Análise Sintáctica

Compiladores, Aula Nº 19

João M. P. Cardoso

Definições: Conjuntos First() e Follow()

Notação

T é terminal,

NT é não-terminal,

S é terminal ou não-terminal,

e α e β representam sequências de terminais e/ou não terminais

- Conjunto First(β)
  - Conjunto de símbolos terminais situados mais à esquerda em todas as possíveis árvores de derivação de β
  - T$\in$ First(β) se T pode aparecer como primeiro símbolo numa derivação começando em β
  - Começar pelo conceito de *NT* derivando ε:
    - NT $\rightarrow$ ε implica que NT deriva ε
    - NT $\rightarrow$ NT_1 ... NT_n e se todos os NT_i (1≤i ≤n) derivam ε implica que NT deriva ε

Definições: Regras para First()

Notação

T é terminal,
NT é não-terminal,
S é terminal ou não-terminal,
e α e β representam sequências de terminais e/ou não terminais

1) T∈ First(T)
2) First(S) ⊆ First(S β)
3) NT deriva ε implica:
   First(β) ⊆ First(NT β)
4) NT → S β implica:
   First(S β) ⊆ First(NT)

Definições: Exemplo First()

- First(Term')?

Gramática

Term' → * INT Term'
Term' → / INT Term'
Term' → ε

Solução

First(Term') = {*,/}
First(* Num Term') = {*}
First(/ Num Term') = {/}
First(*) = {*}
First(/) = {/}

Definições: Conjunto First()

- Se duas ou mais produções diferentes para o mesmo símbolo não-terminal têm conjuntos First com símbolos terminais comuns então:
  - A gramática não pode ser analisada com um *parser* preditivo LL(1) sem retrocesso
    - Exemplo:
    - S $\rightarrow$ X $
    - X $\rightarrow$ a
    - X $\rightarrow$ a b
    - First(X $\rightarrow$ a) = { a }
    - First(X $\rightarrow$ a b) = { a }
    - Qual a produção a escolher quando perante o símbolo terminal a?

Definições: Conjunto Follow()

- Para o símbolo não-terminal A, Follow(A) é o conjunto dos primeiros terminais que podem vir depois de A em alguma derivação
- Regras para Follow()
  - $ $\in$ Follow(S), em que S é o símbolo início
  - Se A $\rightarrow$ αBβ é uma produção então
    First(β) $\subseteq$ Follow(B)
  - Se A $\rightarrow$ αB é uma produção então
    Follow(A) $\subseteq$ Follow(B)
  - Se A $\rightarrow$ αBβ é uma produção e β deriva ε então
    Follow(A) $\subseteq$ Follow(B)

Definições: Algoritmo para Follow()

for all nonterminals NT
  Follow(NT) = {}
Follow(S) = {$}
while Follow sets keep changing
  for all productions A → αBβ
    Follow(B) = Follow(B) ∪ First(β)
    if (β derives ε) Follow(B) = Follow(B)∪Follow(A)
  for all productions A → αB
    Follow(B) = Follow(B)∪Follow(A)

Definições: Exemplo Follow()

- Gramáticas exemplo:
  - S → X $
    X → a
    X → a b
    - Follow(S) = { $ }
    - Follow(X) = { $ }
  - S → X $
    X → “(“ X “)”
    X → ε
    - Follow(S) = { $ }
    - Follow(X) = { “)”, $ }

Parser com *Lookahead* apenas nas reduções

- **Designa-se por *parser* Simple LR: SLR(1) ou simplesmente SLR**
- Se um estado contiver: $A \rightarrow \beta \bullet$
- Reduzir de $A \rightarrow \beta$ apenas se o próximo símbolo na entrada pode seguir (*follow*) A em alguma derivação
- Gramática exemplo:

$S \rightarrow X\ \$$ (1)
$X \rightarrow a$ (2)
$X \rightarrow a\ b$ (3)

Parser sem *Lookahead*: LR(0)

		ACTION		Goto
State	a	b	$	X
s0	shift to s1	error	error	goto s3
s1	reduce(2)	S/R Conflict	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	

reduce(2) ou shift to s2

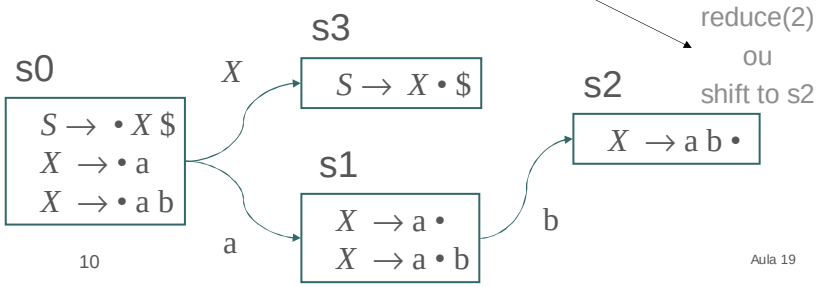


Tabela sintáctica com apenas *lookahead* nas reduções

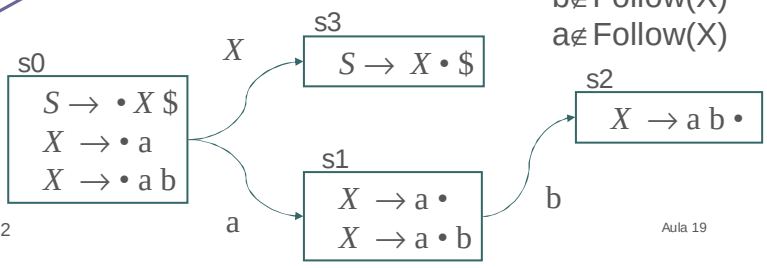
- Para cada estado
  - Transição para outro estado utilizando um símbolo terminal é um deslocamento para esse estado (*shift to sn*) (como anteriormente)
  - Transição para outro estado utilizando um símbolo não-terminal é um goto para esse estado (*goto sn*) (como anteriormente)
  - Se existir um item: $X \rightarrow \alpha \bullet$ no estado, fazer uma redução com essa produção **sempre que o símbolo corrente (T) na entrada possa seguir (follow) X em alguma derivação**
- Elimina acções de redução inúteis

Nova tabela sintáctica

		ACTION		Goto
State	**a**	**b**	**$**	**X**
s0	shift to s1	error	error	goto s3
s1		shift to s2	reduce(2)	
s2			reduce(3)	
s3	error	error	accept	

b nunca segue X nas derivações: resolver conflito *shift/reduce* com *shift*

b∉ Follow(X)
a∉ Follow(X)



Lookahead mais genérico

- Itens contêm informação *lookahead* potencial, resultando em mais estados no controlo de estados finitos
  - Item da forma: [A $\rightarrow \alpha \bullet X \beta$ **c**] diz
    - Próximo símbolo na entrada é **c**
    - *Parser* já consumiu α, espera analisar X β, e depois reduzir utilizando: A $\rightarrow \alpha \bullet X \beta$
- Em adição ao estado corrente na tabela sintáctica, todas as acções do *parser* são função dos símbolos de *lookahead*

Sumário

- Geradores de *Parsers* – dada uma gramática, produz um *parser*
- Técnica de análise sintéctica descendente
  - Construir automaticamente um autómato de pilha (*pushdown automata*)
  - Obter um *parser shift-reduce*
    - Controlo de estados finitos + pilha
    - Implementação baseada em tabela
- Conflitos: *Shift/Reduce*, *Reduce/Reduce*
- Uso de *lookahead* para eliminar conflitos
  - Parser SLR(1) (elimina acções de redução inúteis)
  - Parser LR(k) (uso de *lookahead* genérico)

Ideia básica para LR(1)

- Dividir estados em LR(0)
  - DFA baseado em *lookahead*
- Acção de reduzir é baseada no item e no *lookahead*

Itens LR(1)

- Itens mantêm informação em relação a:
  - Produção
  - Posição *right-hand-side* (o ponto)
  - Símbolo *lookahead*
- Item LR(1) é da forma [A $\rightarrow \alpha \bullet \beta$ T]
  - A $\rightarrow \alpha \beta$ é uma produção
  - O ponto em A $\rightarrow \alpha \bullet \beta$ denota a posição
  - T é um símbolo terminal ou o marcador de término ($)
- Item [A $\rightarrow \alpha \bullet \beta$ T] significa
  - O *parser* já analisou α
  - Se analisa β e o próximo símbolo é T então o *parser* deve reduzir com A $\rightarrow \alpha \beta$

Itens LR(1): exemplo

- Gramática
  - S → X **$**
  - X → **(**X**)**
  - X → **ε**
- Símbolos terminais
  - **‘(‘ ‘)’**
- Símbolo terminal
  - ‘$’
- Itens LR(1)

[S → • X $,)]
[S → • X $, (]
[S → • X $, $]
[S → X • $,)]
[S → X • $, (]
[S → X • $, $]
[X → • (X) ,)]
[X → • (X) , (]
[X → • (X) , $]
[X → (• X) ,)]
[X → (• X) , (]
[X → (• X) , $]
[X → (X • ,)]
[X → (X • , (]
[X → (X • , $]
[X → (X) • ,)]
[X → (X) • , (]
[X → (X) • , $]
[X → • ,)]
[X → • , (]
[X → • , $]

Criação do *parser* LR(1)

- É necessário definir funções Closure() e Goto() para itens LR(1)
- Necessário algoritmo para criar DFA
- Necessário algoritmo para criar a tabela do *parser*

Closure para LR(1)

Closure(I)
  repeat
    for all items $[A \rightarrow \alpha \bullet X \beta \quad c]$ in I
      for any production $X \rightarrow \gamma$
        for any $d \in$ First(β c)
          $I = I \cup \{ [X \rightarrow \bullet \gamma \quad d] \}$
  until I does not change
  return I

Goto para LR(1)

Goto(I, X)
  J = { }
  for any item $[A \rightarrow \alpha \bullet X \beta \quad c]$ in I
    $J = J \cup \{[A \rightarrow \alpha X \bullet \beta \quad c]\}$
  return Closure(J)

Construção do DFA LR(1)

- Começar com o item: [<S'> $\rightarrow$ • <S> $?]
  - ? É irrelevante porque nunca deslocaremos $
- Determinar o fechamento (closure) do item e formar o estado
- Seleccionar um estado I
  - for each item [A$\rightarrow \alpha$ • X β c] in I
    - find Goto(I, X)
    - if Goto(I, X) is not already a state, make one
    - Add an edge X from state I to Goto(I, X) state
- Repetir até que não haja mais adições

Criação da tabela do *parser*

- Para cada estado no DFA LR(1)
  - Transição para outro estado usando um símbolo terminal é um deslocamento para esse estado (*shift to sn*)
  - Transição para outro estado usando um símbolo não-terminal é um goto para esse estado (*goto sn*)
  - Se existir um item [**A** $\rightarrow \alpha$ • **a**] num estado, **reduzir** para o símbolo de entrada **a** com a produção **A** $\rightarrow \alpha$ (*reduce k*)

Parser Look-Ahead LR(1) ou LALR(1)

- Motivação
  - *Parser* LR(1) tem um número elevado de estados
  - Método simples para eliminar estados
- Se dois estados LR(1) são idênticos excepto no símbolo *lookahead* dos itens então juntar estados
- Resultado é um DFA LALR(1)
- Tipicamente tem muito menos estados do que LR(1)
- Pode ter mais conflitos *reduce/reduce*

Classificar uma gramática como LL(1)

- Como verificar se uma gramática é LL(1)?
  - Se a tabela sintáctica não tiver mais do que uma produção em cada célula
- Tabela sintáctica do analisador preditivo
  - Uma linha por cada não-terminal
  - Uma coluna por cada Terminal
  - Colocar produção $X \rightarrow \gamma$ na linha X, coluna T, para cada $T \in First(\gamma)$
  - Se γ pode derivar ε então colocar produção $X \rightarrow \gamma$ na linha X, coluna T, para cada $T \in Follow(X)$

Classificar uma gramática como LL(1)

Gramática:

Z → "d"

Z → X Y Z

Y → ε

Y → "c"

X → Y

X → "a"

- Colocar produção X → γ na linha X, coluna T, para cada T ∈ First(γ)
- Se γ pode derivar ε então colocar produção X → γ na linha X, coluna T, para cada T ∈ Follow(X)

Não-terminais	Terminais		
	"d"	"c"	"a"
Z			
Y			
X			

Classificar uma gramática como LL(1)

Gramática:

Z → "d"

Z → X Y Z

Y → ε

Y → "c"

X → Y

X → "a"

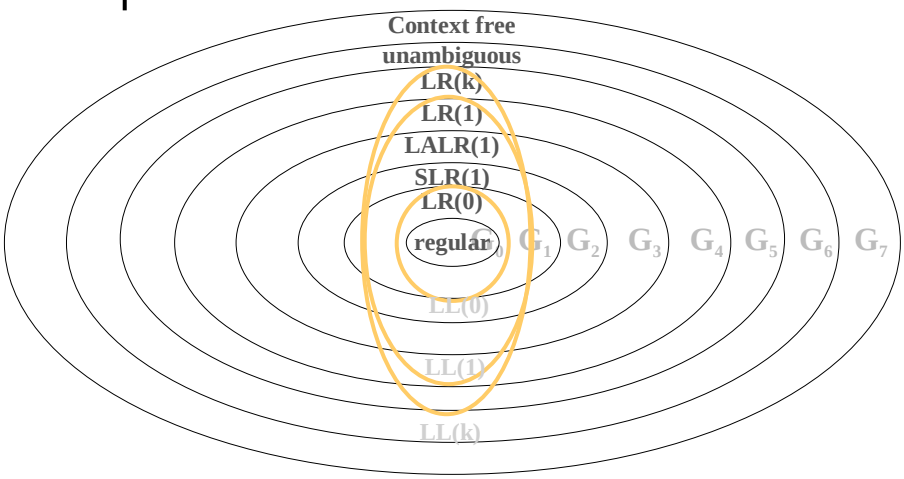
- Como verificar se uma gramática é LL(1)?
  - Se a tabela sintáctica não tiver mais do que uma produção em cada célula
- A gramática não é LL(1)

Não-terminais	Terminais		
	"d"	"c"	"a"
Z	Z → X Y Z Z → "d"	Z → X Y Z	Z → X Y Z
Y	Y → ε	Y → ε Y → "c"	Y → ε
X	X → Y	X → Y	X → "a" X → Y

Classificação de Gramáticas

- Uma gramática diz-se:
  - **LR(0)** se existir uma tabela sintáctica LR(0) sem conflitos (*reduce/reduce*, ou *shift/reduce*)
  - **SLR(1)** se existir uma tabela sintáctica SLR(1) sem conflitos (*reduce/reduce*, ou *shift/reduce*)
  - **LR(k)** se existir uma tabela sintáctica LR(k) sem conflitos (*reduce/reduce*, ou *shift/reduce*)
  - **LL(k)** se puder ser analisada por um analisador sintáctico preditivo com lookahead=k
  - …

Classificação de Gramáticas



Geradores de *Parsers*

- Geram C, http://dinosaur.compilertools.net/
  - Lex & Yacc
  - flex e bison
- Geram Java:
  - JLex e CUP

  http://www.cs.princeton.edu/~appel/modern/java/JLex/
  http://www.cs.princeton.edu/~appel/modern/java/CUP/
  - JavaCC:

  http://www.experimentalstuff.com/Technologies/JavaCC/index.html
- Lista com mais geradores de *parsers*
  - http://catalog.compilertools.net/lexparse.html
  - http://catalog.compilertools.net/java.html