

Compiladores - Análise Preditiva

Fabio Mascarenhas – 2017.2

<http://www.dcc.ufrj.br/~fabiom/comp>

Analizador Preditivo

- Uma simplificação do parser recursivo com retrocesso que é possível para muitas gramáticas são os *parsers preditivos*
- Um parser preditivo não tenta alternativas até uma ser bem sucedida, mas usa um lookahead na entrada para prever qual alternativa ele deve seguir
 - Só falha se realmente o programa está errado!
- Quanto mais tokens à frente podemos examinar, mais poderoso o parser
- Classe de gramáticas LL(k), onde k é quantos tokens de lookahead são necessários

LL(1)

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { ; CMD }

CMD -> if EXP then CMDS [else CMDS] end

| repeat CMDS until EXP

| id := EXP

| read id

| write EXP

EXP -> SEXP { < SEXP | = SEXP }

SEXP -> TERMO { + TERMO | - TERMO }

TERMO -> FATOR { * FATOR | / FATOR }

FATOR -> "(" EXP ")" | num | id

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { ; CMD } **1**

CMD -> if EXP then CMDS [else CMDS] end

| repeat CMDS until EXP

| id := EXP

| read id

| write EXP

EXP -> SEXP { < SEXP | = SEXP }

SEXP -> TERMO { + TERMO | - TERMO }

TERMO -> FATOR { * FATOR | / FATOR }

FATOR -> "(" EXP ")" | num | id

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { ; CMD } **1**

CMD -> if EXP then CMDS [else CMDS] end

| repeat CMDS until EXP

| id := EXP

| read id

| write EXP

EXP -> SEXP { < SEXP | = SEXP }

SEXP -> TERMO { + TERMO | - TERMO }

TERMO -> FATOR { * FATOR | / FATOR }

FATOR -> "(" EXP ")" | num | id

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

$S \rightarrow \text{CMDS}$

$\text{CMDS} \rightarrow \text{CMD} \{ ; \text{CMD} \}$ **1**

$\text{CMD} \rightarrow \text{if EXP then CMDS [else CMDS] end}$ **1**

 | repeat CMDS until EXP

1 | id := EXP

 | read id .

 | write EXP

$\text{EXP} \rightarrow \text{SEXP} \{ < \text{SEXP} \mid = \text{SEXP} \}$

$\text{SEXP} \rightarrow \text{TERMO} \{ + \text{TERMO} \mid - \text{TERMO} \}$

$\text{TERMO} \rightarrow \text{FATOR} \{ * \text{FATOR} \mid / \text{FATOR} \}$

$\text{FATOR} \rightarrow "(" \text{EXP} ")" \mid \text{num} \mid \text{id}$

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { ; CMD } **1**

CMD -> if EXP then CMDS [else CMDS] end **1**

| repeat CMDS until EXP

| id := EXP

| read id **1**

| write EXP **1**

EXP -> SEXP { < SEXP | = SEXP }

SEXP -> TERMO { + TERMO | - TERMO }

TERMO -> FATOR { * FATOR | / FATOR }

FATOR -> "(" EXP ")" | num | id

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { ; CMD } **1**

CMD -> if EXP then CMDS [else CMDS] end **1**

| repeat CMDS until EXP

| id := EXP

| read id **1**

| write EXP

EXP -> SEXP { < SEXP | = SEXP } **1**

SEXP -> TERMO { + TERMO | - TERMO } **1**

TERMO -> FATOR { * FATOR | / FATOR } **1**

FATOR -> "(" EXP ")" | num | id

Voltando a TINY

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

$S \rightarrow \text{CMDS}$

$\text{CMDS} \rightarrow \text{CMD} \{ ; \text{CMD} \}$ **1**

$\text{CMD} \rightarrow \text{if EXP then CMDS [else CMDS] end}$ **1**

 | repeat CMDS until EXP

 | id := EXP

 | read id **1**

 | write EXP

$\text{EXP} \rightarrow \text{SEXP} \{ < \text{SEXP} \mid = \text{SEXP} \}$ **1**

$\text{SEXP} \rightarrow \text{TERMO} \{ + \text{TERMO} \mid - \text{TERMO} \}$ **1**

$\text{TERMO} \rightarrow \text{FATOR} \{ * \text{FATOR} \mid / \text{FATOR} \}$ **1**

$\text{FATOR} \rightarrow "(" \text{EXP} ")" \mid \text{num} \mid \text{id}$

1

Voltando a TINY

| if EXP then CMDS
| if EXP then CMDS else CMDS

- Vamos ver quantos tokens de lookahead precisamos para prever cada opção:

S -> CMDS

CMDS -> CMD { , CMD }

CMD -> if EXP then CMDS [else CMDS] end

| repeat CMDS until EXP

| id := EXP

| read id

| write EXP

EXP -> SEXP { < SEXP | = SEXP }

SEXP -> TERMO { + TERMO | - TERMO }

TERMO -> FATOR { * FATOR | / FATOR }

FATOR -> "(" EXP ")" | num | id

ELSE

1

1

ELSE -> else CMDS

1

1

1

1

TINY é LL(1)!

Analizador Recursivo Preditivo (1) LL(1)

- O analisador recursivo preditivo é parecido com o recursivo com retrocesso, mas ao invés de *try* nas escolhas usa a informação de lookahead
- Um **terminal** continua testando o token atual, e avançando para o próximo token se o tipo for compatível, mas lança um **erro** se não for
- Uma **sequência** simplesmente executa cada termo da sequência
- Uma **alternativa** usa o token de lookahead como índice de um *switch-case*, e o conjunto de lookahead de cada alternativa forma os casos dela;

Analizador Recursivo Preditivo (2)

- Um **opcional** verifica se o token de lookahead está no conjunto do seu termo, caso esteja segue executando ele, senão pula
- Uma **repetição** repete os seguintes passos: verifica se o token de lookahead está no conjunto de seu termo, se estiver executa ele, se não pára a repetição
- Um **não-terminal** vira um procedimento separado, e executa o procedimento correspondente
- Construir a árvore sintática é simples, já que não há retrocesso, e pode-se sempre adicionar nós à árvore atual

Regras de Construção (1)

- `$parser[termo, arvore]` dá o código para análise sintática do termo, guardando o resultado em um ou mais nós de `arvore` caso seja bem sucedido

```
$parser[terminal, arvore] =  
  ($arvore).child(match($terminal));
```

```
$parser[t1...tn, arvore] =  
  $parser[t1, arvore]  
  ...  
  $parser[tn, arvore]
```

```
$parser[NAOTERM, arvore] =  
  ($arvore).child(NAOTERM());
```

Regras de Construção (2)


- \$parser[termo, arvore] dá o código para análise sintática do termo, guardando o resultado em um ou mais nós de arvore caso seja bem sucedido

```
$parser[t1 | t2, arvore] =  
  // { t1la1, ..., t1lan } é o conjunto de lookahead de t1  
  // { t2la1, ..., t2lan } é o conjunto de lookahead de t2  
  if(lookahead.tipo == t1la1 || ... || lookahead.tipo == t1lan) {  
    $parser[t1, arvore]  
  } else if(lookahead.tipo == t2la1 || ... || lookahead.tipo == t2lan) {  
    $parser[t2, arvore]  
  } else erro(t1la1, ..., t1lan, t2la1, ..., t2lan);  
}
```

Regras de Construção (3)

- \$parser[termo, arvore] dá o código para análise sintática do termo, guardando o resultado em um ou mais nós de arvore caso seja bem sucedido

```
$parser[[ termo ], arvore] =  
  // { la1, ..., lan } é o conjunto de lookahead de termo  
  if(lookahead.tipo == la1 || ... || lookahead.tipo == lan) {  
    $parser[termo, arvore]  
  }  
  
$parser[{ termo }, arvore] =  
  // { la1, ..., lan } é o conjunto de lookahead de termo  
  while(lookahead.tipo == la1 || ... || lookahead.tipo == lan) {  
    $parser[termo, arvore]  
  }
```



Analizador preditivo para TINY

- O analisador recursivo preditivo é bem mais simples do que o analisador com retrocesso
- Pode ler os tokens sob demanda: só precisa manter um token de *lookahead*
- Não precisamos de nada especial para detecção de erros: os pontos de falha são pontos de erro, e temos toda a informação necessária lá
- Temos os mesmos problemas com recursão à esquerda