Construção de Compiladores I [BCC328]

Departamento de Computação
Universidade Federal de Ouro Preto
Mateus Vitor Pereira Lana
Thiago Oliveira de Santana
12 de dezembro de 2018

Resumo

O interpretador semântico desenvolvido neste trabalho está baseado em um analisador sintático ascendente implementado em conjunto com um analisador léxico para uma linguagem denominada *Picnic*, disponibilizado pelo professor. Ao longo deste documento temos uma breve descrição de cada uma das categorias aqui mencionadas.

Sumário

1 A linguagem *Picnic*

A linguagem que este analisador interpreta é embasada na gramática "Grammar 4.1 Example language for interpretation" do livro "Introduction to Compiler Design" do Torben. A linguagem é composta basicamente por três sinais de pontuação, dois operadores, literais inteiros, sete palavras reservadas e identificadores. Além dessas características, foram adicionados a linguagem outras particularidades gerando uma linguagem estendida incluindo:

- comentários de linha,
- comentários de bloco,
- literais booleanos,
- o literal text,
- literais reais,
- o literal void,
- operadores aritméticos: `-` (subtração), `*` (multiplicação), `/` (divisão), `+` (adição), `^` (potenciação),
- operadores relacionais: `=` (diferente), `>` (maior que), `>=` (maiour ou igual a), `<` (menor que), e `<=` (menor ou igual a),
- operadores lógicos: `&&` (e lógico), e `||` (ou lógico).
- atribuição: `:=`,
- expressão de repetição: while,
- expressão sequência,
- expressão parentetizada, e
- funções com lista de argumentos vazia.

Apresentamos a seguir uma gramática livre de contexto (com comentários) para *Picnic*, que define a sintaxe de todas as construções da linguagem.

Program → *Funs*		**programa**
Funs → *Fun*		
Funs → *Fun* *Funs*		
Fun → *TypeId* `(` *TypeIds* `)` `=` *Exp*		**declaração de função**
TypeId → `bool` `id`		**tipo booleano**
TypeId → `int` `id`		**tipo inteiro**
TypeId → `string` `id`		**tipo string**
TypeIds →		**lista de parâmetros**
TypeIds → *TypeId* `,` *TypeIds*		
Exp → `litbool`		**literais**
Exp → `litint`		
Exp → `littext`		
Exp → `litreal`		
Exp → `litvoid`		
Exp → `id`		**variável**
Exp → `id` `:=` *Exp*		**atribuição**
Exp → *Exp* `+` *Exp*		**operações aritméticas**
Exp → *Exp* `-` *Exp*		
Exp → *Exp* `*` *Exp*		
Exp → *Exp* `/` *Exp*		
Exp → *Exp* `^` *Exp*		
Exp → *Exp* `=` *Exp*		**operações relacionais**
Exp → *Exp* `~=` *Exp*		
Exp → *Exp* `>` *Exp*		
Exp → *Exp* `>=` *Exp*		
Exp → *Exp* `<` *Exp*		
Exp → *Exp* `<=` *Exp*		
Exp → *Exp* `&&` *Exp*		**operações lógicas**
Exp → *Exp* `||` *Exp*		
Exp → `id` `(` *Exps* `)`		**chamada de função**
Exp → `if` *Exp* `then` *Exp* `else` *Exp*		**expressão condicional**
Exp → `while` *Exp* `do` *Exp*		**expressão de repetição**
Exp → `let` `id` `=` *Exp* `in` *Exp*		**expressão de declaração**
Exp → `(` *Exps* `)`		**expressão sequência**
Exps →		
Exps → *Exp* `,` *Exps*		

A precedência relativa e a associatividade dos operadores é indicada pela tabela a seguir, em ordem decrescente de precedência.

operadores	**associatividade**
`-` (unário)	
`*`, `/`, `^`	esquerda
`+`, `-` (binário)	esquerda
`=`, `<>`, `>`, `>=`, `<`, `<=`	
`&&`	esquerda
`||`	esquerda
`:=`	direita
`then`, `else`, `do`, `in`	direita

Observe que um programa em *Picnic* é uma sequência de declarações de funções.

Um programa deve definir uma função sem argumentos chamada `main` que resulta em um inteiro. A execução do programa inicia-se pela chamada desta função `main`.

2 Aspectos léxicos

Comentários de linha em *Picnic* começam com o caracter `#` e se estendem até o final da linha. **Comentários de bloco** são delimitados pelas sequências de caracteres `{#` e `#}` e podem ser aninhados.

Ocorrências de **caracteres brancos** (espaço, tabulação horizontal e nova linha) e comentários entre os símbolos léxicos são ignoradas, servindo apenas para separar símbolos léxicos.

Os **literais inteiros** são formados por uma sequência de um ou mais dígitos decimais.

Os **literais reais** são formados por uma sequência de um ou mais dígitos decimais seguida do símbolo `.`, seguido de uma sequência de um ou mais dígitos decimais. Uma e somente uma das duas sequências de dígitos é opcional.

Os **literais booleanos** são `true` (verdadeiro) e `false` (falso).

Os **literais text** são formados por uma sequência de caracteres gráficos delimitada por aspas (`"`). Na sequência de caracteres o caracter `\` é especial e inicia uma sequência de escape. As únicas sequências de escape válidas são indicados na tabela a seguir.

sequência de escape	**descrição**
`\\`	`\`
`\"`	`"`
`\t`	tabuação horizontal
`\n`	nova linha
`\r`	retorno de carro
`\f`	avanço de formulário
`\b`	backspace
`\`*ddd*	caracter de código *ddd*, sendo *d* qualquer dígito decimal

Identificadores são sequências de letras maiúsculas ou minúsculas, dígitos decimais e sublinhados (_), começando com uma letra. Letras maiúsculas e minúsculas são distintas em um identificador.

Os sinais de pontuação presentes na linguagem são abre parênteses, fecha parênteses e a vírgula, ou seja "(" , ")" e ",". Quando estes sinais são lidos em uma entrada classificamos os mesmos como os tokens LPAREN, RPAREN e COMMA respectivamente.

As palavras reservadas da linguagem são bool, int, if, then, else, let, in, while, do, void, real e text; que quando são identificadas em uma entrada são catalogadas como os tokens BOOL, INT, IF, THEN, ELSE, LET, IN, WHILE, DO, VOID, REAL e TEXT respectivamente.

3 Aspectos Sintáticos

Como já foi dito anteriormente um prgrama escrito na linguagem *Picnic* deve ser composto por uma sequencia de funções e e conter uma função principal denominada *main*.

A declaração de tipos é feita colocando-se o nome referente ao tipo que se deseja declarar e em seguida um indentificador. Como por exemplo: int x.

As expressões são representadas introduzindo uma expressão seguida do operador(aritmético, relacional ou lógico) e outra expressão na sequência. Com excessão da expressão de atribuição que é composta por um identificador seguido do operador de atribuição e uma expressão na sequência. Além disso, as expressões podem ser parentetizadas, ou seja, delimitadas por parentêses. Exemplo: (x+1) + 2+3.

O comando if é constituido da seguinte forma, a palavra if, seguida de uma expressão, seguida da palavra then, seguida de uma expressão, seguida da palavra else e uma expressão. Exemplo: if (x > 2) then x = x+2 else x=x+1.

O comando while é constituido da seguinte forma, a palavra while, seguida de uma expressão, seguida da palavra do, seguida de uma expressão. Exemplo: while (x < 10) do x = x+1.

O comando let é constituido da seguinte forma, a palavra let, seguida de um identificador, seguida do operador igual, seguida de uma expressão, seguida da palavra in e uma expressão. Exemplo: let x = 2 in y.

A declaração de funções é feita colocando-se em sequência: o tipo da função, o nome(id), abre parêteses, uma lista de 0 ou mais declarações de tipos separadas por vírgula, fecha parênteses, um

símbolo de igual e uma expressão. Podemos exemplificar da seguinte maneira: int sumfunction(int a, int b) = a+b.

A chamada da função é feita colocando-se em sequência: o nome da função, seguido de abre parênteses, em seguida a lista de argumentos(Exps), logo depois temos o fecha parênteses. Exemplo: sumfunction(a, b).

4 Aspectos Semânticos

Após a construção do analisador léxico e do analisador sintático descendente o próximo passo foi a elaboração do analisador semântico.

O analisador semântico implementado neste trabalho, utiliza basicamente as árvores sintáticas e tabelas de símbolos para fazer as análises semânticas.

A análise semântica é responsavel por verificar aspectos relacionados ao significado das instruções, essa é a terceira etapa do processo de compilação e nesse momento ocorre a validação de uma série de regras que não podem ser verificadas nas etapas léxicas e sintáticas.

As validações que não podem ser executadas pelas etapas mencionadas anteriormente, devem ser executadas durante a análise semântica a fim de garantir que o programa fonte esteja coerente e o mesmo possa ser convertido para linguagem de máquina.

A análise semântica percorre a árvore sintática e relaciona os identificadores com seus dependentes de acordo com a estrutura hierárquica. Sendo assim, descreve o significado das expressões, comandos e unidades de programa e define como as construções da linguagem devem ser interpretadas e executadas.

Geralmente os compiladores das linguagens de programação existentes no dia de hoje, fazem as seguintes verificações semânticas:

- **Análise de escopo**
  1. Variáveis não declaradas
  2. Múltiplas declarações de uma mesma variável
- **Compatibilidade de tipos**
- **Coerência entre declaração e uso de identificadores**
- **Correlação entre parâmetros formais e atuais**
- **Referências não resolvidas**
  1. Procedimentos e desvios

Sabemos que a descrição de uma linguagem de programação envolve dois aspectos principais: sintaxe e semântica. Frequentemente esses dois conceitos são confundidos, mas como já foi dito anteriormente a semântica descreve o significado das expressões, comandos e unidades de programa diferentemente da sintaxe que descreve a forma ou estrutura de expressões, comandos e unidades de programa. A seguir mostraremos um exemplo na linguagem *Picnic*,mostrando a diferença entre esses dois conceitos.

- **Exemplo – Comando condicional IF**
  1. Sintaxe: **if** (<expressão>) **then** <instrução> **else** <instrução>
  2. Semântica: se o valor atual da expressão for verdadeiro, uma instrução incorporada será selecionada para execução, caso for falso, uma outra instrução será selecionada para execução.

Na análise semântica realizada na linguagem *Picnic* é feita a verificação do significado dos tipos primitivos criados para avaliar se as instruções do programa escrito constituem expressões válidas semanticamente falando. Nesta perspectiva, o analisador semântico implementado ao verificar se as expressões criadas em um programa fonte na linguagem *Picnic* estão coerentes, ou seja, são expressões que pertencem a linguagem, faz com que o programa seja convertido para linguagem de máquina a fim de que o computador possa realizar as instruções contidas no mesmo.