Universidade Federal de Ouro Preto – UFOP
Departamento de Computação – DECOM
Construção de Compiladores I – BCC238

Palloma Stéphanne Silva Brito – 16.1.4196

Análise Sintática Descendente Recursiva

1. Resumo

O objetivo deste trabalho é desenvolver um analisador sintático preditivo para a linguagem de programação apresentada por Torben em seu livro "Introduction to Compiler Design" no capítulo 4 (gramática 4.1), usando o analisador léxico disponibilizado pelo professor José Romildo Malaquias.

2. Desenvolvimento

A Figura 1.a apresentada a gramática inicial disponibilizada para o desenvolvimento do analisador sintático. Como é possível observar, a gramática em questão possuía recursão à esquerda, ambiguidade e suas regras de produção não estão fatoradas à esquerda. Além disso, não havia um símbolo para o fim de cadeia.

A Figura 1.b) apresenta a gramática após remoção de ambiguidades e a figura 1.c) após a eliminação de recursão à esquerda. Por fim, temos a gramática final, mostrada na figura 1.d) após fatoração das regras de produção e inserção do símbolo para fim de cadeia.

a) Gramática Original	b) Gramática sem Ambiguidades
Program → Funs *Funs → Fun* *Funs → Fun Funs* *Fun → TypeId (TypeIds) = Exp* *TypeId → int id* *TypeId → bool id* *TypeIds → TypeId* *TypeIds → TypeId, TypeIds* *Exp → num* *Exp → id* *Exp → Exp + Exp* *Exp → Exp < Exp* *Exp → if Exp then Exp else Exp* *Exp → id (Exps)* *Exp → let id = Exp in Exp*	*Program → Funs* *Funs → Fun* *Funs → Fun Funs* *Fun → TypeId (TypeIds) = Exp* *TypeId → int id* *TypeId → bool id* *TypeIds → TypeId* *TypeIds → TypeId, TypeIds* *Exp → let id = Exp in Exp* *Exp → if Exp then Exp else Exp* *Exp → A < A* *Exp → A* *A → A + T*

Exps → Exp *Exps → Exp, Exps*	*A → T* *T → id (Exps)* *T → id* *T → num* *Exps → Exp* *Exps → Exp, Exps*
c) Gramática sem Recursão à Esquerda	d) Gramática final: fatoração + símbolo para fim de cadeia ($)
Program → Funs *Funs → Fun* *Funs → Fun Funs* *Fun → TypeId (TypeIds) = Exp* *TypeId → int id* *TypeId → bool id* *TypeIds → TypeId* *TypeIds → TypeId, TypeIds* *Exp → let id = Exp in Exp* *Exp → if Exp then Exp else Exp* *Exp → A < A* *Exp → A* *A → T A'* *A' → + T A'* *A' →* *T → id (Exps)* *T → id* *T → num* *Exps → Exp* *Exps → Exp, Exps*	*S → Program$* *Program → Funs* *Funs → Fun Funs'* *Funs' →* *Funs' → Funs* *Fun → TypeId (TypeIds) = Exp* *TypeId → int id* *TypeId → bool id* *TypeIds → TypeId TypeIds'* *TypeIds' →* *TypeIds' → , TypeIds* *Exp → if Exp then Exp else Exp* *Exp → let id = Exp in Exp* *Exp → A Exp'* *Exp' → < A* *Exp' →* *A → B A'* *A' → + B A'* *A' →* *B → id B'* *B' → (Exps)* *B' ->* *B → num* *Exps → Exp Exps'* *Exps' →* *Exps' → , Exps*

Figura 1. Etapas de modificação da gramática

Após realizar as modificações na gramática construiu-se a tabela de nullable, first e follow, a qual segue:

NT	Nullable	First	Follow
S	F	int, bool	$
Program	F	int, bool	$
Funs	F	int, bool	$
Funs’	T	int, bool	$
Fun	F	int, bool	$, int , bool
TypeId	F	int, bool	(,), ,
TypeIds	F	int, bool, ,	)
TypeIds’	T	,	)
Exp	F	id, num, if, let	$, int, bool, then, else, +, in,), ,
Exp’	T	<	$, int, bool, then, else, +, in,), ,
A	F	id, num	$, int, bool, then, else, +, <, in,), ,
A’	T	+	$, int, bool, then, else, +, <, in,), ,
B	F	id, num	+, $, int, bool, then, else, +, <, in,), ,
B’	T	(	+, $, int, bool, then, else, +, <, in,), ,
Exps	F	id, num, if, let	)
Exps’	T	,	)

A partir da tabela acima, construiu-se a tabela LL1 seguinte:

NT	$	(	)	=	int	id	bool
S					S→Program$		S→Program$
Program					Program → Funs		Program→Funs
Funs					Funs → Fun Funs’		Funs→ Fun Funs’
Funs’	Funs’ →				Funs’→ Funs		Funs’ → Funs
Fun					Fun→TypeId (TypeIds)=Exp		Fun→TypeId (TypeIds) = Exp
TypeId					TypeId→int id		TypeId→ bool id
TypeIds					TypeIds→TypeId TypeIds’		TypeIds → TypeId TypeIds’
TypeIds’			TypeIds’→				
Exp						Exp→A Exp’	
Exp’	Exp’→		Exp’→		Exp’→		Exp’ →
A						A→B A’	
A’	A’→		A’→		A’→		A’ →
B						B→id B’	
B’	B’→	B’→(Exps)	B’→		B’→		B’ →
Exps						Exps→ Exp Exps’	
Exps’			Exps’→				

NT	,	if	then	else	let	in	<	+	num
S									
Program									
Funs									
Funs'									
Fun									
TypeId									
TypeIds									
TypeIds'	TypeIds'→ , TypeIds								
Exp		Exp → if Exp then Exp else Exp			Exp→ let id = Exp in Exp				Exp → A Exp'
Exp'	Exp' →		Exp' →	Exp'→		Exp'→	Exp'→ < A		
A									A → B A'
A'	A' →		A'→	A'→		A' →	A'→	A'→ + B A'	
B									B → num
B'	B'→		B'→	B'→		B'→	B'→	B' →	
Exps		Exps → Exp Exps'			Exps → Exp Exps'				Exps →Exp Exps'
Exps'	Exps' → , Exps								

A partir da tabela acima e utilizando o JFlex desenvolveu-se o analisador sintático para a gramática proposta. Alguns arquivos de testes estão disponibilizados no trabalho.