Projeto de Compiladores

Trabalho para a disciplina Compiladores na Universidade Federal do ABC sob orientação da Profa. Mirtha Lina Fernández Venero

André Peric Tavares, Giulio Parva Denardi

1 de Maio de 2016

Conteúdo

1 Introdução

```
******* New iteration (building all state sets) *******
Analysing state 5: [(1) S -> [a, S, b, S] . [ε]]
~~Analysing rule~~
Analysing rule: (1) S -> [a, S, b, S] . [ε]
Will create Reduce action

Creating action:
 <Reduce 1>
Action position:
 Line: 5
 Columns: [b, $]
```

negrito *itálico* google link imagem:

2 Objetivos

3 Justificativa

4 Funcionamento

4.1 Produção de cadeia vazia

O cálculo dos conjuntos FIRST e FOLLOW exigem em diversos momentos saber se um determinado não terminal produz ϵ. Por exemplo, considere a sequência de símbolos ABC. Queremos calcular FIRST(ABC). Sabemos que

$$\text{FIRST(ABC)} = \text{FIRST(A)} \oplus \text{FIRST(BCD)}$$

e o resultado desta operação é somente FIRST(A) se A não produz ϵ ou FIRST(A) - ϵ se $\epsilon \in$ FIRST(A). Então é conveniente saber de antemão quais símbolos produzem ϵ.

Para tanto, usa-se o método `buildAllNonTerminalsThatProduceEps` na classe `Grammar`. O algoritmo utilizado é simples: primeiro, verifica-se todos os não terminais que produzem diretamente ϵ, isto é, aqueles que têm uma regra que produz ϵ sem etapas intermediárias, como em $A \rightarrow \epsilon$.

Em seguida, todas as regras são percorridas, e se todos os símbolos da parte direita de uma regra produzem ϵ, então adicionamos o produtor dessa regra à lista de não terminais que produzem ϵ. Todas as regras são percorridas novamente até que nenhum símbolo novo tenha sido adicionado à lista

de símbolos que produzem ϵ. Em outras palavras, até que o ponto fixo seja atingido.

Por exemplo, considere a seguinte gramática:

$$\begin{aligned} A &\rightarrow BC \\ B &\rightarrow \epsilon \\ C &\rightarrow \epsilon \end{aligned}$$

A tabela a seguir mostra o resultado desse algoritmo aplicado à gramática anterior em cada iteração.

Produz ϵ?	A	B	C
Iteração 1	não	sim	sim
Iteração 2	sim	sim	sim
Iteração 3	sim	sim	sim

Na iteração 3, o conjunto de elementos que produzem ϵ não mudou, e assim o algoritmo termina.

O código é apresentado a seguir.

```
private final void buildAllNonTerminalsThatProduceEps() {
  Set<Symbol> nonTerminalsThatGenerateEps = new HashSet<Symbol>();

  // rules that directly generate eps
  for (Symbol nonTerminal : nonTerminals) {
    for (Rule rule : rules.get(nonTerminal)) {
      if (rule.producesEmptyString()) {
        nonTerminalsThatGenerateEps.add(nonTerminal);
      }
    }
  }

  // iterates until fp is found
  boolean newNonTerminalThatGeneratesEpsHasBeenFound = true;
  while (newNonTerminalThatGeneratesEpsHasBeenFound) {
    newNonTerminalThatGeneratesEpsHasBeenFound = false;
    int setSizeBeforeIteration = nonTerminalsThatGenerateEps.size();

    for (Symbol nonTerminal : nonTerminals) {
      for (Rule rule : rules.get(nonTerminal)) {
        // verifies if all symbols from rule produce eps
```

```
        List<Symbol> production = rule.getProduction();
        boolean allSymbolsFromProductionProduceEps;
        allSymbolsFromProductionProduceEps = production
            .stream()
            .allMatch(symbol -> nonTerminalsThatGenerateEps.contains(symbol));

        // if so, add it to set
        if (allSymbolsFromProductionProduceEps) {
          nonTerminalsThatGenerateEps.add(nonTerminal);
        }
      }
    }

    // verifies whether some non terminal has been added to set
    int setSizeAfterIteration = nonTerminalsThatGenerateEps.size();
    if (setSizeBeforeIteration != setSizeAfterIteration) {
      newNonTerminalThatGeneratesEpsHasBeenFound = true;
    }
  }

  // initialise Map
  Map<Symbol, Boolean> producesEps = new HashMap<Symbol, Boolean>();
  for (Symbol nonTerminal : nonTerminals) {
    producesEps.put(nonTerminal, nonTerminalsThatGenerateEps.contains(nonTerminal));
  }
  for (Symbol terminal : terminals) {
    producesEps.put(terminal, false);
  }

  this.nonTerminalsToProducesEps = producesEps;
}
```

4.2 Representação dos conjuntos FIRST e FOLLOW

Uma das principais funcionalidades do programa deste trabalho é não só calcular os conjuntos FIRST e FOLLOW, mas fazer isso apresentando as etapas intermediárias, fazendo com que o usuário veja cada passo do algoritmo. Isso faz com que o cálculo desses conjuntos não seja o mais eficiente possível, pois precisamos lidar também com o *output* sem pular nenhuma etapa.

Para isto, criamos classes `First` e `Follow`. Estas classes têm atributos

que indicam a *representação* do conjunto dado em termos de outros conjuntos.

Por exemplo, considere os seguintes atributos da classe `Follow`:

```
private Set<Symbol> firstSets;
private Set<Symbol> firstSetsWithoutEps;
private Set<Symbol> followSets;
private Set<Symbol> terminals;
private boolean hasEOF;
```

Suponha que um objeto dessa classe tenha as seguintes atribuições (aqui em notação de teoria dos conjuntos):

$$\begin{gathered}\text{firstSets} = \{\text{A}\} \\ \text{firstSetsWithoutEps} = \{\text{B, C}\} \\ \text{followSets} = \{\text{D}\} \\ \text{terminals} = \{\text{a, b}\} \\ \text{hasEOF} = \text{true}\end{gathered}$$

Então esse conjunto seria

$$\text{FIRST(A)} \cup (\text{FIRST(B)} - \epsilon) \cup (\text{FIRST(C)} - \epsilon) \cup \text{FOLLOW(D)} \cup \{\text{a}\} \cup \{\text{b}\} \cup \{\$\}$$

Ambas as classes têm o método `toString` sobrescrito para exibir essa representação como mostrado acima e um método `getAllElements` que coleta todos os elementos vindos da união dos conjuntos.

4.3 Cálculo dos conjuntos FIRST e FOLLOW

De maneira semelhante à computação de todos os não terminais que geram ϵ, o cálculo dos conjuntos FIRST e FOLLOW consiste, em essência, em iterar até encontrar um ponto fixo.

Note que a aplicação direta da definição de FIRST e FOLLOW não funciona, pois ela falharia no caso de definições recursivas que são dependentes entre si. Por exemplo, considere o caso em que FIRST(A) = FIRST(B) e FIRST(B) = FIRST(A). Para calcular FIRST(A), calcula-se FIRST(B). Mas FIRST(B) é FIRST(A), o que resulta num *loop* infinito. Em vez disso, começamos com todos os conjuntos FIRST setados para $\emptyset$, e a cada iteração atualizamos todos os conjuntos até atingir um ponto fixo.

O código a seguir mostra a implementação desse algoritmo para o cálculo dos conjuntos FIRST.

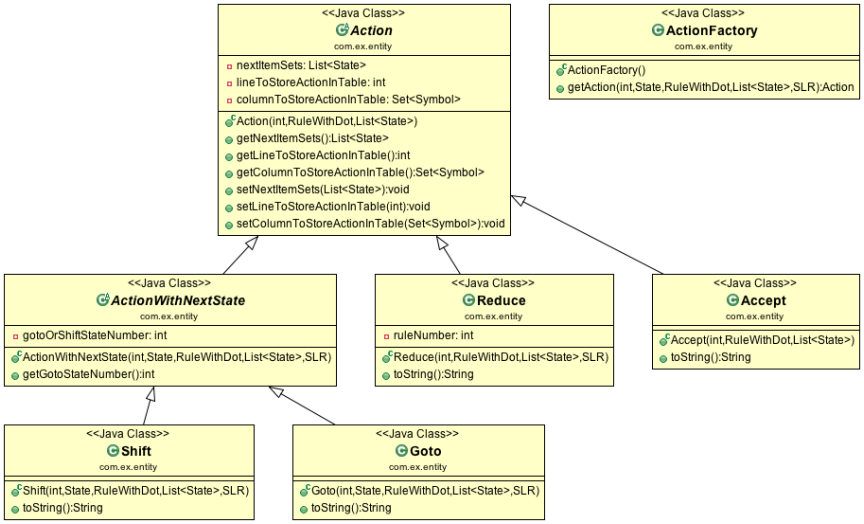
```
public final void buildAllFirstSets() {

  // Initialize set
  // omitido

  // Get description of each first set
  Map<Symbol, First> firstSetDescriptions = buildAllFirstSetDescriptions();

  // Iterate until fixed point is found
  boolean someFirstSetHasChanged = true;
  while (someFirstSetHasChanged) {
    StringBuilder iterationSb = new StringBuilder();
    iterationSb.append("New iteration (building first sets)\n");
    someFirstSetHasChanged = false;

    // Copy elements from old first sets to new first sets
    // omitido

    // Updates, possibly getting new elements
    for (Symbol nonTerminal: nonTerminals){
      iterationSb.append(String.format("Updating First(%s)\n", nonTerminal));
      First firstDescription = firstSetDescriptions.get(nonTerminal);
      iterationSb.append(String.format("First(%s) = %s\n", nonTerminal, firstDescriptio
      int numElementsBefore = firstSetsBeforeIteration.get(nonTerminal).size();
      firstSetsAfterIteration.get(nonTerminal).addAll(firstDescription.getAllElements(f
      iterationSb.append(String.format("Adding elements: %s\n", firstDescription.getAll
      int numElementsAfter = firstSetsAfterIteration.get(nonTerminal).size();
      if (numElementsBefore != numElementsAfter){
        someFirstSetHasChanged = true;
      }
    }

    iterationSb.append(String.format("All elements form first sets before iteration: %s
    iterationSb.append(String.format("All elements form first sets after iteration: %s\

    firstSetsBeforeIteration = firstSetsAfterIteration;
  }
  this.firstSets = firstSetsBeforeIteration;
}
```

O cálculo dos conjuntos FOLLOW é bastante semelhante, e por isso é omitido.

4.4 TODO LL

4.5 SLR

4.5.1 Ações



4.5.2 Regras

4.5.3 Algoritmo

5 Conclusão

6 Adendo - notas sobre algumas decisões de *design*

7 Referências Bibliográficas