204433 วิชาการแปลภาษาโปรแกรม

เมื่อเราได้ parsing table ของ grammar แบบ LR(0) ที่ได้กล่าวมาแล้ว เราจะใช้ตารางนี้ในการ parse อินพุทสตริงที่เข้ามา ก่อนอื่นเรามาทำความเข้าใจกับสิ่งที่อยู่ในตารางนี้ก่อน

แต่ละแถวของตารางคือ state แต่ละอัน และคอลัมน์ของตารางบ่ง terminal และ non-terminal ทั้งหมดที่ใช้ใน grammar

ในคอลัมน์ที่ระบุ terminal มี action อยู่สองแบบคือ shift และ reduce ตัว shift จะระบุโดยตัวอักษร s และตามด้วยตัวเลขระบุ state ที่จะ shift ไปหา ส่วน reduce จะระบุโดย production ที่จะ reduce ไปหา non-terminal ทางด้านซ้ายมือ

ในคอลัมน์ที่ระบุ non-terminal จะมี action อยู่แบบเดียวคือ goto ซึ่งจะระบุโดยตัวอักษร g และตามด้วยตัวเลขระบุ state ที่จะ goto ไป ณ state นั้น

ทุกๆครั้งที่อ่าน token แต่ละตัวเข้ามา ณ state ใด state หนึ่ง เราจะดูไปที่ตำแหน่งแถวของ state นั้นและคอลัมน์ของ token นั้น

- ถ้า action ที่ระบุเป็น shift เราจะ shift ไปที่ state ที่ระบุไว้ในตารางตำแหน่งนั้น และจะ push token นี้ลง stack
- ถ้า action ที่ระบุเป็น reduce
    - เราจะ pop symbol (terminal หรือ non-terminal) ออกจาก stack โดยจำนวน symbol ที่ pop ออกมาจะต้องเท่ากับ จำนวน symbol ที่อยู่ทาง RHS ของ production ที่จะ reduce ไปหา non-terminal ทางด้าน LHS และ symbol ที่ pop ออกมาจาก stack แต่ละตัวนั้น จะต้องเทียบเคียงได้กับ symbol ทางด้าน RHS ของ production ทุกตัว
    - push non-terminal ทางด้าน LHS ของ production ลงบน stack
    - rescan stack จาก bottom ไปจนถึง top เพื่อหาว่าหลังจาก reduce แล้วจะไปอยู่ที่ state ใด เมื่อ rescan ไปพบกับ symbol ที่เป็น non-terminal ใช้ goto action โดยที่ action แบบนี้ไม่กระทบกับสถานะของ stack ที่เป็นอยู่ มันเพียงเปลี่ยนแปลง state ไปยังที่ระบุไว้ตาม goto เท่านั้น

ขั้นตอนท้ายที่สุดที่ต้องมีการ rescan stack นั้น เราสามารถปรับปรุงขั้นตอนนี้ให้เร็วยิ่งขึ้นได้ โดยแทนที่เราจะต้อง rescan จาก bottom ของ stack เสมอ เราจะเก็บ state ที่เราอยู่ ณ ขณะที่เรากำลัง scan token เข้าไว้ใน stack ด้วย เมื่อมีการ reduce และมีการ pop symbol ออกจนครบถ้วนตาม RHS ของ production แล้ว เราจะดู state ที่ top of stack ณ ขณะนั้นและจะเริ่มต้นจาก state นี้ได้เลย โดยไม่ต้องไป rescan stack ตั้งแต่ต้นจาก bottom รูปด้านล่างนี้แสดงการ parse อินพุทสตริง ((x),y) ที่จะตรวจจับได้ด้วย LR(0) grammar นี้

((x),y)

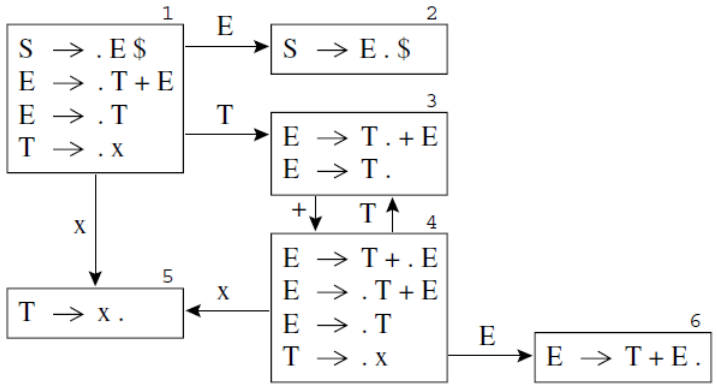
derivation	*stack*	*input*	*action*
((x),y) ←	$_1$	((x),y)	shift, goto 3
((x),y) ←	$_1(_3$	(x),y)	shift, goto 3
((x),y) ←	$_1(_3(_3$	x),y)	shift, goto 2
((x),y) ←	$_1(_3(_3x_2$	),y)	reduce $S \rightarrow id$
((**S**),y) ←	$_1(_3(_3S_7$	),y)	reduce $L \rightarrow S$
((**L**),y) ←	$_1(_3(_3L_5$	),y)	shift, goto 6
((L),y) ←	$_1(_3(_3L_5)_6$	,y)	reduce $S \rightarrow (L)$
(**S**,y) ←	$_1(_3S_7$	,y)	reduce $L \rightarrow S$
(**L**,y) ←	$_1(_3L_5$	,y)	shift, goto 8
(L,y) ←	$_1(_3L_5,_8$	y)	shift, goto 9
(L,y) ←	$_1(_3L_5,_8y_2$	)	reduce $S \rightarrow id$
(L,**S**) ←	$_1(_3L_5,_8S_9$	)	reduce $L \rightarrow L\ ,\ S$
(**L**) ←	$_1(_3L_5$	)	shift, goto 6
(**L**) ←	$_1(_3L_5)_6$		reduce $S \rightarrow (L)$
S	$_1S_4$	$	*done*

SLR grammar

พิจารณา grammar ดังต่อไปนี้

0 $S \rightarrow E\ \$$
1 $E \rightarrow T\ +\ E$
2 $E \rightarrow T$
3 $T \rightarrow x$

เราสามารถสร้าง DFA ของ grammar แบบ LR(0) เพื่อนำไปสู่การ parse ได้ดังต่อไปนี้



จาก DFA ข้างต้น เราสามารถสร้าง parsing table ได้ดังต่อไปนี้

	x	+	$	*E*	*T*
1	s5			g2	g3
2			a		
3	r2	s4,r2	r2		
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

โดยในตารางนี้แทนที่เราจะใส่ production ลงที่ตำแหน่งที่จะ reduce เราใช้สัญลักษณ์ r แทนการ reduce และตามด้วยกฎการ reduce ที่เราได้ระบุๆไว้ใน grammar เช่น ตำแหน่ง r2 หมายถึงการ reduce ด้วย production E → T (ซึ่งใน grammar ระบุเป็น กฎข้อที่ 2)

จะเห็นได้ว่า ณ ตำแหน่ง state ที่ 3 และคอลัมน์ token + เราพบว่ามีความเป็นไปได้ที่จะ shift ไปที่ state 4 หรือ reduce ด้วยกฎที่ 2 สถานการณ์เช่นนี้เรียกว่าการมี shift-reduce conflict การมีสถานการณ์แบบนี้ทำให้ grammar นี้ไม่ใช่ LR(0) อีกต่อไป เราจะแก้ไขสถานการณ์นี้โดยการใช้เกณฑ์ดังต่อไปนี้

"เราจะใส่ reduce action เข้าในแถวของ reduce state ณ คอลัมน์ที่ระบุ token ที่อยู่ใน FOLLOW ของ non-terminal ทาง LHS ของ production ที่จะ reduce เท่านั้น"

เมื่อเราใช้เกณฑ์นี้ เราจะสามารถกำจัด conflict ด้านบนโดยเลือกที่จะทำการ shift แทนที่จะเป็น reduce เพราะว่า + ไม่ใช่ token ที่อยู่ใน FOLLOW(E) ตาราง parsing ใหม่จะมีรูปแบบดังแสดงด้านล่างนี้

	x	+	$	*E*	*T*
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		

grammar ที่เราแก้ shift-reduce conflict ได้โดยเกณฑ์ข้างต้นนี้คือ grammar ที่เรียกว่า SLR (มาจาก Simple LR)

LR(1) grammar

ต่อไปเราจะมาพิจารณา grammar แบบ LR(1) โดยการสร้าง parsing table จะเริ่มจากการสร้าง DFA ที่ state จะประกอบไปด้วย item ที่จะมีนิยามต่างไปจาก item ของ LR(0) ดังต่อไปนี้

item ของ LR(1) จะประกอบไปด้วย

- production
- ด้านขวาของ production (RHS) ที่มี . (dot) ระบุตำแหน่ง top of stack

- lookahead token ซึ่งส่วนนี้เป็นส่วนที่เพิ่มขึ้นมาจาก item ของ LR(0)

แนวคิดคือ item (A →α.β, x) จะระบุว่า α อยู่ที่ top of stack และส่วนหน้าสุดของอินพุทสตริงก็คือส่วนของสตริงที่สามารถได้มาจาก βx

พิจารณา grammar ต่อไปนี้

0	$S' \rightarrow S\ \$$	3	$E \rightarrow V$
1	$S \rightarrow V\ =\ E$	4	$V \rightarrow \mathrm{x}$
2	$S \rightarrow E$	5	$V \rightarrow *\ E$

ในการสร้าง DFA เราจะเริ่มต้นจาก state ที่มี item S' → S$? และหา closure ของ item นี้ โดยกระบวนการนี้จะเหมือนกับการหา closure ของ item แบบ LR(0) แต่ว่าจะเพิ่มเติมกระบวนการดังต่อไปนี้

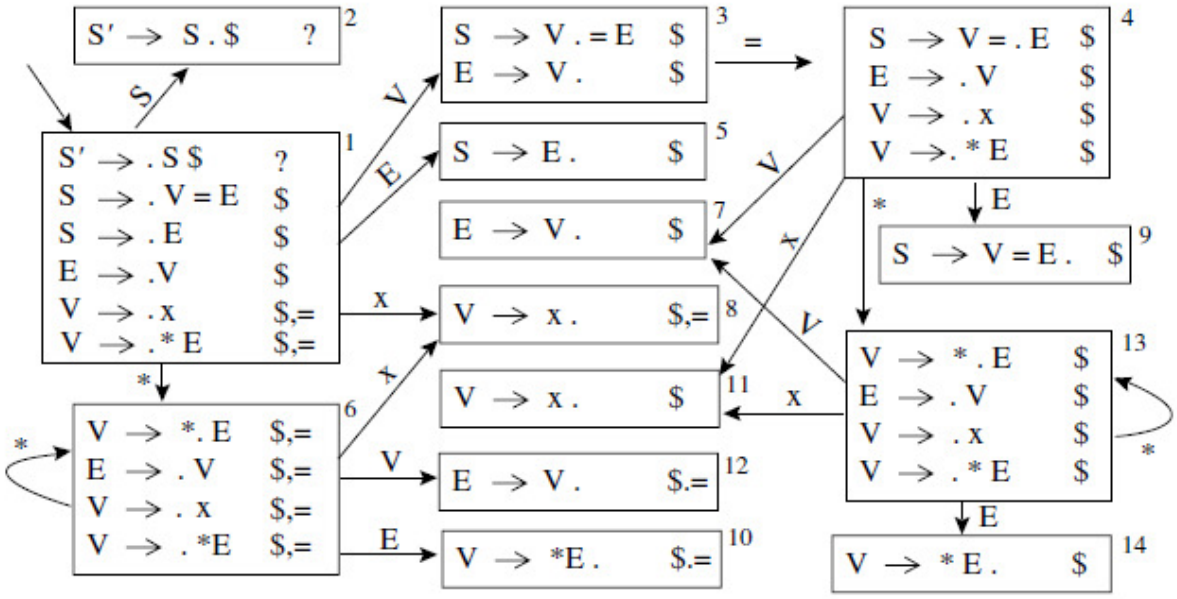
"สำหรับ item (A →α.Xβ, z) ให้เพิ่ม item (X →.γ , w) โดย w ∈ FIRST(βz) ทุกๆ production X →γ และทุกๆ w"

เราจะได้ closure เพื่อสร้าง state แรกของ DFA ตามรูปด้านล่างซ้าย และสำหรับ item ที่มีด้านซ้ายเหมือนกันแต่ lookahead ต่างกัน เราสามารถจะรวมกลุ่ม lookahead ได้ตามรูปด้านล่างขวา

$S' \rightarrow .\ S\ \$$	?
$S \rightarrow .\ V\ =\ E$	$
$S \rightarrow .\ E$	$
$E \rightarrow .\ V$	$
$V \rightarrow .\ \mathrm{x}$	$
$V \rightarrow .\ *\ E$	$
$V \rightarrow .\ \mathrm{x}$	=
$V \rightarrow .\ *\ E$	=

$S' \rightarrow .\ S\ \$$	?
$S \rightarrow .\ V\ =\ E$	$
$S \rightarrow .\ E$	$
$E \rightarrow .\ V$	$
$V \rightarrow .\ \mathrm{x}$	$, =
$V \rightarrow .\ *\ E$	$, =

สำหรับ DFA เต็มๆสำหรับ grammar นี้เป็นไปตามแผนภาพต่อไปนี้



จาก DFA ด้านบน เราจะสร้าง parsing table สำหรับ LR(1) grammar นี้ได้ดังต่อไปนี้

	x	*	=	$	*S*	*E*	*V*
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

สำหรับการหา closure ของ item ใน state I สำหรับ LR(1) grammar นั้น มีอัลกอริทึมอย่างเป็นทางการดังต่อไปนี้

Closure(I) =
 repeat
  for any item $(A \to \alpha.X\beta, z)$ in I
   for any production $X \to \gamma$
    for any $w \in$ FIRST(βz)
     $I \leftarrow I \cup \{(X \to .\gamma,\ w)\}$
 until I does not change
 return I

เราจะเห็นได้ว่า state ของ DFA ใน LR(1) grammar นั้นมีจำนวนมาก ทำให้ตาราง parsing มีขนาดใหญ่ตามไปด้วย ถ้าเราจะพิจารณายุบ state ที่มี item ทางด้านซ้ายเหมือนกันแต่มี lookahead ทางด้านขวามือที่แตกต่างกัน เราจะได้ DFA และตารางของ LALR(1) grammar (ย่อมาจาก Look-Ahead LR) โดยจาก DFA ของ LR(1) ที่เราได้พิจารณามา เราจะเห็นได้ว่าเราสามารถยุบ state คู่เหล่านี้เข้าด้วยกันได้ 6 และ 13 7 และ 12 8 และ 11 10 และ 14 ดังนั้นตาราง parsing ของ LALR(1) grammar นี้จะมีจำนวนแถวเหลือเพียงสิบแถวดังแสดงด้านล่างนี้

	x	*	=	$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			

จากที่ได้เรียนรู้ grammar ชนิดต่างๆที่ผ่านมา เราสามารถจัดกลุ่มความสัมพันธ์ของ grammar ชนิดต่างๆได้ตามแผนภาพนี้