Analisador Semântico

João Victor Cabral de Melo[1]

Universidade de Brasília

Abstract. This project is to show what we have done and learned in the course of compilers and the grammar of experimental language C-IPL.

Keywords: Compilers · Grammar · Lexical Analyser · Flex · Syntax Analyser · Bison

1 Introdução

1.1 Motivação

A maior motivação para a implementação de um compilador é conseguir gerar código executável em máquinas, dada uma linguagem de alto nível como entrada.

1.2 Nova Primitiva

A motivação para entrada da nova primitiva *list* é a possibilidade de resolver problemas onde se precisa de uma estrutura de dados com um tamanho não definido previamente. Facilitando esse conjunto específico de problemas pela não implementação da estrutura de dados manualmente.

2 Análise Léxica

A função da leitura do compilador é identificar lexemas na linguagem e passar uma sequência desses *tokens* para o sintático avaliar se a ordem está correta.

Neste projeto foi utilizado o **flex** [Pro12] para gerar o analisador léxico, além disso foram feitas tais modificações no arquivo, como as variáveis para contagem de linhas, colunas e erros léxicos além do tratamento de caso em que não consiga abrir o arquivo a ser escaneado, para ser entrada do usuário os lexemas a serem lidos.

A variável de escopo esta sendo tratada no léxico como um contador que acrescenta quando encontra uma abertura de escopo e diminui com a fechamento de escopo porém essa implementação será refeita para uma com mesma ideia porém utilizando listas de escopos e um ponteiro que vai atualizando o escopo atual de acordo com que é lido no léxico. A estrutura em que os lexemas são transformados é da forma linha, coluna escopo e valor.

Para cada lexema lido é alocado como folha para o sintático , e também é colocado em uma lista de folhas para um futura desalocação desses lexemas lidos.

3 Análise Sintática

A função desta fase é verificar se as sentenças estão de acordo com a gramática. Como referência para construção da gramática C-IPL foi utilizado o [Hec21] e [Nal13].

Neste projeto foi utilizado o **bison** [RC21] como gerador do analisador sintático, além disso foram feitas modificações no arquivo como a inclusão de variável para contar o número de erros sintáticos e a opção de entrada do teclado do usuário em caso que não consiga abrir o arquivo dado como entrada do executável.

A tabela de símbolos é montada nesta fase utilizando as sentenças de declaração de variáveis e parâmetros de variáveis como uma forma de colocar na lista ligada da tabela de símbolos. Neste projeto toda vez que se encontre uma sentença válida é colocado o id com os parâmetros do léxico sendo eles linha, coluna, escopo e valor do id além de colocar se ele é ou não função.

Nesta fase também se faz a árvore sintática abstrata utilizando como base a árvore dada como saída do **bison** [RC21]. A árvore consiste de algumas produções e terminais que são geradas pelo analisador sintático. Cada produção tem o tipo Nó e cada terminal tem o tipo Folha declarado na união.

4 Análise Semântica

A função desta fase é verificar se a árvore abstrata está com as informações corretas para desenvolvimento do código de três endereços.

5 Testes

No diretório de testes possui dos arquivos com erros semânticos e sintáticos **error.c** e **error1.c** e dois arquivos sem erros **success.c** e **success1.c**.

Os erros semânticos no arquivos **error.c** é a operação : entre dois tipos int e atribuição do tipo float list para o tipo int, os erros sintáticos são for() sem corpo e a operação de relação faltando um parâmetro.

Os erros semânticos no arquivo **error1.c** é não declaração da *main* e uma chamada de função faltando um parâmetro e os erros sintáticos são *and* com um parâmetro faltante e um *if* sem corpo.

6 Instruções

Para se compilar o analisador sintático utiliza-se os seguintes comandos ou make:

```
bison ./src/syntax.y
--defines=./lib/syntax.tab.h -o ./src/syntax.tab.c; \
flex -o ./src/lex.yy.c ./src/lexico.l; \
cc -Wall -Wextra -Wpedantic -g ./src/*.c -o tradutor -Ilib -O0; \
./tradutor <caminho_do_arquivo_teste>
```

References

ALSU06. A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2 edition, 2006.

Hec21. R. Heckendorn. C- grammar, 2021. `http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf`. Acessado por último 02 Set 2021.

Lev09. J. R. Levine. *Flex and Bison*. O'reilly, 1 edition, 2009.

Mar21. J. D. Marangon. Compiladores para humanos, 2021. `https://johnidm.gitbooks.io/compiladores-para-humanos/content/`. Acessado por último 02 Set 2021.

Nal13. C. Nalon. Glc-precedencia, 2013. Notas de aula.

Pro12. Flex Project. Flex Manual, 2012. `https://westes.github.io/flex/manual/index.html`.Acessado por último em 30 Ago 2021.

RC21. R. Stallman R. Corbett. Bison, 2021. `https://www.gnu.org/software/bison/manual/bison.html`. Acessado por último 30 Ago 2021.

SP21. IME SP. Tabela de precendência c, 2021. `https://www.ime.usp.br/~yw/bcc2000/C/tabela.html`.Acessado por último 02 Set 2021.

Appendices

Appendix A

Tabelas de Símbolos

Lexema	Expressão Regular e Descrição Informal	Exemplos
ID	qualquer letra com '_' seguido de um número [A-Za-z_]+[A-Za-z0-9_]*	jujuba_doce
LISTTYPE	qualquer tipo do list list	list
TYPES	tipos int e float int—float	int float
NULL	constante para listas vazias NIL	NIL
STRING	qualquer caracter dentro de " " \"[^\"]*\"—'[^']*	"chocolate"
ASSIGN	Um único caractere = =	=
IF	palavra chave if if	if
ELSE	palavra chave else else	else
FOR	palavra chave for for	for
RETURN	palavra chave return return	return
OUTPUT	Palavras da saída de dados da linguagem writeln — write	write writeln
INPUT	Palavra de entrada de dados read	read
INFIX	Palavras para construtores de lista :	:
FUNCTION	Caracteres para as função de map e filter >>— <<	>><<
COMMENT	Qualquer comentário em linha com // ou comentário multilinha com /**/ \/\/.*—\/*[^*\/]**\/	//pudim /* pudim é bom /
+ — - — * — /	Operações aritmeticas	+ - * /
—— — &&	Operações Lógicas	&& ——
>—<—>=—<=—==—!=	Operação de comparação	<<= >>= == !=
{}	Escopo	{}
()	Expressão	()
;	Separador de Expressão	;
?	Operação head da lista	?
!	Operação tail da lista ou negação	!
%	Operação tail destrutiva da lista	%
123	Número inteiro {DIGIT}+	123
.5	Número decimal {DIGIT}*\.{DIGIT}+(E[+—-]?{DIGIT}+)?	.5

Appendix B

Gramática

⟨program⟩ ::= ⟨paramlist⟩ (1)

⟨paramlist⟩ ::= ⟨param⟩ ⟨paramlist⟩ | ⟨param⟩ (2)

⟨param⟩ ::= ⟨variableParam⟩ | ⟨functionParam⟩ (3)

⟨variableParam⟩ ::= `TYPE LISTTYPE ID ;` (4)

| `TYPE ID ;` (5)

⟨functionParam⟩ ::= `TYPE ID (` ⟨functionParams⟩ `)` ⟨stmt⟩ (6)

| `TYPE LISTTYPE ID (` ⟨functionParams⟩ `)` (7)

⟨stmt⟩ (8)

⟨functionParams⟩ ::= ⟨functionParamsList⟩ | ϵ (9)

⟨functionParamsList⟩ ::= ⟨functionParamsList⟩ `, TYPE ID` (10)

| ⟨functionParamsList⟩ `, TYPE LISTTYPE` (11)

`ID` (12)

| `TYPE ID` (13)

| `TYPE LISTTYPE ID` (14)

⟨call⟩ ::= `ID (` ⟨argList⟩ `)` (15)

⟨argList⟩ ::= ⟨argList⟩ `, ID` (16)

| `ID` (17)

| ϵ (18)

⟨stmList⟩ ::= ⟨stmList⟩⟨stmt⟩ | ⟨stmt⟩ (19)

⟨stmt⟩ ::= ⟨expStatement⟩ | ⟨compoundStatement⟩ (20)

| ⟨ifStatement⟩ | ⟨forStatement⟩ (21)

| ⟨returnStatement⟩ | ⟨inputStatement⟩ (22)

| ⟨outputStatement⟩ (23)

| ⟨variableParam⟩ (24)

⟨expStatement⟩ ::= ⟨expression⟩ `;` (25)

⟨compoundStatement⟩ ::= { ⟨stmList⟩ } (26)

| { } (27)

⟨ifStatement⟩ ::= `if (` ⟨expression⟩ `)` ⟨stmt⟩ (28)

| `if (` ⟨expression⟩ `)` ⟨stmt⟩ `else` ⟨stmt⟩ (29)

⟨forStatement⟩ ::= `for (` ⟨expStatement⟩⟨expStatement⟩ (30)

⟨expression⟩ `)`⟨stmt⟩ (31)

⟨returnStatement⟩ ::= `return` ⟨expression⟩ `;` (32)

⟨inputStatement⟩ ::= ⟨input⟩ `(` `ID` `)` `;` (33)

⟨outputStatement⟩ ::= ⟨output⟩ `(` ⟨term⟩ `)` `;` (34)

⟨expression⟩ ::= `ID` `=` ⟨expression⟩ (35)

| ⟨orExpression⟩ (36)

⟨orExpression⟩ ::= ⟨orExpression⟩ `||` ⟨andExpression⟩ (37)

⟨andExpression⟩ ::= ⟨andExpression⟩ `&&` ⟨relationalExpression⟩ (38)

| ⟨relationalExpression⟩ (39)

⟨relationalExpression⟩ ::= ⟨relationalExpression⟩ ⟨REL_OP⟩ (40)

⟨listExpression⟩ (41)

| ⟨listExpression⟩ (42)

⟨listExpression⟩ ::= ⟨arithmExpression⟩ ⟨listOP⟩ (43)

⟨listExpression⟩ (44)

| ⟨arithmExpression⟩ (45)

⟨arithmExpression⟩ ::= ⟨arithmExpression⟩ ⟨SUB_ADD⟩ (46)

⟨airtmMulDivExpression⟩ (47)

| ⟨arihtmMulDivExpression⟩ (48)

⟨arithmMulDivExpression⟩ ::= ⟨arithmMulDivExpression⟩ ⟨MUL_DIV⟩ (49)

⟨term⟩ (50)

| ⟨term⟩ (51)

⟨term⟩ ::= ⟨const⟩ | ⟨call⟩ | `ID` (52)

| ⟨unaryTerm⟩ (53)

| ⟨immutable⟩ (54)

⟨unaryTerm⟩ ::= `!` ⟨term⟩ (55)

| `?` ⟨term⟩ (56)

| `%` ⟨term⟩ (57)

| ⟨SUB_ADD⟩ ⟨term⟩ (58)

⟨immutable⟩ ::= `(` ⟨expression⟩ `)` (59)

⟨const⟩ ::= `INT` | `FLOAT` | `STRING` | `NIL` (60)

⟨listOP⟩ ::= ⟨FUNCTION⟩ | ⟨INFIX⟩ (61)

⟨output⟩ ::= `writeln` | `write` (62)

⟨input⟩ ::= `read` (63)

⟨TYPE⟩ ::= `int` | `float` (64)

⟨LISTTYPE⟩ ::= `list` (65)

⟨REL_OP⟩ ::= `<` | `<=` | `==` | `>` | `>=` (66)

⟨INFIX⟩ ::= `:` (67)

⟨FUNCTION⟩ ::= `>>` | `<<` (68)

⟨SUB_ADD⟩ ::= `-` | `+` (69)

⟨MUL_DIV⟩ ::= `*` | `/` (70)

(71)