204433 วิชาการแปลภาษาโปรแกรม

การแก้ ambiguity โดย precedence และ associativity

พิจารณา ambiguous grammar ที่เราได้เคยประสบมาต่อไปนี้

$E \rightarrow \text{id}$
$E \rightarrow \text{num}$
$E \rightarrow E * E$
$E \rightarrow E / E$
$E \rightarrow E + E$
$E \rightarrow E - E$
$E \rightarrow (E)$

เราได้เรียนรู้มาว่าการแก้ ambiguous grammar นี้ เราจะต้องเขียน grammar นี้ใหม่โดยเพิ่ม non-terminal เข้ามา โดยตัวอย่างหนึ่งของ grammar ที่มีการเขียนใหม่เพื่อแก้ ambiguous grammar นี้ก็คือ grammar ดังต่อไปนี้

$E \rightarrow E + T$	$T \rightarrow T * F$	$F \rightarrow \text{id}$
$E \rightarrow E - T$	$T \rightarrow T / F$	$F \rightarrow \text{num}$
$E \rightarrow T$	$T \rightarrow F$	$F \rightarrow (E)$

ต่อไปนี้เราจะมาแนะนำวิธีการแก้ ambiguity ที่ทำได้ง่ายกว่าการเขียน grammar ใหม่ แต่วิธีนี้เราไม่ควรใช้เป็นวิธีหลัก ถ้าเราสามารถเขียน grammar ได้ใหม่ให้ไม่มี ambiguity ได้ง่าย ไม่ซับซ้อน เราควรจะกระทำการไปในแนวทางนั้น แต่ grammar บางประเภทที่ดูจะเป็น "ธรรมชาติ" มากกว่าถ้าเขียนในรูปแบบที่มี ambiguity เราอาจจะเลือกมาใช้วิธีดังจะได้กล่าวนี้ในการแก้ ambiguity ได้

ตารางด้านล่างนี้เป็นตาราง parsing LR(1) ของ ambiguous grammar ด้านบน จะเห็นได้ว่าตารางนี้เต็มไปด้วย shift-reduce conflict

	id	num	+	-	*	/	(	)	$	*E*
1	s2	s3					s4			g7
2			r1	r1	r1	r1		r1	r1	
3			r2	r2	r2	r2		r2	r2	
4	s2	s3					s4			g5
5								s6		
6			r7	r7	r7	r7		r7	r7	
7			s8	s10	s12	s14			a	
8	s2	s3					s4			g9
9			s8,r5	s10,r5	s12,r5	s14,r5		r5	r5	
10	s2	s3					s4			g11
11			s8,r6	s10,r6	s12,r6	s14,r6		r6	r6	
12	s2	s3					s4			g13
13			s8,r3	s10,r3	s12,r3	s14,r3		r3	r3	
14	s2	s3					s4			g15
15			s8,r4	s10,r4	s12,r4	s14,r4		r4	r4	

ถ้าเรามีเกณฑ์ในการกำจัด conflict เหล่านี้ เราจะได้ตาราง parsing LR(1) ซึ่งปราศจาก ambiguity และสามารถนำไปใช้ parse expression ทางคณิตศาสตร์ที่ระบุด้วย grammar แบบ ambiguous ตอนเริ่มต้นได้ โดยไม่เกิดปัญหาการมี parse tree มากกว่าหนึ่งรูปแบบสำหรับ input string อันใด อันหนึ่งอีกต่อไป เกณฑ์ดังกล่าวคือการใช้ precedence และ associativity ของ operator เข้ามาเป็นตัวแก้ไข conflict

พิจารณา state ที่ 13 เราจะเห็นได้ว่ามี conflict ที่ token + ถ้าเราลองดู item ที่เกี่ยวข้องที่ทำให้เกิด conflict นี้ เราจะพบ item สองอันใน state นี้คือ

$E \rightarrow E * E .$	$+$
$E \rightarrow E . + E$	(any)

โดย (any) คือ lookahead ใดๆที่เป็นไปได้ การตัดสินใจว่าจะ shift หรือ reduce ขึ้นกับว่าเราจะให้ precedence ของ * หรือ + มากกว่ากัน ถ้า * มี precedence มากกว่า เราจะต้อง reduce ที่ state นี้ ที่ lookahead + เพราะถ้าเราเลือกที่จะ shift เราจะได้ว่า E * E จะ reduce ได้ก็ต่อเมื่อ E + E ได้ทำการ reduce มาแล้ว ซึ่งถ้าเป็นเช่นนี้จริงๆแสดงว่า + มี precedence มากกว่า ในทางกลับกัน พิจารณา shift-reduce conflict ที่เกิดจาก E -> E + E. * และ E -> E. * (any) ในกรณีนี้เราต้องเลือกที่จะ shift แทนที่จะ reduce เพื่อคงความเป็น precedence ที่เหนือกว่าของ * เทียบกับ + (นั่นคือ shift ไปเพื่อให้ E * E ได้รับการ reduce มาก่อนก่อนที่จะมาถึง E + E

นอกจากนี้ยังมีสถานการณ์ shift-reduce conflict ที่เกิดจาก operator แบบเดียวกันดังตัวอย่างด้านล่างนี้

$E \rightarrow E + E .$	$+$
$E \rightarrow E . + E$	(any)

ในกรณีนี้ถ้าเราเลือกที่จะ reduce เราจะได้ว่าการประมวลผลมีลำดับซ้ายไปขวา (left associativity) แต่ถ้าเราเลือกที่จะ shift เราจะได้ว่าการประมวลมีลำดับจากขวามาซ้าย (right associativity)