Análise sintática

O papel do analisador sintático

Prof. Edson Alves

Faculdade UnB Gama

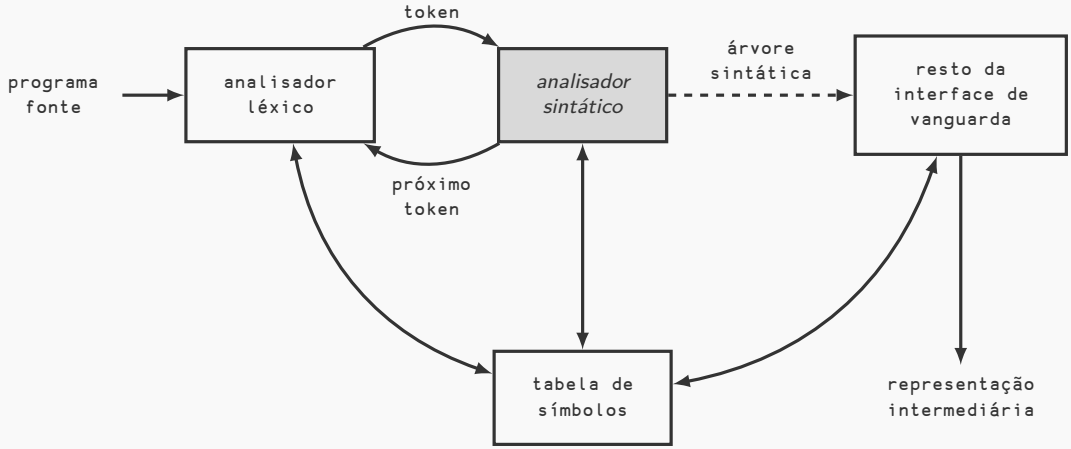
Gramáticas no projeto de linguagens e na escrita de compiladores

- 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
- 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
- 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:

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:

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
- 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
- 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
- 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

Referências

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