Compiladores
(IF688)

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

Árvores Sintáticas Abstratas

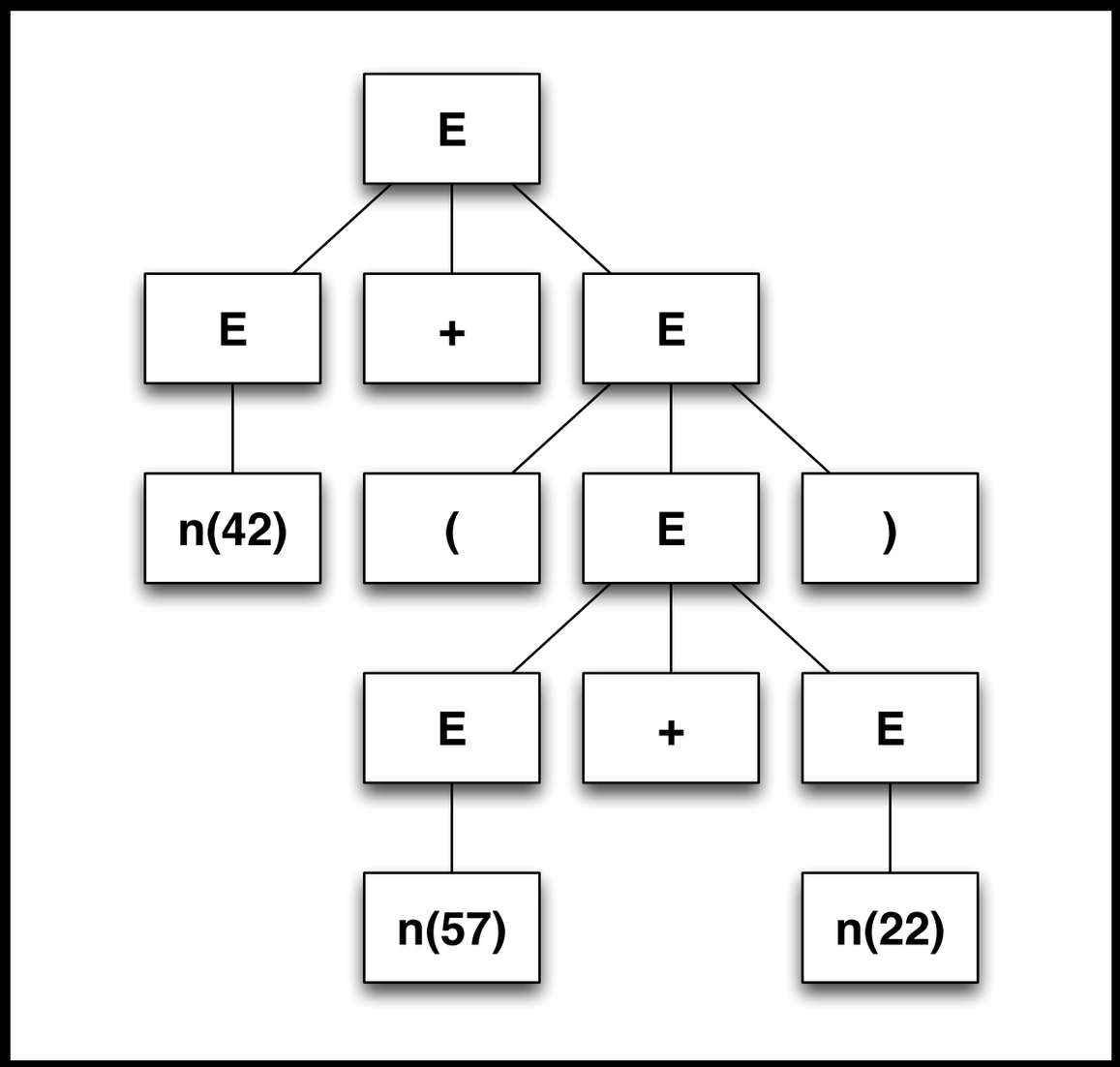
- A parse tree normalmente tem muita informação redundante e desnecessária
  - separadores, não-terminais auxiliares, etc…
- AST foca nas informações importantes, classificando nós de acordo com seu papel na estrutura da linguagem
- Representação compacta da estrutura do programa, facilita o trabalho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

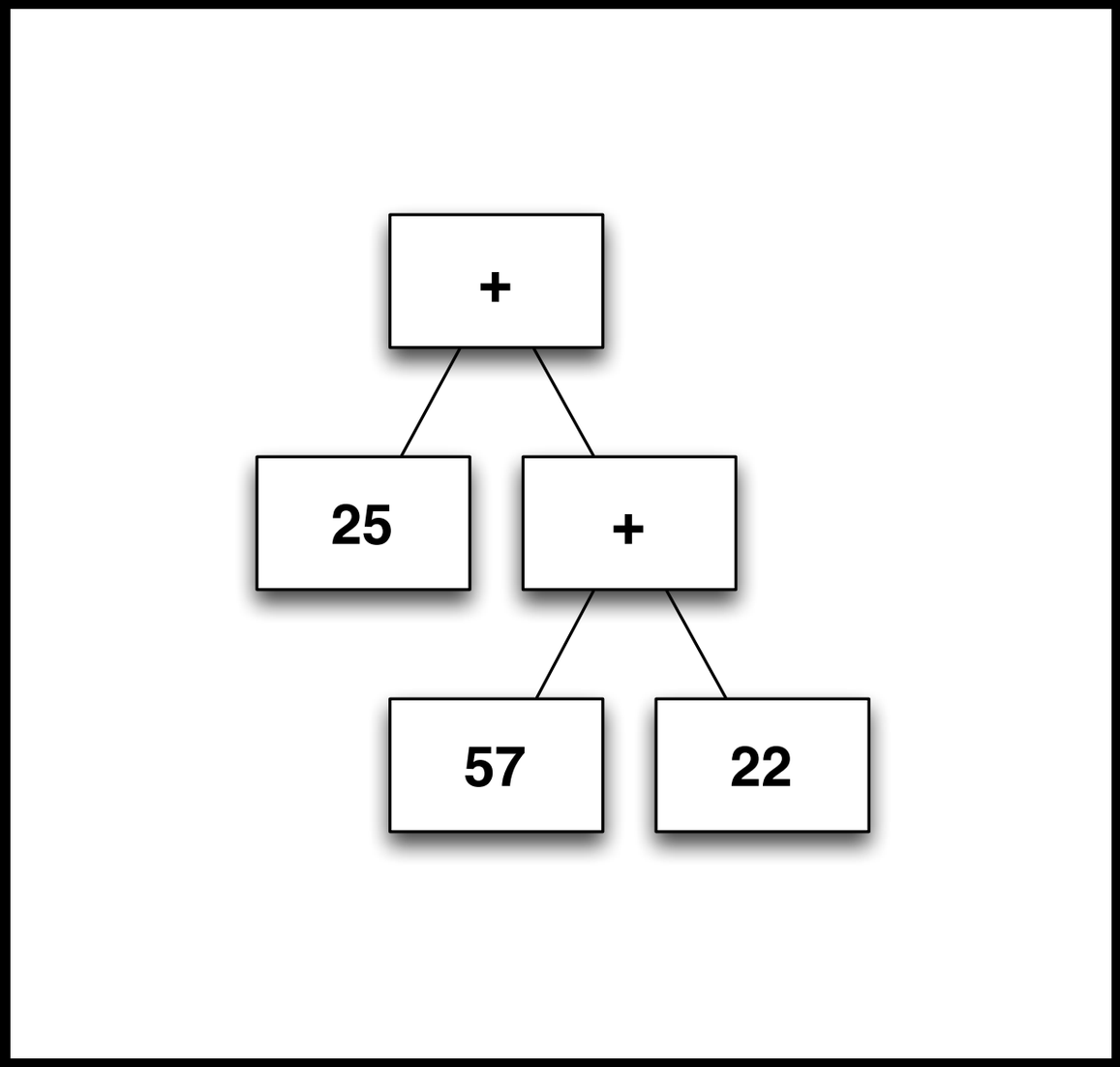
42 + (57 + 22)



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

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

ASTs e OO

- No código visto, cada classe da AST tem um construtor e um método **`eval()`**, para retornar o valor da expressão representada
- Um estilo orientado a objetos de programar
- Considere a alternativa…

Como poderíamos escrever o código para avaliação de expressões separado do código de representação (AST) das expressões?

Separando sintaxe de interpretação.

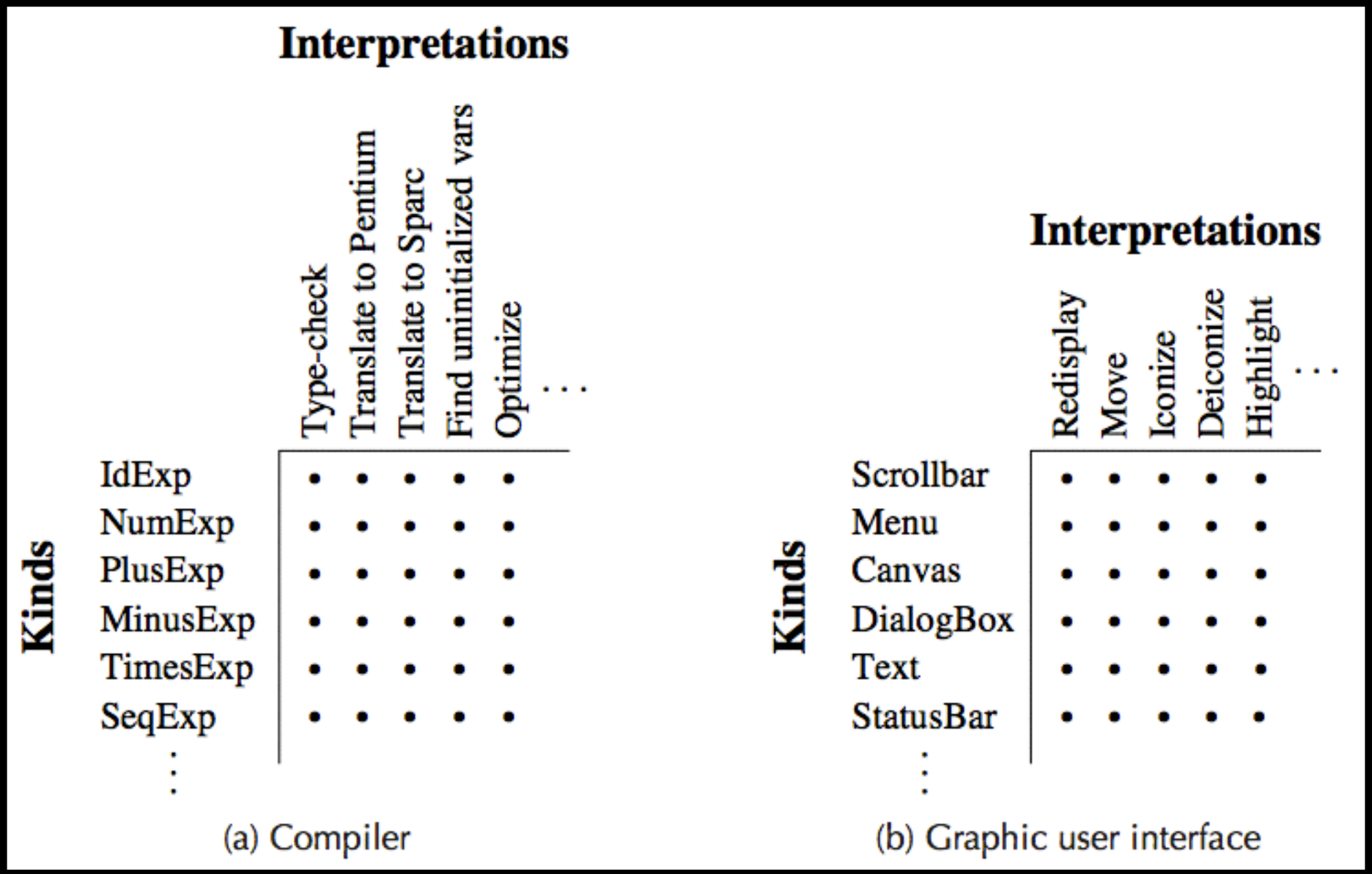
Tipos e Interpretações

- A escolha de um estilo pode afetar a modularidade do compilador
- Em geral, temos vários *tipos* (*kinds*) de objetos: statements de assignment, print, compostos, etc.
- Também podemos ter diferentes *interpretações* de cada um destes objetos: checar tipos, traduzir para código Pentium, traduzir para código Sparc, otimizar, interpretar, etc…

Tipos e Interpretações

- Cada interpretação é aplicada a um tipo
- Se adicionarmos um novo tipo, devemos implementar cada interpretação para este tipo
- Se adicionarmos uma interpretação, temos que implementá-la para cada um dos tipos

Direções de Modularidade



Adaptado de Andrew Appel
Modern Compiler Implementation in Java

Modularidade

- No estilo de sintaxe separada da interpretação, adicionar uma nova interpretação é fácil e modular.
  - Uma nova função é escrita, com cláusulas para todos os diferentes tipos
  - Adicionar novo tipo afeta a implementação de todas as funções de interpretação

Modularidade

- No estilo orientado a objetos, cada interpretação é um método em cada uma das classes. É fácil e modular adicionar um novo tipo.
- Todas as interpretações deste tipo são agrupadas como métodos da nova classe
- Adicionar uma interpretação significa adicionar um método a todas as classes existentes

GUIs vs. Compiladores

qual estilo é mais adequado?

GUIs vs. Compiladores

- Em interfaces gráficas, cada aplicação terá seu conjunto de widgets
  - Embora não seja possível predeterminar os widgets que serão usados, as operações (interpretações) são comuns a todos
- Em compiladores, podemos querer adicionar interpretações, como otimizações, traduzir para código-alvo de uma nova máquina, etc.

Padrão de Projeto

Visitor

Visitor

- Um visitor implementa uma interpretação
- É um objeto que contem um método visit para cada classe da AST
- Cada classe da AST deve conter (pelo menos) um método accept
  - este método serve como gancho para todas as interpretações
  - é chamado pelo visitor e tem apenas uma tarefa, passa o controle de volta ao método apropriado do visitor

Desacoplando estrutura de dados do algoritmo que opera na estrutura

Exemplo código