

Compiladores - Análise Preditiva

Fabio Mascarenhas - 2013.2

<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

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 ] 1 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 ] 1 end
      | repeat CMDS until EXP
1 | id := EXP
      | read id
      | write EXP
EXP    -> SEXP [ < SEXP | = SEXP ]
SEXP   -> TERMO { + TERMO | - TERMO }
TERMO  -> FATOR { * FATOR | / FATOR }
FATOR  -> "(" EXP ")" | num | id
```

Handwritten red annotations:

- A red circle around the `:=` operator in the `CMD` rule.
- A red circle around the `|` operator in the `EXP` rule.
- A red bracket under the `<` and `=` operators in the `EXP` rule.
- A red scribble over the `id` and `EXP` in the `CMD` rule, with the text `id('EXP') := EXP` written next to it.

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 ] 1 end
      | repeat CMDS until EXP
      | id := EXP
1    | read id
      | write EXP
EXP    -> SEXP { < SEXP | = SEXP } 1
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 ] 1end
      | repeat CMDS until EXP
      | id := EXP
1    | read id
      | 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      -> CMDS
CMDS   -> CMD { ; CMD } 1
CMD    -> if EXP then CMDS [ else CMDS ] 1 end
      | repeat CMDS until EXP
      | id := EXP
1    | read id
      | write EXP
EXP    -> SEXP [ < SEXP | = SEXP ] 1
SEXP   -> TERMO { + TERMO | - TERMO } 1
TERMO  -> FATOR { * FATOR | / FATOR } 1
FATOR  -> "(" EXP ")" | num | id
      | 1
```

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 ] 1 end
      | repeat CMDS until EXP
      | id := EXP
1    | read id
      | write EXP
EXP    -> SEXP ( < SEXP | = SEXP ) 1
SEXP   -> TERMO { + TERMO | - TERMO } 1
TERMO  -> FATOR { * FATOR | / FATOR } 1
FATOR  -> "(" EXP ")" | num | id
```

1

TINY é LL(1)!

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

MAS PODÉMOS USAR UM TRUQUE
P/ CONSTRUIR A ÁRVORE COMO
SE A GRAMÁTICA FOSSE R.E.

Recursão à esquerda

- Outra grande limitação dos analisadores recursivos é que as suas gramáticas não podem ter *recursão à esquerda*
- A presença de recursão à esquerda faz o analisador entrar em um laço infinito!
- Precisamos transformar recursão à esquerda em repetição
- Fácil quando a recursão é direta:

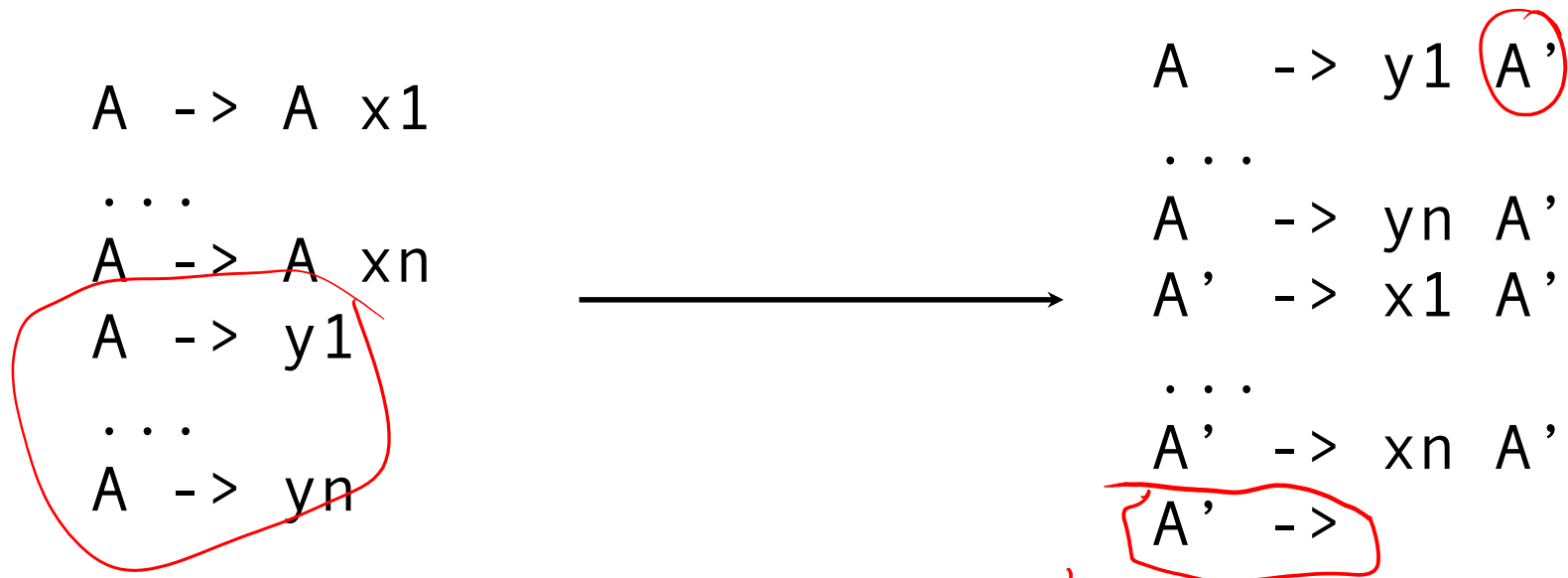
tree P() {
tree t = new("P");
t.children(P());
}

$A \rightarrow A x_1 \mid \dots \mid A x_n \mid y_1 \mid \dots \mid y_n$



$A \rightarrow (y_1 \mid \dots \mid y_n) \{ x_1 \mid \dots \mid x_n \}$

Eliminação de recursão sem EBNF



$S \text{ EXP} \rightarrow + \text{TERMO} S \text{ EXP}'$
 $S \text{ EXP}' \rightarrow - \text{TERMO} S \text{ EXP}'$
 $S \text{ EXP}' \rightarrow \text{TERMO} S \text{ EXP}'$
 $S \text{ EXP}' \rightarrow$