UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO (CTC)
SISTEMAS DE INFORMAÇÃO
INE5622 - Introdução a Compiladores

Docente: Daniel Santana de Freitas
Grupo: Arthur Lorenzetti da Rosa **Matrícula:** 19200621
Jacqueline Correia Beber **Matrícula:** 19200634

Parte 2

- **Gramática LSI-2025-2 transformada:**

MAIN → STMT | FLIST | ε

FLIST → FDEF FLIST'
FLIST' → FLIST | ε

FDEF → def id (PARLIST) { STMTLIST }

PARLIST → int id PARLIST' | ε
PARLIST' → , PARLIST | ε

VARLIST → id VARLIST'
VARLIST' → , VARLIST | ε

STMT → int VARLIST ;
| ATRIBST ;
| PRINTST ;
| RETURNST ;
| IFSTMT
| { STMTLIST }
| ;

ATRIBST → id = EXPR

PARLISTCALL → id PARLISTCALL' | ε
PARLISTCALL' → , PARLISTCALL | ε

PRINTST → print EXPR

RETURNST → return RETURNST'
RETURNST' → id | ε

IFSTMT → if (EXPR) { STMT } IFSTMT'
IFSTMT' → else { STMT }| ε

STMTLIST → STMT STMTLIST'
STMTLIST' → STMTLIST | ε

EXPR → NUMEXPR EXPR'
EXPR' → OP NUMEXPR | ε

OP → < | > | <= | >= | == | !=

NUMEXPR → TERM NUMEXPR'
NUMEXPR' → + TERM NUMEXPR'
| - TERM NUMEXPR'
| ε

TERM → FACTOR TERM'
TERM' → * FACTOR TERM'
| / FACTOR TERM'
| ε

FACTOR → num | (NUMEXPR) | id FACTOR'
FACTOR' → (PARLISTCALL) | ε

➢ **Recursão à Esquerda:**

As produções que possuem recursão à esquerda fazem com que o parser preditivo fique em loop infinito. Para resolver isso devemos tratar a recursão à esquerda, reescrevendo a produção, aplicando a transformação:

$S \rightarrow S\alpha \mid \beta$ $\qquad$ $S \rightarrow \beta\, S'$
$S' \rightarrow \alpha S' \mid \varepsilon$

Abaixo temos as produções originais que possuíam recursão à esquerda e como que reescrevemos elas.

1 - Produção Original: NUMEXPR → NUMEXPR + TERM
| NUMEXPR - TERM
| TERM

Reescrevendo a produção, aplicando a transformação, temos:

NUMEXPR → TERM NUMEXPR'

NUMEXPR' → + TERM NUMEXPR'
| - TERM NUMEXPR'
| ε

2 - Produção Original: TERM → TERM * FACTOR
| TERM / FACTOR
| FACTOR

Reescrevendo a produção, aplicando a transformação, temos:

TERM → FACTOR TERM'

TERM' → * FACTOR TERM'
| / FACTOR TERM'
| ε

➢ **Fatoração à Esquerda:**

As produções que possuem fatoração à esquerda são aquelas onde a escolha entre duas produções não é clara. Por exemplo:

$A \rightarrow \alpha\beta^1 \mid \alpha\beta^2$

Quando o parser lê o **α** ele não sabe imediatamente qual produção escolher, não sabendo se A deve ser expandido para $\alpha\beta^1$ ou $\alpha\beta^2$. Para resolver isso, podemos adiar a decisão, conforme o exemplo abaixo:

$A \rightarrow \alpha\beta^1 \mid \alpha\beta^2$ $\qquad$ $A \rightarrow \alpha\ A'$
$A' \rightarrow \beta^1 \mid \beta^2$

As gramáticas em LL(1) só podem olhar um símbolo à frente por esse motivo devemos acabar com a fatoração à esquerda antes de conseguir fazer o parser preditivo.

Abaixo temos as produções originais que possuíam fatoração à esquerda e como que reescrevemos elas.

1 - Produção Original: FLIST → FDEF FLIST | FDEF

Reescrevendo a produção:

FLIST → FDEF FLIST’

FLIST’ → FLIST | ε

2 - Produção Original: PARLIST → int id, PARLIST | int id | ε

Reescrevendo a produção:

PARLIST → int id PARLIST’ | ε

PARLIST’ → , PARLIST | ε

3 - Produção Original: VARLIST → id, VARLIST | id

Reescrevendo a produção:

VARLIST → id VARLIST’

VARLIST’ → , VARLIST | ε

4 - Produção Original: ATRIBST → id = EXPR | id = FCALL

Reescrevendo a produção:

ATRIBST → id = ATRIBST’

ATRIBST’ → EXPR | FCALL

Aqui temos um problema, tanto EXPR e FCALL podem derivar id:

FCALL → id (PARLISTCALL)

EXPR → NUMEXPR → TERM → FACTOR → id

Quando o parser vê id = id, não se sabe se id = EXPR, onde EXPR deriva id, ou id = FCALL, onde FCALL é id (PARLISTCALL). Podemos resolver isso eliminando o não terminal FCALL e integrando ele no FACTOR, pois:

FACTOR → num | (NUMEXPR) | id

Então temos as seguintes mudanças:

ATRIBST → id = EXPR

FACTOR → num | (NUMEXPR) | id | id (PARLISTCALL)

FCALL é absorvido por FACTOR, mas agora temos que fazer uma fatoração em FACTOR, pois as duas últimas produções começam com id:

FACTOR → num | (NUMEXPR) | id FACTOR'

FACTOR' → (PARLISTCALL) | ε

5 - Produção Original: PARLISTCALL → id, PARLISTCALL | id | ε

Reescrevendo a produção:

PARLISTCALL → id PARLISTCALL' | ε

PARLISTCALL' → , PARLISTCALL | ε

6 - Produção Original: RETURNST → return id | return

Reescrevendo a produção:

RETURNST → return RETURNST'

RETURNST' → id | ε

7 - Produção Original: IFSTMT → if (EXPR) {STMT} else {STMT}
| if (EXPR) {STMT}

Reescrevendo a produção:

IFSTMT → if (EXPR) {STMT} IFSTMT'

IFSTMT’ → else {STMT}| ε

8 - Produção Original: STMTLIST → STMT STMTLIST | STMT

Reescrevendo a produção:

STMTLIST → STMT STMTLIST’

STMTLIST’ → STMTLIST | ε

9 - Produção Original: EXPR → NUMEXPR < NUMEXPR
| NUMEXPR <= NUMEXPR
| NUMEXPR > NUMEXPR
| NUMEXPR >= NUMEXPR
| NUMEXPR == NUMEXPR
| NUMEXPR != NUMEXPR
| NUMEXPR

Reescrevendo a produção:

Para simplificar, criamos um novo não terminal para armazenar as operações:

OP → <= | >= | == | !=

Isso faz com que EXPR fique da seguinte forma:

EXPR → NUMEXPR OP NUMEXPR | NUMEXPR

Agora, podemos perceber que ainda temos que fazer uma fatoração para que a gramática seja aceita como LL(1) pois quando vemos NUMEXPR não sabemos se devemos ir para OP ou se acabou. EXPR com a nova fatoração fica:

EXPR → NUMEXPR EXPR’

EXPR’ → OP NUMEXPR | ε

Agora sim a linguagem LSI-2025-2 é uma LL(1).

Total de não terminais = 28

MAIN STMT FLIST FLIST' FDEF PARLIST PARLIST' (7)
STMTLIST STMTLIST' VARLIST VARLIST' ATRIBST (5)
PRINTST RETURNST RETURNST' IFSTMT IFSTMT' (5)
EXPR EXPR' PARLISTCALL PARLISTCALL' (4)
NUMEXPR NUMEXPR' OP TERM TERM' (5)
FACTOR FACTOR' (2)

Total de terminais = 25

int id print return if else { } ; def num , ((13)
) < > = == != <= >= + - * / (12)

- **FIRST:**

MAIN	STMT	FLIST	FLIST'	FDEF	PARLIST	PARLIST'	STMTLIST
int id print return if { ; def ε	int id print return if { ;	def	def ε	def	int ε	, ε	int id print return if { ;

STMTLIST'	VARLIST	VARLIST'	ATRIBST	PRINTST	RETURNST	RETURNST'
int id print return if { ; ε	id	, ε	id	print	return	id ε

IFSTMT	IFSTMT'	EXPR	EXPR'	PARLISTCALL	PARLISTCALL'	NUMEXPR
if	else ε	num (id	< > <= >= == != ε	id ε	, ε	num (id

NUMEXPR'	OP	TERM	TERM'	FACTOR	FACTOR'
+ - ε	< > <= >= == !=	num (id	* / ε	num (id	(ε

- **FOLLOW:**

MAIN	STMT	FLIST	FLIST'	FDEF	PARLIST	PARLIST'	STMTLIST

$	$
	}
	int
	id
	print
	return
	if
	{
	;
		$	$	$
				def
					)	)	}

STMTLIST'	VARLIST	VARLIST'	ATRIBST	PRINTST	RETURNST	RETURNST'
}	;	;	;	;	;	;

IFSTMT	IFSTMT'	EXPR	EXPR'	PARLISTCALL	PARLISTCALL'	NUMEXPR

$
}
int
id
print
return
if
{
;
	$
	}
	int
	id
	print
	return
	if
	{
	;
		;
		)
			;
			)
				)	)	<
						>
						<=
						>=
						==
						!=
						)
						;

NUMEXPR'	OP	TERM	TERM'	FACTOR	FACTOR'
< > <= >= == !=) ;	num (id	+ - < > <= >= == !=) ;	+ - < > <= >= == !=) ;	* / + - < > <= >= == !=) ;	* / + - < > <= >= == !=) ;

- **Tabela de reconhecimento sintático:**

  Para melhor visualização, criamos a tabela no Google Sheets, segue o link:

https://docs.google.com/spreadsheets/d/1cVK4ZIDSYa3Rqk-xt89RHHR0i-8UtqpUAtw74EK-S_U/edit?usp=sharing

- **Exemplo de aplicação:**

  Programa de entrada: print(num*num);$

Matched	Pilha	Entrada	Ação
	MAIN$	print(num*num);$	
	STMT$	print(num*num);$	MAIN → STMT

	PRINTST;$	print(num*num);$	STMT → PRINTST;
	print EXPR;$	print(num*num);$	PRINTST → print EXPR
print	EXPR;$	(num*num);$	match **print**
print	NUMEXPR EXPR';$	(num*num);$	EXPR → NUMEXPR EXPR'
print	TERM NUMEXPR'EXPR';$	(num*num);$	NUMEXPR → TERM NUMEXPR'
print	FACTOR TERM' NUMEXPR' EXPR';$	(num*num);$	TERM → FACTOR TERM'
print	(NUMEXPR) TERM' NUMEXPR' EXPR';$	(num*num);$	FACTOR → (NUMEXPR)
print(	NUMEXPR) TERM' NUMEXPR' EXPR';$	num*num);$	match **(**
print(	TERM NUMEXPR') TERM' NUMEXPR' EXPR';$	num*num);$	NUMEXPR → TERM NUMEXPR'
print(	FACTOR TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	num*num);$	TERM → FACTOR TERM'
print(	num TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	num*num);$	FACTOR → num
print(num	TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	*num);$	match **num**
print(num	* FACTOR TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	*num);$	TERM' → * FACTOR TERM'
print(num*	FACTOR TERM' NUMEXPR') TERM' NUMEXPR'EXPR';$	num);$	match *
print(num*	num TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	num);$	FACTOR → num
print(num*num	TERM' NUMEXPR') TERM' NUMEXPR' EXPR';$	);$	match **num**
print(num*num	NUMEXPR') TERM' NUMEXPR' EXPR';$	);$	TERM' → ε
print(num*num	) TERM' NUMEXPR' EXPR';$	);$	NUMEXPR' → ε
print(num*num)	TERM' NUMEXPR' EXPR';$	;$	match **)**

print(num*num)	NUMEXPR’ EXPR’;$	;$	TERM’ → ε
print(num*num)	EXPR’;$	;$	NUMEXPR’ → ε
print(num*num)	;$	;$	EXPR’ → ε
print(num*num);	$	$	match **;**
print(num*num);	**$**	**$**	**FIM**