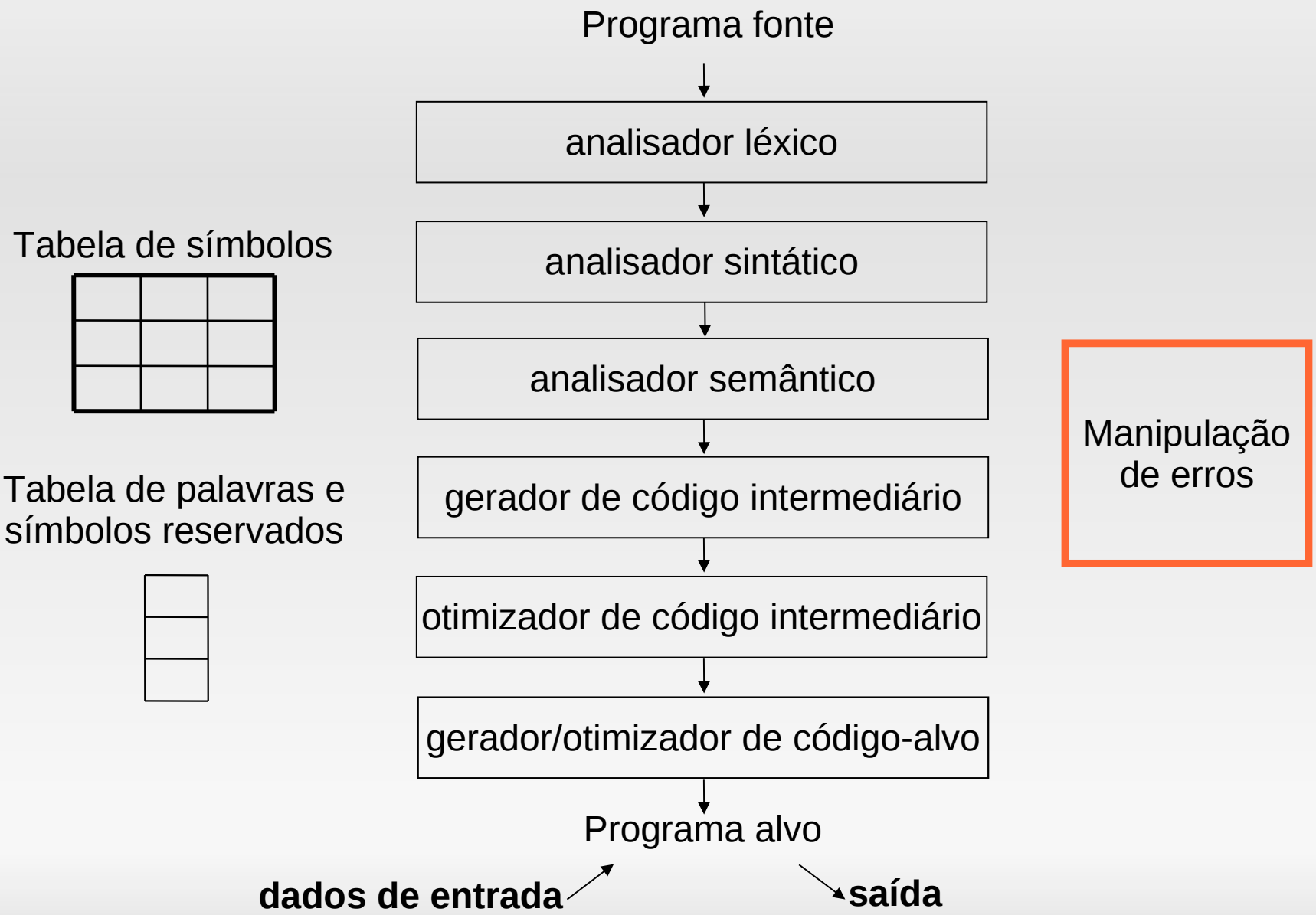
Construção de Compiladores

Análise Sintática – Tratamento de Erros

Profa. Helena Caseli
helenacaseli@dc.ufscar.br

Processo de Tradução



AS – Tratamento de Erros

- O manipulador de erros deve
  - Relatar os erros o quanto antes, de modo claro e preciso
  - (opcionalmente) corrigir os erros mais simples (p. ex. pontuação)
  - Continuar a análise mesmo que um erro seja encontrado
  - Evitar o problema da cascata de erros
    - Um erro gera uma sequência de mensagens espúrias
  - Evitar laços infinitos de erros
    - Uma cascata infinita de mensagens de erros é gerada sem consumir novas entradas
  - Não atrasar muito o processamento de programas corretos

  - → Tratamento efetivo de erros pode ser uma tarefa difícil
  - → A maioria das técnicas é *ad hoc*

AS – Tratamento de Erros

- Felizmente, a maioria dos erros são simples
  - Pesquisa com estudantes de Pascal
    - 80% dos enunciados contém apenas um erro
      - 13% tem dois
    - 90% são erros em um único *token*
    - 60% são erros de pontuação
      - Por exemplo, uso do ponto e vírgula (;)
    - 20% são erros de operadores e operandos
      - Por exemplo, omissão de : no símbolo :=
    - 15% são erros de palavras-chave
      - Por exemplo, erros ortográficos (wrteln)

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Modo pânico
  - Recuperação de frases
  - Produções de erro
  - Correção global

  - ➔ A maioria das técnicas é *ad hoc*

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Modo pânico
    - Como funciona?
      - Consome os *tokens* da entrada até encontrar um ponto de sincronização
    - *Tokens* de sincronização
      - Seguidores e Primeiros
    - Vantagens
      - É simples de implementar
      - Pode ser usado na maioria dos métodos de análise sintática
      - As cascatas de erros são evitadas (até certo ponto), pois novas mensagens de erro não são geradas enquanto os *tokens* de entrada são consumidos

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Modo pânico
    - Exemplo

```
while (x<2 do
read(y)...
```

Ao notar a falta do parênteses:
- relata-se a falta do mesmo e
- consome-se tudo até que o token de sincronização *read*, por exemplo, seja encontrado, a partir de onde a análise é retomada

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Recuperação de frases (correção local)
    - Ao se detectar um erro, realizam-se correções locais na entrada
      - Por exemplo, substituição de vírgula por ponto e vírgula, remover um ponto e vírgula estranho ou inserir um
    - Planejamento das correções possíveis pelo projetista do compilador
    - Atenção: pode gerar um ciclo infinito de erros/correções!
      - Inserem-se símbolos infinitamente

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Produções de erro
    - Aumenta-se a gramática da linguagem com produções para reconhecer as produções ilegais e tratá-las adequadamente
    - Requer um bom conhecimento dos erros que podem ser cometidos
    - → Método disponível no Yacc

AS – Tratamento de Erros

- Estratégias de tratamento de erros
  - Correção global
    - Escolhe-se uma sequência mínima de modificações no programa que o torne correto com o menor custo possível
    - Método muito custoso de se implementar; apenas de interesse teórico

ASD – Tratamento de Erros

- Estratégias de tratamento de erros
  - ASD preditiva recursiva
    - Modo pânico
      - Cada procedimento recursivo tem um parâmetro adicional
        - Conjunto de *tokens* de sincronização
      - Ao encontrar um erro
        - O analisador consome os *tokens* de entrada até encontrar um *token* de sincronização
  - ASD preditiva não recursiva
    - Modo pânico
      - *Tokens* de sincronização em uma nova pilha

      OU
      - Conjuntos de *tokens* de sincronização são construídos diretamente na tabela sintática (nas células de erro originalmente em branco)

ASA – Tratamento de Erros

- Estratégias de tratamento de erros
  - Análise LR
    - Características
      - Os erros são identificados sempre no momento mais cedo (na leitura dos *tokens*)
      - Nesse tipo de analisador a cadeia de símbolos já empilhada está, com certeza, sintaticamente correta
      - As lacunas na tabela AÇÃO representam situações de erro e, como tal, devem acionar rotinas de recuperação
    - Modo pânico – possível implementação
      - Retirar estados da pilha até encontrar uma TRANSIÇÃO não vazia
      - Se houver uma AÇÃO legal para a entrada e um dos estados de TRANSIÇÃO, coloque esse estado na pilha e reinicie a análise
      - Se não houver uma AÇÃO legal para a entrada e um dos estados de TRANSIÇÃO, avance a entrada até encontrar uma ação legal ou o fim da entrada

AS Descendente X Ascendente

- Características
  - Os métodos de análise sintática ascendente são mais poderosos (capazes de lidar com um número maior de gramáticas)
  - As verificações à frente na pilha são mais simples do que as verificações à frente na entrada
  - LL(k) vs. LR(k)
    - LL(k)
      - A produção é selecionada olhando-se apenas os *k* primeiros símbolos que seu lado direito pode derivar
    - LR(k)
      - O lado direito de uma produção é reconhecido conhecendo-se tudo que foi derivado a partir desse lado direito (na pilha) mais o esquadrinhamento antecipado (*lookahead*) de *k* símbolos (da cadeia de entrada)
    - → LR é mais poderoso do que LL