รายงานวิจัย

เรื่อง

โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์

โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ฉบับภาษา (คณิตศาสตร์) ฉบับสมบูรณ์

โดย อิว ไอยราภาญจนกุล

เรื่อง โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์

ผู้วิจัย *อิว ไอยราคาญจนกุล

บทคัดย่อ

โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ เป็นโปรแกรมที่ทำขึ้นเพื่อผลิตตัววิเคราะห์วากยสัมพันธ์ ที่สามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1) การวิเคราะห์กระจายของตัววิเคราะห์วากยสัมพันธ์เป็นแบบบนลงล่าง (top-down parsing) ข้อมูลเข้าของโปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ได้แก่ ไวยากรณ์ LL(1) ที่สามารถเพิ่มสัญลักษณ์กระทำ (action symbol) เข้าไปในโพรดักชัน (production) เพื่อให้สามารถติดต่อกับตัววิเคราะห์ความหมาย (semantic analyzer) ได้

ตัววิเคราะห์วากยสัมพันธ์ที่ผลิตได้ เป็นชุดคำสั่งย่อยที่เขียนด้วยภาษา C ชุดคำสั่งย่อยดังกล่าวจะเรียกใช้ชุดคำสั่งย่อยอื่นอีก 3 ชุด ที่ผู้ใช้ต้องเขียนเพิ่มเติมเอง อันได้แก่ ตัววิเคราะห์ศัพท์ (lexical analyzer) ตัววิเคราะห์ความหมาย และตัวจัดกระทำความผิดพลาด (error handler)

---

* อาจารย์ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่

A Syntax Analyzer Generator

By *Mr. Iew Ayaragarnchanakul

Abstract

A Syntax Analyzer Generator generates a syntax analyzer which can analyze language specified by an LL(1) grammar. The analysis method used employs a form of top-down parsing. Input to the generator consists of an LL(1) grammar which can have action symbols in its production. Action symbol are used for communication with semantic analyzer.

The analyzer is a C language function. It calls another 3 functions, which perform the tasks of lexical analyzer, semantic analyzer and error handler, which must be created by user.

---

* Department of Mathematics, Faculty of Science, Prince of Songkla University, HAT YAI campus, Songkhla, Thailand.

สารบาญ

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การวิจัยเกี่ยวกับการสร้างตัวแปลชุดคำสั่ง ในประเทศไทย ยังไม่แพร่หลายเท่าที่ควร เหตุผลประการหนึ่งก็คือ ความยากลำบากในการเขียนโปรแกรม จากการที่ผู้วิจัยได้ศึกษา และทดลองสร้างตัวแปลชุดคำสั่งขนาดเล็ก[1] เพื่อแปลภาษาง่าย ๆ พบว่าเป็นไปได้ที่จะสร้างตัวแปลชุดคำสั่งเพื่อการค้าหรือสร้างตัวแปลชุดคำสั่งซึ่งแปลโปรแกรมที่เขียนด้วย อักขระไทย เพื่อใช้ภายในประเทศ จากประสบการณ์ของผู้วิจัยพบว่า ในขั้นตอนวิเคราะห์วากยสัมพันธ์ของตัวแปลชุดคำสั่งนั้น เมื่อมีการเปลี่ยนไวยากรณ์ของภาษาที่จะแปล ตัววิเคราะห์วากยสัมพันธ์จะเปลี่ยนแปลงตามไปด้วย ซึ่งถ้าใช้วิธีสร้างตัววิเคราะห์วากยสัมพันธ์ด้วยมือ จะต้องเสียเวลาอย่างมาก จึงน่าจะมีโปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ด้วยเหตุนี้เองผู้วิจัยจึงทำการวิจัยเรื่อง "โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์"

1.2 วัตถุประสงค์

เพื่อสร้างโปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ซึ่งจะเป็นประโยชน์ในการสร้างตัวแปลชุดคำสั่งหรืองานอื่น ๆ ที่ต้องการวิเคราะห์วากยสัมพันธ์ และใช้เป็นเครื่องมือในการศึกษา

1.3 ขอบเขตของการวิจัย

ตัววิเคราะห์วากยสัมพันธ์ที่ผลิตได้ สามารถวิเคราะห์วากยสัมพันธ์ ของภาษาที่ผลิตโดยไวยากรณ์ LL(1) การวิเคราะห์จะทำในลักษณะ การวิเคราะห์กระจายบนลงล่าง และอนุญาตให้เพิ่มสัญลักษณ์กระทำเข้าไปในโพรดักชันได้ เพื่อให้สามารถติดต่อกับตัววิเคราะห์ความหมาย สำหรับตัววิเคราะห์ศัพท์ ตัววิเคราะห์ความหมาย และตัวจัดกระทำความผิดพลาด ผู้ใช้ต้องสร้างเอง

1.4 อุปกรณ์ในการทำวิจัย

เครื่องคอมพิวเตอร์และตัวแปลชุดคำสั่งที่ใช้ในการวิจัย ได้แก่

- ไมโครคอมพิวเตอร์ IBM/PC XT compatible ทำงานภายใต้ระบบปฏิบัติการ MS-DOS version 3.3 ตัวแปลชุดคำสั่งภาษา C ที่ใช้ คือ TURBO C version 2.0

- มินิคอมพิวเตอร์ VAX 11/785 ทำงานภายใต้ระบบปฏิบัติการ ULTRIX-32 version 3.1 ตัวแปลชุดคำสั่งภาษา C ที่ใช้ คือ ULTRIX C
- ไมโครคอมพิวเตอร์ AT&T 386/SX ทำงานภายใต้ระบบปฏิบัติการ UNIX System V/386 Release 3.2.2

1.5 วิธีดำเนินงานการวิจัยโดยย่อ

การวิจัยครั้งนี้ ได้ดำเนินงานโดยการออกแบบ ข้อมูลเข้า ข้อมูลออก โครงสร้างข้อมูล โครงสร้างโปรแกรม ขั้นตอนวิธี แล้วจึงเขียนโปรแกรม ทดสอบโปรแกรม และเขียนคู่มือการใช้ ตามลำดับ

1.6 ประโยชน์ที่คาดว่าจะได้รับ

ก. สามารถนำโปรแกรมในงานวิจัยครั้งนี้ ไปสร้างตัววิเคราะห์วากยสัมพันธ์ของตัวแปลชุดคำสั่ง หรืองานอื่น ๆ ที่ต้องการวิเคราะห์วากยสัมพันธ์

ข. นำไปใช้ในการเรียนการสอน เช่น วิชาการสร้างตัวแปลชุดคำสั่ง เป็นต้น

1.7 คำนิยามศัพท์

นิยาม 1 **ชุดตัวอักษร (alphabet)** คือ เซตจำกัดที่ไม่ใช่เซตว่างของสัญลักษณ์ใด ๆ

นิยาม 2 **สายอักขระ (string)** คือ สัญลักษณ์ใด ๆ หรือการนำสัญลักษณ์ใด ๆ จำนวนจำกัดมาต่อกัน หรือไม่มีสัญลักษณ์ใด ๆ เลย ในกรณีนี้เรียกว่า **สายอักขระว่าง (empty string)** ในรายงานฉบับนี้จะใช้เครื่องหมาย "ϵ" แทนสายอักขระว่าง

นิยาม 3 **สายอักขระย่อย (substring)** คือ สัญลักษณ์ใด ๆ ในสายอักขระ หรือกลุ่มของสัญลักษณ์ใด ๆ ในสายอักขระที่อยู่ติดกัน

นิยาม 4 **สายอักขระบนเซต A (string over set A)** คือ สายอักขระที่ประกอบด้วยสัญลักษณ์ซึ่งเป็นสมาชิกของเซต A เท่านั้น

นิยาม 5 **ทรานสิทีฟโคสเซอของเซต** A (transitive closure of set A) คือ เซตอนันต์ของสายอักขระบนเซต A ที่เป็นไปได้ทั้งหมด ยกเว้นสายอักขระว่าง เขียนแทนด้วย A^+

นิยาม 6 **คลีนโคสเซอของเซต** A (Kleene closure of set A) คือ เซตอนันต์ของสายอักขระบนเซต A ที่เป็นไปได้ทั้งหมด รวมทั้งสายอักขระว่าง เขียนแทนด้วย A^* นั่นคือ $A^* = \{\Phi\} \cup A^+$

นิยาม 7 **ภาษา** (language) คือ เซตย่อยของคลีนโคสเซอของชุดตัวอักษร ภาษาที่ใช้อธิบายภาษาอื่น เรียกว่า **อภิภาษา** (meta language) และภาษาที่ถูกอธิบาย เรียกว่า **ภาษาจุดหมาย** (object language)

นิยาม 8 **ชุดตัวอักษรสิ้นสุด** (terminal alphabet) คือ เซตจำกัดที่ไม่ใช่เซตว่างของสัญลักษณ์ทั้งหมดในภาษาจุดหมาย

นิยาม 9 **ชุดตัวอักษรไม่สิ้นสุด** (nonterminal alphabet) คือ เซตจำกัดที่ไม่ใช่เซตว่างของสัญลักษณ์ทั้งหมดในอภิภาษา

นิยาม 10 **สัญลักษณ์สิ้นสุด** (terminal symbol) คือ สมาชิกใด ๆ ของชุดตัวอักษรสิ้นสุด

นิยาม 11 **สัญลักษณ์ไม่สิ้นสุด** (nonterminal symbol) คือ สมาชิกใด ๆ ของชุดตัวอักษรไม่สิ้นสุด

นิยาม 12 **โพรดักชัน** คือ กฎในการเปลี่ยนสายอักขระย่อยในสายอักขระใด ๆ ในที่นี้จะเขียนอยู่ในรูป a -> b เมื่อ a เป็นสายอักขระซ้ายมือ และ b เป็นสายอักขระขวามือ กฎดังกล่าว มีความหมายว่า ให้นำสายอักขระขวามือไปแทนสายอักขระย่อย ซึ่งเท่ากับสายอักขระซ้ายมือ ในสายอักขระที่ต้องการเปลี่ยน กรณีที่มีหลายโพรดักชันซึ่งสายอักขระทางซ้ายมือมีค่าเท่ากัน สามารถเขียนย่อรวมกันได้โดยใช้เครื่องหมาย "|" คั่นสายอักขระขวามือ เช่น มีโพรดักชันสองโพรดักชัน คือ a -> b และ a -> c สามารถเขียนย่อรวมกันได้เป็น a -> b|c เป็นต้น

นิยาม 13 กำหนดให้ N เป็น ชุดอักษรไม่สิ้นสุด, T เป็น ชุดอักษรสิ้นสุด, S เป็นสัญลักษณ์ไม่สิ้นสุดซึ่งเรียกแตกต่างจากสัญลักษณ์ไม่สิ้นสุดตัวอื่น ๆ ว่า **สัญลักษณ์เริ่มต้น** (starting symbol), P เป็นเซตของโพรดักชัน ที่สายอักขระซ้ายมือและสายอักขระขวามือ เป็นสายอักขระบนเซต $(N \cup T)^*$ โดยที่สายอักขระซ้ายมือต้องมีสัญลักษณ์ไม่สิ้นสุดอย่างน้อยหนึ่งตัวเสมอ และ $N \cap T$ เป็นเซตว่าง เราเรียกสัญการณ์ 4 สิ่งที่เป็นอันดับ (N,T,S,P) ว่า **ไวยากรณ์** (grammar)

นิยาม 14 G=(N,T,S,P) เป็น**ไวยากรณ์ไม่พึ่งบริบท** (context-free grammar) ถ้าทุก ๆ โพรดักชันมีข้อจำกัด ดังนี้ $u \rightarrow v$ เมื่อ $u \in N$ และ $v \in (N \cup T)^*$

นิยาม 15 ให้ G=(N,T,S,P) เป็นไวยากรณ์ และ a, b, c, r, t,u เป็นสมาชิกของเซต $(N \cup T)^*$ เราจะเรียกว่า "u **อนุพัทธ์** r **ทันที**" เขียนแทนด้วย $u \Rightarrow r$ ก็ต่อเมื่อ u = cat, r = cbt และ $a \rightarrow b \in P$

นิยาม 16 ให้ G=(N,T,S,P) เป็นไวยากรณ์ และ $w_0, w_1, \ldots, w_{n-1}, w_n$ เป็นสมาชิกของเซต $(N \cup T)^*$ เมื่อ $u = w_0$ และ $r = w_n$ เราจะเรียกว่า "u **อนุพัทธ์** r **ศูนย์ครั้งหรือมากกว่า**" หรือ " r **อนุพัทธ์ศูนย์ครั้งหรือมากกว่าจาก** u" เขียนแทนด้วย $u \Rightarrow^* r$ ก็ต่อเมื่อ $u = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{n-1} \Rightarrow w_n = r$ โดยที่ $n \geq 0$ และเรียกว่า "u **อนุพัทธ์** r" หรือ "r **อนุพัทธ์จาก** u" เขียนแทนด้วย $u \Rightarrow^+ r$ ก็ต่อเมื่อ $u = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_{n-1} \Rightarrow w_n = r$ โดยที่ $n > 0$

นิยาม 17 ให้ G=(N,T,S,P) เป็นไวยากรณ์ และ $S \Rightarrow^* w$ เราจะเรียก w ว่า **กึ่งประโยค** (sentential form) ถ้า w เป็นสายอักขระบนคลีนโคสเซอของเซต $(N \cup T)$ และเรียก w ว่า **ประโยค** (sentence) ถ้า w เป็นสายอักขระบนคลีนโคสเซอของเซต T และเรียกเซตของประโยคว่า **ภาษาที่ผลิตโดยไวยากรณ์** G (language generated by grammar G)

บทที่ 2

วรรณกรรมที่เกี่ยวข้อง

การศึกษาค้นคว้าเกี่ยวกับ โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ในประเทศไทยนั้น ผู้วิจัยยังไม่พบว่าได้มีผู้ใดทำวิจัยในเรื่องนี้ สำหรับในต่างประเทศ ได้มีการศึกษาค้นคว้าและสร้างโปรแกรมในลักษณะนี้อย่างต่อเนื่องมาตลอด โปรแกรมที่รู้จักกันอย่างแพร่หลาย ได้แก่ โปรแกรม Yacc (Yet another compiler compiler)[4] ซึ่งสร้างโดย Stephen C. Johnson ในปี ค.ศ. 1975 ที่ Bell Laboratories ประเทศสหรัฐอเมริกา โปรแกรมดังกล่าวได้ถูกบรรจุไว้เป็นโปรแกรมอรรถประโยชน์ (utility program) ของระบบปฏิบัติการ UNIX โปรแกรม Yacc เขียนด้วยภาษา C และผลิตชุดคำสั่งย่อยภาษา C ซึ่งสามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LALR (1) ส่วนโปรแกรมอื่น ๆ เช่น โปรแกรม HLP84 (Helsinki language processor)[6] และโปรแกรม GAG (Generator based on attribute grammars)[5] เป็นต้น

โปรแกรม HLP84 ถูกสร้างครั้งแรก ในปี ค.ศ. 1978 ที่มหาวิทยาลัย Helsinki โดย Räihä K.-J. และถูกพัฒนาใหม่อีกครั้งในปี ค.ศ. 1984 โปรแกรมดังกล่าวทำงานบนเครื่อง Burroughs B7800 เขียนด้วยภาษา Pascal และผลิตชุดคำสั่งย่อยภาษา Pascal ซึ่งสามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LALR(1)

โปรแกรม GAG สร้างโดย Kastens, Hutt และ Zimmermann ในปี ค.ศ. 1982 ที่มหาวิทยาลัย Karlsruhe ทำงานบนเครื่อง Siemens 7.760 เขียนด้วยภาษา Standard Pascal และผลิตชุดคำสั่งย่อยภาษา Standard Pascal ซึ่งสามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LALR(1)

สำหรับโปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1) นั้น มีผู้ศึกษาค้นคว้า และสร้างโปรแกรมเช่นกัน เช่น โปรแกรม Coco (Compiler compiler)[8] ซึ่งสร้างโดย Rechenberg และ Mössenböck ที่มหาวิทยาลัย Linz ในปี ค.ศ. 1983 เขียนด้วยภาษา Modula-2 ใช้บนเครื่องไมโครคอมพิวเตอร์ และผลิตชุดคำสั่งย่อยภาษา Modula-2

โปรแกรมที่กล่าวมาทั้งหมดนั้นได้สร้างลักษณะเฉพาะ (attribute) ไว้ให้ใช้ด้วย เช่น โปรแกรม Yacc ลักษณะเฉพาะจะแทนด้วย $$, $1, $2, ... เป็นต้น และอนุญาตให้ผู้ใช้เขียนคำสั่ง เพื่อใช้ลักษณะเฉพาะดังกล่าวได้ ลักษณะเฉพาะจะถูกนำไปใช้ในตัววิเคราะห์ความหมาย

สำหรับตัววิเคราะห์ศัพท์ บางโปรแกรมจะผลิตให้ เช่น โปรแกรม HLP84 เป็นต้น แต่ส่วนใหญ่ผู้ใช้จะต้องสร้างเอง เช่น โปรแกรม Yacc ผู้ใช้จะต้องสร้างเอง หรือเชื่อมโยงกับชุดคำสั่งย่อยที่ผลิตได้จากโปรแกรม Lex[7]

สำหรับโปรแกรมที่จะสร้างในการวิจัยครั้งนี้ เขียนด้วยภาษา C และผลิตชุดคำสั่งย่อยภาษา C ซึ่งสามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1) และอนุญาตให้ใส่สัญลักษณ์กระทำเพิ่มเติมเข้าไปในโพรดักชันได้ แต่ไม่อนุญาตให้เขียนคำสั่งอื่น ๆ สัญลักษณ์กระทำใช้เพื่อติดต่อกับตัววิเคราะห์ความหมาย สำหรับตัววิเคราะห์ศัพท์ สามารถเชื่อมโยงกับชุดคำสั่งย่อยที่ผลิตได้จากโปรแกรม Lex โปรแกรมที่จะสร้าง สามารถใช้ได้กับไมโครคอมพิวเตอร์ IBM PC/XT ซึ่งใช้ระบบปฏิบัติการ MS-DOS หรือ PC-DOS มินิคอมพิวเตอร์ VAX 11/785 ซึ่งใช้ระบบปฏิบัติการ ULTRIX ซึ่งเป็นระบบปฏิบัติการแบบเดียวกันกับ UNIX และ ไมโครคอมพิวเตอร์ AT&T 386/SX ซึ่งใช้ระบบปฏิบัติการ UNIX System V/386 Release 3.2.2

บทที่ 3

การวิเคราะห์วากยสัมพันธ์

วากยสัมพันธ์ของภาษาคอมพิวเตอร์ระดับสูงส่วนใหญ่ สามารถอธิบายและวิเคราะห์ได้ด้วยไวยากรณ์ไม่พึ่งบริบท ดังนั้นไวยากรณ์ที่จะกล่าวถึง จะหมายถึงไวยากรณ์ไม่พึ่งบริบท การวิเคราะห์สายอักขระใด ๆ ว่าเป็นประโยคของภาษาที่ผลิตโดยไวยากรณ์ จะต้องพิสูจน์ให้ได้ว่า สายอักขระนั้นอนุพัทธ์จากสัญลักษณ์เริ่มต้น มิฉะนั้นจะไม่เป็นประโยค หรือผิดวากยสัมพันธ์

พิจารณาไวยากรณ์ ต่อไปนี้

```
<expr> -> <expr> + <expr>
        | <expr> * <expr>                    (3.1)
        | ( <expr> )
        | id
```

ไวยากรณ์ (3.1) เขียนเฉพาะโพรดักชันเท่านั้น สัญลักษณ์ที่เริ่มต้นด้วยเครื่องหมาย "<" และปิดท้ายด้วยเครื่องหมาย ">" เป็นสัญลักษณ์ไม่สิ้นสุด นอกนั้นจะเป็นสัญลักษณ์สิ้นสุด สัญลักษณ์ไม่สิ้นสุดตัวแรกของโพรดักชันแรก เป็นสัญลักษณ์เริ่มต้น ไวยากรณ์ที่กล่าวถึงในบทนี้ จะใช้ตามข้อตกลงนี้

สายอักขระ (id+id) เป็นประโยคของภาษาที่ผลิตโดยไวยากรณ์ (3.1) เพราะ <expr> $\stackrel{+}{\Rightarrow}$ (id+id) แต่สายอักขระ +id*id ไม่เป็นประโยคของภาษาที่ผลิตโดยไวยากรณ์ (3.1) เพราะไม่สามารถทำให้ <expr> อนุพัทธ์ +id*id

การวิเคราะห์วากยสัมพันธ์ ทำได้หลายแบบ เช่น เริ่มที่สัญลักษณ์เริ่มต้น แล้วเลือกโพรดักชันใดโพรดักชันหนึ่ง ทำอนุพัทธ์ทันที จะได้สายอักขระซึ่งเป็นกึ่งประโยค (หรือประโยค) จากกึ่งประโยคที่ได้เลือกโพรดักชันใดโพรดักชันหนึ่ง ทำอนุพัทธ์ทันที เช่นนี้ไปเรื่อย ๆ จนกว่าจะได้ประโยคซึ่งตรงกับสายอักขระที่ต้องการวิเคราะห์ หรือปฏิเสธสายอักขระที่นำมาวิเคราะห์ว่าไม่เป็นประโยค การวิเคราะห์ในลักษณะนี้ เรียกว่า การวิเคราะห์กระจายบนลงล่าง และเรียกชุดคำสั่งที่ใช้ในการวิเคราะห์ว่า ตัววิเคราะห์วากยสัมพันธ์ หรือ ตัววิเคราะห์กระจาย (parser) มีการวิเคราะห์กระจายอีกแบบหนึ่งที่ทำในลักษณะตรงกันข้าม เรียกว่า การวิเคราะห์กระจายล่างขึ้นบน (bottom-up parsing)

ในบทนี้จะกล่าวถึงตัววิเคราะห์กระจาย ที่ใช้ในการวิจัยครั้งนี้เท่านั้น ตัววิเคราะห์กระจายดังกล่าวสามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1)

3.1 ไวยากรณ์ LL(1)

ไวยากรณ์ LL(1) เป็นไวยากรณ์ที่มีข้อกำหนด [9] ดังนี้ สำหรับโพรดักชันใด ๆ ที่อยู่ในรูป $A \rightarrow w_0 \mid w_1 \mid \ldots \mid w_n$ จะต้องได้ว่า $FIRST(w_i) \cap FIRST(w_j)$ ต้องเป็นเซตว่างสำหรับทุก ๆ $i \neq j$ และถ้า $w_i \Rightarrow^* \varepsilon$ แล้ว $FIRST(w_j) \cap FOLLOW(A)$ ต้องเป็นเซตว่าง สำหรับทุก ๆ $j \neq i$

เมื่อ $FIRST(z) = \{ a \mid z \Rightarrow^* ay \}$

$FOLLOW(A) = \{ b \mid S \Rightarrow^* xAby$ และถ้า $S \Rightarrow^* xA$ แล้ว b จะเป็นเครื่องหมายสิ้นสุดสายอักขระที่จะนำมาวิเคราะห์ $\}$

โดยที่

S เป็นสัญลักษณ์เริ่มต้น

A เป็นสัญลักษณ์ไม่สิ้นสุด

a เป็นสัญลักษณ์สิ้นสุดหรือสายอักขระว่าง

b เป็นสัญลักษณ์สิ้นสุด

x, y, z, w_0, w_1, ... ,w_n เป็นสายอักขระใด ๆ บนคลีนโคสเซอของชุดตัวอักษรไม่สิ้นสุดยูเนียนกับชุดตัวอักษรสิ้นสุด

3.2 การวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1)

ภาษาที่ผลิตโดยไวยากรณ์ LL(1) สามารถวิเคราะห์วากยสัมพันธ์ ด้วยวิธีการวิเคราะห์กระจายบนลงล่าง โดยใช้ตารางวิเคราะห์กระจาย (parsing table) และกองซ้อน (stack) ช่วยในการวิเคราะห์ ขั้นตอนวิธีวิเคราะห์ทำดังนี้

ให้ $ เป็นเครื่องหมายสิ้นสุดสายอักขระที่เพิ่มเข้าไปต่อท้ายสายอักขระที่ต้องการวิเคราะห์ และถือว่าเป็นสัญลักษณ์ตัวหนึ่งในสายอักขระที่ต้องการวิเคราะห์

ก. กำหนดให้กองซ้อนว่างเปล่า

ข. พุช (push) สัญลักษณ์เริ่มต้นลงบนกองซ้อน

ค. ให้ a เป็นสัญลักษณ์ตัวแรกของสายอักขระที่ต้องการวิเคราะห์

ง. ให้ X เป็นสัญลักษณ์บนยอดกองซ้อน ถ้า X เป็นสัญลักษณ์สิ้นสุด ทำข้อ ง.1 มิฉะนั้นทำข้อ ง.2

ง.1 ถ้า X = a ป๊อป (pop) สัญลักษณ์บนยอดกองซ้อน และให้ a เป็นสัญลักษณ์ตัวต่อไปของสายอักขระที่ต้องการวิเคราะห์ มิฉะนั้นผิดวากยสัมพันธ์

ง.2 นำ X และ a ไปค้นหาโพรดักชันในตารางวิเคราะห์กระจาย ถ้าพบ นำโพรดักชันนั้นมาทำอนุพัทธ์ทันที โดยการป๊อปสัญลักษณ์บนยอดกองซ้อน แล้วพุชสัญลักษณ์ที่อยู่ในสายอักขระขวามือของโพรดักชันทั้งหมดลงบนกองซ้อน โดยพุชทีละตัว เริ่มจากตัวที่อยู่ขวามือสุดก่อน ตามด้วยตัวที่เรียงมาทางซ้ายตามลำดับ มิฉะนั้นผิดวากยสัมพันธ์

จ. ถ้ากองซ้อนไม่ว่างเปล่า กลับไปทำข้อ ง.

ฉ. ถ้า a = $ แสดงว่าสายอักขระที่นำมาวิเคราะห์เป็นประโยค มิฉะนั้นผิดวากยสัมพันธ์

หมายเหตุ กรณีผิดวากยสัมพันธ์ อาจจะหยุดการทำงาน หรือทำการกู้ (recovery) แล้ววิเคราะห์ต่อก็ได้

วิธีสร้างตารางวิเคราะห์กระจาย ทำดังนี้

พิจารณาแต่ละโพรดักชันว่าจะอยู่ตรงช่องใดในตาราง ตามหลักเกณฑ์ข้อ ก. และข้อ ข. สมมุติว่าโพรดักชันที่จะพิจารณาอยู่ในรูป A -> z เมื่อ A เป็นสัญลักษณ์ไม่สิ้นสุด และ z เป็นสายอักขระใด ๆ บนคลีนโคสเซอของชุดตัวอักษรไม่สิ้นสุดยูเนียนกับชุดตัวอักษรสิ้นสุด

ก. สำหรับทุก ๆ a ที่เป็นสมาชิกของ FIRST(z) และ a ≠ ϵ ให้เพิ่มโพรดักชัน A -> z เข้าไปในตาราง ตรงช่องที่กำหนดโดย A กับ a

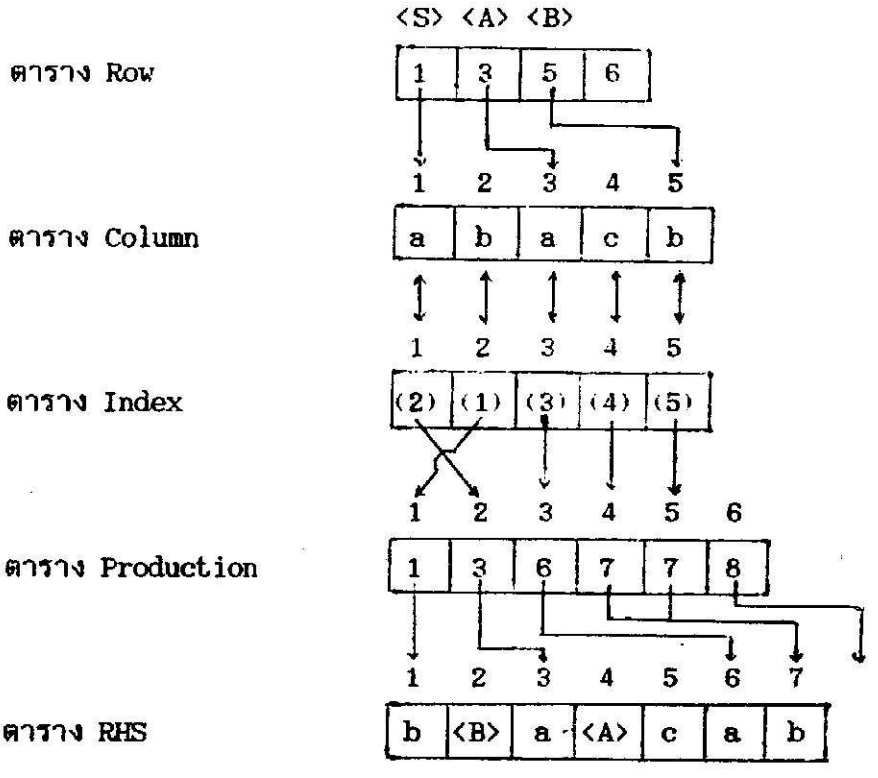
ข. ถ้า ϵ เป็นสมาชิกของ FIRST(z) ให้เพิ่มโพรดักชัน A -> z เข้าไปในตารางตรงช่องที่กำหนดโดย A กับ b สำหรับแต่ละ b ที่เป็นสมาชิกของ FOLLOW(A)

สำหรับช่องว่างในตาราง จะหมายถึงช่องที่ไม่มีโพรดักชันอยู่

ในการวิจัยครั้งนี้จะเก็บตารางวิเคราะห์กระจายแบบย่อแถว[3] โดยใช้ตาราง 5 ตาราง คือ ตาราง Row, ตาราง Column, ตาราง Index, ตาราง Production และตาราง RHS ตามลำดับ เช่น ตารางวิเคราะห์กระจาย ที่มีสัญลักษณ์ไม่สิ้นสุดกำกับแถว และสัญลักษณ์สิ้นสุดกำกับสดมภ์ ต่อไปนี้

	a	b	c	$
<S>	(2) <S> -> a<A>c	(1) <S> -> b<B>		
<A>	(3) <A> -> a		(4) <A> -> @	
<B>		(5) <B> -> b		

ถ้าเก็บแบบย่อแถว จะสามารถแทนด้วยตาราง 5 ตาราง ดังนี้



ตาราง Row มีจำนวนช่องเท่ากับจำนวนสัญลักษณ์ไม่สิ้นสุดบวกหนึ่ง ใช้เก็บหมายเลขช่องของตาราง Column โดยที่ช่องแรกเก็บค่า 1 และค่าในช่องที่ i+1 เท่ากับค่าในช่องที่ i บวกกับจำนวนโพรดักชันในแถวที่ i ของตารางวิเคราะห์กระจายก่อนการย่อแถว

ตาราง Column มีจำนวนช่องเท่ากับ จำนวนโพรดักชันในตารางวิเคราะห์กระจาย ใช้เก็บสดมภ์ของตารางวิเคราะห์กระจายก่อนการย่อแถว สดมภ์ที่เก็บ คือ สดมภ์ที่มีโพรดักชัน โดยใช้แถวเป็นหลัก คือ พิจารณาแถวแรก ถ้ามีโพรดักชัน จะเก็บสดมภ์ที่มีโพรดักชัน เรียงจากซ้ายไปขวา จากนั้นจึงพิจารณาแถวต่อ ๆ ไป

ตาราง Index มีจำนวนช่องเท่ากับ จำนวนช่องของตาราง Column ใช้เก็บหมายเลขโพรดักชันในตารางวิเคราะห์กระจาย โดยใช้แถวเป็นหลัก คือ เก็บหมายเลขโพรดักชันในแถวแรกเรียงจากซ้ายไปขวาก่อน แล้วจึงเก็บหมายเลขโพรดักชันในแถวต่อ ๆ ไป

ตาราง Production มีจำนวนช่องเท่ากับ จำนวนสัญลักษณ์ทั้งหมด (ไม่รวมสายอักขระว่าง) ที่อยู่ในสายอักขระขวามือของโพรดักชัน บวกหนึ่ง ใช้เก็บหมายเลขช่องของตาราง RHS โดยที่ช่องแรกเก็บค่า 1 และค่าในช่องที่ i+1 เท่ากับค่าในช่องที่ i บวกกับจำนวนสัญลักษณ์ในสายอักขระทางขวาของโพรดักชันที่ i

หลังจากย่อแถวแล้ว การค้นหาว่ามีโพรดักชันอยู่ในช่องของตารางวิเคราะห์กระจายหรือไม่ ทำได้โดยการนำสัญลักษณ์ไม่สิ้นสุด ไปเป็นตัวบอกหมายเลขช่องของตาราง Row สมมุติว่า คือ ช่องที่ i เช่น ตามตารางข้างบน ค่า i ของ <S>, <A> และ <B> เท่ากับ 1, 2 และ 3 ตามลำดับ จากนั้นนำสัญลักษณ์สิ้นสุดไปค้นหาในตาราง Column ตั้งแต่ช่องที่ Row[i] ถึง Row[i+1]-1 ถ้าค้นพบสัญลักษณ์สิ้นสุดดังกล่าว แสดงว่ามีโพรดักชันอยู่ในช่อง มิฉะนั้นจะเป็นช่องว่าง กรณีที่มีโพรดักชันในตาราง ค่าในตาราง Index ตรงหมายเลขช่องเดียวกันกับที่ค้นพบสัญลักษณ์สิ้นสุด จะเป็นตัวบอกหมายเลขของโพรดักชัน และสายอักขระขวามือของโพรดักชันดังกล่าวจะอยู่ในตาราง RHS เรียงจากซ้ายไปขวา ตั้งแต่ช่องที่ Production[j] ถึง Production[j+1]-1 เมื่อ j คือ หมายเลขของโพรดักชัน ยกเว้นกรณีสายอักขระว่าง ค่า Production[j] จะมากกว่า Production[j+1]-1 แสดงว่าไม่มีสัญลักษณ์ใดเลย

3.3 การเกิดความขัดแย้ง

ไวยากรณ์ [2] ต่อไปนี้

```
<S> -> i<C>t<S><E>
     : b
<C> -> a
<E> -> e<S>
     : @
```

ไม่เป็นไวยากรณ์ LL(1) ถ้านำมาสร้างตารางวิเคราะห์กระจาย ตามหัวข้อ 3.2 จะได้ตาราง ดังนี้

	i	t	b	e	a	$
<S>	<S> -> i<C>t<S><E>		<S> -> b			
<C>					<C> -> a	
<E>				<E> -> e<S> <E> -> @		<E> -> @

ในตารางตรงช่องที่กำหนดโดย <E> กับ e มีสองโพรดักชัน ทำให้เกิดความขัดแย้ง (conflict) ซึ่งจะไม่มีทางเกิดกับไวยากรณ์ LL(1) ตารางตรวจสอบที่มีความขัดแย้ง คือ ตารางที่มีโพรดักชันมากกว่าหนึ่งโพรดักชันปรากฏอยู่ในช่องเดียวกัน ตารางดังกล่าวไม่สามารถใช้กับขั้นตอนวิธีวิเคราะห์ในหัวข้อ 3.2 เนื่องจากขั้นตอนวิธีวิเคราะห์ดังกล่าว ไม่มีการย้อนรอย (no backtrack) คือ ถ้ามีโพรดักชันมากกว่าหนึ่งโพรดักชันที่สายอักขระซ้ายมือ เป็นสัญลักษณ์ไม่สิ้นสุดตัวเดียวกัน เมื่อมีการทำอนุพัทธ์ทันทีโดยการแทนที่สัญลักษณ์ไม่สิ้นสุดนั้นในกึ่งประโยค ด้วยสายอักขระขวามือของโพรดักชันใดโพรดักชันหนึ่งแล้ว จะไม่มีการย้อนรอยมาแทนที่สัญลักษณ์ไม่สิ้นสุดตัวเดิมในกึ่งประโยคนั้น ด้วยโพรดักชันอื่นอีกเลย

ความขัดแย้งบางกรณีสามารถที่จะแก้ไขได้ เช่น ไวยากรณ์ข้างต้น เป็นตัวอย่างหนึ่งของการอธิบายคำสั่ง if ในภาษาคอมพิวเตอร์ระดับสูง โดยทั่วไปคำสั่ง if อาจจะมี else หรือไม่ก็ได้ และ else จะคู่กับ if ที่อยู่ใกล้กันเสมอ ในกรณีนี้สามารถแก้ไขความขัดแย้งได้ โดยเลือกโพรดักชัน <E> -> e<S> เพียงโพรดักชันเดียวใส่ในช่องที่กำหนดโดย <E> กับ e เป็นต้น แต่ในกรณีของไวยากรณ์ ต่อไปนี้

<S> -> c<A> | c<B>

<A> -> a

<B> -> b

ไม่สามารถจะแก้ไขความขัดแย้งได้ ต้องเขียนไวยากรณ์นี้ใหม่ ให้เป็นไวยากรณ์ LL(1)

3.4 สัญลักษณ์กระทำ

ในการวิจัยครั้งนี้ อนุญาตให้ใส่สัญลักษณ์กระทำเข้าไปในโพรดักชันได้ สัญลักษณ์กระทำใช้เพื่อติดต่อกับตัววิเคราะห์ความหมาย โดยที่ผู้ใช้จะต้องเขียนชุดคำสั่งย่อย ที่เป็นตัววิเคราะห์ความหมายเอง เมื่ออนุญาตให้เพิ่มสัญลักษณ์กระทำเข้าไปในโพรดักชันได้ ขั้นตอนวิธีวิเคราะห์ในหัวข้อ 3.2 ก็เปลี่ยนแต่ แก้ไขขั้นตอนวิธีข้อ ง. ใหม่ดังนี้

"ง. ให้ X เป็นสัญลักษณ์บนยอดกองซ้อน ถ้า X เป็นสัญลักษณ์สิ้นสุด ทำข้อ ง.1 แต่ถ้า X เป็นสัญลักษณ์ไม่สิ้นสุด ทำข้อ ง.2 มิฉะนั้นทำข้อ ง.3"

และเพิ่มข้อ ง.3 ดังนี้

"ง.3 เรียกใช้ชุดคำสั่งย่อยที่เป็นตัววิเคราะห์ความหมาย"

บทที่ 4

วิธีดำเนินงานการวิจัย

การวิจัยเรื่อง "โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์" มีความมุ่งหมายเพื่อสร้างโปรแกรม วิธีดำเนินงานการวิจัย จึงดำเนินงานโดยการออกแบบ ข้อมูลเข้า ข้อมูลออก โครงสร้างข้อมูล โครงสร้างโปรแกรม ขั้นตอนวิธี แล้วจึงเขียนโปรแกรม ทดสอบโปรแกรม และเขียนคู่มือการใช้ ตามลำดับ

4.1 ข้อมูลเข้า

ข้อมูลเข้าได้ออกแบบไว้ ตามคู่มือการใช้ ในภาคผนวก ค

4.2 ข้อมูลออก

ข้อมูลออกได้ออกแบบให้บรรจุอยู่ในแฟ้มข้อมูล 3 แฟ้ม คือ แฟ้มข้อมูลค่าคงที่ แฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ และแฟ้มข้อมูลรายงาน แฟ้มข้อมูลค่าคงที่และแฟ้มข้อมูลรายงาน ได้ออกแบบให้บรรจุค่าต่าง ๆ ตามคู่มือการใช้ในภาคผนวก ค สำหรับแฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ บรรจุชุดคำสั่งย่อย ชื่อ llparse เขียนด้วยภาษา C ทำหน้าที่วิเคราะห์กระจายบนลงล่าง การวิเคราะห์กระจายใช้ตารางวิเคราะห์กระจายแบบย่อแถว 5 ตาราง แทนด้วยแถวลำดับหนึ่งมิติ ชื่อ Row_Table, Column_Table , Index_Table, Production_Table และ RHS_Table และใช้กองซ้อนซึ่งแทนด้วยแถวลำดับหนึ่งมิติเช่นกัน ชื่อ llstack โดยมีตัวแปร lltop เป็นดรรชนีกำกับที่ยอดของกองซ้อน ตัวแปร llstack และ lltop ได้กำหนดไว้นอกชุดคำสั่งย่อย เพื่อให้ชุดคำสั่งย่อยอื่นสามารถอ้างถึงได้ ค่าคงที่ที่ใช้ใน llparse กำหนดไว้ในแฟ้มข้อมูลค่าคงที่ ส่วนขั้นตอนวิธีวิเคราะห์เหมือนกับที่ได้กล่าวแล้วในบทที่ 3 ชุดคำสั่งย่อยที่ออกแบบไว้ เป็นดังนี้

```
#include <stdio.h>
#include "ชื่อแฟ้มข้อมูลค่าคงที่"

int llstack[LLMAXDEPTH+1],lltop=0;

int llparse()
```

```
{ int X,a,Production_number,flag,i,j,k;

  static int Row_Table[]={ -1 /* subscript 0 not used*/
    ค่าในตาราง
    };

  static int Column_Table[]={ -1 /* subscript 0 not used*/
    ค่าในตาราง
    };

  static int Index_Table[]={ -1 /* subscript 0 not used*/
    ค่าในตาราง
    };

  static int Production_Table[]={ -1 /* subscript 0 not used*/
    ค่าในตาราง
    };

  static int RHS_Table[]={ -1 /* subscript 0 not used*/
    ค่าในตาราง
    };

 if (lltop >= LLMAXDEPTH) {
  fprintf(stderr,"Syntax Stack Overflow\n");
  exit(1);
 }
 llstack[++lltop]=LLSTART_SYM;
 a=yylex();
```

```
do {
  X=llstack[lltop];
#ifdef LLDEBUG
  printf("top of stack symbol code: %d\n",X);
#endif
  if ((X > LLTERMINAL_RANGE)&&(X<= LLNONTERMINAL_RANGE)) {
   j=Row_Table[X-LLTERMINAL_RANGE+1]-1;
   flag=1;
   for (i=Row_Table[X-LLTERMINAL_RANGE];i<=j;i++)
    if (Column_Table[i]==a) {
     Production_number=Index_Table[i];
     flag=0;
     if (lltop<=0) {
      fprintf(stderr,"Syntax Stack Underflow\n");
      exit(2);
     }
     --lltop;
#ifdef LLDEBUG
  printf("top of stack symbol is nonterminal: pop\n");
#endif
     k=Production_Table[Production_number+1];
     j=k-Production_Table[Production_number];
     for (i=1;i<=j;++i) {
      if (lltop>=LLMAXDEPTH) {
       fprintf(stderr,"Syntax Stack Overflow\n");
       exit(1);
      }
      llstack[++lltop]=RHS_Table[k-i];
```

```
#ifdef LLDEBUG
  printf("push: %d\n",RHS_Table[k-i]);
#endif
     }
     break;
    }
    if (flag) {
#ifdef LLDEBUG
  printf("syntax error\n");
#endif
     llerror(X,&a);
#ifndef LLRECOVER
  return (1);
#endif
    }
  }
  else if (X <= LLTERMINAL_RANGE) {
#ifdef LLDEBUG
  printf("top of stack symbol is terminal.\n");
#endif
   if (X==a) {
    if (lltop<=0) {
     fprintf(stderr,"Syntax Stack Underflow\n");
     exit(2);
    }
    --lltop;
#ifdef LLDEBUG
  printf("pop and get next token\n");
#endif
```

```
    a=yylex();
   }
   else {
#ifdef LLDEBUG
  printf("syntax error\n");
#endif
    llerror(X,&a);
#ifndef LLRECOVER
  return (1);
#endif
   }
  }
  else {
#ifdef LLDEBUG
  printf("top of stack symbol is action: call llaction then pop\n");
#endif
   llaction(X-LLNONTERMINAL_RANGE-1,&a);
   if (lltop<=0) {
    fprintf(stderr,"Syntax Stack Underflow\n");
    exit(2);
   }
   --lltop;
  }
 }
 while (lltop>0);
 if (a>LL_EOF) {
#ifdef LLDEBUG
  printf("syntax error\n");
#endif
```

```
  llerror(X,&a);
#ifndef LLRECOVER
  return (1);
#endif
 }
 return (0);
}
```

ชุดคำสั่งย่อยข้างต้น มีค่าสองค่าที่ออกแบบให้สามารถนิยาม (define) ได้ในช่วงการแปลด้วยตัว-ประมวลก่อน (preprocessor) คือ LLDEBUG และ LLRECOVER ค่าแรกเมื่อนิยามแสดงว่าต้องการแกะรอย (trace) ดูการวิเคราะห์กระจาย ส่วนค่าที่สองเมื่อนิยามแสดงว่าต้องการกู้ เมื่อผิดวากยสัม-พันธ์ เพื่อให้สามารถวิเคราะห์ต่อไปได้

มีชุดคำสั่งย่อย 3 ชุดที่ตัววิเคราะห์วากยสัมพันธ์เรียกใช้ คือ yylex, llaction และ llerror ทำหน้าที่เป็น ตัววิเคราะห์ศัพท์ ตัววิเคราะห์ความหมาย และตัวจัดกระทำความผิดพลาด ตามลำดับ ชุดคำสั่งย่อยเหล่านี้ ผู้ใช้จะต้องเขียนเพิ่มเติมเอง สำหรับชุดคำสั่งย่อย yylex ได้ออกแบบให้มีชื่อและชนิดตรงกับชุดคำสั่งย่อยที่ผลิตได้จากโปรแกรม Lex

4.3 โครงสร้างข้อมูล

โครงสร้างข้อมูลในหน่วยความจำที่ใช้ เป็นแบบหลายรายการโยง (multiple link list) และใช้แถวลำดับหนึ่งมิติเก็บอักขระที่ประกอบกันเป็น สัญลักษณ์ไม่สิ้นสุด สัญลักษณ์สิ้นสุด และอักขระของชื่อที่ผู้ใช้ตั้งขึ้นในส่วนตั้งชื่อให้สัญลักษณ์ เครื่องหมาย "<" และ ">" ที่เขียนเป็นตัวแรกและตัวสุดท้าย ของสัญลักษณ์ไม่สิ้นสุด และเครื่องหมายฟันหนูที่เขียนเป็นตัวแรกและตัวสุดท้าย ของสัญลักษณ์สิ้นสุด จะไม่เก็บในแถวลำดับ

แถวลำดับหนึ่งมิติที่ใช้ กำหนดด้วยประโยคคำสั่งในภาษา C ดังนี้

```
char *str_tab;
```

ขนาดของแถวลำดับ ซึ่งกำหนดด้วยตัวแปร max_str ได้ออกแบบให้ผู้ใช้กำหนดเองได้ ถ้าผู้ใช้ไม่กำหนดจะมีขนาดเท่ากับ 2048 ไบต์ ประโยคคำสั่งภาษา C ที่ใช้กำหนดเนื้อที่ของแถวลำดับดังกล่าว คือ

```
str_tab = (char *) malloc (max_str);
```

สำหรับหลายรายการโยง มีโหนด (node) ที่ใช้ต่างกันอยู่ 5 รูปแบบ คือ

ก. โหนด nln ใช้ในรายการโยง (link list) สัญลักษณ์ไม่สิ้นสุด มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct nln {
  int code,nameptr,length,alsnameptr,alslength;
  struct gn *head,*tail,*first,*last;
  char del,      /* '0' = no '1' = yes */
  active,  /* '0' = no '1' = yes */
  reach;   /* '0' = no '1' and '2' = yes */
  struct tn *FIRST,*FOLLOW;
  struct rn *RHSFIRST;
  struct nln *next;
);
```

โดยที่

code	ใช้เก็บรหัสของสัญลักษณ์ไม่สิ้นสุดที่เป็นเจ้าของโหนดนี้
nameptr	ใช้เก็บตำแหน่งในแถวลำดับ str_tab ซึ่งเก็บอักขระตัวแรก ของอักขระที่ประกอบกันเป็นสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้
length	ใช้เก็บจำนวนอักขระทั้งหมด ที่ประกอบกันเป็นสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้
alsnameptr	ใช้เก็บตำแหน่งในแถวลำดับ str_tab ซึ่งเก็บอักขระตัวแรกของชื่อ ที่ตั้งให้กับสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้
alslength	ใช้เก็บจำนวนอักขระทั้งหมดของชื่อ ที่ตั้งให้กับสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้
head	เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ตัวแรกของสายอักขระขวามือ ของโพรดักชันแรก ที่มีสัญลักษณ์ไม่สิ้นสุดซึ่งเป็นเจ้าของโหนดนี้ เป็นสายอักขระซ้ายมือ

tail เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ตัวสุดท้ายของสายอักขระขวามือ ของโพรดักชันสุดท้าย ที่มีสัญลักษณ์ไม่สิ้นสุดซึ่งเป็นเจ้าของโหนดนี้ เป็นสายอักขระซ้ายมือ

first เป็นตัวชี้ไปยังโหนด gn ซึ่งเป็นโหนดแรกของลูกโซ่ (chain) ที่เชื่อมระหว่างโหนด gn ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุด ตัวเดียวกันกับสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

last เป็นตัวชี้ไปยังโหนด gn ซึ่งเป็นโหนดสุดท้ายของลูกโซ่ ที่เชื่อมระหว่างโหนด gn ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุด ตัวเดียวกันกับสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

del ใช้เก็บสถานะของสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้ ว่าอนุพัทธ์สายอักขระว่างหรือไม่ โดยใช้ '0' แทนอนุพัทธ์ และ '1' แทนไม่อนุพัทธ์

active ใช้เก็บสถานะของสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้ ว่าอนุพัทธ์สายอักขระว่างหรืออนุพัทธ์สายอักขระที่ประกอบด้วยสัญลักษณ์ไม่สิ้นสุด หรือไม่ โดยใช้ '0' แทนอนุพัทธ์ และ '1' แทนไม่อนุพัทธ์

reach ใช้เก็บสถานะของสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้ว่าอนุพัทธ์ศูนย์ครั้งหรือมากกว่าจากสัญลักษณ์เริ่มต้น หรือไม่ โดยใช้ '0' แทนอนุพัทธ์ และ '1' กับ '2' แทนไม่อนุพัทธ์

FIRST เป็นตัวชี้ไปยังโหนด tn ซึ่งเป็นโหนดแรกของรายการโยงที่เก็บสมาชิกของ FIRST ของสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

FOLLOW เป็นตัวชี้ไปยังโหนด tn ซึ่งเป็นโหนดแรกของรายการโยงที่เก็บสมาชิกของ FOLLOW ของสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

RHSFIRST เป็นตัวชี้ไปยังโหนด rn ซึ่งเป็นโหนดแรกของรายการโยงที่เก็บหมายเลขโพรดักชันและสมาชิกของ FIRST ของสายอักขระขวามือ ของโพรดักชันที่มีสายอักขระซ้ายมือเป็นสัญลักษณ์ไม่สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

next เป็นตัวชี้ไปยังโหนดถัดไป ถ้าไม่มีเป็น NULL

ข. โหนด tln ใช้ในรายการโยงสัญลักษณ์สิ้นสุด มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct tln {
  int code,nameptr,length,alsnameptr,alslength;
  char reach;   /* '0' = no '1' and '2' = yes */
```

```
  struct tln *next;
};
```

โดยที่

code, nameptr, length, alsnameptr, alslength และ reach มีความหมายเหมือนกับโหนด nln แต่สัญลักษณ์ที่เป็นเจ้าของโหนด คือ สัญลักษณ์สิ้นสุด

next เป็นตัวชี้ไปยังโหนดถัดไป ถ้าไม่มีเป็น NULL

ค. โหนด gn ใช้เก็บรายละเอียดของสัญลักษณ์ที่อยู่ในสายอักขระขวามือของโพรดักชัน มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct gn {
  struct nln *lhsnonterm;
  struct gn *lp;
  char type;        /* 't' = terminal symbol
                       'n' = nonterminal symbol
                       'a' = action symbol
                       'e' = empty string */
  union {
   struct gn *chain;
   int action_code;
  } u;
  struct nln *nonbackptr;
  struct tln *termbackptr;
  struct gn *rp;
};
```

โดยที่

lhsnonterm เป็นตัวชี้ไปยังโหนด nln ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุด ที่เป็นสายอักขระซ้ายมือของโพรดักชัน

lp เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ตัวแรกของสายอักขระขวามือของโพรดักชันถัดไปที่มีสายอักขระซ้ายมือเหมือนกัน ถ้าไม่มีเป็น NULL

type เก็บชนิดของสัญลักษณ์ คือ 't' แทนสัญลักษณ์สิ้นสุด, 'n' แทนสัญลักษณ์ไม่สิ้นสุด, 'a' แทนสัญลักษณ์กระทำ และ 'e' แทนสายอักขระว่าง

chain เป็นตัวชี้ไปยังโหนด gn ซึ่งเป็นโหนดถัดไปที่อยู่ในลูกโซ่ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุดตัวเดียวกัน

action_code ใช้เนื้อที่เดียวกันกับ chain แต่ใช้เก็บรหัสของสัญลักษณ์กระทำ

nonbackptr เป็นตัวชี้ไปยังโหนด nln ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุดตัวเดียวกันกับสัญลักษณ์ไม่สิ้นสุดซึ่งเป็นเจ้าของโหนดนี้

termbackptr เป็นตัวชี้ไปยังโหนด tln ซึ่งเก็บสัญลักษณ์สิ้นสุดตัวเดียวกันกับสัญลักษณ์สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

rp เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ของสายอักขระขวามือตัวถัดไป ถ้าไม่มีเป็น NULL

ง. โหนด tn ใช้ในรายการโยงที่เก็บสมาชิกของ FIRST หรือ FOLLOW ของสัญลักษณ์ไม่สิ้นสุดแต่ละตัว มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct tn {
  int code;
  struct tn *next;
};
```

โดยที่

code ใช้เก็บรหัสของสัญลักษณ์ที่เป็นสมาชิกของ FIRST หรือ FOLLOW

next เป็นตัวชี้ไปยังโหนดถัดไป ถ้าไม่มีเป็น NULL

lhsnonterm เป็นตัวชี้ไปยังโหนด nln ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุด ที่เป็นสายอักขระซ้ายมือของโพรดักชัน

lp เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ตัวแรกของสายอักขระขวามือของโพรดักชันถัดไปที่มีสายอักขระซ้ายมือเหมือนกัน ถ้าไม่มีเป็น NULL

type เก็บชนิดของสัญลักษณ์ คือ 't' แทนสัญลักษณ์สิ้นสุด, 'n' แทนสัญลักษณ์ไม่สิ้นสุด, 'a' แทนสัญลักษณ์กระทำ และ 'e' แทนสายอักขระว่าง

chain เป็นตัวชี้ไปยังโหนด gn ซึ่งเป็นโหนดถัดไปที่อยู่ในลูกโซ่ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุดตัวเดียวกัน

action_code ใช้เนื้อที่เดียวกันกับ chain แต่ใช้เก็บรหัสของสัญลักษณ์กระทำ

nonbackptr เป็นตัวชี้ไปยังโหนด nln ซึ่งเก็บสัญลักษณ์ไม่สิ้นสุดตัวเดียวกันกับสัญลักษณ์ไม่สิ้นสุดซึ่งเป็นเจ้าของโหนดนี้

termbackptr เป็นตัวชี้ไปยังโหนด tln ซึ่งเก็บสัญลักษณ์สิ้นสุดตัวเดียวกันกับสัญลักษณ์สิ้นสุด ซึ่งเป็นเจ้าของโหนดนี้

rp เป็นตัวชี้ไปยังโหนด gn ซึ่งเก็บสัญลักษณ์ของสายอักขระขวามือตัวถัดไป ถ้าไม่มีเป็น NULL

ง. โหนด tn ใช้ในรายการโยงที่เก็บสมาชิกของ FIRST หรือ FOLLOW ของสัญลักษณ์ไม่สิ้นสุดแต่ละตัว มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct tn {
  int code;
  struct tn *next;
};
```

โดยที่

code ใช้เก็บรหัสของสัญลักษณ์ที่เป็นสมาชิกของ FIRST หรือ FOLLOW

next เป็นตัวชี้ไปยังโหนดถัดไป ถ้าไม่มีเป็น NULL

จ. โหนด rn ใช้ในรายการโยงที่เก็บหมายเลขโพรดักชัน และสมาชิกของ FIRST ของสายอักขระขวามือของโพรดักชัน มีโครงสร้างอธิบายด้วยภาษา C ดังนี้

```
struct rn {
   int rule,code;
   struct rn *next;
};
```

โดยที่

rule ใช้เก็บหมายเลขโพรดักชัน

code ใช้เก็บรหัสของสัญลักษณ์ที่เป็นสมาชิกของ FIRST

next เป็นตัวชี้ไปยังโหนดถัดไป ถ้าไม่มีเป็น NULL

4.4 โครงสร้างโปรแกรม

โปรแกรมที่สร้างเรียกว่า **โปรแกรม** llgen มีชุดคำสั่งย่อยที่เริ่มต้นการทำงาน ตามข้อกำหนดของภาษา C คือ ชุดคำสั่งย่อย main ซึ่งจะเรียกใช้ชุดคำสั่งย่อยอื่นอีก 13 ชุด คือ openfile, fatal_error, llparse, complete, empty, activenonterm, reachable, assigncode, find_first, find_follow, find_rhsfirst, printparser และ printheader ชุดคำสั่งย่อยดังกล่าวมีหน้าที่ต่าง ๆ ดังนี้

ก. openfile ทำหน้าที่เปิด แฟ้มข้อมูลเข้า แฟ้มข้อมูลค่าคงที่ แฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ และแฟ้มข้อมูลรายงาน นอกจากนั้นยังทำหน้าที่กำหนดขนาดของกองซ้อนที่ใช้วิเคราะห์วากยสัมพันธ์ และขนาดของแถวลำดับ str_tab ชุดคำสั่งย่อย openfile เรียกใช้ชุดคำสั่งย่อย options เพื่อตรวจสอบดูว่าผู้ใช้ใส่ค่า -s หรือ -t หรือไม่ (ดูคู่มือการใช้ในภาคผนวก ค) และถ้าผู้ใช้ไม่ใส่ชื่อแฟ้มข้อมูลเข้า จะเรียกใช้ชุดคำสั่งย่อย help เพื่ออธิบายวิธีใช้โปรแกรม llgen

ข. fatal_error ทำหน้าที่แสดงข้อความผิดพลาดขั้นร้ายแรง ที่ไม่สามารถวิเคราะห์ต่อ

ค. llparse ทำหน้าที่วิเคราะห์วากยสัมพันธ์ของข้อมูลเข้า ครั้งแรกได้สร้างชุดคำสั่งย่อยนี้ด้วยมือ หลังจากได้โปรแกรม llgen แล้ว จึงใช้โปรแกรม llgen สร้างชุดคำสั่งย่อยนี้ ไวยากรณ์ LL(1) และสัญลักษณ์กระทำ ที่ใช้เป็นข้อมูลเข้าของโปรแกรม llgen เพื่อสร้างชุดคำสั่งย่อย llparse ได้แสดงไว้ในภาคผนวก ข ชุดคำสั่งย่อย llparse เรียกใช้ชุดคำสั่งย่อยอื่นอีก 3 ชุด คือ

ค.1 yylex เป็นตัววิเคราะห์ศัพท์ ทำหน้าที่อ่านอักขระจากข้อมูลเข้า แล้วแยกอักขระ หรือกลุ่มอักขระเหล่านั้นออกเป็นโทเคน (token) และส่งรหัสของโทเคนกลับไปให้ชุดคำสั่งย่อย llparse กรณีสิ้นสุดแฟ้ม (eof) จะส่งรหัสกลับเป็น 0 แผนภาพทรานซิชัน (transition diagram) ที่ใช้เพื่อแยกโทเคน แสดงอยู่ในภาคผนวก ก ชุดคำสั่งย่อย yylex เรียกใช้ชุดคำสั่งย่อยอื่นอีก 6 ชุด คือ

ค.1.1 lexical_error ทำหน้าที่แสดงข้อความผิดพลาดทางศัพท์

ค.1.2 Getc ทำหน้าที่อ่านอักขระจากแฟ้มข้อมูลเข้า พร้อมทั้งจดจำตำแหน่งบรรทัดและสดมภ์ของอักขระที่อ่าน เพื่อนำไปแสดงเมื่อมีความผิดพลาดเกิดขึ้น

ค.1.3 Ungetc ทำหน้าที่นำอักขระใส่กลับคืนแฟ้มข้อมูลเข้า พร้อมทั้งแก้ไขตำแหน่งบรรทัดและสดมภ์ให้ถูกต้อง

ค.1.4 searchfound ทำหน้าที่ค้นหาสัญลักษณ์ในแถวลำดับ str_tab

ค.1.5 nonterminal_list ทำหน้าที่ค้นหาสัญลักษณ์ไม่สิ้นสุด ในรายการโยงสัญลักษณ์ไม่สิ้นสุด ในการค้นหาจะเรียกใช้ชุดคำสั่งย่อย searchfound ถ้าขณะนั้นสัญลักษณ์ไม่สิ้นสุดปรากฏอยู่ในส่วนตั้งชื่อให้สัญลักษณ์ ไม่ว่าจะค้นพบหรือไม่ จะส่งตัวชี้กลับไปยังชุดคำสั่งที่เรียกใช้ แต่ถ้าปรากฏอยู่ในโพรดักชัน ถ้าพบจะส่งตัวชี้ของโหนดที่ค้นพบกลับ มิฉะนั้นจะสร้างโหนดใหม่ขึ้นมา แล้วนำโหนดดังกล่าวไปเชื่อมโยงกับรายการโยง และส่งตัวชี้ของโหนดที่สร้างใหม่กลับ กรณีที่ไม่สามารถสร้างโหนดใหม่ได้ จะเรียกใช้ชุดคำสั่งย่อย fatal_error

ค.1.6 terminal_list ทำหน้าที่เหมือน nonterminal_list แต่เป็นการค้นหาสัญลักษณ์สิ้นสุดในรายการตัวโยงสัญลักษณ์สิ้นสุด

ค.2 llaction เป็นตัววิเคราะห์ความหมาย บรรจุชุดคำสั่งประจำของสัญลักษณ์กระทำแต่ละตัว

ค.3 llerror เป็นตัวจัดกระทำความผิดพลาด ทำหน้าที่แสดงข้อความผิดพลาดวากยสัมพันธ์ และทำการกู้ด้วยวิธีแพนนิคโหมด (panic mode) โดยใช้เครื่องหมาย "." เป็นสัญลักษณ์สมวาร (synchronous symbol)

ง. complete ทำหน้าที่ตรวจสอบความสมบูรณ์ของไวยากรณ์ข้อมูลเข้า โดยตรวจสอบว่ามีสัญลักษณ์ไม่สิ้นสุดตัวใดบ้าง ที่ปรากฏในสายอักขระขวามือของโพรดักชัน แต่ไม่เคยปรากฏในสายอักขระซ้ายมือของโพรดักชัน ถ้ามีจะแสดงข้อความผิดพลาด

จ. empty ทำหน้าที่ตรวจสอบว่า สัญลักษณ์ไม่สิ้นสุดตัวใดบ้าง ที่อนุพัทธ์สายอักขระว่าง ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย deletable เพื่อหาสัญลักษณ์ไม่สิ้นสุดตัวใหม่ที่อนุพัทธ์สายอักขระว่าง จนกว่าจะไม่พบ

ฉ. activenonterm ทำหน้าที่ตรวจสอบว่า สัญลักษณ์ไม่สิ้นสุดตัวใดบ้างที่ไม่สามารถอนุพัทธ์สายอักขระว่าง หรือไม่สามารถอนุพัทธ์สายอักขระที่ประกอบด้วยสัญลักษณ์สิ้นสุด ถ้าพบจะแสดงข้อความผิดพลาด ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย rhsactive เพื่อหาสัญลักษณ์ไม่สิ้นสุดตัวใหม่ที่สามารถอนุพัทธ์สายอักขระว่าง หรือสามารถอนุพัทธ์สายอักขระที่ประกอบด้วยสัญลักษณ์สิ้นสุด จนกว่าจะไม่พบ

ช. reachable ทำหน้าที่ตรวจสอบว่า สัญลักษณ์ตัวใดบ้างที่ไม่สามารถปรากฏในกิ่งประโยค ถ้ามีจะแสดงข้อความผิดพลาด ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย rhsreach เพื่อหาสัญลักษณ์ตัวใหม่ที่สามารถปรากฏในกิ่งประโยค จนกว่าจะไม่พบ

ซ. assigncode ทำหน้าที่ออกรายงานไวยากรณ์ข้อมูลเข้า และกำหนดรหัสให้กับ สัญลักษณ์สิ้นสุดและสัญลักษณ์ไม่สิ้นสุด พร้อมทั้งออกรายงานในเรื่องนี้ โดยกำหนดรหัสให้สัญลักษณ์สิ้นสุดก่อน เริ่มจากรหัส 1 แล้วเพิ่มทีละหนึ่ง รหัสสูงสุดของสัญลักษณ์สิ้นสุดจะกำหนดให้เป็นค่าคงที่ LLTERMINAL_RANGE จากนั้นจึงกำหนดให้กับสัญลักษณ์ไม่สิ้นสุด ค่าสูงสุดของสัญลักษณ์ไม่สิ้นสุดจะกำหนดให้เป็นค่าคงที่ LLNONTERMINAL_RANGE กรณีที่รหัสมีค่าเกิน 32767 จะเรียกใช้ชุดคำสั่งย่อย fatal_error นอกจากนี้ยังทำหน้าที่ออกรายงานสัญลักษณ์ไม่สิ้นสุดที่อนุพัทธ์สายอักขระว่างด้วย

ฌ. find_first ทำหน้าที่หาสมาชิกของ FIRST ของสัญลักษณ์ไม่สิ้นสุดแต่ละตัว และออกรายงานในเรื่องนี้ ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย firstfound เพื่อหาสมาชิกตัวใหม่ จนกว่าจะไม่พบ ชุดคำสั่งย่อย firstfound เรียกใช้ชุดคำสั่งย่อย inset เพื่อตรวจดูว่าเป็นสมาชิกตัวใหม่หรือไม่ และ firstinsert เพื่อเพิ่มสมาชิกตัวใหม่

ญ. find_follow ทำหน้าที่หาสมาชิกของ FOLLOW ของสัญลักษณ์ไม่สิ้นสุดแต่ละตัว และออกรายงานในเรื่องนี้ ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย followfound เพื่อหาสมาชิกตัวใหม่ จนกว่าจะไม่พบ ชุดคำสั่งย่อย followtfound เรียกใช้ชุดคำสั่งย่อย inset เพื่อตรวจดูว่าเป็นสมาชิกตัวใหม่หรือไม่ และ followinsert เพื่อเพิ่มสมาชิกตัวใหม่

ฎ. find_rhsfirst ทำหน้าที่หาสมาชิกของ FIRST ของสายอักขระขวามือของแต่ละโพรดักชัน ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย add_or_conflict เพื่อเพิ่มสมาชิกตัวใหม่ และตรวจสอบความขัดแย้ง กรณีที่มีความขัดแย้ง จะแสดงข้อความเตือนให้ผู้ใช้ทราบ

ฏ. print_parser ทำหน้าที่ผลิตตัววิเคราะห์วากยสัมพันธ์ เก็บไว้ในแฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ ชุดคำสั่งย่อยนี้เรียกใช้ชุดคำสั่งย่อย printrow, printcolumn, printindex, printproduction และ printrhs เพื่อสร้างตาราง Row_Table, Column_Table, Index_Table, Production_table และ RHS_Table ตามลำดับ

ฐ. print_header ทำหน้าที่สร้างค่าคงที่ เก็บไว้ในแฟ้มข้อมูลค่าคงที่

ทุก ๆ ชุดคำสั่งย่อยที่กล่าวมาข้างต้น เมื่อมีความผิดพลาดที่ไม่ใช่ความผิดพลาดขั้นร้ายแรงเกิดขึ้นก่อนแสดงข้อความผิดพลาด จะเรียกใช้ชุดคำสั่งย่อย accumulate_error เพื่อบวกสะสมจำนวนความผิดพลาด ซึ่งถ้าเกิน 100 จะหยุดการทำงาน

4.5 ขั้นตอนวิธี

โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ มีขั้นตอนวิธี ดังนี้

ก. เรียกใช้ชุดคำสั่งย่อย openfile

ข. กำหนดเนื้อที่ให้กับแถวลำดับ str_tab ถ้ามีเนื้อที่ไม่พอ แสดงข้อความผิดพลาดขั้นร้ายแรง แล้วเลิกทำ

ค. เรียกใช้ชุดคำสั่งย่อย llparse ถ้ามีความผิดพลาด เลิกทำ มิฉะนั้นทำข้อต่อไป

ง. เรียกใช้ชุดคำสั่งย่อย complete ถ้ามีความผิดพลาด เลิกทำ มิฉะนั้นทำข้อต่อไป

จ. เรียกใช้ชุดคำสั่งย่อย empty

ฉ. เรียกใช้ชุดคำสั่งย่อย activenonterm ถ้ามีความผิดพลาด เลิกทำ มิฉะนั้นทำข้อต่อไป

ช. เรียกใช้ชุดคำสั่งย่อย reachable ถ้ามีความผิดพลาด เลิกทำ มิฉะนั้นทำข้อต่อไป

ซ. เรียกใช้ชุดคำสั่งย่อย assigncode

ฌ. เรียกใช้ชุดคำสั่งย่อย find_first

ญ. เรียกใช้ชุดคำสั่งย่อย find_follow

ฎ. เรียกใช้ชุดคำสั่งย่อย find_rhsfirst

ฏ. เรียกใช้ชุดคำสั่งย่อย print_parser

ฐ. เรียกใช้ชุดคำสั่งย่อย print_header

ฑ. ปิดแฟ้มข้อมูลทั้งหมด

4.6 โปรแกรม

โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์เขียนด้วยภาษา C แยกเป็นส่วน ๆ มีทั้งหมด 8 ส่วน บรรจุอยู่ในแฟ้มข้อมูล ดังนี้

แฟ้มข้อมูล	จำนวนบรรทัด	บรรจุ
typenode.h	38	โครงสร้างข้อมูลที่เป็นโหนดทั้ง 5 รูปแบบ
parser.h	23	ค่าคงที่ (เป็นแฟ้มข้อมูลค่าคงที่ที่ได้จากโปรแกรม llgen)
llgen.c	268	ชุดคำสั่งย่อย help, options, openfile, accumulate_error , fatal_error และ main
scanner.c	280	ชุดคำสั่งย่อย lexical_error, Getc, Ungetc, searchfound, nonterminal_list, terminal_list และ yylex
parser.c	170	ชุดคำสั่งย่อย llparse
synerr.c	61	ชุดคำสั่งย่อย llerror
action.c	148	ชุดคำสั่งย่อย llaction

แฟ้มข้อมูล	จำนวนบรรทัด	บรรจุ
utility.c	582	ชุดคำสั่งย่อย complete, deletable, empty, rhsactive, activenonterm, rhsreach, reachable, assign_code, inset, firstinsert, firstfound, find_first, followinsert, followfound, find_follow
printab.c	487	ชุดคำสั่งย่อย add_or_conflict, find_rhsfirst, print_parser, printrow, printcolumn, printindex, printproduction, printrhs, print_header

ในการแปลและเชื่อมโยง เพื่อให้ได้โปรแกรมที่จะนำไปใช้งาน ได้ใช้โปรแกรมอรรถประโยชน์ make โดยการสร้าง makefile

makefile สำหรับ TURBO C เป็นดังนี้

```
MDL = s         # s is small model
LIB = \tc\lib   # assume turbo C libary is \tc\lib
llgen.exe : llgen.obj scanner.obj parser.obj synerr.obj action.obj \
            printab.obj utility.obj
        tlink $(LIB)\c0$(MDL) llgen.obj scanner.obj parser.obj synerr.obj \
            action.obj printab.obj utility.obj,llgen.exe,,$(LIB)\c$(MDL)
llgen.obj : typenode.h llgen.c
        tcc -c -m$(MDL) llgen.c
scanner.obj : typenode.h parser.h scanner.c
        tcc -c -m$(MDL) scanner.c
parser.obj : parser.h parser.c
        tcc -DLLRECOVER -c -m$(MDL) parser.c
synerr.obj : parser.h synerr.c
        tcc -c -m$(MDL) synerr.c
action.obj : typenode.h parser.h action.c
        tcc -c -m$(MDL) action.c
```

```
printab.obj : typenode.h printab.c
        tcc -c -m$(MDL) printab.c
utility.obj : typenode.h parser.h utility.c
        tcc -c -m$(MDL) utility.c
```

makefile สำหรับ ULTRIX C และ C ใน UNIX System V/386 เป็นดังนี้

```
llgen : llgen.o scanner.o parser.o synerr.o action.o printab.o utility.o
        cc -o llgen llgen.o scanner.o parser.o synerr.o action.o \
                printab.o utility.o
llgen.o : typenode.h llgen.c
        cc -c llgen.c
scanner.o : typenode.h parser.h scanner.c
        cc -c scanner.c
parser.o : parser.h parser.c
        cc -DLLRECOVER -c parser.c
synerr.o : parser.h synerr.c
        cc -c synerr.c
action.o : typenode.h parser.h action.c
        cc -c action.c
printab.o : typenode.h printab.c
        cc -c printab.c
utility.o : typenode.h parser.h utility.c
        cc -c utility.c
```

4.7 การทดสอบโปรแกรม

ได้ทดสอบโปรแกรม llgen โดยใช้ข้อมูลเข้าที่เป็นไวยากรณ์ของภาษา Standard Pascal ซึ่งมีโพรดักชัน 193 โพรดักชัน สัญลักษณ์สิ้นสุด 66 ตัว สัญลักษณ์ไม่สิ้นสุด 102 ตัว และอักขระที่ประกอบกัน

เป็นสัญลักษณ์ไม่สิ้นสุด สัญลักษณ์สิ้นสุด และชื่อที่ตั้งให้สัญลักษณ์ จำนวน 1,796 อักขระ ใช้เวลาวิ่ง (run time) บนเครื่อง IBM PC/XT Compatible ประมาณ 50 วินาที และบนเครื่อง AT&T 386/SX วิ่งโดยมีผู้ใช้คนเดียว (single user) ใช้เวลา ประมาณ 9 วินาที

4.8 คู่มือการใช้

คู่มือการใช้แสดงอยู่ในภาคผนวก ค

บทที่ 5

สรุป

การวิจัยครั้งนี้สำเร็จลุล่วงตามวัตถุประสงค์ ได้โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ตามที่ต้องการ จากการทดสอบโปรแกรมดังกล่าว โดยใช้ข้อมูลเข้าที่เป็นไวยากรณ์ของภาษา Standard Pascal ซึ่งมี โพรดักชัน 193 โพรดักชัน สัญลักษณ์สิ้นสุด 66 ตัว สัญลักษณ์ไม่สิ้นสุด 102 ตัว และอักขระที่ประกอบกันเป็นสัญลักษณ์ไม่สิ้นสุด สัญลักษณ์สิ้นสุด และชื่อที่ตั้งให้สัญลักษณ์ จำนวน 1,796 อักขระ ใช้เวลาวิ่งบนเครื่อง IBM PC/XT Compatible ประมาณ 50 วินาที และบนเครื่อง AT&T 386/SX วิ่งโดยมีผู้ใช้คนเดียว ใช้เวลาประมาณ 9 วินาที

ตัววิเคราะห์วากยสัมพันธ์ที่ได้จากโปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ เป็นชุดคำสั่งย่อยภาษา C สามารถวิเคราะห์วากยสัมพันธ์ของภาษาที่ผลิตโดยไวยากรณ์ LL(1) ชุดคำสั่งย่อยดังกล่าวเรียกใช้ชุดคำสั่งย่อยอื่นอีก 3 ชุด คือ ตัววิเคราะห์ศัพท์ ตัววิเคราะห์ความหมาย และตัวจัดกระทำความผิดพลาด ซึ่งผู้ใช้ต้องเขียนเพิ่มเติมเอง

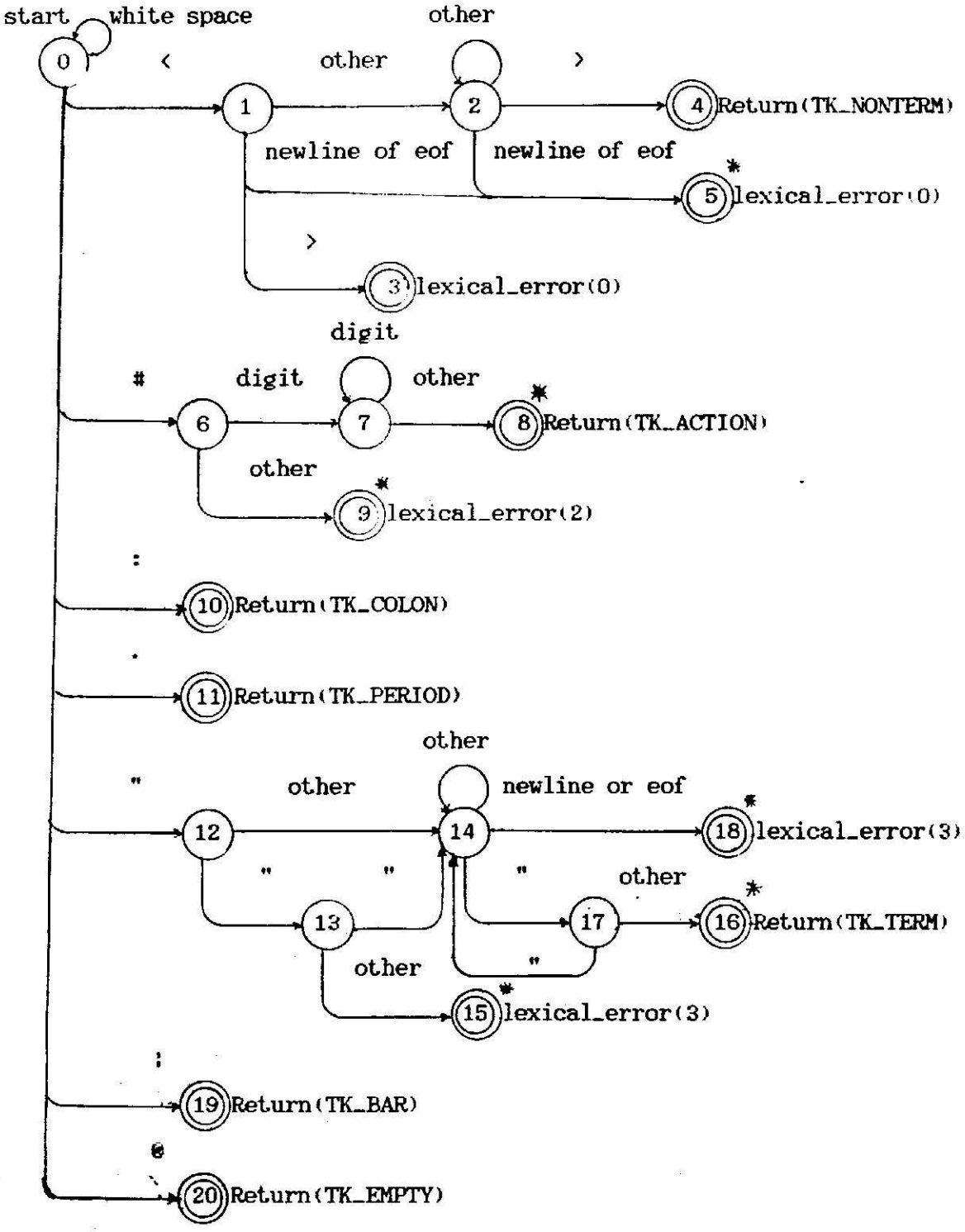
บทที่ 6

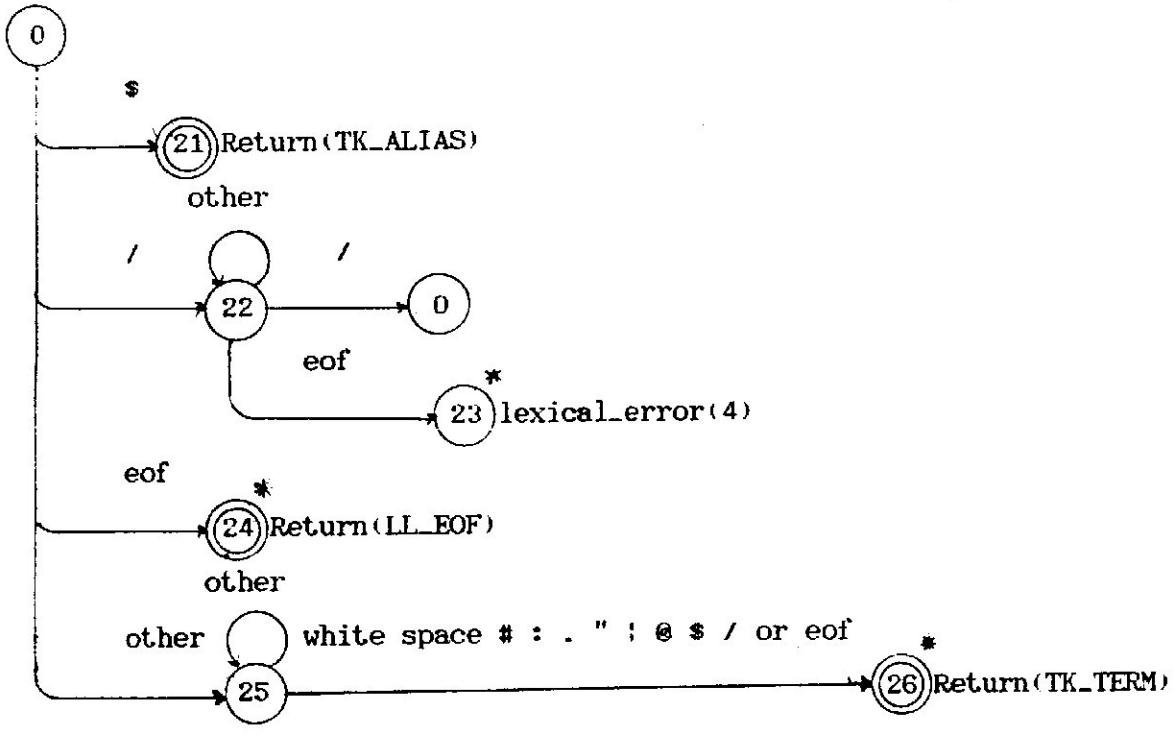
เอกสารอ้างอิง

1. อิว ไอยรากาญจนกุล, 2532, "ไมโครวันคอมไพเลอร์", ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ หาดใหญ่
2. Aho, A.V. and Ullman, J.D., 1977, "Principles of Compiler Design", Reading Mass: Addison-Wesley
3. Fischer, C.N. and LeBlance, R.J., 1988, "Crafting A Compiler", The Benjamin/Cummings Publishing Company, Inc.
4. Johnson, S.C., 1975, "Yacc - Yet Another Compiler-Compiler", Bell Laboratories, Murray Hill, New Jersey 07974
5. Kastens U., Hutt B., Zimmermann E., 1982, "GAG: A Practical Compiler-Generator", Lecture Notes in Computer Science 141, Springer-Verlag
6. Koskimies K., 1984, "A specification language for one-pass semantic analysis", SIGPLAN Notices 19,6,179-189
7. Lesk, M.E. and Schmidt E., 1975, "Lex - A Lexical Analyzer Generator", Bell Laboratories, Murray Hill, New Jersey 07974
8. Rechenberg, P. and Mössenböck, H, 1989, "A Compiler Generator for Microcomputers", Carl Hanser Verlag and Prentice Hall International (UK) Ltd.
9. Tremblay, J.P. and Sorenson, P.G., 1985, "The Theory and Practice of Compiler Writing", McGraw-Hill Inc.
10. Turbo C Reference Guide Version 2.0, Borland International, Inc.
11. UNIX System V Programmer's Guide, 1987, AT&T, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632

ภาคผนวก ก

แผนภาพทรานซิชัน




0
$
21
Return(TK_ALIAS)
other
/
/
22
0
eof
*
23
lexical_error(4)
eof
*
24
Return(LL_EOF)
other
other
white space # : . " ; @ $ / or eof
25
*
26
Return(TK_TERM)

ภาคผนวก ข

ไวยากรณ์ข้อมูลเข้าของโปรแกรม llgen

```
<grammar>    : <rules> <alias> .
<rules>      : <rule> <rules>
             | @ .
<rule>       : #1 nonterminal ":" <expression> "." #2 .
<expression> : <term> <ptail> .
<ptail>      : "|" #3 <term> <ptail>
             | @ .
<term>       : <factor> <ttail> .
<ttail>      : <term>
             | @ .
<factor>     : #4 nonterminal
             | #4 terminal
             | #4 action
             | #4 "@" .
<alias>      : #5 "$" <pairs>
             | @ .
<pairs>      : #6 terminal #7 terminal <pairs>
             | #8 nonterminal #9 terminal <pairs>
             | @ .
$
"$" TK_ALIAS
":" TK_COLON
"." TK_PERIOD
"|" TK_BAR
"@" TK_EMPTY
terminal TK_TERM
```

```
nonterminal TK_NONTERM

action TK_ACTION

<grammar> _GRAMMAR

<rules> _RULES

<alias> _ALIAS

<rule> _RULE

<expression> _EXPR

<term> _TERM

<ptail> _PTAIL

<factor> _FACTOR

<ttail> _TTAIL

<pairs> _PAIRS
```

ภาคผนวก ค
คู่มือการใช้
โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์
(A Syntax Analyzer Generator)

อิว ไอยรากาญจนกุล
ภ.คณิตศาสตร์ คณะวิทยาศาสตร์ ม.สงขลานครินทร์

บทนำ

วากยสัมพันธ์ของภาษาคอมพิวเตอร์ส่วนใหญ่ สามารถอธิบายได้ด้วย ไวยากรณ์ไม่พึ่งบริบท (context-free grammar) เช่น ภาษาคอมพิวเตอร์ภาษาหนึ่ง มีวากยสัมพันธ์ ดังนี้ เริ่มต้นด้วยคำว่า "PROGRAM" ตามด้วยประโยคคำสั่งอย่างน้อยหนึ่งประโยคคำสั่ง สิ้นสุดด้วยคำว่า "END" กรณีที่มีมากกว่าหนึ่งประโยคคำสั่ง ให้คั่นแต่ละประโยคคำสั่งด้วยเครื่องหมาย ";" สมมติว่ามีประโยคคำสั่งเพียงประโยคคำสั่งเดียว คือ "STOP" ภาษาดังกล่าวสามารถอธิบายด้วยไวยากรณ์ไม่พึ่งบริบท ซึ่งเขียนเป็นโพรดักชัน (production) หรือ กฎ (rule) ได้ ดังนี้

```
<program> : PROGRAM <statements> END .          / กฎที่ (1) /
<statements> : <statement> <statement_list> .   / กฎที่ (2) /
<statement_list> : ; <statements> .             / กฎที่ (3) /
<statement_list> : @ .                          / กฎที่ (4) /
<statement> : STOP .                            / กฎที่ (5) /
```

การวิเคราะห์ว่า โปรแกรม ต่อไปนี้

```
PROGRAM
 STOP
END
```

เขียนถูกวากยสัมพันธ์หรือไม่ ทำได้โดยเริ่มที่สัญลักษณ์เริ่มต้น (starting symbol) คือ <program> ใช้กฎที่ (1) แทนสัญลักษณ์ <program> ด้วย PROGRAM <statements> END ใช้กฎที่ (2) แทน <statements> ด้วย <statement> <statement_list> ใช้กฎที่ (5) แทน <statement> ด้วย STOP และใช้กฎที่ (4) แทน <statement_list> ด้วยสายอักขระว่าง แสดงด้วยขั้นตอน ดังนี้

```
<program> => PROGRAM <statements> END                          ใช้กฎที่ (1)
          => PROGRAM <statement> <statement_list> END          ใช้กฎที่ (2)
          => PROGRAM STOP <statement_list> END                 ใช้กฎที่ (5)
          => PROGRAM STOP END                                  ใช้กฎที่ (4)
```

ปรากฏว่าสายอักขระที่ได้ ตรงกับโปรแกรมที่วิเคราะห์ แสดงว่าเขียนถูกวากยสัมพันธ์

สัญลักษณ์ที่เขียนอยู่ภายในเครื่องหมาย "<" กับ ">" เป็นสัญลักษณ์ที่ถูกแทนที่ได้ เรียกว่า สัญลักษณ์ไม่สิ้นสุด (nonterminal symbol) สัญลักษณ์เริ่มต้นก็เป็นสัญลักษณ์ไม่สิ้นสุดตัวหนึ่ง ส่วนสัญลักษณ์ที่ถูกแทนที่ไม่ได้เรียกว่า สัญลักษณ์สิ้นสุด (terminal symbol) กฎที่ใช้ในการแทนที่สัญลักษณ์ไม่สิ้นสุด เขียนอยู่ในรูป

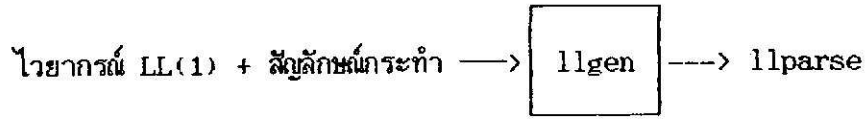
```
สัญลักษณ์ไม่สิ้นสุด : สายอักขระขวามือ .
```

สายอักขระขวามือ อาจจะเป็นสายอักขระว่าง (แทนด้วยเครื่องหมาย "@") หรือ เป็นสัญลักษณ์ไม่สิ้นสุดผสมกับสัญลักษณ์สิ้นสุด เครื่องหมาย ":" ใช้คั่นระหว่าง สัญลักษณ์ไม่สิ้นสุดซ้ายมือ กับสายอักขระขวามือ และเครื่องหมาย "." ใช้บอกการสิ้นสุดของแต่ละกฎ การเลือกกฎมาแทนที่สัญลักษณ์ไม่สิ้นสุด จะต้องเลือกกฎที่มีสัญลักษณ์ไม่สิ้นสุดซ้ายมือ ตรงกับสัญลักษณ์ไม่สิ้นสุดที่ต้องการแทนที่ และสายอักขระที่นำไปแทนที่ก็คือสายอักขระขวามือของกฎดังกล่าว ข้อมูลที่นำมาวิเคราะห์เขียนถูกวากยสัมพันธ์ ก็ต่อเมื่อจากสัญลักษณ์เริ่มต้น สามารถหากฎมาแทนที่สัญลักษณ์ไม่สิ้นสุด จนได้สายอักขระตรงกับข้อมูลดังกล่าว

นอกจากการวิเคราะห์วากยสัมพันธ์แล้ว ยังมีการวิเคราะห์ความหมาย ซึ่งมักจะทำไปพร้อม ๆ กับการวิเคราะห์วากยสัมพันธ์ วิธีการหนึ่งก็คือ เพิ่มสัญลักษณ์กระทำ (action symbol) เข้าไปในกฎแล้วเขียนชุดคำสั่งประจำ (routine) สำหรับสัญลักษณ์กระทำนั้น ๆ

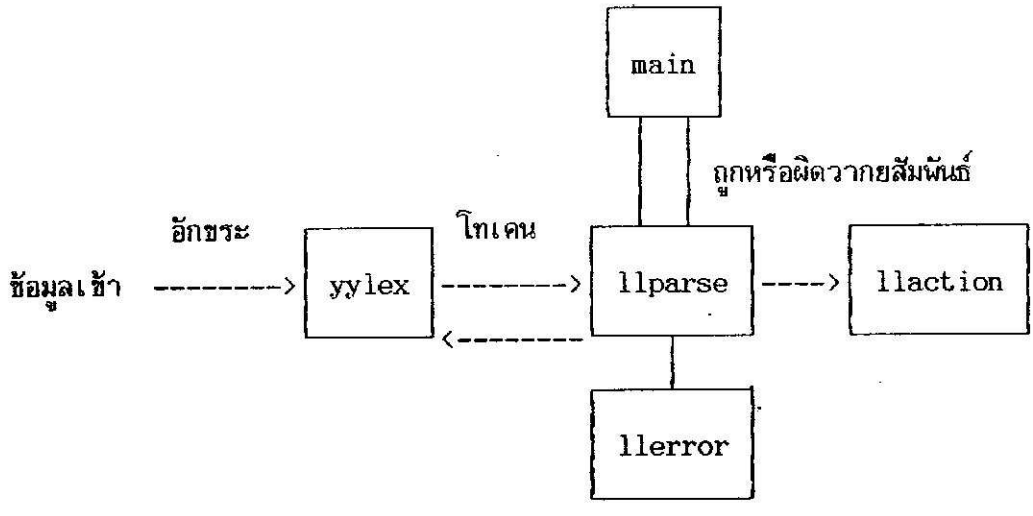
โปรแกรมผลิตตัววิเคราะห์วากยสัมพันธ์ ซึ่งต่อไปจะเรียกว่า **โปรแกรม llgen** เป็นโปรแกรมอรรถประโยชน์ (utility program) ช่วยสร้างตัววิเคราะห์วากยสัมพันธ์ ข้อมูลเข้าของโปรแกรม llgen ได้แก่ ไวยากรณ์ไม่พึ่งบริบทที่เรียกว่า ไวยากรณ์ LL(1) (ดูหัวข้อ 4) และมีสัญลักษณ์กระทำเพิ่มเข้าไปในกฎได้ ข้อมูลออกเป็นชุดคำสั่งย่อยภาษา C ชื่อ llparse ทำหน้าที่เป็นตัววิเคราะห์วากยสัมพันธ์ ข้อมูลเข้า/ออกของโปรแกรม llgen แสดงตามรูปที่ 1

ชุดคำสั่งย่อย llparse เรียกใช้ชุดคำสั่งย่อยอื่นอีก 3 ชุด ซึ่งผู้ใช้จะต้องเขียนเพิ่มเติมเอง ได้แก่ ชุดคำสั่งย่อย yylex, llaction และ llerror ชุดคำสั่งย่อย yylex เป็นตัววิเคราะห์ศัพท์-



รูปที่ 1 ข้อมูลเข้า/ออก ของโปรแกรม llgen

(lexical analyzer) ทำหน้าที่แยกอักขระจากข้อมูลเข้า ออกเป็นโทเคน (token) ซึ่งตรงกับสัญลักษณ์สิ้นสุดในไวยากรณ์ ชุดคำสั่งย่อย llaction เป็นตัววิเคราะห์ความหมาย (semantic analyzer) ชุดคำสั่งประจำสำหรับสัญลักษณ์กระทำจะต้องเขียนอยู่ในชุดคำสั่งย่อยนี้ ส่วนชุดคำสั่งย่อย llerror เป็นตัวจัดกระทำความผิดพลาด (error handler) ผู้ใช้สามารถใช้ชุดคำสั่งย่อยนี้ แสดงข้อความผิดพลาด และทำการกู้ (recovery) เพื่อให้สามารถวิเคราะห์ต่อไป หรือจะหยุดการทำงานก็ได้ นอกจากนี้ผู้ใช้จะต้องเขียนชุดคำสั่งย่อย main เอง เพื่อเรียกใช้ชุดคำสั่งย่อย llparse ชุดคำสั่งย่อยที่เกี่ยวข้องกับชุดคำสั่งย่อย llparse แสดงตามรูปที่ 2



รูปที่ 2 ชุดคำสั่งย่อยที่เกี่ยวข้องกับชุดคำสั่งย่อย llparse

1. <u>ตัววิเคราะห์วากยสัมพันธ์ทำงานอย่างไร</u>

ตัววิเคราะห์วากยสัมพันธ์ ใช้ตารางวิเคราะห์กระจาย (parsing table) และกองซ้อน (stack) ช่วยในการวิเคราะห์ ขั้นตอนวิธีวิเคราะห์เริ่มต้นด้วยกองซ้อนว่างเปล่า ใส่สัญลักษณ์เริ่มต้นลงบนกองซ้อน เรียกใช้ชุดคำสั่งย่อย yylex เพื่ออ่านโทเคนตัวแรก จากนั้นพิจารณาสัญลักษณ์บนยอด-

กองซ้อน (top of stack) ซึ่งจะทำให้เกิดเหตุการณ์ใดเหตุการณ์หนึ่ง ต่อไปนี้

ก. สัญลักษณ์บนยอดกองซ้อนเป็นสัญลักษณ์ไม่สิ้นสุด

ค้นหากฎในตารางวิเคราะห์กระจาย ถ้ามีกฎในการแทนที่ จะเอาสัญลักษณ์บนยอดกองซ้อนออก แล้วเอาสัญลักษณ์จากสายอักขระขวามือของกฎ เรียงจากขวามาซ้าย ใส่ลงบนกองซ้อนทีละตัว ตามลำดับ มิฉะนั้นจะเรียกใช้ชุดคำสั่งย่อย llerror

ข. สัญลักษณ์บนยอดกองซ้อนเป็นสัญลักษณ์สิ้นสุด

ถ้าสัญลักษณ์สิ้นสุดบนยอดกองซ้อนตรงกับโทกเคน จะเอาสัญลักษณ์ดังกล่าวออกจากกองซ้อน แล้วเรียกใช้ชุดคำสั่งย่อย yylex เพื่ออ่านโทกเคนตัวต่อไป มิฉะนั้นจะเรียกใช้ชุดคำสั่งย่อย llerror

ค. สัญลักษณ์บนยอดกองซ้อนเป็นสัญลักษณ์กระทำ

เรียกใช้ชุดคำสั่งย่อย llaction แล้วเอาสัญลักษณ์กระทำออกจากกองซ้อน

สัญลักษณ์บนยอดกองซ้อนจะถูกพิจารณาซ้ำ ๆ (และจะเกิดเหตุการณ์ดังที่กล่าวมาแล้วข้างต้น) จนกว่ากองซ้อนจะว่างเปล่า ถ้ากองซ้อนว่างเปล่าและโทเคนขณะนั้นเป็นโทเคนสิ้นสุดแฟ้ม (eof) แสดงว่าถูกวากยสัมพันธ์ มิฉะนั้นเรียกใช้ชุดคำสั่งย่อย llerror

2. ข้อมูลเข้า

ข้อมูลเข้าของโปรแกรม llgen มีรูปแบบดังนี้

```
กฎ
$
ส่วนตั้งชื่อให้สัญลักษณ์
```

ส่วนตั้งชื่อให้สัญลักษณ์จะมีหรือไม่ก็ได้ ถ้าไม่มี เครื่องหมาย "$" ไม่ต้องเขียนก็ได้ สำหรับกฎสามารถเขียนได้ในลักษณะรูปแบบอิสระ (free format) คือ ในหนึ่งบรรทัดอาจจะไม่เขียนอะไรเลยก็ได้ หรือเขียนกฎได้มากกว่าหนึ่งกฎ หรือหนึ่งกฎอาจจะเขียนอยู่ในหลาย ๆ บรรทัด แต่ละกฎจะต้องมีเครื่องหมาย "." บอกการสิ้นสุด สัญลักษณ์ไม่สิ้นสุดซ้ายมือและสายอักขระขวามือใช้เครื่องหมาย ":" เป็นตัวคั่น เครื่องหมาย "|" ใช้เมื่อต้องการเขียนกฎที่มีสัญลักษณ์ไม่สิ้นสุดซ้ายมือเหมือนกันย่อรวมกัน สัญลักษณ์ไม่สิ้นสุดได้ออกแบบให้คล้ายกับสัญลักษณ์อีบีเอ็นเอฟ (EBNF notation) คือ เริ่มต้นด้วยเครื่อง-

หมาย "<" ตามด้วยอักขระใด ๆ อย่างน้อยหนึ่งตัว ปิดท้ายด้วยเครื่องหมาย ">" สัญลักษณ์ไม่สิ้นสุดตัวแรกของกฎแรกจะถือว่าเป็นสัญลักษณ์เริ่มต้นเสมอ สัญลักษณ์สิ้นสุดเขียนได้สองแบบ แบบแรกเริ่มต้นด้วยเครื่องหมายฟันหนูตามด้วยอักขระใด ๆ อย่างน้อยหนึ่งตัว (ถ้าอักขระดังกล่าวเป็นเครื่องหมายฟันหนูต้องเขียนอักขระดังกล่าวสองตัว) แล้วปิดท้ายด้วยเครื่องหมายฟันหนู แบบที่สองเขียนด้วยอักขระใด ๆ อย่างน้อยหนึ่งตัวที่ไม่ใช่อักขระ ต่อไปนี้ อักขระว่าง (blank character) อักขระตั้งระยะ (tab character) หรือ เครื่องหมาย "<", ":", "¦", ".", "@", "/", "#", "$" และฟันหนู ส่วนสัญลักษณ์กระทำเริ่มต้นด้วยเครื่องหมาย "#" ตามด้วยตัวเลข อย่างน้อยหนึ่งตัว สัญลักษณ์ต่าง ๆ ต้องไม่เขียนข้ามบรรทัด สำหรับสายอักขระว่างใช้แทนด้วยเครื่องหมาย "@" นอกจากนี้ยังสามารถเขียนหมายเหตุ (comment) ได้ด้วย โดยเขียนอยู่ภายในเครื่องหมาย "/" กับ "/"

ส่วนตั้งชื่อให้สัญลักษณ์ มีไว้เพื่อให้ผู้ใช้นำชื่อที่ตั้งไปใช้ในชุดคำสั่งย่อยอื่น ๆ สัญลักษณ์ที่สามารถตั้งชื่อได้ ได้แก่ สัญลักษณ์สิ้นสุด และสัญลักษณ์ไม่สิ้นสุด ในการตั้งชื่อ ให้เขียนสัญลักษณ์ที่ต้องการตั้งชื่อ ตามด้วยอักขระว่าง หรืออักขระตั้งระยะ หรือขึ้นบรรทัดใหม่ (newline) อย่างน้อยหนึ่งตัว แล้วตามด้วยชื่อที่ต้องการตั้ง ชื่อที่ตั้งมีหลักเกณฑ์การเขียนเหมือนสัญลักษณ์สิ้นสุด ฉะนั้นผู้ใช้ต้องตั้งชื่อให้ตรงกับกฎเกณฑ์การตั้งชื่อตัวชี้เฉพาะ (identifier) ของภาษา C เอง

ข้อมูลเข้าของโปรแกรม llgen อธิบายด้วยสัญกรณ์อีบีเอ็นเอฟ ดังนี้

```
<ข้อมูลเข้า> ::= <กฎ> [ <ส่วนตั้งชื่อให้สัญลักษณ์> ]
<กฎ> ::= { <สัญลักษณ์ไม่สิ้นสุด> ':' <สายอักขระขวามือ> '.' }
<สายอักขระขวามือ> ::= <สัญลักษณ์> { <สัญลักษณ์> } | '@'
                     | <สายอักขระขวามือ> '¦' <สายอักขระขวามือ>
<สัญลักษณ์> ::= <สัญลักษณ์ไม่สิ้นสุด> | <สัญลักษณ์สิ้นสุด> | <สัญลักษณ์กระทำ>
<สัญลักษณ์ไม่สิ้นสุด> ::= '<' <อักขระชุดที่ 1> { <อักขระชุดที่ 1> } '>'
<สัญลักษณ์สิ้นสุด> ::= <อักขระชุดที่ 2> { <อักขระชุดที่ 2> }
                      | '"' <อักขระชุดที่ 3> { <อักขระชุดที่ 3> } '"'
<สัญลักษณ์กระทำ>   ::= '#' <ตัวเลข> { <ตัวเลข> }
<อักขระชุดที่ 1> ::= อักขระใด ๆ ยกเว้น เครื่องหมาย '>' หรือ ขึ้นบรรทัดใหม่
<อักขระชุดที่ 2> ::= อักขระใด ๆ ยกเว้น อักขระว่าง อักขระตั้งระยะ หรือ เครื่องหมาย '<',
                     ':', '¦', '.', '@', '/', '#', '$' และ '"' หรือ
                     ขึ้นบรรทัดใหม่
```

<อักขระชุดที่ 3> ::= อักขระใด ๆ หรือ '""' ยกเว้น เครื่องหมาย '"' หรือ ขึ้นบรรทัดใหม่

<ตัวเลข> ::= 0 | 1 | ... | 9

<ส่วนตั้งชื่อให้สัญลักษณ์> ::= '$' { (<สัญลักษณ์ไม่สิ้นสุด>) | (<สัญลักษณ์สิ้นสุด>)
<สัญลักษณ์สิ้นสุด> }

ตัวอย่าง 2.1 ไวยากรณ์ของภาษาที่มีสองประโยค คือ "แดงกินกล้วย" และ "ดำกินกล้วย" เขียนเป็นข้อมูลเข้าได้ดังนี้

```
<ประโยค> : <ประธาน> <กริยา> <กรรม> .   / กฎที่ 1 /
<ประธาน> : #1 "แดง" | #2 "ดำ" | @ .    / กฎที่ 2-4 /
<กริยา>  : #3 "กิน" .                   / กฎที่ 5 /
<กรรม>   : "กล้วย" .                    / กฎที่ 6 /
$
"กิน"      TK_EAT
<ประธาน>   _SUBJECT
```

ตัวอย่าง 2.1 ประกอบด้วยสัญลักษณ์ไม่สิ้นสุด 4 ตัว คือ <ประโยค> <ประธาน> <กริยา> และ <กรรม> สัญลักษณ์สิ้นสุด 4 ตัว คือ "แดง" "ดำ" "กิน" และ "กล้วย" สัญลักษณ์กระทำ 3 ตัว คือ #1 #2 และ #3 มีการตั้งชื่อ TK_EAT และ _SUBJECT ให้กับ "กิน" และ <ประธาน> ตามลำดับ

ตัวอย่าง 2.2 ตัววิเคราะห์วากยสัมพันธ์ของโปรแกรม llgen สร้างจากข้อมูลเข้าต่อไปนี้

```
<grammar>    : <rules> <alias> .
<rules>      : <production> <rules>
             | @ .
<production> : #1 nonterminal ":" <expression> "." #2 .
<expression> : <term> <ptail> .
<ptail>      : "|" #3 <term> <ptail>
             | @ .
```

```
<term>      : <factor> <ttail> .
<ttail>     : <term>
            ¦ @ .
<factor>    : #4 nonterminal
            ¦ #4 terminal
            ¦ #4 action
            ¦ #4 "@" .
<alias>     : #5 "$" <pairs>
            ¦ @ .
<pairs>     : #6 terminal #7 terminal <pairs>
            ¦ #8 nonterminal #9 terminal <pairs>
            ¦ @ .
```

```
$
"$"             TK_ALIAS
":"             TK_COLON
"."             TK_PERIOD
"¦"             TK_BAR
"@"             TK_EMPTY
terminal        TK_TERM
nonterminal     TK_NONTERM
action          TK_ACTION
<grammar>       _GRAMMAR
<rules>         _RULES
<alias>         _ALIAS
```

3. ข้อมูลออก

ข้อมูลออกของโปรแกรม llgen บรรจุอยู่ในแฟ้มข้อมูล 3 แฟ้ม คือ แฟ้มข้อมูลค่าคงที่ แฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ และแฟ้มข้อมูลรายงาน แต่ละแฟ้มบรรจุค่าต่าง ๆ ดังนี้

ก. แฟ้มข้อมูลค่าคงที่ บรรจุค่าคงที่ LLMAXDEPTH, LLTERMINAL_RANGE, LLNONTERMINAL_-RANGE, LL_EOF และ LLSTART_SYM ซึ่งแทน ขนาดของกองซ้อนที่ใช้วิเคราะห์วากยสัมพันธ์, รหัสสูงสุดของสัญลักษณ์สิ้นสุด, รหัสสูงสุดของสัญลักษณ์ไม่สิ้นสุด, รหัสโทเคนสิ้นสุดแฟ้ม และรหัสสัญลักษณ์เริ่มต้นตามลำดับ และถ้าผู้ใช้ตั้งชื่อให้กับสัญลักษณ์ ชื่อและรหัสของสัญลักษณ์เหล่านั้นจะบรรจุอยู่ในแฟ้มข้อมูลนี้ด้วย

ตัวอย่าง 3.1 จากแฟ้มข้อมูลเข้าในตัวอย่าง 2.2 แฟ้มข้อมูลค่าคงที่ จะบรรจุค่าต่อไปนี้

```
#define LLMAXDEPTH 8
#define LLTERMINAL_RANGE 8
#define LLNONTERMINAL_RANGE 11
#define LL_EOF 0
#define TK_NONTERM 1
#define TK_COLON 2
#define TK_PERIOD 3
#define TK_BAR 4
#define TK_TERM 5
#define TK_ACTION 6
#define TK_EMPTY 7
#define TK_ALIAS 8
#define LLSTARTING_SYM 9
#define _GRAMMAR 9
#define _RULES 10
#define _ALIAS 11
```

ข. แฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ บรรจุชุดคำสั่งย่อย llparse ชุดคำสั่งย่อยนี้จะใช้ค่าคงที่ในแฟ้มข้อมูลค่าคงที่เสมอ โดยการใช้คำสั่ง #include "ชื่อแฟ้มข้อมูลค่าคงที่"

ค. แฟ้มข้อมูลรายงาน ถ้าไม่มีความผิดพลาด จะให้รายงานต่างๆ ดังนี้

ค.1 กฎต่าง ๆ ของไวยากรณ์ที่จัดเรียงใหม่ โดยยึดหลักการเรียงตามสัญลักษณ์ไม่สิ้นสุด ซึ่งเขียนมาก่อนหลังในแฟ้มข้อมูลเข้าเป็นหลัก ส่วนสัญลักษณ์สิ้นสุด จะแสดงอยู่ภายในเครื่องหมายฟันหนู

ค.2 รหัสสูงสุดของสัญลักษณ์สิ้นสุด และสัญลักษณ์ไม่สิ้นสุด

ค.3 รหัสของสัญลักษณ์สิ้นสุดทั้งหมด

ค.4 รหัสของสัญลักษณ์ไม่สิ้นสุดทั้งหมด

ค.5 สัญลักษณ์ไม่สิ้นสุดทั้งหมด ที่สามารถถูกแทนที่โดยใช้กฎต่าง ๆ จนได้สายอักขระว่าง

ค.6 รหัสของสัญลักษณ์สิ้นสุด ที่เป็นสมาชิกของเซต FIRST(<X>) เมื่อ <X> เป็นสัญลักษณ์ไม่สิ้นสุดใด ๆ และ FIRST(<X>) คือ เซตของสัญลักษณ์สิ้นสุด ซึ่งปรากฏเป็นสัญลักษณ์ตัวแรกของสายอักขระใดๆ ที่เริ่มต้นจากสัญลักษณ์ไม่สิ้นสุด <X> แล้วเลือกกฎต่าง ๆ มาแทนที่จนได้สายอักขระดังกล่าว และถ้า <X> เป็นสัญลักษณ์ในข้อ ค.5 จะได้ว่าสายอักขระว่างเป็นสมาชิกของ FIRST(<X>) ด้วย

ค.7 รหัสของสัญลักษณ์สิ้นสุดหรือโทเคนสิ้นสุดแฟ้ม ที่เป็นสมาชิกของเซต FOLLOW(<X>) เมื่อ <X> เป็นสัญลักษณ์ไม่สิ้นสุดใด ๆ และ FOLLOW(<X>) คือ เซตของสัญลักษณ์สิ้นสุด ซึ่งปรากฏเป็นสัญลักษณ์ตัวแรกหลังสัญลักษณ์ <X> ในสายอักขระใด ๆ ที่เริ่มต้นจากสัญลักษณ์เริ่มต้น แล้วเลือกกฎต่าง ๆ มาแทนที่จนได้สายอักขระดังกล่าว และถ้า <X> เป็นสัญลักษณ์ตัวสุดท้ายในสายอักขระใด ๆ ดังกล่าวข้างต้น หรือ <X> เป็นสัญลักษณ์เริ่มต้น โทเคนสิ้นสุดแฟ้มจะเป็นสมาชิกตัวหนึ่งของ FOLLOW(<X>)

ตัวอย่าง 3.2 ถ้าแฟ้มข้อมูลเข้า คือ

```
<S>:b<B>.
<S>:a<A>.
<A>:a|@.
<B>:b.
```

ผลลัพธ์ที่เก็บอยู่ในแฟ้มข้อมูลรายงาน จะเป็นดังนี้

```
** GRAMMAR **   <----- ค.1
(1) <S> : "b" <B>
(2)     | "a" <A> "c" .
(3) <B> : "b" .
(4) <A> : "a"
(5)     | @ .
```

```
** TERMINAL_RANGE:3 NONTERMINAL_RANGE:6 **    <----- A.2
** CODE : TERMINAL **     <----- A.3
1 : "b"
2 : "a"
3 : "c"

** CODE : NONTERMINAL **     <----- A.4
4 : <S>
5 : <B>
6 : <A>

** EMPTY NONTERMINAL **     <----- A.5
<A>

** FIND FIRST **     <----- A.6
FIRST(<S>) ...
1 2
FIRST(<B>) ...
1
FIRST(<A>) ...
2

** FIND FOLLOW **     <----- A.7
FOLLOW(<S>) ...
0
FOLLOW(<B>) ...
0
FOLLOW(<A>) ...
3
```

4. การเกิดความขัดแย้ง

ไวยากรณ์ไม่พึงบริบทที่เรียกว่า **ไวยากรณ์ LL(1)** เท่านั้น ที่สามารถวิเคราะห์ได้ด้วยชุดคำสั่งย่อย llparse ไวยากรณ์ LL(1) เมื่อนำมาสร้างตารางวิเคราะห์กระจาย ในหนึ่งช่องของตารางจะมีกฎเพียงกฎเดียวเท่านั้น แต่บางไวยากรณ์ เมื่อนำมาสร้างตารางแล้ว ปรากฏว่าบางช่องของตารางมีกฎมากกว่าหนึ่งกฎ กรณีนี้เรียกว่า เกิดความขัดแย้ง (conflict) แสดงว่าไวยากรณ์ดังกล่าวไม่เป็นไวยากรณ์ LL(1) ดังเช่นตัวอย่าง 4.1

ตัวอย่าง 4.1 ไวยากรณ์ข้อมูลเข้าต่อไปนี้

```
<S>:c<A>|c<B>.
<A>:a.
<B>:b.
```

ไม่เป็นไวยากรณ์ LL(1) เพราะเกิดความขัดแย้ง แสดงด้วยตารางวิเคราะห์กระจาย ดังนี้

	c	a	b	eof
<S>	<S> : c <A> <S> : c <B>			
<A>		<A> : a		
<B>			<B> : b	

ความขัดแย้งในตัวอย่าง 4.1 ไม่สามารถตัดกฎใดกฎหนึ่งระหว่าง <S>:c<A> กับ <S>:c<B> ออกจากตาราง เพราะถ้าตัดกฎแรก จะไม่สามารถวิเคราะห์ความถูกต้องของ ca แต่ถ้าตัดกฎที่สอง จะไม่สามารถวิเคราะห์ความถูกต้องของ cb ในกรณีเช่นนี้จะต้องเขียนไวยากรณ์ใหม่ อย่างไรก็ตาม บางกรณีสามารถแก้ความขัดแย้งได้โดยการตัดกฎบางกฎออกจากตาราง ดังเช่นตัวอย่าง 4.2

ตัวอย่าง 4.2 ไวยากรณ์ข้อมูลเข้าต่อไปนี้

/ (1) / <stmts> : if <expr> then <stmts> <else clause>

/ (2) / ; b.

/ (3) / <expr> : a.

/ (4) / <else clause> : else <stmts>.

/ (5) / <else clause> : @.

เกิดความขัดแย้ง แสดงด้วยตารางวิเคราะห์กระจาย ดังนี้

	if	then	b	else	a	eof
<stmts>	(1)		(2)			
<expr>					(3)	
<else clause>				(4) (5)		(5)

ในตารางช่องที่กำกับด้วย <else clause> กับ else มีกฎสองกฎ ไวยากรณ์ในตัวอย่าง 4.2 เป็นตัวอย่างหนึ่งของการนิยามคำสั่ง if ในภาษาคอมพิวเตอร์ระดับสูงทั่ว ๆ ไป โดยที่คำสั่ง if จะมีส่วน else หรือไม่ก็ได้ และ ส่วน else จะคู่กับ if ที่อยู่ใกล้กัน เช่น if a then if a then b else b เป็นต้น ในกรณีนี้ถ้าตัดกฎที่ (5) คือ <else clause> : @ ออกจากตาราง การวิเคราะห์วากยสัมพันธ์ก็สามารถทำได้ถูกต้อง จึงไม่จำเป็นต้องเขียนไวยากรณ์ใหม่

สำหรับตารางวิเคราะห์กระจาย โปรแกรม llgen จะสร้างเก็บไว้ในชุดคำสั่งย่อย llparse ตารางดังกล่าวเก็บแบบย่อแถว เมื่อเกิดความขัดแย้ง โปรแกรม llgen จะแก้ไขความขัดแย้ง โดยการเลือกกฎที่เขียนมาก่อน เพียงกฎเดียวเท่านั้นใส่ในตาราง และแสดงข้อความเตือนบอกความขัดแย้งในแฟ้มข้อมูลรายงาน เพื่อเตือนให้ผู้ใช้มีความระมัดระวัง เพราะเมื่อมีการตัดกฎบางกฎออกจากตารางแล้ว โปรแกรมย่อย llparse อาจจะไม่สามารถนำไปวิเคราะห์วากยสัมพันธ์

5. ตัววิเคราะห์ศัพท์

ชุดคำสั่งย่อย llparse เรียกใช้ชุดคำสั่งย่อย yylex ซึ่งทำหน้าที่เป็นตัววิเคราะห์ศัพท์ ผู้ใช้จะต้องเขียนชุดคำสั่งย่อย yylex เอง โดยเขียนเป็นฟังก์ชันภาษา C ดังนี้

```
int yylex()
  {
    ... /* คำสั่งภาษา C */ ...
  }
```

หรืออาจจะใช้โปรแกรม Lex ช่วยสร้างให้ก็ได้

ฟังก์ชัน yylex ใช้เพื่ออ่านอักขระจากแฟ้มข้อมูลเข้า แล้วแยกอักขระเหล่านั้นออกเป็นโทเคน ผู้ใช้ต้องกำหนดแฟ้มข้อมูลเข้าเอง ค่าของโทเคนที่ yylex ส่งกลับให้ llparse จะต้องมีรหัสตรงกับรหัสของสัญลักษณ์สิ้นสุด ตามที่กำหนดไว้ในแฟ้มข้อมูลรายงาน (ดูหัวข้อ 3 ข้อ ค.3) ผู้ใช้อาจจะใช้ชื่อที่ตั้งให้กับสัญลักษณ์ แทนรหัสของสัญลักษณ์นั้น ๆ ก็ได้ สำหรับโทเคนสิ้นสุดแฟ้ม ให้ส่งค่าศูนย์หรือ LL_EOF กลับ

6. ตัววิเคราะห์ความหมาย

ผู้ใช้จะต้องเขียนตัววิเคราะห์ความหมายเอง ด้วยฟังก์ชันภาษา C ดังนี้

```
void llaction(X,a)
 int X,*a;
{
   ... /* คำสั่งภาษา C เช่น
      switch (X) {
        case 1: ...  ชุดคำสั่งประจำ สำหรับสัญลักษณ์กระทำ #1  ...
        case 2: ...  ชุดคำสั่งประจำ สำหรับสัญลักษณ์กระทำ #2  ...
        ...
      }
   */ ...
```

ฟังก์ชัน llaction มีอากิวเมนต์สองตัว คือ X เป็นตัวรับหมายเลขสัญลักษณ์กระทำ ซึ่งจะตรงกับหมายเลขที่เขียนในไวยากรณ์ ใช้เพื่อแยกเขียนชุดคำสั่งประจำสำหรับแต่ละการกระทำ ส่วนอากิวเมนต์ a เป็นตำแหน่งที่ llparse ใช้เก็บรหัสของโทเคนที่ได้จาก yylex มีไว้เพื่อให้ผู้ใช้สามารถทำการกู้เมื่อเกิดความผิดพลาดทางความหมาย โดยการเปลี่ยนรหัสโทเคนที่ตำแหน่ง a

7. ตัวจัดกระทำความผิดพลาด

เมื่อผิดวากยสัมพันธ์ ชุดคำสั่งย่อย llparse จะเรียกใช้ชุดคำสั่งย่อย llerror ชุดคำสั่งย่อยนี้ผู้ใช้จะต้องเขียนเพิ่มเติมเองด้วยฟังก์ชันภาษา C ดังนี้

```
void llerror(X,a)
 int X,*a;
{
  ... /* คำสั่งภาษา C */ ...
}
```

ฟังก์ชัน llerror มีอาถิวเมนต์สองตัว คือ X เป็นตัวรับรหัสของสัญลักษณ์บนยอดกองซ้อน และ a มีความหมายเหมือนกับในหัวข้อ 6 อาถิวเมนต์ทั้งสองมีไว้เพื่อให้ผู้ใช้นำไปแสดงข้อความผิดพลาด หรือทำการกู้ เช่น ถ้า X เป็นสัญลักษณ์สิ้นสุด (มีรหัสน้อยกว่าหรือเท่ากับค่า LLTERMINAL_RANGE) สามารถแสดงข้อความผิดพลาดได้ ดังนี้

```
printf("Expected %d but found %d\n",X,*a);
```
เป็นต้น

สำหรับการกู้ ถ้าต้องการเปลี่ยนรหัสของโทเคน ให้เปลี่ยนค่าที่ตำแหน่ง a แต่ถ้าต้องการ เพิ่ม ลบ หรือเปลี่ยนค่าบนยอดกองซ้อน จะต้องใช้ตัวแปร llstack ซึ่งเป็นแถวลำดับหนึ่งมิติ ใช้แทนกองซ้อน โดยมีตัวแปร lltop เป็นดรรชนีกำกับชี้อยู่ที่ยอดกองซ้อน กรณีกองซ้อนว่างเปล่าค่า lltop จะเท่ากับศูนย์ และจะล้น (overflow) เมื่อค่า lltop มากกว่าหรือเท่ากับค่า LLMAXDEPTH ตัวแปร llstack และ lltop ได้ประกาศไว้ในแฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ ดังนี้

```
int llstack[LLMAXDEPTH+1],lltop=0;
```

ฉะนั้นถ้าต้องการอ้างถึงตัวแปรทั้งสองในแฟ้มข้อมูลอื่น ๆ แฟ้มข้อมูลนั้นจะต้องประกาศ ดังนี้

```
extern int llstack[],lltop;
```

ตัวอย่างการกู้ด้วยวิธีแพนนิคโหมด (panic mode) โดยมีสัญลักษณ์สิ้นสุด ";" ซึ่งตั้งชื่อว่า TK_SEMI เป็นสัญลักษณ์สมวาร (synchronous symbol) วิธีการคือ อ่านโทเคนทิ้งจนกว่าจะพบโทเคน ";" หรือสิ้นสุดแฟ้ม และเอาสัญลักษณ์บนยอดกองซ้อนทิ้งจนกว่าสัญลักษณ์บนยอดกองซ้อนเป็น ";" หรือกองซ้อนว่างเปล่า เขียนเป็นภาษา C ได้ ดังนี้

```
/* panic mode recovery, synchronous symbol is ";" */
while ((*a!=TK_SEMI)&&(*a!=LL_EOF)) *a=yylex();
if (*a==LL_EOF) lltop=0;
while ((lltop>0)&&(llstack[lltop]!=TK_SEMI)) --lltop;
```

ในการกู้นั้น ผู้ใช้จะต้องระมัดระวังไม่ให้เกิดการวนไม่สิ้นสุด (infinite loop)

8. วิธีใช้โปรแกรม llgen

รูปแบบของคำสั่ง เมื่อต้องการใช้โปรแกรม llgen คือ

llgen [<options>] [<files>]

เมื่อ <options> และ <files> มีความหมายดังนี้

ก. <options> จะมีหรือไม่ก็ได้ ถ้ามีจะเป็นค่าใดค่าหนึ่ง หรือทั้งสองค่า ต่อไปนี้

-s ใช้เมื่อต้องการเปลี่ยนค่า LLMAXDEPTH ในแฟ้มข้อมูลค่าคงที่ ถ้าใส่ -s จะปรากฏข้อความ STACK SIZE: เพื่อบอกให้ผู้ใช้ใส่ค่าที่ต้องการ ถ้าไม่ใส่ LLMAXDEPTH จะเท่ากับ 128

-t ใช้เมื่อต้องการเปลี่ยนขนาดของตาราง ซึ่งใช้เก็บสัญลักษณ์สิ้นสุด สัญลักษณ์ไม่สิ้นสุด และชื่อที่ตั้งให้กับสัญลักษณ์ ถ้าใส่ -t จะปรากฏข้อความ TABLE SIZE: เพื่อบอกให้ผู้ใช้ใส่ขนาดที่ต้องการ ถ้าไม่ใส่ขนาดของตารางจะเท่ากับ 2048 ไบต์ ตารางดังกล่าวเป็นแถวลำดับหนึ่งมิติ

ข. <files> จะมีหรือไม่ก็ได้ ถ้ามีจะเป็นชื่อแฟ้มข้อมูล ซึ่งสามารถกำหนดได้ตั้งแต่หนึ่งถึงสี่ชื่อ โดยชื่อที่หนึ่งเป็นชื่อแฟ้มข้อมูลเข้า ถ้าไม่มีจะปรากฏข้อความ GRAMMAR FILE: เพื่อให้ผู้ใช้ใส่ชื่อ ชื่อที่สองเป็นชื่อแฟ้มข้อมูลค่าคงที่ ถ้าไม่มีจะถูกกำหนดให้เป็น llparse.h ชื่อที่สามเป็นชื่อแฟ้มข้อมูลตัววิเคราะห์วากยสัมพันธ์ ถ้าไม่มีจะถูกกำหนดให้เป็น llparse.c และชื่อที่สี่เป็นชื่อแฟ้มข้อมูลรายงาน ถ้าไม่มีจะถูกกำหนดให้เป็น llgen.lst

ชุดคำสั่งย่อย llparse ที่ได้จากโปรแกรม llgen เป็นภาษา C จะต้องแปลด้วยตัวแปลชุดคำสั่งภาษา C เช่น คำสั่ง cc บนระบบ UNIX หรือ tcc (TURBO C compiler) บนไมโครคอมพิวเตอร์ IBM/PC เป็นต้น และจะต้องเชื่อมโยงกับชุดคำสั่งย่อยอื่น ๆ ก่อนจึงจะนำไปใช้งานได้ ในช่วงการแปลชุดคำสั่งย่อย llparse มีค่าสองค่าที่สามารถนิยาม (define) ได้ คือ LLDEBUG และ LLRECOVER ค่าแรกเมื่อนิยาม แสดงว่าต้องการแกะรอย (trace) ดูการวิเคราะห์กระจาย ค่าที่สองเมื่อนิยามแสดงว่าต้องการกู้เมื่อผิดวากยสัมพันธ์ เพื่อให้สามารถวิเคราะห์ต่อไปได้ เพราะถ้าไม่นิยามค่านี้ หลังการเรียกใช้ llerror แล้ว llparse จะส่งค่า 1 กลับ แต่ถ้านิยาม หลังจากเรียกใช้ llerror แล้ว llparse จะวิเคราะห์ต่อไป และจะส่งค่า 0 กลับเมื่อกองซ้อนว่างเปล่า ค่าทั้งสองสามารถนิยามได้ในช่วงการแปลด้วยตัวประมวลก่อน (preprocessor)

9. ตัวอย่าง

ต้องการสร้างตัวแปลนิพจน์เลขคณิตให้เป็นภาษาแอสเซมบลี นิพจน์ที่จะแปลมีได้เฉพาะเครื่องหมาย +, -, *, / และใส่วงเล็บได้ ตัวถูกกระทำ (operand) อาจจะเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย หรือ ตัวแปรที่ชื่อประกอบด้วยตัวอักษรภาษาอังกฤษเท่านั้น กรณีที่ชื่อตัวแปรยาวเกิน 8 ตัวอักษร จะเอาเฉพาะ 8 ตัวอักษรแรกเป็นชื่อ ภาษาแอสเซมบลีที่ได้จากการแปล จะมีคำสั่ง 5 แบบ ซึ่งมีความหมายต่าง ๆ ดังนี้

PUSH op	เป็นคำสั่งนำค่าของตัวถูกกระทำไปเก็บในกองซ้อนที่ใช้คำนวณ โดยที่ op อาจจะเป็นเลขจำนวนเต็ม หรือ ตัวแปร
ADD	เป็นคำสั่งบวกเลขที่อยู่บนสุดกับเลขที่อยู่ถัดลงมาของกองซ้อนที่ใช้คำนวณ เอาตัวเลขทั้งสองที่นำมาคำนวณออกจากกองซ้อน แล้วใส่ผลลัพธ์ที่ได้จากการบวกลงไปในกองซ้อนแทน
SUB	เหมือนคำสั่ง ADD แต่เป็นการลบ
MUL	เหมือนคำสั่ง ADD แต่เป็นการคูณ
DIV	เหมือนคำสั่ง ADD แต่เป็นการหาร

ตัววิเคราะห์วากยสัมพันธ์สร้างได้จากแฟ้มข้อมูลเข้า ซึ่งบรรจุกฎและการตั้งชื่อให้กับสัญลักษณ์ต่าง ๆ ดังนี้

```
<expr>   : <term> <ptail> .
<ptail>  : + <term> #1 <ptail>
         ; - <term> #2 <ptail>
         ; @ .
<term>   : <factor> <ttail> .
<ttail>  : "*" <factor> #3 <ttail>
         ; "/" <factor> #4 <ttail>
         ; @ .
<factor> : ( <expr> )
         ; #5 id
         ; #6 number .
```

```
$
id          ID
number      NUMBER
"+"         ADD
"-"         SUB
"*"         MUL
"/"         DIV
"("         LPAREN
")"         RPAREN
```

ตัววิเคราะห์ศัพท์ที่ใช้เชื่อมกับชุดคำสั่งย่อย llparse เขียนได้ ดังนี้

```
#include <stdio.h>
#include <ctype.h>
#include "llparse.h"
#define NAME_LENGTH  8

int val,position=0;
char name[NAME_LENGTH+1];

int GetChar()
{
 ++position;
 return(getchar());
}

int yylex()
{ int i;
```

```
static int c,retract=0;

for (;;) {
while ((c=retract?c:GetChar())==' ');  /* skip blanks */
if (isalpha(c)) {
retract=0;
i=(-1);
do if (++i<NAME_LENGTH) name[i]=c;
while (isalpha(c=retract?c:GetChar()));
name[++i]='\0';
retract=1;
return(ID);
}
if (isdigit(c)) {
 retract=0;
 val=0;
 do {
  c=c-'0';
  val=10*val+c;
 }
 while (isdigit(c=retract?c:GetChar()));
 retract=1;
 return(NUMBER);
}
if (c=='\n') {
 retract=1;
 return(LL_EOF);
}
retract=0;
```

```
   switch (c) {
    case '+':return(ADD);
    case '-':return(SUB);
    case '*':return(MUL);
    case '/':return(DIV);
    case '(':return(LPAREN);
    case ')':return(RPAREN);
    default :printf("* error * unknown character %c : code %d position %d\n"
    ,c,c,position);  /* skip error character */
   }
  }
}
```

ตัวจัดกระทำความผิดพลาดที่ใช้เชื่อมกับชุดคำสั่งย่อย llparse เขียนได้ ดังนี้

```
extern int lltop;
void llerror(X,a)
int X,*a;
{
 printf("* error * before position %d top stack code %d token code %d\n"
 ,position,X,*a);
 if (*a==LL_EOF) lltop=0; /* set syntax stack empty */
 else *a=yylex(); /* skip error token */
}
```

ตัววิเคราะห์ความหมายที่ใช้เชื่อมกับชุดคำสั่งย่อย llparse เขียนได้ ดังนี้

```
void llaction(X,a)
int X,*a;
```

```
{
 switch(X) {
  case 1: printf("ADD\n"); break;
  case 2: printf("SUB\n"); break;
  case 3: printf("MUL\n"); break;
  case 4: printf("DIV\n"); break;
  case 5: printf("PUSH %s\n",name); break;
  case 6: printf("PUSH %d\n",val); break;
 }
 *a=*a; /* unused *a */
}
```

ชุดคำสั่งย่อยที่ใช้เรียก llparse เขียนได้ ดังนี้

```
main()
{
  llparse();
}
```

10. ข้อความเตือนและข้อความผิดพลาด

ก. ข้อความเตือน

หมายเลข	ข้อความ (สาเหตุ)
01	Alias name duplicate terminal name (ชื่อที่ตั้งให้กับสัญลักษณ์ไม่สิ้นสุดหรือสัญลักษณ์สิ้นสุด ตรงกับชื่อของสัญลักษณ์สิ้นสุด)
02	Table conflict :rule <r1> and rule <r2> :terminal <t> (ตารางวิเคราะห์กระจายเกิดความขัดแย้ง ตรงกฎที่ <r1> กับ <r2> ของไวยากรณ์ในแฟ้มข้อมูลรายงาน สัญลักษณ์สิ้นสุดรหัส <t> ทำให้เกิดความขัดแย้ง)

ข. ข้อความผิดพลาด

หมายเลข	ข้อความ (สาเหตุ)
03	Invalid nonterminal name (เขียนสัญลักษณ์ไม่สิ้นสุดผิดกฎเกณฑ์)
04	Action value overflow (หมายเลขกระทำโตเกินไป ต้องอยู่ในช่วง 0 ถึง 32766 - LLNONTERMINAL_RANGE)
05	Invalid action value (เขียนสัญลักษณ์กระทำผิดกฎเกณฑ์)
06	Invalid terminal name (ชื่อที่ตั้งให้สัญลักษณ์สิ้นสุดเขียนผิดกฎเกณฑ์)
07	Invalid comment (เขียนหมายเหตุผิด ต้องเขียนอยู่ภายในเครื่องหมาย "/" กับ "/")
08	Expected <m> But found <n> (เขียนสัญลักษณ์สิ้นสุด <n> ผิดที่ ที่ถูกควรเป็นสัญลักษณ์สิ้นสุด <m>)
09	Expected nonterminal $ or eof (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
10	Expected $ or eof (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
11	Expected nonterminal terminal action or @ (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
12	Expected ! or . (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
13	Expected nonterminal terminal action @ ! or . (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
14	Expected nonterminal terminal or eof (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)

หมายเลข	ข้อความ (สาเหตุ)
15	Syntax error (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
16	Syntax error can not continue (ข้อมูลเข้าเขียนผิดวากยสัมพันธ์)
17	Before . invalid RHS string (สายอักขระขวามือ มีสัญลักษณ์ @ ผสมกับสัญลักษณ์อื่น หรือมีมากกว่าหนึ่งตัว)
18	Before ! invalid RHS string (สายอักขระขวามือ มีสัญลักษณ์ @ ผสมกับสัญลักษณ์อื่น หรือมีมากกว่าหนึ่งตัว)
19	terminal not found (สัญลักษณ์สิ้นสุดที่ต้องการตั้งชื่อ ไม่มีอยู่ในกฎ)
20	nonterminal not found (สัญลักษณ์ไม่สิ้นสุดที่ต้องการตั้งชื่อ ไม่มีอยู่ในกฎ)
21	no active <n> (สัญลักษณ์ไม่สิ้นสุด <n> ไม่มีประโยชน์ที่จะเขียนในกฎ)
22	no reach <r> (สัญลักษณ์ <r> ไม่มีประโยชน์ที่จะเขียนในกฎ)
23	no lhs <n> (สัญลักษณ์ไม่สิ้นสุด รหัส <n> ปรากฏในสายอักขระขวามือ แต่ไม่เคยปรากฏเป็นสัญลักษณ์ไม่สิ้นสุดซ้ายมือ)
24	Name table overflow (see -t option) (ตารางที่ใช้เก็บสัญลักษณ์ หรือชื่อที่ตั้งให้สัญลักษณ์เต็ม สามารถเพิ่มขนาดของตารางนี้ได้ ด้วยการใส่ -t ดูหัวข้อ 8)
25	Stack underflow (มีความผิดพลาดในโปรแกรม llgen กรุณาติดต่อผู้เขียน)
26	Stack overflow (มีความผิดพลาดในโปรแกรม llgen กรุณาติดต่อผู้เขียน)

หมายเลข	ข้อความ (สาเหตุ)
27	Integer overflow (เลขจำนวนเต็มมีค่าเกิน 32767)
28	Memory overflow (เนื้อที่ในหน่วยความจำมีไม่พอที่จะทำงานต่อไป)
29	unknown (มีความผิดพลาดในโปรแกรม llgen กรุณาติดต่อผู้เขียน)

11. เอกสารอ้างอิง

[1] อิว ไอยรากาญจนกุล, 2532, "ไมโครวันคอมไพเลอร์", ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ หาดใหญ่

[2] Aho, A.V. and Ullman, J.D., 1977, "Principles of Compiler Design", Reading Mass: Addison-Wesley

[3] Fischer, C.N. and LeBlanc, R.J., 1988, "Crafting A Compiler", The Benjamin/Cummings Publishing Company, Inc.

[4] Tremblay, J.P. and Sorenson, P.G., 1985, "The Theory and Practice of Compiler Writing", McGraw-Hill Inc.

[5] Turbo C Reference Guide Version 2.0, Borland International, Inc.

[6] UNIX System V Programmer's Guide, 1987, AT&T, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632