สภาพแวดล้อมในการเขียนโปรแกรมบนฐานข้อมูลแบบอนุมานกฎได้

DEDUCTIVE DATABASE PROGRAMMING ENVIRONMENT

โดย

นายจำเริญ ภู่สว่าง

นายเฉลิมเอก อินทนากรวิวัฒน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2534

ปริญญานิพนธ์ปีการศึกษา 2534

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

ผู้จัดทำ

1. นาย จำเริญ ภู่สว่าง 31.1040

2. นาย เฉลิมเอก อินทนากรวิวัฒน์ 31.1051

..... อาจารย์ที่ปรึกษา

(ดร. คุภมิตร จิตตะยโศธร)

..... อาจารย์ที่ปรึกษา

(อาจารย์ ธนา หงษ์สุวรรณ)

สภาพแวดล้อมในการเขียนโปรแกรมบนฐานข้อมูลแบบอนุมานกฎได้

(Deductive Database Programming Environment)

จำเริญ ภู่สว่าง
เฉลิมเอก อินทนากรวิวัฒน์

ดร.คุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
อ. ชนา ทงษ์สุวรรณ อาจารย์ที่ปรึกษา

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้กล่าวถึง สภาพแวดล้อมของการเขียนโปรแกรมบนฐานข้อมูลแบบอนุมานกฎได้ โดยมีวัตถุประสงค์ เพื่อสร้างเครื่องมือ ในการพัฒนาโปรแกรมประยุกต์บนฐานข้อมูลแบบอนุมานกฎได้อย่างกว้างขวาง โดยเฉพาะอย่างยิ่ง ระบบผู้เชี่ยวชาญ และโปรแกรมประยุกต์ทางด้านปัญญาประดิษฐ์ทั้งหลาย เครื่องมือดังกล่าวข้างต้นคือ ภาษาโปรลอกไลค์ ซึ่งเป็น ภาษาเชิงตรรก โดยอาศัยรูปแบบทางภาษาของภาษาโปรลอกเป็นหลัก แต่ เพิ่มเติมความสามารถพิเศษบางอย่าง อันได้แก่ ความสามารถในการติดต่อและเรียกใช้กฎจากฐานข้อมูลได้ โครงงานนี้ได้สร้างตัวแปลภาษาโปรลอกไลค์ไว้ เพื่ออำนวยความสะดวก นักพัฒนาระบบในการพัฒนาโปรแกรมประยุกต์ที่สามารถดึงเอาความสามารถ และ ประสิทธิภาพของฐานข้อมูลแบบอนุมานกฎที่มีเหนือกว่าฐานข้อมูลแบบปกติออกมาได้อย่างเต็มประสิทธิภาพ ข้อดีนั้นคือ หากฐานข้อมูลสามารถอนุมานกฎเองได้ จะทำให้ลดจำนวนครั้งในการติดต่อกับฐานข้อมูลและเพิ่มความรวดเร็วให้แก่โปรแกรมประยุกต์ทางด้านปัญญาประดิษฐ์ที่ใช้ฐานข้อมูลเป็นฐานความรู้ได้อย่างมาก

นอกจากการสร้างตัวแปลภาษาโปรลอกไลค์แล้ว โครงงานนี้ยังครอบคลุมไปถึงการสร้างฐานข้อมูลที่อนุมานกฎได้ โดยเฉพาะอย่างยิ่งส่วนที่ใช้เก็บกฎ และ ส่วนที่ใช้ในการอนุมานกฎแบบเรียกตนเอง สำหรับรายละเอียดของโครงงานได้แสดงไว้ในปริญญานิพนธ์ฉบับนี้แล้ว

Deductive Database Programming Environment

JUMROEN BHOOSAWANG

CHALERMEK INTANAGONWIWAT

DR.SUPHAMIT CHITTAYASOTHORN ADVISOR

THANA HONGSUWAN ADVISOR

ABSTRACT

This project is Deductive database programming environment whose objective is to make a tool for developement of application programs especially in artificial intelligence approach such as the expert system.The tool,Prolog-like language, is a logic programming language with the same syntax with Prolog but it has the special ability to interface and to use the rules from the deductive database. We make the Prolog-like compiler to support the developer in developement of application programs that can take all advantage and efficiency of the deductive database to the normal database.The advantage is reducing the number of interfacing with the database so it make the artificial intelligence applications faster.

The scope of the project is not only to make the Prolog-like compiler but also to implement the deductive database especially in the part of storing and deducing the recursive rules.

สารบัญ

เรื่อง หน้าที่

สารบัญตาราง

สารบัญรูปภาพ

สภาพแวดล้อมในการเขียนโปรแกรมบนฐานข้อมูลแบบอนุมานกฎได้ เป็นตัวแปลภาษาโปรลอกไลค์ที่พัฒนามาให้มีความสามารถในการแปลภาษาจากภาษาโปรลอกไลค์เป็นภาษาซี โดยใช้หลักการของตัวแปลชุดคำสั่ง (Compiler) อันประกอบไปด้วยส่วนวิเคราะห์ศัพท์ (Lexical Analyzer) ,ส่วนวิภัช (Parser) , ส่วนก่อกำเนิดชุดคำสั่ง (Code Generator) ซึ่งจะมีส่วนตรวจสอบข้อผิดพลาดเพื่อช่วยนักพัฒนาโปรแกรมในการหาจุดบกพร่องในโปรแกรมได้รวดเร็วขึ้น นอกจากนี้ยังมีส่วนฟังก์ชันในตัว (Built-in Function) เพื่อช่วยในการติดต่อระหว่างโปรแกรมประยุกต์กับผู้ใช้ ทั้งในเรื่องการรับข้อมูลจากผู้ใช้,การแสดงผล

ภาษาโปรลอกไลค์เป็นภาษาเชิงตรรกที่สร้างขึ้นโดยขัดรูปแบบทางภาษาของภาษาโปรลอกเป็นหลักแล้วคงไว้เฉพาะความสามารถที่จำเป็นต้องใช้ในการพัฒนาโปรแกรมบนฐานข้อมูลแบบอนุมานกฎเท่านั้น แต่เพิ่มเติมความสามารถพิเศษในเรื่องของการติดต่อและเรียกใช้ข้อมูลหรือกฎจากฐานข้อมูลแบบอนุมานกฎได้ นอกจากการสร้างตัวแปลภาษาโปรลอกไลค์แล้ว โครงงานนี้ยังครอบคลุมไปถึงการสร้างฐานข้อมูลที่อนุมานกฎได้ โดยเฉพาะอย่างยิ่งส่วนที่ใช้เก็บกฎ และ ส่วนที่ใช้ในการอนุมานกฎแบบเรียกตนเอง

ในการพัฒนาระบบนี้ให้สามารถนำไปใช้งานได้จริงนั้นจำต้องอาศัยพื้นฐานความรู้ในหลายเรื่อง อาทิเช่น ความรู้ในเรื่องการเขียนโปรแกรมเชิงตรรก , ความสัมพันธ์ระหว่างภาษาเชิงตรรกและฐานข้อมูลแบบสัมพันธ์, ความรู้พื้นฐานของฐานข้อมูลแบบอนุมานกฎได้ ,ความรู้พื้นฐานในเรื่องคอมไพเลอร์ เป็นต้น ในปริญญานิพนธ์นี้จึงพยายามเรียบเรียงเนื้อหาอย่างเป็นลำดับขั้นตอนและ เต็มไปด้วยทฤษฎีต่างๆหลากหลายเรื่องราว เริ่มตั้งแต่

บทที่ 1 จะวางพื้นฐานทางทฤษฎีที่สำคัญในการพัฒนาโครงงานนี้ให้บรรลุผลสำเร็จไปได้ ประกอบไปด้วย ทฤษฎีทางด้านการเขียนโปรแกรมเชิงตรรกซึ่งจะกล่าวถึงเรื่อง พีชคณิตบลูลีน , เพิร์สออร์เดอร์เพรดิเคทโลจิก ,คลอสซัวส์ฟอร์ม ,อนุประโยคของฮอร์น ,ภาษาโปรลอก ,ข้อเท็จจริง ,กฎ ,คำถามทางตรรก แล้วอธิบายถึงความสัมพันธ์ของภาษาเชิงตรรกและฐานข้อมูลแบบสัมพันธ์เพื่อเป็นพื้นฐานสำหรับการสร้างฐานข้อมูลแบบอนุมานกฎได้ต่อไป นอกจากนี้ยังกล่าวถึงฐานข้อมูลแบบอนุมานกฎได้ไว้เล็กน้อยเป็นการปูทางสำหรับเนื้อหาในบทถัดไป โดยกล่าวถึงการอนุมานกฎในฐานข้อมูลทั้งแบบไม่เรียกตนเองซึ่งทำได้โดยง่ายด้วยการใช้วิว กับ กฎแบบเรียกตนเองที่มีความซับซ้อนกว่ามากอย่างคร่าวๆ

บทที่ 2 เป็นการกล่าวถึงฐานข้อมูลแบบอนุมานกฎโดยอย่างละเอียด โดยได้กล่าวถึงพื้นฐานคุณสมบัติของฐานข้อมูลระบบอนุมานกฎได้ที่ดีเหนือฐานข้อมูลแบบปกติ ทั้งยังกล่าวถึงทฤษฎีรองรับ

อย่างละเอียดโดยเฉพาะอย่างยิ่งการจัดการกฎแบบเรียกตนเองซึ่งในปัจจุบันก็ยังเป็นหัวข้อวิจัยที่ศึกษากันอยู่ และได้เปรียบเทียบวิธีการต่างๆที่มีผู้คิดค้นไว้ ซึ่งในโครงงานนี้ได้เลือกเอาวิธีของ Henshen และ Naqvi เป็นแบบอย่างในการพัฒนาการจัดการกฎแบบเรียกตนเองซึ่งนับว่าเป็นส่วนที่ซับซ้อนที่สุดในฐานข้อมูลแบบอนุมานกฎที่เดียว

บทที่ 3 จะกล่าวถึงภาษาโปรลอกไลค์ซึ่งเป็นภาษาที่เราสร้างขึ้นมาโดยอาศัยภาษาต้นแบบคือภาษาโปรลอก มาเพิ่มเติมความสามารถในการติดต่อกับฐานข้อมูลแบบอนุมานกฎได้ เนื่องด้วยหากมีฐานข้อมูลแบบอนุมานกฎได้แต่ไม่มีโปรแกรมประยุกต์ใด ที่เข้าถึงได้ฐานข้อมูลนั้นก็เป็นสิ่งที่ไร้ค่า ภาษาโปรลอกไลค์ถูกพัฒนามาด้วยสาเหตุนี้ โดยมุ่งหวังจะให้เป็นเครื่องมือสำคัญในการพัฒนาโปรแกรมประยุกต์บนฐานข้อมูลแบบอนุมาน อันจะช่วยดึงประสิทธิภาพของฐานข้อมูลแบบอนุมานที่มีเหนือฐานข้อมูลแบบปกติได้เด่นชัดยิ่งขึ้น ในบทนี้ประกอบไปด้วยรูปแบบทางภาษา ,การใช้งานและการพัฒนาตัวแปลภาษาโปรลอกไลค์ ที่จะต้องใช้ความรู้ทางด้านคอมไพเลอร์ นอกจากนี้ยังกล่าวถึงการนำโปรแกรม Lex และ Yacc มาช่วยในการสร้างส่วนวิเคราะห์ศัพท์ และ ส่วนวิภัชสุดท้ายของบทจะกล่าวถึง โทเคน, สัญกรณ์บีเอ็นเอฟ และ บิวท์อินฟังก์ชันของภาษาโปรลอกไลค์

บทที่ 4 กล่าวถึง การพัฒนาระบบควบคุมการทำงานของภาษาโปรลอกไลค์ อันประกอบไปด้วยโครงสร้างข้อมูลที่ใช้เก็บความสัมพันธ์ของเพรดิเคทต่างๆของโปรแกรมอันได้แก่ กฎและข้อเท็จจริงต่างๆที่เขียนขึ้นในโปรแกรมภาษาโปรลอกไลค์ และ ขั้นตอนวิธีในการนำโครงสร้างนั้นมาประมวลผลให้ได้ผลลัพธ์ตามที่โปรแกรมเขียนไว้

บทที่ 5 กล่าวถึง การเชื่อมประสานระหว่างโปรแกรมภาษาโปรลอกไลค์กับฐานความรู้ อันได้แก่ วิธีในการเขียนโปรแกรมเพื่อที่จะนำเอาข้อเท็จจริงที่เก็บอยู่ในฐานความรู้มาทำการประมวลผลในโปรแกรมภาษาโปรลอกไลค์

บทที่ 6 เป็น บทสรุป และวิจารณ์ โครงงานนี้เพื่อประเมินว่าบรรลุจุดประสงค์ของโครงงานที่ตั้งเอาไว้เพียงใด และ อุปสรรคต่างๆที่มีในการพัฒนาโครงงาน

บทที่ 7 เป็น แนวทางการพัฒนาต่อ อันจะช่วยให้ผู้สนใจสามารถมองเห็นแนวทางในการนำไปพัฒนาและประยุกต์ใช้งานได้ดี

ส่วนของฟแวร์ที่ต้องใช้ร่วมในการทำงานของโปรแกรมประกอบด้วย

1. ORACLE RDBMS
2. Microsoft C version 5.1 compiler
3. PRO*C Precompiler
4. Lex
5. Yacc

คุณสมบัติของฮาร์ดแวร์ที่จำเป็นในการทำงานของโปรแกรม

1. เครื่องไมโครคอมพิวเตอร์ไอบีเอ็มพีซี หรือ คอมแพทที่เบิ้ล ที่มีหน่วยความจำอย่างน้อย 2 เมกกะไบต์

2. ฮาร์ดดิสค์

วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาความสัมพันธ์ระหว่างภาษาเชิงตรรกและฐานข้อมูลแบบสัมพันธ์

2. เพื่อวิจัยและทดลอง การนำฐานข้อมูลแบบสัมพันธ์มาสร้างเป็นฐานความรู้ โดยเฉพาะการเก็บกฎและอนุมานกฎบนฐานข้อมูลแบบสัมพันธ์

3. เพื่อสร้างภาษาที่เหมาะสมสำหรับ การสร้างโปรแกรมประยุกต์บนฐานข้อมูลแบบอนุมานกฎได้

4. เพื่อสร้างตัวแปลภาษาดังวัตถุประสงค์ข้อที่ 3 เพื่อให้นักพัฒนาสามารถนำภาษาดังกล่าวไปใช้งานจริงได้

บทที่ 1

ทฤษฎีและหลักการที่เกี่ยวข้อง

1.1 การเขียนโปรแกรมเชิงตรรก (Logic Programming)

เนื่องจากในโครงงานนี้ ได้มีบางส่วนของโครงงานที่นำทฤษฎีทางด้านตรรกศาสตร์ (Symbolic Logic) ไปใช้อีกทั้งจากความจริงที่ว่าทั้งทฤษฎีทางด้านรีเลชันแนลและทฤษฎีทางด้านตรรกศาสตร์นั้นมีความใกล้ชิดกันอยู่มาก และสามารถที่จะพิสูจน์ได้ว่ามีความเหมือนกันภายใต้เงื่อนไขบางประการซึ่ง ดังนั้นในบทนี้จะเรากล่าวถึงทฤษฎีเบื้องต้นทางด้านตรรกศาสตร์

1.1.1 บทนำ

ภาษาคอมพิวเตอร์ที่พัฒนากันมาจากอดีตถึงปัจจุบันนี้ส่วนใหญ่จะเป็นภาษาที่มีการกำหนดลำดับขั้นตอนในการทำงานที่แน่นอน ซึ่งเราเรียกกันว่าเป็น ภาษาโปรแกรมเชิงกระบวนคำสั่ง (Procedural Programming) ตัวอย่างเช่น ภาษาฟอร์แทน ภาษาเบสิค ภาษาซี เป็นต้น ในการใช้งานภาษาเหล่านี้ ผู้ที่จะเขียนโปรแกรมจะต้องกำหนดลำดับขั้นตอน (Algorithm) สำหรับการแก้ปัญหาขึ้นเสียก่อน จากนั้นจึงเริ่มเขียนโปรแกรมให้สามารถทำงานตามลำดับขั้นตอนที่เรากำหนดไว้แล้วอย่างแน่นอน โดยไม่มีความยืดหยุ่นในการแปรเปลี่ยนลำดับขั้นตอนได้ ดังจะเห็นได้จากการเขียนผังงาน (Flow Chart) ของทุกโปรแกรม จะแสดงถึงลำดับการทำงานที่แน่นอน อย่างไรก็ตามยังมีกลุ่มของปัญหาอีกกลุ่มหนึ่ง ซึ่งไม่สามารถที่จะใช้ภาษาคอมพิวเตอร์เชิงกระบวนการในการช่วยแก้ปัญหาได้ หรือกระทำได้ยากมาก และไม่มีประสิทธิภาพ กลุ่มของปัญหานี้เรียกชื่อกันว่า ปัญญาประดิษฐ์ (Artificial Intelligence หรือ AI หรือ เอไอ) หรืออาจจะอธิบายเป็นภาษาธรรมดาได้ว่าเป็นสิ่งที่ชาญฉลาด ดังนั้นในช่วงปี 1970 จึงมีการพัฒนาภาษาเพื่อใช้กับปัญญาประดิษฐ์ขึ้น 2 ภาษาคือ ภาษาลิสป์ (LISP) และ ภาษาโปรลอก (PROLOG) ภาษาโปรลอกนั้นเป็นภาษาเชิงตรรก (Logic Programming Language) และย่อมาจากคำว่า PROgramming In LOGic หรือ การเขียนโปรแกรมเชิงตรรก

การเขียนโปรแกรมเชิงตรรก คือ การใช้ความรู้ทางตรรกเพื่อเขียนโปรแกรมคอมพิวเตอร์ ภาษาโปรลอกเริ่มต้นการพัฒนาในปี ค.ศ. 1972 โดยกลุ่มนักวิชาการของประเทศฝรั่งเศส ต่อมาในช่วงปี ค.ศ. 1973 - 1980 ภาษาโปรลอกก็เริ่มเป็นที่สนใจกันกลุ่มประเทศยุโรป เพื่อการ

ประยุกต์ใช้กับระบบปัญญาประดิษฐ์ ในช่วงเวลาเดียวกันนี้ในสหรัฐอเมริกา ยังไม่ค่อยให้ความสนใจภาษาโปรลอกเท่าใดนัก เพราะนักวิจัยในสหรัฐอเมริกาเองกำลังให้ความสนใจในการพัฒนาภาษาลิสป์ เพื่อการใช้งานในระบบปัญญาประดิษฐ์อยู่เช่นกัน

ภาษาโปรลอกมีคุณสมบัติตรงข้ามกับพวกภาษาเชิงกระบวนคำสั่งที่เด่นชัดคือ ความสามารถในการวินิจฉัย และ สรุปความเห็นชอบโดยการใช้เหตุผลจากข้อเท็จจริง โดยอาศัยฐานความรู้ (Knowledge Base) ที่มีอยู่ในโปรแกรมประกอบกับกลไกการวินิจฉัย (Inference Mechanism) ที่เราสร้างให้กับโปรแกรม โดยอาศัยการทำงานทางด้านตรรกวิทยา ดังนั้นภาษาโปรลอกจึงเหมาะสำหรับการนำไปแก้ปัญหาที่มีโครงสร้างไม่แน่นอน ซึ่งโปรแกรมในภาษาที่ใช้อยู่โดยทั่วไป ไม่สามารถกระทำได้ หรือ สามารถกระทำได้แต่ไม่สู้จะมีประสิทธิภาพนัก ในทางกลับกันภาษาโปรลอกก็ไม่เหมาะสมที่จะนำไปใช้กับปัญหาที่มีการคำนวณหรือทำงานตามลำดับขั้นตอนที่แน่ชัด อย่างไรก็ตามการพัฒนาภาษาโปรลอกจนถึงปัจจุบันนี้ก็ยังไม่บรรลุถึงวัตถุประสงค์สุดยอดของการนำไปใช้กับปัญญาประดิษฐ์ ซึ่งเราจะเห็นได้ว่าภาษายังมีลักษณะของภาษาเชิงกระบวนคำสั่งอยู่บ้าง

โปรแกรมภาษาโปรลอกนั้นนับว่าเป็นนิติใหม่ของการใช้คอมพิวเตอร์ในการช่วยแก้ปัญหาโดยเฉพาะในระยะเวลาตั้งแต่ปี ค.ศ. 1980 เป็นต้นมา การใช้ไมโครคอมพิวเตอร์ในหน่วยงานต่างๆได้ขยายออกไปอย่างกว้างขวาง ในขณะที่ความสามารถของเครื่องเพิ่มขึ้นเรื่อยๆ แต่ราคาของระบบกับลดลงอย่างรวดเร็ว ดังนั้นจึงมีผู้พยายามที่จะพัฒนาภาษาโปรลอกให้มีประสิทธิภาพสูงขึ้นและเหมาะสมที่จะใช้กับเครื่องไมโครคอมพิวเตอร์ จนกระทั่งได้เกิดโปรลอกในรูปแบบต่างๆกันหลายรูปแบบ อาทิเช่น เทอร์โบโปรลอก, โปรลอกพู, ซีโปรลอก, อาริดีโปรลอก และ ไมโครโปรลอก เป็นต้น แต่ละรูปแบบของโปรลอกเหล่านี้ต่างก็ได้รับการแต่งเติมให้มีความสามารถสูงขึ้นจนกระทั่งบางรูปแบบยังไม่มีมาตรฐานสากล ดังเช่น ภาษาฟอร์แทน หรือ เบสิค เป็นต้น สำหรับตำราที่อาจจะถือได้ว่ามีมาตรฐานของรูปแบบซึ่งเป็นที่ยอมรับกันอย่างกว้างขวางในการอ้างอิงก็คือตำราที่เขียนโดย คลอกซิน (Clocksin) และ เมลลิช (Mellish) แต่สิ่งที่เป็นตัวเร่งเร้าให้เกิดการตื่นตัวในการพัฒนาและการใช้ภาษาโปรลอกอย่างกว้างขวางคือ การที่ทางการประเทศญี่ปุ่นได้ประกาศที่จะเลือกภาษาโปรลอกให้เป็นภาษายุคที่ 5 สำหรับการพัฒนาเทคโนโลยีทางด้านปัญญาประดิษฐ์ ตัวอย่างความเหมาะสมของการใช้ภาษาโปรลอกในปัญญาประดิษฐ์คือ

- ระบบผู้เชี่ยวชาญ (Expert System)
- การโต้ตอบด้วยภาษาธรรมชาติ (Natural Language Processing)
- การสร้างเกมที่ต้องใช้การชักเหตุผล
- การประยุกต์ใช้กับหุ่นยนต์อุตสาหกรรม (Robotics)

ด้วยคุณสมบัติที่ดีเด่นของภาษาโปรลอก โครงงานนี้จึงเลือกเอาภาษาโปรลอกเป็นภาษาต้นแบบของภาษาโปรลอกไลค์สำหรับพัฒนาโปรแกรมประยุกต์บนฐานข้อมูลแบบอนุมานกฎได้

อย่างที่กล่าวไว้แล้วข้างต้นว่า การเขียนโปรแกรมเชิงตรรก นั้นตั้งอยู่บนพื้นฐานทางด้านตรรก ดังนั้น ผู้เริ่มศึกษาจึงควรมีพื้นฐานในเรื่อง ตรรกศาสตร์ หรือ พีชคณิตบลูลีน (Boolean Algebra) เป็นอย่างดี ตลอดจนมีความเข้าใจในตัวดำเนินการทางตรรกทั้งหลายอาทิเช่น หรือ (OR), และ(AND) โดยเฉพาะอย่างยิ่งตัวดำเนินการ ถ้าแล้ว (IMPLICATION) เพราะตัวดำเนินการ ถ้าแล้ว จะมีบทบาทสำคัญมากในการสร้างภาษาทางตรรก

อันที่จริงแล้วการโปรแกรมเชิงตรรก (Logic Programming) เป็นวิถีทางหนึ่งในสาขาวิทยาการคอมพิวเตอร์ ซึ่งตามความเป็นจริงแล้วก็เป็นทฤษฎีที่มีพัฒนาการมาเป็นเวลานาน เพราะการโปรแกรมเชิงตรรกนั้นก็คือรูปแบบหนึ่งของเฟิร์สออร์เดอร์เพรดดิเคทโลจิก (First Order Predicate Logic หรือ FOPL) ที่เรียกกันว่า อนุประโยคของฮอร์น (Horn Clause) นั่นเอง และ เฟิร์สออร์เดอร์เพรดดิเคทโลจิก ก็ไม่ใช่อะไรอื่นนอกจากผลจากวิวัฒนาการทางด้านซิมโบลิกโลจิก (Symbolic Logic) นอกจากนั้นหากเราต้องการจะย้อนกลับไปดูการเริ่มต้นของการศึกษาในเรื่องของซิมโบลิกโลจิก นั้นก็ต้องย้อนกลับไปในสมัยอริสโตเติลนั่นทีเดียวเพราะเฟิร์สออร์เดอร์เพรดดิเคทโลจิก เป็นสาขาหนึ่งของซิมโบลิกโลจิก ที่แยกตัวออกมาในช่วงศตวรรษที่ 20

ซิมโบลิกโลจิก คืออะไร จริงๆ แล้ว หากเราจะศึกษาซิมโบลิกโลจิกเราจะสามารถ ทำได้ในหลาย ๆ มุมมอง เช่นอาจจะมองในแง่ของปรัชญา ในแง่ของคณิตศาสตร์ แต่ในที่นี้เราจะมองในแง่ของการนำไปประยุกต์ใช้งานในการแทนปัญหา เราลองมาดูกันว่าซิมโบลิกโลจิกสามารถแทนปัญหาได้อย่างไร ลองมาดูตัวอย่างจากประโยคต่อไปนี้

F1 : If it is hot and humid, then it will rain.

F2 : If it is humid, then it is hot.

F3 : It is humid now.

และปัญหาก็คือ Will it rain?

และสมมติให้ประโยค F1,F2 และ F3 เป็นจริงและจากความจริงดังกล่าว เราสามารถใช้สัญลักษณ์ทางตรรกในการแทนความจริงเหล่านั้นได้ โดยกำหนดให้ P,Q,R แทนความจริงว่า "It is hot", "It is humid" และ "It will rain" ตามลำดับ และใช้เครื่อง

หมาย ^ ในการแทนความหมายของ and และ -> ในการแทนความหมายของ If...Then (Imply) ดังนั้นจากตัวอย่างข้างต้น เราสามารถนำประโยคปกติมาเขียนใหม่ในรูปของประโยคสัญญลักษณ์ได้ดังนี้

F1 : P ^ Q -> R

F2 : Q -> P

F3 : Q

และเมื่อเราทำการแปลงประโยคภาษาอังกฤษให้อยู่ในรูปแบบของประโยคทางตรรก (Logical Formulas) เราก็จะสามารถที่จะพิสูจน์ได้ว่าหาก F1, F2 และ F3 เป็นจริง แล้ว F4 : R จะเป็นจริงด้วย

และในลักษณะเช่นนี้เรากล่าวได้ว่า F4 logically follows from F1, F2 และ F3 นั่นคือจะได้คำตอบว่าฝนตก ทีนี้เราลองมาดูกันอีกตัวอย่างหนึ่ง สมมติว่าเรามีความจริงต่อไปนี้

F1 : Confucius is a man.

F2 : Every man is mortal.

จากรูปแบบของความจริงข้างต้น จะเห็นได้ว่าเราไม่สามารถแปลงเป็นรูปแบบทางตรรกในลักษณะเช่นเดิมได้ (ไม่มีรูปแบบของ every) ดังนั้นเราจะใช้วิธีการแทนความจริงเหล่านี้ในอีกรูปแบบหนึ่ง และจะเรียกว่าพรีดิเคต (Predicate) โดยกำหนดให้ P(x) และ Q(x) แทนความหมายว่า x is a man และ x is mortal และเราจะใช้เครื่องหมาย (x) หมายถึงสำหรับทุก x (For all x) ดังนั้นสำหรับความจริงข้างต้น เราจะเขียนในรูปแบบของพรีดิเคตได้เป็น

F1 : P(Confucius)

F2 : (x) (P(x) -> Q(x))

และจากความจริง F1 และ F2 นั้นเราสามารถที่จะอนุมานในเชิงตรรก (logically deduce) ได้ว่า F3 : Q(Confucius) หรือหมายความว่า Confucius is mortal.

จากตัวอย่างที่ได้กล่าวมาข้างต้น จะเห็นได้ว่า เราสามารถที่จะสรุปหรืออนุมานความจริงใดๆ ได้จากความจริงอื่นๆ และเราอาจจะมองได้อีกว่า ซิมโบลิกโลจิก คือภาษาหนึ่งที่มีลักษณะพิเศษคือสามารถใช้สัญญลักษณ์ในการแทนและสรุปเหตผลได้ ต่อไปเราจะศึกษา ซิมโบลิกโลจิก ที่มีรูปแบบที่ง่ายที่สุด คือ โพรโพสิชันนัลโลจิก (Propositional Logic)

1.1.2 พล็อบโพซิชันนัลลอจิก (Propositional Logic)

ในพล็อบโพซิชันนัลลอจิกนั้น ขั้นแรกเราจะต้องกำหนดประโยคขึ้นมา และจากนั้นเราจะสนใจว่าประโยคที่กำหนดนั้นมีค่าความจริงเป็นอะไร ซึ่งในพล็อบโพซิชันนัลลอจิก เรากำหนดว่าค่าความจริงจะต้องเป็น "จริง" หรือ "เท็จ" เท่านั้นจะเป็นอย่างอื่นไม่ได้ และจะเรียกแต่ละประโยคที่กำหนดนั้นว่า พล็อบโพซิชัน เช่น

P : Snow is white.

Q : Sugar is a hydrocarbon.

R : Smith has a Ph. D. Degree.

ซึ่งเราจะแทนค่าความจริง (Truth Value) ของแต่ละพล็อบโพซิชันด้วย T (True) และ F (False) และจากประโยคข้างต้น เราจะทำการแทนพล็อบโพซิชัน แต่ละพล็อบโพซิชัน ด้วยสัญญลักษณ์อักษรตัวใหญ่ หรือสตริงที่ขึ้นต้นด้วยอักษรตัวใหญ่ ซึ่งในที่นี้ก็คือ P, Q และ R และเราจะเรียกสัญญลักษณ์ดังกล่าวว่า อะตอม (atoms)

และจากพล็อบโพซิชันต่างๆ นั้นเราสามารถที่จะสร้างพล็อบโพซิชันขึ้นมาใหม่ โดยที่พล็อบโพซิชันที่สร้างขึ้นมาใหม่นั้นมีรูปแบบที่ซับซ้อนกว่าเก่า ซึ่งจะเรียกว่า คอมปาวพล็อบโพซิชัน (Compound Proposition) โดยใช้ ตัวเชื่อมทางตรรก (logical connective) เช่นจากประโยคที่ว่า

Snow is white and sky is clear. และ

If John is not at home then mary is at home.

ซึ่งตัวเชื่อมทางตรรกของประโยคข้างต้นก็คือ "and" และ "if ... then" ซึ่งตัวเชื่อมทางตรรกทั้งหมดที่ใช้งานนั้นก็มีอยู่ 5 ชนิดดังต่อไปนี้ ~(not) ^ (and) v (or) -> (if .. then) และ <-> (if and only if) และใน พล็อบโพซิชันนัลลอจิก เราจะเรียก

ทั้ง พล็อบโพซิชัน และ คอมปาวพล็อบโพซิชัน เรียกว่า (Well-Formed Formula) และในกรณีที่ใน WFF มีตัวเชื่อมมากกว่า 1 ตัวขึ้นไป จะต้องมีการจัดลำดับของตัวเชื่อม โดยกำหนดให้ลำดับความสำคัญของตัวเชื่อม (Connective) เรียงจากน้อยไปหามากดังต่อไปนี้

<-> -> ^ v ~

และนั่นก็หมายความว่า WFF ที่ว่า P -> Q ^ ~R v S จะมีความหมายในทางตรรกเทียบเท่ากับ (P -> (Q^((~R)vS)))

เมื่อกำหนดให้ G และ H เป็น WFF ค่าของ ~G, (G ^ H), (G v H), (G -> H), (H <-> G) จะมีได้ดังตารางที่ 1-1

G	H	~G	G ^ H	G v H	G -> H	G <-> H
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

ตารางที่ 1-1 แสดงค่าความจริงของตัวเชื่อมแบบต่างๆ
เมื่อกำหนดให้ G และ H เป็น WFF ที่มีค่าใดๆ

และจากตารางนี้เองทำให้เราสามารถที่จะหาค่าความจริงของ WFF ใดๆ ได้จากค่าความจริงของทุกๆ อะตอมที่อยู่ใน WFF ได้

1.1.2.1 การแทนค่าสูตร (Formula) ใน พล็อบโพซิชันนัลลอจิก

สมมติว่าเรามีอะตอมอยู่ 2 ตัวคือ P และ Q ซึ่งมีค่าความจริงเป็น T และ F ตามลำดับ และจากตารางที่ 2-1 หากเราแทน P ด้วย G และ Q ด้วย H เราจะได้ว่าค่าความจริงของ (~P), (P ^ Q), (P v Q), (P -> Q), (P <-> Q) คือ F,F,T,F และ F ตามลำดับ ดังนั้นเราสามารถสรุปได้ว่าค่าความจริงของสูตร (Formula) ใด ๆ นั้นสามารถหาได้จากการสรุปจากค่าความจริงของทุกๆ อะตอมที่อยู่ในสูตรนั้น

ตัวอย่าง

กำหนด สูตร

$$G = (P \wedge Q) \rightarrow (R \leftrightarrow (\sim S))$$

อะตอมของสูตรข้างต้นก็คือ P,Q,R และ S ซึ่งในที่นี้กำหนดให้มีค่าความจริงเป็น T,F,T และ T ตามลำดับ และจากค่าความจริงที่เราได้กำหนดให้อะตอม เราสามารถ สรุปได้ว่า (P ^ Q) มีค่าความจริงเป็น F และเนื่องจาก (~S) มีค่าเป็น F ดังนั้น (R <-> (~S) มีค่าเป็น F และจากที่กล่าวมาทั้งหมดจะสามารถสรุปได้ว่า สูตร ข้างต้นมีค่าความจริงเป็น T เมื่อกำหนดให้ P,Q,R และ S มีค่าเป็น T,F,T และ T ตามลำดับ

การกำหนดค่าความจริง {T,F,T,T} ให้กับ {P,Q,R,S} นี้เราเรียกว่าการแทนค่า (Interpretation) ของ สูตร G ดังนั้นสำหรับ P,Q,R และ S ก็จะสามารถมีการแทนค่าได้ทั้งหมด $2^4 = 16$ รูปแบบ

ต่อไปเราลองมาดู สูตร G = ((P -> Q) ^ P) -> Q จะเห็นว่าในสูตร นี้มีอะตอมเพียง 2 ตัวคือ P และ Q ดังนั้นการแทนค่าทั้งหมดจะแสดงได้ดังตารางที่ 1-2 และจากตารางจะเห็นได้ว่า สูตร G มีค่าเป็นจริงภายใต้การแทนค่าในทุกรูปแบบ ซึ่งเราจะเรียก สูตร ประเภทนี้ว่า ทอโทโลยี (Tautology)

P	Q	(P -> Q)	((P -> Q) ^ P)	((P -> Q) ^ P) -> Q
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

ตารางที่ 1-2 ตารางค่าความจริงของ G = ((P -> Q) ^ P) -> Q

ในลักษณะเช่นเดียวกันเราจะเรียก สูตร ที่มีค่าเป็นเท็จทั้งหมดภายใต้การแทนค่าในทุกรูปแบบว่า คอนทราดิคชั่น (Contradiction)

1.1.2.2 รูปแบบทั่วไป (Normal Form) ของพล็อบโพซิชันนัลลอจิก

ในการแทนปัญหาและการแก้ปัญหานั้น สิ่งหนึ่งที่จะพบอยู่เสมอคือการแปลงรูปของสูตร หนึ่งไปยังอีก สูตร หนึ่ง โดยเฉพาะการแปลงไปสู่รูปแบบที่เรียกว่า รูปแบบทั่วไป (Normal Form) ซึ่งการแปลงนี้สามารถทำได้โดยการแทน สูตร ด้วยอีก สูตร หนึ่งที่เท่ากัน (Equivalent) จนกระทั่งได้รูปแบบที่ต้องการ

แต่ปัญหาก็คือการเท่ากันของ สูตร มีความหมายว่าอย่างไร เพื่อให้มีความชัดเจนที่จุดนี้ เราจะนิยามว่า สูตร F และ G มีความเท่ากัน (Equivalent) ก็ต่อเมื่อค่าความจริงของ G เท่ากับค่าความจริงของ F ภายใต้การแทนค่าเดียวกันเช่น สมมติว่าเรามี สูตร P -> Q และ ~P v Q เราสามารถแสดงว่า สูตร ทั้งสองเท่ากันได้โดยการแทนค่าความจริงดังตารางที่ 1-3 ซึ่งจากตารางเราจะสามารถสรุปได้ว่าทั้ง สูตร P -> Q และ ~P v Q นั้นมีความเท่ากัน

P	Q	(P -> Q)	~P	(~P v Q)
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

ตารางที่ 1-3 แสดงตารางค่าความจริงของ (P -> Q) และ (~P v Q)

และเพื่อที่จะสามารถอธิบายถึง พล็อบโพซิชันนัลลอจิก ต่อไปได้ เราจะนิยามคำบางคำดังต่อไปนี้

นิยาม ลิทเทอร์รัล (Literal) เป็นอะตอมหรือค่าลบ (negation) ของอะตอม

นิยาม สูตร F จะสามารถเรียกว่าเป็น คอนจังค์ทีฟนอร์มัลฟอร์ม (Conjunctive Normal Form) ก็ต่อเมื่อ F อยู่ในรูปของ F = F1 ^ F2 ^ ... ^ Fn โดยที่ n >= 1 โดยที่ F1,F2....,Fn เป็น ดีสจังค์ทีฟ (Disjunctive) ของลิทเทอร์รัลเช่นหาก F = (P v ~Q v R) ^ (~P v Q) แล้ว F จะเป็น คอนจังค์ทีฟนอร์มัลฟอร์ม

นิยาม สูตร F จะสามารถเรียกว่าเป็น ดีสจังค์ทีฟนอร์มัลฟอร์ม (Disjunctive Normal Form) ก็ต่อเมื่อ F อยู่ในรูปของ F = F1 v F2 v ... v Fn โดยที่ n >= 1 โดยที่ F1,F2....,Fn เป็น คอนจังค์ทีฟ (Conjunctive) ของลิทเทอร์รัลเช่นหาก F = (P ^ ~Q ^ R) v (~P ^ Q) แล้ว F จะเป็น ดีสจังค์ทีฟนอร์มัลฟอร์ม

ถึงตรงนี้เราก็จะสามารถกล่าวได้ว่าสำหรับ สูตร ใดๆ นั้นเราสามารถที่จะแปลงรูปของ สูตร นั้นไปเป็นรูปแบบทั่วไป ได้ เช่น

ตัวอย่าง จงแปลง (P v ~Q) -> R ให้อยู่ในรูปของดีสจังค์ทีฟนอร์มัลฟอร์ม

$$
\begin{aligned}
(P \vee \sim Q) \rightarrow R &= \sim(P \vee \sim Q) \vee R \\
&= (\sim P \wedge \sim(\sim Q)) \vee R \\
&= (\sim P \wedge Q) \vee R
\end{aligned}
$$

1.1.2.2.1 คลอสซอลฟอร์ม (Clausal Form)

ในการใช้ภาษาบรรยายสิ่งต่างๆนั้น ไม่ว่าสิ่งใดก็ตามจะไม่สามารถถูกเขียนบรรยายเพียงแค่ประโยคเดียวได้ จำเป็นจะต้องใช้หลายประโยค และ ประโยคเหล่านั้นต้องเป็นจริงทั้งหมดด้วย เมื่อรวมประโยคทั้งหมดเข้าด้วยกัน จึงจะเป็นการบรรยายที่ถูกต้อง

รูปแบบในการบรรยายสิ่งต่างๆ ในทางตรรกได้แก่ รูปแบบคลอสชัว (Clausal Form) ,(Kowalski,1974) ซึ่งเป็นรูปแบบหนึ่งของ คอนจังค์ทีฟนอร์มัลฟอร์ม ที่มีการใช้งานในโปรแกรมเชิงตรรกมาก ซึ่งมีลักษณะดังนี้

A1,A2,....,Am <- B1,....,Bn

ซึ่งมีค่าเท่ากับ

A1 or ... or Am or ~B1 or ... or ~Bn

โดย A คือ ข้อสรุป (Conclusions)

B คือ เงื่อนไข (Conditions)

จำนวนของ A และ B (ค่าของ m และ n) ที่ต่างๆกันจะถูกแยกเป็นกรณี (CASE) ต่างๆได้ดังนี้

m>1 : ข้อสรุปนั้นถือว่า ไม่แน่นอน (Indefinite) เพราะมีข้อสรุปมากกว่า 1 ข้อสรุป

m<=1 : เรียกว่า ฮอร์นคลอส (Horn Clause) ซึ่งย่อมหมายความว่ามีข้อสรุปเดียว หรือ ไม่มีข้อสรุป

m=1,n>0 : เรียกว่า บทสรุปที่แน่นอน (Definite) เพราะมีข้อสรุปเดียว

m=1,n=0 : เรียกว่า อนุประโยคที่เป็นจริงโดยไม่มีเงื่อนไข (Unconditional definite clause) หรือ ข้อเท็จจริง (Fact) เพราะข้อเท็จจริงตามธรรมชาตินั้นจะเป็นจริงเสมอ โดยไม่มีเงื่อนไข

m=0,n>0 : เรียกว่า อนุประโยคเท็จ (Pure Negation) เพราะไม่มีข้อสรุป

m=0,n=0 : เรียกว่า อนุประโยคว่าง (Empty Clause)

ยกตัวอย่างเช่น

ฝนตก <-

ฉันเปียก <- ฝนตก,ฉันอยู่ข้างนอก

ฉันอยู่ข้างนอก <-

<- ฉันเปียก

พีชคณิตบลูลีนมีชื่อเรียกอีกชื่อหนึ่งว่า โพรโพสิชันนัลแคลคลูลัส (Propositonal Calculus) หรือ โพรโพสิชันนัลโลจิก (Propositional Logic) ซึ่งถือว่าเป็น ซับเซต ของระบบตรรก ที่ชื่อว่า เฟิร์สออร์เดอร์พรีดิเคทโลจิก (First Order Predicate Logic) หรือ เฟิร์สออร์เดอร์พรีดิเคทแคลคลูลัส (First order predicate calculus)

โปรลอกเป็นภาษาหนึ่งในการเขียนโปรแกรมเชิงตรรกที่ยาคัยรูปแบบคลอสช้วเป็นรูปแบบของภาษาและทุกพรีดิเคทของภาษาโปรลอกจะต้องเป็นกรณีฮอร์นคลอสทั้งหมด เช่น

ancestor(X,Z) :- parent(X,Y),ancestor(Y,Z).

โดย

ancestor(X,Z) คือ ข้อสรุป

parent(X,Y) คือ เงื่อนไขที่ 1

ancestor(Y,Z) คือ เงื่อนไขที่ 2

ส่วนเครื่องหมาย ':-' คือ เครื่องหมาย '<-' ในรูปแบบคลอสช้วนั่นเอง

โดย พรีดิเคท ancestor(X,Y) ในที่นี้จะเรียกว่า กฎ (Rules)

พิจารณาตัวอย่างต่อไปนี้

parent(dang,dum).

ตัวอย่างนี้เป็นรูปแบบคลอสช้วเช่นกัน โดยมีค่า m =1 และ n = 0 ซึ่งคือ ข้อเท็จจริง (Facts) นั่นเอง

1.1.2.3 <u>ข้อสรุปทางลอจิก</u> (Logical Consequence)

ในชีวิตประจำวัน เรามักจะพบว่าเหตุการณ์อย่างหนึ่งมักจะต้องเกิดหลังอีกเหตุการณ์หนึ่ง และจากความจริงที่ว่า พล็อบโพซิชันนัลลอจิก นั้นสามารถที่จะแทนปัญหาในชีวิตประจำวันได้ได้เช่นกัน จากความจริงที่ได้กล่าวมาทั้ง 2 ประการนี้เองที่จะนำเราไปสู่แนวความคิดของ ข้อสรุปทางลอจิก (Logical Consequence) แต่ก่อนที่เราจะกล่าวถึง ลำดับทางลอจิก เรามาดูตัวอย่างต่อไปนี้ก่อน

<u>ตัวอย่าง</u> กำหนดให้ราคาหุ้น (stock price) จะมีราคาลดลงเมื่อภาษี (Prime interest rate) เพิ่มขึ้น และประชาชน (People) จะไม่สบายใจ (Unhappy) เมื่อราคาหุ้นลดลง สมมติว่าภาษีเพิ่มขึ้น จงแสดงว่าประชาชนจะไม่สบายใจ

จากโจทย์ เราสามารถเขียนเป็นประโยคได้ดังต่อไปนี้

P = Prime interest rate goes up,

S = Stock prices go down,

U = Most people are unhappy.

และจากโจทย์สามารถเขียนได้เป็น

1) If the prime interest rate goes up, stock prices go down.

2) If stock prices go down, most people are unhappy.

3) The prime interest rate goes up.

4) Most People are unhappy.

ซึ่งสามารถเขียนให้อยู่ในรูปของสัญญลักษณ์ได้เป็น

1') P -> S

2') S -> U

3') P

4') U

ซึ่งต่อไปเราก็จะแสดงว่า 4') เป็นจริง เมื่อ 1') ^ 2') ^ 3') เป็นจริง โดยขั้นแรกนั้นเราจะต้องแปลง 1') ^ 2') ^ 3') ให้อยู่ในรูปของ รูปแบบทั่วไป เสียก่อน

$$((P\to S) \wedge (S\to U) \wedge P) = ((\neg P \vee S) \wedge (\neg S \vee U) \wedge P)$$

$$= (P \wedge (\neg P \vee S) \wedge (\neg S \vee U))$$

$$= (((P \wedge \neg P) \vee (P \wedge S)) \wedge (\neg S \vee U))$$

$$= ((f \vee (P \wedge S)) \wedge (\neg S \vee U)$$

$$= (P \wedge S) \wedge (\neg S \vee U)$$

$$= (P \wedge S \wedge \neg S) \vee (P \wedge S \wedge U)$$

$$= (P \wedge f) \vee (P \wedge S \wedge U)$$

$$= f \vee (P \wedge S \wedge U)$$

$$= P \vee S \vee U$$

และในกรณีเช่นนี้จะเรียก U ว่าเป็น ข้อสรุปทางลอจิก (Logical Consequence) ของ (P->S), (S->U) และ P

1.1.3 เพรดิเคตลอจิกลำดับที่หนึ่ง (First-Order Predicate Logic)

ใน พล็อบโพซิชันนัลลอจิก นั้นเราทราบว่าหน่วยที่เล็กที่สุดของข้อมูลก็คืออะตอมและจากอะตอมเราก็สามารถที่จะสร้างเป็น สูตร ได้ อย่างไรก็ตามใน พล็อบโพซิชันนัลลอจิกแต่ละอะตอมจะใช้แทนประโยคที่มีค่าความจริงเป็นจริงหรือเท็จเท่านั้น แต่ก็ยังมีเงื่อนไขบางประการที่ไม่สามารถใช้ พล็อบโพซิชันนัลลอจิก เขียนให้อยู่ในรูปแบบของ สูตร ได้เช่นจากประโยคต่อไปนี้

P : Every man is mortal.

Q : Confucius is a man.

R : Confucius is mortal.

จากตัวอย่างข้างต้นจะเห็นได้ว่าเราไม่อาจพิสูจน์ได้ว่า R เป็น ลำดับทางลอจิกของ P และ Q ใน พล็อบโพซิชันนัลลอจิก ทั้งนี้เพราะโครงสร้างของ P,Q และ R นั้นไม่สามารถจะแทนด้วย พล็อบโพซิชันนัลลอจิก ได้ ในบทนี้เราจะเราจะโครงสร้างการแทนข้อมูลในอีกรูปแบบหนึ่งที่เรียกว่า เพรดิเคตลอจิกลำดับที่หนึ่ง (First-Order Predicate Logic,FOPL) ซึ่งมีความเหมาะสมในการแทนความจริงมากกว่า เพรดิเคตลอจิกลำดับที่หนึ่ง มีองค์ประกอบอยู่ 3 ส่วนได้แก่ เทอม(terms), เพรดิเคต(predicates), ตัวบ่งปริมาณ(quantifier)

และในเช่นเดียวกันกับ พล็อบโพซิชันนัลลอจิก เราจะเริ่มด้วยการกำหนด อะตอมซึ่งเป็นหน่วยที่เล็กที่สุดของข้อมูล เราลองมาดูตัวอย่างต่อไปนี้

สมมติว่าเราต้องการแทนประโยค "x is greater then 3" โดยใช้ เพรดิเคตลอจิกลำดับที่หนึ่งเราสามารถทำได้โดยกำหนดพรีดิเคต GREATER(x,y) ซึ่งจะมีความหมายว่า x มีค่ามากกว่า y (เราอาจจะมองได้ว่าพรีดิเคตก็คือความสัมพันธ์ (Relation)) ดังนั้นประโยคข้างต้นเรา สามารถแทนโดยเขียนว่า GREATER(x,3)

ในลักษณะเช่นเดียวกันเราจะแทนประโยค "x loves y" ด้วยพรีดิเคต LOVE(x,y) ดังนั้นประโยคที่ว่า "John loves mary" ก็จะสามารถแทนได้โดยใช้พรีดิเคต LOVE(John,mary)

นอกจากที่ได้กล่าวมาแล้วนั้น เราสามารถที่จะมีการแทนในลักษณะที่เป็นฟังก์ชันใน เพรดิเคตลอจิกลำดับที่หนึ่ง ได้อีกด้วยเช่นเราอาจใช้ plus(x,y) ในการแทนความหมายของ x+y ดังนั้นสำหรับ ประโยคที่ว่า "x+1 greater then x" ก็จะสามารถเขียนให้อยู่ในรูปของ พรีดิเขตได้เป็น GREATER(plus(x,1),x)

และสิ่งที่ได้กล่าวมาทั้งหมดไม่ว่าจะเป็น GREATER(x,3) LOVE(John,mary) GREATER(plus(x,1),x) ล้วนจัดเป็นอะตอมในเพรดิเคตลอจิกลำดับที่หนึ่ง ทั้งสิ้นโดยที่ GREATER และ LOVE เป็นเครื่องหมายแสดงพรีดิเขต (Predicate Symbol) ขณะที่ x เป็น ตัวแปร และ 3,John,mary เป็นค่าคงที่ สำหรับ plus นั้นจัดเป็นฟังก์ชัน ดังนั้นหากจะกล่าว โดยสรุปก็คือในเพรดิเคตลอจิกลำดับที่หนึ่ง จะมีองค์ประกอบอยู่ 4 ชนิดคือค่าคงที่ ตัวแปร ฟังก์ชัน และพรีดิเขต เท่านั้นที่จะสามารถประกอบขึ้นมาเป็นพรีดิเขตได้

สำหรับฟังก์ชันแล้ว หากจะสังเกตให้ดีจะพบว่า ฟังก์ชันนั้นจะทำหน้าที่ในการแมป (Mapping) ระหว่างค่าคงที่ไปยังค่าคงที่ เช่น ฟังก์ชัน father(John) นั้นก็จะใช้ในการแทน บุคคลอีกคนหนึ่งเช่นกัน และเราเรียก father(John) นี้ว่าเทอม (Term) โดยที่สำหรับคำว่า เทอมในเพรดิเคตลอจิกลำดับที่หนึ่ง แล้วจะหมายถึงค่าคงที่ ตัวแปร หรือฟังก์ชัน เช่น plus (plus(x,1),1), father(father(John)) ล้วนจัดเป็นเทอม

แต่สำหรับกรณีของพรีดิเขตแล้วจะทำหน้าที่ในการแมประหว่างค่าคงที่กับค่าความจริง ซึ่งได้แก่ T และ F เช่นหาก GREATER เป็นพรีดิเขต ดังนั้น GREATER(5,3) จะเป็นจริง ขณะ ที่ GREATER(1,3) จะเป็นเท็จ หลังจากที่เราได้กำหนดความหมายของคำว่าเทอมต่อไปเราจะ กำหนดความหมายของอะตอมใน เพรดิเคตลอจิกลำดับที่หนึ่ง โดยกำหนดว่าหาก P เป็นเครื่อง หมายพรีดิเขต (Predicate Symbol) และ t1,...,tn เป็นเทอมแล้ว P(t1,...,tn) คือ อะตอม

เมื่อเราได้ทำการกำหนดอะตอมเป็นที่เรียบร้อยแล้ว เราก็สามารถใช้ตัวเชื่อมทาง ตรรก (Logical Connective) ได้เหมือนกับใน พร็อบโพซิชันนัลลอจิก เพื่อใช้ในการสร้าง สูตร และยิ่งไปกว่านั้นในเพรดิเคตลอจิกลำดับที่หนึ่ง เรายังกำหนดเครื่องหมาย E และ A โดย จะเรียกว่า Existential และ Universal Quantifier โดยที่หาก x เป็นตัวแปรแล้ว Ax จะหมายถึงสำหรับ x ทุกตัว (For al) ขณะที่ Ex หมายถึงสำหรับ x บางตัว (there exists an x) ลองดูตัวอย่างต่อไปนี้

สมมติว่าเรามีประโยค

1) Every rational number is a real number.

2) There exists a number that is a prime.

3) For every number x, there exists a number y such that x < y.

กำหนดให้ประโยค "x is a prime number" แทนด้วย P(x), "x is a rational number" แทนด้วย R(x) และ "x is less than y" แทนด้วย LESS(x,y) ดังนั้นสำหรับประโยคทั้ง 3 ข้างต้น เราสามารถเขียนให้อยู่ในรูปของ เพรดิเคตลอจิกลำดับที่หนึ่งได้ดังนี้

1) (Ax) (Q(x) -> R(x))

2) (Ex) P(x)

3) (Ax) (Ey) LESS(x,y)

แต่จะเรียกทั้งลักษณะการเขียนในลักษณะเช่นนี้ว่า สูตร แต่ก่อนที่เราจะกล่าวถึงเรื่องของ สูตร ต่อไป เราจะมาทำความรู้จักกับคำ 2 คำคือ ฟรี (free) และ บาว (bound) โดย โดยที่หากตัวแปรใดที่อยู่ใน สูตร จะถูกเรียกว่า บาว ก็ต่อเมื่อเกิดขึ้นอยู่ในช่วงของ ตัวบ่งปริมาณ ของตัวแปรนั้น และจะถูกเรียกว่า ฟรี ก็ต่อเมื่อตัวแปรนั้นไม่เป็นแบบ บาว เช่นสมมติว่าเรามี สูตร (Ax) P(x,y) ตัวแปร x จะเป็นตัวแปรแบบ บาว แต่สำหรับตัวแปร y นั้นจะเป็นแบบ ฟรี อย่างไรก็ตามสำหรับตัวแปรหนึ่งใน สูตร หนึ่งอาจจะเป็นทั้งแบบ บาว และ ฟรี เลยก็เป็นได้ เช่น (Ax) P(x,y) ^ (Ay) Q(y) ตัวแปร y จะเป็นทั้งแบบ บาว และ ฟรี

จากรูปแบบที่ได้กล่าวมาทั้งหมด จะเห็นได้ว่าเราสามารถใช้เพรดิเคตลอจิกลำดับที่หนึ่ง ในการแทนความจริงต่าง ๆ ในชีวิตประจำวันได้

1.1.3.1 <u>การแทนค่าของ สูตร ใน เพรดิเคตลอจิกลำดับที่หนึ่ง</u>

ใน พร็อบโพซิชันนัลลอจิก การแทนค่าก็คือการกำหนดค่าความจริงให้กับอะตอม แต่สำหรับ เพรดิเคตลอจิกลำดับที่หนึ่ง แล้วเนื่องจากมีตัวแปรที่เกี่ยวข้องมากกว่า ดังนั้นการการกำหนดการแทนในเพรดิเคตลอจิกลำดับที่หนึ่ง เราจะใช้วิธีกำหนด 2 ประการ ประการแรกคือกำหนดโดเมน (Domain) และกำหนดค่าคงที่ ฟังก์ชัน และพรีดิเคต เช่น หากเรามี สูตร (Ax) P(x) และ (Ex) ~P(x)

ขั้นแรกเราจะกำหนดให้โดเมนเป็น {1,2} และกำหนดค่าให้ P

P(1)	P(2)
T	F

จากตัวอย่างที่เรากำหนดค่า จะเห็นได้อย่างชัดเจนว่า (Ax) P(x) นั้นเป็นเท็จในการแทนค่าเพราะ P(x) ไม่เป็นจริงในทั้งสองกรณี แต่ (Ex) ~P(x) เป็นจริงเพราะมีอย่างน้อย 1 ค่าของการแทนที่ทำให้ ~P(x) เป็นจริง

จะเห็นได้ว่าเราไม่สามารถใช้เทคนิคของการสร้างตารางค่าความจริงใน เพรดิเคตลอจิกลำดับที่หนึ่งได้ในการหาว่า สูตร ใดเป็น T หรือ F นั้น จะต้องอาศัยการแทนค่าที่ประกอบด้วยการกำหนดขอบเขตของการแทนค่าหรือที่เรียกว่าโดเมน จากนั้นจึงแทนค่าในโดเมนให้กับส่วนของ พรีดิเคตและใช้การพิจารณาจึงสามารถระบุค่าความจริงได้

1.1.4 การโปรแกรมเชิงตรรก (Logic Programming)

จากที่ได้กล่าวมาแล้วทั้งหมดนั้นเป็นทฤษฎีทางด้าน ลอจิก ซึ่งจะเห็นได้ว่ามีความยุ่งยากซับซ้อนมาก ปัญหาบางปัญหาเมื่อแทนด้วย เพรดิเคตลอจิกลำดับที่หนึ่ง แล้วการแก้ปัญหาจะยุ่งยากมาก จนกระทั่งมีการนำคอมพิวเตอร์เข้ามาใช้ในการแก้ปัญหาทาง เพรดิเคตลอจิกลำดับที่หนึ่ง ซึ่งทฤษฎีที่นับว่ามีความสำคัญ อันก่อให้เกิดก้าวกระโดดในการศึกษาวิทยาการในสาขานี้ได้แก่ รีโซลูชัน (Resolution) ซึ่งคิดโดย Robinson [1979] โดยมีขอบเขตอยู่ว่าปัญหาจะต้องอยู่ในรูปของ ฮอร์นคลอส เท่านั้น

วิธีการ รีโซลูชัน นี้มีข้อดีหลายประการคือ ง่าย มีความถูกต้องสูง และมีความสมบูรณ์ในตัวเอง ซึ่งทฤษฎี รีโซลูชัน นี่เองที่เป็นรากฐานของภาษาโปรลอก (Prolog) [Roussel 1975] ซึ่งเป็นภาษาที่นิยมใช้มากในการโปรแกรมเชิงตรรก

1.1.4.1 รีโซลูชัน (Resolution)

ทฤษฎี รีโซลูชัน (Resolution) นี้จริงๆ แล้วก็เป็น กฎการวินิจฉัย (Inference Rule) อันหนึ่ง โดยมีข้อดีที่ว่าเหมาะกับการใช้งานโดยคอมพิวเตอร์ ทฤษฎี รีโซลูชัน มีหลักการดังต่อไปนี้ คลอส (Clause) จำนวน 2 คลอส สามารถที่จะแก้ปัญหา (Resolve) กับอีก คลอส ได้หากใน คลอส หนึ่งมี ลิทเทอร์รัล (Literal) ที่เป็นบวก (Positive) และอีก คลอส หนึ่งมี ลิทเทอร์รัล ที่เป็นลบ (Negative) ที่มาจากพรีดิเคตเดียวกัน และ อาร์กิวเมนต์

(Argument) ของ คลอส 2 คลาสนี้สามารถที่จะรวมกัน (Unified) กันได้ ลองมาดูตัวอย่างต่อไปนี้

$$P(a) \vee \neg Q(b,c) \qquad (1)$$

$$Q(b,c) \vee \neg R(b,c) \qquad (2)$$

เนื่องจากใน คลอส ที่ 1 มี ลิทเทอร์รัล ที่เป็นลบคือ ~Q(b,c) ในขณะที่ คลอส ที่ 2 มี ลิทเทอร์รัล ที่มาจากพรีดิเขตเดียวกันหากเป็น ลิทเทอร์รัล ที่เป็นบวกและ อาร์กิวเมนต์ของทั้ง 2 ลิทเทอร์รัล สามารถที่จะรวม (Unified) กันได้ นั่นคือ b ก็จะรวมกับ b และ c ก็จะรวมกับ c ดังนั้นผลรวมของ คลอส (1) และ (2) โดยวิธีการ รีโซลูชั่นนี้คือ คลอส ที่ (3) ดังข้างล่างนี้ และจะเรียก คลอส ที่ (3) นี้ว่า รีซอลเวนท์ (Resolvent)

$$P(a) \vee \neg R(b,c) \qquad (3)$$

1.1.4.2 <u>โปรลอก</u>

จากเฟิร์สออร์เดอร์เพรดิเคทโลจิก และ รีโซลูชั่น ได้มีนักวิจัยที่ชื่อว่า Kowalski ซึ่งมีส่วนสำคัญต่อการพัฒนาภาษาโปรลอกได้มีความเห็นว่าภาษาทางตรรก (Logic Language) น่าจะสามารถนำมาใช้เป็นภาษาทางโปรแกรมได้ (Progeamming Langiage) และนี่ก็เป็นจุดเริ่มต้นของภาษาโปรลอก

1.1.4.2.1 <u>โครงสร้างของภาษาโปรลอก</u>

ภาษาโปรลอกเป็นภาษาทางคอมพิวเตอร์ที่ ใช้ในการแก้ปัญหาที่เกี่ยวข้องกับวัตถุ (Objects) และความสัมพันธ์ระหว่างวัตถุนั้น ตัวอย่างเช่นจากประโยคที่ว่า "John owns the book" เราจะกำหนดความสัมพันธ์ชื่อ Ownership (การเป็นเจ้าของ) ระหว่างวัตถุ "John" และ "the book" และนอกจากนั้นความสัมพันธ์นี้ จะต้องเป็นความสัมพันธ์ที่มีลำดับมีกด้วย เพราะ "John" เป็นเจ้าของหนังสือ แต่หนังสือไม่ได้เป็นเจ้าของ "John"

แต่ความสัมพันธ์ที่กล่าวถึงนี้ก็ไม่ได้หมายถึงความสัมพันธ์ระหว่างวัตถุเสมอไป เช่นจากประโยคที่ว่า "The Jewel is valuable" นั่นคือความสัมพันธ์ที่จะเกี่ยวข้องกับวัตถุ "Jewel" ก็คือ "being valuable" ซึ่งจะเห็นได้ว่าไม่มีวัตถุอีกตัวหนึ่งที่จะสัมพันธ์ด้วย ในภาษาโปรลอกก็จะต้องมีการกำหนดความสัมพันธ์ในลักษณะเช่นนี้ และขึ้นอยู่กับว่าเราจะนำคอมพิว

เตอร์ไปใช้งานในรูปแบบใด

อย่างไรก็ตามในโปรลอกนั้นยังมีรูปแบบความสัมพันธ์ระหว่างวัตถุที่สำคัญที่ยังจะต้องกล่าวถึงอยู่อีกรูปแบบหนึ่ง ก่อนที่เราจะเริ่มกล่าวถึงการเขียนโปรแกรม ความสัมพันธ์ในรูปแบบนั้นโดยทั่วไปเรามักจะคุ้นเคยกับมันอยู่แล้ว โดยที่เราอาจจะไม่รู้ตัว นั่นคือการใช้กฎในการอธิบายความสัมพันธ์ระหว่างวัตถุต่างๆ เช่นจากประโยคที่ว่า "Two people are sisters if they are both female and have the same parents" ซึ่งหมายความว่าทั้งสองคนจะเป็น sister กันนั้นเมื่อทั้งสองคนเป็นผู้หญิงและมีพ่อแม่เดียวกัน

เมื่อเราได้กล่าวถึงความสัมพันธ์ในรูปแบบต่างๆ แล้ว ต่อไปเราก็จะกล่าวถึงการโปรแกรมภาษาโปรลอกในคอมพิวเตอร์ ซึ่งจะประกอบด้วยขั้นตอนต่อไปนี้

- กำหนดความจริง (facts) เกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น
- กำหนดกฎ (Rule) เกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น
- ถามคำถาม (Query) ที่ต้องการเกี่ยวกับเกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุนั้น

ตัวอย่างเช่น หากเรากำหนดกฎเกี่ยวกับ sister ที่ได้กล่าวไปแล้ว จากนั้นหากเราถามโปรลอกว่า Mary และ Jane เป็น sister กันหรือไม่ โปรลอกจะค้นหาว่าเราได้กำหนดอะไรไว้ในโปรแกรมบ้าง และจากนั้นก็จะตอบว่า yes หรือ no ซึ่งก็จะขึ้นกับว่าความจริงที่เราได้กำหนดไว้ในโปรแกรมนั้น สอดคล้องกับกฎที่เรากำหนดไว้หรือไม่ ดังนั้นหากจะมองได้ดีแล้วจะเห็นได้ว่าภาษาโปรลอกนั้นแท้จริงก็เพียงแต่เป็นที่เก็บความจริงและกฎ และความสามารถในการหาคำตอบจากความจริงและกฎนั้นเท่านั้น

1.1.4.2.1.1 <u>ความจริง</u> (facts)

ในขั้นแรกนี้เราจะกล่าวถึง ความจริง (facts) เกี่ยวกับวัตถุต่างๆ สมมติว่าเราต้องการกำหนดในภาษาโปรลอกว่า "John likes Mary" ซึ่งก็จะมีวัตถุ 2 อย่างคือ Mary และ John และจะเรียกความสัมพันธ์นี้ว่า likes และในภาษาโปรลอกเราจะกำหนดความจริงนี้ ดังนี้

```
likes(John,Mary).
```

ซึ่งจะเห็นว่ามีส่วนที่สำคัญ 3 ส่วนดังนี้

- ชื่อของทุกความสัมพันธ์ (Relation) และวัตถุ (Objects) จะต้องเริ่มต้นด้วยอักษรตัวเล็ก

- จะต้องเขียนความสัมพันธ์ก่อนและคั่นแต่ละวัตถุด้วยเครื่องหมาย "," และใช้เครื่องหมายวงเล็บครอบทุกวัตถุ

- จะต้องใช้เครื่องหมาย "." ปิดท้าย ความจริง (facts)

เมื่อเราได้กำหนดความสัมพันธ์ระหว่างวัตถุโดยใช้ ความจริง แล้วต่อไปเราจะเรียกชื่อของความสัมพันธ์ว่าพรีดิเคต (Predicate) และเรียกแต่ละวัตถุว่า อาร์กิวเมนต์ (Arguments) ดังนั้น ความจริง ที่ว่า likes(John,mary) ก็เป็นพรีดิเคตที่มี 2 อาร์กิวเมนต์ และจำนวน อาร์กิวเมนต์ ในพรีดิเคตจะมีเท่าใดก็ได้ ตั้งแต่ 1 ขึ้นไป และในโปรลอกกลุ่มของความจริง จะเรียกว่า ฐานข้อมูล

1.1.4.2.1.2 คำถาม (Questions)

เมื่อเราได้กำหนดความจริงบางประการให้กับโปรลอก เราสามารถจะถามปัญหาบางอย่างได้ ในภาษาโปรลอกคำถามจะมีความคล้ายกับ ความจริง (fact) มากเว้นแต่จะมีเครื่องหมายพิเศษเท่านั้น เช่น

```
?- owns(mary,book).
```

จากคำถามดังกล่าวก็จะมีความหมายว่า mary เป็นเจ้าของ book หรือไม่ และเมื่อเราถามคำถามดังกล่าวกับโปรลอก โปรลอกจะค้นหาใน ฐานข้อมูล ของมันว่ามี ความจริงดังกล่าวกำหนดไว้หรือไม่ หากมีโปรลอกก็จะตอบว่า yes มิฉะนั้นจะตอบว่า no เช่นหากเรามี ฐานข้อมูล ต่อไปนี้

```
likes(Joe,fish).
likes(Joe,mary).
likes(mary,book).
likes(John,book).
```

เมื่อเราใส่ ความจริง ทั้งหมดในโปรลอก และหากเราถามคำถามต่อไปนี้โปรลอกก็จะตอบมาในบรรทัดต่อไปที่เราถามคำถาม

```
?- likes(Joe,money).
no
?- likes(mary,Joe).
```

no

?- likes(mary,book).

yes

? king(John,france).

no

ในตอนต่อไป เราจะกล่าวถึงคำถามในลักษณะที่ว่า ใครบ้างที่ mary ชอบ

1.1.4.2.1.3 ตัวแปร (variable)

จากที่ผ่านมาเราจะสามารถถามได้ในลักษณะที่ว่า "Does John like Mary?" ซึ่งจะได้คำตอบในลักษณะของ yes หรือ no แต่สำหรับคำถามในลักษณะของ "Does John like X" เราไม่ทราบว่า X แทนอะไร แต่สำหรับโปรลอกแล้ว เราไม่เพียงแต่ใช้ชื่อเฉพาะเท่านั้น แต่ยังสามารถใช้ชื่อในลักษณะของ X ในการแทนวัตถุ โดยจะเรียกว่าตัวแปร (Variable) และในโปรลอกจะใช้คำที่ขึ้นต้นด้วยอักษรตัวใหญ่ในการแทนตัวแปร

เมื่อโปรลอกถามปัญหาที่มีตัวแปรอยู่ โปรลอกจะค้นหาผ่าน ความจริง ทั้งหมดที่มีอยู่ ซึ่งเช่นคำถามที่ว่า :- likes(John,X) และ ฐานข้อมูล ข้างต้นโปรลอกก็จะพยายามที่จะหาพรีดิเคตที่ชื่อ likes และมี John เป็น อาร์กิวเมนต์ ตัวแรก จากนั้นก็จะแทน X ด้วย อาร์กิวเมนต์ ตัวที่ 2 ของพรีดิเคตนั้น ซึ่งจะได้คำตอบว่า

X = book

1.1.4.2.1.4 ตัวเชื่อม (Conjunctions)

สมมติว่าเราถามคำถามที่มีความซับซ้อนขึ้นไปอีก เช่น "Do John and Mary like each other?" วิธีหนึ่งที่เราจะหาคำตอบก็คือขั้นแรกก็จะหาว่า John likes mary. หรือไม่จากนั้นก็จะถามว่า mary likes John. หรือไม่ ดังนั้นจะเห็นว่าคำถามประกอบด้วย 2 เป้าหมายย่อย (sub goal) ที่โปรลอกจะต้องหา แต่คำถามในลักษณะนี้จะมีการใช้บ่อยมากและสามารถถามได้ในลักษณะต่อไปนี้

?- likes(John,mary),likes(mary,John).

สมมติว่าเรามี ฐานข้อมูล

likes(mary,food).

likes(mary,wine).

likes(John,wine).

likes(John,mary).

ในที่นี้เครื่องหมาย "," จะใช้ในลักษณะความหมายว่า "และ" (And) และใช้แบ่งเป้าหมายย่อยที่ประกอบกันเป็นคำตอบ เมื่อลำดับของเป้าหมาย (Goal) ที่คั่นด้วย "," ถูกใช้ในโปรลอก โปรลอกจะพยายามหา ความจริง (fact) ที่สอดคล้องกับแต่ละเป้าหมายย่อยในคำถาม หากมี ความจริง ที่สอดคล้องกับทุกๆ เป้าหมายย่อย โปรลอกก็จะใช้คำตอบว่า yes มิฉะนั้นก็จะเป็น no ซึ่งในกรณีนี้จะเห็นได้ว่าเป้าหมายที่ 2 เป็นเท็จ ดังนั้นคำตอบจึงเป็น no

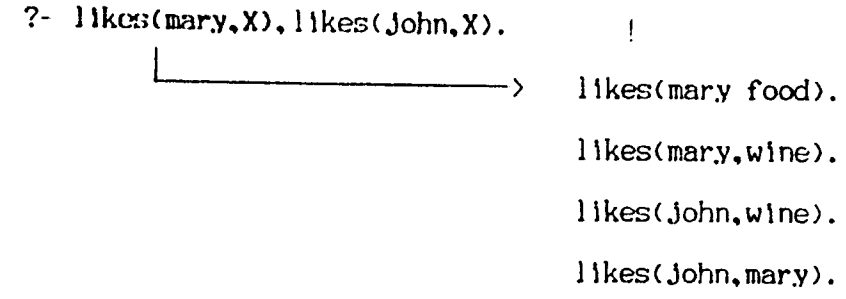
การใช้ตัวเชื่อมและการใช้ตัวแปรสามารถใช้ร่วมกัน เพื่อถามคำถามที่ซับซ้อนกว่านี้ได้ เช่นคำถามที่ว่า "Is there anything that John and Mary both like?" จากคำถามนี้ประกอบด้วยเป้าหมายย่อยดังนี้ ทำการหาว่ามีบางอย่าง X ที่ Mary ชอบและจากนั้นหาว่า John ชอบ X เช่นเดียวกันหรือไม่ สามารถเขียนเป็นภาษาโปรลอกได้ดังนี้

?- likes(mary,X),likes(John,X).

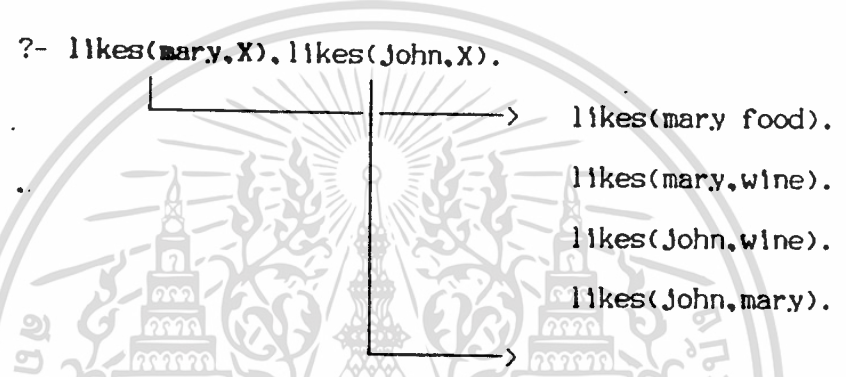
โปรลอกจะพยายามหาคำตอบของเป้าหมายแรกก่อน หากพบเป้าหมายแรกใน ฐานข้อมูล ก็จะพยายามหาคำตอบของเป้าหมายที่ 2 ต่อไป หากพบว่าเป้าหมายที่ 2 อยู่ใน ฐานข้อมูลก็จะ ได้คำตอบออกมา

อย่างไรก็ตามในกรณีของคำถามในลักษณะเช่นนี้ โปรลอกจะสร้าง ตัวระบุ (Marker) ขึ้นมาสำหรับแต่ละเป้าหมายย่อย เพราะหากว่าเป้าหมายที่ 2 เกิดเป็นเท็จขึ้นมา โปรลอกก็จะค้นหาเป้าหมายที่ 1 ใน ฐานข้อมูล ต่อจากเดิมเพื่อหาคำตอบอื่นที่สอดคล้องกับเป้าหมายแรกได้อีก กลไกเช่นนี้เราเรียกว่า การย้อนรอย (backtracking) ซึ่งเป็นลักษณะเฉพาะตัวของภาษาโปรลอก และเพื่อให้เข้าใจกลไกในการ ย้อนรอย (backtrack) ดีขึ้นลองมาดูการทำงานทีละขั้นตอนดังนี้

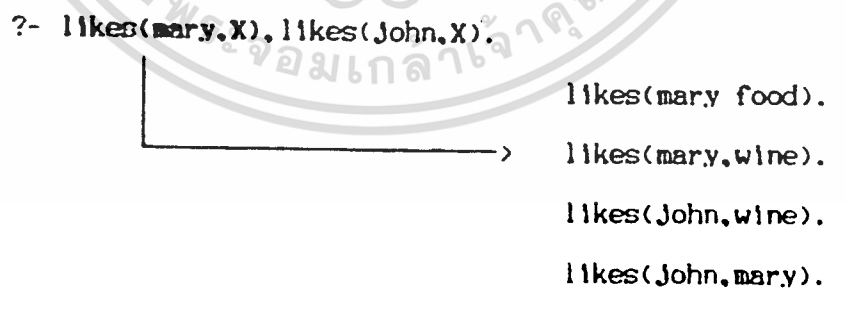
จากข้อมูลเดิมที่ผ่านมา และคำถามที่ว่า



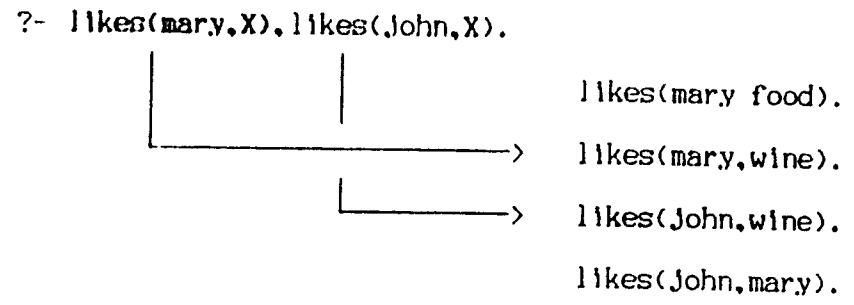
จะเห็นว่าเป้าหมายแรกมี ความจริง ที่สอดคล้องอยู่ในฐานข้อมูลได้แก่ likes(mary, food). ดังนั้น ตัวระบุ ของเป้าหมายแรกจะอยู่ที่ ความจริง ดังกล่าว และ X ก็จะมีค่าเป็น food จากนั้นจะพยายามหาคำตอบของเป้าหมายที่ 2 คือ likes(John,food). ซึ่งไม่มีใน ฐานข้อมูล



จากนั้นก็จะทำการ ย้อนรอย โดยทิ้งค่าเดิมของ X ไปและทำการหาค่าของ X ใหม่ จากเป้าหมายย่อยแรกต่อจากเดิมที่ ตัวระบุ ของเป้าหมายแรกชี้ไว้ ซึ่งก็จะได้คำตอบว่า X มีค่าเป็น wine



จากนั้นก็จะหาคำตอบของเป้าหมายที่ 2 ก็คือ likes(John,wine) ซึ่งปรากฏว่ามีใน ฐานข้อมูล ดังนั้นทั้งสองเป้าหมายก็เป็นจริง และได้คำตอบว่า X=wine



1.1.4.2.1.5 กฎ (Rule)

ที่นี้หากเราต้องการจะสร้างความสัมพันธ์ที่ว่า "John likes all person" หาก เราจะสร้างเป็น ความจริง เพื่อเก็บใน ฐานข้อมูล ก็จะต้องสร้างความสัมพันธ์กับทุกคน ดังนี้

```
likes(John,david).
like(John,mary).
...
```

และหากเราเพิ่มคนใดคนหนึ่งเข้าไปใน ฐานข้อมูล ก็จะต้องสร้างความสัมพันธ์ใน ลักษณะ เช่นนี้ขึ้นมาทุกครั้ง ในลักษณะ เช่นนี้จะเห็นได้ว่าเป็นรูปแบบที่ไม่สะดวกต่อการใช้งานอย่างยิ่ง และหากเราแทนด้วยอีกรูปแบบหนึ่งของโปรลอกคือกฎ (Rule) ซึ่งเป็นรูปแบบที่เหมาะกับการเก็บความจริงในลักษณะ เช่นนี้กว่า จะได้ดังนี้

```
likes(John,X) :- person(X).
```

โดยที่เครื่องหมาย ":-" จะแทนความหมายของ "IF" ดังนั้นในความสัมพันธ์ข้างต้นจะมีความหมายว่า "John likes any object provided it is a preson." ซึ่งจะสังเกตุได้ว่า กฎ ก็จะมีความหมายเหมือนกับเป็นการสร้างความสัมพันธ์ขึ้นมาใหม่บนความสัมพันธ์เก่า ซึ่งหากจะมองว่าข้างหลังของเครื่องหมาย ":-" เป็นเป้าหมายและหากเป้าหมายเป็นจริงก็จะ ได้คำตอบออกมา

และ เพื่อให้มีความเข้าใจกับกลไกของการใช้กฎในโปรลอก เราจะลองสมมติข้อมูลและสร้างกฎขึ้นมาบนข้อมูลต่อไปนี้

```
male(albert).
male(edward).
female(alice).
female(victoria).
parents(edward,victoria,albert).
parents(alice,victoral,albert).
```

และมีกฎว่า

```
sister_of(X,Y) :- female(X),
                  parents(X,M,F),
                  parents(Y,M,F).
```

โดยที่ความสัมพันธ์ sister_of นั้นหมายถึงความสัมพันธ์ของ X และ Y ซึ่งหมายความว่า X จะเป็น sister ของ Y เมื่อ X เป็นผู้หญิงและมีพ่อแม่เดียวกัน และเมื่อเราถามว่า :- sister_of(alice,edward). โปรลอกก็จะทำงานตามขั้นตอนต่อไปนี้

- จากคำถามเมื่อค้นหาใน ฐานข้อมูล ก็จะพบว่า sister_of เป็นกฎก็จะทำการเทียบก็จะได้ว่า X=alice และ Y=edward จากนั้นโปรลอกก็จะพยายามที่จะหาคำตอบของเป้าหมายย่อยทั้ง 3 ต่อไป

- ครั้งแรกจะทำการค้นหาว่า female(alice). เป็นจริงหรือไม่ ซึ่งก็จะพบว่าเป็นจริงดังนั้นก็จะทำการค้นหาเป้าหมายที่ 2 ต่อไป

- จาก parents(alice,M,F) ก็จะได้ว่า M=victoria และ F=albert

- จากนั้นก็จะค้นหา ความจริง parents(edward,victoria,albert) ซึ่งปรากฏว่ามีในฐานข้อมูล

จากขั้นตอนทั้งหมดที่ผ่านมาก็จะพบว่าความเป้าหมายย่อยทั้ง 3 เป็นจริง ดังนั้นโปรลอกก็จะตอบกลับมาว่า yes อย่างไรก็ตามในแง่ของกฎแล้วเราสามารถที่สามารถตั้งคำถามในลักษณะของ sister_of(alice,X) ได้อีกด้วย ซึ่งกลไกในการหาคำตอบของกฎนี้ก็จะคล้ายกับกลไกการหาคำตอบแบบธรรมดา รวมทั้งการทำ backtracking ด้วย ซึ่งจะไม่กล่าวในที่นี้

ที่ได้กล่าวมาทั้งหมดนี้ เพื่อเป็นแนวทางและเป็นทฤษฎีพื้นฐานที่จะใช้ในบทต่อไป

1.2 ตรรกและฐานข้อมูลแบบสัมพันธ์ (Logic and Relational Database)

ภาษาเชิงตรรก และ ฐานข้อมูลแบบสัมพันธ์นั้นมีพื้นฐานทางคณิตศาสตร์เดียวกันคือ เฟิร์สออร์เดอร์ลอจิก เราสามารถเก็บกฎและข้อเท็จจริงในฐานข้อมูลแบบสัมพันธ์ได้ โดยฐานข้อมูลที่เก็บข้อเท็จจริงถูกเรียกว่า ฐานข้อมูลแบบเอกซ์เทนชันนัล (Extensional Database)ส่วนฐานข้อมูลที่ใช้เก็บกฎถูกเรียกว่า ฐานข้อมูลแบบอินเทนชันนัล (Intensional Database)

ขอให้พิจารณาฐานข้อมูลแบบสัมพันธ์ต่อไปนี้

ตาราง Person

Name	Sex	Father	Mother
halvard	m	tore	catherine
tore	m	ole	regnhild
catherine	f	harold	helga
anne	f	tore	catherine
robin	m	harold	helga
ragnhild	f	olaf	alma

ตาราง Car

Number	make	owner	colour
123	fiat	ole	brown
321	volvo	ragnhild	green
314	citroen	tore	blue
111	ferrari	catherine	yellow

ซึ่งตารางทั้งคู่นั้นคือการเก็บข้อเท็จจริงต่างๆดังนี้คือ

```
person(halvard,m,tore,catherine).
person(tore,m,ole,regnhild).
person(catherine,f,harold,helga).
person(anne,f,tore,catherine).
person(robin,m,harold,helga).
person(ragnhild,f,olaf,alma).
```

และ

```
car(123,fiat,ole,brown).
car(321,volvo,ragnhild,green).
car(314,citroen,tore,blue).
car(111,ferrari,catherine,yellow).
```

ซึ่งตัวอย่างข้างต้นเป็นฐานข้อมูลแบบเอกซ์เทนชันนัล ส่วนตัวอย่างการเก็บกฎในฐานข้อมูลแบบอินเทนชันนัลขอให้พิจารณากฎและข้อเท็จจริงต่อไปนี้

```
parent(A,B) :- father(A,B).
father(dang,dum).
father(green,dang).
```

กฎและข้อเท็จจริงดังตัวอย่างข้างต้น จะสามารถถูกเก็บในฐานข้อมูลแบบสัมพันธ์ได้ในลักษณะของวิว (View) และ ตาราง ดังนี้

```
create view PARENT(A,B) as
     select  A,B
     from    FATHER;
```

ตาราง FATHER

A	B
dang	dum
green	dang

เมื่อมีคำถามว่า " ใครเป็นผู้ปกครองของดำ ?" จะเทียบได้กับภาษาเชิงตรรกดังนี้

```
:- parent(A,dum)
```

และจะเทียบได้กับภาษาเอสคิวแอล (SQL) ดังนี้

```
Select  A
from  PARENT
where PARENT.A = "dum"
```

คำตอบสำหรับคำถามดังกล่าวก็คือ

```
A = dang
```

จากตัวอย่างทั้งสองดังกล่าวข้างต้นจะพบว่าฐานข้อมูลแบบสัมพันธ์สามารถนำมาใช้เก็บกฎและอนุมานกฎได้ ดังนั้นในโครงงานนี้จึงได้นำฐานข้อมูลแบบสัมพันธ์มาสร้างเป็นฐานข้อมูลแบบอนุมานกฎได้ แต่อย่างไรก็ตามจะพบว่ากฎที่ยกตัวอย่างไว้มิใช่กฎแบบเรียกตนเอง ซึ่งการอนุมานกฎแบบเรียกตนเองได้บนฐานข้อมูลนั้นมีความซับซ้อนกว่ามาก จึงจะขอกล่าวถึงในหัวข้อ "กฎแบบเรียกตนเอง"

1.3 ฐานข้อมูลแบบอนุมานกฎได้ (Deductive Database)

ฐานข้อมูลแบบอนุมานกฎได้คือ ฐานข้อมูลที่สามารถเก็บกฎและอนุมานกฎได้ สำหรับในโครงงานนี้ได้สร้างฐานข้อมูลแบบอนุมานกฎได้บนฐานข้อมูลแบบสัมพันธ์ โดยมีส่วนประกอบที่สำคัญคือ ส่วนเก็บข้อเท็จจริง, ส่วนเก็บกฎและส่วนอนุมานกฎ

ส่วนเก็บกฎ,ส่วนเก็บข้อเท็จจริง และการอนุมานกฎแบบไม่เรียกตนเองได้กล่าวไว้ในหัวข้อตรรกและฐานข้อมูลแบบสัมพันธ์แล้ว จะไม่ขอกล่าวถึงอีก แต่จะกล่าวถึงการอนุมานกฎแบบเรียกตนเองบนฐานข้อมูลในหัวข้อต่อไปนี้

1.3.1 การอนุมานกฎแบบเรียกตนเองบนฐานข้อมูล

การอนุมานกฎแบบเรียกตนเองบนฐานข้อมูลได้ใช้หลักการของ เฮนเชนและแนควิ (Henschen and Naqvi) ซึ่งคิดค้นในปี 1984 วิธีนี้ทำได้โดยการสร้างกฎแบบไม่เรียกตนเองจากกฎแบบเรียกตนเองหลังจากนั้นจะทำการแปลกฎที่ได้ไปเป็นภาษาเอสคิวแอล เพื่ออนุมานกฎต่อไป

กฎแบบเรียกตนเองมีรูปแบบดังนี้

L11 and L12 and ... -> L1n

L21 and L22 and ... -> L2n

.....

Lm1 and Lm2 and .... -> Lmn

โดยที่ ทางซ้ายมือของเครื่องหมาย '->' (IMPLICATION) มีชื่อเรียกว่า ส่วนตัว (BODY) ส่วนทางขวามือของเครื่องหมาย '->' มีชื่อเรียกว่า ส่วนหัว (Head)

กฎดังตัวอย่างข้างต้นจะเป็นกฎแบบเรียกตนเองก็ต่อเมื่อ ส่วนหัว L2n มีชื่อเดียวกับ ตัวใดตัวหนึ่งใน ส่วนตัวของกฎ L1n ,ส่วนหัว L3n มีชื่อเดียวกับ ตัวใดตัวหนึ่งในส่วนตัวของกฎ L2n , ... และส่วนหัว L1n มีชื่อเดียวกับ ตัวใดตัวหนึ่งในส่วนตัวของกฎ Lmn

วิธีของเฮนเชนและแนควินั้นจะสามารถตรวจสอบกฎว่าเป็นแบบเรียกตนเองหรือไม่ ได้โดยการแทนอนุประโยคทั้งหมดให้อยู่ในรูปของคอนเนคชันกราฟ (Connection Graphs หรือ CG หรือ ซีจี) และคอนเนคชันกราฟ ที่เชื่อมต่อกันเป็นวงจะถูกเรียกว่า การวนซ้ำแบบเรียกตนเองที่ซ่อนอยู่ภายใน (Potential Recursive Loop หรือ PRL หรือ พีอาร์แอล) อนุประโยคทุกอนุประโยคที่ประกอบกันเป็นพีอาร์แอลจะเป็นกฎแบบเรียกตนเองทั้งหมด ส่วนอนุประโยคที่อยู่นอกพีอาร์แอลจะเป็นกฎแบบไม่เรียกตนเอง

อนุประโยคที่เป็นคำถามที่นำเข้าสู่พีอาร์แอลถูกเรียกว่า อนุประโยคเริ่มต้น (start clause) ส่วนอนุประโยคที่จะทำให้ออกจากพีอาร์แอลได้คือ อนุประโยคสิ้นสุด (end clause)

วิธีของเฮนเชนและแนควิในการอนุมานกฎแบบเรียกตนเองนั้นมีส่วนที่สำคัญที่จะได้จากพีอาร์แอล 2 ส่วนคือ

- ส่วนตกค้างในการวนซ้ำที่เพิ่มขึ้น (Augment loop residue) ได้จากการแทนอนุประโยคเริ่มต้นด้วยอนุประโยคในพีอาร์แอลไปเรื่อยๆจนมาถึงอนุประโยคเริ่มต้น อีกครั้ง

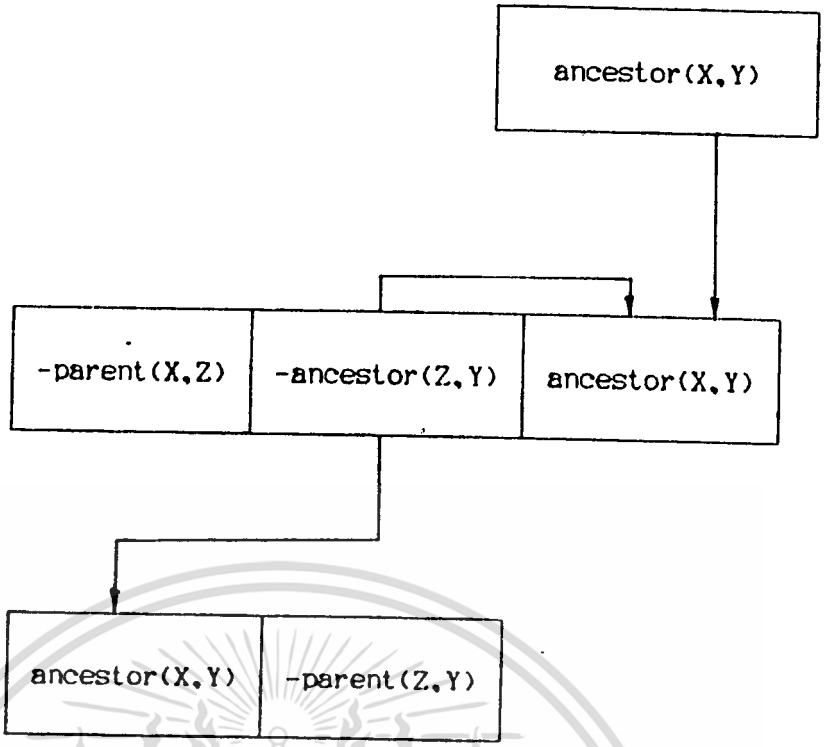
- นิพจน์ทางออกที่เพิ่มขึ้น (Augment exit expression) ได้จากการแทนอนุประโยคเริ่มต้นด้วยอนุประโยคในพีอาร์แอลไปเรื่อยๆในทิศทางที่สิ้นสุดจนถึงอนุประโยคสิ้นสุด

ขอให้พิจารณาตัวอย่างต่อไปนี้

```
ancestor(X,Y) :-  parent(X,Y).

ancestor(X,Y) :- ancestor(X,Z),parent(Z,Y).
```

เมื่อนำมาวาดเป็นกราฟพีอาร์แอลจะได้ดังรูปที่ 1-1



รูปที่ 1-1 รูปแสดงตัวอย่างของ PRL

จากรูปที่ 1-1 จะได้ ส่วนตกค้างในการวนซ้ำที่เพิ่มขึ้น และ นิพจน์ทางออกที่เพิ่มขึ้น ดังนี้

ส่วนตกค้างในการวนซ้ำที่เพิ่มขึ้น :

```
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
```

นิพจน์ทางออกที่เพิ่มขึ้น :

```
ancestor(X,Y) :- parent(X,Z),parent(Z,Y).
```

จากนั้นเราจะนำทั้งสองส่วนมาสร้างกฎแบบไม่เรียกตนเองไปเรื่อยๆ ด้วยการแทนอนุประโยคเริ่มต้นที่เป็นส่วนตัวในส่วนตกค้างในการวนซ้ำที่เพิ่มขึ้น ด้วยนิพจน์ทางออกที่เพิ่มขึ้น แล้วแปลงเป็นภาษาเอสคิวแอล เพื่อรับข้อมูลจนกระทั่งไม่ได้รับข้อมูลเพิ่มขึ้น นั่นย่อมหมายความว่าการอนุมานกฎได้เสร็จสิ้นลงแล้ว ดังตัวอย่างต่อไปนี้

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),parent(Z,Y).
ancestor(X,Y) :- parent(X,Z),parent(Z,D1),parent(D1,Y).
ancestor(X,Y) :- parent(X,Z),parent(Z,D1),parent(D1,D2),
                 parent(D2,Y).
```

จากกฎข้างต้นสามารถสร้างเป็นภาษาเอสคิวแอลได้ดังนี้

```
INSERT INTO ancestor
SELECT NAME1,NAME2
FROM parent

INSERT INTO ancestor
SELECT FT1.NAME1,FT2,NAME2
FROM parent FT1,parent FT2
WHERE FT1.NAME2 = FT2.NAME1

INSERT INTO ancestor
SELECT FT1.NAME1,FT3,NAME2
FROM parent FT1,parent FT2,parent FT3
WHERE FT1.NAME2 = FT2.NAME1 AND
      FT2.NAME2 = FT3.NAME1

INSERT INTO ancestor
SELECT FT1.NAME1,FT4,NAME2
FROM parent FT1,parent FT2,parent FT3,parent FT4
WHERE FT1.NAME2 = FT2.NAME1 AND
      FT2.NAME2 = FT3.NAME1 AND
      FT3.NAME2 = FT1.NAME1
```

บทที่ 2

ระบบฐานข้อมูลอนุมาน

2.1 บทนำ

ดังที่ได้กล่าวมาแล้วว่า ระบบฐานข้อมูลอนุมานได้รับการพัฒนาโดยมีพื้นฐานมาจากสาขาหลัก 2 สาขาด้วยกันคือ การโปรแกรมเชิงตรรก (Logic Programming) และระบบฐานข้อมูลแบบรีเลชันแนล (Relational Database System) และที่เรียกว่า "อนุมาน" (Deductive) นั้นก็เนื่องมาจากระบบฐานข้อมูลชนิดนี้มีความสามารถในการอนุมานข้อมูลใหม่จากข้อมูลที่มีอยู่แล้วได้ ซึ่งข้อมูลนี้ก็จะเก็บอยู่ในรูปของความจริง (Facts) และกฎ (Rule) (ระบบฐานข้อมูลอนุมานนี้อาจถูกเรียกในชื่ออื่นได้ เช่น ฐานข้อมูลเชิงตรรก (Logic Database) [Gallaire and Minker 1978], ฐานข้อมูลสัมพันธ์แบบอนุมาน (Deductive Relational Databases) [Minker 1982] และ ฐานข้อมูลสัมพันธ์แบบเสมือน (Virtual Relational Database) [Debanham and McGrath 1983] ซึ่งความสามารถในการอนุมานนั้นก็เป็นผลมาจากพัฒนาการทางด้านปัญญาประดิษฐ์นั่นเอง

ทฤษฎีทางด้านฐานข้อมูลอนุมาน สามารถที่จะมองได้ว่าเป็นส่วนที่เพิ่มเติมขึ้นมาจากทฤษฎีของระบบฐานข้อมูลแบบรีเลชันแนล โดยมองว่าฐานข้อมูลแบบรีเลชันแนลนั้นจะเก็บเฉพาะความจริง (Collection of Facts) ในขณะที่ระบบฐานข้อมูลอนุมานจะเก็บข้อมูลทั้งในรูปของความจริงและในรูปของกฎอีกด้วย โดยที่ทั้งความจริงและกฎจะแทนด้วย เฟิร์สออร์เดอร์เพรดดิเคทโลจิก (First Order Predicate Logic) และจากการใช้ เฟิร์สออร์เดอร์เพรดดิเคทโลจิก มาใช้นี่เองทำให้ฐานข้อมูลนั้นไม่เพียงแต่จะใช้เก็บข้อมูลเท่านั้น แต่จะเก็บทั้งโปรแกรม คำถาม วิว และกฎข้อบังคับเพื่อความถูกต้องของข้อมูลต่างๆ ได้

แรงบันดาลใจอีกประการหนึ่งของการพัฒนาระบบฐานข้อมูลอนุมาน ก็มาจากความจริงที่ว่า เฟิร์สออร์เดอร์เพรดดิเคทโลจิก นั้นสามารถใช้งานในลักษณะของโปรแกรมได้ ซึ่งทำให้เป็นที่สนใจมากในระยะหลังมานี้ จนกระทั่งได้ผลลัพธ์ออกมาเป็นภาษาที่สามารถใช้งานได้นั่นก็คือ ภาษาโปรลอก (PROgramming in LOGic) ภาษาโปรลอกเป็นภาษาที่มีลักษณะที่ง่ายและมีประสิทธิภาพสูง สามารถนำไปใช้งานได้ในหลายๆ ด้านรวมทั้งด้านปัญญาประดิษฐ์และระบบฐานข้อมูลด้วย

2.2 ระบบฐานข้อมูลอนุมาน

ระบบฐานข้อมูลอนุมานนั้น เมื่อมองในมุมของแนวคิดแล้ว ก็ไม่ใช่อะไรอื่นนอกจากโปรแกรมภาษาโปรลอกนั่นเอง นั่นคือลำดับของความจริงและกฎ (Facts ans Rules) อย่างไรก็ตามในทางปฏิบัติแล้วก็ยังมีความแตกต่างบางประการ ระหว่างโปรแกรมภาษาโปรลอกและระบบฐานข้อมูลอนุมาน กล่าวคือโปรแกรมภาษาโปรลอกมักจะประกอบด้วยกฎเป็นจำนวนมากขณะที่ใช้ความจริงเป็นจำนวนน้อย สำหรับระบบฐานข้อมูลอนุมานนั้นจะเป็นตรงข้ามกล่าวคือ จะประกอบด้วยความจริงเป็นจำนวนมากในขณะที่มีกฎเป็นจำนวนที่ไม่มากนัก และนั่นก็หมายความว่ามันเป็นไปไม่ได้ที่จะ โหลดข้อมูลทั้งหมดลงในหน่วยความจำเพื่อประมวลผล ตัวโปรแกรมภาษาโปรลอกจะต้องมีขนาดเล็กเพียงพอที่จะ โหลดลงในหน่วยความจำหลักได้ เพื่อที่จะง่ายต่อการอ้างถึงโดยตัวแปลภาษาแบบอินเทอร์พรีเตอร์ (Interpreter)

สำหรับระบบฐานข้อมูลอนุมานจะต้องเก็บความจริงในหน่วยความจำสำรองเช่น ดิสก์ และโหลดเฉพาะข้อมูลที่จะ ใช้ในการหาคำตอบเท่านั้น ลงในหน่วยความจำหลักดังนั้นระบบฐานข้อมูลอนุมานจึงต้องมีความสามารถในการจัดการกับไฟล์อีกด้วย

เราได้กล่าวว่าระบบฐานข้อมูลแบบรีเลชันแนลเป็นกรณีพิเศษของระบบฐานข้อมูลอนุมาน เพื่อที่จะแสดงว่าคำกล่าวนั้นเป็นจริง เราจะอธิบายโดยใช้ตัวอย่าง Supplier-Part จากหนังสือ An Introduction to Database Systems [Date 1981] ฐานข้อมูลนี้มีความสัมพันธ์อยู่ 3 ชนิดได้แก่ S เป็นความสัมพันธ์ของ Supplier, P เป็นความสัมพันธ์ของ Part และ SP เป็นความสัมพันธ์ระหว่าง Supplier และ Part และเพื่อให้ชัดเจน เราจะเขียนความสัมพันธ์ในรูปแบบต่อไปนี้

```
S(sno,sname,status,city)
P(pno,pname,colour,weight,city)
SP(sno,pno,qty)
```

ในแต่ละความสัมพันธ์สามารถแทนด้วยตารางที่มีข้อมูล อย่างไรก็ตาม เราจะเขียนโดยใช้รูปแบบของภาษาโปรลอก ดังต่อไปนี้

S(s1,smith,20,london)	SP(s1,p1)
S(s2,Jones,10,paris)	SP(s1,p2)
S(s3,blake,30,paris)	SP(s1,p3)
S(s4,clark,20,london)	SP(s1,p4)
S(s5,adams,30,athens)	SP(s1,p5)
	SP(s1,p6)
P(p1,nut,red,12,london)	SP(s2,p1)
P(p2,bolt,green,17,paris)	SP(s2,p2)
P(p3,screw,blue,17,rome)	SP(s3,p2)
P(p4,screw,red,14,london)	SP(s4,p2)
P(p5,cam,blue,12,paris)	SP(s4,p4)
P(p6,cog,red,19,london)	SP(s4,p5)

แต่ละ ทรัพเพิล ของความสัมพันธ์จะเขียนในรูปแบบที่คล้ายกับความจริง (fact) ในภาษาโปรลอก ซึ่งในความเป็นจริงแล้วหากจะตัดความแตกต่างเล็กน้อยออกไป เราอาจกล่าวได้ว่าทั้ง ทรัพเพิล และ ข้อเท็จจริง เป็นสิ่งเดียวกัน และนั่นก็หมายความว่าฐานข้อมูลแบบรีเลชันแนลจึงเรียกได้ว่าเป็นกรณีพิเศษของฐานข้อมูลอนุมาน เพราะฐานข้อมูลอนุมานนั้นจะมีส่วนของกฎอีกด้วย ซึ่งส่วนของกฎนี่เองที่ทำให้สามารถใช้งานในลักษณะของภาษาได้อีกด้วย

ที่เราได้กล่าวผ่านมาทั้งหมด เราได้ใช้ความสัมพันธ์ในลักษณะของตารางในการแทนความจริง (facts) หรือความรู้ (Knowledge) ซึ่งการแทนในลักษณะดังกล่าวนั้น ยังมีข้อบกพร่องหรือปัญหาอยู่บางประการ ซึ่งต่อไปเราก็จะมาพิจารณาปัญหานั้น

2.3 <u>การใช้ฐานข้อมูลแบบรีเลชันแนล (Relational Database) เป็นแบบแทนความรู้</u>

แบบจำลองข้อมูลที่นิยมในระดับตรรก (logical Data Model) มีอยู่ 3 แบบ คือ แบบรีเลชันแนล (Relational Model) แบบไฮราคี (Hierachical Model) และแบบโครงข่าย (Network Model) ซึ่งแบบที่กำลังเป็นที่นิยมและถูกสร้างขึ้นมาโดยทฤษฎีทางคณิตศาสตร์รองรับอย่างชัดเจนและมีความสัมพันธ์อย่างใกล้ชิดกับเพรดิเคตออร์เดอร์เพรดดิเคทโลจิก (FOPL) คือแบบรีเลชันแนล นักวิทยาศาสตร์เป็นจำนวนมากที่เป็นผู้บุกเบิกงานทางด้านความสัมพันธ์ระหว่างงานทั้งสองสาขานี้ ในจำนวนนี้ Kelter ถึงกับนิยามฐานข้อมูลแบบรีเลชันแนลขึ้นใหม่ตามกฎเกณฑ์

ทาง เฟิร์สออร์เดอร์เพรดดิเคทโลจิก โดยจะมองคำถาม (Query) แต่ละคำถามที่ให้แก่ระบบฐานข้อมูล ถ้ามองทางทฤษฎีฐานข้อมูล จะมองเป็นโจทย์ที่ระบบจัดการฐานข้อมูลจะต้องทำการค้นหาและดึงข้อมูลออกมาเป็นคำตอบ แต่ถ้ามองทางทฤษฎี เฟิร์สออร์เดอร์เพรดดิเคทโลจิก จะมองคำถามเป็นทฤษฎี (Theorem) ที่จะต้องพิสูจน์ว่าจริงหรือไม่ จากความจริง (fact) ที่เก็บอยู่ในฐานข้อมูล ผู้ที่พิสูจน์ว่าทฤษฎีทั้งสองแนวทางนี้ต่างให้ผลลัพธ์ที่เหมือนกันภายใต้เงื่อนไขบางประการ คือ ศาสตราจารย์ Kowalski แห่ง วิทยาลัยอิมพีเรียล แห่ง ลอนดอน (Imperial Collage London)

อย่างไรก็ดีการใช้แบบจำลองข้อมูลรีเลชันแนลเป็นแบบแทนความรู้ในระบบปัญญาประดิษฐ์โดยตรงยังมีอุปสรรคอยู่บางประการ ซึ่งแม้แต่ในบทความของ Kowalski เองก็ยังไม่ได้ให้คำตอบที่จะเป็นแนวทางในการแก้ไขอุปสรรคดังกล่าว

2.4 <u>การจัดการกับกฎในฐานข้อมูลอนุมาน</u>

จากที่กล่าวมาข้างต้นนั้น จะเห็นได้ว่าเราสามารถใช้แบบจำลองข้อมูลไนแอมเป็นแบบแสดงความรู้สำหรับฐานข้อมูลอนุมานในระบบปัญญาประดิษฐ์ซึ่งมีข้อดีหลายประการดังที่ได้กล่าวไปแล้ว แต่อีกสิ่งที่จะต้องคำนึงถึงในการใช้งานฐานข้อมูลอนุมานก็คือการจัดการกับกฎ (Rule) ซึ่งต่อไปเราก็จะกล่าวถึงกลไกที่จะใช้กับกฎเพื่อใช้กับฐานข้อมูลอนุมานที่ใช้แบบจำลองข้อมูลไนแอมเป็นฐานข้อมูลอนุมาน

2.4.1 <u>การจัดการกับกฎชนิดไม่รีเคอร์ซีฟ</u>

สมมติว่าเรามีฐานข้อมูลที่เก็บความสัมพันธ์ของวิชาบังคับของวิชาต่างๆ เช่น Imm_Prereq ('Data Base','Data Structure') หมายถึงความสัมพันธ์ที่ว่าการลงทะเบียนเรียนวิชา 'Data Base' จะต้องผ่านวิชา 'Data Structure' มาก่อน แต่สมมติว่าถ้าเราต้องการเก็บความสัมพันธ์ว่าผู้ที่เรียนวิชา 'Data Base' จะผ่านวิชาอะไรมาบ้าง (ทั้งนี้เพราะว่าวิชา 'Data Structure' ก็อาจจะมีวิชาบังคับที่จะต้องผ่านเสียก่อนได้เหมือนกัน) ซึ่งจะเห็นว่ากรณีเช่นนี้เราไม่จำเป็นต้องเก็บความสัมพันธ์นี้ลงในฐานข้อมูลจริงๆ แต่เราจะเก็บเป็นกฎของความสัมพันธ์นั้นแทนดังนี้ (เขียนในรูปแบบของภาษาโปรลอก)

```
Prereq(X,Y) :- Prereq(X,Z),Imm_Prereq(Z,Y)
```

ความสัมพันธ์ Imm_Prereq จะอยู่ในส่วนของ ฐานข้อมูลแบบเอกซ์เทนชันนัล (EDB) ซึ่งหมายถึงความสัมพันธ์ที่เก็บลงในฐานข้อมูลจริง ส่วน Prereq จะอยู่ในส่วนของฐานข้อมูลแบบอินเทนชันนัล (IDB) ซึ่งหมายถึงความสัมพันธ์ที่ไม่ต้องเก็บจริงในฐานข้อมูล แต่จะอนุมานได้จากกฎแห่งความสัมพันธ์นั้นแทน ภาษาที่ใช้ในระบบฐานข้อมูลในปัจจุบันนั้น ยังไม่มีกลไกในการจัดการกับกฎได้โดยตรง ในขณะที่ภาษาทางด้านตรรก เช่น โปรลอก มีกลไกที่เรียกว่า รีโซลูชัน (resolution) ที่จะใช้ในการจัดการกับกฎต่างๆ ได้ แม้กระนั้นเราก็ยังสามารถที่จะประยุกต์ใช้ความสามารถบางอย่าง ของระบบฐานข้อมูลเพื่อใช้ในการจัดการกับกฎดังกล่าวนี้ได้ ความสามารถดังกล่าวของฐานข้อมูลก็คือความสามารถในการสร้างวิว (View) ซึ่งฐานข้อมูลโดยทั่วไปความสามารถอันนี้อยู่แล้ว

เช่นสมมติว่ามีกฎ

P1(X,Y),P2(Y,Z) ---> P3(X,Z)

เราสามารถที่จะใช้คำสั่งภาษาฐานข้อมูลเอสคิวแอล (SQL) ทำการสร้างวิว (View) ที่มีความเท่าเทียมกับกฎข้างต้นได้ดังนี้

```
CREATE VIEW P3(X,Z) AS
    SELECT X,Z
    FROM P1,P2
    WHERE P1.Y = P2.Y ;
```

แต่ในกรณีที่คำตอบที่เราต้องการประกอบด้วยกฎหลายๆ กฎ ดังนี้

```
MaJor_In (X,'AI') <-- Take_SubJ (X,'CS345'),
MaJor_In (X,'Database') <-- Take_SubJ (X,'CS314'),
MaJor_In (X,'Statistics') <-- Take_SubJ (X,'MS312'),
                              Take_SubJ (X,'MS390')
```

ถ้าเราใช้เพรดิเคทแทนชนิดความจริงของแบบจำลองข้อมูลในแอมชนิดความจริง Major_In หมายถึงนักศึกษา X จะเรียนในสาขาหลักสาขาใดสาขาหนึ่งได้เมื่อนักศึกษา X ได้ลงเรียนใน วิชาที่กำหนด เช่น ถ้า X จะเป็นนักศึกษาในสาขาหลัก 'AI' เมื่อ X ลงเรียนในวิชา 'CS345' ซึ่งกฎทั้งสามนี้ก็จะมีความหมายว่าให้หานักศึกษาที่อยู่ในสาขาหลัก 'AI','Database' และ 'Statistics'

โดยทั่วไปคำตอบที่เราต้องการจะหาได้จากเซ็ทของกฎทั้งสาม ก็คือผลลัพธ์ของแต่ละกฎที่จะถูกนำมายูเนียน (Union) กัน ซึ่งในกรณีเช่นนี้เราไม่สามารถใช้วิวเข้าช่วยได้ ทั้งนี้เพราะในระบบฐานข้อมูลส่วนมากไม่มีความสามารถในการทำยูเนียนในวิว แต่ในปัจจุบันระบบฐานข้อมูลบางตัว เช่น ออราเคิล (Oracle) ได้สร้างความสามารถนี้ไว้แล้ว อย่างไรก็ตามในกรณีเช่นนี้เราจะใช้วิธีการอีกแบบหนึ่ง โดยการสร้างตารางที่ชื่อ Major_In ขึ้นมาใหม่ ซึ่งตารางที่สร้างขึ้นนี้จะถูกใช้งานเพียงชั่วคราวเพื่อช่วยในการรวมคำตอบที่ได้จากกฎทั้งสาม เมื่อเราสร้างตารางดังกล่าวเรียบร้อยแล้ว เราก็จะทำการเก็บข้อมูลที่ได้จากกฎทั้งสามไว้รวมกัน สามารถเขียนเป็นคำสั่งเอสคิวแอล ได้ดังนี้

```
INSERT INTO Major_In
    SELECT STUDENT,'AI'
    FROM Take_SubJ
    WHERE SUBJECT = 'CS345'
INSERT INTO Major_In
    SELECT STUDENT,'Database'
    FROM Take_SubJ
    WHERE SUBJECT = 'CS314'
INSERT INTO Major_In
    SELECT STUDENT,'Statistics'
    FROM Take_SubJ FT1,Take_SubJ FT2
    WHERE FT1.STUDENT = FT2.STUDENT AND
        FT1.SUBJECT = 'MS312' AND
        FT2.SUBJECT = 'MS390'
```

เมื่อมาถึงจุดนี้เราก็ได้ทราบแนวทางที่จัดการกับกฎ โดยที่แต่ละเพรดิเคทแทนชนิดความจริงที่กำหนดไว้ในแบบจำลองข้อมูลไนแอม อย่างไรก็ตามกฎที่สามารถจัดการโดยใช้เทคนิคข้างต้นนั้นได้ ก็จะต้องมีข้อแม้อยู่ว่าจะต้องเป็นกฎที่ไม่เป็นรีเคอร์ซีฟ ซึ่งในขั้นตอนต่อไปเราก็จะแสดงให้เห็นเพิ่มเติมว่าสำหรับกฎที่อยู่ในรูปแบบที่เป็นรีเคอร์ซีฟนั้น เรามีวิธีจัดการอย่างไร

2.4.2 กฎชนิดรีเคอร์ซีฟ

นักวิทยาศาสตร์ทางด้านฐานข้อมูล ต่างทราบกันดีถึงการที่จะประมวลผลคำถามทางลอจิก (Logic Query) โดยผ่านทางวิว (View) ซึ่งเป็นกลไกหนึ่งที่มีอยู่ในฐานข้อมูลแบบรีเลชันแนลทั่วไป แต่คำถามเหล่านั้นมีขอบเขตอยู่ว่าจะต้องไม่เป็นคำถามแบบรีเคอร์ซีฟ (Non-Recursive Query) ทั้งนี้เพราะถ้าเราทำการประมวลผลคำถามแบบรีเคอร์ซีฟในลักษณะเช่นเดียวกับคำถามที่ไม่เป็นรีเคอร์ซีฟ จะทำให้การประมวลผลเกิดวงรอบการทำงานที่เป็นอนันต์ ซึ่งแม้กระทั่งทุกวันนี้ปัญหาทางของการแก้ไขคำถามแบบรีเคอร์ซีฟก็ยังมีผู้วิจัยอยู่

เมื่อหลายปีก่อน ได้มีนักวิทยาศาสตร์กลุ่มหนึ่งซึ่งประกอบด้วย Chang , Shapiro และ McKay , Henschen และ Naqvi ได้บุกเบิกงานทางด้านการจัดการกับกฎแบบรีเคอร์ซีฟทำให้งานทางด้านนี้ได้พัฒนาไปมาก แต่งานที่ได้พัฒนาไปนั้นส่วนใหญ่แล้วจะอยู่ในรูปของเพรดิเคทลอจิก อย่างไรก็ตามอย่างที่ได้กล่าวไปแล้วว่า พื้นฐานของทฤษฎีทางรีเลชันแนลและเพรดิเคทลอจิก มีความสัมพันธ์กันอย่างใกล้ชิดภายใต้เงื่อนไขบางประการ

ก่อนที่เราจะกล่าวถึงการแก้ปัญหาของคำถามแบบรีเคอร์ซีฟ เราจะมาดูรูปแบบของคำถามประเภทนี้ว่ามีกี่แบบ และในแต่ละแบบมีลักษณะอย่างไรบ้าง

2.4.2.1 กฎชนิดลิเนียร์รีเคอร์ซีฟ (Linear Recursive Rule)

สมมติว่าเรามีเพรดิเคทต่อไปนี้

```
(1) Imm_Prereq ('Data Base','Data Structure').
(2) Imm_Prereq ('Data Structure','Programming Language').
(3) Prereq (X,Y) :- Prereq(X,Z),Imm_Prereq(Z,Y).
```

จะสังเกตเห็นได้ว่าในข้อที่ (1)-(2) จะเป็นฐานข้อมูลแบบเอกซ์เทนชันนัล (Extensional Database) ซึ่งเป็นชนิดความจริงที่จัดเก็บในฐานข้อมูล และข้อที่ (3) เป็น

การกำหนดฐานข้อมูลแบบอินเทนชันนัล (Intensional Database) ซึ่งเป็นข้อมูลที่ไม่ต้องเก็บลงในฐานข้อมูลจริง และสามารถหามาได้ในภายหลัง กฎแบบรีเคอร์ซีฟนั้นสามารถเขียนขึ้นในรูปอย่างง่ายให้อยู่ในรูปทั่วไปได้ดังนี้

$$p(t) :- \ldots, p(t'), \ldots.$$

เช่น Prereq (X,Y) :- Prereq(X,Z), Imm_Prereq(Z,Y).

ซึ่งกฎที่อยู่ในรูปแบบดังกล่าวเราจะเรียกได้อีกชื่อว่า การเรียกตนเองเชิงเส้น (Linear Recursive) ซึ่งจะมีลักษณะที่ว่าชนิดความจริงที่เป็นส่วนหัว (Head) ของกฎนั้นไปปรากฏที่ส่วนตัว (Body) ของกฎนั้นด้วย

2.4.2.2 กฎชนิดมิวชัวล์รีเคอร์ซีฟ

ต่อไปเราลองพิจารณากฎต่อไปนี้

```
p (X,Y) :- b1 (X,Z),q (Z,Y).
q (X,Y) :- p (X,Z),b2 (Z,Y).
```

จะเห็นได้ว่าไม่มีกฎใดเลยที่เป็นรีเคอร์ซีฟตามข้อกำหนดข้างต้น แต่เมื่อพิจารณาให้ดีแล้วจะพบว่าทั้งชนิดความจริง p และ q ต่างก็เป็นรีเคอร์ซีฟของซึ่งกันและกัน ถึงตรงนี้เราก็จะหาข้อกำหนดสำหรับรีเคอร์ซีฟแบบที่เกี่ยวข้องกับกฎหลายๆ กฎ

เรากล่าวว่า ถ้า p แล้ว q (p --> q) ถ้ามีชนิดความจริง p ปรากฏอยู่ในส่วนตัวของกฎที่มี q เป็นส่วนหัว และทั้ง p และ q จะถูกเรียกว่าเป็นการเรียกตนเองแบบมิวชัล (Mutually Recursive) ของซึ่งกันและกันเมื่อ p --> q และ q --> p

2.4.3 การจัดการกับกฎแบบรีเคอร์ซีฟ

จากที่ได้กล่าวมาข้างต้นว่าคำถามแบบรีเคอร์ซีฟไม่สามารถแก้ได้โดยวิธีทั่วไปได้ ดังนั้นก็น่าจะมีวิธีที่จะสามารถจัดการกับมันได้โดยเฉพาะ ซึ่งวิธีในการจัดการกฎแบบรีเคอร์ซีฟนี้ จากการศึกษาจากบทความเปรียบเทียบวิธีการจัดการกับกฎแบบรีเคอร์ซีฟ An Amateur's Introduction to Recursive Query Processing Strategies ดังต่อไปนี้

2.4.3.1 การเปรียบเทียบวิธีการจัดการกับกฎแบบรีเคอร์ซีฟ

จากการศึกษาบทความเปรียบเทียบเรื่อง An Amature's Introduction to Recursive Query Processing Strategies ซึ่งวิจัยโดย Francois Banchon และ Raghu Ramakrishnan [1986] โดยบทความนี้จะทำการเปรียบเทียบวิธีการต่างๆ เพื่อใช้ในการประมวลผลคำถามที่มีลักษณะเป็นกฎชนิดรีเคอร์ซีฟ (Recursive Query) ในระบบฐานข้อมูลแบบรีเลชันแนล (Relational Database Systems) โดยที่คำถามนั้นจะต้องมีลักษณะเป็นอนุประโยคของฮอร์น และไม่มีฟังก์ชันซิมโบล (Function Symbol)

โดยที่งานทางด้านนี้ได้เพิ่งถูกบุกเบิก เมื่อประมาณ 10 ปีที่ผ่านมานี่เอง โดยนักวิจัย เช่น Chang, Shapiro และ Mckey, Henschen และ Naqvi จนได้ผลลัพธ์เป็นอัลกอริทึมออกมามากมาย แต่ปัญหาก็คือไม่ทราบว่าจะเลือกใช้อัลกอริทึมใด ที่เหมาะสมกับงานที่ต้องการ

ในบทความเปรียบเทียบนี้จะใช้หลักการเปรียบเทียบในหลายๆ แง่มุมแต่หลักการที่เป็นหลักใหญ่ๆและเป็นหลักการที่เราสนใจได้แก่

(1) ใช้วิธีการอินเทอร์พรีต (Interpretation) หรือ คอมไพล์ (Compilation) วิธีการใดจะเรียกว่าเป็นคอมไพล์ก็ต่อเมื่อวิธีการนั้นมีการแบ่งขั้นตอนการทำงานออกเป็น 2 ขั้นตอนคือขั้นตอนสร้างคำถาม (Compile Query) และขั้นตอนนำคำถามมาทำงาน (Execute Query) ซึ่งในขบวนการสร้างคำถามนั้น จะต้องอ้างถึงแต่เฉพาะข้อมูลในส่วนที่เป็นฐานข้อมูลแบบอินเทนชันนัลเท่านั้น และจะอ้างถึงข้อมูลในส่วนที่เป็นฐานข้อมูลแบบเอกซ์เทนชันนัล เมื่อมีการนำคำถามมาทำงาน วิธีการใดที่ไม่เป็นไปตามนี้ ถือได้ว่าเป็นการอินเทอร์พรีต

(2) มีการทำงานในแบบเรียกตนเอง (Recursion) หรือ ใช้วิธีทำซ้ำ (Iteration) การทำงานในแบบรีเคอร์ชันนั้นเป็นการทำงานที่มีประสิทธิภาพกว่า เพราะการที่เป็นรีเคอร์ชันทำให้ไม่มีขอบเขตของการทำซ้ำ ขณะที่การทำซ้ำจะต้องมีขอบเขต

(3) มีการทำงานแบบบนลงล่าง (Top-Down) หรือ ล่างขึ้นบน (Bottom-Up) การทำงานทั้งสองแบบนี้ เป็นการมองว่า การหาคำตอบนี้เหมือนกับ การตรวจสอบไวยากรณ์ (Parsing) ในตัวแปลภาษา โดยที่จะมองแต่ละเพรดิเคทใน ฐานข้อมูลแบบเอกเทนชันนัล เป็นสัญญลักษณ์สิ้นสุด (Terminal Symbol) และเพรดิเคทในฐานข้อมูลแบบอินเทนชันนัลเป็นสัญญลักษณ์ไม่สิ้นสุด (Non-Terminal Symbol) และการค้นหาคำตอบก็คือการพยายามทำให้คำถามซึ่งเป็น สัญญลักษณ์ไม่สิ้นสุด กลายเป็น สัญญลักษณ์สิ้นสุด ทั้งหมด อย่างไรก็ตามในกรณีของคำถามที่เป็นรีเคอร์ชันนี้จะทำให้เกิดวงวกที่ไม่สิ้นสุด ดังนั้นก็จะต้องมีเงื่อนไขสิ้นสุด (Termination Condition) คือเมื่อ สัญญลักษณ์สิ้นสุด ที่สร้างขึ้นประเมินผลได้เช่นว่างก็ถือเป็นการสิ้นสุด

และความแตกต่างระหว่างการทำงานแบบบนลงล่าง และ การทำงานแบบล่างขึ้นบน ก็คือ การทำงานแบบบนลงล่าง จะเริ่มจากคำถาม (Non-Terminal Symbol) และการทำงานแบบล่างขึ้นบน จะเริ่มจากคำตอบ (Terminal SYmbol) ซึ่งวิธีการทำงานแบบบนลงล่าง จะมีประสิทธิภาพมากกว่า เพราะว่าเรารู้ว่าคำถามอะไรที่เราต้องการจะหาคำตอบ แต่วิธีการจะซับซ้อนกว่า ขณะที่การทำงานแบบล่างขึ้นบน จะง่ายกว่าแต่ไม่มีประสิทธิภาพเพราะจะมีผลลัพธ์จำนวนมากที่ไม่ได้ใช้งาน

ซึ่งจากบทความดังกล่าวได้ทำการเปรียบเทียบวิธีการต่างๆ จำนวน 10 วิธีดังตารางต่อไปนี้

วิธีการ	Top-Down	Compiled	Iterative
	Bottom-Up	Interpreted	Recursive
Naive Evaluation	Bottom-Up	Compiled	Iterative
Semi-Naive Evaluation	Bottom-Up	Compiled	Iterative
Query/Subquery	Top-Down	Interpreted	Iterative
Query/Subquery	Top-Down	Interpreted	Recursive
APEX	Mixed	Compiled	Recursive
Prolog	Top-Down	Interpreted	Recursive
Henschen-Naqvi	Top-Down	Compiled	Iterative
Aho-Ullman	Bottom-Up	Compiled	Iterative
Kifer-Lozinskii	Bottom-Up	Compiled	Iterative
Counting	Bottom-Up	Compiled	Iterative
Magic Sets	Bottom-Up	Compiled	Iterative

และสำหรับวิธีการที่เราเลือกใช้นั้น ที่แน่นอนคือต้องเป็นแบบการทำซ้ำ เพราะภาษาเอสคิวแอล ที่เราจะทำการเชื่อมต่อด้วยนั้น (Interface) ไม่มีความสามารถในการเรียกตัวเอง และควรจะเป็นการทำงานแบบบนลงล่าง ซึ่งเป็นวิธีการที่มีประสิทธิภาพในการหาคำถามมากกว่า ซึ่งวิธีการที่มีคุณสมบัติ 2 ประการก็มีอยู่ 2 วิธี แต่ที่เราเลือกใช้วิธีการของ Henschen-Naqvi นั้นก็เพราะใช้วิธีการคอมไพล์ ซึ่งมีการทำงานแยกกันระหว่างการสร้างคำถาม และการหาคำตอบซึ่งจะทำให้การพัฒนาเป็นไปได้ง่ายกว่า

2.4.3.2 วิธีการของ Henschen และ Naqvi

เมื่อเราเลือกใช้วิธีการของ Henchen และ Naqvi จากนั้นเราจะนำวิธีการของนักวิจัยทั้งสองท่านมาประยุกต์ใช้กับปัญหาการแทนชนิดความจริงด้วยเพรดิเคทของเรา ซึ่งวิธีการดังกล่าวนั้นจะทำการแปลงกฎชนิดรีเคอร์ซีฟให้เป็นลำดับของกฎชนิดไม่รีเคอร์ซีฟ และจากนั้นเราก็จะแทนลำดับของกฎเหล่านี้ด้วยภาษาฐานข้อมูลเอสคิวแอล เพื่อให้ชัดเจนเรามาลองพิจารณากฎ R1 และ R2 ต่อไปนี้

R1 : Prereq(X,Y) <-- Imm_Prereq(X,Y).

R2 : Prereq(X,Y) <-- Prereq(X,Z), Imm_Prereq(Z,Y).

Query : Prereq(X,Y) ?

ซึ่งความสัมพันธ์ Prereq(X,Y) หมายถึงนักศึกษาผู้ที่จะลงในวิชา X และต้องลงในราย วิชา Y อะไรมาบ้างและความสัมพันธ์ Imm_Prereq(X,Y) หมายถึงนักศึกษาที่จะลงวิชา X จะต้องลงในรายวิชา Y ในเทอมที่ผ่านมา ในการหาคำตอบของคำถามและกฎข้างต้นก็คือเซ็ทของกฎที่หามาได้จากกฎ R1 และ R2 ซึ่งกฎใหม่ที่ได้จากกฎทั้งสองนั้นจะไม่เป็นแบบรีเคอร์ซีฟ ซึ่งกฎใหม่ดังกล่าวนี้ได้แสดงไว้ข้างล่างนี้แล้ว

{ Imm_Prereq(X,Y);
Imm_Prereq(X,Y1), Imm_Prereq(Y1,Y);
Imm_Prereq(X,Y1), Imm_Prereq(Y1,Y2), Imm_Prereq(Y2,Y);
... }

กฎที่ได้แสดงไว้ข้างต้นนี้ กฎแรกได้มาจาก R1 และกฎต่อมาหามาได้จากการแทนกฎ R1 ลงในกฎ R2 และต่อจากนั้นก็จะเป็นการแทนกฎ R1 ลงในกฎที่ได้จากกฎที่ 2 และแทนลงเช่นนี้ จะเห็นได้ว่ากฎที่สร้างขึ้นใหม่นี้มีจำนวนที่ไม่จำกัด ดังนั้นเราจะต้องหาเงื่อนไขที่จะทำให้การสร้างกฎนี้มีการสิ้นสุด เรารู้ว่ากฎแต่ละกฎที่สร้างขึ้นจะได้จากจำนวนครั้งที่เราทำการแทนกฎ R1 และ R2 ซึ่งจะเทียบกับการวงรอบทำรีเคอร์ซีฟแต่ละรอบนั่นเอง และเงื่อนไขที่จะทำให้การสร้างกฎนี้สิ้นสุดก็คือ เมื่อไรก็ตามที่เราหาคำตอบที่ได้จากกฎที่เราสร้างขึ้นใหม่แล้วได้ผลลัพธ์เป็น

เซ็ทว่าง (Empty Set) ก็จะถือว่าเป็นเงื่อนไขที่สิ้นสุดการสร้างกฎใหม่ และคำตอบของกฎที่เป็นรีเคอร์ซีฟ ก็คือยูเนียนของผลลัพธ์ที่ได้จากกฎที่สร้างขึ้นก่อนที่จะถึงเงื่อนไขที่สิ้นสุดนั่นเอง

ต่อไปเราก็จะกล่าวถึงวิธีของ Henschen-Naqvi โดยละเอียด จากที่ได้กล่าวไปแล้วข้างต้น

ข้อกำหนด ฐานข้อมูลอนุมานประกอบด้วยสองส่วน โดยส่วนแรกคือ ฐานข้อมูลแบบเอกเทนชันนัล (Extensional database) ซึ่งก็คือข้อมูลที่เป็นอนุประโยคพื้นฐาน (Ground Clause) ซึ่งก็จะสอดคล้องกับข้อมูลในแต่ละทรัพเพิลที่ถูกเก็บอยู่ในรูปของความสัมพันธ์ในฐานข้อมูลและส่วนที่สองคือ ฐานข้อมูลแบบอินเทนชันนัล (Intensional Database) ซึ่งก็คือเซ็ตของ อนุประโยคของฮอร์น นั่นเอง

ข้อกำหนด เพรดดิเคทใดๆ ที่เกิดขึ้นใน ฐานข้อมูลแบบเอกเทนชันนัล จะเรียกว่าเพรดดิเคทแบบอีดีบี (EDB) และเพรดดิเคทใดๆ ที่เกิดขึ้นในฐานข้อมูลแบบอินเทนชันนัล จะเรียกว่าเพรดดิเคทไอดีบี (IDB) ซึ่งวิธีการของ Henschen-Naqvi จะแทนแต่ละรีเลชันด้วยเพรดดิเคท เช่นถ้าเรามีความสัมพันธ์ P ซึ่งมี ทรัพเพิล เป็น <a1,....,an) เราสามารถแทนด้วยเพรดดิเคทได้เป็น P(a1,....,an) และแทนอนุประโยคของฐานข้อมูลแบบอินเทนชันนัลนี้ในรูปของคอนเนคเทดกราฟ (Connected Graphs หรือ CG หรือ ซีจี) (Sickel [20])

ข้อกำหนด ซีจี คือความสัมพันธ์ (N,E,S,P) โดยที่

(1) N เป็นเซ็ตของโหนด โดยที่แต่ละ โหนดจะแทนแต่ละ ลิตเทอรัล (Literal) ที่เกิดขึ้นใน IDB

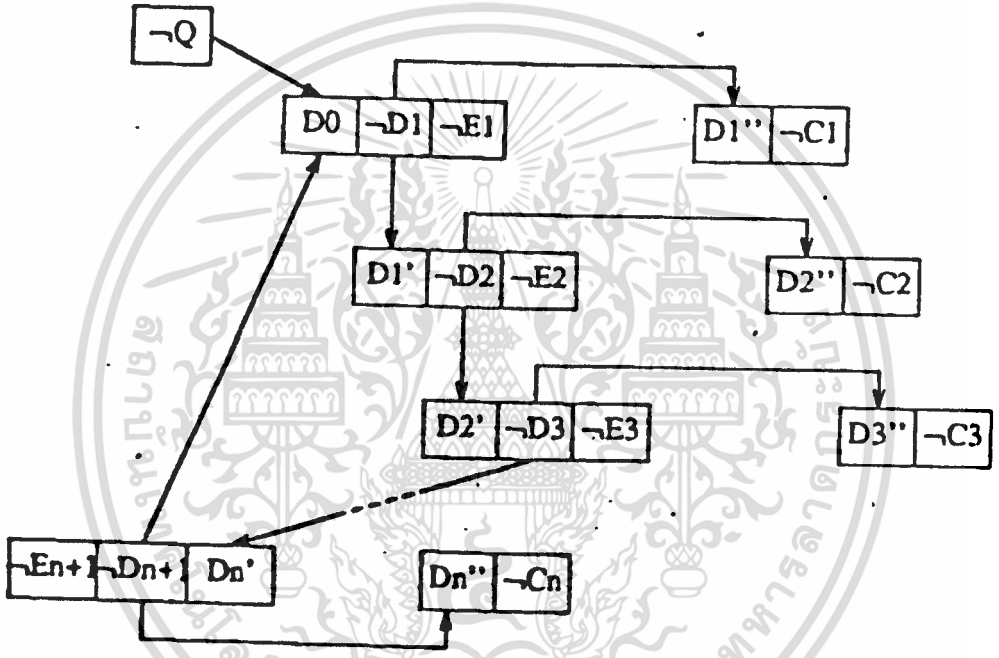
(2) E คือเซ็ตของเอดจ์ (Edge) โดยสมมติว่า e เป็น เอดจ์ ที่เชื่อมระหว่าง L1 และ ~L2 และ e จะอยู่ในเซ็ต E ก็ต่อเมื่อ L1 และ L2 สามารถรวมกันได้ (Unificable)

(3) S เป็นเซ็ตของการแทน (Substitution) โดยสมมติว่า s เป็นการแทน และ s จะอยู่ใน S ก็ต่อเมื่อ s เป็นเอ็มจียู (m.g.u. หรือ most general unifier) ของ ลิตเทอรัล (Literal) L1 และ L2 บน เอดจ์ E

(4) P เป็นเซ็ตของส่วนของ อนุประโยค โดยสมมติให้ p เป็นส่วนของ อนุประโยค และ p จะอยู่ในเซ็ต P ก็ต่อเมื่อมันมี ลิตเทอรัล (Literal) บางตัวที่อยู่ใน อนุประโยค ของฐานข้อมูลแบบอินเทนชันนัล

วิธีการของ Henschen และ Naqvi จะแทนกฎทั้งหมดด้วย ซีจี และโดยอาศัยวิธีการของ Sickel (1976) เราจะสามารถตรวจพบวงรอบขึ้นใน ซีจี นั้นในกรณีที่มี อนุประโยคที่เป็นรีเคอร์ซีฟเกิดขึ้นและเราจะเรียกวงรอบนี้ว่าการวนซ้ำแบบเรียกตนเองที่ซ่อนอยู่ภายใน (Potential Recursive Loop หรือ PRL หรือ พีอาร์แอล) ซึ่งรูปแบบโดยทั่วไปของ พีอาร์แอล แสดงได้ดังรูปที่ 2-1

ต่อไปเราก็จะกล่าวถึงข้อกำหนดต่างๆ ใน พีอาร์แอล ดังต่อไปนี้



รูปที่ 2-1 แสดงรูปแบบทั่วไปของพีอาร์แอล

<u>ข้อกำหนด</u> อนุประโยค ของ พีอาร์แอล ที่เชื่อมกับคำถามจะเรียกว่า อนุประโยคเริ่มต้น (Start Clause) ซึ่งในรูปที่ 2-1 ก็คืออนุประโยค D0' ~D1 ~E1

<u>ข้อกำหนด</u> ลิตเทอรัล (Literal) ใน อนุประโยคเริ่มต้น (Start Clause) ที่เชื่อมกับคำถามจะเรียกว่า ลิตเทอรัลเริ่มต้น (Start Literal)

ข้อกำหนด ลิดเทอรัล (Literal) ภายใน พีอาร์แอล ที่เชื่อมกับ ลิดเทอรัลเริ่มต้น (Start Literal) เรียกว่า ลิดเทอรัลสิ้นสุด (End Literal)

ข้อกำหนด อนุประโยค ที่มี ลิดเทอรัลสิ้นสุด (End Literal) อยู่ภายในเรียกว่า อนุประโยคสิ้นสุด (End Clause) ซึ่งจากตัวอย่างในรูปที่ 2-1 ก็คือ อนุประโยค Dn' ~Dn+1 ~En+1

ข้อกำหนด ลิดเทอรัล (Literal) ~D1 และ D1' ที่แสดงไว้ในรูปที่ 2-1 เรียกว่าไซเคิลลิดเทอรัล (Cycle Literals)

ข้อกำหนด โคลสซิงเอดจ์ (Closing Edge) ของ พีอาร์แอล คือ เอดจ์ ที่เชื่อมต่อกับ ลิดเทอรัลสิ้นสุด (End Literal) และ ลิดเทอรัลเริ่มต้น (Start Literal)

ข้อกำหนด เอดจ์ ที่ต่อกับไซเคิลลิดเทอรัล (Cycle Literal) และทำให้ออกจาก พีอาร์แอล ได้เรียกว่าเอกซิทเอดจ์ (Exit Edge)

ข้อกำหนด เศษเหลือที่เกิดจากการวนซ้ำสะสม (Augmented Loop Residue) หาได้จากการหาผลลัพธ์ของ อนุประโยคเริ่มต้น (Start Clause) ในภาพด้วยการแทน D1' ~D2 ~E2 ลงใน อนุประโยคเริ่มต้น (Start Clause) และหาผลลัพธ์ของ อนุประโยค ที่ได้ใหม่นี้ โดยการแทนอนุประโยค D2' ~D3 ~E3 และทำเช่นนี้จนถึง อนุประโยคสิ้นสุด

ข้อกำหนด นิพจน์สิ้นสุดการสะสม (Augmented Exit Expression) คือการหาผลลัพธ์ของอนุประโยคเริ่มต้น (Start Clause) โดยการแทน D1" ~C1 หรือโดยการแทน D1' ~D2 ~E2 ลงใน อนุประโยคเริ่มต้น (Start Clause) แล้วจึงแทนอนุประโยค D2" ~C2 ลงในผลลัพธ์ที่ได้นั้น

เพื่อให้เข้าใจยิ่งขึ้น จะขอยกตัวอย่างประกอบดังนี้ สมมติว่า M,F,P เป็น ฐานข้อมูลแบบเอกเทนชันนัล และ S,T เป็น ฐานข้อมูลแบบอินเทนชันนัล และเป็นการเรียกตนเองแบบมิวชัวล์ซึ่งกันและกัน เพรดิเคทเหล่านี้จะแทนความหมายของ Mother, Father และ Parent และ S กับ T เป็นแต่ละชนิดของ Ancestor

```
R1 : S (X1,Z1) <-- M (X1,Y1),T(Y1,Z1).
R2 : T (Y1,Z1) <-- S (Y1,W1),P(W1,Z1).
R3 : T (Y1,Z1) <-- F(Y1,Z1).
```

ซึ่งกฎดังกล่าวสามารถเขียนให้อยู่ในรูปของรูปแบบหลอลชัวล์ (Clausal Form) ได้

ดังนี้

C1 : ~M(X1,Y1), ~T(Y1,Z1),S(X1,Z1).

C2 : ~S(Y1,W1), ~P(W1,Z1),T(Y1,Z1).

C3 : ~F(Y1,Z1),T(Y1,Z1).

Query : S(X,'a').

จากกฎข้างต้นนั้นเราสามารถนำมาเขียนเป็น พีอาร์แอล ได้ดังรูปที่ 2-2

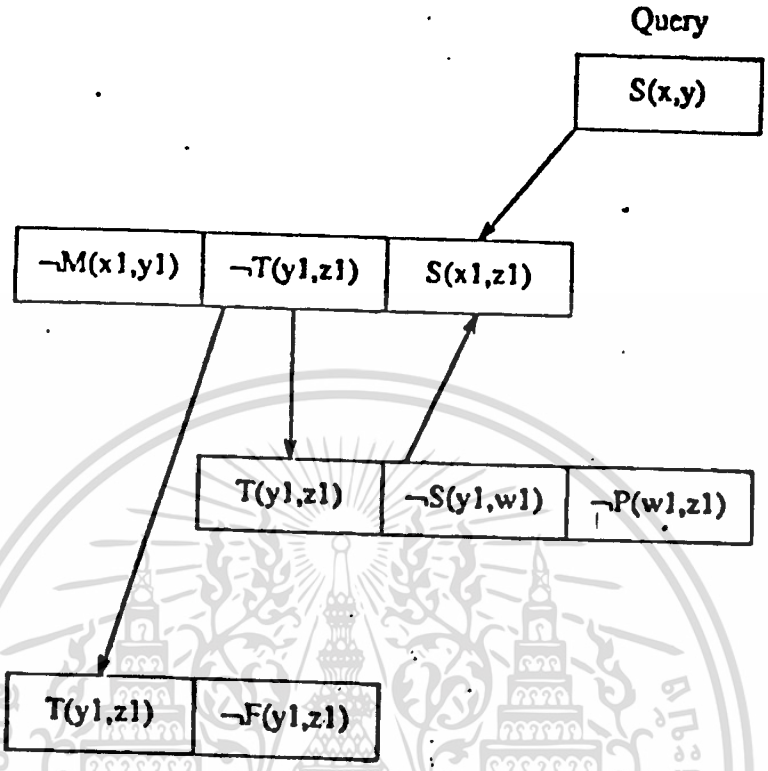
และจากข้อกำหนด อนุประโยคเริ่มต้น (Start Clause) คือ C1 และ ลิตเทอรัลเริ่มต้น คือ S(X1,Z1) และ ไซเคิลลิตเทอรัล ~T(Y1,Z1) ของ อนุประโยค C1 และ T (Y1,Z1) ของ อนุประโยค C2 สามารถรวมกันได้ (Unification) และเชื่อมกันโดย เอดจ์ e2 และเนื่องจาก S(X1,Z1) เป็น ลิตเทอรัลเริ่มต้น (Start Literal) ~S(Y1,W1) เชื่อมกับ ลิตเทอรัลสิ้นสุด (End Literal) ดังนั้น C2 ก็จะเป็น อนุประโยคสิ้นสุด โดยมี เอดจ์ e3 ออกจากไซเคิลลิตเทอรัล ~T(Y1,Z1) ไปยังเอกซิทเอดจ์

และจากที่กล่าวมาเราจะได้เศษเหลือในการรวมซ้ำสะสม ของ พีอาร์แอล ข้างต้นคือ

S(X1,Z1), ~M(X1,Y1), ~S(Y1,W1), ~P(W1,Z1).

และจะได้ นิพนธ์สิ้นสุดจากสะสม ดังนี้

S(X1,Z1), ~M(X1,Y1), ~F(Y1,Z1).



รูปที่ 2-2 ตัวอย่างของการวนซ้ำแบบเรียกตนเองที่ซ่อนอยู่ภายใน

เมื่อถึงจุดนี้จาก พีอาร์แอล เราก็จะได้ เศษเหลือในการวนซ้ำสะสม และ นิพจน์สิ้นสุดจากการสะสม และลำดับของกฎที่ไม่เป็นรีเคอร์ซีฟที่จะนำมาใช้หาคำตอบของกฎที่เป็น รีเคอร์ซีฟ ซึ่งจะสร้างได้จากการแทน นิพจน์สิ้นสุดจากการสะสม ลงใน เศษเหลือในการวนซ้ำสะสม จากนั้นก็ทำการ การประเมินผล กฎใหม่ที่เราสร้างขึ้นและถ้าผลลัพธ์จากการ การประเมินผล กฎใหม่ที่ได้นี้ยังไม่ให้ผลลัพธ์เป็นเซ็ทว่าง เราก็จะนำนิพนธ์สิ้นสุดจากการสะสม แทนลงในกฎใหม่นี้อีก และทำเช่นนี้จนกระทั่งได้ผลลัพธ์ของการ การประเมินผล เป็นเซ็ทว่างและคำตอบของกฎที่เป็นรีเคอร์ซีฟดังกล่าวก็คือนิเนียนของกฎที่เราสร้างขึ้นนั่นเอง แต่เนื่องจากภาษาเอลคิวแอล ไม่มีความ

สามารถในการยูเนียนดังที่ได้กล่าวมาก่อนหน้านี้ ดังนั้นเราก็จะใช้วิธีการเช่นเดียวกับกรณีที่มีกฎหลายๆ กฎ กล่าวคือจะใช้วิธีการสร้างตารางขึ้นมาเพื่อเก็บผลลัพธ์ จากทั้งหมดที่ได้กล่าวมาแล้วสามารถนำมาเขียนเป็นอัลกอริธึมได้ดังนี้

อัลกอริธึม

[1] สร้างพีอาร์แอลสำหรับชนิดความจริงที่อนุมานได้แบบเรียกตนเองแต่ละตัว เพรดิเคทของชนิดความจริงจะเป็นคิวรีที่บ่งบอกถึงอนุประโยคเริ่มต้น

[2] การแก้ปัญหาพีอาร์แอลจากอนุประโยคเริ่มต้นทำได้โดยการใช้วิธีของ Henschen-Naqvi ในการสร้างเคษเหลือในการวนซ้ำสะสมและนิพจน์สิ้นสุดจากการสะสม

[3] สร้างรีเลชันสำหรับกฎแบบไม่เรียกตนเอง โดยการใช้กฎอนุประโยคของยอร์น

[4] สร้างรีเลชันสำหรับนิพจน์สิ้นสุดจากการสะสมโดยการใช้กฎอนุประโยคของยอร์น

[5] ให้ MAIN เป็นเคษเหลือในการวนซ้ำสะสม

ให้ SUB เป็นนิพจน์สิ้นสุดจากการสะสม

[6] ทำซ้ำจนกระทั่งพบเงื่อนไขสิ้นสุด

[6.1] ให้ RULE := MAIN.

[6.2] แทนที่เพรดิเคทที่อนุมานได้ที่เป็นส่วนเงื่อนไขของ RULE ด้วย SUB

[6.3] สร้างรีเลชันสำหรับ RULE โดยการใช้กฎอนุประโยคของยอร์น

ถ้ารีเลชันที่ได้เป็นเซ็ทว่าง

แล้ว ให้ TERMINATE := True;

มิฉะนั้น ให้ SUB := RULE; TERMINATE := False.

[7] รีเลชันที่เป็นคำตอบคือการยูเนียนผลลัพธ์ทั้งหมดจาก [2] ถึง [6].

จากตัวอย่างที่ผ่านมา M,F และ T เป็น ฐานข้อมูลแบบเอกเทนชันนัล และ S,T เป็น ฐานข้อมูลแบบอินเทนชันนัล นั้น

จะได้ เคษเหลือในการวนซ้ำสะสม คือ

$$S(X1,Z1),\ \sim M(X1,Y1),\ \sim S(Y1,W1),\ \sim P(W1,Z1).$$

และจะได้ นิพจน์สิ้นสุดจากการสะสม ดังนี้

S(X1,Z1), ~M(X1,Y1), ~F(Y1,Z1).

ซึ่ง อนุประโยค ทั้งสองนี้สามารถเขียนในรูปของ Logic ได้ดังนี้

S(X1,Z1) <- M(X1,Y1),S(Y1,W1),P(W1,Z1)

S(X1,Z1) <- M(X1,Y1),F(Y1,Z1)

และจาก อนุประโยค ทั้งสองนั้น จะสามารถสร้างกฎขึ้นมาใหม่ตามวิธีการของ Henschen และ Naqvi ได้ดังนี้

S(X1,Z1) <- M(X1,Y1),F(Y1,Z1)

S(X1,Z1) <- M(X1,Y1),M(Y1,D1),F(Y1,Z1),P(W1,Z1)

S(X1,Z1) <- M(X1,Y1),M(Y1,D1),M(D1,D2),F(D2,D3),P(D3,W1),
P(W1,Z1)

...

และจากที่ได้กล่าวมาแล้วว่าระบบฐานข้อมูลโดยทั่วๆไป ไม่สนับสนุนความสามารถของการสร้างยูเนียนบนวิว และจากความจริงที่ว่าเราไม่อาจที่จะทราบได้ถึงจำนวนกฎที่จะถูกสร้างขึ้น ดังนั้นเราจะใช้วิธีการสร้างความสัมพันธ์ขึ้นมาเป็นตารางถาวร และจากนั้นในแต่ละครั้งของการสร้างกฎ เราก็จะทำการแปลงกฎที่สร้างขึ้นนี้ไปเป็นภาษาเอสคิวแอล เพื่อทำการอินเสิร์ทผลของแต่ละกฎลงในตารางซึ่งจากกฎข้างต้นก็จะนำมาสร้างเอสคิวแอลได้ดังต่อไปนี้

```
INSERT TO S
SELECT M.NAME1,F.NAME2
FROM M,F
WHERE M.NAME2 = F.NAME1

INSERT TO S
SELECT FT1.NAME1,P.NAME2
FROM M FT1,M FT2,F,P
WHERE FT1.NAME2 = FT2.NAME1 AND
      FT2.NAME2 = F.NAME1 AND
```

```
        F.NAME2 = P.NAME1
INSERT INTO S
SELECT FT1.NAME1,FT5.NAME2
FROM M FT1,M FT2,M FT3,F,P FT4, P FT5
WHERE FT1.NAME2 = FT2.NAME1 AND
        FT2.NAME2 = FT3.NAME1 AND
        FT3.NAME2 = F.NAME1 AND
        F.NAME2 = FT4.NAME1 AND
        FT4.NAME2 = FT5.NAME1
```

และในกรณีที่ระบบจัดการฐานข้อมูลมีความสามารถในการทำยูเนียนบนวิวแล้ว การสร้างตารางขึ้นมารองรับนั้นก็ไม่จำเป็น ซึ่งในกรณีเช่นนี้เราจะเก็บกฎที่สร้างขึ้นนั้นอยู่ในรูปของข้อกำหนดของวิวแทน

ที่ผ่านมาเราได้แสดงการจัดการกับกฎแบบรีเคอร์ซีพซึ่งเป็นแบบเรียกตนเองแบบมิวชัวล์ต่อไปเราจะแสดงการจัดการกับกฎแบบเรียกตนเองเชิงเส้น โดยใช้ตัวอย่างเรื่องของวิชาบังคับก่อนการลงวิชาต่างๆ ที่ได้กล่าวมาแล้วครั้งหนึ่ง ซึ่งมีดังต่อไปนี้

```
R1 : Prereq(X,Y) <-- Imm_Prereq(X,Y).
R2 : Prereq(X,Y) <-- Prereq(X,Z),Imm_Prereq(Z,Y).
Query  : Prereq('Database',Y) ?
```

และสมมติว่าข้อมูลมีดังต่อไปนี้

```
Imm_Prereq('Database','Data Structure')
Imm_Prereq('Data Structure','Programming Language')
```

โดยที่ Prereq(X,Y) เป็นกฎชนิดรีเคอร์ซีพที่หมายถึงว่า ถ้าจะลงเรียนในวิชา X จะต้องลงทะเบียนเรียนในรายวิชา Y อะไรมาบ้างและ Imm_Prereq(X,Y) เป็นชนิดความจริงที่มีความหมายว่าการลงทะเบียนเรียนในวิชา X จะต้องผ่านในวิชา Y มาก่อน ในกรณีเช่นนี้ทั้งอนุประโยคเริ่มต้น (Start Clause) และ อนุประโยคสิ้นสุด จะเป็น อนุประโยค เดียวกัน

คือ R2 และจากการสร้างพีอาร์แอล เราจะสามารถหา เศษเหลือในการวนซ้ำสะสม ได้เป็น

Prereq(X,Y), ~Imm_Prereq(X,Z), ~Prereq(Z,Y)

และจะได้ นิพจน์สิ้นสุดจากการสะสม เป็น

Prereq(X,Y), ~Imm_Prereq(X,Z), ~Imm_Prereq(Z,Y)

คำตอบแรกจะได้จากการแทน นิพจน์สิ้นสุดจากการสะสม ลงในเศษเหลือในการวนซ้ำสะสม และคำตอบต่อไปจะได้จากการแทน นิพจน์สิ้นสุดจากการสะสม ลงในกฎที่ได้จากขั้นตอนที่ผ่านมาจากนั้นก็ทำการ การประเมินผล ผลลัพธ์ที่ได้จนกว่าผลลัพธ์ที่ การประเมินผล จะเป็นเซ็ทว่าง ซึ่งกฎที่สร้างขึ้นนี้จะมีดังต่อไปนี้

```
Prereq(X,Y) <-- Imm_Prereq(X,Y)
Prereq(X,Y) <-- Imm_Prereq(X,Z),Imm_Prereq(Z,Y)
Prereq(X,Y) <-- Imm_Prereq(X,Z),Imm_Prereq(Z,D1),Imm_Prereq(D1,Y)
...
```

และเช่นเดียวกับตัวอย่างที่ผ่านมา โดยการสร้างเป็นภาษาเอสคิวแอล เพื่อหาคำตอบจากกฎข้างต้นซึ่งจะไม่แสดงตัวอย่างในกรณีนี้ และเมื่อทำการ การประเมินผล กฎแรกจะได้คำตอบเป็น 'Data Structure' และกฎที่สองได้คำตอบเป็น 'Programming Language' และผลลัพธ์จากกฎที่สามจะได้เป็นเซ็ทว่างซึ่งหมายถึงการสิ้นสุดการทำงาน ดังนั้นสำหรับคำถามข้างต้นที่ว่านักศึกษาที่จะลงเรียนในวิชา 'Database' จะต้องลงเรียนในวิชาอะไรมาบ้าง ซึ่งคำตอบก็คือ 'Data Structure' และ 'Programming Language'

2.4.3.3 <u>การพัฒนาโปรแกรมการจัดการกฎแบบเรียกตนเองบนฐานความรู้</u>

โปรแกรมที่เขียนขึ้นในการจัดการกฎแบบเรียกตนเองประกอบด้วยส่วนต่างๆ ดังต่อไปนี้คือ

1. รับกฎเป็นอินพุทเข้ามาทำการสร้างซีจี
2. จากซีจีที่ได้ นำมาตรวจสอบหาพีอาร์แอล
3. จากพีอาร์แอล นำมาสร้างเศษเหลือในการวนซ้ำสะสม และ นิพจน์สิ้นสุดจาก

การสะสม

4. นำเศษเหลือในการวนซ้ำสะสม และ นิพจน์สิ้นสุดจากการสะสม มาทำการสร้างกฎแบบไม่เรียกตนเอง ตามวิธีของ Henschen-Naqvi
5. จากกฎแบบไม่เรียกตนเองที่ได้นำมาทำการสร้างคิวรีภาษาเอสคิวแอล เพื่อนำไปทำการดึงข้อมูลในฐานข้อมูลแบบสัมพันธ์ แล้วเก็บลงในตารางที่สร้างขึ้น
6. ทำซ้ำข้อ 4,5 จนกว่าไม่สามารถดึงข้อมูลออกมาได้อีก

2.4.3.3.1 <u>โครงสร้างข้อมูลในการพัฒนา</u>

หัวข้อนี้จะอธิบายถึงโครงสร้างข้อมูลของอินพุท และยังเป็น โครงสร้างข้อมูลหลักที่ใช้ตลอดการทำงานของโปรแกรม ซึ่งอยู่ในรูปแบบของภาษาซี โดยมีโครงสร้างดังนี้

```
typedef struct PREDICATE {
                    char p_name[15];
                    char parameter[10][20];
                    unsigned char param_flag[10];
                    unsigned char n_edge_flag;
                    struct  {
                        struct PREDICATE *rule;
                        unsigned char flag;
                          }s_rule[10];
                    struct PREDICATE *n_pred;
                    };
```

รายละเอียดของแต่ละฟิลด์ มีดังนี้

char p_name[15] : เก็บชื่อของเพรดิเคท ซึ่งมีความยาวสูงสุด 14 ตัว

char parameter[10][20] : เก็บอาร์กิวเมนต์ของเพรดิเคท ซึ่งมีความยาวสูงสุด 19 ตัว และ จำนวนอาร์กิวเมนต์สูงสุด 10 อาร์กิวเมนต์

unsigned char param_flag[10] : แฟลกแสดงสถานะของอาร์กิวเมนต์

unsigned char n_edge_flag : แฟลกแสดงสถานะของเพรดิเคท

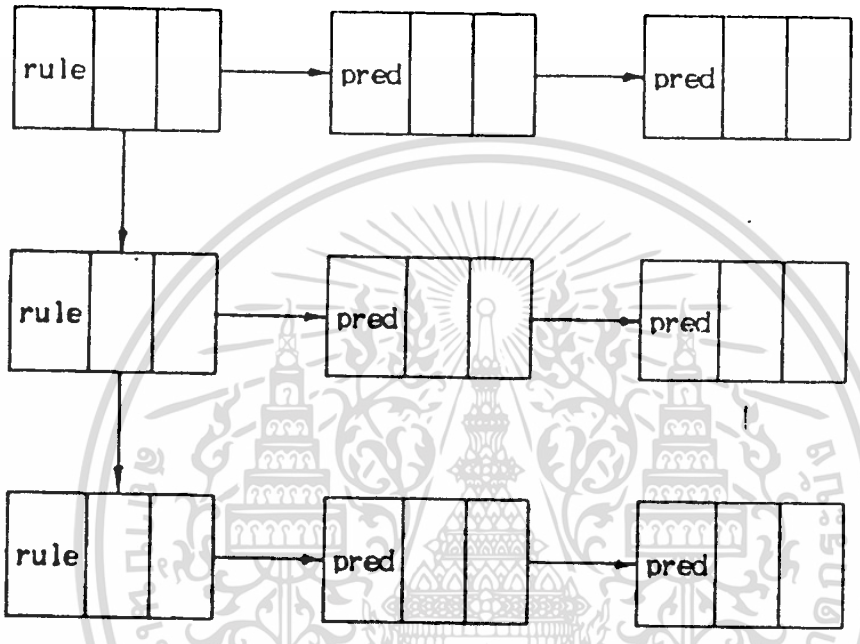
```
struct  {
        struct PREDICATE *rule;
        unsigned char flag;
```

)s_rule[10]

: พอยเตอร์ชี้ไปยังกฎซึ่งมีได้สูงสุด 10 กฎ และ แฟลกแสดงสถานะของเพรดดิเคทนั้น

struct PREDICATE *n_pred : พอยเตอร์ชี้ไปยังกฎถัดไป

จากรายละเอียดของโครงสร้างข้อมูลนี้จะ ได้รูปแบบของอินพุท ที่รับมาเป็นดังนี้



รูปที่ 2-3 แสดงลักษณะของอินพุท

จากอินพุท ที่รับเข้ามานี้จะทำการสร้างซีจีโดยมีอัลกอริธึม ดังนี้

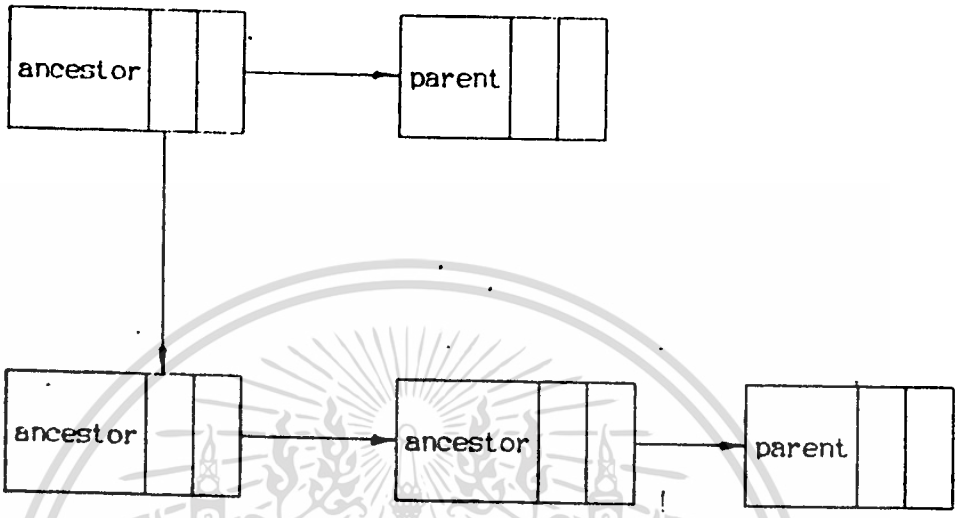
1. นำกฎที่อยู่ในลิสท์มาทีละกฎ
2. นำชื่อเพรดดิเคทที่เป็นสมาชิกของแต่ละกฎมาทีละ เพรดดิเคท
3. นำชื่อเพรดดิเคทที่อ่านมาไปทำการเปรียบเทียบกับชื่อของกฎทุกกฎที่มีอยู่ในลิสท์ ถ้ามีชื่อเหมือนกัน ให้พอยเตอร์ชี้ไปยังกฎนั้น
4. เมื่อตรวจสอบครบหมดทุกกฎให้กลับไปทำข้อ 2 ใหม่จนกว่าจะหมด
5. เปลี่ยนเป็นกฎถัดไป แล้วกลับไปทำข้อ 1 ใหม่จนกว่าจะหมด

ตัวอย่าง จากอัลกอริธึมข้างต้นจะสามารถสร้างซีจีของ

ancestor(X,Y) <- parent(X,Y).

ancestor(X,Y) <- ancestor(X,Z),parent(Z,Y).

ได้เป็น



รูปที่ 2-4 แสดงอินพุทจากตัวอย่าง

2.4.3.3.2 <u>การแปลงกฎแบบไม่เวียกตนเองเป็นเอสคิวแอลคิวรี</u>

ในการนำกฎแบบไม่เวียกตนเองมาทำการสร้างเอสคิวแอลคิวรี เพื่อนำไปทำการดึงข้อมูลในฐานข้อมูลแบบสัมพันธ์ แล้วเก็บลงในตารางที่สร้างขึ้น

โปรแกรมในส่วนนี้ทำการรับอินพุทมาในรูปแบบของกฎแบบไม่เวียกตนเอง ซึ่งจากอินพุทนี้นำมาทำการตามอัลกอริธึม ดังต่อไปนี้

1. ทำการสร้างตาราง ขึ้นมาโดยใช้ชื่อตาราง ตามชื่อของกฎ ส่วนแอตทริบิวต์ และชนิด จะใช้ตามเพรดิเคทที่อยู่ในกฎนั้น

2. สร้างอนุประโยค Select ในการดึงข้อมูล โดยการพิจารณาอาร์กิวเมนต์ที่อยู่ในกฎว่าตัวแปรนั้นตรงกับ อาร์กิวเมนต์ใดในเพรดิเคท เช่น

ancester(X,Y) <- parent(X,Z),parent(Z,Y).

จะเห็นว่าอาร์กิวเมนต์ตัวแรกของ ancestor คือ X ตรงกับอาร์กิวเมนต์ตัวแรกของเพรดิเคท parent(X,Z) และ Y จะตรงกับ ตัวที่ 2 ของ parent(Z,Y)

ดังนั้นจะสร้างได้เป็น

SELECT T1.NAME1,T2.NAME2

3. สร้างอนุประโยค from โดยพิจารณาจากเพรดดิเคทที่อยู่ในกฎซึ่ง ชื่อตารางจะใช้ตามชื่อเพรดดิเคทและทำการตั้งชื่อใหม่ให้กับตาราง นั้นด้วยเพราะในกฎ อาจจะมีการอ้างถึงกฎนั้นหลายครั้งได้ กฎจากตัวอย่างในข้อ 2 จะนำมาเขียนได้เป็น

FROM PARENT T1,PARENT T2

ซึ่งชื่อตารางที่ตั้งขึ้นมาใหม่นี้นำไปใช้ในการสร้างอนุประโยค select ด้วยดังจะเห็นได้จากตัวอย่าง

4. สร้างอนุประโยค where จะประกอบด้วย 2 ส่วนด้วยกันคือ ส่วนของค่าคงที่ที่คิวรีถามมา และส่วนของความสัมพันธ์ระหว่าง อาร์กิวเมนต์ ที่อยู่ใน เพรดดิเคท

ในกรณีของคิวรีจะดูว่า ค่าคงที่ที่ส่งมาตรงกับตัวแปรตัวไหนในกฎก็จะนำอาร์กิวเมนต์ ตัวนั้นไปเปรียบเทียบกับ อาร์กิวเมนต์ในเพรดดิเคท เมื่อได้ค่าที่ตรงกันก็จะนำตาราง และ แอตทริบิวต์นั้นมาทำการสร้างเป็นเงื่อนไข โดยนำมาเปรียบเทียบกับ ค่าคงที่ที่ส่งมาเช่น ถ้ามีคิวรีดังต่อไปนี้

```
ancester(X,Y) <- parent(X,Z),parent(Z,Y).
ancester(X,John).
```

จะได้เป็น

```
WHERE  T2.NAME2 = 'John'
```

จะเห็นว่าค่าคงที่ 'John ' ตรงกับตัวแปร Y และตรงกับแอตทริบิวต์ NAME2 ของ ตาราง T2

ส่วนที่ 2 คือส่วนของการพิจารณาความสัมพันธ์ระหว่างอาร์กิวเมนต์ของเพรดดิเคท แต่ละตัวโดยทำการหาดูว่าอาร์กิวเมนต์ ในแต่ละเพรดดิเคท เหมือนกับอาร์กิวเมนต์ใน เพรดดิเคทอื่นหรือไม่ ถ้าพบก็นำมาสร้างเงื่อนไข จากตัวอย่างเดียวกันจะเห็นว่าอาร์กิวเมนต์ Z ของ ตาราง T1 กับ T2 เป็นตัวเดียวกันเมื่อพบแล้วจะสร้างได้เป็น

```
WHERE T2.NAME2 = 'John' AND T1.NAME2 = T2.NAME1
```

5. เมื่อได้แต่ละส่วนแล้วจะนำมารวมกันและเพิ่มคำสั่ง Insert เข้าไปจะได้เป็น

```
INSERT INTO ANCESTOR
SELECT T1.NAME1,T2.NAME2
FROM PARENT T1,PARENT T2
WHERE T2.NAME2 = 'John' AND
      T1.NAME2 = T2.NAME1
```

6. จากคิวรี ที่สร้างขึ้นมาเมื่อนำไปทำงาน แล้วตรวจสอบว่ามีการ Insert ลงไปหรือไม่ เพื่อที่จะเป็นการตรวจสอบว่าเป็นการสิ้นสุด ของกฎแบบเรียกตนเอง หรือไม่

บทที่ 3

ภาษาโปรลอกไลค์

3.1 บทนำ

ภาษาโปรลอกไลค์เป็นภาษาคอมพิวเตอร์ที่เราสร้างขึ้นเพื่อใช้สำหรับการแก้ปัญหาซึ่งเกี่ยวข้องกับวัตถุ (Object) และ ความสัมพันธ์ระหว่างวัตถุ (Relation) ในหัวข้อนี้จะกล่าวถึงหลักการที่สำคัญของภาษาโปรลอกไลค์ พร้อมทั้งตัวอย่าง ให้เกิดความเข้าใจในตัวภาษานี้ดีขึ้น

หลักการขั้นต้นของภาษาโปรลอกไลค์ คือ การทำความเข้าใจกับคำว่า ข้อเท็จจริง (Facts), กฎ (Rules) และ ตัวแปร (Variables) ส่วนใหญ่โปรแกรมโปรลอกไลค์ประกอบไปด้วยการรวบรวมฐานความรู้เกี่ยวกับเรื่องใดเรื่องหนึ่งโดยเฉพาะ ฐานความรู้นี้จะถูกเรียบเรียงในรูปแบบของข้อเท็จจริง และ กฎ สมมุติว่าเราต้องการจะบอกโปรลอกไลค์เกี่ยวกับข้อเท็จจริงว่า เมธีชอบอรัญญา เราจะสามารถเขียนเป็นรูปแบบมาตรฐานได้ดังนี้คือ

```
likes(mathee,arunya).
```

ในที่นี้ทั้งเมธีและอรัญญาเรียกว่า วัตถุ ส่วน likes เรียกว่าความสัมพันธ์ นั่นคือ โปรลอกไลค์จะบอกถึงความสัมพันธ์ระหว่างวัตถุคือ เมธี และ อรัญญาว่า เมธีชอบอรัญญา สิ่งสำคัญที่เป็นข้อสังเกตคือ

1. ทั้งชื่อของวัตถุและความสัมพันธ์จะต้องขึ้นต้นด้วยอักษรตัวเล็ก
2. ในการเขียนประโยค ให้เขียนความสัมพันธ์ก่อน ส่วนวัตถุให้เขียนอยู่ในวงเล็บและแยกจากกันด้วยเครื่องหมายจุลภาค (,)
3. เมื่อสิ้นสุดประโยคของข้อเท็จจริงจะต้องมีเครื่องหมายมหัพภาค (.) เสมอ

ในขณะที่เราให้คำนิยามแสดงความสัมพันธ์ระหว่างวัตถุที่อยู่ภายในวงเล็บ เราจะต้องให้ความสำคัญของลำดับการเอ่ยถึงวัตถุ เช่น

```
likes(mathee,arunya).
```

หมายถึง เมธีชอบอรัญญา

```
likes(arunya,mathee).
```

หมายถึง อรัญญาชอบเมธี

ซึ่งความหมายของทั้งสองประโยคข้างบนนี้จะไม่เหมือนกัน ให้เราลองมาพิจารณาการเขียนข้อเท็จจริงในภาษาโปรลอกไลค์อีกดังนี้

valuable(diamond).

หมายถึง เพชรเป็นของมีค่า

owns(mathee,diamond).

หมายถึง เมธีเป็นเจ้าของเพชร

female(arunya).

หมายถึง อรัญญาเป็นผู้หญิง

male(mathee).

หมายถึง เมธีเป็นผู้ชาย

father(mathee,suvit).

หมายถึง เมธีเป็นพ่อของสุวิทย์

ซึ่งเราจะเห็นได้ว่า การดีความหมายของประโยคขึ้นอยู่กับการให้ความหมายของผู้เขียนโปรแกรมเอง ดังจะเห็นได้ว่า เราอาจดีความในประโยคสุดท้ายว่า พ่อของเมธีคือสุวิทย์ ก็ได้ ฉะนั้นในการเขียนแต่ละประโยคในโปรแกรม ผู้เขียนจะต้องให้ความหมายอย่างสมนัยกันตลอดโปรแกรม คำบรรยายทั้งหมดซึ่งไม่รวมถึงเครื่องหมายมหัพภาคมีชื่อเรียกว่า เพรดดิเคท (Predicate) วัตถุที่อยู่ในวงเล็บมีชื่อเรียกอีกอย่างหนึ่งว่า อาร์กิวเมนต์ (Argument) และความสัมพันธ์ที่อยู่หน้าวงเล็บก็มีชื่อเรียกอีกอย่างหนึ่งว่า ฟังก์เตอร์ (Functor) ในแต่ละประโยคเราจะให้มีวัตถุจำนวนเท่าใดก็ได้ เช่น

watch(mathee,arunya,movies).

หมายถึง เมธีและอรัญญาดูหนัง

play(mathee,thanit,tennis).

หมายถึง เมธีและธนิตเล่นเทนนิส

ประโยคทุกประโยคที่แสดงเป็นตัวอย่างมานั้น เราเรียกว่า ข้อเท็จจริง เมื่อเรามีข้อเท็จจริงเก็บสะสมอยู่ในโปรแกรมแล้ว ก็เท่ากับว่าเราได้จัดให้โปรแกรมมีความรู้ระดับหนึ่งเช่นกัน ฉะนั้น เราจึงสามารถที่จะสอบถามความรู้นี้จากโปรแกรมได้ การสอบถามหรือตั้งคำถามในภาษาโปรลอกไซค์ ก็คือการตั้งเป้าประสงค์ (Goal) นั่นเอง สมมุติว่าเรามีข้อเท็จจริงในโปรแกรมดังกล่าวมาแล้ว เมื่อเราตั้งคำถามเราจะได้ผลลัพธ์ดังต่อไปนี้

คำถาม likes(mathee,Who).

คำตอบ Who = arunya

ให้สังเกตว่าคำถามข้างบนนี้จะมีคำที่อยู่ในวงเล็บซึ่งใช้อักษรตัวใหญ่นำหน้าคือ Who และโปรลอกไซค์จะทำการค้นหาคำตอบให้เราโดยการเทียบกับประโยคหรือฐานข้อมูลที่มีอยู่ในโปร

แกรม. คำที่ขึ้นต้นด้วยตัวอักษรตัวใหญ่สามารถใช้แทนคำใดๆ ก็ได้ที่สามารถจะเข้าคู่กับคำถามได้พอดี และ เราเรียกว่า ตัวแปร (Variable) การมีตัวแปรที่สามารถแทนคำอื่นๆ ได้เช่นนี้ ทำให้โปรลอกไลค์มีความยืดหยุ่นตัวในการใช้งานได้ดีขึ้น

รูปแบบทั่วไปของกฎคือ ผลสรุปจะขึ้นมาอยู่ก่อน แล้วตามด้วยเครื่องหมาย ':-' ต่อจากนั้นจึงเป็นรายการเงื่อนไขที่ผลสรุปนี้ต้องใช้อ้างอิง โดยที่แต่ละเงื่อนไขต้องถูกเชื่อมด้วยเครื่องหมายจุลภาค (,) และ เมื่อหมดเงื่อนไขแล้วให้จบด้วยเครื่องหมายมหัพภาพ (.) ส่วนสรุปนี้เราเรียกได้ว่าเป็นเป้าประสงค์หลัก (Goal) และ เงื่อนไขแต่ละเงื่อนไขก็ถือได้ว่าเป็นเป้าประสงค์รอง (Sub Goal) เป้าประสงค์หลักมีอีกชื่อเรียกว่า ส่วนหัว (Head) ส่วนเงื่อนไขมีชื่อเรียกว่า เหตุผลทางตรรกวิทยา (Premise) เงื่อนไขทุกเงื่อนไขรวมกันมีชื่อเรียกว่า ส่วนทาง (Antecedent) ดังนั้นเป้าประสงค์หลักจะเป็นจริงได้ก็ต่อเมื่อเงื่อนไขหรือเป้าประสงค์รองทุกส่วนเป็นจริง แต่ถ้าเป้าประสงค์รองใดไม่เป็นจริงแล้ว เป้าประสงค์หลักก็ไม่เป็นจริงด้วย

ตัวอย่างของกฎเป็นดังนี้

```
career(air_hostess) :-
        qualification(good_health),
        qualification(self_confidence),
        qualification(good_appearance).
```

หมายความว่า ถ้ามีคุณลักษณะสุขภาพดี, เชื่อมั่นในตนเอง, บุคลิกภาพดี แสดงว่า มีอาชีพพนักงานต้อนรับบนเครื่องบิน

เราอาจสรุปหลักการเบื้องต้นของภาษาโปรลอกไลค์ได้ว่าการเขียนโปรแกรมจะต้องประกอบไปด้วย

1. การบอกกล่าวถึงข้อเท็จจริงเกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุ
2. การกำหนดกฎที่เกี่ยวข้องกับวัตถุและความสัมพันธ์ระหว่างวัตถุ
3. การตั้งเป้าประสงค์เกี่ยวกับวัตถุและความสัมพันธ์ระหว่างวัตถุ

3.2 รูปแบบของภาษาโปรลอกไลค์

ในการเขียนโปรแกรมภาษาโปรลอกไลค์จะประกอบด้วยส่วนต่างๆ เรียงตามลำดับกันดังนี้คือ

1. ส่วนโดเมน (Domains Section) เป็นส่วนที่นักเขียนโปรแกรมใช้ในการกำหนดชนิดข้อมูลของตนเองในชื่อใหม่ เพื่อประโยชน์ในการใช้โปรแกรมภายหลัง

2. ส่วนแฟกเพรดดิเคท (Fact_predicates Section) เป็นส่วนที่จะบอกตัวแปลภาษาว่าชื่อของ ข้อเท็จจริง หรือ กฎต่างๆที่ประกาศไว้ในส่วนนี้อยู่ในฐานข้อมูลแบบอนุมานกฎได้

3. ส่วนเพรดดิเคท (Predicates Section) เป็นส่วนที่จะบอกตัวแปลภาษาว่าชื่อของ ข้อเท็จจริง หรือ กฎต่างๆที่ประกาศไว้ในส่วนนี้อยู่ในหน่วยความจำ

4. ส่วนอนุประโยค (Clauses Section) เป็นส่วนที่กำหนดรายละเอียดของข้อเท็จจริง หรือ กฎต่างๆ ที่จะถูกเก็บไว้ในหน่วยความจำ

5. ส่วนเป้าประสงค์ (Goal Section) เป็นจุดมุ่งหมายของโปรแกรม หรือ อาจกล่าวได้ว่า เป็นโปรแกรมหลัก (Main Program)

แต่อย่างไรก็ตามโปรแกรมต่างๆไม่จำเป็นต้องประกอบด้วยทั้ง 5 ส่วนแต่อย่างน้อยต้องประกอบด้วย 3 ส่วนคือ

- ส่วนเพรดดิเคท
- ส่วนอนุประโยค
- ส่วนเป้าประสงค์

รูปแบบของตัวภาษาโปรลอกไลค์ที่สมบูรณ์แบบเป็นดังนี้

```
domains
    โดเมน1, โดเมน2, ... , โดเมนN = ชนิดข้อมูลมาตรฐาน
                .
                .
                .
fact_predicates
    ชื่อข้อเท็จจริง หรือ กฎ ในฐานข้อมูล 1
                .
                .
                .
predicates
    ชื่อข้อเท็จจริง หรือ กฎในหน่วยความจำ(ลิสท์ของโดเมน)
                .
                .
                .
```

clauses

ชื่อกฎในหน่วยความจำ(ลิสท์ของอาร์กิวเมนต์) :-

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูล1(ลิสท์ของอาร์กิวเมนต์),

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูล2(ลิสท์ของอาร์กิวเมนต์),

.

.

.

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูลN(ลิสท์ของอาร์กิวเมนต์).

หรือ

ชื่อข้อเท็จจริงในหน่วยความจำ(ลิสท์ของอาร์กิวเมนต์).

goal

ชื่อกฎในหน่วยความจำ(ลิสท์ของอาร์กิวเมนต์) :-

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูล1(ลิสท์ของอาร์กิวเมนต์),

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูล2(ลิสท์ของอาร์กิวเมนต์),

.

.

.

ชื่อข้อเท็จจริง หรือ กฎ ในหน่วยความจำ หรือ ในฐานข้อมูลN(ลิสท์ของอาร์กิวเมนต์).

โดย

โดเมน ต้องเป็นตัวอักษรภาษาอังกฤษตัวเล็กทั้งหมด

โดเมนมาตรฐาน มี 6 ชนิดคือ สายวลี (string),สัญญลักษณ์ (symbol), เลขจำนวนเต็ม (integer) , เลขจำนวนจริง (real) ,ตัวอักษร (char)

ข้อเท็จจริงหรือกฎในฐานข้อมูล หรือ ในหน่วยความจำก็ตาม ต้องเป็นตัวอักษรภาษาอังกฤษตัวเล็กทั้งหมด

ลิสท์ของโดเมน คือ ชุดของโดเมน หรือ โดเมนมาตรฐานที่คั่นด้วยเครื่องหมายจุลภาค (,)

ลิสท์ของอาร์กิวเมนต์ คือ ชุดของ ค่าคงที่ หรือ ตัวแปร ที่คั่นด้วยเครื่องหมายจุลภาค (,)

ตัวแปรต้องขึ้นต้นด้วยตัวอักษรตัวใหญ่และเป็นตัวอักษรล้วน

ค่าคงที่มี 5 ชนิด คือ

-สายอักขระ (String) มีรูปแบบดังนี้

" ตัวอักษรจำนวนมากกว่าหรือเท่ากับ 0 "

เช่น "Varunya", "Chocolate", "radio", " 70 New Road, Bangkok 10500" เป็นต้น

-สัญญลักษณ์ (Symbol) เป็นตัวอักษรภาษาอังกฤษตัวเล็กทั้งหมด

เช่น varunya, music เป็นต้น

-เลขจำนวนเต็ม (Integer) เป็นตัวเลขที่ปราศจากจุดทศนิยมมีค่าได้ระหว่าง -32,768 ถึง 32,767

-เลขจำนวนจริง (Real) เป็นตัวเลขที่มีเลขทศนิยมมีค่าได้ระหว่าง $1*10_{-307}$ ถึง $1*10_{308}$

-ตัวอักษร (Char) มีรูปแบบดังนี้

' ตัวอักษร 1 ตัวอักษร '

เช่น 'y', 'n', 't', 'f' เป็นต้น

ตัวแปลภาษาโปรลอกไลค์ที่สร้างขึ้นนี้จะมีชื่อจำนวนหนึ่งซึ่งมีความหมายเฉพาะของตนเองไม่สามารถนำมาใช้เป็นวัตถุในโปรแกรมที่เราเขียนเป็นอันขาด รายชื่อดังกล่าวจะถูกแสดงในรูปที่ 3-1

add	char	clauses	div
domains	equal	fail	fact_predicates
goal	integer	less	less_equal
more	more_equal	mul	nl
predicates	readl	real	string
sub	symbol	write	!

รูปที่ 3-1 แสดงรายชื่อเฉพาะที่ไม่สามารถนำไปใช้เป็นวัตถุได้

ตัวอย่างโปรแกรมภาษาโปรลอกไลค์อย่างง่าย เป็นดังนี้

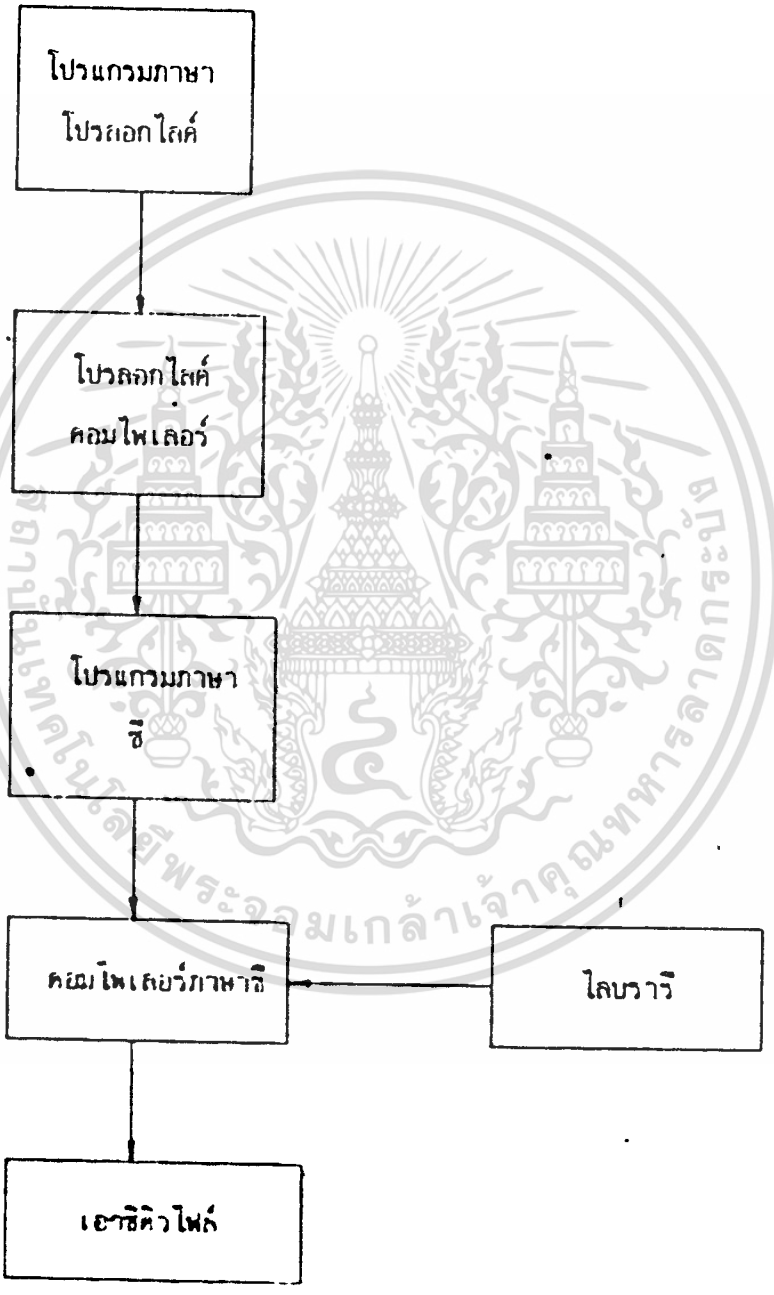
```
domains
        profession,personability = symbol
predicates
        career(profession,personality)
        main(profession,personality)
clauses
        career(air_hostess,good_health).
        career(air_hostess,self_confidence).
        career(air_hostess,good_appearance).
        career(engineer,good_at_number).
        career(engineer,self_confidence).
        career(doctor,good_health).
        career(doctor,determination).
        career(doctor,devotion).
goal
        main(A,B) :- career(A,B),fail.
```

โปรแกรมข้างต้นเป็นโปรแกรมเก็บอาชีพและคุณลักษณะของอาชีพนั้น โดยมีเป้าประสงค์คือการหาอาชีพและคุณลักษณะของอาชีพทั้งหมดของทุกอาชีพที่มีในหน่วยความจำ

เมื่อนำโปรแกรมนี้ไปผ่านตัวแปรภาษาแล้วสั่งให้ทำงานจะได้ผลลัพธ์ดังนี้

```
A= air_hostess , B= good_health
A= air_hostess , B= self_confidence
A= air_hostess , B= good_appearance
A= engineer , B= good_at_number
A= engineer , B= self_confidence
A= doctor , B= good_health
A= doctor , B= determination
A= doctor , B= devotion
```

เนื่องจากคุณสมบัติของภาษาโปรลอกไลค์ที่จัดเป็นภาษาในยุคที่ 5 ดังนั้นผลลัพท์ที่ได้จากการแปลภาษาจึงมีลักษณะที่แตกต่างจากภาษาในยุคที่ต่ำกว่า เช่นภาษาซี โดยผลลัพท์ที่ได้จะมีโครงสร้างที่สามารถทำงานได้ด้วยตัวเอง อัลกอริทึม ต่างๆ โปรแกรมเมอร์จะเป็นผู้ที่จัดการเองทั้งหมด ต่างกับภาษาโปรลอกไลค์ ที่จะต้องถูกจัดการด้วย อัลกอริทึม ที่สร้างขึ้นมาแยกต่างหาก โดยได้ออกแบบให้เป็นไลบรารีเพื่อนำมาจัดการกับผลลัพท์ที่คอมไพล์ได้ สามารถเขียนเป็นบลอกไดอาแกรมแสดงโครงสร้างการทำงานได้ดังรูปที่ 3-2



รูปที่ 3-2 แสดงกระบวนการในการคอมไพล์ภาษาโปรลอกไลค์

3.3 การพัฒนาตัวแปลภาษาโปรลอกไลค์

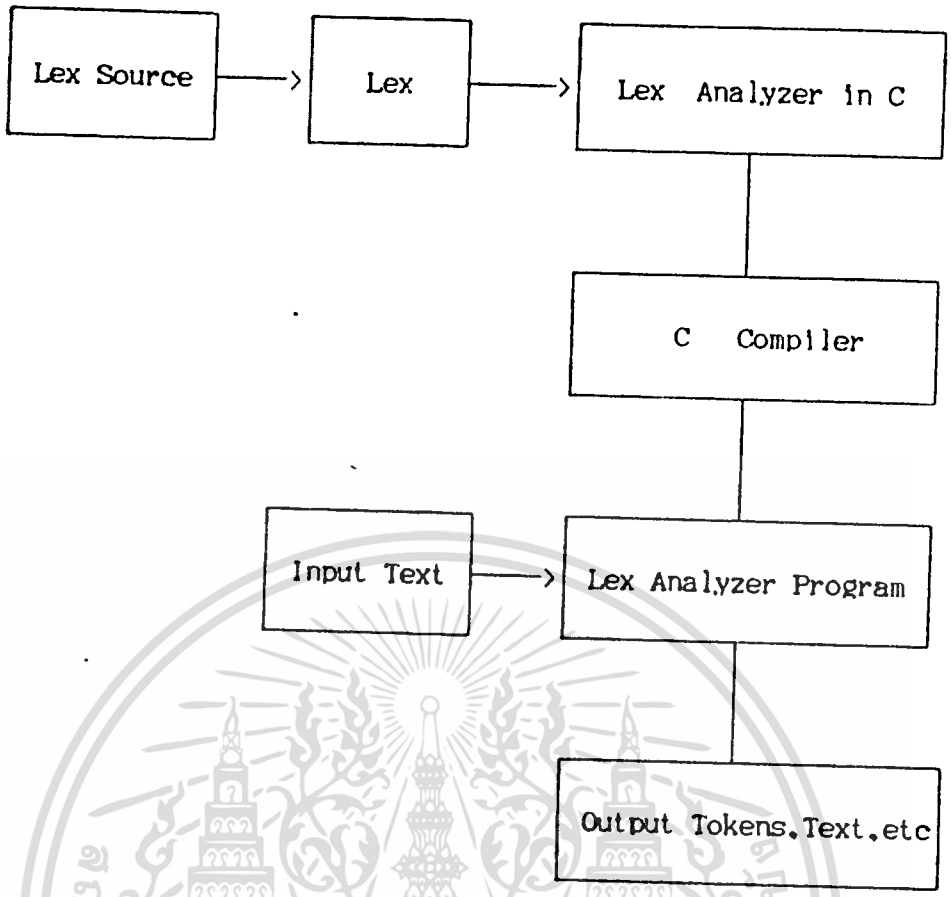
ในการสร้างตัวแปลภาษาโปรลอกไลค์นั้น เราจะใช้หลักการของการสร้างคอมไพเลอร์ ซึ่งจะต้องประกอบไปด้วยส่วน วิเคราะห์ศัพท์ (Lexical Analyzer) , ส่วนวิภัช (Parser) และ ส่วนก่อกำเนิดรหัสคำสั่ง (Code Generator) ในการจะสร้างตัวแปลภาษาให้ได้สำเร็จภายในเวลาจำกัดและมีความผิดพลาดน้อยที่สุดนั้น จำเป็นต้องใช้เครื่องมือช่วยในการพัฒนา และเพื่อความสะดวกรวดเร็วที่สุด เราจะใช้เครื่องมือที่เป็นซอฟแวร์ 2 โปรแกรมในการช่วยสร้าง ส่วนวิเคราะห์ศัพท์ และ ส่วนวิภัช แต่ส่วนก่อกำเนิดรหัสคำสั่งจะต้องพัฒนาเองทั้งหมด

สำหรับส่วนวิเคราะห์ศัพท์ซึ่งเป็นส่วนที่ใช้ในการแยกคำจะใช้โปรแกรมที่ช่วยในการสร้างคือ lex และ ส่วนวิภัชที่ใช้สำหรับเช็คไวยากรณ์จะใช้โปรแกรมในการช่วยพัฒนาคือ โปรแกรม yacc

Lex และ Yacc นั้นอยู่บนระบบปฏิบัติการยูนิกซ์ (Unix) แต่โครงงานนี้อยู่บนระบบปฏิบัติการดอส (Dos) และไม่สามารถหาตัวโปรแกรม Lex และ Yacc บนระบบปฏิบัติการดอสได้ ดังนั้นจึงต้องใช้วิธีการเลี่ยงโดยการสร้างส่วนวิเคราะห์ศัพท์ และ ส่วนวิภัช โดยใช้โปรแกรม Lex และ Yacc บนระบบปฏิบัติการยูนิกซ์ก่อน แล้วค่อยเอา ผลลัพธ์ที่เป็น โปรแกรมภาษาซีที่ได้มาใช้บนระบบปฏิบัติการดอสอีกที ที่เราสามารถทำเช่นนี้ได้ก็ด้วยความสามารถในเรื่อง การโยกย้ายโปรแกรมได้ง่าย (Portable)ของภาษาซีนั่นเอง

3.3.1 Lex

ในการใช้ Lex มาช่วยในการสร้างส่วนวิเคราะห์ศัพท์นั้น จำเป็นต้องทำความเข้าใจรูปที่ 3-3 ก่อน



รูปที่ 3-3 ไดอะแกรมแสดงการนำ Lex ไปใช้งาน

จากรูปที่ 3-3 จะเห็นว่าอินพุท (Input) ของโปรแกรม Lex ต้องเป็น ภาษาที่ Lex เข้าใจ ดังนั้น ก่อนที่จะใช้งาน Lex ได้จึงควรมีความรู้เรื่องส่วนวิเคราะห์ศัพท์เป็นอย่างดี เพื่อจะได้กำหนดให้ Lex รู้ได้ว่า โทเคน (Tokens) ต่างๆในภาษาที่เราต้องการเป็นเช่นไร หลังจาก Lex ได้รับอินพุท จะสร้าง โปรแกรมวิเคราะห์ศัพท์ ออกมาเป็นผลลัพท์ภาษาซี ซึ่งจะใช้งานได้ก็ต่อเมื่อนำไปคอมไพล์ด้วยคอมไพเลอร์ภาษาซีเสียก่อน ผลลัพธ์ก็คือ โปรแกรมส่วนวิเคราะห์ศัพท์ ที่สามารถทำงานได้ทันที โดยอินพุทของมันจะเป็นเท็กซ์ไฟล์ (Text File) ที่บรรจุตัวโปรแกรมภาษาที่เราต้องการคอมไพล์ไว้ โดยผลลัพธ์ที่ได้จะเป็นโทเคน (Tokens) ของตัวภาษาที่เราต้องการ

สำหรับโครงงานนี้จะนำส่วน โปรแกรมวิเคราะห์ศัพท์ ที่ยังเป็นภาษาซีไปใช้ โดยจะนำไปรวมกับส่วนวิภัช ที่ได้จาก Yacc และ สวมก่อกำเนิดรหัสคำสั่ง เป็นตัวคอมไพเลอร์ที่สมบูรณ์ต่อไป

3.3.1.1 การเขียนโปรแกรม Lex

โปรแกรม Lex จะประกอบด้วย 3 ส่วนคือ

1. ส่วนนิยาม (Definitions)

2. ส่วนกฎ (Rules)

3. ส่วนยูสเซอร์ซับรูวีน (User Subroutines)

โดยส่วนของส่วนกฎ จะเป็นส่วนบังคับที่จำเป็นต้องมี สำหรับส่วนนิยาม และ ส่วนยูสเซอร์ซับรูวีน ไม่ต้องมีก็ได้

3.3.1.1.1 ส่วนนิยาม

ส่วนนี้จะถูกประกาศระหว่างเครื่องหมาย %{ และ %} โดยมีลักษณะเหมือนกับส่วนนิยาม (Definition) ในภาษาซี อันประกอบไปด้วย ส่วน extern เพื่อบอกว่า ข้อมูล หรือฟังก์ชันนั้นเป็น เอกซ์เทอร์นัลลิงเกจ (External Linkage), ส่วนนิพจน์ #include และส่วนประกาศตัวแปร ดังตัวอย่าง

```
%{
#include "y.tab.h"
extern  int  tokval;
int  lineno;
%}
```

3.3.1.1.2 ส่วนกฎ

จะขึ้นต้นด้วย %% และจบด้วย %% โดยกฎแต่ละกฎ จะประกอบไปด้วย 2 ส่วนคือ ส่วนที่เป็นการบรรยายหรือรูปแบบของโทเคนเรียกว่า แพทเทิร์น (Pattern) และ ส่วนที่จะทำงานเมื่อพบแพทเทิร์นดังกล่าว ซึ่งส่วนนี้จะเป็นนิพจน์ของ ภาษาซี 1 นิพจน์ ดังตัวอย่าง

```
%%
-[0-9]+          printf("negative integer");
+?[0-9]+         printf("positive integer");
-0.[0-9]+        printf("negative fraction.no whole number part");
rail[ ]*road     printf("railroad is one word");
crook            printf("Here's a crook.");
function         subprogcount++;
```

```
G[a-zA-Z]*      { printf("may have a G word here: ",yytext);
                  Gstringcount++; }
%%
```

3.3.1.1.3 ส่วนยูสเซอร์ซับรูทีน (User Subroutine)

เขียนต่อจากส่วนกฎ โดยในส่วนนี้ ให้โปรแกรมเมอร์เขียนฟังก์ชันภาษาซี ตามปกติ ซึ่งฟังก์ชันที่เขียนไว้ในส่วนนี้สามารถถูกเรียกใช้จากส่วนของกฎได้ ดังตัวอย่าง

```
%%
.
.
"/*"                    skipcmnts();
.
.
%%
skipcmnts()
{
    for(;;)
    {
        while (input() != '*');
        if (input() != '/') {
            unput(yytext[yyleng-1]);
        else
            return;
        }
    }
}
```

3.3.1.2 โทเคนต่างๆในภาษาโปรลอกไลค์

โทเคนต่างๆในภาษาโปรลอกไลค์นั้น ประกอบไปด้วยโทเคนของชื่อบิวท์อินฟังก์ชัน, โทเคนของลายวลี, โทเคนของเลขจำนวนเต็ม, โทเคนของเลขจำนวนจริง, โทเคนของตัวอักษร, โทเคนของสัญญลักษณ์ ; โทเคนของชื่อเพรดดิเคท เป็นต้น โดยมีการกำหนดค่าของโทเคนต่างๆเพื่อเป็นอินพุทสำหรับโปรแกรมส่วนวิภัชต่อไป สำหรับรายละเอียดและค่าของโทเคนต่างๆปรากฏอยู่ในไฟล์ y.tab.h ซึ่งนำมาแสดงไว้ที่นี้แล้ว

```
# define DOMAINS 257
# define T_INT 258
# define T_REAL 259
# define T_STR 260
# define T_SYM 261
# define T_CHAR 262
# define FACT_PRED 263
# define PREDICATES 264
# define CLAUSES 265
# define VARIABLE 266
# define NAME 267
# define _INTEGER 268
# define _REAL 269
# define _STRING 270
# define _CHAR 271
# define IF 272
# define DOT 273
# define COMMA 274
# define EQUAL 275
# define OPEN 276
# define CLOSE 277
# define GOAL 278
# define ERR_STR 279
# define ERR_SYN 280
```

```
# define ADD 281
# define CUT 282
# define DIV 283
# define EQU 284
# define FAIL 285
# define LESS 286
# define LESS_EQU 287
# define MORE 288
# define MORE_EQU 289
# define MUL 290
# define NL 291
# define READ 292
# define SUB 293
# define WRITE 294
```

สำหรับโปรแกรม Lex สำหรับภาษาโปรลอกไลค์เป็นดังนี้

```
%%{
#include "y.tab.h"
char      flag=0;
char      str[253]="";
int       i=0;
%}
D       [0-9]
C       [A-Za-z0-9 ~!@#$%^&\*()_=|;:',<\.>/\\\+\-\?\{\}\[\] ]
C       [A-Za-z0-9 ~!@#$%^&\*()_=|;:",<\.>/\+\-\?\{\}\[\] ]
%%
add             return(ADD);
\!              return(CUT);
div             return(DIV);
equal           return(EQU);
```

```
fail            return(FAIL);
less            return(LESS);
less_equal      return(LESS_EQU);
more            return(MORE);
more_equal      return(MORE_EQU);
mul             return(MUL);
nl              return(NL);
read            return(READ);
sub             return(SUB);
write           return(WRITE);
goal            {   printf("GOAL !\n");
                    return(GOAL);
                }
predicates      { printf("PREDICATES \n");
                  return(PREDICATES);
                }
clauses         { printf("CLAUSES \n");
                  return(CLAUSES);
                }
symbol          { printf("SYMBOL");
                  return(T_SYM);
                }
domains         { printf("DOMAINS");
                  return(DOMAINS);
                }
integer         { printf("INTEGER");
                  return(T_INT);
                }
real            { printf("REAL");
                  return(T_REAL);
```

```
                }
string          { printf("STRING");
                  return(T_STR);
                }
char            { printf("CHAR");
                  return(T_CHAR);
                }
fact_predicates { printf("FACT_PRED");
                  return(FACT_PRED);
                }
-{D}+           { printf("OK! NEGATIVE INTEGER : %s\n",yytext);
                  return(INTEGER);
                }
\+?{D}+         { printf("OK! POSITIVE INTEGER : %s\n",yytext);
                  return(INTEGER);
                }
-{D}+\.{D}+     { printf("OK! NEGATIVE REAL : %s\n",yytext);
                  return(REAL);
                }
\+?{D}+\.{D}+   { printf("OK! POSITIVE REAL : %s\n",yytext);
                  return(REAL);
                }
'{C}'           { printf("OK! CHARACTER : %s\n",yytext);
                  return(CHAR);
                }
'\\''           { printf("OK! ESCAPE CHARACTER : %s\n",yytext);
                  return(CHAR);
                }
'\\\\'          { printf("OK! ESCAPE CHARACTER : %s\n",yytext);
                  return(CHAR);
```

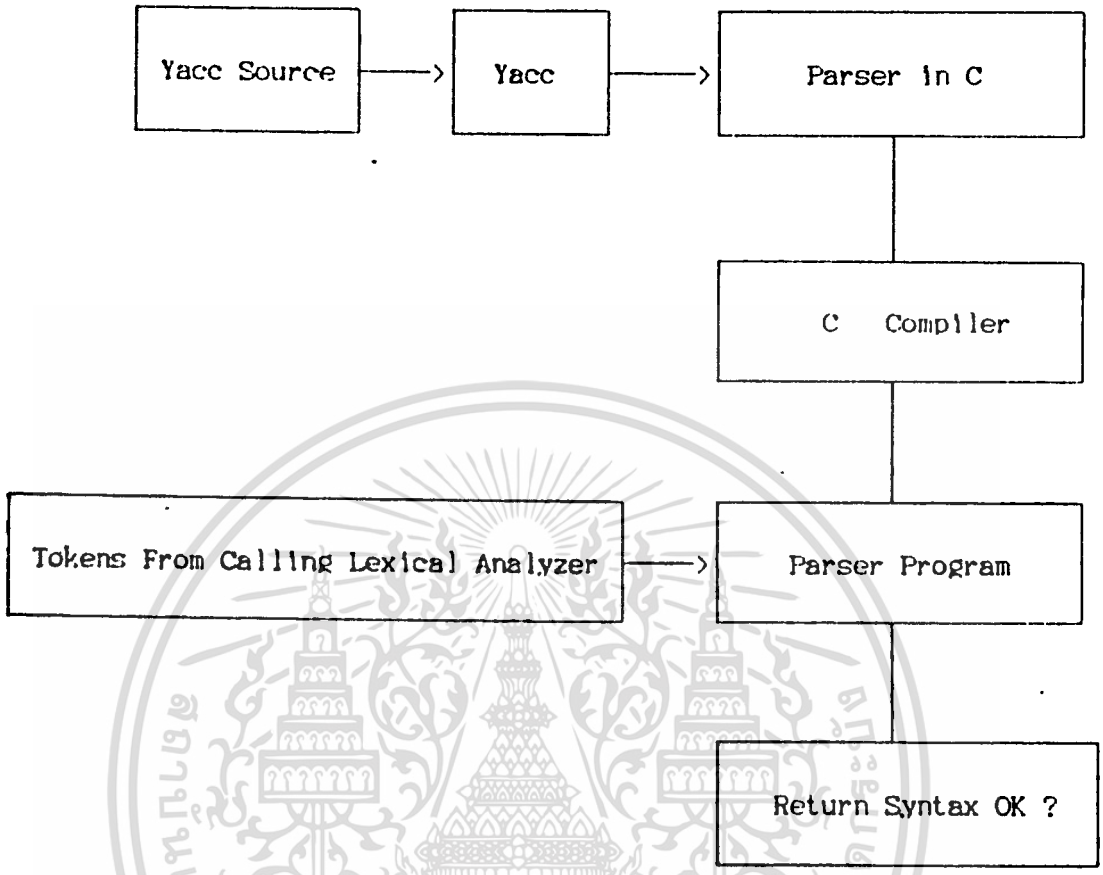
```
                }
[a-z]+          { printf("OK! Name : %s\n",yytext);
                  return(NAME);
                }
[A-Z][a-zA-Z0-9_]* { printf("OK! VARIABLE : %s\n",yytext);
                    return(VARIABLE);
                  }
\"              {
                  if (i==0)
                     str[i++]='"';
                  while (flag==0)
                  {
                    printf("i=%d\n",i);
                    if (i>=250)
                    {
                       str[0]='\0';
                       flag=0;
                       i=0;
                       printf("ERROR!\n");
                       return(ERR_STR);
                    }
                    if ((str[i]=(char)input())=='"')
                       {
                       if (str[i-1]!='\\')
                        {
                            str[i+1]='\0';
                            strcpy(yytext,str);
                            flag=1;
                        }
                       }
```

```
                    i++;
                }
                flag=0;
                str[0]='\0';
                i=0;
                printf("OK! STRING = %s\n",yytext);
                return(STRING);
            }
"/*"        skipcmnts();
:-          { printf("IF : %s\n",yytext);
              return(IF);
            }
\,          { printf("COMMA : %s\n",yytext);
              return(COMMA);
            }
\.          { printf("DOT : %s\n",yytext);
              return(DOT);
            }
\(          { printf("OPEN BRACKET\n");
              return(OPEN);
            }
\)          { printf("CLOSE BRACKET\n");
              return(CLOSE);
            }
\=          { printf("EQUAL! \n");
              return(EQUAL);
            }
%%
skipcmnts()
{
```

```
for(;;)
{
  while (input() != '*');
  if (input() != '/')
    unput(yytext[yyleng-1])
  else
   {
       printf("Comment %s\n",yytext);
       return;
   }
 }
}
```

3.3.2 Yacc

ในการใช้นำ Yacc ไปใช้สามารถอธิบายได้ด้วยไดอะแกรมต่อไปนี้



รูปที่ 3-4 ไดอะแกรมแสดงการนำ Yacc ไปใช้งาน

การจะใช้ Yacc ต้องเตรียมส่วนต่างๆดังนี้

1. สัญกรณ์บีเอ็นเอฟ (BNF) ของ ภาษาที่ต้องการทำตรวจสอบไวยากรณ์
2. รหัสคำสั่งที่จะถูกเรียกเมื่อกระทำการตรวจสอบไวยากรณ์แล้ว ตรงตามสัญกรณ์บีเอ็นเอฟ ที่กำหนดไว้
3. ส่วนนิยาม หรือ ส่วนประกาศของรูทีน ที่จะใช้ตรวจสอบ

การเขียนโปรแกรม Yacc ประกอบด้วย 3 ส่วนเช่นเดียวกับ Lex คือ ส่วนประกาศ (Declarations), กฎไวยากรณ์ทางภาษา (grammar rules) และ ส่วนซับรูทีน (Subroutine) ซึ่งส่วนต่างๆเหล่านี้จะถูกขั้นด้วย %% เพราะฉะนั้น โปรแกรม Yacc จะมีโครงร่างดังนี้

```
declarations
%%
rules
%%
subroutions
```

ในส่วนของการประกาศนั้น ชื่อทุกชื่อที่เป็นเทอร์มินัล (terminal) จะต้องถูกประกาศไว้ ดังนี้

```
%token  name1,name2 ...
```

สำหรับชื่อนอกเหนือจากนี้จะถือว่าเป็น นอนเทอร์มินัล (nonterminal) เพราะฉะนั้นชื่อเหล่านั้นจะต้องถูกพบทางซ้ายของกฎไวยากรณ์ (grammar rule) ใด กฎไวยากรณ์หนึ่ง

สำหรับในส่วนของกฎจะเป็น ส่วนที่เกี่ยวกับกฎไวยากรณ์โดยตรง โดยจะมีรายละเอียดของกฎไวยากรณ์ และ การดอบสนองเมื่อพบว่าตรงตามกฎไวยากรณ์นั้น เป็นภาษาซี 1 นิพจน์ เช่น

```
A : '(' B ')'
    {
        hello(1,"abc");
    }
XXX : YYY ZZZ
    {
        (void)printf("a message\n");
        flag=25;
    }
```

ส่วนของซับรูทีน จะเป็นฟังก์ชันภาษาซีเหมือนกับส่วนของยูสเซอร์ซับรูทีนของ Lex เช่นกัน

ผลลัพธ์ของ Yacc จะเป็น ฟังก์ชันภาษาซี ที่ทำหน้าที่ในการตรวจสอบไวยากรณ์ตามสัญกรณ์ บีเอ็นเอฟ ว่าถูกต้องหรือไม่ และเมื่อนำผลลัพธ์ของ Yacc ซึ่งเป็นฟังก์ชันภาษาซีชื่อ yyparse() สำหรับใช้ในการตรวจสอบไวยากรณ์ มารวมกับ ผลลัพธ์ของ Lex ที่เป็นฟังก์ชันภาษาซีชื่อ yylex ที่ทำหน้าที่ในการตัดคำแยกแยะ โทเคนต่างๆแล้ว ก็จะเหลือเพียงส่วนก่อกำเนิดวัสดุสิ่ง ที่เราจะต้องสร้างเองทั้งหมด

3.3.2.1 สัญกรณ์บีเอ็นเอฟของภาษาโปรลอกไลค์

สัญกรณ์บีเอ็นเอฟของภาษาโปรลอกไลค์จะต้องถูกนำมาเขียนเป็นโปรแกรมอินพุทสำหรับโปรแกรม Yacc ดังต่อไปนี้

```
%token    DOMAINS
%token    T_INT
%token    T_REAL
%token    T_STR
%token    T_SYM
%token    T_CHAR
%token    FACT_PRED
%token    PREDICATES
%token    CLAUSES
%token    VARIABLE
%token    NAME
%token    INTEGER
%token    REAL
%token    STRING
%token    CHAR
%token    IF
%token    DOT
%token    COMMA
%token    EQUAL
%token    OPEN
%token    CLOSE
%token    GOAL
%token    ERR_STR
%token    ERR_SYN
%token    AND
%token    CUT
```

```
%token      DIV
%token      EQU
%token      FAIL
%token      LESS
%token      LESS_EQU
%token      MORE
%token      MORE_EQU
%token      MUL
%token      NL
%token      READ
%token      SUB
%token      WRITE

%start      program

%%

namelist        :   NAME | NAME COMMA namelist
{
    name_list();
}
;
domaintype      :   T_INT | T_STR | T_REAL | T_SYM | T_CHAR
{
    domain_type();
}
;
domaindef       :   namelist  EQUAL  domaintype
{
    domain_def();
```

```
}
;
domaindeflist   :   domaindef | domaindef  domaindeflist
{
    domain_def_list();
}
;
domainsec       :   DOMAINS domaindeflist
{
    domain_sec();
}
;
factpredsec     :   FACT_PRED  preddeflist1
{
    fact_pred_sec();
}
;
preddeflist1    :  preddef1  | preddef1  preddeflist1
{
   pred_def_list1();
}
;
preddef1        :  NAME OPEN typelist CLOSE
{
   pred_def1();
}
;
preddef2        :  NAME | preddef1
{
   pred_def2();
```

```
}
;
preddeflist2    :  preddef2 | preddef2 preddeflist2
{
   pred_def_list2();
}
;
typelist        :  type | type COMMA typelist
{
   type_list();
}
;
type            :  NAME | domaintype
{
   type();
}
;
predsec         :  PREDICATES  preddeflist2
{
   pred_sec();
}
;
clausesec       :  CLAUSES  clauselist
{
   clause_sec();
}
;
clauselist      :  clause | clause clauselist
{
   clause_list();
```

```
}
;
clause          :  fact | predicate
{
   clause();
}
;
fact            :  NAME  | NAME  fact_element
{
   fact();
}
;
fact_element :  OPEN constant list CLOSE DOT
{
fact_element();
}
;
constant list    :  constant | constant COMMA constant list
{
   constant_list();
}
;
constant        :  INTEGER | REAL | STRING | NAME | CHAR
{
   constant();
}
;
predicate       :  NAME IF bodylist DOT |
                   NAME pred_element IF bodylist DOT
{
```

```
    predicate();
}
;
pred_element :  OPEN argumentlist CLOSE
{
pred_element();
}
;
argumentlist     :  argument | argument COMMA argumentlist
{
   argument_list();
}
;
argument         :  VARIABLE | constant
{
   argument();
}
;
bodylist         :  body | body COMMA bodylist
{
   body_list();
}
;
body             :  NAME | NAME pred_element | built_in
{
   body();
}
;
built_in         :  add | CUT | div | equ | FAIL | less | less_equ |
                    more | more_equ | mul | NL | readf | sub | writef
```

```
    {
       built_in();
    }
    ;
    add             :  ADD OPEN argument COMMA argument COMMA
                       argument CLOSE
    {
       add();
    }
    ;
    div             :  DIV OPEN argument COMMA argument COMMA
                       argument CLOSE
    {
       div();
    }
    ;
    equ             :  EQU OPEN argument COMMA argument CLOSE
    {
       equ();
    }
    ;
    less            :  LESS OPEN argument COMMA argument CLOSE
    {
       less();
    }
    less_equ        :  LESS_EQU OPEN argument COMMA argument CLOSE
    {
       less_equ();
    }
    ;
```

```
more            :  MORE OPEN argument COMMA argument CLOSE
{
   more();
}
;
more_equ        :  MORE_EQU OPEN argument COMMA argument CLOSE
{
   more_equ();
}
;
mul             :  MUL OPEN argument COMMA argument COMMA
                   argument CLOSE
{
   mul();
}
;
readf           :  READ OPEN argument CLOSE
{
   readf();
}
;
sub             :  SUB  OPEN argument CLOSE
{
   sub();
}
;
writef          :  WRITE OPEN argument CLOSE
{
   writef();
}
```

```
;
goalsec          : GOAL predicate
{
   goal_sec();
}
;
program          : constainsec | optionsec constainsec
{
   program();
}
;
option           : domainsec | factpredsec
{
   option();
}
;
constainsec      : predsec clausesec goalsec
{
   constain_sec();
}
;
optionsec        : option | option optionsec
{
   option_sec();
}
;

%%
option_sec() {}
constain_sec(){}
```

option(){}

program(){}

goal_sec(){}

body(){}

body_list(){}

argument(){}

argument_list(){}

predicate(){}

constant(){}

constant_list(){}

fact(){}

clause(){}

clause_list(){}

clause_sec(){}

pred_sec(){}

type(){}

type_list(){}

pred_def1(){}

pred_def_list1(){}

pred_def2(){}

pred_def_list2(){}

fact_pred_sec(){}

domain_sec(){}

domain_def(){}

domain_def_list(){}

domain_type(){}

name_list(){}

add(){}

div(){}

equ(){}

```
less(){}
less_equ(){}
more(){}
more_equ(){}
mul(){}
readf(){}
sub(){}
writef(){}
bullt_in(){}
fact_element(){}
pred_element(){}
```

3.4 บิวท์อินฟังก์ชัน (Bullt-in Function)

หัวข้อสุดท้ายในบทนี้ที่จะกล่าวถึงคือ บิวท์อินเพรดดิเคท (Bullt-in Predicate) ซึ่งจะอำนวยประโยชน์ให้กับนักพัฒนาโปรแกรมมากทั้งในด้านของการคำนวณ, การติดต่อกับผู้ใช้, การควบคุมทิศทางของโปรแกรมว่าจะให้เกิดผลอย่างไร เป็นต้น อย่างไรก็ดี เนื่องด้วยภาษาโปรลอกโดยธรรมชาติจะมีฟังก์ชันมาตรฐานน้อยมาก และไม่เหมาะกับงานด้านการคำนวณ จึงจะเห็นว่าภาษาโปรลอกไลค์มีบิวท์อินฟังก์ชันที่ใช้ในการคำนวณอยู่น้อย ประกอบกับเป็นงานวิจัยในขั้นแรก เราจึงเน้นไปที่แนวความคิดใหญ่ๆ คือ การติดต่อกับฐานข้อมูลแบบอนุมานกฎได้ สำหรับส่วนบิวท์อินฟังก์ชันจึงเป็นส่วนที่สร้างมาเพื่อความสมบูรณ์ของระบบในการใช้งานขั้นต้นเท่านั้น บิวท์อินฟังก์ชันที่เราสร้างขึ้นมาจึงประกอบด้วย

เพรดดิเคทในการคำนวณคือ

add(a,b,c)	การบวก (a=b+c)
sub(a,b,c)	การลบ (a=b-c)
mul(a,b,c)	การคูณ (a=b*c)
div(a,b,c)	การหาร (a=b/c)

เพรดดิเคทในการกระทำการเปรียบเทียบ คือ

more(a,b)	มากกว่า	(a > b)
more_equal(a,b)	มากกว่าหรือเท่ากับ	(a >= b)
less(a,b)	น้อยกว่า	(a < b)

less_equal(a,b) น้อยกว่าหรือเท่ากับ (a <= b)

equal(a,b) เท่ากับ (a = b)

เพรดดิเคททางด้านอินพุทและเอาท์พุท

read(a) อ่านค่าจาก อินพุทมาตรฐานคือ คีย์บอร์ด

write(a) เขียนค่าออก เอาท์พุทมาตรฐานคือ จอภาพ

เพรดดิเคทในการควบคุมการทำงานซึ่งมีอยู่ในภาษาโปรลอก และ เห็นว่ามีประโยชน์จึงคงไว้ในภาษาโปรลอกไลค์ คือ

! เรียกว่า เพรดดิเคทคัท

fail เรียกว่า เพรดดิเคทเฟล

ซึ่งทั้งสองเพรดดิเคทมีรายละเอียดเหมือนในภาษาโปรลอกทุกอย่าง จึงไม่ขอกล่าวถึงในที่นี้

บทที่ 4

การพัฒนาระบบควบคุมการทำงานของภาษาโปรลอกไลค์

ในการออกแบบคอมไพเลอร์ภาษาโปรลอกไลค์ซึ่งใช้หลักการในการทำยูนิฟิเคชั่นเช่นเดียวกับภาษาโปรลอกปกติ ได้แบ่งส่วนประกอบของกลไกการทำงานออกเป็น 2 ส่วนด้วยกันคือ ส่วนของโครงสร้างข้อมูลที่ใช้ในการเก็บกฎและข้อเท็จจริงของโปรแกรมที่ผู้พัฒนาโปรแกรมเขียนขึ้นมา และส่วนขั้นตอนวิธีในการนำเอาโครงสร้างข้อมูลที่ถูกสร้างขึ้นมาโดยการคอมไพล์ จากคอมไพเลอร์ซึ่งเป็นอีกส่วนหนึ่งที่โครงงานนี้ทำขึ้นมา มากระทำการเพื่อหาคำตอบของโกล์ตามขั้นตอนตามหลักของการยูนิฟิเคชั่น ในบทนี้จะกล่าวถึงรายละเอียดของโครงสร้างข้อมูลชนิดต่างๆที่นำมาเชื่อมโยงกันจนสามารถใช้เก็บรายละเอียดต่างๆได้ครบถ้วน และส่วนของขั้นตอนวิธีในการทำยูนิฟิเคชั่น ซึ่งทั้ง 2 ส่วนมีความเกี่ยวข้องกัน โดยในบางโครงสร้างข้อมูลจะทำการอธิบายขั้นตอนวิธีประกอบไปพร้อมๆกันเลย

4.1 ซิมโบลเทเบิล (SYMBOL TABLE)

เมื่อสังเกตลักษณะของโปรแกรมภาษาโปรลอก จะพบว่าโปรแกรมจะประกอบด้วยเพรดิเคตแบบต่างๆกับคือ เพรดิเคตที่เป็นส่วนหัวของกฎ ส่วนตัวของกฎ และเพรดิเคตที่ใช้ในการเก็บข้อเท็จจริง ภายในแต่ละเพรดิเคตจะมีอาร์กิวเมนต์อยู่จำนวนหนึ่ง ซึ่งมีอยู่ 2 ชนิดคืออาร์กิวเมนต์ที่เป็นค่าคงที่และเป็นตัวแปร นอกจากนี้เรายังพบว่าจำนวนเพรดิเคตภายในแต่ละโปรแกรมจะมีจำนวนคงที่เสมอ ดังนั้นในการเก็บเราสามารถใช้อาเรย์มาทำการเก็บได้เลย ไม่จำเป็นที่จะต้องใช้โครงสร้างประเภทลิงค์ลิสท์ ที่มีความเหมาะสมกับงานที่ต้องมีการเปลี่ยนแปลงข้อมูลอยู่ตลอด และมีความยุ่งยากในการเขียนโปรแกรม การใช้อาเรย์ก็สามารถเก็บรายละเอียดได้ครบถ้วนอยู่แล้ว ก่อนอื่นเรามาดูเพรดิเคตนี้ก่อน

p(X, "a", Y)

เราสามารถนำไปเก็บในซิมโบลเทเบิล(Symbol Table) ได้ดังนี้

	NAME	NAME TYPE	DATA_TYPE	ARITY
1	p	PREDICATE	USER DEFINE	2
2	X	VARIABLE	INTEGER	1
3	a	CONSTANT	STRING	0
4	Y	VARIABLE	STRING	2

รูปที่ 4-1 แสดงรายละเอียดของข้อมูลที่เก็บใน symbol table

จากรูปที่ 4-1 ซึ่งแสดงรายละเอียดของข้อมูลที่เก็บไว้ในซิมโบลเทเบิล ออกแบบให้เป็นอาเรย์ของโครงสร้างที่มีสมาชิกอยู่ 4 ตัวด้วยกันคือ

1. NAME ใช้เก็บชื่อต่างๆที่อยู่ในโปรแกรม คือชื่อของ เพรดิเคต,ตัวแปรและค่าคงที่
2. NAME TYPE บอกให้รู้ถึงชนิดของชื่อว่าเป็น เพรดิเคต,ตัวแปร หรือ ค่าคงที่
3. DATA TYPE บอกชนิดของ NAME TYPE ว่าค่าที่เก็บเป็นชนิดอะไร โดยถ้าเป็นเพรดิเคต ก็จะบอกว่าเป็นชนิด ที่ผู้เขียนโปรแกรมสร้างขึ้นมา หรือ ฟังก์ชันในตัว ค่าคงที่ หรือ ตัวแปร ก็จะบอกว่าเป็น ค่าคงที่ประเภทอะไร ซึ่งมีอยู่ 5 ชนิดคือ ตัวเลขจำนวนเต็ม, เลขจำนวนจริง,ตัวอักษร,สายวลี และ เครื่องหมาย
4. ARITY ในกรณีที่เป็นเพรดิเคตจะบอกจำนวนของอาร์กิวเมนต์ที่อยู่ในเพรดิเคตนั้น ถ้าเป็นตัวแปรจะบอกว่าเป็นตัวแปรตัวที่เท่าไรในเพรดิเคตนั้น และค่านี้จะไม่มีความหมายในกรณีของค่าคงที่

4.2 <u>ลิสท์</u> (LIST)

การใช้อาเรย์ SYMTAB เพียงอย่างเดียวยังไม่สามารถที่จะเก็บรายละเอียดได้เพียงพอ โดยเมื่อเราพิจารณาในส่วนที่เป็นส่วนตัวของกฎ ตัวแปรแต่ละตัวสามารถที่ไปปรากฏอยู่ได้ในหลายเพรดิเคต ดังนั้นแทนที่แทนที่จะอ้างเพรดิเคตผ่านทาง SYMTAB เราจะทำการสร้างอาเรย์ชื่อ LIST ขึ้นมาทำหน้าที่ในการเชื่อมโยงความสัมพันธ์ระหว่างค่าต่างๆที่เก็บอยู่ใน SYMTAB และใช้อาเรย์นี้ในการอ้างถึงเพรดิเคตแทนการอ้างโดยตรงจาก SYMTAB เช่น

```
:- p(X,"a"),q(X).
```

SYMTAB	NAME	NAME TYPE	DATA TYPE	ARITY
1	p	PREDICATE	USER DEFINE	1
2	X	VARIABLE	INTEGER	1
3	a	CONSTANT	STRING	0
4	q	PREDICATE	USER DEFINE	1

LIST	SYMTAB INDEX	NEXT LIST	MEMBER
1	1	4	2
2	2	3	-
3	3	-	-
4	4	-	5
5	2	-	-

รูปที่ 4-2 แสดงรายละเอียดการเชื่อมโยง SYMTAB กับ LIST

จากรูปที่ 4-2 อาเรย์ LIST ประกอบด้วยสมาชิก 3 ตัวซึ่งมีลักษณะเป็นลิสท์เชื่อมโยงความสัมพันธ์ของข้อมูลที่อยู่ใน SYMTAB จนสามารถประกอบกันเป็นส่วนตัวของกฎได้ ความหมายของสมาชิกแต่ละตัวเป็นดังนี้

1. SYMTAB INDEX อินเดกซ์ชี้ไปยัง SYMTAB

2. NEXT LIST ถ้าเป็นเพรดิเคตจะหมายถึงอินเดกซ์ที่ชี้ไปยังเพรดิเคตถัดไป และถ้าเป็นอากิวเมนต์จะเป็นอินเดกซ์ชี้ไปยังอาร์กิวเมนต์ตัวถัดไป

3. MEMBER เป็นอินเดกซ์ชี้ไปยังอาร์กิวเมนต์ของเพรดิเคต

เมื่อเราสร้างอาเรย์นี้ขึ้นมาแล้ว ต่อไปเมื่อเราทำการอ้างเพรดิเคตเราจะอ้างผ่านทางอาเรย์นี้เสมอเพราะจะมีประสิทธิภาพที่ดีกว่า

4.3 คลอส (CLAUSE)

ตอนนี้เราสามารถสร้างส่วนหัวกับส่วนตัวของกฎมาได้แล้ว ต่อไปจะนำส่วนประกอบทั้ง 2 นี้มารวมกันเป็นกฎที่สมบูรณ์ โดยที่ในโปรแกรมโปรลอก กฎแต่ละกฎสามารถมีอยู่ได้หลายแบบ และมีลำดับการทำจากบนไปล่าง ในการทำรีโซลลูชั่น (resolusion) แต่ละครั้งจะเริ่มทำตั้งแต่กฎลำดับแรกที่อยู่ในโปรแกรม ถ้าไม่สำเร็จก็จะไปพิจารณากฎลำดับต่อไป ดังนั้นรูปแบบของอาเรย์ที่ใช้ในการเก็บกฎดังกล่าว ก็จะมีลักษณะคล้ายกับของ LIST และได้เพิ่มข้อมูลลงไปใน SYMTAB เพื่อใช้ในการอ้างอิงอาเรย์นี้ โดยเพิ่มเขตที่เป็นอินเดกซ์ชี้ไปยังกฎตัวแรกของเพรดิเคตนั้น เพราะถ้าเกิดการรีโซลลูชั่นก็สามารถมาดูที่เขตนี้ได้เลย จะได้ลักษณะของอาเรย์ที่เป็นดังนี้

```
anc(A,B) :- par(A,B).
anc(A,B) :- par(A,C),anc(C,B).
par("dang","dum").
```

SYMTAB	NAME	NAME TYPE	DATA TYPE	ARITY	CLAUSE INDEX
1	anc	PREDICATE	USER DEFINE	2	1
2	A	VARIABLE	STRING	1	-
3	B	VARIABLE	STRING	2	-
4	par	PREDICATE	USER DEFINE	2	3
5	anc	PREDICATE	USER DEFINE	3	1
6	A	VARIABLE	STRING	1	-
7	B	VARIABLE	STRING	2	-
8	par	PREDICATE	USER DEFINE	2	3
9	C	VARIABLE	STRING	3	-
10	anc	PREDICATE	USER DEFINE	2	1
11	par	PREDICATE	USER DEFINE	2	3
12	dang	CONSTANT	STRING	-	-
13	dum	CONSTANT	STRING	-	-

รูปที่ 4-3 แสดงความสัมพันธ์ระหว่างอาเรย์ SYMTAB,LIST และ CLAUSE

LIST	SYMTAB INDEX	NEXT LIST	MEMBER
1	1	-	2
2	2	3	-
3	3	-	-
4	4	-	5
5	2	5	-
6	3	6	-
7	5	-	8
8	6	9	-
9	7	-	-
10	8	13	11
11	6	12	-
12	9	-	-
13	10	-	14
14	9	15	-
15	7	-	-
16	11	-	17
17	12	18	-
18	13	-	-

CLAUSE	HEAD	BODY	NEXT CLAUSE
1	1	4	2
2	7	10	-
3	16	-	-

รูปที่ 4-3(ต่อ) แสดงความสัมพันธ์ระหว่างอาเรย์ SYMTAB,LIST และ CLAUSE

อาเรย์ต่อมาดังแสดงในรูปที่ 4-3 คืออาเรย์ CLAUSE ประกอบด้วยสมาชิก 3 ตัวคือ

1. HEAD อินเดกซ์ที่ชี้ไปยัง LIST ที่เป็นส่วนหัวของกฎ

2. BODY อินเดกซ์ที่ชี้ไปยัง LIST ที่เป็นส่วนตัวของกฎ

ในกรณีที่เป็นข้อเท็จจริงที่อยู่ในโปรแกรม จะมองว่าเพรดิเคตนั้นเป็นส่วนหัวของกฎและไม่มีส่วนตัว

3. NEXT CLAUSE อินเดกซ์ที่ชี้ไปยัง CLAUSE ถัดไปของกฎ

ส่วนสมาชิกที่เพิ่มเข้าไปใน SYMTAB คือเซต CLAUSE INDEX ซึ่งเป็นอินเดกซ์ชี้ไปยังอาเรย์ CLAUSE ที่เป็นกฎอันดับแรกที่อยู่ในโปรแกรม ในเวลาที่ทำการรีโซลลูชั่นเมื่อมีการย้อนรอยและพิจารณากฎถัดไป จะทำได้โดนผ่านทางเซต NEXT CLAUSE ในอาเรย์ CLAUSE

4.4 บายด์ดิ้งอาเรย์ (BINDING ARRAY)

ในการทำยูนิฟิเคชั่นเมื่อเกิดการรีโซลลูชั่น จะมีการสร้างโกล์ย่อยขึ้นมาเรื่อยๆ ในลักษณะของแผนภูมิต้นไม้ ตัวแปรแต่ละตัวของโกล์ย่อยจะต้องมีการเชื่อมโยงไปถึงโกล์แรกที่ตัวแปรตัวนี้ถูกสร้างขึ้นมา และสามารถที่จะตรวจสอบได้ว่าตัวแปรตัวนี้ถูกกำหนดค่าแล้วหรือไม่ สามารถที่จะกำหนดค่าของตัวแปรตัวนี้ได้ถ้าหากตัวแปรนี้ยังว่างอยู่ และค่านี้จะต้องเชื่อมไปถึงโกล์ย่อยตัวแรกของตัวแปรนี้ เช่น

```
p(X,Y) :- q(X,Y).
q(A,B) :- r(B,A).
r("b","a").
```

เมื่อทำการยูนิฟายแล้วตัวแปรต่างๆจะถูกกำหนดค่าดังนี้

```
b/B/Y
a/A/X
```

การกำหนดค่าให้ตัวแปรในลักษณะนี้ ได้ออกแบบโครงสร้างข้อมูลที่เรียกว่าบายด์ดิ้งอาเรย์ (binding array)ขึ้นมาซึ่งมีความสามารถกำหนดค่าเป็นเช่นๆแบบนี้ได้ จากตัวอย่างข้างต้นจะได้บายด์ดิ้งอาเรย์เป็นดังนี้

	BARRAY	INDEX	FLAG
X	1	^a	CONSTANT
Y	2	^b	CONSTANT
A	3	1	VARIABLE
B	4	2	VARIABLE

รูปที่ 4-4 แสดงโครงสร้างของบายด์ดั้งอาเรย์

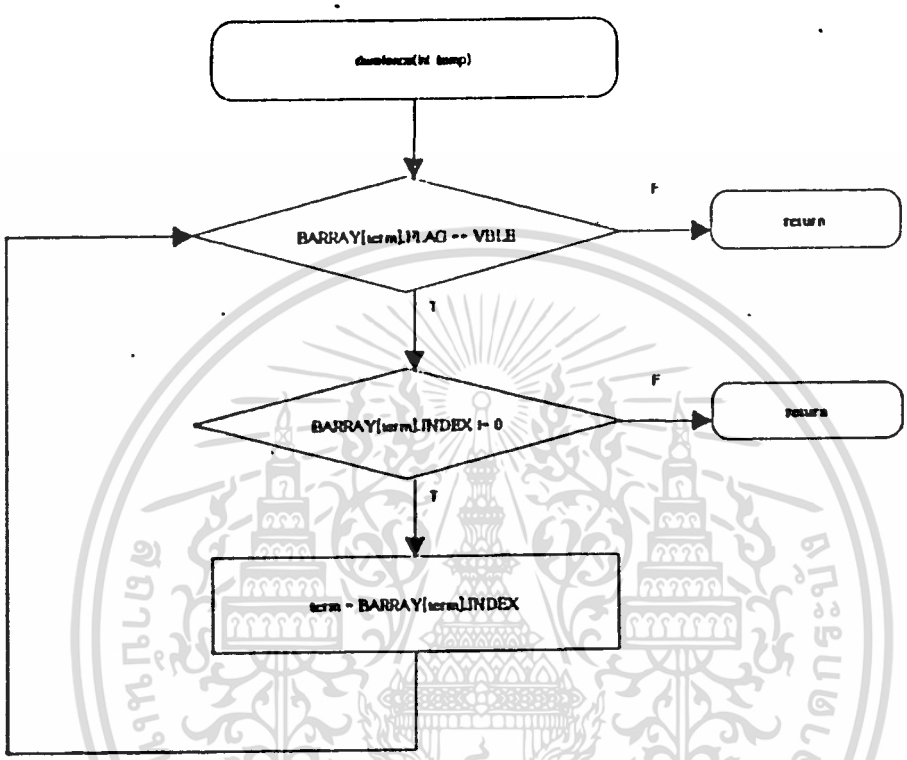
บายด์ดั้งอาเรย์ประกอบด้วยสมาชิก 2 ตัวด้วยกันคือ

1. INDEX ถ้าเป็นตัวแปรค่านี้จะเป็นอินเดกซ์ชี้ไปที่บายด์ดั้งอาเรย์ตัวก่อนหน้านี้ และถ้าเป็นค่าคงที่จะเป็นอินเดกซ์ชี้ไปยังค่าที่อยู่ใน SYMTAB และถ้าตัวแปรนี้ยังว่างอยู่ค่านี้จะเป็น 0

2. FLAG บอกให้รู้ว่าสถานะตอนนี้เป็นค่าคงที่หรือตัวแปร

โครงสร้างของบายด์ดั้งอาเรย์แบบนี้ทำให้เราสามารถที่จะอ้างถึงค่าที่แท้จริงของตัวแปรตัวนี้ได้ตลอดไม่ว่าเราจะอ้างจากตำแหน่งไหน เช่นไม่ว่าเราจะอ้างจากตัวแปร B หรือ Y ก็จะได้ค่าคงที่ b ออกมาเสมอ ซึ่งโครงสร้างดังกล่าวเมื่อนำมาใช้จริงก็พบว่ามีประสิทธิภาพในการทำงานอย่างดี โดยที่จะต้องอาศัยโครงสร้างอื่นช่วยบ้างอีกเล็กน้อยซึ่งจะกล่าวถึงต่อไป

ต่อไปจะแสดงโฟลว์ชาร์ตในการไล่ค่าของบายด์ดั้งอาเรย์ไปจนถึงตัวแปรตัวแรก เรียกฟังก์ชันนี้ว่า dereference โดยรับค่าอาร์กิวเมนต์ที่เป็นอินเดกซ์ของอาเรย์ และคืนค่ากลับเป็นอินเดกซ์ที่ผ่านการไล่ค่าเสร็จแล้ว



รูปที่ 4-5 แสดงโฟลว์ชาวด์ของฟังก์ชัน dereference

4.5 การเปรียบเทียบค่าระหว่าง 2 เพรดิเคต

ที่ผ่านมาเราทราบถึงวิธีการที่จำกำหนดค่าให้กับตัวแปรแล้ว ต่อไปเราจะมาดูว่าในการเปรียบเทียบค่าระหว่าง 2 เพรดิเคต บายด์ดิ้งอาเรย์มีการเปลี่ยนแปลงอย่างไร ในการเปรียบเทียบนี้เราจะต้องพิจารณาด้วยว่าถ้าหากการเปรียบเทียบนี้ไม่สำเร็จ ก็ต้องทำการรีเซ็ตค่าที่อยู่ในอาเรย์กลับด้วย ดังนั้นในการเปลี่ยนแปลงแต่ละครั้งต้องมีการเก็บรายละเอียดว่าเราเปลี่ยนแปลงค่าในอาเรย์ตัวไหนบ้างตลอดเวลา เพื่อใช้ดูว่าต้องรีเซ็ตค่าไหนกลับบ้างเวลาทำการย้อนรอย

ในการเปรียบเทียบเพรดิเคตนี้เราแบ่งการเปรียบเทียบออกเป็น 3 กรณีด้วยกัน ซึ่งแต่ละแบบมีกลไกในการเปรียบเทียบแตกต่างกันไป และเป็นไปตามหลักการของการยูนิฟิเคชั่นคือ

กรณีที่ 1

เพรดิเคตแรกมีอาร์กิวเมนต์เป็นค่าคงที่ และเพรดิเคตที่สองเป็นตัวแปรที่ยังว่าง เช่น

```
p("a","b")
p(X,Y)
```

ในการจัดสรรพื้นที่ในบายด์ดิ้งอาเรย์ให้กับเพรดิเคตนั้น เราจะจัดสรรให้กับค่าที่มีชนิดเป็นตัวแปรเท่านั้น ส่วนค่าคงที่นั้นเราไม่จำเป็นที่จะต้องจัดสรรให้เพราะสามารถดูค่าได้โดยตรงจากอาเรย์ LIST จากเพรดิเคตทั้งคู่สามารถสร้างบายด์ดิ้งอาเรย์ได้เป็นดังนี้

```
BARRAY
    X    1    0    VARIABLE
    Y    2    0    VARIABLE
```

รูปที่ 4-6 แสดงลักษณะของบายด์ดิ้งอาเรย์ที่เกิดจากเพรดิเคต p(X,Y)

เมื่อเริ่มต้นสร้างตัวแปร X กับ Y ยังว่างอยู่ดังนั้นจะมีค่าเป็น 0 และมีชนิดเป็นตัวแปร

เมื่อเริ่มทำการเปรียบเทียบ จะเริ่มทำการเปรียบเทียบจากตัวแปรตัวแรกก่อนคือ X กับ a เพราะเมื่อเราตรวจสอบค่าที่อยู่ใน SYMTAB เราจะทราบว่า X เป็นตัวแปรตัวที่ 1 และ a เป็นค่าคงที่ตัวแรกเช่นกัน จากนั้นก็จะทำการเรียกฟังก์ชัน dereference ค่า X พบว่ายังว่างอยู่ก็จะทำการกำหนดค่าให้ชี้ไปยังค่าคงที่ a ในทำนองเดียวกันเราจะได้ค่า Y ถูกกำหนดให้เป็น b

และเมื่อค่าที่อยู่ในบายด์ดิ้งอาเรย์มีการเปลี่ยนแปลง เราก็จะบันทึกการเปลี่ยนแปลงไว้ในอาเรย์ที่เรียกว่า TRAILSTACK ค่าที่ได้หลังการเปลี่ยนแปลงแล้วจะเป็นดังนี้

BARRAY				TRAILSTACK
X	1	^a	CONSTANT	1
Y	2	^b	CONSTANT	2

รูปที่ 4-7 แสดงบายด์ดิ้งอาเรย์และ TRAILSTACK หลังการเปลี่ยนแปลง

กรณีที่สอง

เพรดิเคตทั้งสองมีอาร์กิวเมนต์ที่มีการกำหนดค่าแล้วหรือเป็นค่าคงที่ เช่น

```
p("a","b")
p("a",X)
```

โดยที่

```
b/X/Y/Z
```

เมื่อเราให้ตัวแปร X ถูกกำหนดค่าเป็น b ดังนั้น BARRAY ในตอนเริ่มต้นจะมีลักษณะเป็นดังนี้

BARRAY			
Z	1	^b	CONSTANT
Y	2	1	VARIABLE
X	3	2	VARIABLE

รูปที่ 4-8 แสดงบายด์ดิ้งอาเรย์ของกรณีที่ 2

เมื่อเริ่มต้นการเปรียบเทียบ อาร์กิวเมนต์ตัวแรกคือ a ของทั้งสองเพรดิเคตจะถูกนำมาเปรียบเทียบ พบว่ามีค่าเท่ากันก็จะนำอาร์กิวเมนต์ตัวที่สองคือค่าคงที่ b และตัวแปร X เมื่อพบว่า X เป็นตัวแปรก็จะทำการเรียกฟังก์ชัน deference X จะได้อินเดกซ์ที่ชี้ไปยังตัวแปร Z ซึ่งเป็นค่าคงที่ เสร็จแล้วก็นำค่าคงที่ทั้งสองมาเปรียบเทียบกันว่ามีค่าเท่ากันหรือไม่ ถ้าเท่าก็จะ

คืนค่าว่าจริง ถ้าไม่เท่าก็จะคืนค่าเป็นเท็จ

กรณีที่สาม

เพรดิเคตแรกเป็นตัวแปรที่ยังไม่ได้ถูกกำหนดค่า ส่วนเพรดิเคตที่สองเป็นตัวแปรหรือค่าคงที่ เช่น

```
p(X,Y)
p(A,B)
```

โดยที่

```
X/M
a/A/C
B/D
```

จะเห็นว่า B,X และ Y เป็นตัวแปรที่ยังว่าง ในขณะที่ A เป็นตัวแปรที่ถูกกำหนดค่าแล้ว เราสามารถเขียนลักษณะของบายด์ดิ้งอาเรย์ได้ดังนี้

```
BARRAY
        M    1    0    VARIABLE
        C    2    a    CONSTANT
        D    3    0    VARIABLE
        A    4    2    VARIABLE
        B    5    3    VARIABLE
        X    6    1    VARIABLE
        Y    7    0    VARIABLE
```

รูปที่ 4-9 แสดงบายด์ดิ้งอาเรย์ก่อนการเปรียบเทียบของกรณีที่ 3

เมื่อเริ่มทำการเปรียบเทียบโดยตัวแปร X กับ A จะถูกนำมาทำการ derefence ก่อนได้อินเดกซ์ของตัวแปร M กับ C มา เมื่อพบว่า M เป็นตัวแปร ก็จะนำอินเดกซ์ที่ชี้ไปยังค่าคงที่ a

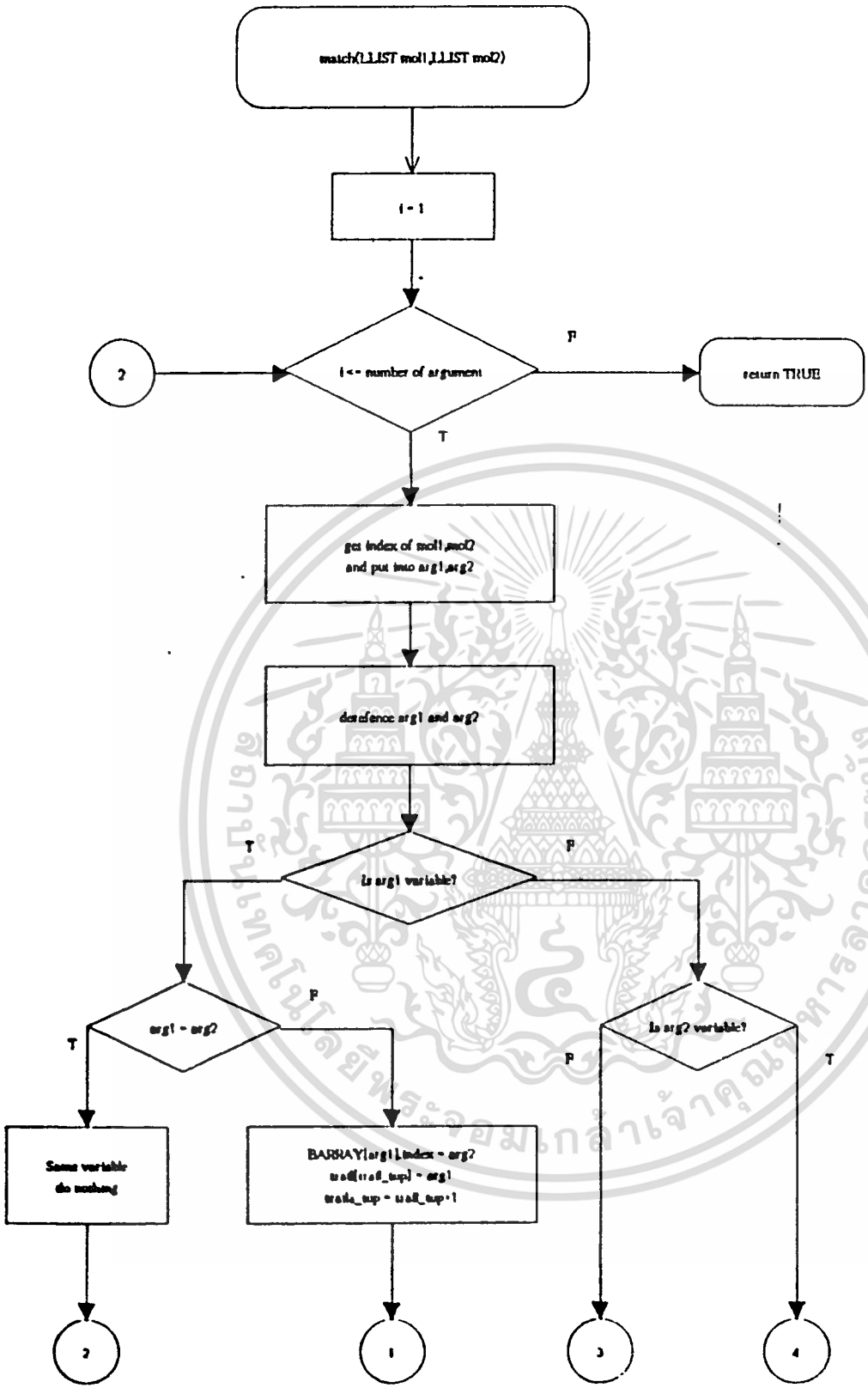
มาใส่ใน M และเปลี่ยนชนิดให้เป็นค่าคงที่ ส่วนตัวแปร B กับ Y ก็มีการทำงานเช่นเดียวกัน แต่จะไม่มีการเปลี่ยนแปลงชนิด เพราะ D กับ Y มีชนิดเป็นตัวแปรทั้งคู่ เราจะได้บายด์ดิ้งอาเรย์หลังจากการเปรียบเทียบเป็นดังนี้

BARRAY TRAILSTACK

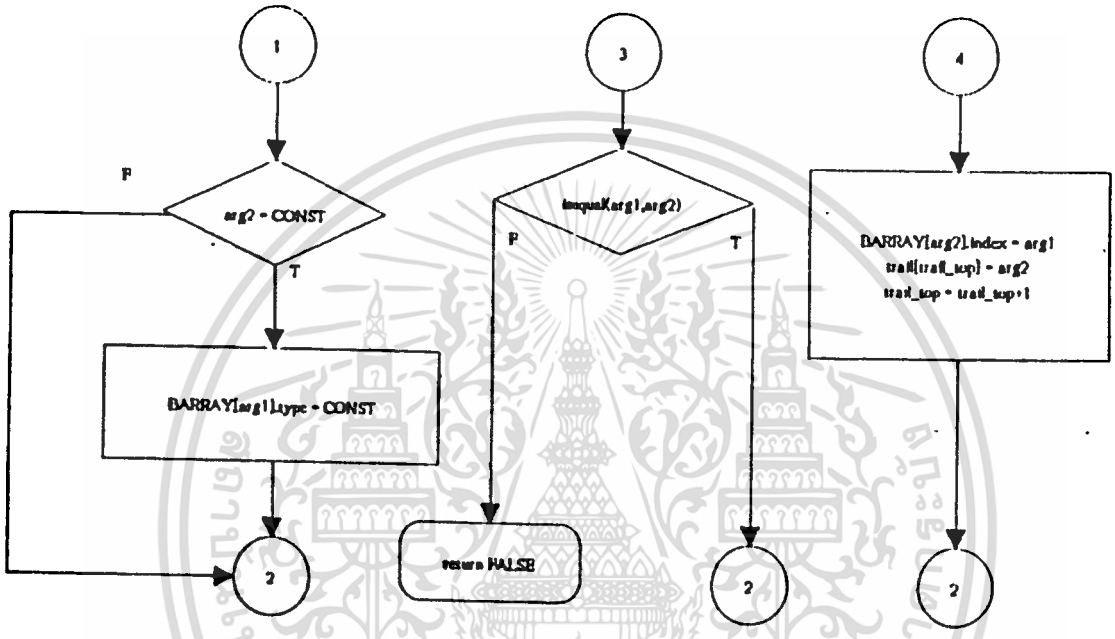
M	1	^a	CONSTANT	1
C	2	^a	CONSTANT	3
D	3	7	VARIABLE	
A	4	2	VARIABLE	
B	5	3	VARIABLE	
X	6	1	VARIABLE	
Y	7	0	VARIABLE	

รูปที่ 4-10 แสดงของบายด์ดิ้งอาเรย์หลังการเปรียบเทียบของกรณีที่ 3

จากการเปรียบเทียบทั้ง 3 กรณี สามารถเขียนโฟลว์ชาร์ดแสดงการทำงานของฟังก์ชันนี้ได้ดังรูปที่ 4-11 เรียกฟังก์ชันนี้ว่า match โดยรับอาร์กิวเมนต์เป็นโครงสร้างข้อมูลที่มีอินเดกซ์ชี้ไปยังเพรดิเคตทั้งสอง และคืนค่ากลับเป็นค่าจริงหรือเท็จ แสดงถึงการเปรียบเทียบว่าสามารถทำได้ตามกรณีทั้งสามข้างต้นหรือไม่



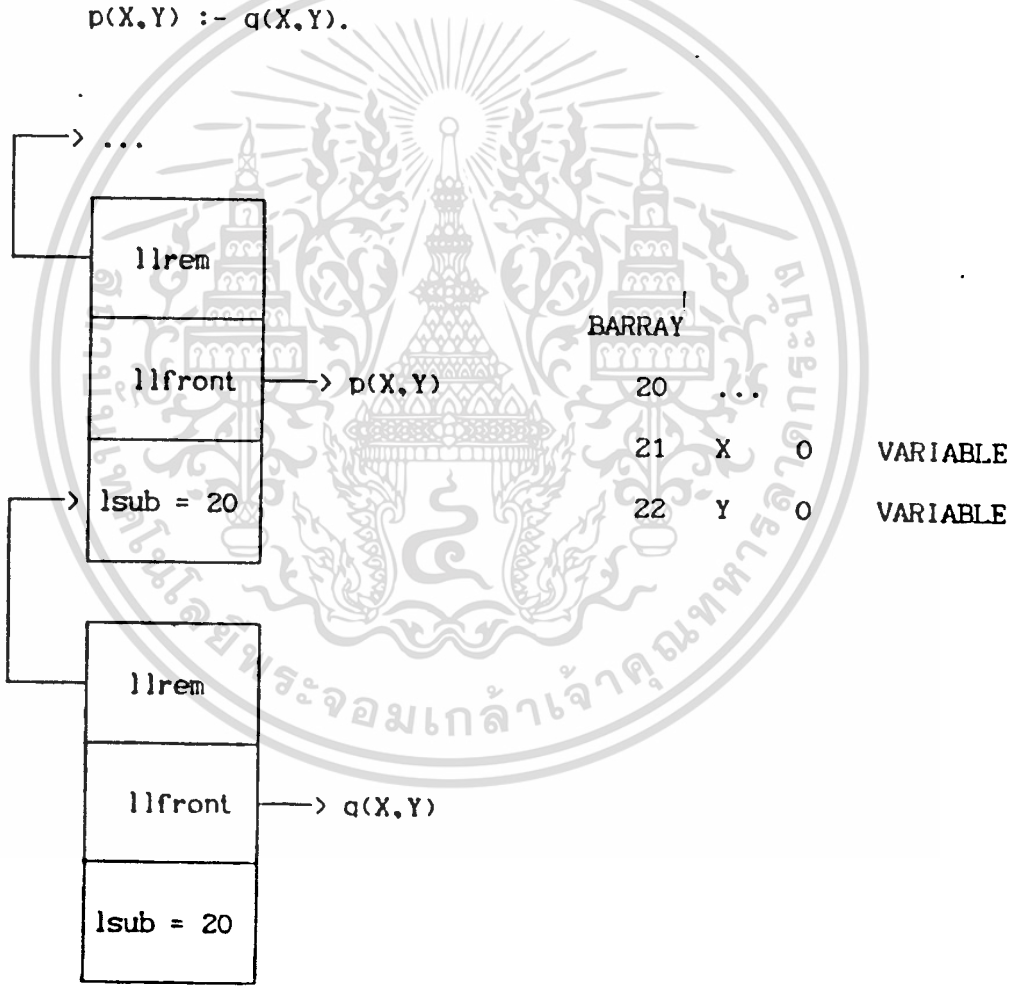
รูปที่ 4-11 แสดงโฟลว์ชาวด์ของฟังก์ชัน match



รูปที่ 4-11(ต่อ) แสดงโฟลว์ชาร์ตของฟังก์ชัน match

4.6 ขั้นตอนในการสร้างโกล์ย่อย

เมื่อโปรแกรมโปรลอกเริ่มทำงานกระบวนการในการรีโซลูชั่นก็จะเริ่มขึ้น โกล์ย่อยต่างๆ ก็จะถูกสร้างขึ้นมาเรื่อยๆ ที่มีโครงสร้างเป็นแผนภูมิต้นไม้ แต่ละโกล์ย่อยก็จะมีตัวแปรเป็นของตัวเอง ดังนั้นจะต้องทำการจัดสรรพื้นที่ในบายด์ดั้งอาเรย์ให้กับตัวแปรเหล่านั้น และขนาดของบายด์ดั้งอาเรย์ก็จะมีขนาดเพิ่มขึ้นเรื่อยๆ ในแต่ละเพรดิเคตจำเป็นที่จะต้องมีตัวแปรที่ใช้บอกให้รู้ว่าตำแหน่งปัจจุบันของบายด์ดั้งอาเรย์อยู่ที่ใด เพรดิเคตนี้ได้รับการรีโซลูชั่นมาจากเพรดิเคตไหน เพื่อที่จะได้รู้ว่าถ้าหากมีการย้อนรอยกลับไปยังเพรดิเคตก่อนหน้านี้จะได้กลับอย่างถูกต้อง หรือใช้ในการค้นหาโกล์ย่อยใหม่ จากความต้องการข้อมูลต่างๆดังกล่าว โครงสร้างข้อมูลอีกตัวหนึ่งที่ต้องใช้โดยใช้ชื่อว่า LLIST จะมีลักษณะดังนี้



รูปที่ 4-12 แสดงโครงสร้างของ LLIST

โครงสร้างข้อมูลนี้ประกอบด้วยสมาชิก 3 ตัวด้วยกันคือ

1. llrem เป็นพอยต์เตอร์ชี้ไปยัง LLIST ที่อยู่ในชั้นก่อนหน้านี้ ซึ่งทำการรีโซลลูชั่นได้ LLIST ในชั้นปัจจุบัน

2. llfront เป็นอินเดกซ์ชี้ไปยัง LIST ของเพรดิเคตนี้

3. lsub ตำแหน่งของบายด์ดิ้งอาเรย์ที่ใช้ในการอ้างตัวแปรที่สร้างขึ้นมา เช่นจากตัวอย่างมีค่าเป็น 20 เมื่อนำมาบวกกับค่า ARITY ของ X กับ Y แล้วจะทำให้ทราบว่าอยู่ในตำแหน่งที่ 21 กับ 22

กลไกการทำงานของ LLIST นี้นับว่ามีส่วนสำคัญมาก เพราะจะเป็นตัวควบคุมความเคลื่อนไหวของการรีโซลลูชั่นทั้งหมดของโปรแกรม ว่าปัจจุบันอยู่ที่สถานะไหน ถ้าเกิดการย้อนรอยจะกลับไปที่ไหน และจะสร้างโกล์ย่อยใหม่ได้อย่างไร โดยทำงานเป็นชั้นซ้อนกันไปเรื่อยๆ และมีโปรแกรมที่เขียนในแบบเรียกตนเองเป็นตัวจัดการ นอกจาก LLIST แล้วยังมีตัวแปรที่ใช้ในการบอกสถานะอีก 2 ตัวซึ่งเป็นตัวแปรแบบโกลบอล ประกอบด้วย

1. trail_top บอกตำแหน่งบนสุดของ TRAIL STACK เพื่อให้เรานำค่าไปใส่ในอาเรย์นี้ได้ถูกต้อง เวลาที่มีการเปลี่ยนแปลงข้อมูลในบายด์ดิ้งอาเรย์

2. newvarindex บอกตำแหน่งบนสุดของบายด์ดิ้งอาเรย์ ใช้ในการจัดสรรพื้นที่ให้กับตัวแปร

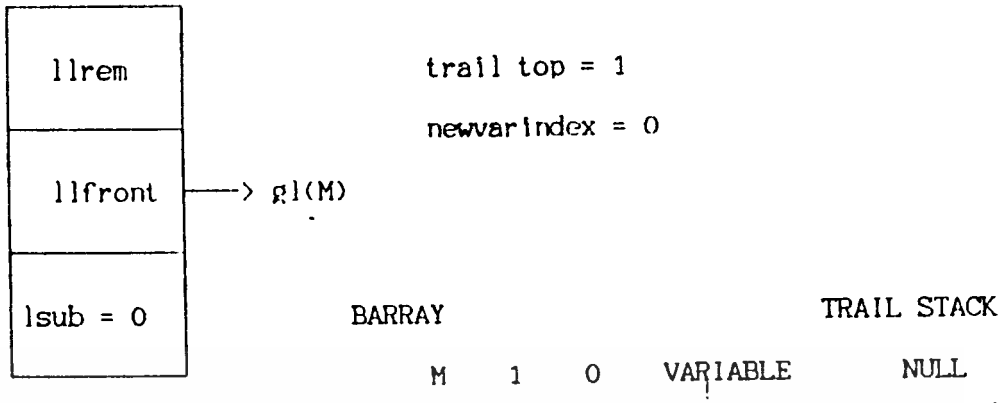
ตอนนี้เรามีโครงสร้างข้อมูลต่างๆและตัวแปรที่ใช้ในการทำงานครบหมดแล้ว ต่อไปเราจะมาดูกันว่าค่าต่างๆ จะเปลี่ยนแปลงไปอย่างไรในการทำรีโซลลูชั่นและยูนิฟิเคชั่น รวมถึงการย้อนรอยกลับด้วย โดยพิจารณาจากตัวอย่างต่อไปนี้

```
p(X) :- q(X,"a"),r(X,"b").
q(A,B) :- s(A,B).
s("c","a").
s("d","a").
r("d","b").
```

โดยโกล์ของโปรแกรมมีค่าเป็น

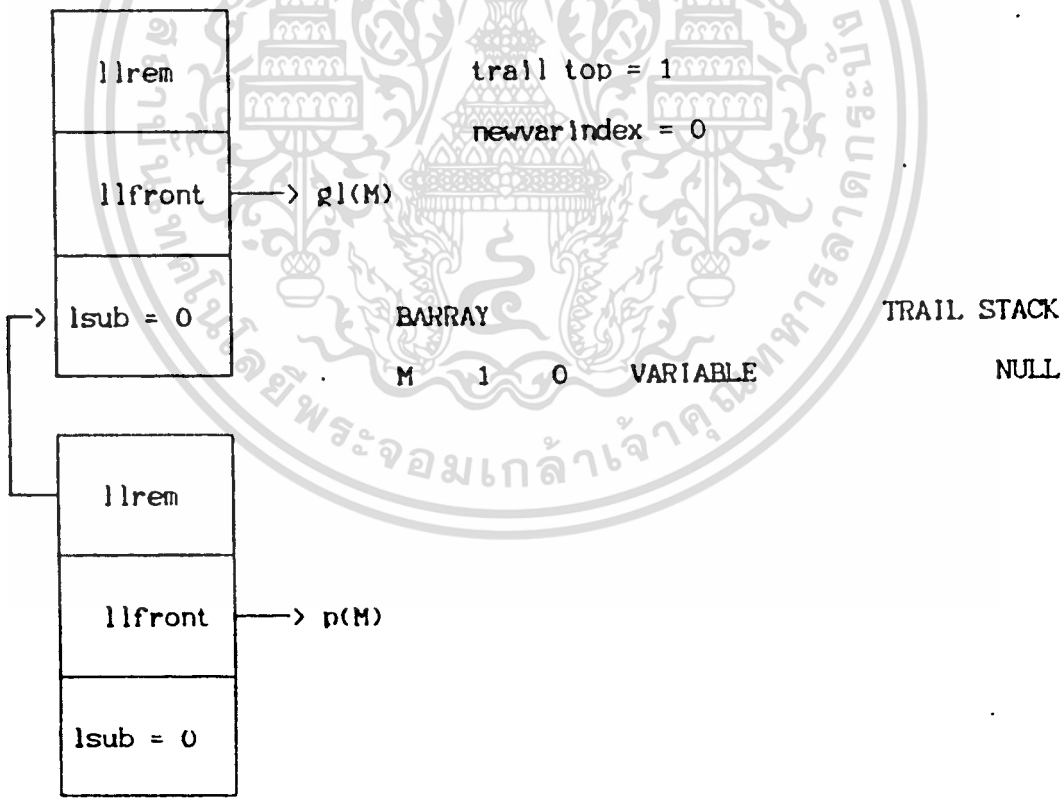
```
g1(M) :- p(M).
```

เริ่มต้นจะสร้าง LLIST ของโกล์ชั้นมาก่อนคือ



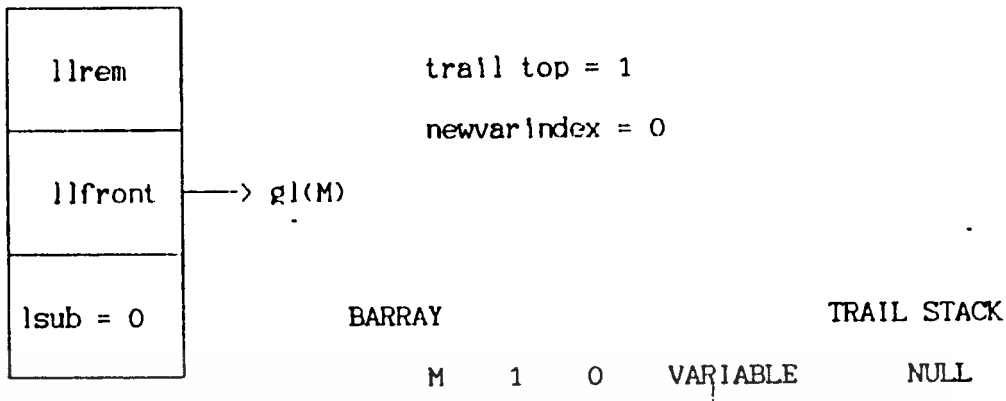
รูปที่ 4-13 แสดง LLIST ตอนเริ่มต้น

เมื่อพบว่า gl มีส่วนตัวจะสร้าง LLIST มาต่อกับ gl ได้เป็น



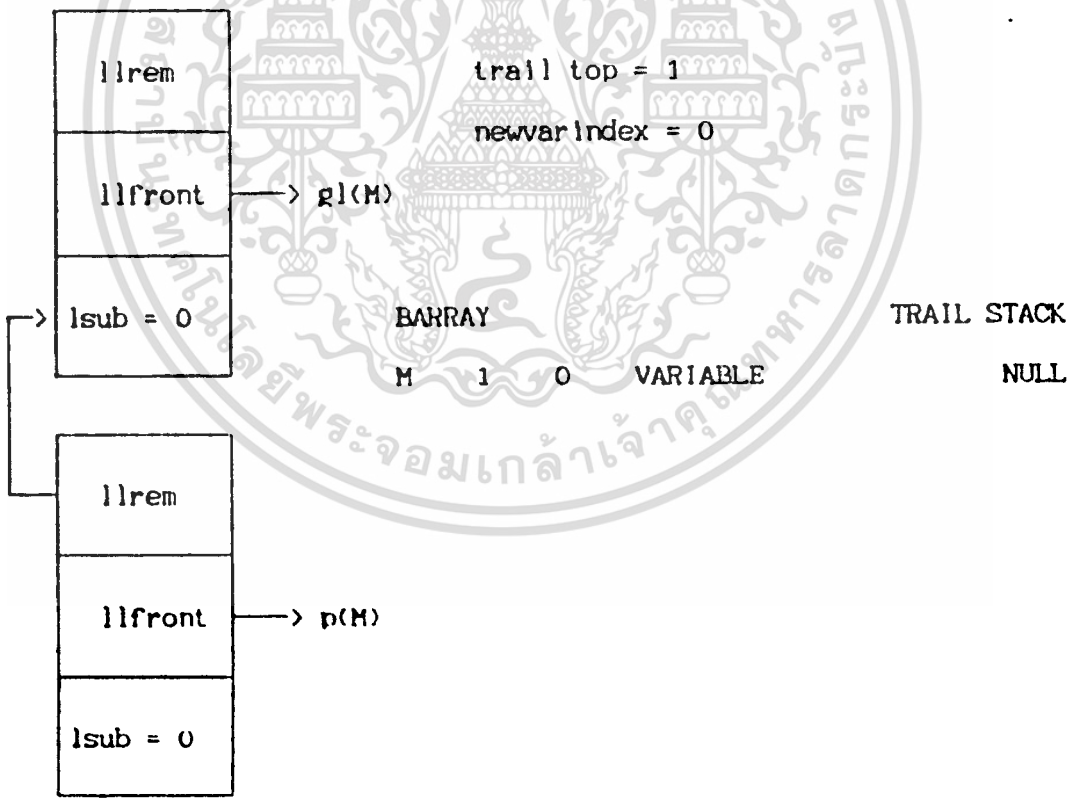
รูปที่ 4-14 แสดง LLIST หลังจากการสร้างโกล์ย่อย p(M)

เริ่มต้นจะสร้าง LLIST ของโกล์ชั้นมาก่อนคือ



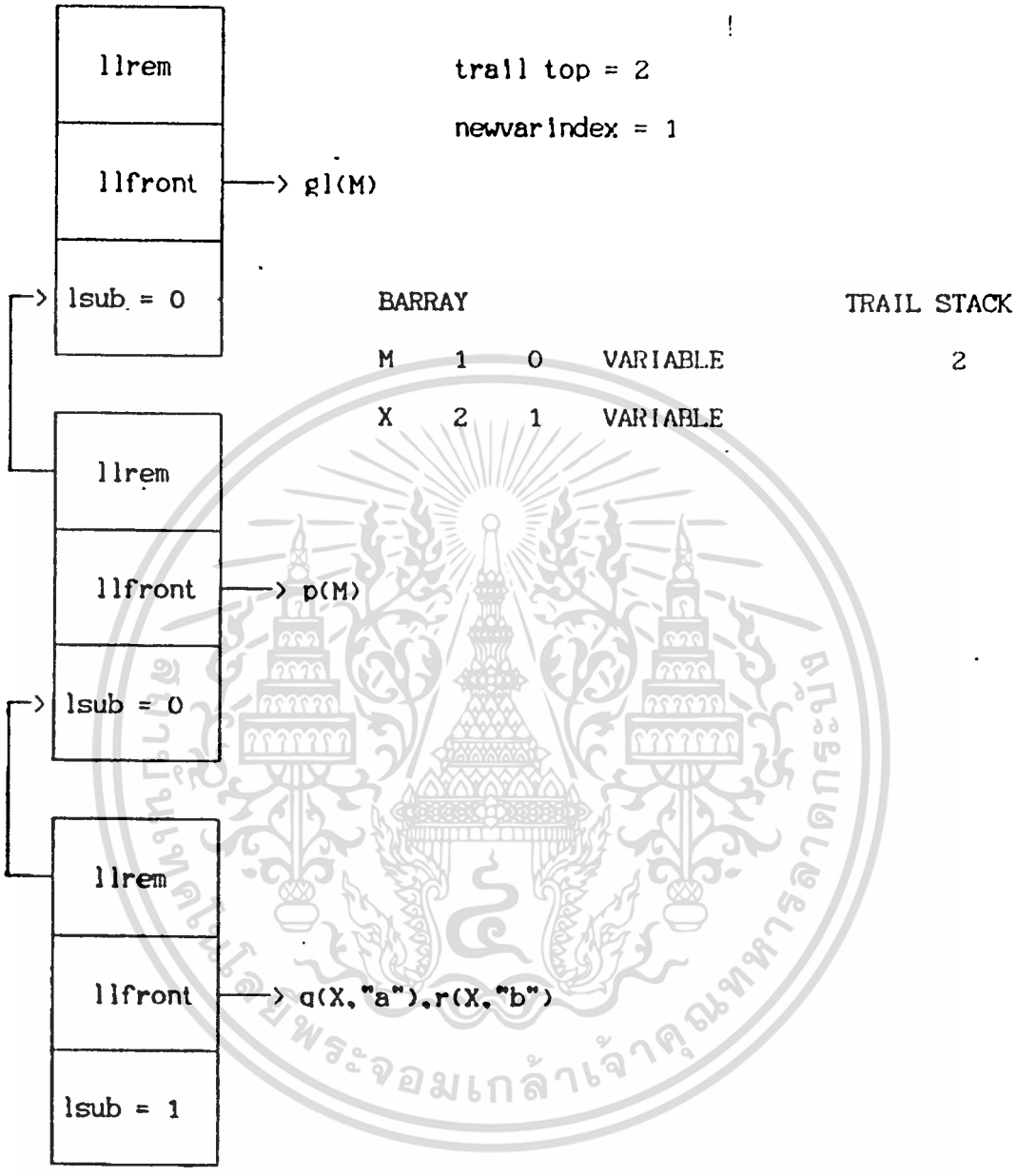
รูปที่ 4-13 แสดง LLIST ตอนเริ่มต้น

เมื่อพบว่า gl มีส่วนตัวจะสร้าง LLIST มาต่อกับ gl ได้เป็น



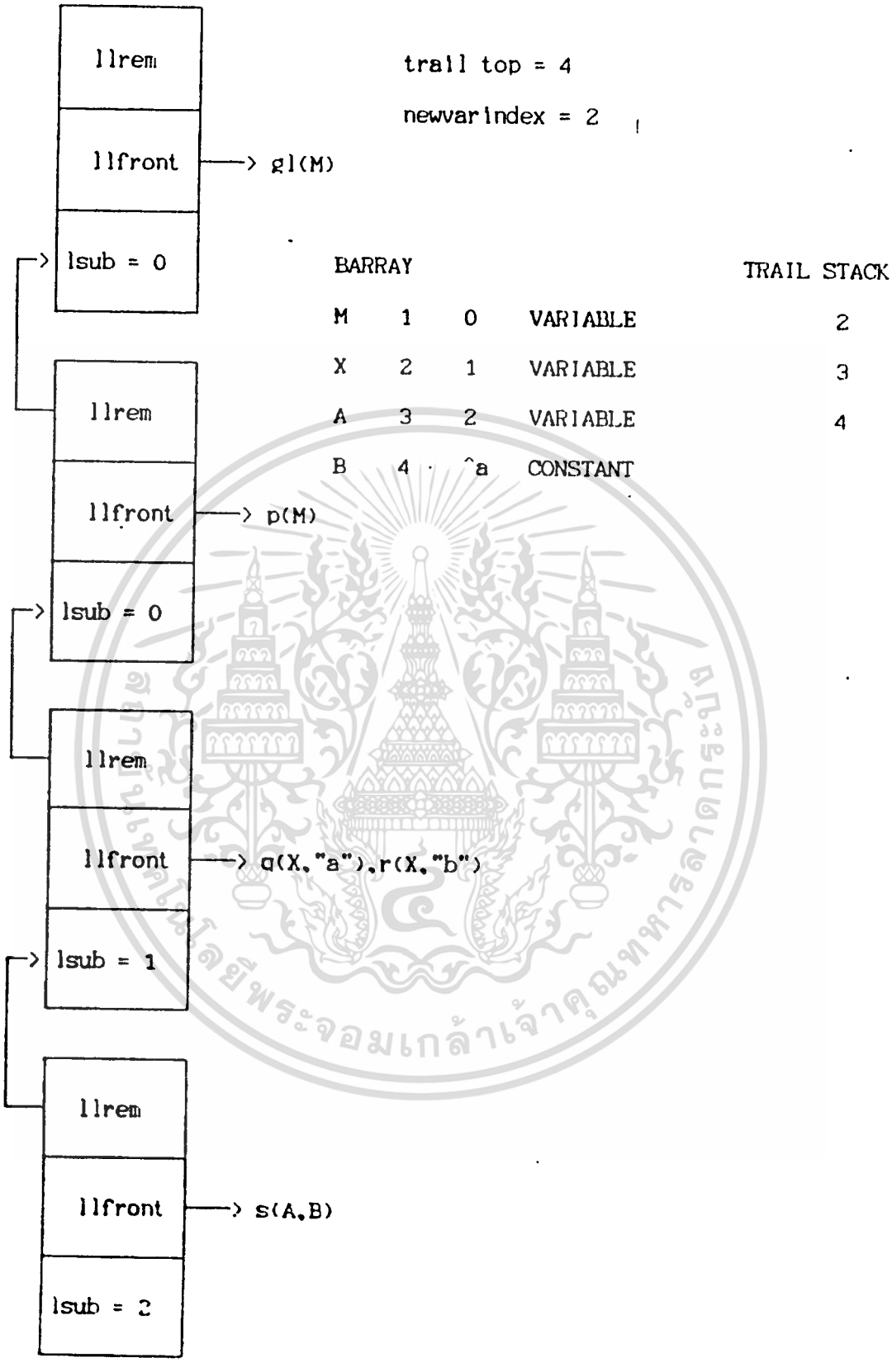
รูปที่ 4-14 แสดง LLIST หลังจากการสร้างโกล์ย่อย p(M)

p(M) เป็นกฎดังนั้นจะสร้างส่วนตัวของกฎมาต่อ และทำการ match ระหว่าง p(M) กับ p(X)



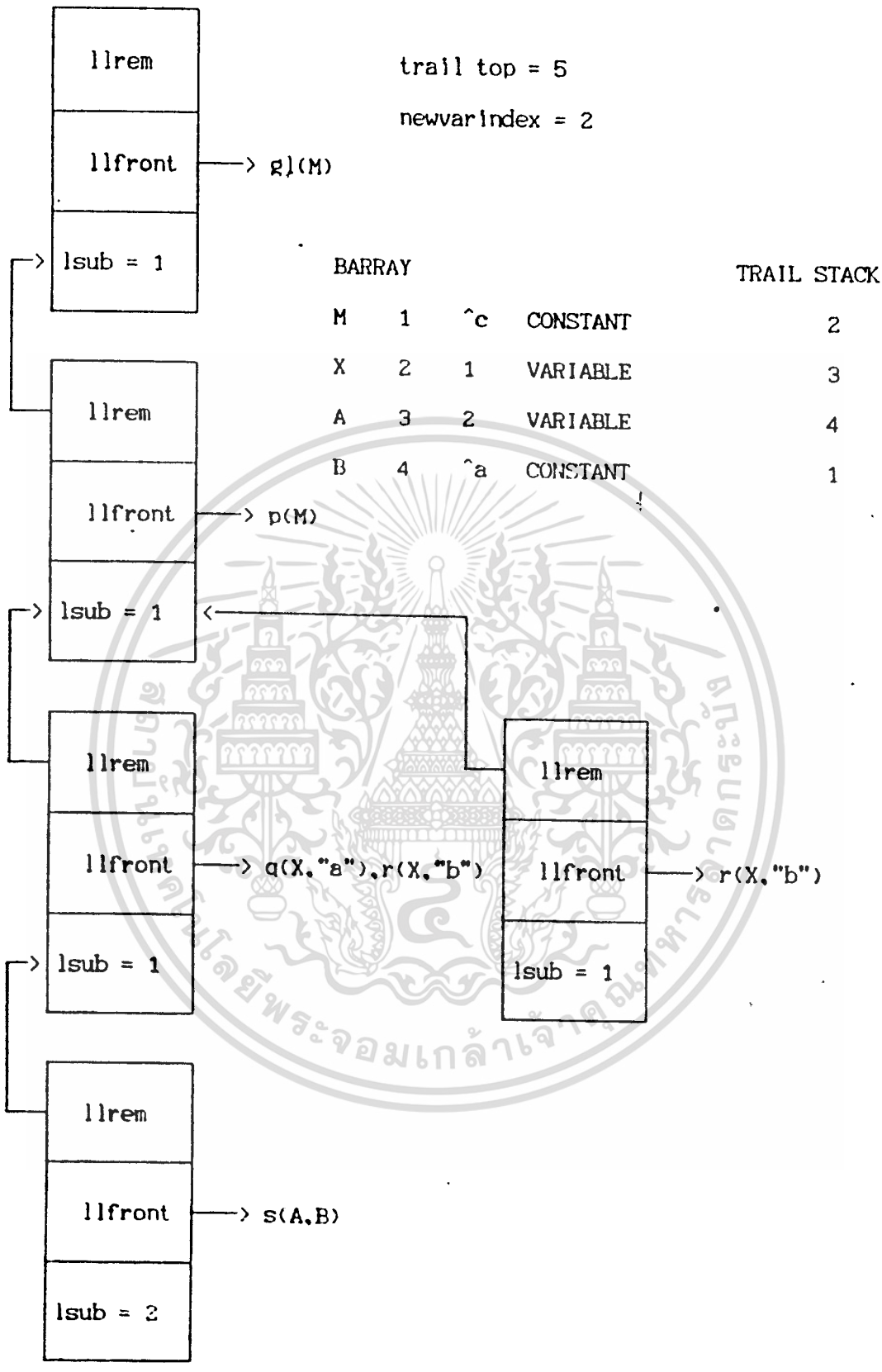
รูปที่ 4-15 แสดง LLIST หลังจากการสร้างส่วนตัวของ p(M)

ทำการ match ระหว่าง q(X,"a") กับ q(A,B) และสร้าง LLIST



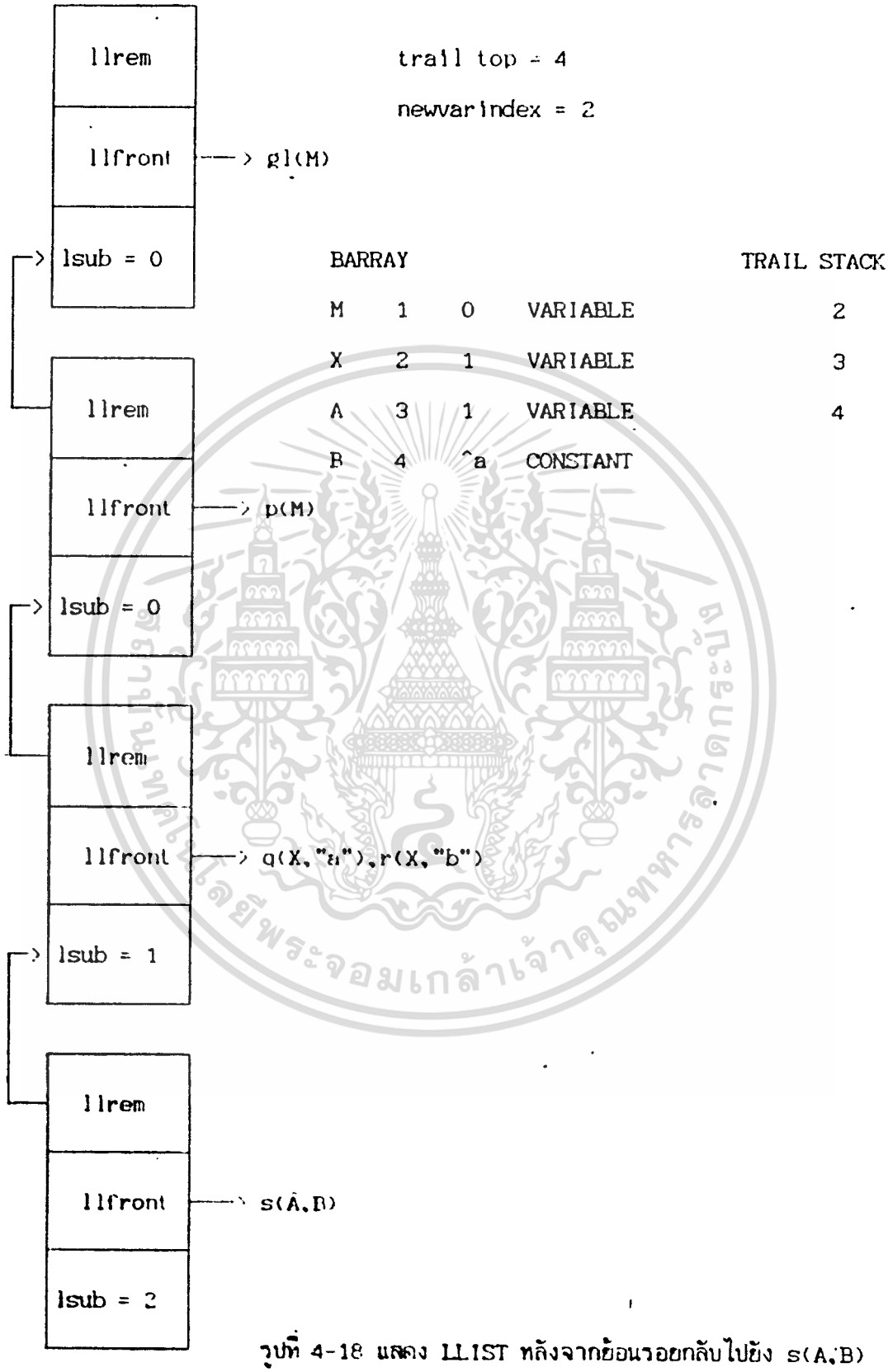
รูปที่ 4-16 แสดง LLIST หลังจากการสร้าง s(A,B)

ทำการ match ระหว่าง s(A,B) กับ s("c","a") และสร้างชั้นใหม่สำหรับ r(X,"b")



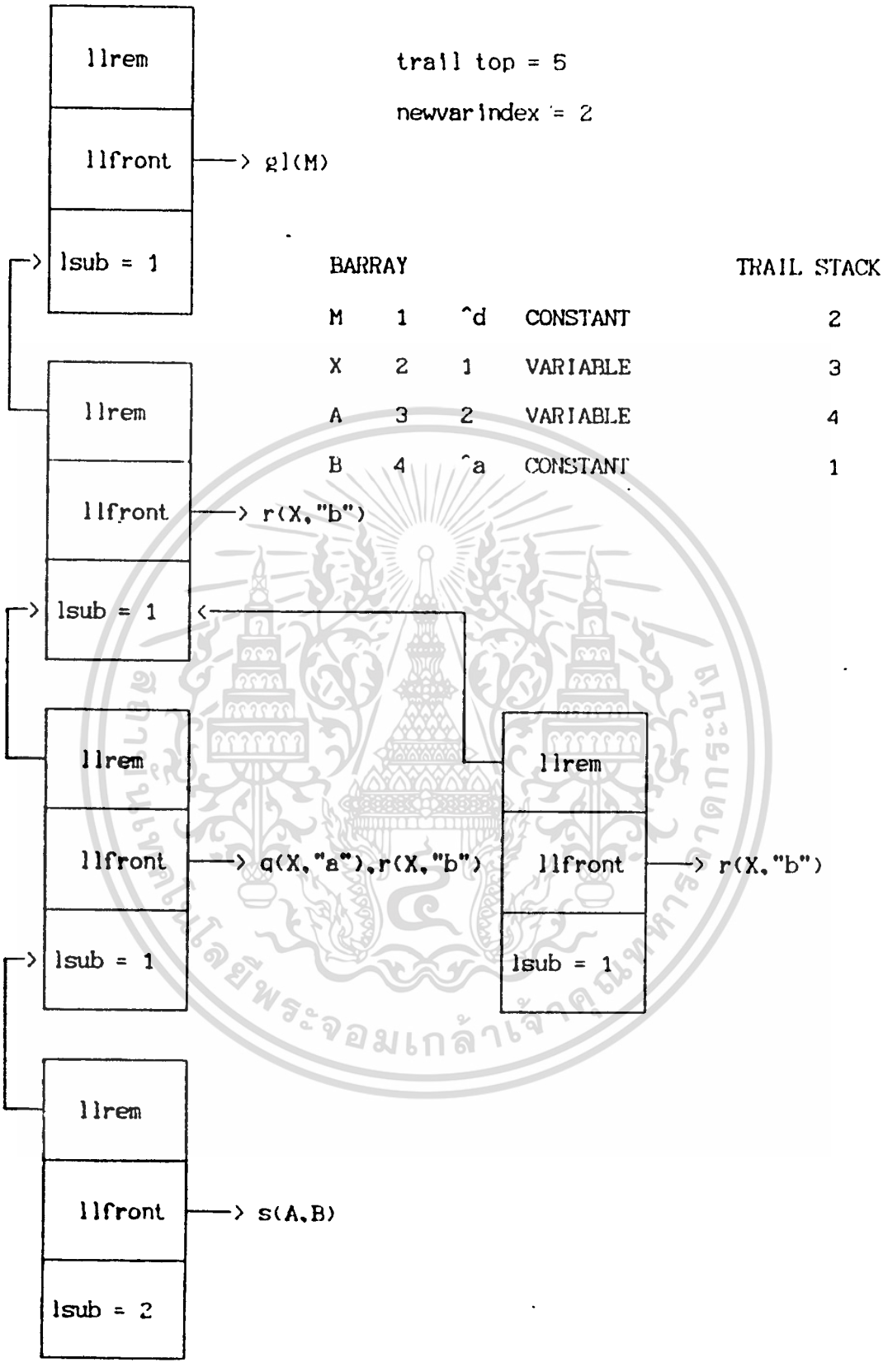
รูปที่ 4-17 แสดง LLIST หลังจากการสร้าง r(X,"b")

match r(X,"b") กับ r("d","b") ไม่สำเร็จก็จะทำการย้อนรอยกลับไปยัง s(A,B) และคืนค่าที่เปลี่ยนแปลงโดยดูจาก TRAIL STACK



รูปที่ 4-18 แสดง LLIST หลังจากย้อนรอยกลับไปยัง s(A,B)

match s(A,B) กับ s("d","a") แล้วสร้างชั้น r(X,"b") ขึ้นอีกครั้ง



รูปที่ 4-19 แสดง LLIST หลังจากการสร้าง r(X,"b") ขึ้นอีกครั้ง

ซึ่งบายด์ดิ้งอาเรย์ในชั้นนี้สามารถเปรียบเทียบระหว่าง r(X,"b") กับ r("d","b") สำเร็จ และไม่สามารถสร้างโกล์ย่อยเพิ่มได้ ก็จะจบโปรแกรม

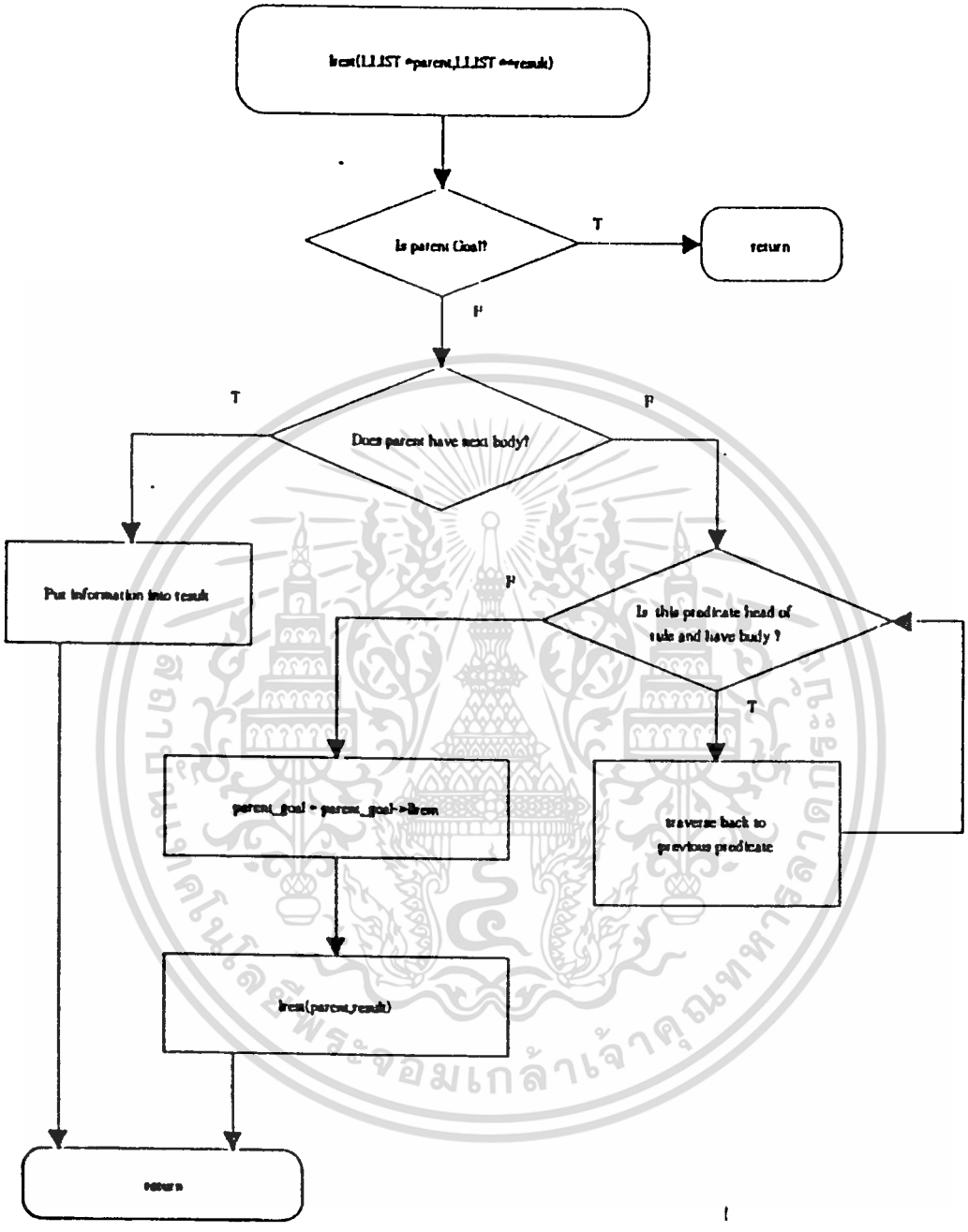
จากตัวอย่างที่ผ่านมา จะเห็นว่าการรีโซลูชั่นจะมีฟังก์ชันในการจัดการเกี่ยวกับ LLIST อยู่ 2 ฟังก์ชันด้วยกันคือ ฟังก์ชันที่ใช้ในการสร้างชั้นใหม่ของ LLIST ขึ้นมาเมื่อการเปรียบเทียบในชั้นเดิมสำเร็จ และฟังก์ชันในการคืนค่าต่างๆที่เกิดการเปลี่ยนแปลงไปในชั้นนี้กลับ ถ้าหากการเปรียบเทียบไม่สำเร็จ และย้อนกลับไปยังชั้นก่อนหน้านั้น ซึ่งก็คือการย้อนรอยนั่นเอง

ฟังก์ชันแรกมีชื่อว่า lconcat_rest รับอาร์กิวเมนต์เป็น LLIST ของชั้นปัจจุบันที่ผ่านการเปรียบเทียบสำเร็จไปแล้ว ค่าที่บอกให้ทราบว่าเพรดิเคตนี้มีส่วนตัวหรือไม่ และค่าที่บอกตำแหน่งของตัวแปรในบายด์ดิ้งอาเรย์ ถ้าไม่มีส่วนตัวฟังก์ชันนี้จะเรียกฟังก์ชัน lrest ซึ่งเป็นฟังก์ชันแบบเรียกตัวเอง เพื่อสร้าง LLIST ของส่วนตัวตัวถัดไป และถ้าไม่มีส่วนตัวแล้วก็จะสร้างโกล์ย่อยตัวใหม่ขึ้นมา

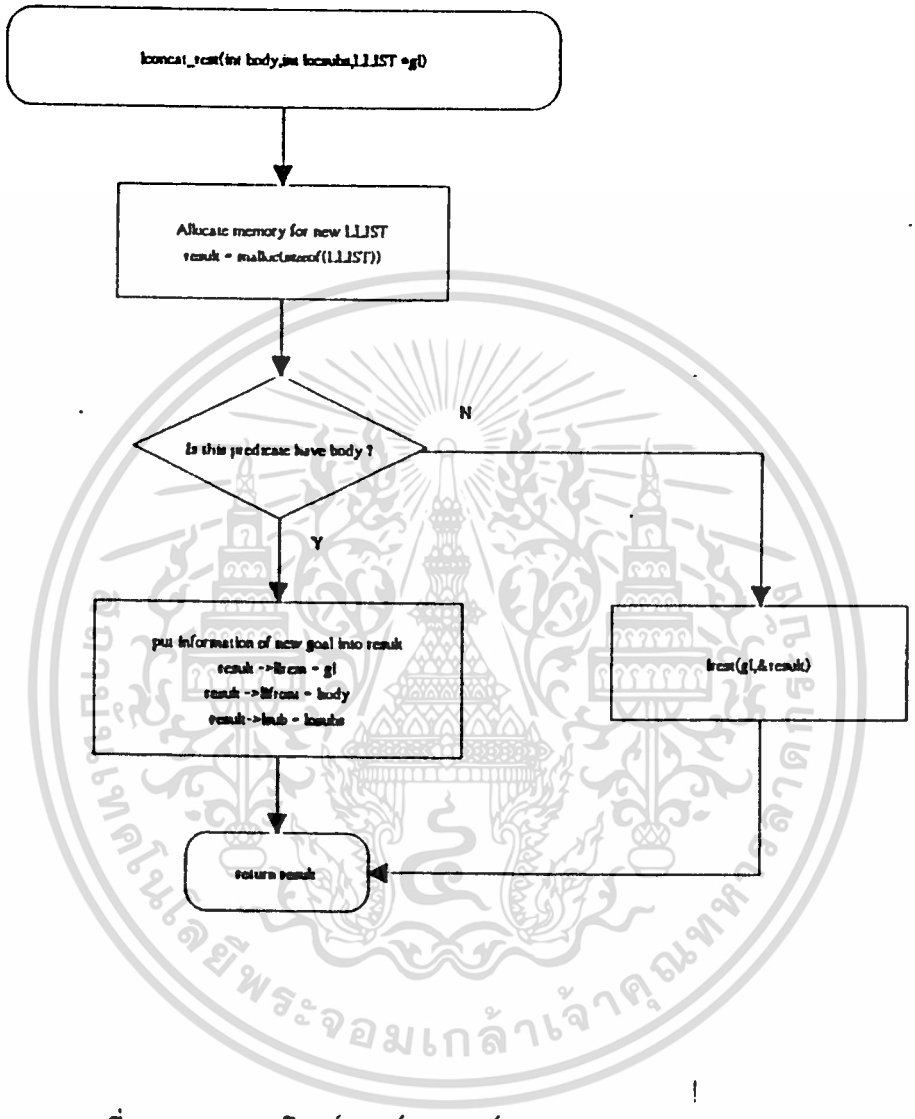
ฟังก์ชันที่สองคือฟังก์ชัน restore ทำหน้าที่คืนค่าต่างๆในบายด์ดิ้งอาเรย์ที่เปลี่ยนแปลง รับอาร์กิวเมนต์ เป็นอินเดกซ์ของบายด์ดิ้งอาเรย์และ TRAIL STACK ของชั้นเดิม

ฟังก์ชันสุดท้ายที่มีหน้าที่ในการจัดการทั้งระบบเข้าด้วยกัน เป็นฟังก์ชันแบบเรียกตัวเอง รับอาร์กิวเมนต์เป็นโกล์ย่อยที่สร้างขึ้นมาใหม่จากโกล์ย่อยเดิม ที่เปรียบเทียบได้สำเร็จ ฟังก์ชันนี้มีชื่อว่า establish

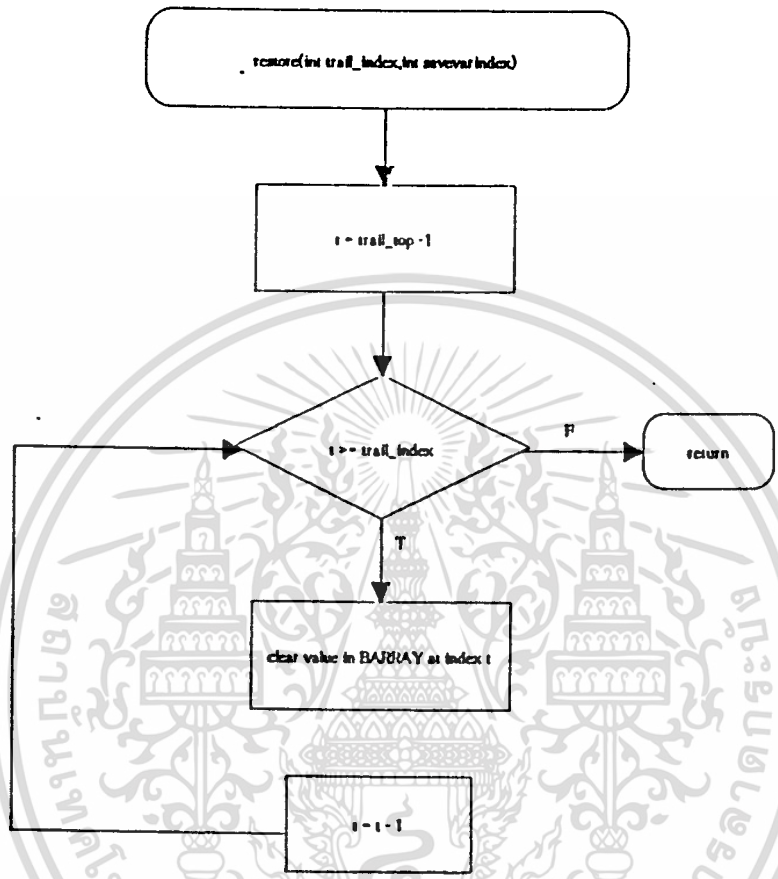
รายละเอียดการทำงานของฟังก์ชันทั้งหมดแสดงดังโฟลว์ชาร์ตดังนี้



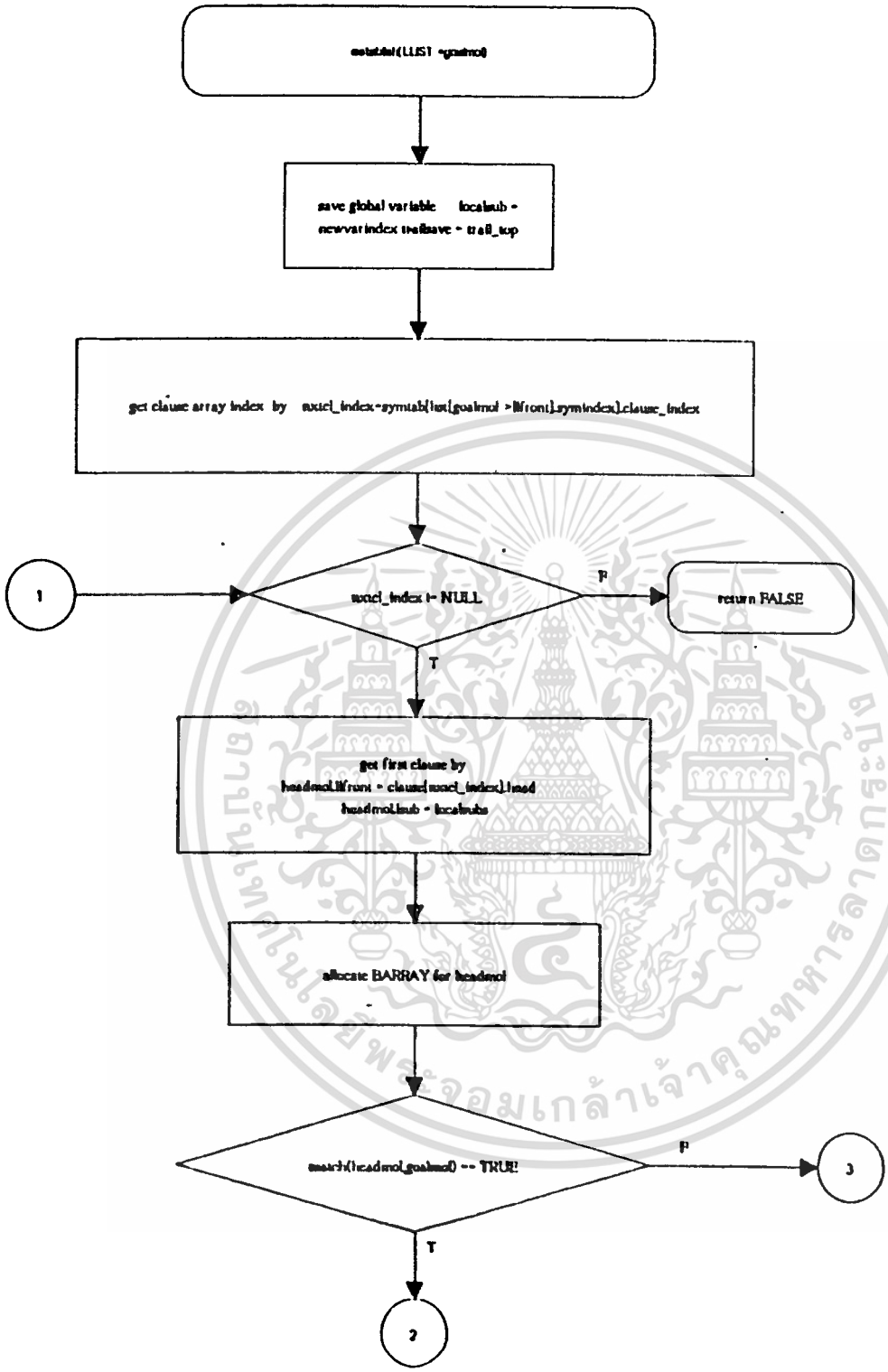
รูปที่ 4-20 แสดงโฟลว์ชาวด์ของฟังก์ชัน lrest



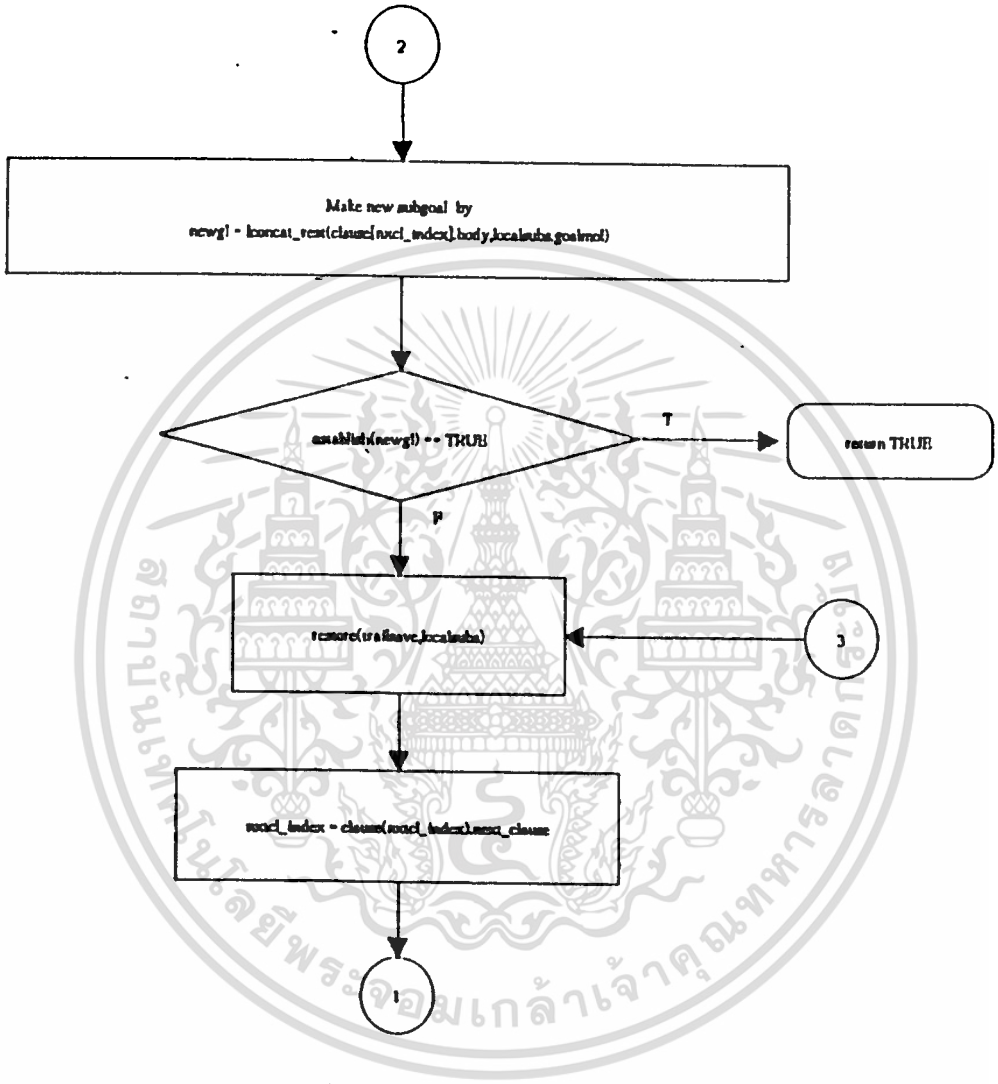
รูปที่ 4-21 แสดงโฟลว์ชาวด์ของฟังก์ชัน lconcat_rest



รูปที่ 4-22 แสดงโฟลว์ชาวด์ของฟังก์ชัน restore



รูปที่ 4-23 แสดงโฟลว์ชาวด์ของฟังก์ชัน establish



รูปที่ 4-23(ต่อ) แสดงโฟลว์ชาวด์ของฟังก์ชัน establish

บทที่ 5

การเชื่อมประสานระหว่างภาษาโปรลอกไลค์กับฐานความรู้

ความสามารถที่จัดว่าเป็นจุดเด่นของโครงงานนี้ก็คือ ความสามารถในการเชื่อมประสานระหว่างโปรลอกไลค์กับฐานความรู้ โดยที่โปรแกรมโปรลอกสามารถดึงเอาข้อเท็จจริงและข้อเท็จจริงที่ได้จากการอนุมานที่เก็บอยู่ในฐานความรู้มาใช้ได้ ซึ่งวิธีในการอนุมานข้อเท็จจริงได้กล่าวรายละเอียดไปแล้วในบทที่ 2 ในบทนี้จะเป็นการกล่าวถึงรายละเอียดในการเชื่อมประสานระหว่างภาษาโปรลอกไลค์กับฐานความรู้

5.1 ลักษณะของโปรแกรมโปรลอกในการเชื่อมโยงกับฐานความรู้

ในหัวข้อนี้จะเป็นการอธิบายวิธีการในการเขียนโปรแกรมโปรลอกไลค์ที่มีการเชื่อมโยงกับฐานความรู้ ซึ่งสามารถเชื่อมโยงได้ 2 แบบด้วยกันคือ

ในแบบแรกเป็นการเชื่อมโยงกับข้อเท็จจริงที่เก็บไว้ในฐานความรู้โดยตรง ทำได้โดยการประกาศชื่อเลเบิลของข้อเท็จจริงไว้ในโปรแกรมในส่วนของ fact_predicates ส่วนรายละเอียดต่างๆของข้อเท็จจริงนั้น คอมไพล์เลอร์จะทำการตรวจสอบเองว่าข้อเท็จจริงนั้นมีอาร์กิวเมนต์นั้นอยู่กี่ตัว แต่ละตัวมีชนิดอะไร เพราะเราถือว่าสิ่งต่างๆเหล่านี้ถูกสร้างขึ้นมาเก็บไว้ในฐานความรู้อยู่แล้ว ดังนั้นไม่มีความจำเป็นที่จะต้องบอกซ้ำอีก หลังจากการประกาศว่าในโปรแกรมนี้เราใช้ข้อเท็จจริงอะไรบ้างไปแล้ว ต่อไปเราจะมองข้อเท็จจริงเหล่านี้มีลักษณะเช่นเดียวกับเพรดิเคตปกติที่อยู่ในโปรลอก สามารถนำมาสร้างเป็นส่วนประกอบของส่วนตัวของกฎได้เลย โดยที่ผู้เขียนโปรแกรมไม่ต้องทราบเลยว่าการจัดการติดต่อกับฐานความรู้อย่างไร ปล่อยให้ระบบเป็นผู้จัดการทั้งหมด สามารถทำการเขียนโปรแกรมดังเช่นภาษาโปรลอกทั่วไป

ตัวอย่างโปรแกรมภาษาโปรลอกไลค์ต่อไปนี้เป็นการแสดงถึงการเขียนโปรแกรมที่ใช้ข้อเท็จจริงที่เก็บอยู่ในฝั่งฐานความรู้มาทำการทางฝั่งโปรลอก ในคำถามที่ว่าใครเป็นบรรพบุรุษของนายแดง โดยในฐานความรู้เก็บข้อเท็จจริงไว้ว่าใครเป็นบุพการีของใครบ้าง ส่วนกฎที่ใช้อนุมานนั้นสร้างไว้ในฝั่งโปรลอก แล้วดึงข้อเท็จจริงจากฐานความรู้มาทำการยูนิฟิเคชั่นจนได้รับคำตอบ

```
domains
        people = string
fact_predicates
        parent
predicates
        anc(people,people)
        print(people)
        gl(poeple)
clauses
        anc(A,B) :- parent(A,B).
        anc(A,B) :- parent(A,C),anc(C,B).
        print(X) :- write(X),nl.
goal
        gl(A) :- anc(A,"dang"),print(A),fail.
```

โปรแกรมนี้ทำการประกาศข้อเท็จจริง parent ไว้ในส่วนไว้ในส่วนของ fact_predicates และกฎต่างๆอยู่ในส่วนของ clauses ของโปรแกรม ถ้าเราให้ข้อเท็จจริงที่อยู่ในฐานความรู้เป็นดังนี้

PARENT

FATHER	CHILD
dum	dang
superman	dum
adum	superman
green	black

ตารางที่ 5-1 แสดงข้อเท็จจริงของ parent(FATHER,CHILD) ในฐานความรู้

เมื่อนำโปรแกรมนี้มาทำการคอมไพล์ด้วยคอมไพเลอร์ที่สร้างขึ้นมา จะได้ผลลัพธ์เป็นภาษาซี นำมาคอมไพล์อีกครั้งด้วยคอมไพเลอร์ภาษาซี แล้วนำมาลิงค์กับไลบรารีที่สร้างขึ้นมา นำโปรแกรมที่ได้มารันจะได้ผลเป็น

```
dum
superman
adum
```

การเชื่อมประสานอีกแบบหนึ่งเป็นการเรียกใช้กฎที่เก็บไว้ในฐานความรู้ ซึ่งกฎต่างๆเราจะต้องทำการสร้างขึ้นไว้ก่อน โดยวิธีการในการเก็บกฎในฐานความรู้จะกล่าวในหัวข้อต่อไป เมื่อโปรลอกนำกฎต่างๆที่เก็บไว้ในฐานความรู้มาใช้ โปรลอกจะมองกฎเหล่านี้เป็นเหมือนข้อเท็จจริงทั่วไปดังเช่นในแบบแรก ซึ่งเราจะต้องประกาศเลเบิลของกฎไว้ในส่วนของ fact_predicate เช่นกัน ในขณะทำการรันโปรแกรม เมื่อพบเพรดิเคตที่มีชนิดเป็นข้อเท็จจริงที่อยู่ในฐานความรู้ โปรลอกก็จะทำการติดต่อกับฐานความรู้เพื่อนำกฎนั้นไปทำการอนุมานข้อเท็จจริงออกมา เมื่อได้ข้อเท็จจริงนั้นแล้วโปรลอกก็จะทำดิงข้อเท็จจริงนั้นมาทำการเช่นเดียวกับเพรดิเคตปกติต่อไป จากโปรแกรมในตัวอย่างที่ผ่านมาเราจะเปลี่ยนเป็นโปรแกรมใหม่ที่ให้กฎว่าใครเป็นบรรพบุรุษของใครเก็บไว้อยู่ในฝั่งฐานความรู้ ได้โปรแกรมใหม่เป็น

```
domains
    people = string
fact_predicates
    anc
predicates
    gl(people)
    print(people)
clauses
    print(X) :- write(X),nl.
goal
    gl(A) :- anc(A,"dang"),print(A),fail.
```

โดยที่กฎที่เก็บในฐานความรู้คือ

```
anc(A,B) :- parent(A,B).
anc(A,B) :- parent(A,C),anc(C,B).
```

จากโปรแกรมนี้ถ้าข้อเท็จจริงว่าใครเป็นบุพการีของใครมีลักษณะเหมือนกับโปรแกรมก็จะได้ผลลัพธ์ออกมาเหมือนกัน จะเห็นว่าในโปรแกรมเรามอง anc เป็นข้อเท็จจริงปกติ การทำงานต่างๆ ระบบจะเป็นผู้จัดการเองทั้งหมด

การทำงานดังกล่าว โปรแกรมจะเชื่อมประสานกับฐานความรู้ครั้งเดียวเท่านั้น ครั้งแรกที่มีการเรียกใช้ข้อเท็จจริงหรือกฎที่เก็บไว้ในฐานความรู้ จากนั้นก็จะนำข้อเท็จจริงที่ฐานความรู้สร้างขึ้นมาเก็บไว้ในหน่วยความจำ ดังนั้นถ้าเกิดการย้อนรอยก็จะนำข้อมูลที่เก็บไว้ในหน่วยความจำมาใช้เลย ไม่ต้องติดต่อกับฐานความรู้อีก ทำให้มีการทำงานที่รวดเร็วขึ้นมาก ซึ่งการทำงานในหลักการดังกล่าวจัดเป็นจุดเด่นของโครงงานนี้อีกประการหนึ่ง

5.2 <u>รายละเอียดของตารางที่ใช้เก็บรายละเอียดของกฎและข้อเท็จจริงในฐานความรู้</u>

ในระบบของฐานความรู้ซึ่งมีความสามารถในการจัดการกฎได้ ดังนั้นจึงต้องมีกรรมวิธีในการที่จะเก็บกฎต่างๆไว้ในฐานความรู้ ในการทำงานจริงนั้นเราใช้ระบบฐานข้อมูลแบบสัมพันธ์เป็นตัวช่วยในการทำงาน การออกแบบเราจึงเก็บให้อยู่ในรูปตาราง ซึ่งมีอยู่ 3 ตารางด้วยกัน ซึ่งรายละเอียดและความสัมพันธ์ของตารางต่างๆเป็นดังนี้

ตารางแรกคือตาราง view_column เป็นตารางที่ใช้บอกรายละเอียดของแต่ละเพรดิเคตที่อยู่ในฐานความรู้ ประกอบด้วย

view_name	บอกชื่อของเพรดิเคต
column_name	บอกชื่อของอาร์กิวเมนต์ที่อยู่ในเพรดิเคตนั้น
column_type	บอกชนิดของอาร์กิวเมนต์นั้น
column_order	บอกลำดับของอาร์กิวเมนต์ตัวนั้น
column_width	บอกขนาดความกว้างของอักขระกรณีอาร์กิวเมนต์มีชนิดเป็นอักขระ
pri_key_id	เครื่องหมายบอกอาร์กิวเมนต์ตัวนี้เป็นพลายมารีคีย์หรือไม่
deducible_id	บอกชนิดของเพรดิเคตว่าเป็น ข้อเท็จจริง,กฎแบบเรียกตัวเอง หรือกฎแบบไม่เรียกตัวเอง ใช้สัญลักษณ์ F,R,N ตามลำดับ

จากโปรแกรมที่ผ่านมาเพคดิเคตที่เก็บไว้ในฐานความรู้ประกอบด้วยเพรดิเคต

parent(FATHER CHAR(30),CHILD CHAR(30))

anc(FATHER CHAR(30),CHILD CHAR(30))

จะเก็บในตาราง view_column ดังนี้

view_column

v_name	c_name	c_type	c_order	c_width	p_id	d_id
anc	FATHER	CHAR	1	30	N	R
anc	CHILD	CHAR	2	30	N	R
par	FATHER	CHAR	1	30	N	N
par	CHILD	CHAR	2	30	N	N

ตาราง 6-2 แสดงการเก็บข้อมูลในตาราง view_column

ตารางต่อไปคือตาราง predicate และตาราง rule ทั้งสองตารางเป็นตารางที่ใช้ร่วมกันในการเก็บกฎต่างๆที่มีอยู่ในฐานความรู้ รายละเอียดของตารางทั้งสองมีดังนี้

ตาราง predicate และ rule มีคีย์ที่ใช้เชื่อมกันคือ

head_number เป็นตัวเลขบอกลำดับของกฎในโปรแกรม

body_number เป็นตัวเลขบอกลำดับของส่วนตัวของกฎที่อยู่ในแต่ละกฎ

ส่วนคอลัมภ์ที่มีอยู่ในตาราง predicate คือ

argument บอกชื่อตัวแปรที่อยู่ในเพรดิเคต

arg_order บอกลำดับของอาร์กิวเมนต์ที่อยู่ในเพรดิเคต

คอลัมภ์ที่อยู่ในตาราง rule คือ

head บอกชื่อเพรดิเคตที่เป็นส่วนหัวของกฎ

body บอกชื่อเพรดิเคตที่เป็นส่วนตัวของกฎ

ในตัวอย่างที่ผ่านมาสามารถนำมาสร้างเป็นข้อมูลในตารางทั้ง 2 ดังนี้

rule

HEAD	BODY	HEAD_NUMBER	BODY_NUMBER
anc	parent	1	1
anc	parent	2	1
anc	anc	2	2

ตาราง 5-3 แสดงการเก็บข้อมูลในตาราง rule

predicate

ARGUMENT	ARG_ORDER	HEAD_NUMBER	BODY_NUMBER
A	1	1	0
B	2	1	0
A	1	1	1
B	2	1	1
A	1	2	0
C	2	2	0
A	1	2	1
B	2	2	1
B	1	2	2
C	2	2	2

ตาราง 5-4 แสดงการเก็บข้อมูลในตาราง predicate

บทที่ 6

บทสรุปและวิจารณ์

โครงงานสภาพแวดล้อมในการเขียนโปรแกรมแบบอนุมานกฎได้ เป็นโครงงานที่รวมเอาทฤษฎีทางด้านปัญญาประดิษฐ์และระบบฐานข้อมูลแบบสัมพันธ์มาใช้ เพื่อทำการออกแบบภาษาที่มีความสามารถในการนำข้อมูลที่เก็บอยู่ในฐานข้อมูลแบบสัมพันธ์มาทำการอนุมานตามหลักการทางด้านปัญญาประดิษฐ์ ซึ่งโครงงานนี้ได้ออกแบบตัวภาษาให้มีลักษณะการทำงานเหมือนกับภาษาโปรลอก ซึ่งเป็นภาษาที่นิยมใช้กันในงานทางด้านปัญญาประดิษฐ์อย่างแพร่หลายในปัจจุบัน และเรียกภาษานี้ว่าโปรลอกไลค์

เนื่องจากข้อจำกัดทางด้านระยะเวลา ทำให้ไม่สามารถที่จะดึงเอาคุณสมบัติของภาษาโปรลอกมาได้ทั้งหมด จึงดึงเอาเฉพาะส่วนที่เป็นจุดเด่นของภาษามาเท่านั้น ซึ่งก็อยู่ในระดับที่สนับสนุนหลักการพื้นฐานต่างๆได้ครบถ้วน อาทิเช่นสามารถจัดการกฎแบบเรียกตนเองได้ มีเพรดิเคดคัดและเฟล ซึ่งเป็นเพรดิเคดแบบฟังก์ชันในตัวที่มีความสำคัญอย่างมากในการเขียนโปรแกรมโปรลอก นอกจากนี้ยังมีฟังก์ชันในตัวในการรับและอ่านค่าเพื่อใช้ในการติดต่อกับผู้ใช้

การพัฒนาโปรแกรมในโครงงานนี้ จัดว่าเป็นงานที่อยู่ในระดับที่ค่อนข้างยาก เนื่องจากภาษาโปรลอกเป็นภาษาในยุคที่ 5 ซึ่งจัดว่าเป็นภาษาที่ง่ายในการใช้งาน แต่โครงสร้างการทำงานจริงที่อยู่ในระดับที่ต่ำลงมามีความซับซ้อนอย่างมาก ความยุ่งยากในการออกแบบโครงสร้างข้อมูลที่สามารถให้รายละเอียดได้ครบถ้วน ซึ่งบางครั้งกว่าจะรู้ว่าโครงสร้างนี้ไม่สามารถให้ข้อมูลที่ต้องการได้ก็ต้องอยู่ในช่วงของการเขียนโปรแกรมแล้ว จึงกลับมาทำการแก้ไขโครงสร้างข้อมูลอีกครั้ง นอกจากนี้การออกแบบขั้นตอนวิธีการทำงานของโปรแกรมจะต้องถูกต้องตามหลักการของภาษาโปรลอกทุกประการ ดังนั้นก่อนทำการออกแบบจะต้องทำการศึกษาหลักการของภาษาโปรลอกให้ละเอียด

ส่วนระบบฐานความรู้ของโครงงานนี้ได้นำเอาระบบจัดการฐานข้อมูลแบบสัมพันธ์ของออราเคิลมาใช้ ซึ่งจัดเป็นโปรแกรมฐานข้อมูลที่มีผู้นิยมใช้อย่างแพร่หลาย แต่มีปัญหาอยู่ที่มีกระบวนการในการติดต่อที่ล่าช้า ซึ่งในอนาคตถ้าหากมีระบบฐานข้อมูลที่มีประสิทธิภาพเพิ่มขึ้นปัญหาดังกล่าวก็จะลดลง

ในการออกแบบคอมไพเลอร์ที่ใช้ในการคอมไพล์โปรแกรม ได้มีการใช้โปรแกรม LEX กับ YACC ซึ่งเป็นโปรแกรมที่อยู่บนระบบปฏิบัติการยูนิกซ์ ทำให้การออกแบบต้องมีความยุ่งยาก เพราะจะต้องออกแบบบนระบบยูนิกซ์แล้วจึงทำการย้ายมาบนระบบปฏิบัติการ DOS อีกครั้งหนึ่ง

และยังประสบปัญหาเรื่องการที่เครื่องคอมพิวเตอร์ที่ใช้ในระบบปฏิบัติการยูนิกซ์มักจะเสียอยู่เป็นประจำ ทำให้เกิดความล่าช้าในการปฏิบัติงาน

การทำโครงงานในครั้งนี้เป็นเรื่องที่เน้นหนักไปทางด้านทฤษฎีค่อนข้างมาก ดังนั้นในขั้นตอนการศึกษาหาข้อมูลนั้นต้องใช้ระยะเวลานานพอสมควร และต้องพยายามดึงเนื้อหาต่างๆที่เกี่ยวข้องกับชิ้นงานมา เพราะโครงงานนี้จัดเป็นโครงงานที่เข้าใจถึงจุดประสงค์ในการทำงานได้ยาก ดังนั้นถ้าผู้อ่านที่มีความสนใจการทำงานของโครงงานนี้ควรจะศึกษาถึงทฤษฎีต่างๆที่เกี่ยวข้องให้ละเอียดก็จะช่วยให้เข้าใจจุดประสงค์ของโครงงานชิ้นนี้ได้ดีขึ้น

บทที่ 7

แนวทางการพัฒนาต่อ

ในการพัฒนาโปรแกรมของโครงงานนี้นับว่าอยู่ในระดับเป็นที่น่าพอใจของผู้จัดทำ เพราะสามารถออกแบบตัวภาษาได้มีคุณสมบัติที่ดีพอสมควร ปัญหาทางด้านเทคนิคส่วนใหญ่ได้รับการแก้ไขครบถ้วนแล้ว ดังนั้นแนวทางการพัฒนาส่วนใหญ่จะเป็นการพัฒนาคุณสมบัติปลีกย่อยของภาษาให้มีความสามารถได้ดีขึ้น และความเร็วในการทำงานของระบบ ซึ่งประกอบด้วยหัวข้อต่างๆต่อไปนี้ คือ

1. พัฒนาให้มีความสามารถในการทำงานของฟังก์เตอร์ ซึ่งโครงสร้างข้อมูลที่ได้ออกแบบมานี้เป็นโครงสร้างที่สามารถสร้างฟังก์เตอร์ได้อยู่แล้ว ดังนั้นการพัฒนาให้นั้นเพียงแต่ทำการปรับปรุงในส่วนของขั้นตอนวิธีเท่านั้น

2. พัฒนาฟังก์ชันในตัวต่างๆ เพื่อให้สามารถนำไปใช้งานได้จริง เพราะการเขียนโปรแกรมจำเป็นที่จะต้องอาศัยฟังก์ชันในตัวต่างๆอยู่เป็นจำนวนมาก อาทิเช่นฟังก์ชันทางด้านการคำนวณ การเปรียบเทียบ และการเชื่อมประสานระหว่างโปรแกรมกับผู้ใช้ ซึ่งเราได้สร้างไว้บ้างแล้ว แต่ก็ไม่มากพอที่จะนำไปใช้จริงได้อย่างสะดวกนัก จึงควรพัฒนาฟังก์ชันในตัวให้มีมากขึ้นไม่ว่าจะเป็นฟังก์ชันที่ใช้ในการคำนวณ เช่น การยกกำลัง , การหาราก เป็นต้น หรือฟังก์ชันในการติดต่อกับผู้ใช้ เช่น ลบหน้าจอ , สร้างหน้าต่าง

3. เปลี่ยนแปลงโครงสร้างข้อมูลในส่วนของฐานความรู้ให้อยู่ในรูปของตารางทั้งหมด เนื่องจากเป็นการทำงานอยู่บนระบบฐานข้อมูลสัมพันธ์ ดังนั้นการใช้โครงสร้างข้อมูลแบบลิงค์ลิสท์จัดว่าเป็นการใช้งานที่ไม่สอดคล้องกับแนวคิดของระบบนี้ แต่เนื่องจากอุปสรรคด้านความเร็วทำให้จำเป็นต้องใช้ ถ้าหากในอนาคตระบบฐานข้อมูลมีประสิทธิภาพดีขึ้นก็ควรจะเปลี่ยนไปใช้ตารางทั้งหมด

4. พัฒนาระบบในการจัดการของฐานความรู้ให้ดีขึ้น เนื่องจากข้อจำกัดในด้านเวลาทำให้ไม่สามารถรวมระบบการจัดการกฎแบบไม่เรียกตัวเองที่กลุ่มโครงงานการพัฒนากฎในระบบฐานความรู้จัดทำขึ้นได้ ในโครงงานนี้จึงทำได้เฉพาะในส่วนของระบบการจัดการกฎแบบเรียกตัวเองได้เท่านั้น นอกจากนี้ยังขาดโปรแกรมที่มีหน้าที่ในการจัดเก็บกฎลงไปยังฐานความรู้

บรรณานุกรม

1. ศ.ดร. วริทธิ์ อึ้งภากรณ์ (1988) : เทอร์โบโปรลอกและระบบผู้เชี่ยวชาญ ,ฟิสิกส์เซ็นเตอร์.

2. ดร. ศุภมิตร จิตตะยโศธร (1988) : การใช้แบบจำลองข้อมูลไนแอมเป็นแบบแสดงความรู้สำหรับฐานความรู้ขนาดใหญ่ (The NIAM Conceptual Schema Model as a Knowledge Representation Paradigm for Large Knowledge Bases),การประชุมทางวิชาการวิศกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 11 เล่ม 1.

3. Amble,T. (1987) : Logic Pragramming and Knowledge Engineering, Addison Wesley Publishing Company,Inc.

4. Bencilhan,F.,Ramakrishnan,R. : An Amateur's Introduction to Recursive Query Processing Strategies.

5. Chang,C.L.,Walker,A. (1986) : Prosql - A Prolog Programming Interface with SQL/DS,Expert Database Systems,The Benjamin Cummings Publishing Company Inc.

6. Clocksin,W.F.,Mellish,C.S. (1984) : Programming in Prolog, Springer-Verlag Berlin Heidelberg New York Tokyo.

7. Date,C.J. (1986) : An Introduction to Database System Vol.1 , Addison-Wesley,Messachusatts.

8. Falkenberg,E.G.,Nyssen,G.M. (1984) : Introduction to IBM SQL, Nyssen Data Base PLY.LTD.,1984.

9. Gray,P.m.d.,Lucas,R.J. (1988),Prolog and Databases,Ellis Horwood Limited,John Wiley&Son.

10. Henschen,L.J.,Naqvi,S.A. : Representing Infinite Sequences of Resolvents in Recursive First-Order Horn Databases.

11. Holub,A.I. (1990) : Compiler Design in C , Prentice-Hall International,Inc.

12. Maier,D.,Warren,D. : Computing With Logic ,The Benjamin / Cummings Publishing Company, Inc.

13. Malpas,J. (1987) : Prolog - A relational language and its applications, Prentice-Hall Introduction,Inc.

14. Neville,D. (1987) : PRO*C User's Guide,Oracle Corporation, Belmant..

15. Schalkoff,R.J. (1989) : Artificial Intelligence, An Engineering Approach , McGraw-Hill Publishing Company.

16. Sickel,S. (1976) : A Search Techique for Clause Interconnectivity Graphs,IEEE TRANSACTIONS ON COMPUTERS VOL C.25,NO. 8,AUGUST 1976.

17. Sorenson,P.G.,Tremblay,J.P. (1984) : An Introduction to Data Structures with Applications ,McGraw-Hill International.

18. Technical Publication Manager : Unix System V ,Programmer Guide, AT&T.

19. Technical Publication Manager (1986) : Oracle Error Message and Codes Manual,Oracle Corporation,Belnamt,California.

20. Technical Publication Manager : Microsoft C 5.0 ,Microsoft.

กิตติกรรมประกาศ

การทำโครงงานครั้งนี้ประสบผลสำเร็จด้วยดี เนื่องมาจากความเอาใจใส่ของอาจารย์ที่ปรึกษาทั้ง 2 ท่านคือ ดร.คุภมิดร จิตตะยโศธร และอาจารย์ ธนา ทงษ์สุวรรณ ที่ให้คำปรึกษาและช่วยในการแก้ปัญหาต่างๆที่เกิดขึ้นตลอดเวลาในการทำ ขอบคุณพ่อกับแม่ที่ช่วยเลี้ยงดูมาจนให้มีวันนี้สำหรับลูกๆ ขอบคุณเพื่อนๆชาวห้องโปรเจคท์ชั้น 5 ทุกคนที่สร้างความครื้นเครงลดบรรยากาศความเครียดในการทำงาน ประกอบด้วยคุณ ชัย, เป้ล, กล้ง, จรูญ, ปีเตอร์, ไก่, ฝน และ ตี๋ ขอบคุณเพื่อนๆภาควิชาวิศวกรรมคอมพิวเตอร์รุ่น 27 ที่ร่วมทุกข์ร่วมสุขกันมาตั้งแต่ร่วมทำงาน single board ในช่วงน้ำท่วมตอนปี 3 ขอบคุณสำหรับ ฮาร์ดดิสค์ตัวใหม่ที่ช่วยให้โปรเจคอยู่รอดมาจนถึงวันนี้

ภาคผนวก ก

โปรสตาร์ซี (Pro*c)

ระบบจัดการฐานข้อมูลแบบสัมพันธ์ของออราเคิล (ORACLE Relational Database Management System) ได้ทำการสร้างโปรแกรมที่ใช้ในการเปลี่ยนประโยคคำสั่งของเอสคิวแอลที่ประกอบอยู่ในโปรแกรมภาษาซี ซึ่งจะนำหน้าด้วย "EXEC SQL" ให้เป็นประโยคคำสั่งในภาษาซี ซึ่งสามารถจัดการฐานข้อมูลของออราเคิลได้ และเรียกกิริยานี้ว่า "การพรีคอมไพล์" (Precompiler) และจากนั้นจึงนำผลที่ได้ มาทำการคอมไพล์หรือลิงก์ โดยใช้คอมไพเลอร์ของภาษาสูงที่ใช้ เพื่อให้ได้โปรแกรมที่สามารถปฏิบัติการได้ ซึ่งโปรแกรมที่ใช้ในการพรีคอมไพล์นี้คือ "โปรสตาร์ซีทูล" (Pro*c Tool)

ดังนั้น เราสามารถสรุปขั้นตอนในการเขียนโปรแกรมโปรสตาร์ซี ได้ดังนี้

1. เขียนโปรแกรมโปรสตาร์ซี
2. ทำการพรีคอมไพล์ โดยใช้โปรสตาร์ซีทูล
3. ทำการคอมไพล์แฟ้มข้อมูลที่ได้จากการคอมไพล์
4. ทำการลิงค์แฟ้มข้อมูลที่เกี่ยวข้องกัน
5. รันโปรแกรม

1. ประโยคคำสั่งของเอสคิวแอลในการปฏิบัติการและการกำหนด (Execute and Declarative SQL Statements)

ประโยคคำสั่งของเอสคิวแอลที่ใช้ในการโปรแกรมโปรสตาร์ซี สามารถแบ่งได้ 2 ประเภทคือ

1. ประโยคคำสั่งปฏิบัติการ คือคำสั่งที่เกี่ยวกับการจัดการข้อมูล (Data Manipulation Language), การกำหนดข้อมูล (Data Definition Language) และการควบคุมข้อมูล (Data Control Language) ซึ่งการปฏิบัติการของประโยคคำสั่งเหล่านี้จะมีผลต่อส่วนการติดต่อข้อมูลของเอสคิวแอล (SQL Communication Area) หรือเรียกอีกอย่างว่าเอสคิวแอลซีเอ (SQLCA) โดยจะมีการแสดงรหัสสถานะที่ได้จากผลของประโยคคำสั่งการปฏิบัติการที่ส่วนนี้ นอกจากนี้ ยังมีผลต่อการเริ่มต้น หรือการสิ้นสุดของส่วนการเก็บการเรียงลำดับของประโยคคำสั่งเอสคิวแอล (Logical Unit of Work)

2. ประโยคคำสั่งการกำหนด คือประโยคคำสั่งที่จะไม่มีการแสดงรหัสสถานะของการปฏิบัติที่ส่วนการติดต่อข้อมูลของเอสคิวแอล และส่วนการเก็บการเรียงลำดับของประโยคคำสั่งของเอสคิวแอล

2. ส่วนประกอบในโปรแกรมโปรสดาร์ซี (Parts of the Pro*C Program)

โปรแกรมโปรสดาร์ซี สามารถแบ่งได้เป็น 2 ส่วน คือ

1. ส่วนโครงสร้างการประยุกต์ (Application Prologue) คือส่วนกำหนดตัวแปรที่ใช้ในการติดต่อกับฐานข้อมูล , การเตรียมพร้อม และการเริ่มต้นเพื่อที่จะติดต่อกับฐานข้อมูล

2. ส่วนตัวโปรแกรมประยุกต์ (Application body) คือ ส่วนของโปรแกรมที่ประโยคคำสั่งต่าง ๆ ของเอสคิวแอล ในการจัดการฐานข้อมูลที่ปรากฏอยู่

3. ส่วนโครงสร้างการประยุกต์

สามารถแบ่งออกได้เป็น 3 ส่วน คือ

1. ส่วนกำหนดตัวแปร (DECLARE section) คือ ส่วนที่ใช้กำหนดตัวแปรและชนิดของตัวแปรในภาษาซีที่จะใช้ในการติดต่อกับฐานข้อมูล มีรูปแบบในการใช้โดยจะเริ่มต้นด้วยประโยค

```
EXEC SQL BEGIN DECLARE SECTION;
```

และจบด้วยประโยค

```
EXEC SQL END DECLARE SECTION;
```

ระหว่างประโยคทั้ง 2 จะเป็นการกำหนดตัวแปรและชนิดของตัวแปร ซึ่งตัวแปรที่กำหนดในส่วนนี้สามารถแบ่งได้เป็น ตัวแปรหลัก (Host Valable) และตัวแปรบ่งชี้ (Indicator Variable) ตัวแปรหลักเป็นตัวแปรที่เก็บค่าข้อมูลต่าง ๆ ที่ใช้ในการติดต่อกับฐานข้อมูล ส่วนตัวแปรบ่งชี้จะใช้ในการเพิ่มข้อมูลในคอลัมน์ที่ไม่ทราบค่า (NULL) และสามารถแสดงให้ทราบว่าข้อมูลที่ตัวแปรหลักได้รับจากฐานข้อมูลมีลักษณะอย่างไร เช่น ไม่ทราบค่า หรือ มีการตัดข้อมูลบางส่วนทิ้ง เป็นต้น

สำหรับชนิดของตัวแปรหลักนั้น จะเหมือนกับชนิดของตัวแปรในภาษาซี แต่ที่เพิ่มขึ้นมาคือประเภท วาร์ชาร์ (VARCHAR) ซึ่งมีลักษณะเป็นแบบเรคอร์ด (Record)

สำหรับข้อผิดพลาด (error) ที่เกิดขึ้นในส่วนการกำหนดตัวแปรนี้ ในกรณีที่ผู้ใช้มิได้กำหนดตัวแปรที่ใช้ในโปรแกรม แต่มีการเรียกใช้ จะมีการแสดงข้อความผิดพลาด คือ

```
Undeclared host variable <a> at line <b> in file <c>
```

2. INCLUDE SQLCA จะมีรูปแบบการใช้คือ

```
EXEC SQL INCLUDE SQLCA;
```

ประโยคคำสั่งนี้ จะเป็นการกำหนดให้โปรสดาร์ซี รวมส่วนของการติดต่อข้อมูลกับเอสคิวแอลเข้าไว้ในโปรแกรม โดยขณะที่ทำการพรีคอมไพล์ ออราเคิลจะทำหน้าที่เปลี่ยนหรือแทนที่ตัวแปรหลักในโปรแกรมด้วยตัวแปรที่ได้จากแฟ้มข้อมูลที่นำมารวม และหน้าที่สำคัญของการติดต่อข้อมูล

กับเอสคิวแอล อีกอย่างหนึ่ง นอกเหนือจากการติดต่อกับออราเคิล ก็คือ การแสดงข้อผิดพลาดและข้อระวังต่างๆ (Warning) ที่เกิดขึ้นในการปฏิบัติคำสั่งของเอสคิวแอลโดยจะแสดงในรูป

- sqlca.sqlcode : ค่ามากกว่า 0 จะแสดงถึงการกระทำคำสั่ง
  ค่าเท่ากับ 0 แสดงว่าทำคำสั่งได้สมบูรณ์
  ค่าน้อยกว่า 0 เกิดการผิดพลาดขึ้น

- sqlca.sqlwarn : จะประกอบด้วยอาร์เรย์ (array) ของแฟลกส์ (flags) 8 ตัว ซึ่งแต่ละตัวก็จะแสดงถึงลักษณะของข้อระวังที่แตกต่างกันออกไป

นอกจากจะสามารถเรียกใช้เอสคิวแอลได้แล้ว ยังสามารถใช้คำสั่งเฉพาะหรือติดต่อกับออราเคิลได้โดยตรง โดยใช้คำสั่ง

```
EXEC SQL INCLUDE ORACA;
```

3. การติดต่อกับออราเคิล (Connecting to ORACLE) มีรูปแบบการใช้คือ

```
EXEC SQL CONNECT (:oracleid> IDENTIFIED BY <:oraclepasswd>
```

หรือ

```
EXEC SQL CONNECT <:oracleid>
```

โดยที่ oracleid อยู่ใน <:oracleid>/<:oraclepasswd> จะเป็นส่วนที่ต้องใช้เพื่อให้โปรแกรมสามารถติดต่อกับระบบจัดการฐานข้อมูลแบบสัมพันธ์ของออราเคิลได้ ซึ่งจะทำให้สามารถเข้าถึงฐานข้อมูลของออราเคิลได้

4. ตัวโปรแกรม (Application Body)

เป็นส่วนที่ภาษาหลัก (Host) และภาษา Embeded รวมกันอยู่ ลักษณะโดยทั่วไปของโปรแกรม คือ

- ภาษาหลัก จะเป็นตัวจัดการเกี่ยวกับการแสดงผล (display) และรูปแบบการใช้งานต่าง ๆ ของโปรแกรม เช่น เมนู เป็นต้น

- ภาษา Embeded จะทำงานในด้านการจัดการเกี่ยวกับข้อมูล รวมทั้งการเรียกใช้คำสั่งของเอสคิวแอลและออราเคิลด้วย ซึ่งการเรียกใช้นั้น จะต้องมี "EXEC SQL" นำหน้าก่อนเสมอ

5. การถามตอบกับเอมเบดด์ (Query)

เป็นส่วนหนึ่งในตัวโปรแกรม (Application Body) ซึ่งจะใช้ในการเรียกข้อมูลมาใช้หรือ เก็บข้อมูลต่างๆ

สำหรับคำสั่งที่ใช้ในการสอบถาม (Query) จะประกอบด้วย

- SELECT
- INTO
- FROM
- WHERE
- CONNECT
- UNION
- INTERSECT
- MINUS
- GROUP BY
- HAVING
- ORDER BY

สำหรับตัวแปรที่ใช้ในการสอบถามนั้น มาจาก 2 ที่ คือ จากตาราง (table) ในภาษาเอมเบดด์ และจากตัวแปรในภาษาหลัก ซึ่งตัวแปรในภาษาหลักที่ใช้ในการสอบถามจะต้องมีเครื่องหมาย ":" (colon) นำหน้าชื่อตัวแปรเสมอ

ลักษณะการสอบถาม มี 2 แบบ คือ

5.1. การสอบถามที่ให้ผลลัพธ์ออกมาเพียง 1 แถว (Query which return SINGLE ROW only) เป็นการสอบถามที่จะต้องอ้างกับค่าที่มีเพียง 1 แถวในตารางเท่านั้น (Unique Index) ซึ่งถ้าให้ค่ามากกว่า1 แถวจะแสดงข้อผิดพลาดออกมา

5.2. การสอบถามที่ให้ผลลัพธ์มากกว่า 1 แถว (Query which return MULTIPLE ROWS) การสอบถามลักษณะนี้มักจะใช้กับการเรียกข้อมูลที่มีเป็นกลุ่มในตาราง ซึ่งเมื่อกระทำการสอบถามนี้แล้ว เอสคิวแอลจะให้ผลลัพธ์ ทั้งหมดออกมาในครั้งเดียว ดังนั้น การใช้การสอบถามแบบนี้จึงจำเป็นต้องเตรียมเนื้อที่ส่วนหนึ่งในออราเคิลหรือเอสคิวแอล เพื่อที่จะใช้ในการเก็บผลลัพธ์นั้นไว้ แล้วจึงเรียกออกมาใช้ตามที่ต้องการ ซึ่งพื้นที่นั้นเรียกว่า เคอร์เซอร์ (Cursor)

เคอร์เซอร์มีลักษณะการใช้ ดังนี้คือ

- การประกาศเคอร์เซอร์ (DECLARE CURSOR) เพื่อกำหนดพื้นที่ , ชื่อ และการสอบถามที่ต้องการ

  รูปแบบ : EXEC SQL DECLARE <cursorname> CURSOR FOR [Query];

- การเปิดเคอร์เซอร์ (OPEN CURSOR) เพื่อเปิดให้สามารถเรียกใช้เคอร์เซอร์ได้

  รูปแบบ : EXEC SQL OPEN <cursorname>;

- การเฟทช์ (FETCH) ให้เคอร์เซอร์แสดงผลลัพธ์ตัวต่อไป

  รูปแบบ : EXEC SQL FETCH <cursorname> INTO <HostVar>;

- การปิดเคอร์เซอร์ (CLOSE CURSOR) เป็นการยกเลิกเคอร์เซอร์ที่ระบุออกไป

  รูปแบบ : EXEC SQL CLOSE <cursorname>;

- ตำแหน่งของเคอร์เซอร์ปัจจุบัน (CURRENT OF CURSOR) ใช้ในการอ้างถึงตำแหน่ง

แถวปัจจุบันในเคอร์เซอร์ที่มีการเฟทช์ข้อมูล เพื่อนำมาใช้ในการแก้ไขหรือลบข้อมูล ตำแหน่งของเคอร์เซอร์ปัจจุบันสามารถใช้แทนด้วย โรว์ไอดี (ROWID)

6. คอมมิท และ โรลแบค (Commit and Rollback)

ในการทำงานของโปรแกรม คำสั่งที่เป็นภาษาเอสคิวแอลแต่ละคำสั่ง จะถูกออราเคิลมองเป็นส่วนย่อย (logical unit of work) ซึ่งในแต่ละส่วนนี้ จะถูกประมวลผลเป็นลำดับขั้นไปจนจบ หรืออาจมีการถูกยกเลิกกลางคันก็ได้ สำหรับการยกเลิก unit of work นั้นเกิดได้จาก 2 กรณี คือ

1. ผู้ใช้ (user) ยกเลิกเอง

2. ระบบ (system) ไม่สามารถทำงานต่อไปได้ เช่น เกิดเดดล็อก (Deadlock) ขึ้น และการจบของ unit of work มี 2 แบบ คือ

- การคอมมิทเวอร์ก (Commit work) เป็นการจบ unit of work โดยให้ผ่านการเก็บการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นไว้ในฐานข้อมูล มีรูปแบบการใช้คือ

```
EXEC SQL COMMIT WORK [RELEASE];
```

โดยที่ทางเลือก RELEASE จะเป็นการคืนเนื้อที่ในหน่วยความจำทั้งหมดและออกจากระบบ (log off) ซึ่งจะเป็นการจบ unit of work สุดท้าย

- โรลแบคเวอร์ก (Rollback Work) เป็นการจบ unit of work เช่นกัน แต่จะทำการยกเลิกการแก้ไขข้อมูลทั้งหมด จะใช้ในกรณีที่เกิดการผิดพลาด ในการทำงานของโปรแกรม มีรูปแบบการใช้ลักษณะเดียวกับคอมมิท คือ

```
EXEC SQL ROLLBACK WORK [RELEASE]
```

7. การแสดงความผิดพลาด (Error and Warning)

หน้าที่สำคัญอีกประการหนึ่งของส่วนติดต่อข้อมูลของเอสคิวแอล คือ เป็นส่วนที่จะกระทำเกี่ยวกับการแสดงความผิดพลาดของโปรแกรม หรือในส่วนต่างๆของเอสคิวแอล โดยที่ลักษณะของส่วนการติดต่อข้อมูลของเอสคิวแอล

8. การตรวจข้อผิดพลาด (Error Detection)

ในการตรวจสอบข้อผิดพลาด จะใช้คำสั่ง WHENEVER ซึ่งจะทำการตรวจสอบสถานะของส่วนการติดต่อข้อมูลของเอสคิวแอลทุก ๆ ครั้งที่กระทำคำสั่งเอสคิวแอล มีรูปแบบการใช้ ดังนี้

```
EXEC SQL WHENEVER [SQLERROR]   [STOP]
                  [SQLWARNING] [CONTINUE]
                  [NOT FOUND]  [GOTO label]
```

โดยที่

- SQLERROR : จะถูก set เมื่อ sqlca.sqlcode เป็นลบ
- SQLWARNING : จะถูก set เมื่อ sqlca.sqlwarn[0] = "W"
- NOT FOUND : จะถูก set เมื่อ sqlca.sqlcode = +1403 (no row found)
- STOP : หยุดการทำงานของโปรแกรม และ โรลแบค
- CONTINUE : ทำงานต่อไป ไม่ว่า sqlca จะเป็นอย่างไร
- GOTO label : ข้ามไปทำงานที่ label

ภาคผนวก ข

ระบบจัดการฐานข้อมูลแบบสัมพันธ์ออราเคิล

(ORACLE RELATIONAL DATA BASE MANAGEMENT SYSTEM)

โดยทั่วไประบบจัดการฐานข้อมูลแบบสัมพันธ์ (RDBMS : Relational Database Management System) เป็นตัวเชื่อมต่อระหว่างหน่วยเก็บข้อมูลแบบกายภาพ กับข้อมูลทางตรรกะ ในทางปฏิบัติจะแบ่งออกเป็นกลุ่มๆ ตามลักษณะการกระทำกับข้อมูลบนเครื่องมือ (Tools) โดยสามารถใช้เครื่องมือเหล่านี้ในการทำงานดังต่อไปนี้

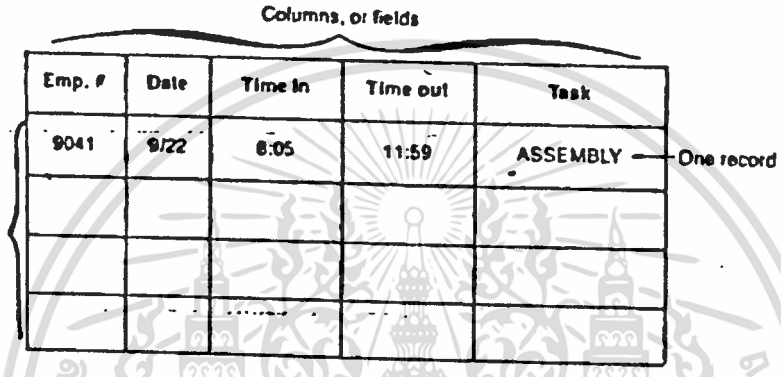
- ให้นิยามของฐานข้อมูล (Define a database)
- สอบถามข้อมูลจากฐานข้อมูล (Query tho database)
- เพิ่มเติม แก้ไข และลบข้อมูล (Add,edit and delete data)
- เปลี่ยนแปลงโครงสร้างของฐานข้อมูล (Modify the structure of the database)
- รักษาความปลอดภัยแก่ข้อมูลจากการใช้งาน (Secure data from public access)
- ติดต่อสื่อสารระหว่างระบบเครือข่าย (Communicate within networks)
- ส่งข้อมูลออก และรับข้อมูลเข้า (Export and Import data)

เนื่องจากระบบจัดการฐานข้อมูลแบบสัมพันธ์อนุญาตให้ผู้ใช้ ควบคุมข้อมูลได้เด็มที่ ทำให้มีผลิตภัณฑ์รองรับการจัดการฐานข้อมูลแบบสัมพันธ์เกิดขึ้นมากมาย ซึ่งความสามารถเหล่านี้ ทำให้ผู้ใช้สามารถเลือกทำงานตามที่ต้องการได้

ในปัจจุบัน ระบบจัดการฐานข้อมูลแบบสัมพันธ์ ต้องการที่จะให้ผู้ใช้มากกว่า 1 คนใช้งานกับฐานข้อมูลเดียวกัน โดยมีความปลอดภัยเท่าที่ควร ดังนั้นโปรแกรมประยุกต์เกี่ยวกับฐานข้อมูลได้มีการพัฒนามาเป็นระบบที่มีผู้ใช้หลายคน (Multiuser) ออราเคิลจะมีเครื่องมือช่วยหลายๆ แบบเพื่อจัดการข้อมูลและการพัฒนาโปรแกรมประยุกต์ สำหรับเวอร์ชั่น 5.1 ออราเคิลได้แบ่งกลุ่มของโปรแกรมกำหนดหน้าที่ ออกเป็นผลิตภัณฑ์ ซึ่งมีหลายทางที่จะเข้าไป และใช้ระบบตามที่ต้องการ

1 การจัดการข้อมูลของออราเคิล (ORACLE)

ในออราเคิลข้อมูลทั้งหมดจะถูกเก็บและแสดงในรูปของตาราง (Table) ในแต่ละตารางจะประกอบไปด้วยคอลัมน์ (Columns) หรือบางทีเรียกว่า ฟิลด์ (fields) และแถว (Rows) แต่ละแถวข้อมูลจะเรียกว่า 1 เรกคอร์ด จากตารางผู้ใช้อาจจะเลือกสับเซตของแถวหรือคอลัมน์แต่ผลลัพธ์จะแสดงออกมาบนหน้าจอ หรือถูกพิมพ์ออกมาเป็นรูปแบบของตารางด้วย



รูปที่ ข1 แสดงโครงสร้างพื้นฐานของตาราง

วิว (View) ได้มาจากตารางซึ่งผู้ใช้สามารถสร้างเพื่อจุดประสงค์สำหรับการแสดงผล ถึงแม้ว่าวิวจะมองดูเหมือนกับว่าเป็นตารางจริง ๆ แต่ในฐานข้อมูล วิวจะเก็บในลักษณะนิยามเท่านั้น ด้วยเหตุผลนี้ วิวก็จะถูกอ้างว่าเป็นตารางเสมือน (Virtual Tables) โดยตารางที่ถูกวิวถ่ายทอดเรียกว่า ตารางพื้นฐาน (Base Tables) วิวอาจจะเกิดจากการรวมกันของ 2 ตารางพื้นฐาน หรือเป็นซับเซตของตารางพื้นฐาน ผู้ใช้สามารถใช้วิวเพื่อที่จะเข้าถึงตารางได้ และเพื่อทำให้งานที่มีความยุ่งยากให้มีความง่ายขึ้น

เนื่องจาก ORACLE เป็นระบบจัดการฐานข้อมูลแบบสัมพันธ์ ผู้ใช้จึงสามารถที่จะเชื่อมโยงข้อมูลที่ถูกเก็บไว้ในหลาย ๆ ตาราง เพื่อเพิ่มประสิทธิภาพในการทำงานและเพื่อการหลีกเลี่ยงการทำซ้ำ การเลือก (Selection) เป็นกระบวนการเพื่อสร้างตารางใหม่ซึ่งประกอบไปด้วยกลุ่มของแถวจากตารางใดๆ ซึ่งตรงกับเกณฑ์ที่ต้องการ โปรเจ็คชั่น (Projection) เป็นกระบวนการเพื่อสร้างตารางจากกลุ่มของคอลัมน์จากตารางใดๆ ซึ่งตรงกับเกณฑ์ที่ต้องการ การจอยน์ (Join) จะสร้างตารางใหม่ซึ่งเป็นการเชื่อมกันของแถวทั้งหมดใน 2 ตาราง โดยไม่รวมแถวที่มี

ข้อมูลเหมือนกัน

2 การเข้าถึงข้อมูลของออราเคิล

ส่วนแก่น(Core) ของออราเคิลคือ เอสคิวแอล(SQL:Structured Query Language) เป็นภาษาซึ่งผู้ใช้สามารถติดต่อกับออราเคิล เอสคิวแอลประกอบไปด้วยกลุ่มคำพื้นฐานของภาษาอังกฤษ เช่น Select และ Create เป็นต้น ซึ่งผู้ใช้สามารถใช้คำสั่งโครงสร้างเพื่อที่จะเข้าถึงและจัดการข้อมูลที่ถูกเก็บไว้ในฐานของมูลแบบสัมพันธ์

ออราเคิลใช้เอสคิวแอลเพื่อที่จะเข้าถึง (Access), เปลี่ยนแปลง (Modify) และแสดงผล(Display) ข้อมูล ออราเคิลยังสามารถใช้เครื่องมือ (Tools) อื่นในการกระทำกับข้อมูลได้เช่นกัน

3 เหตุผลของการเลือกออราเคิล

ออราเคิลมีมาหลายปีในวงการระบบฐานข้อมูลแบบสัมพันธ์ ถึงแม้จะไม่มีความเป็นฟังก์ชันและระบบที่ใช้กับไมโครคอมพิวเตอร์ เหมือนกับ dBASE IV แต่ออราเคิลมีความสามารถในการใช้งานแบบผู้ใช้หลายคน การเข้าถึงการควบคุม และ เอสคิวแอลคอมเพททิบิลิตี้ (SQL-Compatibility) ก็ถูกเพิ่มเข้ามา นอกจากนี้ออราเคิลยังมีข้อดีสำหรับการใช้งาน คือ

3.1 ออราเคิลให้ความปลอดภัยแก่ผู้ใช้

ออราเคิลมีหลายลักษณะ เพื่อที่จะให้ความแน่นอนในความถูกต้องของฐานข้อมูล เมื่อมีการขัดจังหวะ (Interruption) เกิดขึ้นในขณะที่ทำงานอยู่ การกลับคืนสู่ระดับเดิม (Rollback) สามารถที่จะเริ่มให้ฐานข้อมูลชี้ไปยังตำแหน่งที่ก่อนที่จะเกิดความผิดพลาด

ออราเคิลมีฟังก์ชันมากมายสำหรับการรักษาความปลอดภัยของข้อมูล คำสั่งแกรนท์(Grant) และรีโว้ค (Revoke) จะจำกัดการเข้าถึงของข้อมูลลงไปถึงระดับแถวและคอลัมน์ วิว ก็เป็นรูปแบบหนึ่งสำหรับการจำกัดการเข้าถึงตารางพื้นฐานในระบบฐานข้อมูล ซึ่งผู้ใช้จะพบว่ามีหลายทางที่จะควบคุมการเข้าถึงฐานข้อมูลในออราเคิล

3.2 ออราเคิลสามารถดำเนินงานได้อย่างเหมาะสม

ออราเคิล ได้ถูกปรับปรุงอย่างต่อเนื่อง เพื่อที่จะให้มีความสามารถดำเนินงานได้อย่างเหมาะสมบนฐานข้อมูลที่มีขนาดใหญ่ เนื่องจากระบบฐานข้อมูลแบบสัมพันธ์มีข้อเสียในเรื่องของ

ความล่าช้าในการเข้าถึงข้อมูลเป็นอย่างมาก ออราเคิลได้พัฒนาตัวเองอย่างต่อเนื่อง ซึ่งผลก็คือ ชุดคำสั่งประจำสามารถ คำนวณหาทางที่ดีที่สุดที่จะไปยังข้อมูล อย่างรวดเร็ว โดยอัตโนมัติเทคนิคในการจัดกลุ่มของออราเคิล เพื่อที่จะเก็บข้อมูลลงบนดิสก์ ก็เป็นอีกคุณสมบัติหนึ่ง และยังมีฟังก์ชันเพิ่มเติมช่วยควบคุมการดิดฐานข้อมูลที่มีความซับซ้อนด้วย

3.3 ออราเคิลสนับสนุนการพัฒนาโปรแกรมประยุกต์

SQL*Forms เป็นเครื่องมือที่ให้ผู้ใช้ใช้งานได้ง่าย (User Friendly) อย่างดียิ่ง ในการสร้างฟอร์มผู้ใช้สามารถเริ่มต้นด้วย การใช้ฟอร์มที่มีอยู่แล้วหรือฟังก์ในการจัดการหน้าจอเพื่อสร้างรายละเอียดบนหน้าจอ สำหรับการเข้าถึง และการแก้ไข้ ตารางหลาย ๆ ตาราง และเพื่อควบคุมและป้อนข้อมูลเข้าไป

ใน SQL*Forms ออราเคิลได้จัดให้มีหน่วยควบคุมที่เรียกว่าทริกเกอร์ (Trigger) โดยมีผลในการทำงานของผู้ใช้บนฟิลด์ก่อน ระหว่าง และหลัง การป้อนข้อมูลทริกเกอร์เหล่านี้สามารถปฏิบัติตามคำสั่งเอสคิวแอล และคำสั่งใน SQL*Forms หรือโปรแกรมย่อยภายนอกของภาษาเชิงกระบวนคำสั่ง(Procedural Language) SQL*Forms เป็นเครื่องมือของภาษายุคที่ 4 (4GL) ซึ่งจะปรับปรุงตามความต้องการของผู้ใช้ได้อย่างดี

3.4 ออราเคิลใช้ชุดคำสั่งเอสคิวแอล

ออราเคิลใช้ชุดคำสั่งเอสคิวแอล ซึ่งใกล้เคียงกับมาตราฐาน ANSI และมีความเข้ากันได้กับ DB2 ของ IBM และ DS/SQL

ผู้ใช้จะได้รับประโยชน์ของการปรับปรุงอย่างมากของออราเคิล เพื่อให้เข้าใกล้มาตราฐานเอสคิวแอล ออราเคิลได้เพิ่มคำสั่งรูปแบบของการรายงานเพื่อที่จะขยายความสามารถในการรายงานผลของผลลัพธ์ และเพื่อที่จะให้มีการแสดงผลการรายงานอย่างต่อเนื่อง นอกจากนี้ออราเคิลได้เพิ่มฟังก์ชันทางสถิติ (Statistical), ทางคณิตศาสตร์ (Arithmetic), ชุดอักขระ (String) และวันที่เวลา (Date/Time) ด้วย