Trabalho Final de Compiladores

Rian Beskow Friedrich

1 Análise Léxica:

1.1 O que é:

A análise léxica é a primeira fase de um compilador. Ela lê o código-fonte caractere por caractere e transforma tudo em tokens, que são unidades significativas (palavras-chave, identificadores, números, operadores etc.).

1.2 Input:

O input da análise léxica é o código-fonte bruto, como texto (string).

1.3 Output:

O output da análise léxica é uma sequência de tokens, cada token com tipo (ex.: IDENTIFIER, NUMBER, IF, ASSIGN) e valor.

2 Análise Sintática:

2.1 O que é:

A análise Sintática é a parte que recebe a sequência de tokens e monta uma estrutura chamada árvore sintática (AST – Abstract Syntax Tree), verificando se a sequência obedece às regras gramaticais da linguagem.

2.2 Input:

O input da análise sintática é a sequência de tokens produzida pela análise léxica.

2.3 Output:

O output da análise sintática é uma árvore sintática abstrata (AST) ou uma árvore de derivação, representando a estrutura do programa. Se houver erro de sintaxe, ele é detectado aqui.

3 Análise Semântica

3.1 O que é:

A análise semântica é a parte que verifica se o programa faz sentido de acordo com as regras da linguagem. Como por exemplo: Tipos corretos (ex.: não somar string com inteiro), variáveis declaradas antes do uso, regras de escopo, funções chamadas com argumentos válidos e entre diversos outros.

3.2 Input:

O input da análise semântica é a AST produzida pela análise sintática e tabelas de símbolos (variáveis, funções, tipos).

3.3 Output:

O output da análise semântica é a AST enriquecida com informações semânticas (tipos, escopos, referências), uma tabela de símbolos completa ou erros semânticos, caso existam.

4 Geração de Código

4.1 O que é:

A geração de código é o processo que na qual transforma a AST (já verificada) em código de máquina, bytecode ou código intermediário (como LLVM IR ou código de 3 endereços), dependendo do compilador.

4.2 Input

O input da geração de código é a AST anotada (como resultado da análise semântica).

4.3 Output

O output da geração de código é um código executável, bytecode ou código intermediário (IR). Em compiladores reais, ainda há etapas como otimização antes do código final.

5 Construindo um AFD que reconheça os números hexadecimais:

q0,q_hex,q_dead

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,a,b,c,d,e,f

q0

q_hex:HEX

q0:0:q_hex, q0:1:q_hex, q0:2:q_hex, q0:3:q_hex, q0:4:q_hex, q0:5:q_hex, q0:6:q_hex, q0:7:q_hex, q0:8:q_hex, q0:9:q_hex, q0:A:q_hex, q0:B:q_hex, q0:C:q_hex, q0:D:q_hex, q0:E:q_hex, q0:F:q_hex, q0:a:q_hex, q0:b:q_hex, q0:c:q_hex, q0:d:q_hex, q0:e:q_hex, q0:f:q_hex

q0:*:q_dead

q_hex:0:q_hex, q_hex:1:q_hex, q_hex:2:q_hex, q_hex:3:q_hex, q_hex:4:q_hex, q_hex:5:q_hex, q_hex:6:q_hex, q_hex:7:q_hex, q_hex:8:q_hex, q_hex:9:q_hex, q_hex:A:q_hex, q_hex:B:q_hex, q_hex:C:q_hex, q_hex:D:q_hex, q_hex:E:q_hex, q_hex:F:q_hex, q_hex:a:q_hex, q_hex:b:q_hex, q_hex:c:q_hex, q_hex:d:q_hex, q_hex:e:q_hex, q_hex:f:q_hex

q_hex:*:q_dead

q_dead:*:q_dead

5.1 Explicação do Funcionamento do AFD:

O AFD começa em q0.

Se o primeiro caractere for um dígito hexadecimal (0–9, A–F, a–f), ele vai para q_hex, que é estado de aceitação.

Em q_hex, cada novo dígito hexadecimal mantém o autômato em q_hex. Qualquer símbolo que não seja hexadecimal leva ao estado q_dead, do qual não se sai. A string é aceita se terminar em q_hex.

5.1.1 O que significa o * no AFD?

No AFD, o asterisco * significa "qualquer símbolo que não foi especificado nas transições acima". Quando é escrito q0:*:q_dead, isso quer dizer: "Se o autômato estiver em q0 e ler qualquer caractere que não seja um dígito/hexadecimal listado, vá para q_dead."

É uma forma curta de representar todas as entradas restantes sem precisar listar caractere por caractere. Ele funciona como um coringa, equivalente a "todos os símbolos possíveis do alfabeto exceto os já mencionados nas regras anteriores daquele estado".

5.1.2 Estado Final no AFD:

O estado final do AFD é o q_hex. Ele reconhece números hexadecimais. Sempre que a entrada for composta só por caracteres válidos de hexadecimal (0–9, A–F, a–f), o autômato permanece em q_hex e aceita a cadeia ao final. Qualquer símbolo diferente leva ao estado morto (q_dead), que rejeita a entrada. Portanto, a cadeia só é reconhecida como hexadecimal se terminar em q_hex.

6 O que pode ser implementado antes da análise léxica?

Antes da análise léxica podem ser realizadas etapas de pré-processamento, como remoção de comentários, expansão de macros, inclusão de arquivos externos e normalização do código-fonte (como conversão de caracteres ou padronização de quebras de linha). Essas ações preparam o código para que o analisador léxico trabalhe sobre uma entrada mais limpa e consistente.

7 O que faz o comando LOAD em compiladores?

O comando LOAD é usado pelo compilador ou pelo código gerado para carregar um valor da memória para um registrador. Ele transfere dados de um endereço específico para um registrador da máquina, permitindo que instruções posteriores manipulem esse valor diretamente no processador. É uma instrução fundamental para mover dados entre memória e CPU.