Análise sintática

O papel do analisador sintático

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Gramáticas no projeto de linguagens e na escrita de compiladores

- Uma gramática oferece uma especificação sintática precisa para uma linguagem de programação

Gramáticas no projeto de linguagens e na escrita de compiladores

- Uma gramática oferece uma especificação sintática precisa para uma linguagem de programação
- Para certas classes de gramática é possível gerar o analisador sintático automaticamente

Gramáticas no projeto de linguagens e na escrita de compiladores

- Uma gramática oferece uma especificação sintática precisa para uma linguagem de programação
- Para certas classes de gramática é possível gerar o analisador sintático automaticamente
- O processo de construção do analisador sintático permite identificar ambiguidades sintáticas e outras construções difíceis de analisar, ajudando o projeto inicial de uma linguagem

Gramáticas no projeto de linguagens e na escrita de compiladores

- Uma gramática oferece uma especificação sintática precisa para uma linguagem de programação
- Para certas classes de gramática é possível gerar o analisador sintático automaticamente
- O processo de construção do analisador sintático permite identificar ambiguidades sintáticas e outras construções difíceis de analisar, ajudando o projeto inicial de uma linguagem
- Uma gramática bem projetada implica uma estrutura na linguagem que é útil para a tradução para a linguagem alvo e para a detecção de erros

Gramáticas no projeto de linguagens e na escrita de compiladores

- Uma gramática oferece uma especificação sintática precisa para uma linguagem de programação
- Para certas classes de gramática é possível gerar o analisador sintático automaticamente
- O processo de construção do analisador sintático permite identificar ambiguidades sintáticas e outras construções difíceis de analisar, ajudando o projeto inicial de uma linguagem
- Uma gramática bem projetada implica uma estrutura na linguagem que é útil para a tradução para a linguagem alvo e para a detecção de erros
- A inclusão de novas características em uma linguagem é facilitada se a implementação foi baseada em uma descrição gramatical

O papel do analisador sintático

- O analisador sintático recebe, do analisador léxico, uma cadeia de *tokens* e verifica se esta cadeia pode ser gerada pela gramática da linguagem-fonte

O papel do analisador sintático

- O analisador sintático recebe, do analisador léxico, uma cadeia de *tokens* e verifica se esta cadeia pode ser gerada pela gramática da linguagem-fonte
- Ele também deve relatar quaisquer erros de sintaxe que possam surgir, de forma inteligível

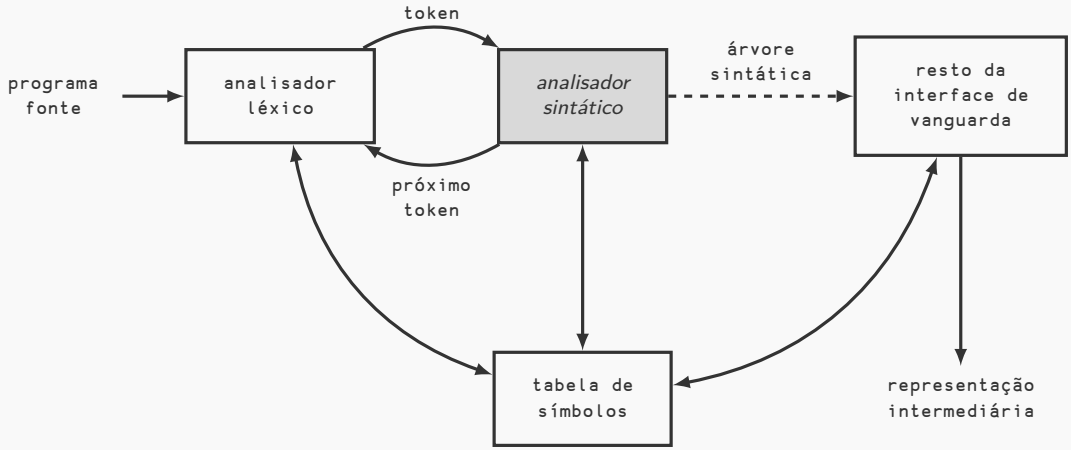
O papel do analisador sintático

- O analisador sintático recebe, do analisador léxico, uma cadeia de *tokens* e verifica se esta cadeia pode ser gerada pela gramática da linguagem-fonte
- Ele também deve relatar quaisquer erros de sintaxe que possam surgir, de forma inteligível
- Se possível, ele deve se recuperar dos erros mais comuns

O papel do analisador sintático

- O analisador sintático recebe, do analisador léxico, uma cadeia de *tokens* e verifica se esta cadeia pode ser gerada pela gramática da linguagem-fonte
- Ele também deve relatar quaisquer erros de sintaxe que possam surgir, de forma inteligível
- Se possível, ele deve se recuperar dos erros mais comuns
- Ele pode produzir, explicitamente ou implicitamente, uma árvore sintática

Posição do analisador sintático num modelo de compilador



Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos

Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos
- O primeiro tipo são os métodos universais, que podem tratar quaisquer gramáticas

Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos
- O primeiro tipo são os métodos universais, que podem tratar quaisquer gramáticas
- Exemplos de métodos gerais: o algoritmo de Younger-Kasami e o algoritmo de Early

Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos
- O primeiro tipo são os métodos universais, que podem tratar quaisquer gramáticas
- Exemplos de métodos gerais: o algoritmo de Younger-Kasami e o algoritmo de Early
- Tais métodos são ineficientes para um compilador de produção

Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos
- O primeiro tipo são os métodos universais, que podem tratar quaisquer gramáticas
- Exemplos de métodos gerais: o algoritmo de Younger-Kasami e o algoritmo de Early
- Tais métodos são ineficientes para um compilador de produção
- Os outros dois tipos são os analisadores *top-down* e os analisadores *bottom-up*

Tipos de analisadores sintáticos

- Há três tipos gerais de analisadores sintáticos
- O primeiro tipo são os métodos universais, que podem tratar quaisquer gramáticas
- Exemplos de métodos gerais: o algoritmo de Younger-Kasami e o algoritmo de Early
- Tais métodos são ineficientes para um compilador de produção
- Os outros dois tipos são os analisadores *top-down* e os analisadores *bottom-up*
- Estes métodos trabalham apenas com uma determinada subclasse de gramáticas, porém são mais eficientes que os métodos universais

Tipos de erro que um compilador pode encontrar

Dentre os diferentes tipos de erro que um compilador podem encontrar, há quatro importantes tipos de erro:

Tipos de erro que um compilador pode encontrar

Dentre os diferentes tipos de erro que um compilador podem encontrar, há quatro importantes tipos de erro:

1. erros léxicos, relacionados aos tipos e a identificação dos tokens

Tipos de erro que um compilador pode encontrar

Dentre os diferentes tipos de erro que um compilador podem encontrar, há quatro importantes tipos de erro:

1. erros léxicos, relacionados aos tipos e a identificação dos tokens
2. erros sintáticos, que surgem da inconformidade da disposição dos tokens em relação às regras gramaticais

Tipos de erro que um compilador pode encontrar

Dentre os diferentes tipos de erro que um compilador podem encontrar, há quatro importantes tipos de erro:

1. erros léxicos, relacionados aos tipos e a identificação dos tokens
2. erros sintáticos, que surgem da inconformidade da disposição dos tokens em relação às regras gramaticais
3. erros semânticos, que lidam com questões de compatibilidade de tipos e o significado das expressões da linguagem

Tipos de erro que um compilador pode encontrar

Dentre os diferentes tipos de erro que um compilador podem encontrar, há quatro importantes tipos de erro:

1. erros léxicos, relacionados aos tipos e a identificação dos tokens
2. erros sintáticos, que surgem da inconformidade da disposição dos tokens em relação às regras gramaticais
3. erros semânticos, que lidam com questões de compatibilidade de tipos e o significado das expressões da linguagem
4. erros lógicos, que envolvem expressões semanticamente corretas mas que podem levar a laços infinitos e outros erros

Metas de um tratador de erros sintáticos

Há três metas principais a serem atingidas por um tratador de erros sintáticos:

Metas de um tratador de erros sintáticos

Há três metas principais a serem atingidas por um tratador de erros sintáticos:

1. Relatar a presença de erros de forma clara e precisa

Metas de um tratador de erros sintáticos

Há três metas principais a serem atingidas por um tratador de erros sintáticos:

1. Relatar a presença de erros de forma clara e precisa
2. Recuperar-se de cada erro o mais rápido possível e de forma que se possa identificar os erros subsequentes

Metas de um tratador de erros sintáticos

Há três metas principais a serem atingidas por um tratador de erros sintáticos:

1. Relatar a presença de erros de forma clara e precisa
2. Recuperar-se de cada erro o mais rápido possível e de forma que se possa identificar os erros subsequentes
3. Não atrasar o processamento de programas sintaticamente corretos

Metas de um tratador de erros sintáticos

Há três metas principais a serem atingidas por um tratador de erros sintáticos:

1. Relatar a presença de erros de forma clara e precisa
2. Recuperar-se de cada erro o mais rápido possível e de forma que se possa identificar os erros subsequentes
3. Não atrasar o processamento de programas sintaticamente corretos

Cada uma destas metas representa um desafio à parte.

Relato de erros

- Uma vez encontrado um erro, o tratador deve produzir um relato (saída) que indique a natureza e a localização do erro no programa-fonte

Relato de erros

- Uma vez encontrado um erro, o tratador deve produzir um relato (saída) que indique a natureza e a localização do erro no programa-fonte
- Em geral, o erro acontece no próprio token, ou alguns tokens antes, do token que está sendo avaliado quando o erro é identificado

Relato de erros

- Uma vez encontrado um erro, o tratador deve produzir um relato (saída) que indique a natureza e a localização do erro no programa-fonte
- Em geral, o erro acontece no próprio token, ou alguns tokens antes, do token que está sendo avaliado quando o erro é identificado
- Uma estratégia simples é reportar a linha do programa-fonte onde se encontra o token em questão

Relato de erros

- Uma vez encontrado um erro, o tratador deve produzir um relato (saída) que indique a natureza e a localização do erro no programa-fonte
- Em geral, o erro acontece no próprio token, ou alguns tokens antes, do token que está sendo avaliado quando o erro é identificado
- Uma estratégia simples é reportar a linha do programa-fonte onde se encontra o token em questão
- Se for possível identificar a natureza do erro, deve ser produzida uma mensagem compreensível sobre o erro e uma possível sugestão de correção

Exemplo de código C++ com erro e a mensagem produzida pelo GCC

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World":

    return 0;
}
```

```
codes/erro.cpp: In function ‘int main()’:
codes/erro.cpp:7:26: error: expected ‘;’ before ‘:’ token
    7 |     cout << "Hello World":
      |                          ^
      |                          ;
compilation terminated due to -Wfatal-errors.
```

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização
- Em geral, os tokens de sincronização são delimitadores

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização
- Em geral, os tokens de sincronização são delimitadores
- Esta é a mais simples de implementar dentre as quatro estratégias de recuperação de erros e tem a garantia de não entrar em um laço infinito

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização
- Em geral, os tokens de sincronização são delimitadores
- Esta é a mais simples de implementar dentre as quatro estratégias de recuperação de erros e tem a garantia de não entrar em um laço infinito
- A segunda estratégia é a recuperação de frases, onde é feita uma pequena correção local que permita ao analisador seguir adiante

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização
- Em geral, os tokens de sincronização são delimitadores
- Esta é a mais simples de implementar dentre as quatro estratégias de recuperação de erros e tem a garantia de não entrar em um laço infinito
- A segunda estratégia é a recuperação de frases, onde é feita uma pequena correção local que permita ao analisador seguir adiante
- Um exemplo seria inserir um ponto-e-vírgula ausente

Estratégias de recuperação de erros

- Dentre as diferentes estratégias de recuperação de erros, há quatro delas que possuem ampla aplicabilidade
- Na modalidade de desespero o compilador, ao descobrir um erro, segue descartando símbolos da entrada, um por vez, até que seja encontrado um token que pertence ao conjunto dos denominados tokens de sincronização
- Em geral, os tokens de sincronização são delimitadores
- Esta é a mais simples de implementar dentre as quatro estratégias de recuperação de erros e tem a garantia de não entrar em um laço infinito
- A segunda estratégia é a recuperação de frases, onde é feita uma pequena correção local que permita ao analisador seguir adiante
- Um exemplo seria inserir um ponto-e-vírgula ausente
- Não funciona muito bem se o erro aconteceu antes do token que identificou o erro

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção
- Tem como desvantagem o fato de impactar na performance, por conta das verificações adicionais que não faziam parte da gramática original da linguagem

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção
- Tem como desvantagem o fato de impactar na performance, por conta das verificações adicionais que não faziam parte da gramática original da linguagem
- Por fim, na estratégia de correção global o compilador tentar realizar o mínimo de alterações na entrada para que a mesma seja válida de acordo com a gramática da linguagem

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção
- Tem como desvantagem o fato de impactar na performance, por conta das verificações adicionais que não faziam parte da gramática original da linguagem
- Por fim, na estratégia de correção global o compilador tentar realizar o mínimo de alterações na entrada para que a mesma seja válida de acordo com a gramática da linguagem
- Esta técnica se baseia na heurística que grande parte dos erros pode ser corrigida com poucas modificações

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção
- Tem como desvantagem o fato de impactar na performance, por conta das verificações adicionais que não faziam parte da gramática original da linguagem
- Por fim, na estratégia de correção global o compilador tentar realizar o mínimo de alterações na entrada para que a mesma seja válida de acordo com a gramática da linguagem
- Esta técnica se baseia na heurística que grande parte dos erros pode ser corrigida com poucas modificações
- Tem grande custo de tempo e de memória

Estratégias de recuperação de erros

- Outra estratégia consiste nas produções de erro, as quais geram construções ilegais associadas a erros frequentes
- Ao inserir tais produções na gramática da linguagem, o analisador seria capaz de identificar tais erros com precisão e produzir diagnósticos precisos sobre o erro e sua eventual correção
- Tem como desvantagem o fato de impactar na performance, por conta das verificações adicionais que não faziam parte da gramática original da linguagem
- Por fim, na estratégia de correção global o compilador tentar realizar o mínimo de alterações na entrada para que a mesma seja válida de acordo com a gramática da linguagem
- Esta técnica se baseia na heurística que grande parte dos erros pode ser corrigida com poucas modificações
- Tem grande custo de tempo e de memória
- Além disso, o programa modificado pode não ser o que o programador desejava escrever

Gramática livre de contexto

Definição

Uma gramática livre de contexto é composta por terminais, não-terminais, um símbolo de partida e produções, onde

1. os terminais (tokens) são símbolos básicos para a formação de cadeias;
2. os não-terminais são variáveis sintáticas que identificam cadeias de tokens e que impõem uma estrutura hierárquica na linguagem;
3. um dentre os não-terminais é designado como símbolo de partida e o conjunto de cadeias geradas por ele é a linguagem definida pela gramática; e
4. as produção estabelecem as relações entre terminais e não-terminais e como novas cadeias podem ser formadas. Cada produção é composta por um não-terminal seguido de uma seta, a qual é sucedida por uma cadeia de terminais e não-terminais.

Convenções de notação

1. São terminais:
   - **(i)** Letras minúsculas do alfabeto (por exemplo, $a, b, c, \ldots$)
   - **(ii)** Simbolos de operadores (por exemplo, `+`, `-`, `×`, etc)
   - **(iii)** Símbolos de pontuação, parêntesis, vírgulas, etc
   - **(iv)** Os dígitos decimais `0`, `1`, `2`, ..., `9`
   - **(v)** Cadeias em negrito (por exemplo, **if**, **else**, **for**, etc)

Convenções de notação

1. São terminais:
   - **(i)** Letras minúsculas do alfabeto (por exemplo, $a, b, c, \ldots$)
   - **(ii)** Simbolos de operadores (por exemplo, `+`, `-`, `×`, etc)
   - **(iii)** Símbolos de pontuação, parêntesis, vírgulas, etc
   - **(iv)** Os dígitos decimais `0`, `1`, `2`, ..., `9`
   - **(v)** Cadeias em negrito (por exemplo, **if**, **else**, **for**, etc)
2. São não-terminais:
   - **(i)** Letras maiúsculas do início do alfabeto (por exemplo, $A, B, C, \ldots$)
   - **(ii)** A letra S, em geral indicado o símbolo de partida
   - **(iii)** Nomes em itálico formados por letras minúculas (por exemplo, cmd e $expr$)

Convenções de notação

1. São terminais:
   - **(i)** Letras minúsculas do alfabeto (por exemplo, $a, b, c, \ldots$)
   - **(ii)** Simbolos de operadores (por exemplo, `+`, `-`, `×`, etc)
   - **(iii)** Símbolos de pontuação, parêntesis, vírgulas, etc
   - **(iv)** Os dígitos decimais `0`, `1`, `2`, ..., `9`
   - **(v)** Cadeias em negrito (por exemplo, **if**, **else**, **for**, etc)
2. São não-terminais:
   - **(i)** Letras maiúsculas do início do alfabeto (por exemplo, $A, B, C, \ldots$)
   - **(ii)** A letra S, em geral indicado o símbolo de partida
   - **(iii)** Nomes em itálico formados por letras minúculas (por exemplo, cmd e $expr$)
3. Letras maiúsculas do final do alfabeto (em geral, X, Y, Z) representam símbolos gramaticais, isto é, terminais ou não-terminais

Convenções de notação

4. Letras minúsculas do fim do alfabeto (em geral, x, y, z) representam cadeias de terminais

Convenções de notação

4. Letras minúsculas do fim do alfabeto (em geral, x, y, z) representam cadeias de terminais
5. Letras gregas minúsculas (por exemplo, $\alpha, \beta, \gamma, \ldots$) representam cadeias de símbolos gramaticais (por exemplo, $A \to \alpha$ seria uma produção)

Convenções de notação

4. Letras minúsculas do fim do alfabeto (em geral, x, y, z) representam cadeias de terminais
5. Letras gregas minúsculas (por exemplo, $\alpha, \beta, \gamma, \ldots$) representam cadeias de símbolos gramaticais (por exemplo, $A \to \alpha$ seria uma produção)
6. Se $A \to \alpha_1, A \to \alpha_2, \ldots A \to \alpha_N$ são produções com A à esquerda (denominadas produções-A) então estas produções podem ser grafadas em uma só linha como

$$A \to \alpha_1 \mid \alpha_2 \mid \ \ldots \ \mid \alpha_N,$$

sendo cada α_i uma alternativa para A

Convenções de notação

4. Letras minúsculas do fim do alfabeto (em geral, x, y, z) representam cadeias de terminais
5. Letras gregas minúsculas (por exemplo, $\alpha, \beta, \gamma, \ldots$) representam cadeias de símbolos gramaticais (por exemplo, $A \to \alpha$ seria uma produção)
6. Se $A \to \alpha_1, A \to \alpha_2, \ldots A \to \alpha_N$ são produções com A à esquerda (denominadas produções-A) então estas produções podem ser grafadas em uma só linha como

$$A \to \alpha_1 \mid \alpha_2 \mid \ \ldots \ \mid \alpha_N,$$

   sendo cada α_i uma alternativa para A
7. O lado esquerdo da primeira produção é o símbolo de partida, salvo indicação contrária

Exemplo de gramática sem e com as convenções de notação

$$
\begin{aligned}
&expr \rightarrow expr\ op\ expr \\
&expr \rightarrow (expr) \\
&expr \rightarrow \text{-}\ expr \\
&expr \rightarrow \textbf{id} \\
&op \rightarrow\ \ + \\
&op \rightarrow\ \ - \\
&op \rightarrow\ \ \times \\
&op \rightarrow\ \ \div \\
&op \rightarrow \uparrow
\end{aligned}
$$

$$
\begin{aligned}
&E \rightarrow E\ A\ E \mid (E) \mid \text{-}\ E \mid \textbf{id} \\
&A \rightarrow + \mid - \mid \times \mid \div \mid \uparrow
\end{aligned}
$$

Derivações

Definição de derivação

Sejam E um não-terminal, $E \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_N$ produções-E e $\sigma = \beta\ E\ \gamma$. É dito que σ deriva $\beta\ \alpha_i\ \gamma$, e notamos $\sigma \Rightarrow \beta\ \alpha_i\ \gamma$, se a instância de E em σ é substituída por uma das alternativas α_i das produções-E.

Uma sequência de substituições em σ que resulte em X é chamada derivação de X a partir de σ.

Derivações em zero ou mais passos

Definição em zero ou mais passos

Se $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$, então α_1 deriva α_N. O símbolo $\Rightarrow$ significa "deriva em um passo". O símbolo $\overset{*}{\Rightarrow}$ significa "deriva em zero ou mais passos" e o símbolo $\overset{+}{\Rightarrow}$ significa "deriva em um ou mais passos".

A derivação em zero ou mais passos tem duas importantes propriedades:

1. $\alpha \overset{*}{\Rightarrow} \alpha$ para qualquer cadeia α, e
2. se $\alpha \overset{*}{\Rightarrow} \beta$ e $\beta \overset{*}{\Rightarrow} \gamma$, então $\alpha \overset{*}{\Rightarrow} \gamma$

Exemplo de derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$

$$
\begin{aligned}
E &\Rightarrow E \times E \\
&\Rightarrow - E \times E \\
&\Rightarrow - (E) \times E \\
&\Rightarrow - (E + E) \times E \\
&\Rightarrow - (E + E) \times \mathbf{id} \\
&\Rightarrow - (\mathbf{id} + E) \times \mathbf{id} \\
&\Rightarrow - (\mathbf{id} + \mathbf{id}) \times \mathbf{id}
\end{aligned}
$$

Linguagem gerada por G

Definição

Seja G uma gramática e S um símbolo de partida. O conjunto $L(G)$, denominado linguagem gerada por G, contém uma cadeia w se, e somente se, w contém apenas terminais e $S \overset{+}{\Rightarrow} w$. A cadeia w é denominada uma sentença de G. Uma linguagem que pode ser gerada por uma gramática é chamada linguagem livre de contexto.

Se duas gramáticas G_1 e G_2 geram a mesma linguagem, então G_1 e G_2 são ditas equivalentes.

Se $S \overset{*}{\Rightarrow} \alpha$, onde α pode conter tanto terminais quanto não-terminais, α é denominada uma forma sentencial de G.

Árvores gramaticais e derivações

- A ordem de substituição em uma derivação é arbitrária

Árvores gramaticais e derivações

- A ordem de substituição em uma derivação é arbitrária
- Convém, portanto, assumir uma ordem de substituição, sendo as mais comuns substuir sempre o não-terminal mais à esquerda (gerando a forma sentencial mais à esquerda) ou mais à direita (derivações canônicas)

Árvores gramaticais e derivações

- A ordem de substituição em uma derivação é arbitrária
- Convém, portanto, assumir uma ordem de substituição, sendo as mais comuns substuir sempre o não-terminal mais à esquerda (gerando a forma sentencial mais à esquerda) ou mais à direita (derivações canônicas)
- Uma árvore gramatical pode ser vista como uma representação gráfica de uma derivação com uma ordem específica de substituição

Árvores gramaticais e derivações

- A ordem de substituição em uma derivação é arbitrária
- Convém, portanto, assumir uma ordem de substituição, sendo as mais comuns substuir sempre o não-terminal mais à esquerda (gerando a forma sentencial mais à esquerda) ou mais à direita (derivações canônicas)
- Uma árvore gramatical pode ser vista como uma representação gráfica de uma derivação com uma ordem específica de substituição
- A ordem de substituição definirá o formato da árvore

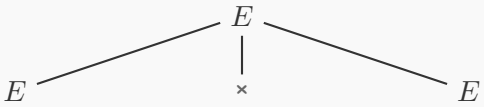
Árvores gramaticais e derivações

- A ordem de substituição em uma derivação é arbitrária
- Convém, portanto, assumir uma ordem de substituição, sendo as mais comuns substuir sempre o não-terminal mais à esquerda (gerando a forma sentencial mais à esquerda) ou mais à direita (derivações canônicas)
- Uma árvore gramatical pode ser vista como uma representação gráfica de uma derivação com uma ordem específica de substituição
- A ordem de substituição definirá o formato da árvore
- Uma gramática que produz mais de uma árvore gramatical para alguma sentença é dita ambígua

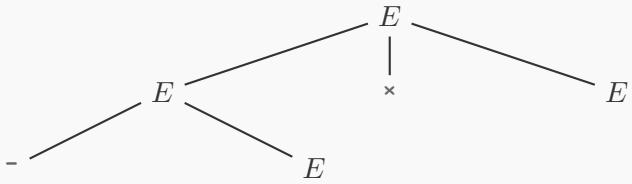
Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$

E

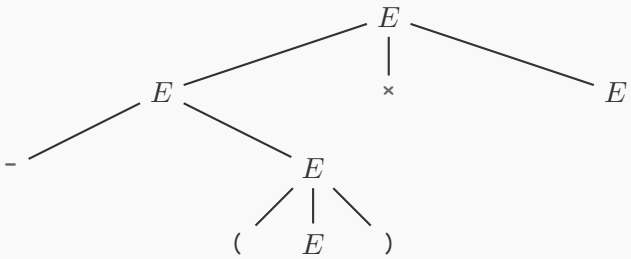
Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$



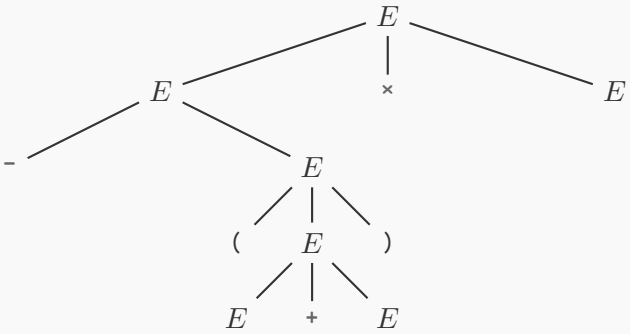
Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$



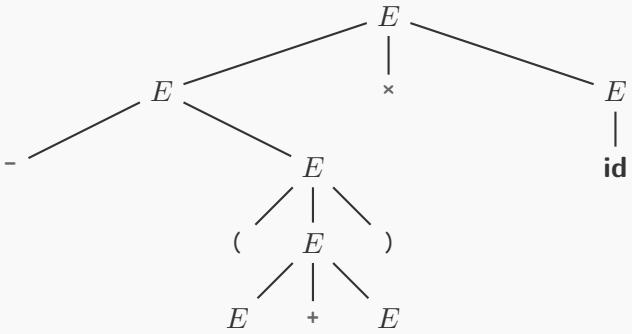
Árvore gramatical da derivação da expressão $-(\mathrm{id} + \mathrm{id}) \times \mathrm{id}$



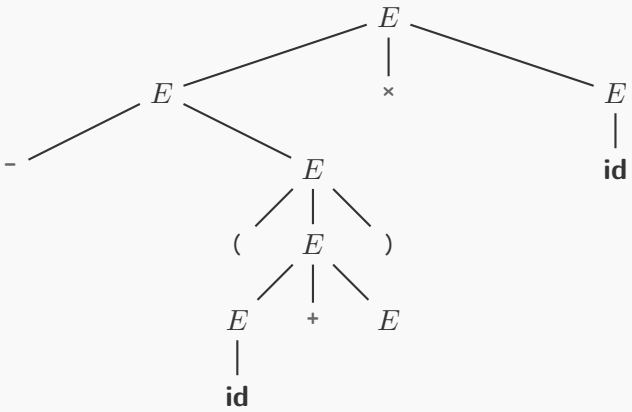
Árvore gramatical da derivação da expressão $-(\mathrm{id} + \mathrm{id}) \times \mathrm{id}$



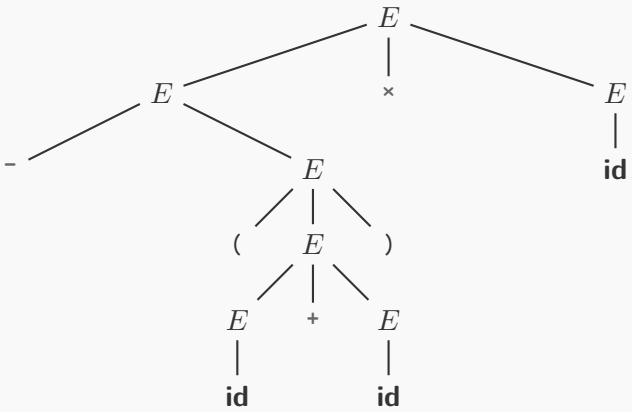
Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$



Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$



Árvore gramatical da derivação da expressão $-(\mathbf{id} + \mathbf{id}) \times \mathbf{id}$

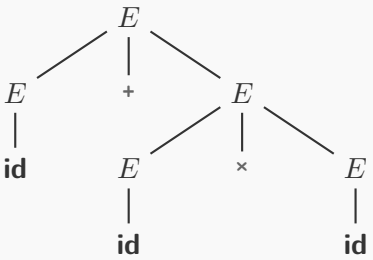


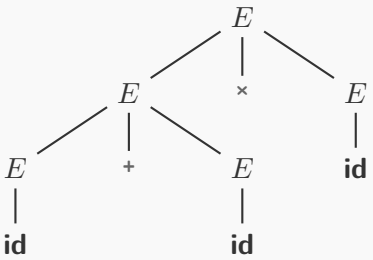
Duas derivações diferentes para a mesma expressão

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow \mathbf{id} + E \\
&\Rightarrow \mathbf{id} + E \times E \\
&\Rightarrow \mathbf{id} + \mathbf{id} \times E \\
&\Rightarrow \mathbf{id} + \mathbf{id} \times \mathbf{id}
\end{aligned}
$$

$$
\begin{aligned}
E &\Rightarrow E \times E \\
&\Rightarrow E + E \times E \\
&\Rightarrow \mathbf{id} + E \times E \\
&\Rightarrow \mathbf{id} + \mathbf{id} \times E \\
&\Rightarrow \mathbf{id} + \mathbf{id} \times \mathbf{id}
\end{aligned}
$$

Árvores sintáticas distintas que geram a mesma expressão





Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática

Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática
- A recíproca nem sempre é verdadeira

Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática
- A recíproca nem sempre é verdadeira
- Por exemplo, a expressão regular $(a \mid b)^* abb$ e a gramática

$$
\begin{aligned}
A_0 &\rightarrow aA_0 \mid bA_0 \mid aA_1 \\
A_1 &\rightarrow bA_2 \\
A_2 &\rightarrow bA_3 \\
A_3 &\rightarrow \epsilon
\end{aligned}
$$

descrevem a mesma linguagem

Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática
- A recíproca nem sempre é verdadeira
- Por exemplo, a expressão regular $(a \mid b)^*abb$ e a gramática

$$
\begin{aligned}
A_0 &\rightarrow aA_0 \mid bA_0 \mid aA_1 \\
A_1 &\rightarrow bA_2 \\
A_2 &\rightarrow bA_3 \\
A_3 &\rightarrow \epsilon
\end{aligned}
$$

  descrevem a mesma linguagem
- É possível converter automaticamente um autômatico finito não-determinístico em uma gramática que gere a mesma linguagem do AFN

Algoritmo de conversão de um AFN para uma gramática livre de contexto

Input: um AFN
Output: uma gramática livre de contexto

1: **for** cada estado i do AFN **do**
2: crie um símbolo não-terminal A_i da gramática
3: **if** o estado i possui um transição para o estado j com rótulo a **then**
4: introduza a produção $A_i \rightarrow aA_j$ na gramática
5: **else if** o estado i possui um transição para o estado j com rótulo ϵ **then**
6: introduza a produção $A_i \rightarrow A_j$ na gramática
7: **if** o estado i é um estado de aceitação **then**
8: introduza a produção $A_i \rightarrow \epsilon$ na gramática
9: **else if** o estado i é o estado de partida **then**
10: torne o estado A_i o símbolo de partida da gramática

Razões para o uso de expressões regulares para definir a estrutura léxica

1. As regras léxicas de uma linguagem geralmente são simples, sendo as expressões regulares suficientes para descrevê-las

Razões para o uso de expressões regulares para definir a estrutura léxica

1. As regras léxicas de uma linguagem geralmente são simples, sendo as expressões regulares suficientes para descrevê-las
2. As expressões regulares, em geral, descrevem os tokens da linguagem de forma mais concisa e clara do que as gramáticas livres de contexto

Razões para o uso de expressões regulares para definir a estrutura léxica

1. As regras léxicas de uma linguagem geralmente são simples, sendo as expressões regulares suficientes para descrevê-las
2. As expressões regulares, em geral, descrevem os tokens da linguagem de forma mais concisa e clara do que as gramáticas livres de contexto
3. É possível gerar analisadores léxicos mais eficientes a partir de expressões regulares do que a partir de gramáticas arbitrárias

Razões para o uso de expressões regulares para definir a estrutura léxica

1. As regras léxicas de uma linguagem geralmente são simples, sendo as expressões regulares suficientes para descrevê-las
2. As expressões regulares, em geral, descrevem os tokens da linguagem de forma mais concisa e clara do que as gramáticas livres de contexto
3. É possível gerar analisadores léxicos mais eficientes a partir de expressões regulares do que a partir de gramáticas arbitrárias
4. A separação da estrutura léxica da estrutura sintática permite a modularização da interface de vanguarda

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G
- Por exemplo, considere a gramática

$$S \rightarrow (S)S \mid \epsilon$$

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G
- Por exemplo, considere a gramática

$$S \rightarrow (S)S \mid \epsilon$$

- Esta gramática gera todas as cadeias de parêntesis balanceadas

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G
- Por exemplo, considere a gramática

$$S \rightarrow (S)S \mid \epsilon$$

- Esta gramática gera todas as cadeias de parêntesis balanceadas
- Para provar esta afirmação, primeiro é preciso provar que qualquer cada sentença derivável de S é uma cadeia de parêntesis balanceada

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G
- Por exemplo, considere a gramática

$$S \rightarrow (S)S \mid \epsilon$$

- Esta gramática gera todas as cadeias de parêntesis balanceadas
- Para provar esta afirmação, primeiro é preciso provar que qualquer cada sentença derivável de S é uma cadeia de parêntesis balanceada
- Esta prova é feita por indução no número de passos da derivação

Verificando a linguagem gerada por uma gramática

- Em apenas um passo de derivação, a única cadeia gerada é a cadeia vazia ϵ, a qual é trivialmente balanceada

Verificando a linguagem gerada por uma gramática

- Em apenas um passo de derivação, a única cadeia gerada é a cadeia vazia ϵ, a qual é trivialmente balanceada
- Suponha que qualquer derivação com menos do que n passos gere uma cadeia balanceada

Verificando a linguagem gerada por uma gramática

- Em apenas um passo de derivação, a única cadeia gerada é a cadeia vazia ϵ, a qual é trivialmente balanceada
- Suponha que qualquer derivação com menos do que n passos gere uma cadeia balanceada
- Uma derivação com exatamente n passos tem a forma

$$S \Rightarrow (S)S \overset{*}{\Rightarrow} (x)S \overset{*}{\Rightarrow} (x)y$$

onde x e y são cadeias obtidas por meios de derivações que totalizam exatamente $n-1$ passos

Verificando a linguagem gerada por uma gramática

- Em apenas um passo de derivação, a única cadeia gerada é a cadeia vazia ϵ, a qual é trivialmente balanceada
- Suponha que qualquer derivação com menos do que n passos gere uma cadeia balanceada
- Uma derivação com exatamente n passos tem a forma

$$S \Rightarrow (S)S \overset{*}{\Rightarrow} (x)S \overset{*}{\Rightarrow} (x)y$$

  onde x e y são cadeias obtidas por meios de derivações que totalizam exatamente $n-1$ passos
- Pela hipótese de indução, x e y são cadeias de parêntesis balanceadas e, portanto, a derivação S com exatamente n passos também é balanceada

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda
- Seja (x) o menor prefixo de w com o mesmo número de parêntesis à esquerda e à direita

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda
- Seja (x) o menor prefixo de w com o mesmo número de parêntesis à esquerda e à direita
- Assim, $w = (x)y$, onde x e y são cadeias balanceadas com comprimento menor do que $2n$

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda
- Seja (x) o menor prefixo de w com o mesmo número de parêntesis à esquerda e à direita
- Assim, $w = (x)y$, onde x e y são cadeias balanceadas com comprimento menor do que $2n$
- Pela hipótese de indução, x e y são deriváveis a partir de S

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \rightarrow \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda
- Seja (x) o menor prefixo de w com o mesmo número de parêntesis à esquerda e à direita
- Assim, $w = (x)y$, onde x e y são cadeias balanceadas com comprimento menor do que $2n$
- Pela hipótese de indução, x e y são deriváveis a partir de S
- Assim, w é derivável a partir de S por meio da derivação

$$S \Rightarrow (S)S \stackrel{*}{\Rightarrow} (x)S \stackrel{*}{\Rightarrow} (x)y$$

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades
- Por exemplo, considere a gramática abaixo, que torna o `else` opcional:

$$
\begin{array}{rcl}
cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \\
 & | & \textbf{if } expr \textbf{ then } cmd \textbf{else } cmd \\
 & | & \textbf{outro}
\end{array}
$$

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades
- Por exemplo, considere a gramática abaixo, que torna o `else` opcional:

$$
\begin{array}{rcl}
cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \\
 & | & \textbf{if } expr \textbf{ then } cmd \textbf{else } cmd \\
 & | & \textbf{outro}
\end{array}
$$

- Na gramática, **outro** significa qualquer outro enunciado

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades
- Por exemplo, considere a gramática abaixo, que torna o `else` opcional:

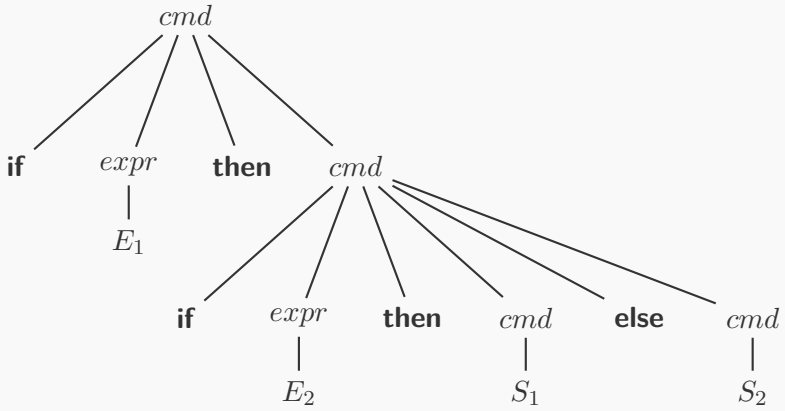
$$
\begin{array}{rcl}
cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \\
 & | & \textbf{if } expr \textbf{ then } cmd \textbf{else } cmd \\
 & | & \textbf{outro}
\end{array}
$$

- Na gramática, **outro** significa qualquer outro enunciado
- Esta gramática é ambígua: a cadeia

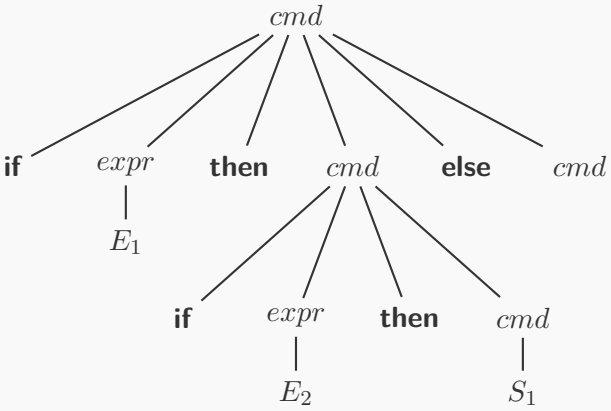
$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$

possui duas árvores gramaticais distintas

Primeira árvore gramatical para a expressão 'if E_1 then if E_2 then S_1 else S_2'



Segunda árvore gramatical para a expressão 'if E_1 then if E_2 then S_1 else S_2'



Reescrita para a eliminação da ambiguidade

- Na maioria das linguagens, a primeira das duas árvores seria a esperada

Reescrita para a eliminação da ambiguidade

- Na maioria das linguagens, a primeira das duas árvores seria a esperada
- A regra geral é associar cada **else** ao **then** anterior mais próximo ainda não associado

Reescrita para a eliminação da ambiguidade

- Na maioria das linguagens, a primeira das duas árvores seria a esperada
- A regra geral é associar cada **else** ao **then** anterior mais próximo ainda não associado
- Para reescrita, a ideia é que um enunciado entre um **then** e um **else** precisa estar associado, isto é, não pode terminar em um **then** não associado a um **else**

Reescrita para a eliminação da ambiguidade

- Na maioria das linguagens, a primeira das duas árvores seria a esperada
- A regra geral é associar cada **else** ao **then** anterior mais próximo ainda não associado
- Para reescrita, a ideia é que um enunciado entre um **then** e um **else** precisa estar associado, isto é, não pode terminar em um **then** não associado a um **else**

$$
\begin{array}{rcl}
cmd & \rightarrow & cmd\_associado \\
 & | & cmd\_nao\_associado \\
cmd\_associado & \rightarrow & \textbf{if } expr \textbf{ then } cmd\_associado \textbf{ else } cmd\_associado \\
 & | & \textbf{outro} \\
cmd\_nao\_associado & \rightarrow & \textbf{if } expr \textbf{ then } cmd \\
 & | & \textbf{if } expr \textbf{ then } cmd\_associado \textbf{ else } cmd\_nao\_associado
\end{array}
$$

Gramáticas recursivas à esquerda

- Uma gramática é recursiva à esquerda se possui um não-terminal A tal que existe um derivação $A \stackrel{+}{\Rightarrow} A\alpha$ para alguma cadeia α

Gramáticas recursivas à esquerda

- Uma gramática é recursiva à esquerda se possui um não-terminal A tal que existe um derivação $A \stackrel{+}{\Rightarrow} A\alpha$ para alguma cadeia α
- Métodos *top-down* não podem processar gramáticas recursivas à esquerda, demandando uma reescrita da gramática que elimine a recursão à esquerda

Gramáticas recursivas à esquerda

- Uma gramática é recursiva à esquerda se possui um não-terminal A tal que existe um derivação $A \stackrel{+}{\Rightarrow} A\alpha$ para alguma cadeia α
- Métodos *top-down* não podem processar gramáticas recursivas à esquerda, demandando uma reescrita da gramática que elimine a recursão à esquerda
- Uma recursão simples à esquerda acontece se existe um produção $A \rightarrow A\alpha$

Gramáticas recursivas à esquerda

- Uma gramática é recursiva à esquerda se possui um não-terminal A tal que existe um derivação $A \stackrel{+}{\Rightarrow} A\alpha$ para alguma cadeia α
- Métodos *top-down* não podem processar gramáticas recursivas à esquerda, demandando uma reescrita da gramática que elimine a recursão à esquerda
- Uma recursão simples à esquerda acontece se existe um produção $A \rightarrow A\alpha$
- A recursão simples à esquerda de uma produção $A \rightarrow A\alpha \mid \beta$ pode ser eliminada ao substituí-la pelas produções

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \epsilon$$

Exemplo de eliminação de recursão simples à esquerda

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T \times F \mid F \\
F &\to (E) \mid \textbf{id}
\end{aligned}
$$

$$
\begin{aligned}
E &\to TE' \\
E' &\to + TE' \mid \epsilon \\
T &\to FT' \\
T' &\to \times FT' \mid \epsilon \\
F &\to (E) \mid \textbf{id}
\end{aligned}
$$

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez
- Primeiramente, organize todas as produções-A na forma

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

onde nenhum β_j começa com um A

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez
- Primeiramente, organize todas as produções-A na forma

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

  onde nenhum β_j começa com um A
- Em seguida, substitua estas produções-A pelas produções

$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$$

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez
- Primeiramente, organize todas as produções-A na forma

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

  onde nenhum β_j começa com um A
- Em seguida, substitua estas produções-A pelas produções

$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$$

- Esta substituição elimina todas as recursões simples à esquerda de uma só vez, desde que $\alpha_i \neq \epsilon$ para todo $i = 1, 2, \ldots, m$

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez
- Primeiramente, organize todas as produções-A na forma

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

  onde nenhum β_j começa com um A
- Em seguida, substitua estas produções-A pelas produções

$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$$

- Esta substituição elimina todas as recursões simples à esquerda de uma só vez, desde que $\alpha_i \neq \epsilon$ para todo $i = 1, 2, \ldots, m$
- Esta técnica, porém, não elimina recursões à esquerda envolvendo derivações com dois ou mais passos

Algoritmo para eliminação de recursão à esquerda

Input: Uma gramática G sem ciclos (isto é, produções $A \stackrel{+}{\Rightarrow} A$) e sem produções-$\epsilon$ (do tipo $A \to \epsilon$)

Output: Uma gramática equivalente a G sem recursão à esquerda

1: Liste, em alguma ordem, os não-terminais $A_1, A_2, \ldots, A_n$
2: **for** $i \leftarrow 1, n$ **do**
3: **for** $j \leftarrow 1, i-1$ **do**
4: substitua cada produção $A_i \to A_j\gamma$ pelas produções

$$A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ \ldots \ \mid \delta_k\gamma,$$

onde $A_j \to \delta_1 \mid \delta_2 \mid \ \ldots \ \mid \delta_k$ são todas as produções-A_j atuais

5: elimine todas as recursões simples à esquerda nas produções-A_i

Exemplo de eliminação de recursão à esquerda

- Considere a gramática

$$S \rightarrow Aa \mid b$$
$$A \rightarrow Ac \mid Sd \mid \epsilon$$

Exemplo de eliminação de recursão à esquerda

- Considere a gramática

$$S \to Aa \mid b$$
$$A \to Ac \mid Sd \mid \epsilon$$

- Observe que o não-terminal S é recursivo à esquerda, pois

$$S \Rightarrow Aa \Rightarrow Sda$$

Exemplo de eliminação de recursão à esquerda

- Considere a gramática

$$S \to Aa \mid b$$
$$A \to Ac \mid Sd \mid \epsilon$$

- Observe que o não-terminal S é recursivo à esquerda, pois

$$S \Rightarrow Aa \Rightarrow Sda$$

- A aplicação do algoritmo de eliminação de recursão poderia não funcionar, por conta da produção-ϵ do não-terminal A, mas neste caso em particular o algoritmo de fato elimina a recursão

Exemplo de eliminação de recursão à esquerda

- Usando a ordenação S, A, a primeiro iteração do algoritmo não altera a gramática, uma vez que S não tem recursão simples à esquerda

Exemplo de eliminação de recursão à esquerda

- Usando a ordenação S, A, a primeiro iteração do algoritmo não altera a gramática, uma vez que S não tem recursão simples à esquerda
- Na segunda iteração, as produções-A que envolvem S devem ser substituídas, obtendo

$$A \rightarrow Ac \mid Aad \mid bd \mid \epsilon$$

Exemplo de eliminação de recursão à esquerda

- Usando a ordenação S, A, a primeiro iteração do algoritmo não altera a gramática, uma vez que S não tem recursão simples à esquerda
- Na segunda iteração, as produções-A que envolvem S devem ser substituídas, obtendo

$$A \rightarrow Ac \mid Aad \mid bd \mid \epsilon$$

- A eliminação da recursão simples à esquerda nas produções-A resulta na gramática livre de recursão à esquerda

$$\begin{aligned} S &\rightarrow Aa \mid b \\ A &\rightarrow bdA' \mid A' \\ A' &\rightarrow cA' \mid adA' \mid \epsilon \end{aligned}$$

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva
- A ideia central da fatoração à esquerda é evitar ambiguidades, quando duas ou mais produções tenham prefixos comuns

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva
- A ideia central da fatoração à esquerda é evitar ambiguidades, quando duas ou mais produções tenham prefixos comuns
- Por exemplo, considere as produções $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva
- A ideia central da fatoração à esquerda é evitar ambiguidades, quando duas ou mais produções tenham prefixos comuns
- Por exemplo, considere as produções $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
- Se a entrada contém uma cadeia não-vazia derivada a partir de α, não é possível decidir, de antemão, qual das duas produções usar

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva
- A ideia central da fatoração à esquerda é evitar ambiguidades, quando duas ou mais produções tenham prefixos comuns
- Por exemplo, considere as produções $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
- Se a entrada contém uma cadeia não-vazia derivada a partir de α, não é possível decidir, de antemão, qual das duas produções usar
- A fatoração à esquerda propõe a reescrita das produções da seguinte forma, que elimina a ambiguidade

$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta_1 \mid \beta_2$$

Exemplo de fatoração à esquerda

- Muitas linguagens permitem que o comando **if-then-else** tenha um **else** vazio:

$$\begin{array}{rcl} cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \textbf{ else } cmd \\ & | & \textbf{if } expr \textbf{ then } cmd \end{array}$$

Exemplo de fatoração à esquerda

- Muitas linguagens permitem que o comando **if-then-else** tenha um **else** vazio:

$$\begin{array}{rcl} cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \textbf{ else } cmd \\ & | & \textbf{if } expr \textbf{ then } cmd \end{array}$$

- Esta gramática pode ser abstraída da seguinte forma

$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Exemplo de fatoração à esquerda

- Muitas linguagens permitem que o comando **if-then-else** tenha um **else** vazio:

$$\begin{array}{rcl} cmd & \rightarrow & \textbf{if } expr \textbf{ then } cmd \textbf{ else } cmd \\ & | & \textbf{if } expr \textbf{ then } cmd \end{array}$$

- Esta gramática pode ser abstraída da seguinte forma

$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

- A fatoração à esquerda esta gramática resulta em

$$S \rightarrow iEtSS' \mid a$$
$$S' \rightarrow eS \mid \in$$
$$E \rightarrow b$$

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

- As sentenças de L_1 são formada por uma cadeia w de a's e b's, seguida de um c e repetida em seguida (por exemplo, $abaabcabaab$)

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

- As sentenças de L_1 são formada por uma cadeia w de a's e b's, seguida de um c e repetida em seguida (por exemplo, $abaabcabaab$)
- É possível demostrar que a linguagem L_1 não é livre de contexto

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

- As sentenças de L_1 são formada por uma cadeia w de a's e b's, seguida de um c e repetida em seguida (por exemplo, $abaabcabaab$)
- É possível demostrar que a linguagem L_1 não é livre de contexto
- Tal linguagem abstrai o problema de se verificar se um identificador (w) foi declarado antes de seu uso (c é o que separa a declaração do primeiro uso)

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

- As sentenças de L_1 são formada por uma cadeia w de a's e b's, seguida de um c e repetida em seguida (por exemplo, $abaabcabaab$)
- É possível demostrar que a linguagem L_1 não é livre de contexto
- Tal linguagem abstrai o problema de se verificar se um identificador (w) foi declarado antes de seu uso (c é o que separa a declaração do primeiro uso)
- Não sendo possível definir tal regra sintaticamente, esta verificação fica postergada para a análise semântica

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \text{ e } m \geq 1\}$$

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \text{ e } m \geq 1\}$$

- As senteças de L_2 são cadeias onde o número de a's coincide com o número de c's, e o número de b's coincide com o número de d's, em ordem lexicográfica

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \text{ e } m \geq 1\}$$

- As senteças de L_2 são cadeias onde o número de a's coincide com o número de c's, e o número de b's coincide com o número de d's, em ordem lexicográfica
- L_2 também não é livre de contexto

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \text{ e } m \geq 1\}$$

- As senteças de L_2 são cadeias onde o número de a's coincide com o número de c's, e o número de b's coincide com o número de d's, em ordem lexicográfica
- L_2 também não é livre de contexto
- Ela abstrai o problema de ser verificar se o número de parâmetros na declaração de uma função é igual ao número de parâmetros na chamada (a^n e b^m seriam as listas de parâmetros de dois procedimentos e c^n e d^n o número de parâmetros na chamada destes procedimentos)

Exemplos de linguagens não livres de contexto

- A linguagem

$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$

também não é livre de contexto

Exemplos de linguagens não livres de contexto

- A linguagem

$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$

  também não é livre de contexto
- A diferença entre uma linguagem livre de uma não-livre pode ser sutil

Exemplos de linguagens não livres de contexto

- A linguagem
$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$
também não é livre de contexto
- A diferença entre uma linguagem livre de uma não-livre pode ser sutil
- Por exemplo, a linguagem
$$L_1' = \{wcw^R \mid w \in (a \mid b)^*\},$$
onde w^R é o reverso de w, é livre de contexto

Exemplos de linguagens não livres de contexto

- A linguagem
$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$
também não é livre de contexto
- A diferença entre uma linguagem livre de uma não-livre pode ser sutil
- Por exemplo, a linguagem
$$L_1' = \{wcw^R \mid w \in (a \mid b)^*\},$$
onde w^R é o reverso de w, é livre de contexto
- Ela pode ser gerada pela gramática $S \rightarrow aSa \mid bSb \mid \epsilon$

Exemplos de linguagens não livres de contexto

- A linguagem

$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1 \text{ e } m \geq 1\}$$

também é livre de contexto

Exemplos de linguagens não livres de contexto

- A linguagem

$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1 \text{ e } m \geq 1\}$$

  também é livre de contexto
- A gramática abaixo gerada L_2':

$$S \rightarrow aSd \mid aAd$$
$$A \rightarrow bAc \mid bc$$

Exemplos de linguagens não livres de contexto

- A linguagem
$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto
- A gramática abaixo gerada L_2':
$$S \to aSd \mid aAd$$
$$A \to bAc \mid bc$$

- A linguagem
$$L_2'' = \{a^n b^n c^m d^m \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto

Exemplos de linguagens não livres de contexto

- A linguagem
$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto
- A gramática abaixo gerada L_2':
$$S \to aSd \mid aAd$$
$$A \to bAc \mid bc$$

- A linguagem
$$L_2'' = \{a^n b^n c^m d^m \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto

Exemplos de linguagens não livres de contexto

- Ela pode ser gerada pela gramática

$$
\begin{aligned}
S &\to AB \\
A &\to aAb \mid ab \\
B &\to cBd \mid cd
\end{aligned}
$$

Exemplos de linguagens não livres de contexto

- Ela pode ser gerada pela gramática

$$S \to AB$$
$$A \to aAb \mid ab$$
$$B \to cBd \mid cd$$

- Por fim, a linguagem

$$L_3' = \{a^n b^n \mid n \geq 1\}$$

é livre de contexto, gerada pela gramática $S \to aSb \mid ab$

Exemplos de linguagens não livres de contexto

- Ela pode ser gerada pela gramática

$$S \to AB$$
$$A \to aAb \mid ab$$
$$B \to cBd \mid cd$$

- Por fim, a linguagem

$$L'_3 = \{a^n b^n \mid n \geq 1\}$$

é livre de contexto, gerada pela gramática $S \to aSb \mid ab$

- Informalmente, pode-se afirmar que um autômato finito não pode realizar contagens e que uma gramática pode manter uma contagem de dois itens, mas não de três

Referências

1. **AHO**, Alfred V, **SETHI**, Ravi, **ULLMAN**, Jeffrey D. *Compiladores: Princípios, Técnicas e Ferramentas*, LTC Editora, 1995.