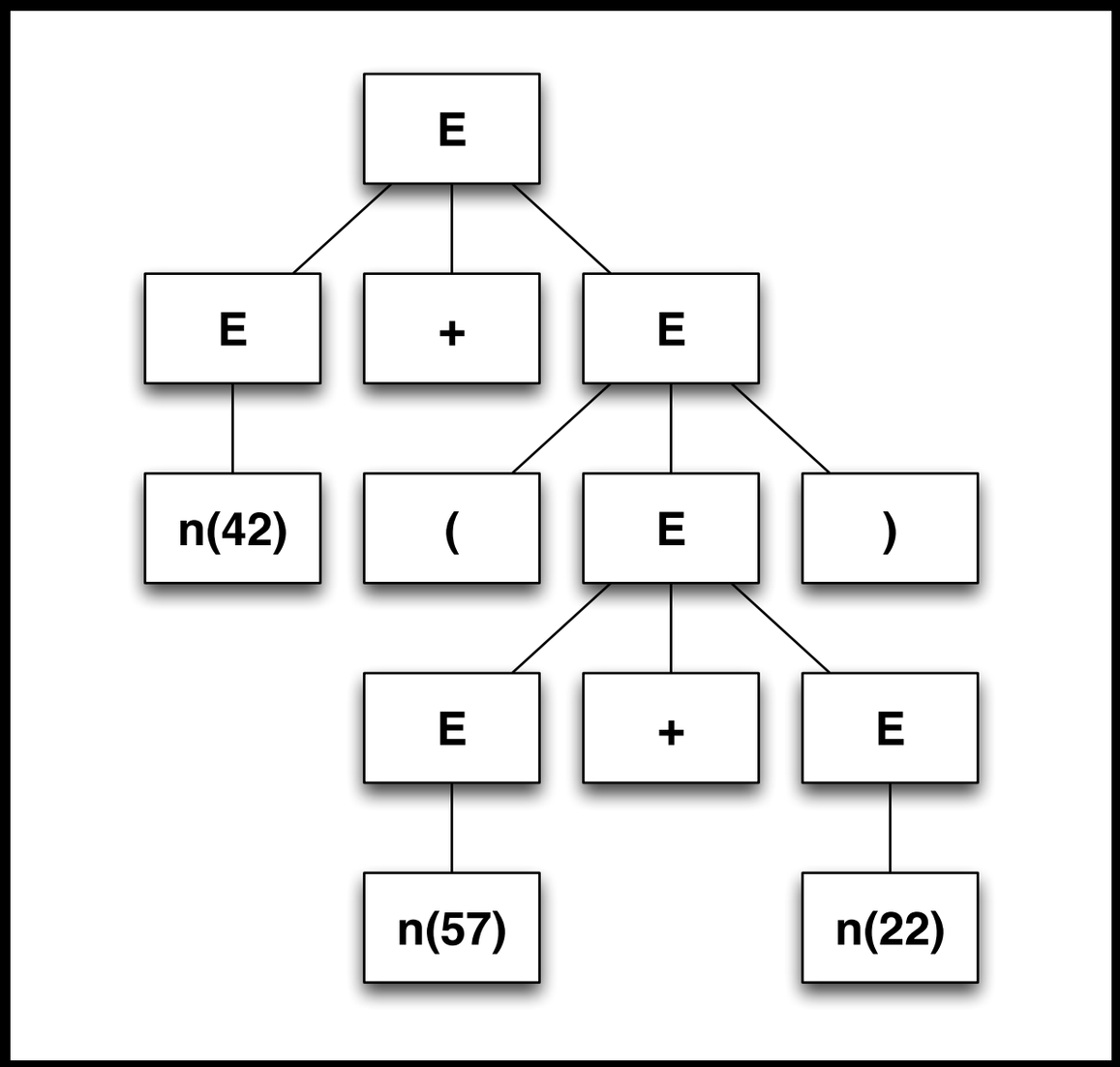
- A parse tree normalmente tem muita informação redundante e desnecessária
  - separadores, não-terminais auxiliares, etc…
- AST foca nas informações importantes, classificando nós de acordo com seu papel na estrutura da linguagem
- Representação compacta da estrutura do programa, facilita o trabalho do compilador

Exemplo

- Seja a gramática
  - $E \rightarrow \mathbf{n} \mid (\, E \,) \mid E + E$
- E a entrada 42 + (57+22)
- Após a fase de análise léxica, temos os tokens $\mathbf{n}(42)$ “+” “(“ $\mathbf{n}(57)$ “+” $\mathbf{n}(22)$ “)”

Árvore Sintática

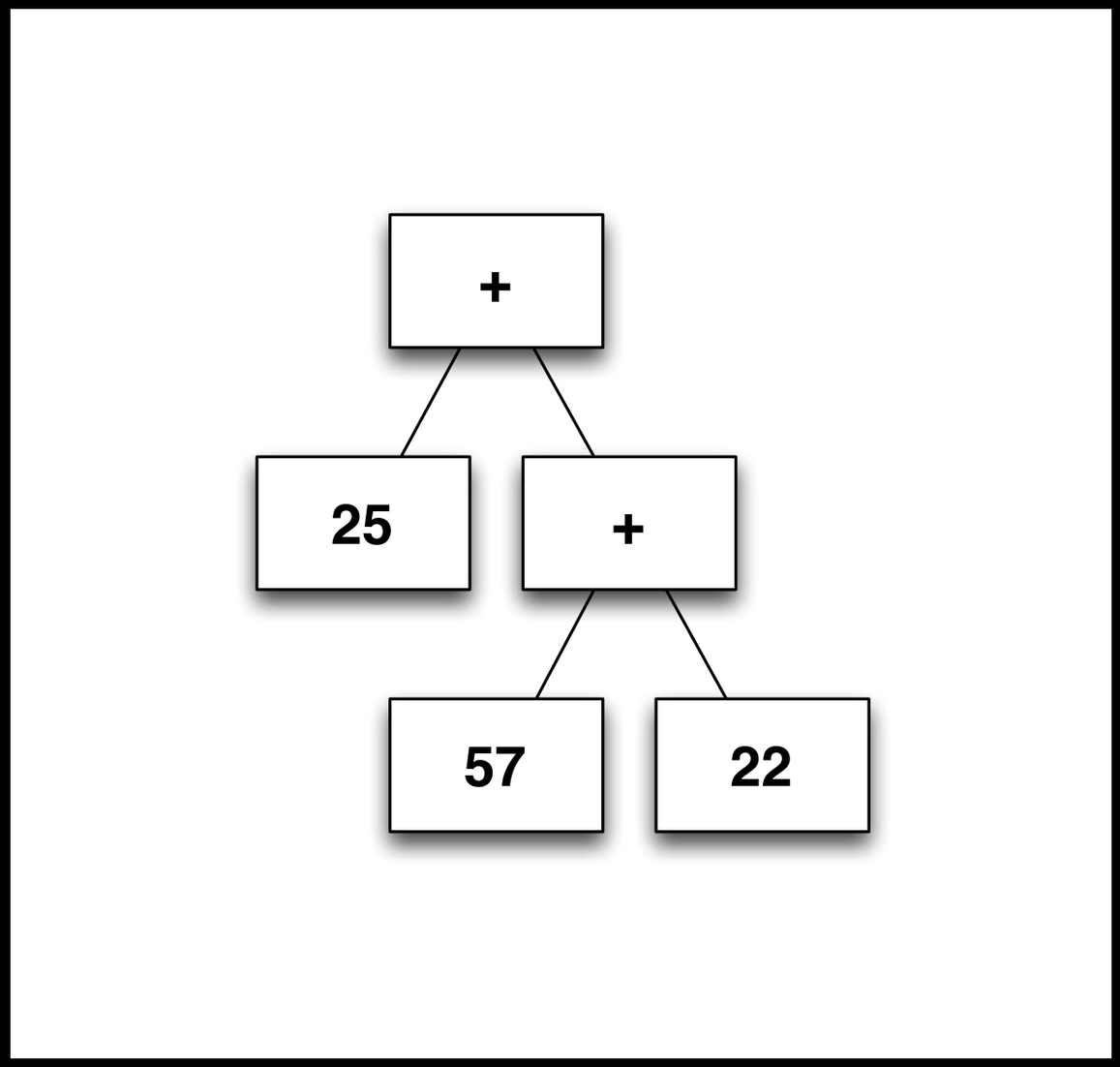
42 + (57 + 22)



$$E \rightarrow \mathrm{n} \mid (\, E \,) \mid E + E$$

AST

25 + (57 + 22)



$$E \rightarrow \mathrm{n} \mid (\, E \,) \mid E + E$$

Representando ASTs

- Cada estrutura sintática representativa da linguagem (geralmente associada com produções da gramática) é um nó da AST

- Em termos de Java e OO, criamos uma classe para cada tipo de nó

- Não-terminais com várias produções ganham interface ou classe abstrata, e cada produção implementa ou estende a interface/classe pai