Linguagens de Domínio Específico

Fabio Mascarenhas – 2017.1

http://www.dcc.ufrj.br/~fabiom/dsl

Estratégias de recuperação

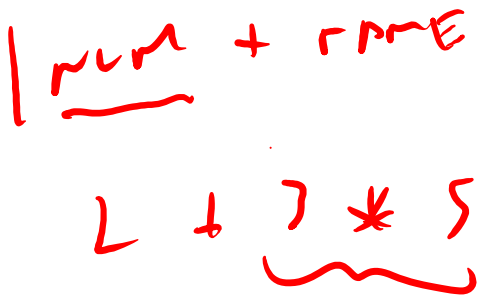
- Usamos três estratégias de recuperação de erros em nosso parser
- A primeira é ignorar o erro, e seguir adiante como se encontrássemos o termo que estávamos procurando
- A segunda assume que o token atual corresponde ao termo que estávamos procurando, e avança para o próximo token ~ types em keywords
- A terceira avança um número arbitrário de tokens até chegar em um *token de sincronização,* onde em teoria sabemos qual será o próximo termo
- Qual a melhor? Depende... Queremos evitar *erros em cascata*

Analisando expressões algébricas

- Nossa gramática de máquinas de estado não tem expressões algébricas, mas muitas linguagens têm

- Uma gramática de expressões ingênua é complicada de analisar com um analisador recursivo

```
exp := exp '+' exp
     | exp '-' exp
     | exp '*' exp
     | exp '/' exp
     | '(' exp ')'
     | NUM | NAME
```



- A recursão à esquerda faz ele entrar em loop, mesmo se ignorarmos o problema do lookahead

Ambiguidade

- Qual a ordem de precedência dos operadores na gramática do slide anterior? E sua associatividade?

- A gramática não define nenhum dos dois pois ela é *ambígua;* para transformar ela em uma gramática adequada para um analisador recursivo precisamos resolver essa ambiguidade

- Na linguagem anterior isso é óbvio: a multiplicação e divisão têm precedência maior que a soma e subtração, que têm a mesma precedência

- A subtração associa à esquerda, logo a soma também tem que associar, por causa da mesma precedência

- O mesmo para divisão e multiplicação

Gramática de expressões LL(1)

- Cada nível de precedência na nossa gramática ganha um não-terminal, com as expressões atômicas sendo o nível mais alto; o nível mais baixo fica com o não-terminal `exp` original

- Dentro de cada nível de expressão binária, os termos das expressões viram uma repetição, com uma escolha distinguindo entre operações de mesma precedência

- Cada nível de precedência referencia o próximo

```
exp := termo (‘+’ termo | ‘-’ termo)*
termo := fator (‘*’ fator | ‘/’ fator)*
fator := ‘(‘ exp ‘)’ | NUM | NAME
```

Recuperando erros em expressões

- Na gramática de expressões do slide anterior temos dois tipos de erro que queremos recuperar

- O primeiro é quando esperávamos uma expressão atômica (`fator`) e não encontramos, esse é fácil

- O segundo é quando o programador esqueceu um operador, nesse caso temos que dar mais uma volta na repetição em `termo` mesmo se o próximo token indicar um `fator` ao invés de um operador