การสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

นางสาว นุชรี ลาภจิตรกุศล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2543

ISBN 974-346-475-1

A CONSTRUCTION OF PROLOG PROGRAMS FROM FORMAL SPECIFICATION IN THE Z NOTATION

MISS NUCHAREE LAPJITGKUSOL

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2000

ISBN 974-346-475-1

หัวข้อวิทยานิพนธ์	การสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด
โดย	**นางสาวนุชรี ลาภจิตรกุศล**
ภาควิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

……..คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

……..ประธานกรรมการ
(อาจารย์ ดร.พรศิริ หมื่นไชยศรี)

……..อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ)

……..กรรมการ
(อาจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์)

……..กรรมการ
(รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์)

……..กรรมการ
(อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

นุชรี ลาภจิตรกุศล : การสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด (A CONSTRUCTION OF PROLOG PROGRAMS FROM FORMAL SPECIFICATION IN THE Z NOTATION) อ.ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ, 98 หน้า.
ISBN 974-346-475-1.

วิทยานิพนธ์นี้ได้ออกแบบขั้นตอนวิธีและพัฒนาเครื่องมือสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด โดยใช้ข้อกำหนดรูปนัยในรูปสัญกรณ์เซดซึ่งอยู่ในรูปแทคของลาเทกซ์เป็นข้อมูลนำเข้า นอกจากนี้ได้นำเสนอไลบรารีสำหรับอธิบายตัวดำเนินการของสัญกรณ์เซดในรูปกฎของภาษาโพรล็อก กระบวนการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซดประกอบด้วย 2 ขั้นตอนวิธีหลักคือ ขั้นตอนวิธีการเรียงลำดับภาคแสดง และขั้นตอนวิธีการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโพรล็อก ขั้นตอนวิธีการเรียงลำดับภาคแสดงทำการลำดับภาคแสดงในส่วนสัจพจน์ของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดให้ถูกต้อง ส่วนขั้นตอนวิธีการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโพรล็อกจะได้ผลลัพธ์โปรแกรมโพรล็อกที่แสดงถึงข้อกำหนดที่เป็นข้อมูลเข้า

ผลลัพธ์โปรแกรมโพรล็อกที่ได้ดังกล่าวสามารถแสดงฟังก์ชันการทำงานของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดได้อย่างครบถ้วนในระยะต้นของการพัฒนาซอฟต์แวร์

ภาควิชา วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา
ปีการศึกษา 2543	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม

##4170378021 : MAJOR COMPUTER SCIENCE

KEYWORD : FORMAL SPECIFICATION / Z NOTATION / PROLOG PROGRAMS

NUCHAREE LAPJITGKUSOL : A CONSTRUCTION OF PROLOG PROGRAMS FROM FORMAL SPECIFICATION IN THE Z NOTATION

THESIS ADVISOR : ASSIST. PROF. WIWAT VATANAWOOD, 98 pp.

ISBN 974-346-475-1.

This thesis designs algorithms and develops a tool for constructing Prolog programs from formal specification in the Z notation by using Latex format of Z specifications as the input. Moreover, a Prolog's library is proposed as to describe the operators of Z notation in terms of Prolog's rules. The scheme of constructing Prolog programs from formal specifications in Z notation consists of two main algorithms: Axiom Part Reordering Algorithm and Z-to-Prolog Transforming Algorithm. The Axiom Part Reordering Algorithm ensures the appropriate sequence of predicates in axiom part of the Z specifications and the Z-to-Prolog Transforming Algorithm produces the final Prolog programs to represent the input specifications

The Prolog programs are interpreted and successfully demonstrate all functions of formal specifications in Z notation in the early stage of software process.

Department Computer Engineering	Student's signature
Field of study Computer Science	Advisor's signature
Academic year 2000	Co-advisor's signature

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของ ผู้ช่วยศาสตราจารย์วิวัฒน์ วัฒนาวุฒิ อาจารย์ที่ปรึกษา และขอขอบคุณอาจารย์ ดร.พรศิริ หมื่นไชยศรี ประธานกรรมการ และคณะกรรมการดังต่อไปนี้ อาจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์ รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์ และ อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา ที่กรุณาเสียสละเวลาให้คำแนะนำ ตรวจและแก้ไขต้นฉบับวิทยานิพนธ์

ขอขอบคุณ เพื่อน ๆ กลุ่ม Formal Method Group ในห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ที่สละเวลาในการให้คำปรึกษา และช่วยตรวจสอบผลการวิจัยที่ได้ ขอขอบคุณเพื่อน ๆ ที่ให้กำลังใจและข้อเสนอแนะต่าง ๆ และ ขอขอบคุณท่านอื่น ๆ ที่มีส่วนช่วยในการทำวิทยานิพนธ์ที่ไม่ได้กล่าวนามมา ณ โอกาสนี้ด้วย

สุดท้ายนี้ ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา มารดาที่สนับสนุนในด้านต่าง ๆ และให้กำลังใจแก่ผู้วิจัยเสมอมา

นุชรี ลาภจิตรกุศล

สารบัญ

หน้า

สารบัญรูป

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1. ความเป็นมาและความสำคัญของปัญหา

ข้อกำหนดรูปนัย (Formal Specification) คือข้อกำหนดของระบบซึ่งใช้สัญลักษณ์ทางคณิตศาสตร์[1] เพื่ออธิบายพฤติกรรม และโครงสร้างของระบบซอฟต์แวร์ ภาษาข้อกำหนดรูปนัยที่นิยมใช้มักมีลักษณะเป็นตรรกศาสตร์ภาคแสดงลำดับแรก (First Order Predicate Logic) ทำให้การเขียนข้อกำหนดมีความแน่นอน ชัดเจน และไม่กำกวม[2] ข้อกำหนดของระบบซอฟต์แวร์ที่เขียนในลักษณะนี้ ทำให้ผู้พัฒนาระบบในแต่ละขั้นตอนรวมทั้งผู้ใช้สามารถเข้าใจระบบได้ตรงกันอย่างไม่ผิดพลาด ทั้งยังสามารถตรวจสอบข้อกำหนดว่ามีความถูกต้องได้ก่อนเริ่มลงมือออกแบบซึ่งเป็นการลดค่าใช้จ่ายและเวลาในการพัฒนาระบบ[3]

ข้อกำหนดรูปนัยโดยทั่วไปยากที่จะเตรียมขึ้น มีรูปแบบที่ค่อนข้างตายตัว ตรวจสอบความถูกต้องได้ยาก[4] และยากต่อการเข้าใจสำหรับผู้ใช้ทั่วไป เพราะสัญลักษณ์ทางคณิตศาสตร์ที่มีรูปแบบค่อนข้างตายตัว ดังนั้นจึงมีการจำลองการทำงานของข้อกำหนดรูปนัยขึ้นเพื่อแก้ปัญหาดังกล่าวเพราะการจำลองการทำงานของข้อกำหนดรูปนัยสามารถแสดงลักษณะ รวมทั้งฟังก์ชันการทำงานต่าง ๆ ให้ผู้ใช้ได้เกิดจินตทัศน์[5] และทำให้ผู้ใช้เข้าใจได้ง่ายขึ้นว่าระบบสามารถทำอะไรได้บ้าง และตรงต่อความต้องการหรือไม่ เปรียบเสมือนการตรวจสอบความถูกต้องของระบบในขั้นต้น เพื่อช่วยลดความเข้าใจผิดกันระหว่างผู้พัฒนาและผู้ใช้ ทำให้ระบบมีความน่าเชื่อถือมากขึ้น นอกจากนี้ การจำลองการทำงานของข้อกำหนดรูปนัยยังสามารถใช้เป็นต้นแบบเพื่อแสดงการทำงานของระบบในความต้องการ (Requirements) ที่แตกต่างกันได้[6]

วิทยานิพนธ์นี้ จะทำการพัฒนาเครื่องมือเพื่อจำลองการทำงานของข้อกำหนดรูปนัย โดยการแปลงข้อกำหนดรูปนัยไปเป็นภาษาที่สามารถทำงานได้ งานวิจัยครั้งนี้ได้ทำการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซด (formal specification in Z Notation) ซึ่งเป็นข้อกำหนดที่ใช้สัญกรณ์เซด (Z Notation) ซึ่งอยู่บนพื้นฐานของทฤษฎีเซตและตรรกศาสตร์ภาคแสดงลำดับแรกเขียน[3] ไปเป็นภาษาโพรล็อกซึ่งเป็นภาษาที่อยู่บนพื้นฐานทางตรรกศาสตร์ภาคแสดงลำดับแรกเช่นเดียวกัน ทำให้ลดความยุ่งยากในการแปลง[7] และโพรล็อกเป็นภาษาที่ไม่มีชนิด (No Type) ทำให้กฎที่สร้างขึ้นสามารถใช้กับตัวแปรชนิดใดก็ได้และไม่ต้องมีการตรวจสอบชนิดของตัวแปร

1.2. วัตถุประสงค์

1) เพื่อออกแบบขั้นตอนวิธีการแปลงสัญกรณ์เซดให้อยู่ในรูปกฎของภาษาโพรล็อก
2) เพื่อออกแบบและพัฒนาซอฟต์แวร์ที่สามารถแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดให้เป็นภาษาโพรล็อก

1.3. ขอบเขตการวิจัย

1) ข้อมูลเข้าคือ ข้อกำหนดรูปนัยในรูปสัญกรณ์เซด โดยสัญกรณ์เซดเข้าในรูปของแทค(Tag) ของลาเทกซ์ (Latex)
2) ข้อกำหนดรูปนัยที่รับเข้าถูกต้องตามวากยสัมพันธ์แล้ว
3) สัญกรณ์เซดที่ใช้ในข้อกำหนดรูปนัยจะครอบคลุมกลุ่มสัญลักษณ์ต่อไปนี้ ความสัมพันธ์ (Relations) เซต (Sets) ลำดับ (Sequences) แบ็ก (Bags)[8]
4) สร้างไลบราลี (Library) ของภาษาโพรล็อกที่ประกอบด้วยสัญกรณ์เซด เพื่ออธิบายการทำงานของตัวดำเนินการต่าง ๆ ให้อยู่ในรูปที่โปรแกรมโพรล็อกเข้าใจ
5) ซอฟต์แวร์ที่พัฒนาใช้งานบนระบบปฏิบัติการวินโดวส์(Windows)
6) ทดสอบโปรแกรมโดยใช้ตัวอย่างข้อกำหนดรูปนัยในรูปสัญกรณ์เซด โดยให้ครอบคลุมกลุ่มสัญกรณ์เซดในข้อ 3

1.4. ขั้นตอนการวิจัย

1) ศึกษางานวิจัยต่าง ๆ ที่เกี่ยวข้อง
2) ศึกษาข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ได้แก่ การศึกษาสัญลักษณ์ต่าง ๆ และความหมายของสัญลักษณ์ต่าง ๆ ของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด
3) ศึกษาภาษาโพรล็อกได้แก่ การศึกษาวิธีการเขียนภาษาโพรล็อกและวากยสัมพันธ์ต่าง ๆ
4) ศึกษาวิธีการแปลงจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด เป็นภาษาโพรล็อกรวมทั้งการศึกษาการเรียงลำดับภาคแสดงให้ถูกต้อง
5) พัฒนาโปรแกรม
6) ทดสอบโปรแกรม
7) สรุปงานวิจัย
8) จัดทำเอกสาร

1.5. ประโยชน์ที่จะได้รับ

1) สามารถแสดงการจำลองการทำงานข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเบื้องต้นได้
2) สามารถช่วยวิจัยและเป็นเครื่องมือสนับสนุนการเขียนข้อกำหนดรูปนัยด้วยเซด

บทที่ 2
งานวิจัยและทฤษฎีที่เกี่ยวข้อง

2.1. งานวิจัยที่เกี่ยวข้อง

2.1.1) การพัฒนาซอฟต์แวร์:แนวทางการจำลองการทำงานของข้อกำหนดที่เขียนด้วยเซดด้วยภาษาโพรล็อก (Software Development: Two Approaches To Animation Of Z Specifications Using Prolog) โดย Magarett M. West และ Barry M. Eaglestone [9]

เป็นงานวิจัยที่นำเสนอ 2 แนวทางในการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นภาษาโพรล็อกคือ การสังเคราะห์โปรแกรมรูปนัย (Formal Program Synthesis) และวิธีสร้างและทดสอบ (Generate And Test) วิธีสร้างและทดสอบนี้จะทำการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดทั้งในส่วนการประกาศตัวแปร (Declaration Part) และในส่วนสัจพจน์ (Axiom Part) เมื่อมีการประกาศตัวแปรจะทำการสร้างค่าให้กับตัวแปรทุกตัว โดยเริ่มจากค่าของชนิดข้อมูลนั้นลำดับแรก และทดสอบค่าของตัวแปรว่าถูกต้องหรือไม่ด้วยการแทนค่าภาคแสดง (Predicate) ถ้าทดสอบแล้วภาคแสดงเป็นจริงแสดงว่าค่าที่แทนให้กับตัวแปรเป็นค่าที่ถูกต้อง แต่ถ้าไม่ถูกต้องก็จะกลับมาแทนค่าให้กับตัวแปรใหม่ด้วยค่าของชนิดข้อมูลนั้นลำดับถัดมา เช่น ถ้าตัวแปรมีชนิดเป็น จำนวนเต็มบวก (N) ค่าของตัวแปรจะเริ่มจาก 0 ถ้านำเอาตัวแปรไปทดสอบแล้วไม่ถูกต้องก็จะมีการแทนค่าให้กับตัวแปรด้วยข้อมูลลำดับต่อมาคือ 1,2,3..,n ไปเรื่อย ๆ เป็นต้น แล้วนำตัวแปรที่ถูกแทนค่านั้นมาทดสอบใหม่จนกว่าจะพบค่าของตัวแปรที่ทำให้ภาคแสดงเป็นจริง หรือแทนค่าของชนิดข้อมูลให้กับตัวแปรจนหมดทุกค่าแล้วแต่ไม่มีคำตอบที่ถูกต้อง

แต่วิธีสร้างและทดสอบยังมีข้อเสียเนื่องจากตัวแปรทุกตัวจะมีการแทนค่าตั้งแต่มีการประกาศตัวแปรดังที่กล่าวมาแล้ว ซึ่งจะทำให้เกิดการแทนค่าตัวแปรและทดสอบหลายครั้ง ทำให้ใช้เวลานานกว่าจะเจอค่าที่ทำให้ภาคแสดงเป็นจริง หรืออาจจะไม่เจอค่าที่ทำให้ภาคแสดงเป็นจริงเลย และยิ่งถ้าชนิดข้อมูลมีจำนวนมากก็จะเสียเวลาในการแทนค่าและทดสอบมากและอาจจะไม่ได้คำตอบเลย

2.1.2) การจำลองการทำงานของข้อกำหนดที่เขียนด้วยเซดด้วยภาษาโพรล็อกแบบอัตโนมัติ (Automated Animation Of Z Using Prolog) โดย M. A. Hewitt [10]

เป็นงานวิจัยที่เสนอแนวทางในการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นภาษาโพรล็อกโดยใช้วิธีเชิงกระบวนการ (Procedural Approach) วิธีนี้จะคล้ายกับวิธีสร้างและทดสอบของ West และ Eaglestone แต่จะแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเฉพาะส่วนสัจพจน์เท่านั้น โดยได้กำหนดตัวดำเนินการ (Operator) ของสัญกรณ์เซดไว้ในรูปของกฎในภาษาโพรล็อกก่อน และเมื่อทำการแปลงก็จะนำกฎนั้นมาใช้โดยไม่ต้องมีการกำหนดใหม่ การทำงานในส่วนสัจพจน์จะเป็นการหาค่าที่ถูกต้องให้กับตัวแปร ไม่ได้เป็นการทดสอบค่าของตัวแปรเหมือนวิธีสร้างและทดสอบ

แต่วิธีนี้ก็ยังมีปัญหาคือ เนื่องจากในการแปลงจะแปลงเฉพาะส่วนสัจพจน์เท่านั้น ตัวแปรที่เป็นข้อมูลออกจะยังไม่ถูกแทนค่าจนกว่าจะมีภาคแสดงที่ใช้ตัวแปรนั้น แต่ถ้ามีภาคแสดงที่มีอาร์กิวเมนต์เป็นตัวแปรที่เป็นข้อมูลออกทั้งหมด ก็จะทำให้ภาคแสดงนั้นหาคำตอบไม่ได้ ดังนั้น ลำดับของภาคแสดงจึงมีความสำคัญคือ ควรมีภาคแสดงที่เป็นการแทนค่าก่อนมีการเรียกใช้ในกรณีที่ตัวแปรนั้นเป็นข้อมูลออกของภาคแสดง

2.2. ทฤษฎีที่เกี่ยวข้อง

ในงานวิทยานิพนธ์นี้ มีทฤษฎีต่าง ๆ ที่เกี่ยวข้องดังนี้

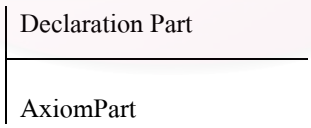
2.2.1) สัญกรณ์เซด (Z Notation)[8]

การเขียนข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้น จะต้องใช้รูปแบบทางคณิตศาสตร์ช่วย โดยในที่นี้จะอธิบายถึงโครงสร้างทั่วไปของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด และรูปแบบทางคณิตศาสตร์ที่ใช้

1) โครงสร้างทั่วไปของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด[11]

ข้อกำหนดที่เขียนโดยใช้สัญกรณ์เซดจะถูกแบ่งเป็นส่วน ๆ เรียกว่าเค้าร่าง เค้าร่างแบ่งออกเป็น 2 ชนิดคือ เค้าร่างเชิงสัจพจน์ (Axiomatic Box) และเค้าร่างเชิงสคีมา (Schema Box) เค้าร่างเชิงสัจพจน์จะใช้ในการประกาศตัวแปร และตัวดำเนินการที่เป็นแบบโกลบอล (Global) ในขณะที่เค้าร่างเชิงสคีมาจะใช้ในการประกาศตัวแปร และตัวดำเนินการที่เป็นโลคอล (Local) เท่านั้น ทั้งสองเค้าร่างจะมีส่วนประกอบเหมือนกัน และมีลักษณะคล้ายกัน

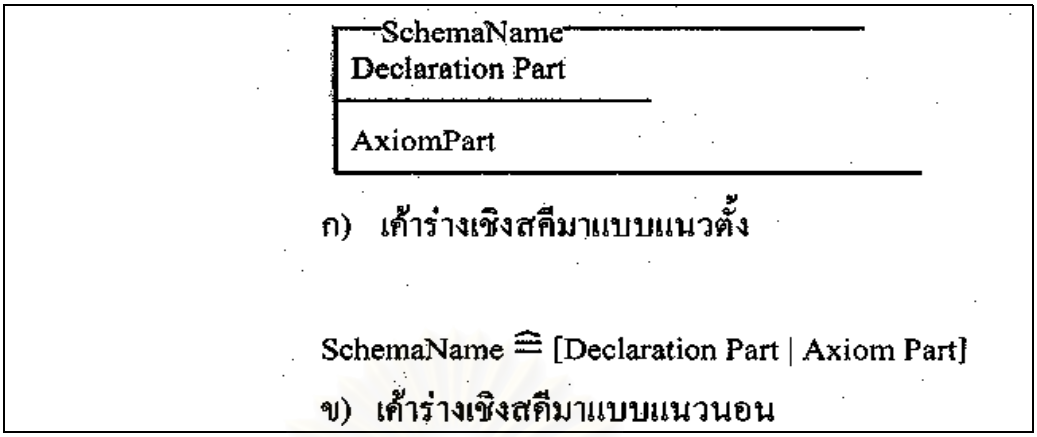
เค้าร่างแต่ละเค้าร่างจะแบ่งออกเป็น 2 ส่วนคือ ส่วนประกาศตัวแปร (Declaration Part) และส่วนสัจพจน์ (Axiom Part) เค้าร่างเชิงสัจพจน์สามารถเขียนได้ดังรูปที่ 2.1



รูปที่ 2.1แสดงเค้าร่างเชิงสัจพจน์

ส่วนเค้าร่างเชิงสคีมาสามารถเขียนได้ 2 แบบคือ แบบแนวตั้ง และแบบแนวนอนดังรูปที่ 2.2 ก) และ ข)

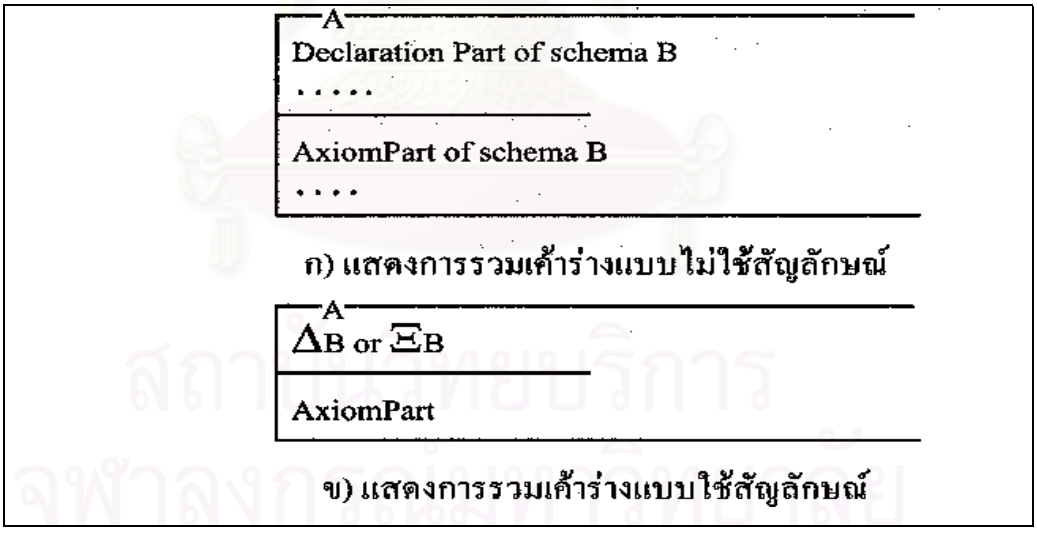
ส่วนการประกาศตัวแปรจะเป็นส่วนที่แสดงตัวแปรทั้งหมดที่ใช้ในเค้าร่างนั้น ๆ ตัวแปรที่ประกาศในเค้าร่างจะเป็นตัวแปรประเภทโลคอล คือใช้ได้เฉพาะในเค้าร่างนั้น ๆ เท่านั้น ส่วนชนิดของตัวแปรที่เซดกำหนดไว้ให้จะมี 3 ชนิดคือ จำนวนเต็ม ($\mathbb{N}$) จำนวนเต็มบวก ($\mathbb{N}_1$) และจำนวนจริง ($\mathbb{Z}$) ส่วนชนิดของตัวแปรอื่น ๆ สามารถกำหนดได้โดยใช้กีเวนเซต(Given Set) ตัวแปรแต่ละตัวสามารถมีสัญลักษณ์ต่อท้ายตัวแปรได้ 3 ชนิดคือ สัญลักษณ์ ? (ปรัศนี:ตัวแปรเข้า) สัญลักษณ์ !(อัศเจรีย์:ตัวแปรออก) และ สัญลักษณ์ Ɔ(ฝนทอง:ตัวแปรสถานะหลัง) การประกาศตัวแปรมีรูปแบบดังนี้ name? : STRING



รูปที่ 2.2 แสดงเค้าร่างเชิงสคีมา

ส่วนสัจพจน์จะเป็นส่วนแสดงการทำงานทั้งหมดของเค้าร่าง แต่ละภาคแสดงจะเชื่อมต่อกันด้วยเครื่องหมายทางตรรกศาสตร์ ในเค้าร่างหนึ่ง ๆ จะมีส่วนสัจพจน์หรือไม่ก็ได้

แต่ละเค้าร่างสามารถเรียกใช้เค้าร่างอื่นได้ เรียกว่าการรวมเค้าร่าง (Schema Inclusion) โดยการรวมเค้าร่างมี 2 ชนิดคือ การรวมแบบมีการเปลี่ยนแปลงสถานะเค้าร่าง(State Changed Schema: Δ) และการรวมแบบไม่มีการเปลี่ยนแปลงสถานะเค้าร่าง(State Unchanged Schema: Ξ) โดยการประกาศการรวมเค้าร่างนี้จะประกาศในส่วนการประกาศตัวแปร ดังรูปที่ 2.3 ก) และ ข)



รูปที่ 2.3 แสดงการรวมเค้าร่าง

2.2.2) รูปแบบทางคณิตศาสตร์[8]

1) เซต (Set)

คือกลุ่มของข้อมูลที่มีชนิดเดียวกัน ใช้สัญลักษณ์ {} (ปีกกา) แสดงความเป็นเซตโดยข้อมูลที่อยู่ภายในเครื่องหมายปีกกาหมายถึงอยู่ภายในเซตเดียวกันและเรียกข้อมูลที่อยู่ภาย

ในว่าเป็นสมาชิก (Member) เช่น {1,2,3,4} เป็นต้น ตัวดำเนินการที่เกี่ยวกับเซตเช่น สัญลักษณ์ == (เครื่องหมายเท่ากับ) สัญลักษณ์ ε (เครื่องหมายสมาชิก) สัญลักษณ์ Y (เครื่องหมายยูเนียน) สัญลักษณ์ I (เครื่องหมายอินเตอร์เซค) และสัญลักษณ์ 0 (เครื่องหมายเซตว่าง) เป็นต้น

2) ความสัมพันธ์ (Relation)

เป็นการบอกความสัมพันธ์ของสมาชิกภายในเซต 2 เซตซึ่งสมาชิกของเซต หนึ่งอาจมีความสัมพันธ์ หรือไม่มีความสัมพันธ์กับสมาชิกของอีกเซตหนึ่ง และอาจมีความสัมพันธ์กับสมาชิกของอีกเซตมากกว่าหนึ่งตัวก็ได้ ซึ่งความสัมพันธ์ระหว่างเซต 2 เซตใช้สัญลักษณ์ x (เครื่องหมายครอส)แทนได้คือ

A=={a,b,c,d,e}

B=={1,2,3,4,5}

$C \subset AxB$ C เซตย่อยของความสัมพันธ์จากเซต A ไปเซต B ดังนั้น C อาจเป็น {(a,1),(b,2),(c,1),(a,3)}

เนื่องจากสมาชิกในเซตอาจมีหรือไม่มีความสัมพันธ์กับสมาชิกในเซตอื่นก็ได้ ดังนั้นในความสัมพันธ์จะเป็นเฉพาะเซตย่อย (Subset) เท่านั้น เซตย่อยของเซตต้นทาง (Source Set) ที่มีความสัมพันธ์กับสมาชิกของเซตเป้าหมาย(Target Set) จะเรียกว่าโดเมน(Domain) และในทางกลับกันสมาชิกของเซตเป้าหมายที่สมาชิกของเซตต้นทาง มีความสัมพันธ์ด้วยอย่างน้อย 1 ตัวจะเรียกว่าพิสัย(Range) จากตัวอย่างข้างต้น

โดเมนของ C คือ {a,b,c} และพิสัยคือ {1,2,3}

3) ฟังก์ชัน (Function)

จะเป็นลักษณะหนึ่งของความสัมพันธ์ จะแสดงถึงความสัมพันธ์ของโดเมนกับพิสัย โดยสมาชิกในโดเมน 1 ตัวจะมีความสัมพันธ์กับสมาชิกของพิสัยได้เพียง 1 ตัวเท่านั้น แต่สมาชิกของพิสัย 1 ตัวจะมีความสัมพันธ์กับสมาชิกของโดเมนกี่ตัวก็ได้

4) ลำดับ (Sequence)

จะมีลักษณะเป็นแถวที่สมาชิกภายในมีลำดับ การแสดงลำดับจะใช้สัญลักษณ์ <> (เครื่องหมายลำดับ)

ลำดับ <a,b>$\bigsqcup$ <b,a> ซึ่งลำดับจะสามารถแสดงในรูปของเซตได้คือ {1->a,2->b}

5) แบ็ก (Bags)

จะมีลักษณะเป็นแถวที่สมาชิกภายในสามารถที่ค่าซ้ำกันได้ และเรียงตามลำดับจากน้อยไปมากแล้ว การแสดงว่าเป็นแบ็กจะใช้สัญลักษณ์ ® ^{TM} (เครื่องหมายแบ็ก) ตัวดำเนินการที่เกี่ยวกับแบ็ก เช่น สัญลักษณ์ in (สมาชิกของแบ็ก) สัญลักษณ์ $\oplus$ (ยูเนียนแบ็ก) สัญลักษณ์ $\uplus$(ลบแบ็ก) เป็นต้น

6) ตรรกศาสตร์

ประกอบไปด้วยตรรกศาสตร์ประพจน์ (Propositional Logic) และตรรกศาสตร์ภาคแสดง (Predicate Logic)

ตรรกศาสตร์ประพจน์ เป็นสัญลักษณ์ที่ใช้ในการเชื่อมภาคแสดงซึ่งจะให้ข้อมูลออกเป็นค่าความจริงคือถูกกับผิดเท่านั้น มี 4 สัญลักษณ์คือ สัญลักษณ์ f (และ:Conjunction) สัญลักษณ์ ϖ (หรือ:Disjunction) สัญลักษณ์ $\Rightarrow$ (ถ้าแล้ว:Implication) และสัญลักษณ์ $\Leftrightarrow$ (ก็ต่อเมื่อ:Equivalence)

ตรรกศาสตร์ภาคแสดงจะมีตัวบ่งปริมาณคือ สิ่งที่บอกปริมาณมี 2 แบบด้วยกันคือ สัญลักษณ์ A (ตัวบ่งปริมาณทั้งหมด:Universal Quantification) และสัญลักษณ์ E (ตัวบ่งปริมาณบางส่วน: Existential Quantification)

2.2.3) ภาษาโพรล็อก [12]

ภาษาโพรล็อกเป็นภาษาโปรแกรมเชิงตรรกะ (Logic Programming) ผู้ใช้จะต้องกำหนดข้อเท็จจริง (Facts) และกฎ(Rules) ของระบบงานเพื่อให้โปรแกรมเรียนรู้ เทคนิคในการเขียนโปรแกรมโพรล็อกจะมีความแตกต่างจากภาษาโปรแกรมที่ใช้กันอยู่เช่น ภาษาซี (C) หรือภาษาปาสคาล (Pascal) ซึ่งเป็นภาษาโปรแกรมเชิงฟังก์ชัน (Functional Programming) ในงานวิจัยนี้จะใช้ SICStus™ Prolog [12] องค์ประกอบหลักของภาษาโพรล็อกที่สำคัญมีดังนี้

1) ส่วนประกอบของวากยสัมพันธ์ในภาษาโพรล็อก (Prolog Component Syntax) โปรแกรมของโพรล็อกจะสร้างจากพจน์ (Term) โดยพจน์สามารถเป็นค่าคงที่ (Constant) ตัวแปร (Variable) หรือโครงสร้าง (Structure) ก็ได้ โครงสร้างอาจเป็นการรวมกันของค่าคงที่ ตัวแปร หรือโครงสร้างอื่น ๆ
   - ค่าคงที่ จะเขียนด้วยตัวอักษรเล็ก
   - ตัวแปร จะเขียนด้วยตัวอักษรใหญ่ หรือสัญลักษณ์ _ (เครื่องหมายขีดล่าง) ซึ่งหมายถึงตัวแปรที่ไม่รู้จัก (Anonymous Variable)
   - โครงสร้างหรือพจน์ผสม (Compound Term) เกิดจากการรวมตัวกันของฟังก์เตอร์ (Functor) และส่วนประกอบโดยฟังก์เตอร์จะตามด้วยเครื่องหมาย ( ) (วงเล็บ) ซึ่งมีอาร์กิวเมนต์อยู่ภายใน ซึ่งภาคแสดงของโพรล็อกจะเขียนในรูปแบบเดียวกัน
   - ลิสต์ (List) เป็นโครงสร้างข้อมูล (Data Structure) ที่สำคัญของภาษาโพรล็อก ลิสต์จะสร้างจากสมาชิก (Element) ที่เป็นพจน์โดยข้อมูลที่เก็บในลิสต์นั้นจะเป็นอะไรก็ได้ โดยทั่วไปลิสต์จะเขียนได้โดยใช้เครื่องหมาย [ ] (ก้ามปู)ซึ่งโดยทั่วไปลิสต์จะประกอบด้วย หัว (Head) และ ท้าย (Tail)
2) ส่วนประกอบของโปรแกรม (Program Component) จะสร้างจากประโยคซึ่งจะประกอบไปด้วย ส่วนหัว (Head) และส่วนตัวของโปรแกรม (Body) มีเครื่องหมาย :- (เครื่องหมายแสดงคำสั่ง) ใช้คั่นระหว่างส่วนหัวและตัวของโปรแกรมและลงท้ายด้วยเครื่องหมาย .

(มหัพภาค) ประโยคจะมีส่วนหัวหรือตัวอย่างเดียวก็ได้ ถ้ามีส่วนหัวจะเรียกว่าข้อความ (Clause) แต่ถ้าไม่มีส่วนหัวจะเรียกว่าหน่วย (Unit)

การประมวลผลของภาษาโพรล็อก (Prolog Processing) ภาษาโพรล็อกจะประกอบไปด้วยข้อเท็จจริงและกฎ ข้อเท็จจริงคือค่าความจริงซึ่งจะเป็นจริงเสมอ ส่วนกฎเป็นสิ่งที่ใช้ในการกำหนดเงื่อนไข กฎจะเป็นจริงได้ต่อเมื่อทุก ๆ เงื่อนไขในกฎนั้นเป็นจริงทั้งหมด ส่วนการถามคำถามจะใช้ข้อคำถาม (Queries) เมื่อมีคำถามเกิดขึ้นโปรแกรมก็จะพยายามหาคำตอบโดยการใช้วิธีการจับคู่รูปแบบ (Pattern Matching) คือเริ่มหาประโยคที่ตรงกันกับเป้าหมาย (Goal) ในลักษณะบนไปล่าง (Top-Down) และจากซ้ายไปขวา (Left To Right) ถ้าเจอประโยคที่ตรงกันก็จะทำในส่วนตัวของประโยคเพื่อพิสูจน์เป้าหมายย่อยว่าถูกต้องหรือไม่ ถ้าถูกก็จะทำเป้าหมายย่อยอันถัดไป แต่ถ้าผิดก็จะมีการถอยหลังกลับไปยังประโยคสุดท้ายที่ทำถูกเพื่อหาว่ามีคำตอบอื่นที่ทำให้ประโยคนี้ถูกหรือไม่ ถ้ามีจะหาคำตอบใหม่แล้วทำต่อไปเรื่อย ๆ ถ้าไม่มีก็แสดงว่าข้อคำถามนี้ไม่เป็นจริง

การเรียกตัวเอง (Recursion) เป็นวิธีที่นิยมใช้ในการเขียนโปรแกรมโดยใช้ภาษาโพรล็อก การเขียนเงื่อนไขในกฎ จะมีเงื่อนไขที่ทำการเรียกตัวเองในการทำงาน เพื่อให้มีการทำงานเป็นวง (Loop) และต้องมีการกำหนดกฎที่เป็นจุดสิ้นสุดของการเรียกตัวเองนั้นไว้

2.2.3) โปรแกรม Z/EVES[13]

โปรแกรม Z/EVES เป็นเครื่องมือที่ใช้ในการวิเคราะห์ข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ซึ่งสามารถทำการตรวจสอบข้อกำหนดรูปนัยในรูปสัญกรณ์เซดในด้านต่าง ๆ เช่น การตรวจสอบวากยสัมพันธ์ (Syntax Checking) การตรวจสอบประเภทข้อมูล (Type Checking) การตรวจสอบโดเมน (Domain Checking) การพิสูจน์ทฤษฎีบท (Theorem Proving) เป็นต้น ภาษาที่ Z/EVES ใช้ในการทำงานคือ ลาเทกซ์ที่สอดคล้องกับสัญกรณ์เซด โดย Z/EVES ได้กำหนดแทคของลาเทกซ์ที่จะใช้สำหรับเขียนข้อกำหนดรูปนัยในรูปสัญกรณ์เซดขึ้น โดยแทคของลาเทกซ์ของ Z/EVES แสดงอยู่ในภาคผนวก ง. และข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่จะใช้ตรวจสอบวากยสัมพันธ์โดย Z/EVES ได้นั้น จะต้องใช้แทคของลาเทกซ์ของ Z/EVES เขียนเช่นกัน

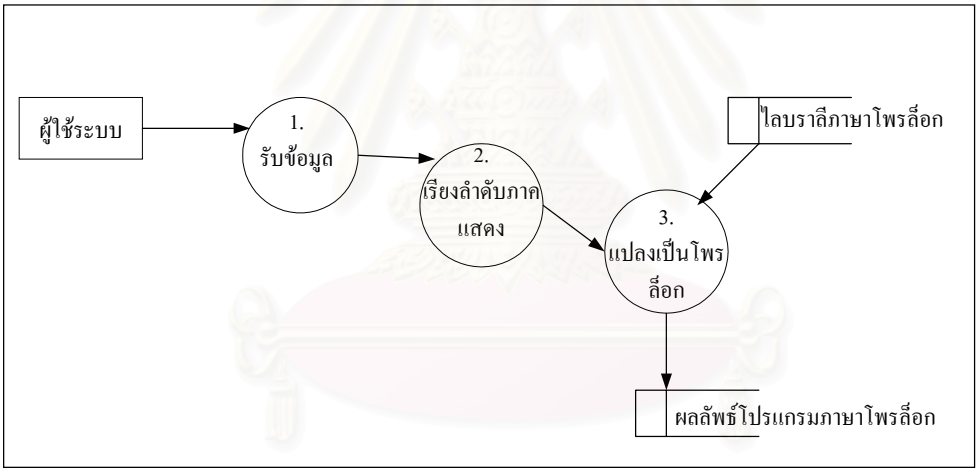
ข้อกำหนดรูปนัยในรูปสัญกรณ์เซดในรูปสัญกรณ์เซดที่จะนำมาแปลงเป็นโปรแกรมโพรล็อกนั้นจะมีการตรวจสอบวากยสัมพันธ์โดย Z/EVES ก่อนเพื่อเป็นการพิสูจน์ว่าข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้นมีวากยสัมพันธ์ถูกต้อง และสามารถทำงานได้จริง ในการพิสูจน์นั้นจะมีการตรวจสอบวากยสัมพันธ์ และการสร้างทฤษฎีเพื่อสร้างเป้าหมายในการทำงาน และสามารถทำงานไปถึงเป้าหมายนั้น ๆ ได้

บทที่ 3
ขั้นตอนวิธีสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

ในบทนี้ จะกล่าวถึงขั้นตอนและวิธีการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซดซึ่งอยู่ในรูปแบบของลาเทกซ์ โดยได้แบ่งขั้นตอนในการทำงานออกเป็น 3 ส่วนคือ

1. ส่วนการรับข้อมูล เป็นส่วนที่ทำการรับแฟ้มข้อมูลข้อกำหนดรูปนัยในรูปสัญกรณ์เซด
2. ส่วนการเรียงลำดับภาคแสดงของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด เป็นส่วนที่ทำการจัดลำดับของภาคแสดงในแต่ละเค้าร่างให้ถูกต้อง
3. ส่วนแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นภาษาโพรล็อก โดยในส่วนนี้ได้มีการกำหนดไลบราลีของภาษาโพรล็อกไว้ก่อนทำการแปลงด้วย

ขั้นตอนวิธีการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซดสามารถแสดงได้ดังรูปที่ 3.1



รูปที่ 3.1 แผนภาพแสดงขั้นตอนการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

3.1. ส่วนรับข้อมูล

ส่วนรับข้อมูลเป็นส่วนที่รับข้อกำหนดรูปนัยในรูปสัญกรณ์เซดซึ่งอยู่ในรูปแทคของลาเทกซ์ของ Z/EVES ดังแสดงในภาคผนวก ง. การเขียนข้อกำหนดรูปนัยในรูปสัญกรณ์เซดในรูปแทคของลาเทกซ์นั้นจะแบ่งข้อกำหนดออกเป็น 2 ส่วนคือ ส่วนการกำหนดชนิดของข้อมูล และส่วนเค้าร่าง

3.1.1) ส่วนการกำหนดชนิดข้อมูล

ส่วนการกำหนดชนิดข้อมูล เป็นส่วนที่ใช้ในการประกาศชนิดของข้อมูลที่ใช้ ซึ่งจะเป็นการประกาศแบบโกลบอล ส่วนการประกาศกีเวนเซต และการนิยามชนิดอิสระ

ตัวอย่างการเขียนส่วนประกาศกีเวนเซตในรูปแทคของลาเทกซ์

[NAME,DATE]

สามารถเขียนในรูปแทคของลาเทกซ์ได้ดังนี้

\begin{zed}

[NAME,DATE]

\end{zed}

ตัวอย่างการเขียนการนิยามอิสระในรูปแทคของลาเทกซ์

REPORT ::= Ok | Not_known | Already_known

สามารถเขียนในรูปแทคของลาเทกซ์ได้ดังต่อไปนี้

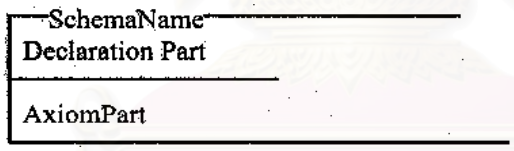
\begin{zed}

REPORT ::= Ok | Not_known | Already_known

\end{zed}

3.1.2) ส่วนเค้าร่างคือส่วนที่เป็นการอธิบายการทำงานของระบบ ซึ่งจะเขียนอยู่ในรูปของเค้าร่างเชิงสคีมาแบบแนวตั้ง และแนวนอน

ตัวอย่างการเขียนเค้าร่างเชิงสคีมาแบบตั้งในรูปแทคของลาเทกซ์



สามารถเขียนในรูปแทคของลาเทกซ์ได้ดังต่อไปนี้

\begin{schema}{SchemaName}

Declaration Part

\where

Axiom Part

\end{schema}

ตัวอย่างการเขียนเค้าร่างเชิงสคีมาแบบแนวนอนในรูปแทคของลาเทกซ์

RAddBirthday $\hat{=}$ (AddBirthday $\fatsemi$ Success) ϖ AlreadyKnown

สามารถเขียนในรูปแทคของลาเทกซ์ได้ดังต่อไปนี้

\begin{zed}

RAddBirthday \defs (AddBirthday \fatsemi Success) \lor AlreadyKnown

\end{zed}

3.2. ส่วนการเรียงลำดับภาคแสดง

เนื่องจากการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อกได้เลือกใช้วิธีแบบอัติโนมัติ ซึ่งมีปัญหาในการแปลงคือ ถ้าลำดับของภาคแสดงในข้อกำหนดรูปนัยในรูปสัญกรณ์เซดไม่ถูกต้องจะทำให้เกิดข้อผิดพลาดเมื่อกระทำการโปรแกรมโพรล็อกที่สร้างได้ตามที่กล่าวมาแล้วในงานวิจัยที่เกี่ยวข้อง ดังนั้น ขั้นตอนนี้จึงเป็นการแก้ปัญหาดังกล่าว โดยให้มีการเรียงลำดับภาคแสดงให้ถูกต้องก่อนจึงแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก โดยการเรียงลำดับภาคแสดงนี้สามารถตรวจสอบได้โดยการเรียกกฎที่มีลำดับของภาคแสดงผิดพลาดทำงาน หากการเรียงลำดับภาคแสดงยังไม่ถูกต้อง ก็จะมีข้อผิดพลาดเกิดขึ้น

ในการเรียงลำดับภาคแสดงนั้น จะต้องมีการพิจารณาถึงชนิดตัวแปร และชนิดของภาคแสดง

3.2.1) ตัวแปร

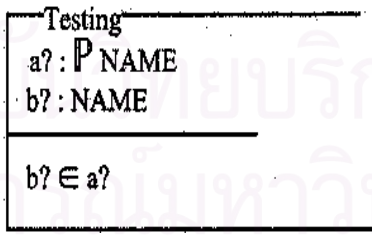
การพิจารณาตัวแปรเพื่อเรียงลำดับภาคแสดงนั้น จะแบ่งตัวแปรออกเป็น 2 ชนิดคือ

- ตัวแปรที่มีค่าแล้ว
  ตัวแปรที่มีค่าแล้วได้แก่ตัวแปรที่เป็นข้อมูลเข้า คือตัวแปรที่มีสัญลักษณ์ ? อยู่ท้ายตัวแปร
- ตัวแปรที่ยังไม่มีค่า
  ตัวแปรที่ยังไม่มีค่าได้แก่ตัวแปรที่เป็นข้อมูลออก และตัวแปรที่เป็นสถานะหลัง คือตัวแปรที่มีสัญลักษณ์ ! และ ∋ อยู่ท้ายตัวแปร

3.2.2) ภาคแสดง

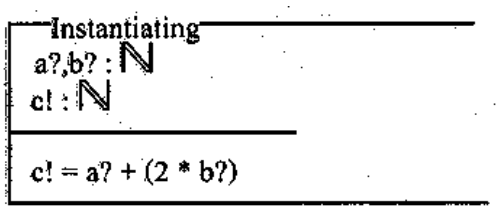
การพิจารณาภาคแสดงเพื่อเรียงลำดับนั้น จะแบ่งภาคแสดงออกเป็น 3 ชนิดคือ

- ภาคแสดงตรวจสอบ (Testing Predicate) คือภาคแสดงที่ตัวแปรทั้งหมดในภาคแสดงเป็นตัวแปรที่มีค่าแล้ว ดังนั้นการทำงานของภาคแสดงจึงเป็นการตรวจสอบค่าของตัวแปรว่าถูกต้องหรือไม่เท่านั้น ไม่ได้เป็นการสร้างค่าให้กับตัวแปรใด ๆ ตัวอย่างเช่น



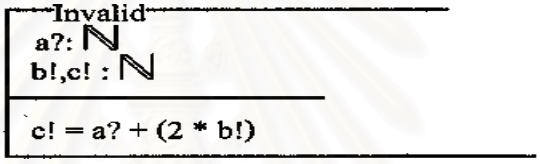
จากเค้าร่าง Testing ตัวแปร a? และ b? เป็นตัวแปรที่มีค่าแล้ว เพราะเป็นตัวแปรที่เป็นข้อมูลเข้าของเค้าร่าง ดังนั้น ภาคแสดง b? ε a? จึงเป็นภาคแสดงตรวจสอบโดยตรวจสอบว่า b? เป็นสมาชิกของ a? หรือไม่

- ภาคแสดงที่มีการแทนค่า (Instantiating Predicate) คือภาคแสดงที่ตัวแปรใดตัวแปรหนึ่งในภาคแสดงเป็นตัวแปรที่ยังไม่มีค่า ส่วนตัวแปรอื่น ๆ ที่เหลือเป็นตัวแปรที่มีค่าแล้วทั้งหมด การทำงานของภาคแสดงนี้จะเป็นการหาค่าให้กับตัวแปรที่ยังไม่มีค่านั้น ๆ ตัวอย่างเช่น



จากเค้าร่าง Instantiating ตัวแปร a?และ b? เป็นตัวแปรที่มีค่าแล้ว เพราะเป็นตัวแปรที่เป็นข้อมูลเข้าของเค้าร่าง แต่ตัวแปร c! เป็นตัวแปรที่ยังไม่มีค่า เพราะเป็นตัวแปรที่เป็นข้อมูลออกของเค้าร่างดังนั้นภาคแสดง c! = a? + (2 * b?) จึงเป็นภาคแสดงที่มีการแทนค่า เพราะหลังจากการทำงานของภาคแสดงนี้แล้ว ตัวแปร c! จะกลายเป็นตัวแปรที่มีค่า

- ภาคแสดงที่ผิด (Invalid Predicate) คือภาคแสดงที่มีตัวแปรที่ยังไม่มีค่ามากกว่าหนึ่งตัวแปรขึ้นไป ทำให้ภาคแสดงนั้นไม่สามารถทำงานได้ เนื่องจาก ไม่สามารถเปรียบเทียบค่าตัวแปร หรือหาค่าตัวแปรให้กับตัวแปรที่ยังไม่มีค่าได้นั่นเอง ตัวอย่างเช่น



จากเค้าร่าง Invalid ตัวแปร a? เป็นตัวแปรที่มีค่าแล้ว เพราะเป็นตัวแปรที่เป็นข้อมูลเข้าของเค้าร่าง แต่ตัวแปร b! และ c! เป็นตัวแปรที่ยังไม่มีค่า เพราะเป็นตัวแปรที่เป็นข้อมูลออกของเค้าร่างดังนั้นภาคแสดง c! = a? + (2 * b!) จึงเป็นภาคแสดงที่ผิด เพราะมีตัวแปรที่ยังไม่มีค่ามากกว่าหนึ่งตัวแปร

เมื่อแบ่งภาคแสดงออกเป็น 3 กลุ่มแล้ว ภาคแสดงตรวจสอบและภาคแสดงที่มีการแทนค่าเท่านั้นที่เป็นภาคแสดงที่อยู่ในลำดับที่ถูกต้องแล้ว ส่วนสัจพจน์ที่ผิดเป็นภาคแสดงที่อยู่ในลำดับที่ไม่ถูกต้อง จึงต้องมีการเรียงลำดับภาคแสดงที่ผิดใหม่ให้อยู่ในลำดับที่ถูกต้อง โดยเมื่อมีการเรียงลำดับภาคแสดงที่มีการแทนค่านั้น ตัวแปรหนึ่งตัวจะถูกแทนค่า ซึ่งจะทำให้ตัวแปรที่ยังไม่มีค่าเป็นตัวแปรที่มีค่า ดังนั้น จึงต้องมีการย้ายตัวแปรที่ถูกแทนค่ามาอยู่ในกลุ่มตัวแปรที่มีค่าแล้ว ซึ่งอาจทำให้ภาคแสดงที่ผิด กลายเป็นภาคแสดงที่ถูกต้องขึ้นมาได้

<u>ขั้นตอนวิธีการเรียงลำดับภาคแสดง(Axiom Part Reordering Algorithm)</u>[10]

ให้ Input เป็นเซตของตัวแปรเข้า

Output เป็นเซตของตัวแปรออก

P เป็นเซตของภาคแสดงในส่วนสัจพจน์

var? เป็นตัวแปรเข้าของเค้าร่าง

var! เป็นตัวแปรออกของเค้าร่าง

varƎ เป็นตัวแปรสถานะหลังของเค้าร่าง

p เป็นภาคแสดง

ในการเรียงลำดับมีขั้นตอนดังต่อไปนี้

ก) ในส่วนภาคประกาศตัวแปร ให้ตัวแปร var? อยู่ในเซต Input ตัวแปร var! และ var϶ อยู่ในเซต Output

ข) ให้ทุก p ในส่วนสัจพจน์อยู่ใน P

ค) ถ้า P ยังไม่หมด

{ ถ้า p เป็นภาคแสดงตรวจสอบ

ใส่ p ในส่วนสัจพจน์

ลบ p ออกจาก P

}

{ ถ้า p เป็นภาคแสดงที่มีการแทนค่า

ย้ายตัวแปร var! หรือ var϶ ไปที่เซต Input

ลบตัวแปร var! หรือ var϶ ออกจากเซต Output

ใส่ p ในส่วนสัจพจน์

ลบ p ออกจาก P

}

{ ถ้า p เป็นภาคแสดงที่ผิด

ไปที่ p ต่อไป

}

ตัวอย่างการเรียงลำดับภาคแสดง

AddBirthday1

known!, known' : ℙ NAME
birthday', birthday : NAME ⇸ DATE
name? : NAME
date? : DATE

---

known! = dom birthday ----------(1)
name? ∉ known! ----------(2)
known!' = dom birthday' ----------(3)
birthday' = birthday ∪ ({name? ↦ date?}) ---------- (4)

หลังจากทำการเรียงลำดับภาคแสดง

AddBirthday2

known!, known!' : ℙ NAME
birthday', birthday : NAME ⇸ DATE
name? : NAME
date? : DATE

---

known! = dom birthday ----------(1)
name? ∉ known! ----------(2)
birthday' = birthday ∪ ({name? ↦ date?}) ----------(3)
known!' = dom birthday' ----------(4)

จากเค้าร่าง AddBirthday1 ตัวแปรที่มีค่าแล้วได้แก่ตัวแปร birthday, name? และ date? ส่วนตัวแปรที่ยังไม่มีค่าได้แก่ตัวแปร known! ,known!' และ birthday'ในเค้าร่าง AddBirthday1 นี้มีภาคแสดงทั้งหมด 4 ภาคแสดง

ภาคแสดงที่ 1 เป็นภาคแสดงที่มีการแทนค่า เพราะหลังจากภาคแสดงที่ 1 ทำงาน จะทำให้ตัวแปร known! เป็นตัวแปรที่มีค่า

ภาคแสดงที่ 2 เป็นภาคแสดงตรวจสอบ เพราะตัวแปรในภาคแสดงที่ 2 นี้เป็นตัวแปรที่มีค่าแล้ว

ภาคแสดงที่ 3 เป็นภาคแสดงที่ผิด เพราะว่าทั้ง known!' และ birthday' เป็นตัวแปรที่ไม่มีค่า

ภาคแสดงที่ 4 เป็นภาคแสดงที่มีการแทนค่า เพราะหลังจากภาคแสดงที่ 4 ทำงาน จะทำให้ตัวแปร birthday' เป็นตัวแปรที่มีค่า

ดังนั้น ควรนำภาคแสดงที่ 3 มาไว้ต่อจากภาคแสดงที่ 4 เพราะหลังจากภาคแสดงที่ 4 ทำงานจะทำให้ตัวแปร birthday' มีค่า และจะทำให้ภาคแสดงที่ 3 กลายเป็นภาคแสดงที่มีการแทนค่า และทำให้ตัวแปร known!' เป็นตัวแปรที่มีค่า ดังแสดงในเค้าร่าง AddBirthday2

3.3. ส่วนการแปลงเป็นโปรแกรมโพรล็อก

ส่วนการแปลงเป็นโปรแกรมโพรล็อกได้แบ่งออกเป็น 2 ขั้นตอนย่อยคือ การสร้างไลบราลีของภาษาโพรล็อก เพื่อนำมาใช้ในการแปลง และส่วนการแปลงเป็นโปรแกรมโพรล็อก ซึ่งจะต้องทำการแปลงให้สอดคล้องกับไลบราลีที่ได้สร้างขึ้น

3.3.1) ส่วนการสร้างไลบราลีภาษาโพรล็อก

ส่วนการสร้างไลบราลีภาษาโพรล็อกเป็นส่วนที่นำเอาตัวดำเนินการของสัญกรณ์เซตแต่ละตัวดำเนินการมาสร้างเป็นกฎในภาษาโพรล็อก เพื่อทำการอธิบายการทำงานของตัวดำเนินการดังกล่าวให้อยู่ในรูปที่โพรล็อกเข้าใจ และสามารถนำไปใช้งานได้ การเตรียมไลบราลีภาษาโพรล็อกนี้ ทำด้วยมือ (Manual) ซึ่งเป็นการทำครั้งเดียวและสามารถใช้ได้กับข้อกำหนดรูปนัยในรูปสัญกรณ์เซตได้ทุกข้อกำหนด

ก่อนที่จะมีการสร้างไลบราลีของภาษาโพรล็อกนั้น จะต้องมีการออกแบบชนิดของข้อมูลของสัญกรณ์เซตให้อยู่ในรูปแบบของภาษาโพรล็อกก่อน เพราะสัญกรณ์เซตมีกลุ่มของข้อมูล 5 ชนิดคือ เซต ความสัมพันธ์ ฟังก์ชัน ลำดับ และแบ็ก ในขณะที่ในภาษาโพรล็อกจะมองกลุ่มของข้อมูลเป็นแบบลิตส์เพียงอย่างเดียว ดังนั้น จึงต้องมีการแสดงกลุ่มของข้อมูลต่าง ๆ ของสัญกรณ์เซตในรูปแบบลิสต์ของภาษาโพรล็อกดังตารางที่ 3.1

จากนั้นในขั้นตอนการแปลงจะทำการแบ่งกลุ่มของตัวดำเนินการออกเป็น 5 กลุ่มซึ่งพิจารณาจากรูปแบบของข้อมูลได้แก่ เซต ความสัมพันธ์ ลำดับ แบ็ก และอื่น ๆ ในส่วนการสร้างกฎของตัวดำเนินการนั้นจะสร้างโดยพิจารณาจากการทำงานของตัวดำเนินการ โดยส่วนการเขียนกฎในภาษาโพรล็อกจะใช้วิธีการเรียกตัวเอง โดยที่ 1 ตัวดำเนินการอาจต้องใช้หลายกฎในการอธิบายการทำงาน

ตารางที่ 3.1 แสดงชนิดข้อมูลของสัญกรณ์เซตและภาษาโพรล็อก

กลุ่มของข้อมูล	สัญกรณ์เซต	ภาษาโพรล็อก
เซต	{a,b,c}	[a,b,c]
ความสัมพันธ์	{a□1,b□2}	[(a,1),(b,2)]
ฟังก์ชัน	{1□1,2□2}	[(1,1),(2,2)]
ลำดับ	©a,b,c〉	[(1,a),(2,b),(3,c)]
แบ็ก	®a,b,c™	[a,b,c]

แต่ละกฎของตัวดำเนินการ จะประกอบด้วยชื่อ ตัวแปร และเงื่อนไขของกฎนั้น ๆ ชื่อของกฎจะนำมาจากชื่อของตัวดำเนินการจากแทคของลาเทกซ์ ส่วนตัวแปรจะมีตัวแปรเข้า และตัวแปรออก ส่วนเงื่อนไขจะเป็นส่วนที่อธิบายการทำงานของตัวดำเนินการนั้น ๆ

ไลบราลีของภาษาโพรล็อก

1) กลุ่มเซต เป็นกลุ่มตัวดำเนินการที่ทำงานเกี่ยวกับข้อมูลที่เป็นเซต

- สัญลักษณ์ ≠ (ไม่เท่ากับ:Inequlity) เช่น A≠B แสดงว่าเซต A ที่ไม่เท่ากับเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
neq(A,B):- \+ (A=B).
```

- สัญลักษณ์ ∉ (ไม่เป็นสมาชิก:Non Membership) เช่น A ∉ B แสดงว่า A ไม่ได้เป็นสมาชิกของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
notin(A,B):- \+(member(A,B)).
```

- สัญลักษณ์ ⊂ (เซตย่อย:Subset) เช่น A⊂B แสดงว่าเซต A เป็นเซตย่อยของอีกเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
subset([],L).
subset([H|T],L):- subset(T,L), member(H,L).
```

- สัญลักษณ์ ∪ (การยูเนียน:Set Union) เช่น A∪B เป็นการเอาสมาชิกของเซต A และเซต B มารวมกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
cup([],List,List).
cup([H|T],List,Union):- \+ (member(H,List)), cup(T,List,U1), append
                                    ([H],U1,Union).
cup([H|T],List,Union):- member(H,List),cup(T,List,U1), append
                                    ([],U1,Union).
```

- สัญลักษณ์ ∩ (การอินเตอร์เซค:Set intersection) เช่น A∩B เป็นการเอาสมาชิกของเซต A / และเซต มารวมกันเฉพาะที่เหมือนกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
cap([],List,[]).
```

```
cap([H|T],List,Intersect):- \+ (member([H],List)), cap(T,List,U1), append
                                ([],U1,Intersect).
cap([H|T],List,Intersect):- member([H],List), cap(T,List,U1), append
                                ([H],U1,Intersect).
```

- สัญลักษณ์ \ (การลบเซต:Set Difference) เช่น A\B เป็นการเอาสมาชิกของเซต A ที่ไม่มีในเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
setminus(List,[],List).
setminus(List,[H|T],Different):- setminus(List,T,R1), delete(H,R1,Different).
```

- สัญลักษณ์ ∪ (การยูเนียนเซต:Generalizaed Union) เช่น ∪A นำเอาสมาชิกในทุกเซตใน A มารวมกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
bigcup([],[]).
bigcup([H|T],Gunion):- bigcup(T,L1), cup(H,L1,Gunion).
```

- สัญลักษณ์ ∩ (การอินเตอร์เซคเซต:Generalizes Intersection) เช่น ∩ A นำเอาสมาชิกในทุกเซตใน A เฉพาะที่เหมือนกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
bigcap([L],L).
bigcap([H|T],Gintersect):-bigcap(T,L1),cap(H,L1,Gintersect).
```

- สัญลักษณ์ first (การหาคู่ลำดับตัวแรก:First) เช่น first A เอาสมาชิกตัวแรกของคู่ลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
first(P,First):- P=..[A,B,C|D], First=B.
```

- สัญลักษณ์ second (การหาคู่ลำดับตัวหลัง : Second) เช่น second B เอาสมาชิกตัวที่ 2 ของคู่ลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
second(P,Second):- P=..[A,B,C|D], Second=C.
```

2) กลุ่มความสัมพันธ์ เป็นกลุ่มตัวดำเนินการที่เกี่ยวกับความสัมพันธ์

- สัญลักษณ์→(การแปลงคู่ลำดับ:Maplet) เช่น A→B เป็นการแปลงให้อยู่ในรูปคู่ลำดับ(A,B) ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
mapsto(A,B,C):-C=[(A,B)].
```

- สัญลักษณ์ dom (การหาโดเมน:Domain) เช่น dom A เป็นการหาโดเมนของความสัมพันธ์ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
dom([],[]).
dom([H|T],Domain):- H=..[A,B,C|Z], dom(T,D1), append([B],D1,Domain).
```

- สัญลักษณ์ ran (การหาพิสัย:Range) เช่น ran B เป็นการหาพิสัยของความสัมพันธ์ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
ran([],[]).
ran([H|T],Range):- H=..[A,B,C|D], ran(T,R1), append([C],R1,Range).
```

- สัญลักษณ์ ; (การเชื่อมความสัมพันธ์:Relational Composition) เช่น A;B เป็นการหาพิสัยของเซต B ที่มีโดเมนเหมือนกับพิสัยของเซต A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
comp(P,Q,R):- ran(P,R1), funran1(R1,Q,R).
```

- สัญลักษณ์ O (การหาความสัมพันธ์แบบถอยหลัง:Backward Relation) เช่น AOB เป็นการหาโดเมนของเซต A ที่มีพิสัยเหมือนกับโดเมนของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
circ(P,Q,R):- dom(Q,R1), fundom1(R,P,R1).
```

- สัญลักษณ์ ◁ (การจำกัดโดเมน:Domain Restriction) เช่น A◁B เป็นการหาคู่ลำดับของเซต A ที่มีโดเมนเป็นสมาชิกของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
dres([],Relation,[]).
dres([H|T],Relation,Domres):- domone(H,Relation,R1),
dres(T,Relation,Re1), append(R1,Re1,Domres).
```

- สัญลักษณ์ ▷ (การจำกัดพิสัย:Range Restriction) เช่น A▷B เป็นการหาคู่ลำดับของเซต A ที่มีพิสัยเป็นสมาชิกของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
rres(Relation,[],[]).
rres(Relation,[H|T],Ranres):- ranone(H,Relation,R1), rres(Relation,T,Re1),
append(R1,Re1,Ranres).
```

- สัญลักษณ์ ⩤ (การหาโดเมนที่ไม่ถูกจำกัด:Domain Anti Restriction) เช่น A⩤B เป็นการหาคู่ลำดับของเซต A ที่มีโดเมนไม่เป็นสมาชิกของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
ndres(B,[],B).
ndres(Relation,[H|T],Domantires):- domantione(H,Relation,R1), ndres
                                   (R1,T,Domantires).
```

- สัญลักษณ์ ⩥ (การหาพิสัยที่ไม่ถูกจำกัด:Range Anti Restriction) เช่น A ⩥B เป็นการหาคู่ลำดับของเซต A ที่มีพิสัยไม่เป็นสมาชิกของเซต B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
nrres(B,[],B).
nrres(Relation,[H|T],Ranantires):- ranantione(H,Relation,R1), nrres
                                   (R1,T,Ranantires).
```

- สัญลักษณ์ ~ (การย้อนความสัมพันธ์:Relational Inverse) เช่น A~ เป็นการหาความสัมพันธ์แบบกลับกันของ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
inv([],[]).
inv([H|T],Inverse):- H=..[A,B,C|D], inv(T,I), append([(C,B)],I,Inverse).
```

- สัญลักษณ์ ◁ ▷ (เซตสัมพันธ์:Relational Image) เช่น A◁B▷เป็นการหาเซตของพิสัยในเซต A ที่มีโดเมน B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
limg(Relation,[],[]).
```

```
limg(Relation,[H|T],Relimage):- domran(H,Relation,R1), limg
                                (T,Relation,R2), append(R1,R2,Relimage).
```

- สัญลักษณ์ ⊕ (การแทนที่:Overriding) เช่น A⊕B เป็นการแทนที่คู่ลำดับในเซต A ด้วยเซต B ที่มีโดเมนเหมือนกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
oplus(F,G,R):-dom(G,R1),ndres(F,R1,R2),cup(R2,G,R).
```

3) กลุ่มลำดับ เป็นกลุ่มตัวดำเนินการที่เกี่ยวกับลำดับ

- สัญลักษณ์ ⌢ (การต่อลำดับ:Concatenation) เช่น A⌢B เป็นการนำลำดับ A มาต่อกับลำดับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
cat(List1,[],List1).
cat([],List2,List2).
cat(List1,List2,List3):- append(List1,List2,List3).
```

- สัญลักษณ์ rev (ย้อนลำดับ:Reversal) เช่น rev A เป็นการย้อนลำดับของ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
rev(List, Reversed) :-rev(List, [], Reversed).
rev([], Reversed, Reversed).
rev([Head|Tail], SoFar, Reversed) :-rev(Tail, [Head|SoFar], Reversed).
```

- สัญลักษณ์ head (ตัวแรก:Head) เช่น head A เป็นการหาสมาชิกตัวแรกของลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
head([Head|Tail],H):- H=Head.
```

- สัญลักษณ์ last (ตัวท้าย:Last) เช่น last A เป็นการหาสมาชิกตัวสุดท้ายของลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
last([Head|Tail], Element) :- last(Tail, Head, Element).
last([], Element, Element).
last([Head|Tail], _, Element) :- last(Tail, Head, Element).
```

- สัญลักษณ์ tail (ส่วนท้าย:Tail) เช่น tail A เป็นการสมาชิกทั้งหมดยกเว้นตัวแรกของลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
tail([Head|Tail],T):- T=Tail.
```

- สัญลักษณ์ front (ส่วนหน้า:Front) เช่น front A เป็นการสมาชิกทั้งหมดยกเว้นตัวสุดท้ายของลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
front([H],[]).
front([H|T],Front):- front(T,F1), append([H],F1,Front).
```

- สัญลักษณ์ ↾ (การแยก:Extraction) เช่น A↾B เป็นการหาสมาชิกลำดับที่ A ในลำดับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
extract(N, List, Element) :-integer(N), !,N >= 1,N1 is N-1,nth0i(N1, List,
Element).
extract(N, List, Element) :-var(N),nth0v(List, Element, 1, N).
```

- สัญลักษณ์ ⌈ (ฟิลเตอร์:Filter) เช่น A ⌈ B เป็นการหาสมาชิกลำดับที่ B ในลำดับ A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
filter([],List,[]).
filter([H|T],List,Filter):-filterone(H,List),filterdel(H,List,List1),
        filter(T,List1,F1),append([H],F1,Filter).
filter([H|T],List,Filter):-filter(T,List,Filter).
```

- สัญลักษณ์ squash (การจัดลำดับ:Compaction) เช่น squash A เป็นการจัดลำดับในลำดับ A ใหม่ ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
squash([T],[T]).
squash([H|T],S):-sort(T,S1),sqin(S1,H,S).
```

- สัญลักษณ์ ⌒/ (การกระจายลำดับ:Distributed Concatenation) เช่น ⌒/A เป็นการนำลำดับในลำดับ A มาเรียงกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
dcat([],[]).
dcat([H|T],List):- dcat(T,List2), append(H,List2,List).
```

4) กลุ่มแบ็ก เป็นกลุ่มตัวดำเนินการที่เกี่ยวกับแบ็ก

- สัญลักษณ์ # (การหาตัวซ้ำ:Multiplicity) เช่น A#B เป็นการหาสมาชิกในแบ็ก A ที่มีจำนวน B ตัว ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
bcount([],Element,N):-N=0.
bcount([H|T],Element,N):- H=Element,bcount(T,Element,Temp),
        N is Temp+1.
bcount([H|T],Element,N):- \+(H=Element), bcount(T,Element,N).
```

- สัญลักษณ์ ⊗ (การคูณแบ็ก:Bags Scaling) เช่น A⊗B เป็นการเพิ่มสมาชิกในแบ็ก B จำนวน A ครั้ง ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
otimes([],N,[]).
otimes([H|T],N,List):- scalbagone(H,N,L1), otimes(T,N,L2), append
                (L1,L2,List).
```

- สัญลักษณ์ in (สมาชิกแบ็ก:Bags Membership) เช่น A in B เป็นการแสดงว่า A เป็นสมาชิกของแบ็ก B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
memberbag(X,[X|_]).
memberbag(X,[_|L]):-memberbag(X,L).
```

- สัญลักษณ์ $\subset$ (แบ็กย่อย:Sub Bags) เช่น A$\subset$B เป็นการแสดงว่า A เป็นแบ็กย่อยของแบ็ก B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
subbag([],L2).
subbag([H|T],L2):- member([H],L2), deleteone(H,L2,L3),subbag(T,L3).
subbag([H|T],L2):- \+(member([H],L2)), fail.
```

- สัญลักษณ์ $\uplus$ (การยูเนียนแบ็ก:Bags Union) เช่น A$\uplus$B เป็นการแสดงนำแบ็ก A และแบ็ก B มารวมกัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
uplus(L1,[],L1).
uplus(L1,[H|T],L):- uplus(L1,T,L3), insertbag(L3,H,L).
```

- สัญลักษณ์ $\uplus$ (การลบแบ็ก:Bags Difference) เช่น A$\uplus$B เป็นการหาสมาชิกที่มีในแบ็ก A แต่ไม่มีในแบ็ก B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
uminus([],L,[]).
uminus([H|T],List,Intersect):- \+ (member([H],List)), uminus(T,List,U1),
                                append([],U1,Intersect).
uminus([H|T],List,Intersect):- member([H],List),deleteone(H,List,L), uminus
                                (T,L,U1), append([H],U1,Intersect).
```

5) กลุ่มอื่น ๆ เป็นกลุ่มตัวดำเนินการที่เกี่ยวกับตัวเลขและภาคแสดงอื่น ๆ

- สัญลักษณ์ + (การบวก:Addition) เช่น A+B เป็นการหาผลบวกของ A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
plus(A,B,C):- C is A+B.
```

- สัญลักษณ์ − (การลบ:Substraction) เช่น A−B เป็นการหาผลลบของ A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
sub(A,B,C):- C is A-B.
```

- สัญลักษณ์ * (การคูณ:Multiplication) เช่น A*B เป็นการหาผลคูณของ A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
mul(A,B,C):-C is A*B.
```

- สัญลักษณ์ div (การหาร:Division) เช่น A div B เป็นการหาผลหารของ A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
div(A,B,C):- C is A/B.
```

- สัญลักษณ์ mod (เศษจากการหาร: Modulus) เช่น A mod B เป็นการหาเศษจากการหารของ A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
mod(A,B,C):- C is A mod B.
```

- สัญลักษณ์ < (น้อยกว่า:Less Than) เช่น A<B เป็นการแสดงว่า A น้อยกว่า B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
lessthan(A,B):-A@<B.
```

- สัญลักษณ์ <= (น้อยกว่าหรือเท่ากับ:Less Than Or Equal) เช่น A<=B เป็นการแสดงว่า A น้อยกว่าหรือเท่ากับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
leq(A,B):-A@=<B.
```

- สัญลักษณ์ > (มากกว่า:Greater Than) เช่น A>B เป็นการแสดงว่า A มากกว่า B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
greathan(A,B):-A@>B.
```

- สัญลักษณ์ >= (มากกว่าหรือเท่ากับ:Greater Than Or Equal) เช่น A>=B เป็นการแสดงว่า A มากกว่าหรือเท่ากับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
geq(A,B):-A@>=B.
```

- สัญลักษณ์ .. (ช่วง:Number Range) เช่น A..B เป็นการหาช่วงระหว่าง A กับ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
upto(A,B,C):-integer(A),A@<B,Next is A+1,upto(Next,B,C1),append
        ([A],C1,C).
upto(A,B,C):-A@<B,name(A,Num),Next1 is
        Num+1,Next1=91,Next='a',upto(Next,B,C1),append
        ([A],C1,C).
upto(A,B,C):-A@<B,name(A,Num),Next1 is Num+1,name(Next,
        [Next1]),upto(Next,B,C1),append([A],C1,C)
```

- สัญลักษณ์ min (ค่าต่ำสุด:Minimum) เช่น min A เป็นการหาจำนวนที่น้อยที่สุดในเซต A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
min([Head|Tail], Min) :- min(Tail, Head, Min).
min([], Min, Min).
min([Head|Tail], Element, Min) :-Head >= Element, !,min(Tail, Element,
                                        Min).
min([Head|Tail], _, Min) :-min(Tail, Head, Min).
```

- สัญลักษณ์ max (ค่าสูงสุด:Maximum) เช่น max A เป็นการหาจำนวนที่มากที่สุดในเซต A ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
max([Head|Tail], Max) :- max(Tail, Head, Max).
max([], Max, Max).
max([Head|Tail], Element, Max) :-Head =< Element, !,max(Tail, Element,
                                        Max).
max([Head|Tail], _, Max) :-max(Tail, Head, Max).
```

- สัญลักษณ์ ~ (ฟังก์ชัน:Function Applicable) เช่น A~B เป็นการหาโดเมนหรือพิสัยของคู่ลำดับ ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
function(Relation,Domain,Range):
        var(Domain),fundom(Domain,Relation,Range).
function(Relation,Domain,Range):-funran(Domain,Relation,Range).
```

- สัญลักษณ์ ∈ (สมาชิก:Membership) เช่น A∈B เป็นการแสดงว่า A เป็นสมาชิกของ B ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
member(X,[X|_]).
member(X,[_|L]):-member(X,L).
```

- สัญลักษณ์ \/ (หรือ:Conjunction) เช่น A\/B เป็นการนำภาคแสดง A มา or กัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
lor(P1,P2):-(P1;P2).
```

- สัญลักษณ์ /\ (และ:Disjunction) เช่น A/\B เป็นการนำภาคแสดง A มา and กัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
land(P1,P2):-P1,P2.
```

- สัญลักษณ์ ⟹ (ถ้าแล้ว:Implication) เช่น A⟹B เป็นการนำภาคแสดง A มา Imply กัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
implies(P1,P2):-(P1->P2).
```

- สัญลักษณ์ ⟷ (ก็ต่อเมื่อ:Equivalence) เช่น A⟷B เป็นการนำภาคแสดง A มา Equivalence กัน ซึ่งสร้างเป็นกฎในภาษาโพรล็อกได้ดังนี้

```
iff(P1,P2):-(P1,P2);(\+P1,\+P2).
```

3.3.2) ส่วนการแปลงเป็นโปรแกรมโพรล็อก

ส่วนการแปลงเป็นโปรแกรมโพรล็อก เป็นส่วนที่ทำการแปลงเค้าร่างต่าง ๆ ของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่อยู่ในรูปแทคของลาเทกซ์ให้เป็นโปรแกรมโพรล็อก โดยจะทำการแปลงทีละเค้าร่าง ซึ่งการแปลง 1 เค้าร่างในสัญกรณ์เซดจะได้ 1 กฎในโปรแกรมโพรล็อก

ในสัญกรณ์เซดนั้น จะมีเค้าร่าง 2 แบบคือ เค้าร่างเชิงสัจพจน์ และเค้าร่างเชิงสคีมา เค้าร่างเชิงสัจพจน์นั้นจะใช้ในการประกาศตัวแปรและตัวดำเนินการแบบโกลบอล ดังที่ได้กล่าวแล้ว ซึ่งเมื่อทำการแปลงเป็นโปรแกรมโพรล็อกจะไม่สามารถใช้ได้ เพราะว่าโพรล็อกมีข้อจำกัดคือตัวแปรในกฎใด ๆ ก็จะสามารถใช้ได้ในกฎนั้น ๆ เท่านั้น ดังนั้น ข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่มีเค้าร่างเชิงสัจพจน์จึงไม่สามารถใช้ได้ แต่ผู้ใช้สามารถเปลี่ยนข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่มีเค้าร่างเชิงสัจพจน์ให้เป็นเค้าร่างเชิงสคีมาได้ โดยการเปลี่ยนเค้าร่างเชิงสัจพจน์ให้เป็นเค้าร่างเชิงสคีมาโดยการตั้งชื่อให้กับเค้าร่าง จากนั้น เค้าร่างใดที่ใช้ตัวแปรหรือตัวดำเนินการที่ประกาศไว้ในเค้าร่างเชิงสัจพจน์เดิมก็ทำการรวมเค้าร่างเชิงสคีมาใหม่เข้าไปโดยรวมแบบไม่มีการเปลี่ยนแปลงสถานะเค้าร่าง ก็จะได้ข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่ไม่มีเค้าร่างเชิงสัจพจน์ ตัวอย่างเช่น

ในการแปลงเค้าร่างนั้น ชื่อของเค้าร่างจะกลายเป็นชื่อของกฎ ตัวแปรในส่วนการประกาศตัวแปร ตัวแปรของเค้าร่างที่รวมเข้ามา รวมทั้งตัวแปรสถานะหลังของตัวแปรของเค้าร่างที่รวมเข้ามาแบบมีการเปลี่ยนแปลงสถานะของเค้าร่างจะกลายเป็นอาร์กิวเมนต์ของกฎ และภาคแสดงต่าง ๆ ก็จะกลายเป็นเงื่อนไขของกฎ

โดยปกติ ภาคแสดงในสัญกรณ์เซดจะประกอบด้วยตัวแปร และตัวดำเนินการต่าง ๆ สามารถแบ่งออกได้เป็น ประเภทคือ ตัวดำเนินการที่มี 1 ตัวแปร, ตัวดำเนินการที่มี 2 ตัวแปร และตัวบ่งปริมาณ

- ตัวดำเนินการที่มี 1 ตัวแปร คือตัวดำเนินการที่มีตัวแปรที่เป็นข้อมูลเข้า 1 ตัวแปร และมีตัวแปรเป็นข้อมูลออก 1 ตัวแปร โดยชื่อของเงื่อนไขที่สร้างจากตัวดำเนินการนั้นจะต้องสอดคล้องกับไลบราลีที่ได้สร้างไว้ ตัวอย่างของตัวดำเนินการที่มี 1 ตัวแปรเช่น สัญลักษณ์ dom (โดเมน: Domain) สัญลักษณ์ ran (พิสัย:Range) สัญลักษณ์ ¬ (นิเสธ:Not) เป็นต้น ตัวอย่างการแปลงภาคแสดงที่มีตัวดำเนินการ 1 ตัวแปรได้แก่

  a = ¬ b แปลงได้เป็น lnot(B,A)

  c = dom d แปลงได้เป็น dom(D,C)

- ตัวดำเนินการที่มี 2 ตัวแปร คือตัวดำเนินการที่มีตัวแปรที่เป็นข้อมูลเข้า 2 ตัวแปร และมีตัวแปรเป็นข้อมูลออก 1 ตัวแปร โดยชื่อของเงื่อนไขที่สร้างจากตัวดำเนินการนั้นจะต้องสอดคล้องกับไลบราลีที่ได้สร้างไว้ ตัวอย่างของตัวดำเนินการที่มี 2 ตัวแปรเช่น สัญลักษณ์ ∈ (สมาชิก:Member) สัญลักษณ์ ⊂ (เซตย่อย:Subset) สัญลักษณ์ ∪ (ยูเนียน:Union) เป็นต้น ตัวอย่างการแปลงภาคแสดงที่มีตัวดำเนินการ 2 ตัวแปร

  a = b ∈ c แปลงได้เป็น in(B,C,A)

  d = e ∪ f แปลงได้เป็น cup(E,F,D)

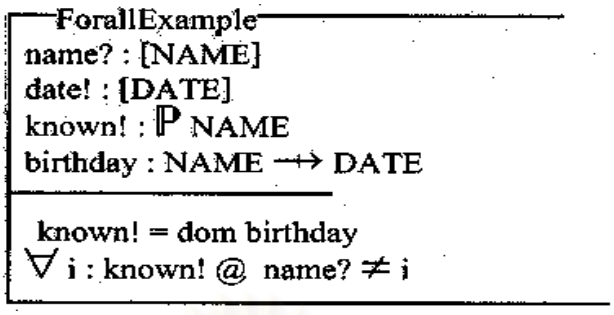
- ตัวบ่งปริมาณ ∀ (Forall) ∃ (Forsome)
  - Forall จะสนใจสมาชิกทั้งหมดในเซตใดเซตหนึ่งว่าเป็นไปไปตามเงื่อนไขที่กำหนดไว้หรือไม่ ตัวดำเนินการ forall จะแบ่งภาคแสดงออกเป็น 2 ส่วนคือ ส่วนประกาศตัวแปร และส่วนเงื่อนไข ส่วนประกาศตัวแปรจะเป็นส่วนที่ใช้ในการกำหนดเซตของข้อมูลที่จะใช้ในการตรวจสอบ ส่วนเงื่อนไขจะเป็นส่วนที่ใช้ในการตรวจสอบว่าข้อมูลในส่วนประกาศตัวแปรนั้นทุกตัวเป็นไปตามเงื่อนไขหรือไม่ ถ้าข้อมูลทุกตัวเป็นไปตามเงื่อนไขทั้งหมดภาคแสดงนี้ก็จะให้ค่าจริง แต่ถ้ามีข้อมูลใดข้อมูลหนึ่งไม่เป็นไปตามเงื่อนไข ภาคแสดงนี้ก็จะให้ค่าเท็จทันที ตัวดำเนินการ Forall จะมีรูปแบบดังต่อไปนี้

    ∀ declaration part @ condition part

    สามารถแปลงได้เป็น

    \+((declaration part), \+ (conditionpart))

ตัวอย่างเช่น



สามารถแปลงได้เป็น

```
forallExample(Name,Date,Known,Birthday):-
        dom(Birthday,Known),\+((in(I,Known),\+(neq(Name,I)))).
```

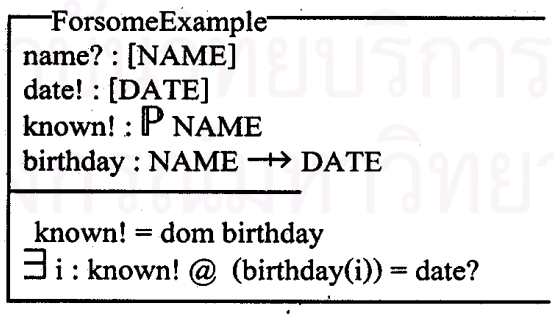
— Forsome จะสนใจสมาชิกบางตัวในเซตใดเซตหนึ่งว่าเป็นไปตามเงื่อนไขที่กำหนดไว้หรือไม่ตัวดำเนินการ Forsome จะแบ่งภาคแสดงออกเป็น 2 ส่วนคือ ส่วนประกาศตัวแปร และส่วนเงื่อนไข ส่วนประกาศตัวแปรจะเป็นส่วนที่ใช้ในการกำหนดเซตของข้อมูลที่จะใช้ในการตรวจสอบ ส่วนเงื่อนไขจะเป็นส่วนที่ใช้ในการตรวจสอบว่าข้อมูลในส่วนประกาศตัวแปรนั้นมีตัวใดตัวหนึ่งเป็นไปตามเงื่อนไขหรือไม่ ถ้ามีข้อมูลตัวใดตัวหนึ่งเป็นไปตามเงื่อนไขภาคแสดงนี้ก็จะให้ค่าจริง แต่ถ้าข้อมูลทุกตัวในเซตไม่เป็นไปตามเงื่อนไข ภาคแสดงนี้ก็จะให้ค่าเท็จทันที ดำเนินการ Forsome จะมีรูปแบบดังต่อไปนี้

∃ declaration part @ condition part

สามารถเป็นได้เป็น

(declaration part),(condition part)

ตัวอย่างเช่น



สามารถแปลงได้เป็น

```
forsomeExample(Name,Date,Known,Birthday):-
dom(Birthday,Known),in(I,Known),function(Birthday,I,Temp1),Temp1=Date.
```

ขั้นตอนวิธีการแปลงเป็นโปรแกรมโพรล็อก(Z-to-Prolog Transforming Algorithm)

ให้ SchName เป็นชื่อของเค้าร่าง

Var เป็นตัวแปรของเค้าร่าง SchName

P เป็นภาคแสดงของเค้าร่าง SchName

InSchUn เป็นเค้าร่างที่ถูกรวมแบบสถานะไม่เปลี่ยนแปลง

InUnVar เป็นตัวแปรของเค้าร่าง InSchUn

InSchChange เป็นเค้าร่างที่ถูกรวมแบบสถานะเปลี่ยนแปลง

InChVar เป็นตัวแปรของเค้าร่าง InSchChange

ในการแปลงมีขั้นตอนดังต่อไปนี้

ก) ให้ SchName เป็นชื่อของกฎ

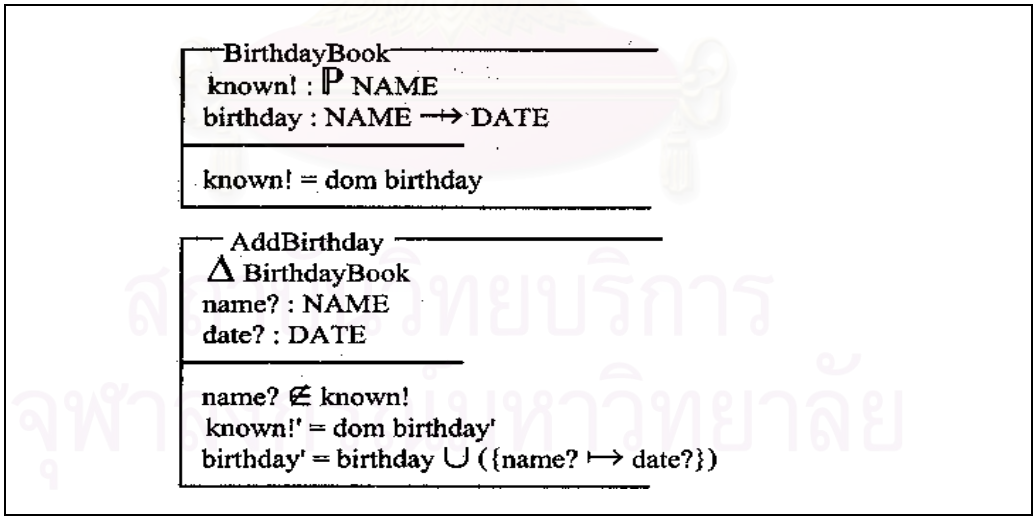
ข) ให้ Var เป็นตัวแปรของกฎ

ค) ถ้ามีการรวมเค้าร่าง InSchUn ให้ InUnvar เป็นตัวแปรของกฎด้วย

ง) ถ้ามีการรวมเค้าร่าง InSchChange ให้ InChVar และสถานะหลังของ InChVar เป็นตัวแปรของกฎด้วย

จ) ทำการแปลง P

ตัวอย่างการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตเป็นโปรแกรมโพรล็อก



รูปที่ 3.2 ตัวอย่างข้อกำหนดรูปนัยในรูปสัญกรณ์เซต

จากตัวอย่างข้อกำหนดรูปนัยในรูปสัญกรณ์เซตจากรูปที่ 3.2 สามารถแปลงเป็นโปรแกรมโพรล็อกได้ดังรูปที่ 3.3

```
birthdayBook(Birthday,Known):- dom(Birthday,Known).
addBirthday(Name,Date,Birthday,Known,Birthdayp,Knownp):- birthdayBook
    (Birthday,Known),notin(Name,Known),Temp1 = [(Name, Date)],cup
    (Birthday,Temp1, Birthdayp),dom(Birthdayp,Knownp).
```

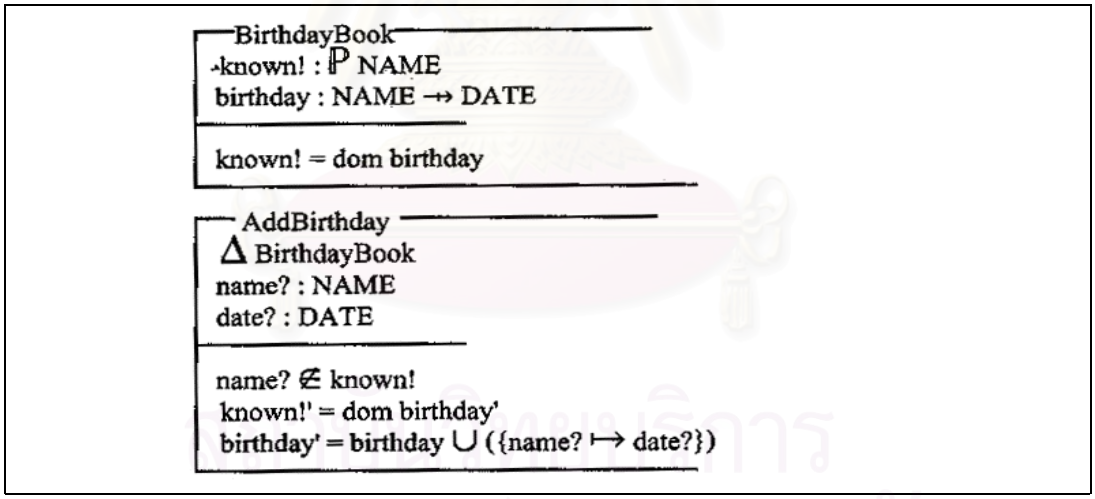
รูปที่ 3.3 ตัวอย่างโปรแกรมโพรล็อก

บทที่ 4
การพัฒนาเครื่องมือซอฟต์แวร์สร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

ในการทำงานของระบบสามารถแบ่งได้เป็น 3 ส่วน ได้แก่ ส่วนข้อมูลเข้า ส่วนสร้างโปรแกรมโพรล็อก และส่วนบันทึกข้อมูล

4.1. ส่วนข้อมูลเข้า

ส่วนรับข้อมูลเป็นส่วนเริ่มต้นการทำงานโดย การรับชื่อของแฟ้มข้อมูลซึ่งจะใช้ในการทำงานโดยลักษณะของข้อมูลนำเข้าจะอยู่ในรูปของแฟ้มข้อความ (Text File) รูปแบบแฟ้มข้อมูลจะอยู่ในรูปแทคของลาเทกซ์ตามแบบของ Z/EVES ดังแสดงในภาคผนวก ง. และแต่ละแฟ้มข้อมูลจะต้องมีนามสกุลเป็น .TEX เค้าร่างแต่ละเค้าร่างจะเขียนเรียงต่อกันไป สำหรับบรรทัดที่เป็นคำอธิบาย (Comment) นั้นจะมีเครื่องหมาย % (Percent) นำหน้าบรรทัด จากรูปที่ 4.1 แสดงตัวอย่างของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ซึ่งสามารถเปลี่ยนให้อยู่ในรูปแทคของลาเทกซ์ ได้ดังรูปที่ 4.2



รูปที่ 4.1 แสดงตัวอย่างข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

4.2. ส่วนสร้างโปรแกรมโพรล็อก

ส่วนสร้างโปรแกรมโพรล็อกสามารถแบ่งออกได้เป็น 2 ขั้นตอนใหญ่ๆ คือ ส่วนการเรียงลำดับภาคแสดง และส่วนการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก สำหรับส่วนการเรียงลำดับภาคแสดงนั้น จะเป็นส่วนที่ทำการจัดรูปแบบของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดก่อนที่จะนำไปแปลงเป็นโปรแกรมโพรล็อก โดยจะทำการเรียงลำดับของภาคแสดงในส่วนสัจพจน์ให้ถูกต้องก่อนจึงจะ

ส่งข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่มีลำดับภาคแสดงในส่วนสัจพจน์ถูกต้องแล้วนั้น ไปแปลงเป็นโปรแกรมโพรล็อกในส่วนการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก โดยในส่วนนี้จะต้องมีการแปลงให้สอดคล้องกับไลบราลีของโพรล็อกที่ได้สร้างไว้แล้ว ขั้นตอนการทำงานของส่วนการเรียงลำดับภาคแสดง และส่วนการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อกมีดังต่อไปนี้

1) รับชื่อแฟ้มข้อมูลของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด
2) อ่านข้อมูลจากแฟ้มข้อมูลของข้อกำหนดรูปนัยในรูปสัญกรณ์เซด
3) ทำการเรียงลำดับภาคแสดงของแต่ละเค้าร่าง
4) ทำการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อกทีละเค้าร่าง
5) รับชื่อแฟ้มข้อมูลผลลัพธ์
6) บันทึกโปรแกรมโพรล็อกลงในแฟ้มผลลัพธ์

```
\begin{schema}{BirthdayBook}
known! : \power NAME  \\
birthday : NAME \pfun DATE
\where
  known! = \dom birthday
\end{schema}

\begin{schema}{AddBirthday}
\Delta BirthdayBook  \\
name? : NAME  \\
date? : DATE
\where
name? \notin known!
known!' = \dom birthday'
birthday' = birthday \cup (\{name? \mapsto date?\})
\end{schema}
```

รูปที่ 4.2 แสดงตัวอย่างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดในรูปแทคของลาเทกซ์

4.3. ส่วนบันทึกข้อมูล

ส่วนบันทึกข้อมูลเป็นส่วนที่ทำการบันทึกข้อมูลที่ได้จากส่วนการแปลงสัญกรณ์เซดเป็นภาษาโพรล็อกลงในแฟ้มข้อมูลที่ต้องการ โดยจะอยู่ในรูปแฟ้มตัวอักษร ซึ่งแบ่งเป็นกฎต่าง ๆ โดย 1 เค้าร่างจะ

สามารถแปลงได้เป็น 1 กฎของโปรแกรมโพรล็อก ซึ่งแฟ้มข้อมูลสำหรับบันทึกข้อมูลนี้ จะมีนามสกุลเป็น .PL ตัวอย่างโปรแกรมโพรล็อกแสดงดังรูปที่ 4.3

```
:-use_module(library(z)).
birthdayBook(Birthday,Known):- dom(Birthday,Known).
addBirthday(Name,Date,Birthday,Known,Birthdayp,Knownp):- birthdayBook
    (Birthday,Known),notin(Name,Known),Temp1 = [(Name ,
    Date)],cup(Birthday,Temp1,Birthdayp),dom(Birthdayp,Knownp).
```

รูปที่ 4.3 แสดงตัวอย่างโปรแกรมโพรล็อกที่ได้จากการแปลง

4.4. ผังโครงสร้างของระบบ

จากขั้นตอนการทำงานที่กล่าวมาทั้งหมด สามารถแสดงให้เห็นถึงภาพรวมของการทำงานของระบบได้ทั้งหมด เริ่มตั้งแต่การรับชื่อแฟ้มข้อมูล ไปจนถึงการแสดงผลลัพธ์โปรแกรมโพรล็อก ซึ่งสามารถแสดงเป็นผังโครงสร้างได้ดังรูปที่ 4.4

จากรูปที่ 4.4 ซึ่งแสดงภาพรวมของการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้น สามารถแบ่งการทำงานของส่วนต่าง ๆ ออกเป็นโมดูลย่อย ๆ ได้ ดังแสดงในรูปที่ 4.5 ซึ่งสามารถอธิบายการทำงานของแต่ละโมดูลได้ดังต่อไปนี้

โมดูล 1 รับชื่อแฟ้มข้อมูล

โมดูลรับชื่อแฟ้มข้อมูล คือการรับชื่อแฟ้มข้อมูลข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

โมดูล 2 อ่านข้อมูล

โมดูลอ่านข้อมูล คือ การอ่านข้อมูลจากแฟ้มข้อมูลข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเข้ามาในระบบ

โมดูล 3 สร้างโปรแกรมภาษาโพรล็อก

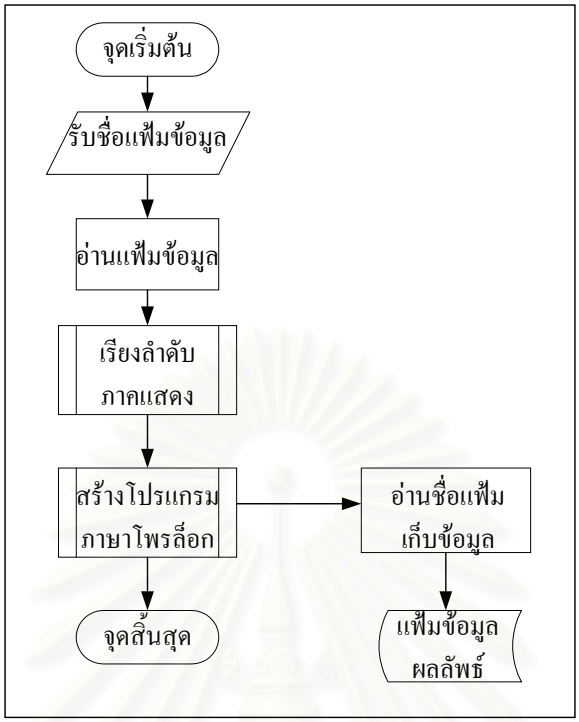
โมดูลสร้างโปรแกรมภาษาโพรล็อกคือ ส่วนการเรียงลำดับภาคแสดง และแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก

โมดูล 3.1 เรียงลำดับภาคแสดง

โมดูลเรียงลำดับภาคแสดงคือ ส่วนที่ทำการเรียงลำดับภาคแสดงในส่วนสัจพจน์ของแต่ละเค้าร่างให้อยู่ในลำดับที่ถูกต้อง

โมดูล 3.2 แปลงเป็นโปรแกรมโพรล็อก

โมดูลแปลงเป็นโปรแกรมโพรล็อกคือ ส่วนที่ทำการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดให้เป็นโปรแกรมโพรล็อก โดยจะทำการแปลงทีละเค้าร่างของข้อกำหนด



รูปที่ 4.4 แสดงผังงานการสร้างโปรแกรมโพรล็อกจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

โมดูล 4 บันทึกผลลัพธ์

โมดูลบันทึกผลลัพธ์ คือ ส่วนที่ทำการรับชื่อไฟล์ที่ต้องการบันทึกผลลัพธ์ และทำการบันทึกผลลัพธ์ลงในไฟล์นั้น ๆ

โมดูล 4.1 รับชื่อแฟ้มข้อมูล

โมดูลรับชื่อแฟ้มข้อมูลจะทำการรับชื่อแฟ้มข้อมูลที่ต้องการบันทึกเข้ามาในระบบ

โมดูล 4.2 บันทึกข้อมูล

โมดูลบันทึกข้อมูล ทำการบันทึกผลลัพธ์ลงในแฟ้มข้อมูล

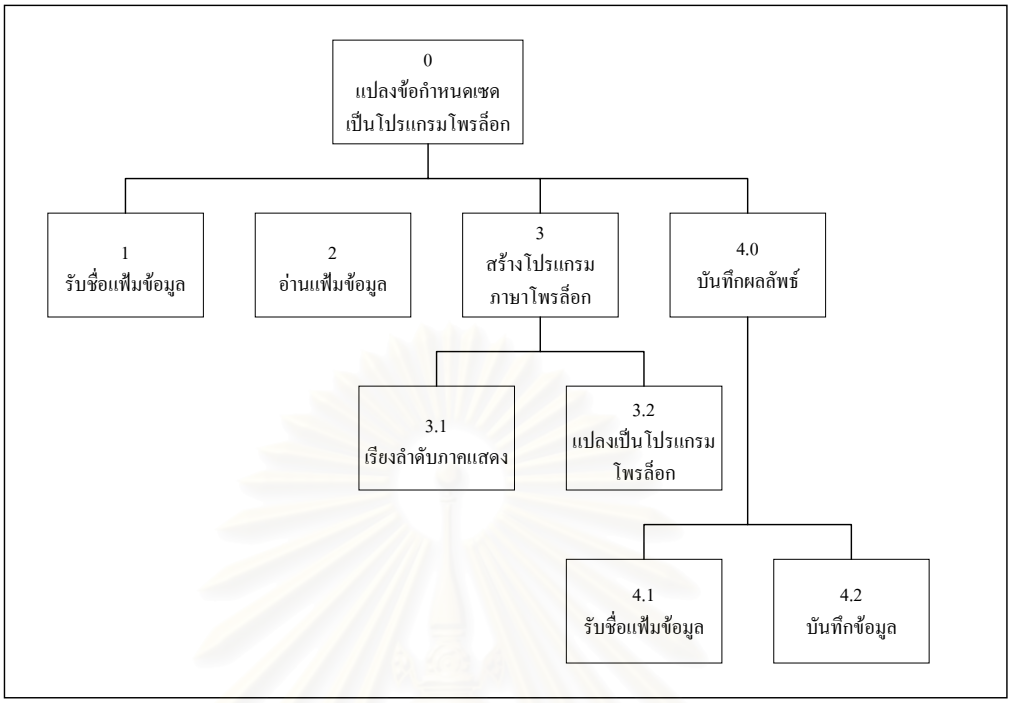
4.5. สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือซอฟต์แวร์

เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนามีรายละเอียดดังนี้

- คอมพิวเตอร์แบบพีซี PentiumII 450 MHz.
- หน่วยความจำ 64 เมกะไบต์
- ฮาร์ดดิสต์ 6 กิกะไบต์

ซอฟต์แวร์ที่ใช้ในการพัฒนามีรายละเอียดดังนี้

- ระบบปฏิบัติการวินโดว์ส98
- เดลไฟล์ 4.0



รูปที่ 4.5 ผังโครงสร้างแสดงส่วนการทำงานต่าง ๆ ของระบบ

บทที่ 5
การทดสอบ และสรุปผล

ในบทนี้จะกล่าวถึงการทดสอบเครื่องมือแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก ภายใต้ระบบปฏิบัติการวินโดว์ส 98

5.1. ขั้นตอนการติดตั้ง

ทำการติดตั้งเครื่องมือนี้โดยการเรียก SETUP.EXE แล้วปฏิบัติตามขั้นตอนที่โปรแกรมแนะนำเพื่อเป็นการติดตั้งเครื่องมือ

5.2. สภาวะที่ใช้ทดสอบโปรแกรม

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบมีรายละเอียดดังนี้

- คอมพิวเตอร์แบบพีซี Pentium II 450 เมกะเฮิร์ส
- หน่วยความจำ 64 เมกะไบต์
- ฮาร์ดดิสต์ 6 กิกะไบต์

5.3. กรณีทดสอบที่ใช้ทดสอบโปรแกรม

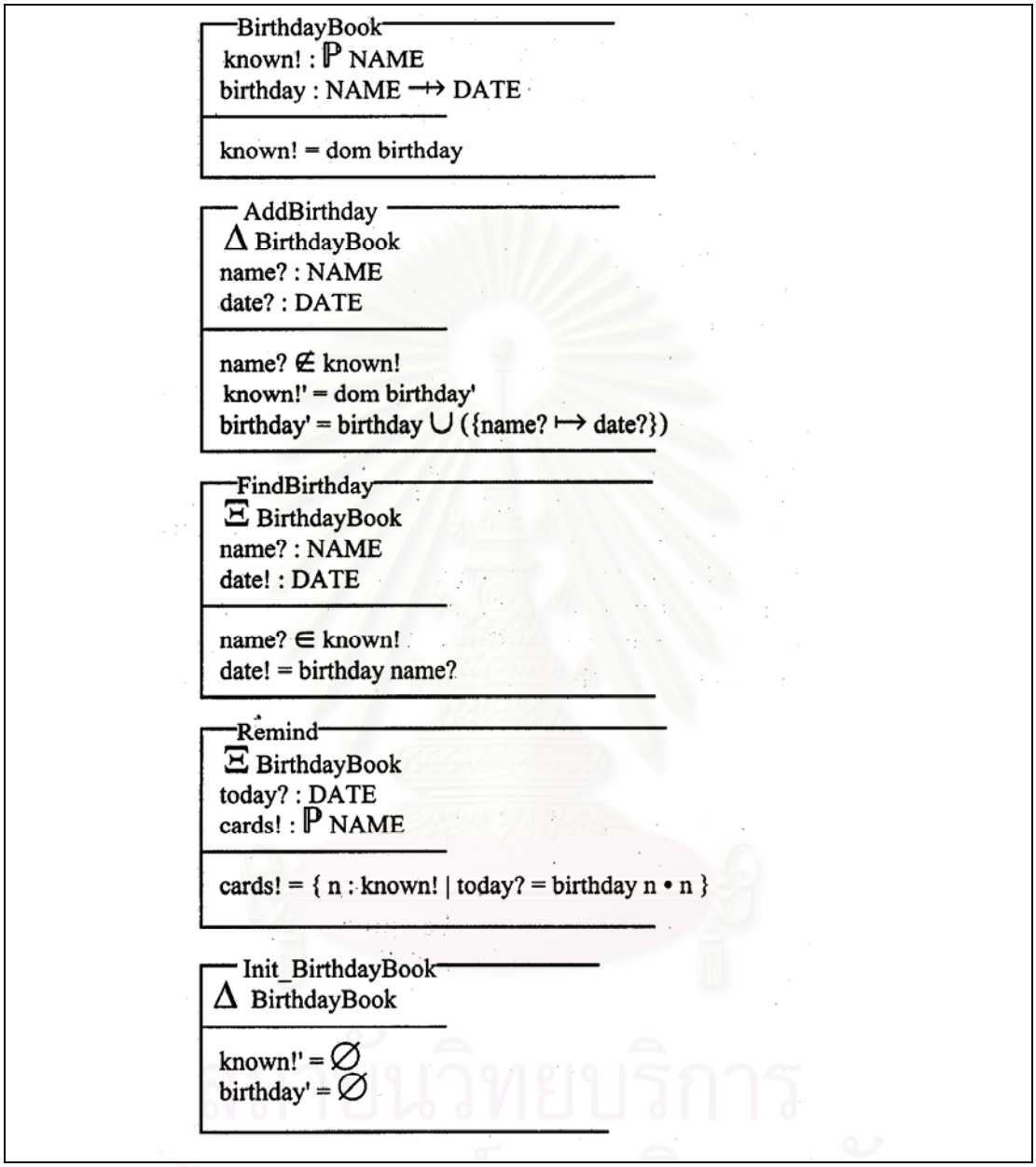
1) ระบบ BirthdayBook

ระบบ Birthday Book เป็นระบบที่ใช้ในการเก็บชื่อและวันเกิดของบุคคล มีลักษณะคล้ายกับสมุดบันทึกวันเกิด โดยแต่ละคนจะมีวันเกิดได้ 1 วัน และชื่อคนที่บันทึกนั้นห้ามซ้ำกัน โดยระบบนี้สามารถเพิ่มและค้นหาวันเกิดของคนได้ นอกจากนี้ยังสามารถบอกได้ว่าวันนี้เป็นวันเกิดของใครหรือไม่ ข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ Birthday Book แสดงอยู่ในภาคผนวก ก. ตัวอย่างของสัญกรณ์เซดของระบบ Birthday Book แสดงอยู่ในรูปที่ 5.1

2) ระบบ PDB (Phone Database System)

เป็นระบบที่ใช้ในการเก็บชื่อและเบอร์โทรศัพท์ของบุคคล มีลักษณะคล้ายกับสมุดโทรศัพท์ ผู้ที่จะมีชื่อและเบอร์โทรศัพท์อยู่ในระบบได้จะต้องสมัครเป็นสมาชิกก่อน สมาชิกแต่ละคนจะมีเบอร์โทรศัพท์ได้หลายเบอร์ และเบอร์โทรศัพท์ 1 เบอร์ก็อาจเป็นได้ของสมาชิกหลาย ๆ คน โดยระบบนี้สามารถเพิ่มและค้นหาเบอร์โทรศัพท์ได้ นอกจากนี้ยังสามารถบอกได้ว่าเบอร์โทรศัพท์นี้เป็นของใคร ข้อกำหนดรูปนัยใน

รูปสัญกรณ์เซตของระบบ PDB แสดงอยู่ในภาคผนวก ข. ตัวอย่างของสัญกรณ์เซตของระบบ PDB แสดงอยู่ในรูปที่ 5.2

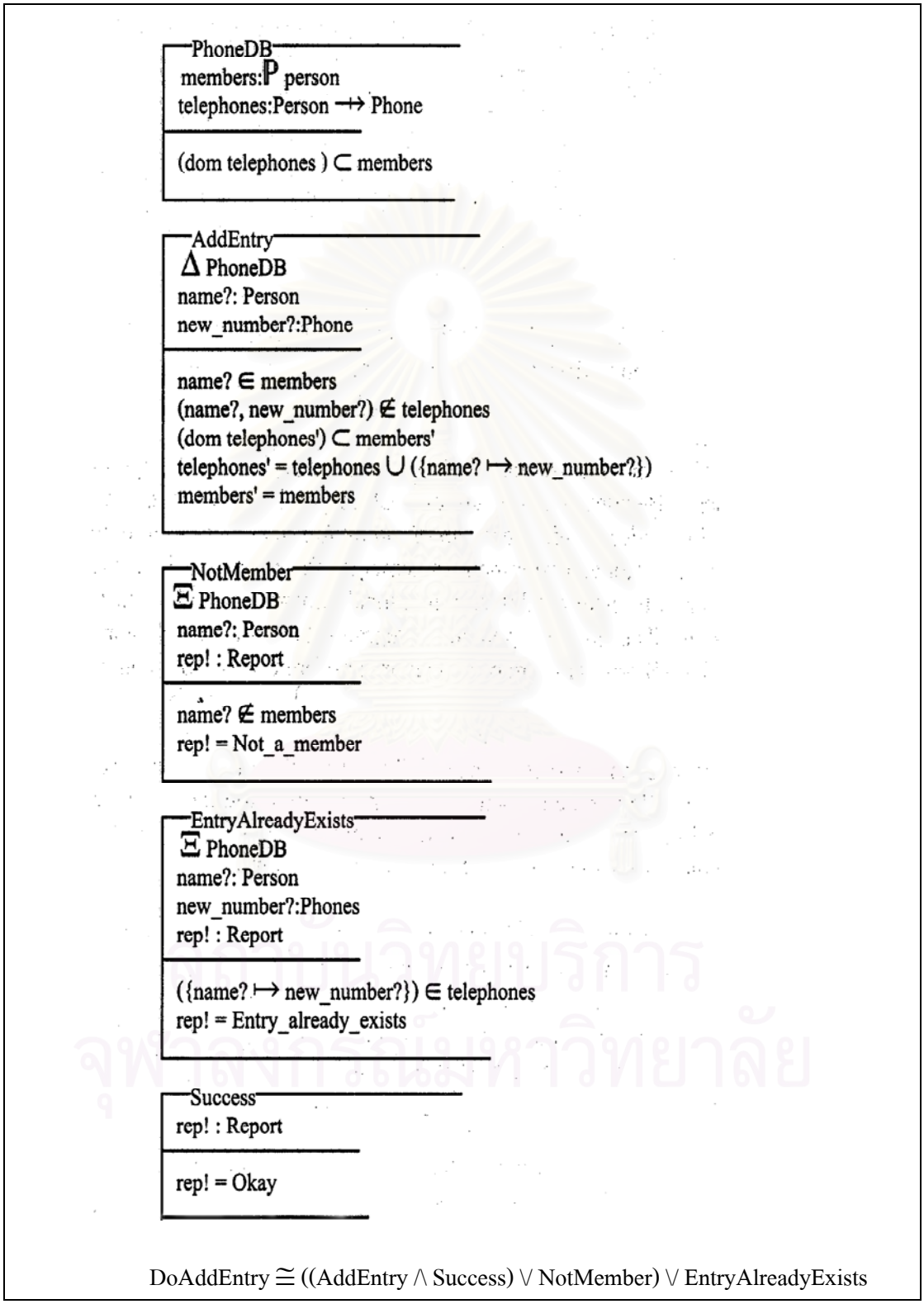


รูปที่ 5.1 แสดงตัวอย่างของข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ Birthday Book

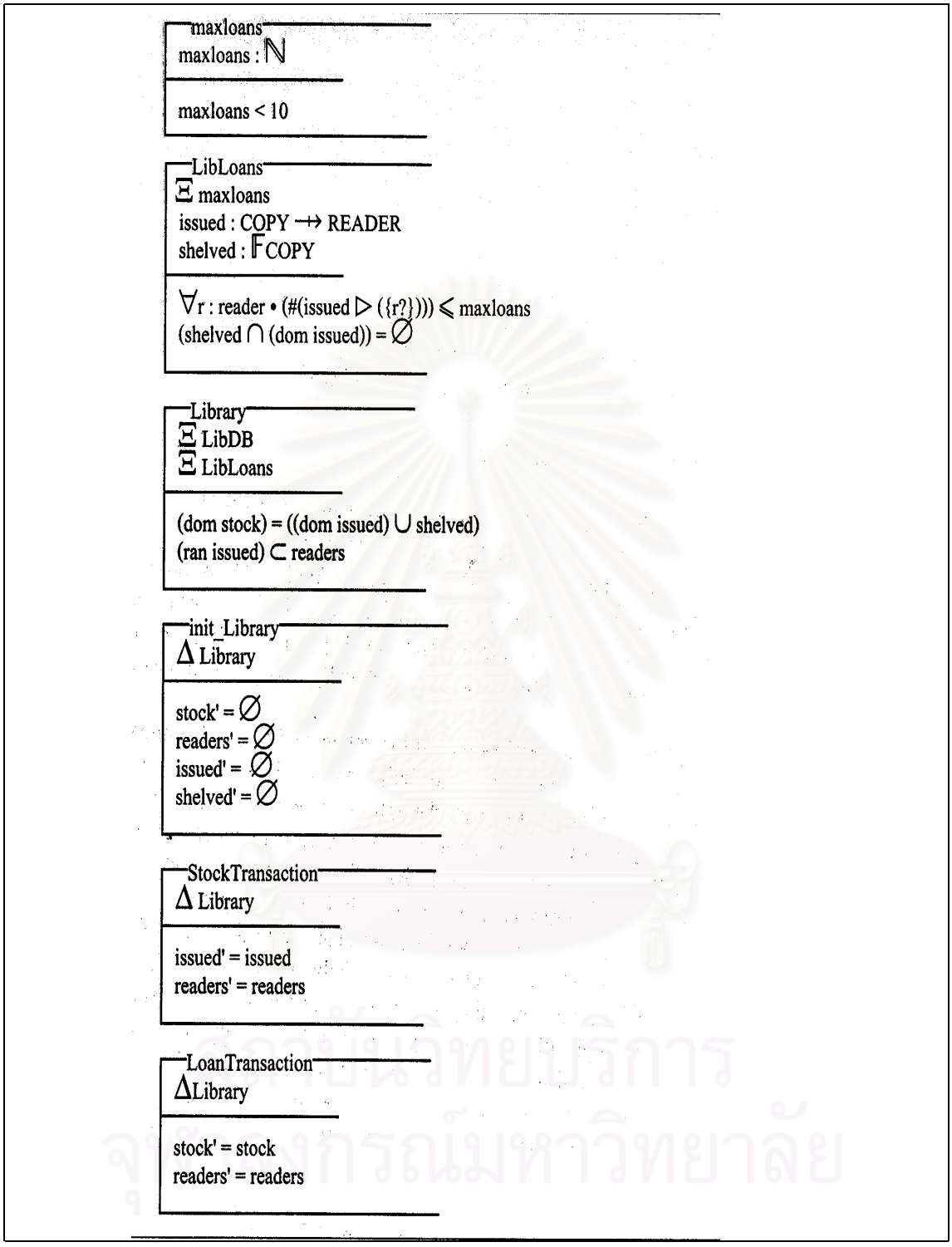
3) ระบบห้องสมุด (Library)

เป็นระบบห้องสมุดทั่ว ๆ ไปที่สามารถมีการสมัครเข้าเป็นสมาชิกเพื่อทำการยืมคืนหนังสือได้ โดยสามารถกำหนดจำนวนหนังสือที่สามารถยืมได้สูงสุด นอกจากนี้ ยังสามารถทำการเพิ่มหนังสือใหม่ ๆ เข้าห้องสมุดได้ หรือตรวจสอบว่าหนังสือเล่มใดอยู่ที่ใคร และใครยืมหนังสือเล่มใดไปบ้าง ข้อกำหนดรูปนัยใน

รูปสัญกรณ์เซตของระบบห้องสมุด แสดงอยู่ในภาคผนวก ค. ตัวอย่างของสัญกรณ์เซตของระบบห้องสมุด แสดงอยู่ในรูปที่ 5.3



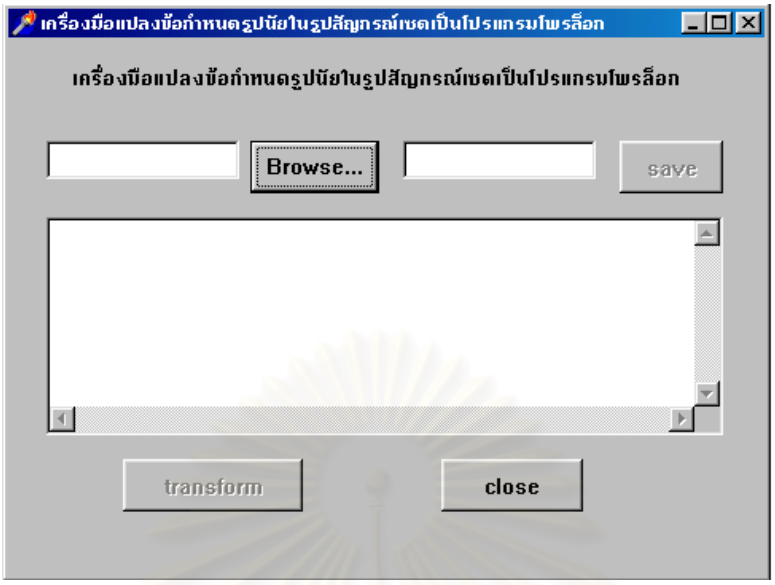
รูปที่ 5.2 แสดงตัวอย่างของข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ PDB



รูปที่ 5.3 แสดงตัวอย่างของข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุด

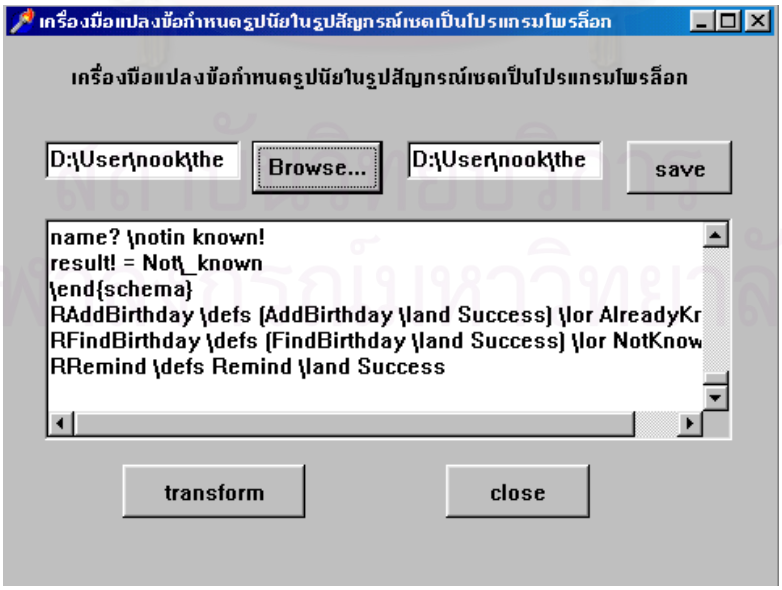
5.4. ขั้นตอนการทดสอบ

1) เรียกโปรแกรม เพื่อทำการทดสอบ ดังรูปที่ 5.4

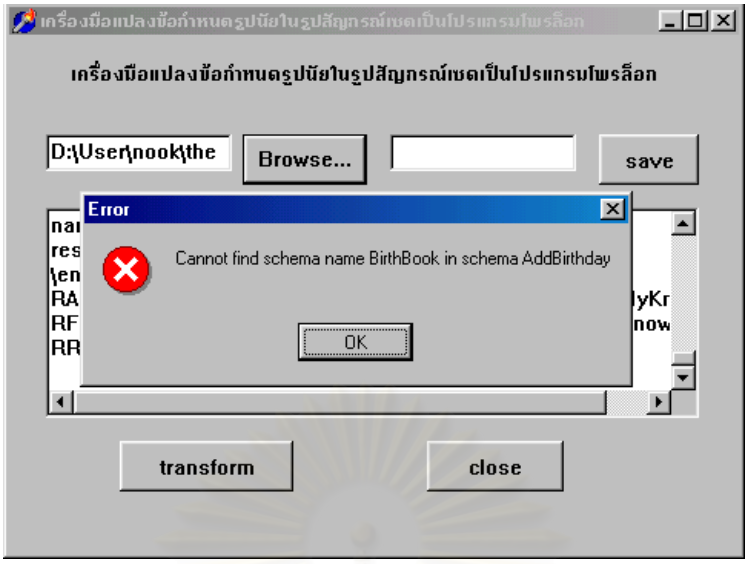


รูปที่ 5.4 หน้าจอเครื่องมือ

2) ใส่ชื่อแฟ้มข้อกำหนดรูปนัยในรูปสัญกรณ์เซต(*.TEX)
3) กดปุ่ม เพื่อสร้างโปรแกรมโพรล็อก
   - ถ้าหากข้อมูลที่ได้รับมีความสมบูรณ์ สามารถสร้างโปรแกรมโพรล็อกได้อย่างสมบูรณ์ จะได้ผลดังรูปที่ 5.5
   - ถ้าหากข้อมูลที่ได้รับมีบางส่วนไม่สมบูรณ์ทำให้ไม่สามารถสร้างโปรแกรมโพรล็อกได้อย่างสมบูรณ์ จะได้ผลดังรูปที่ 5.6



รูปที่ 5.5 แสดงการทำงานของเครื่องมือถ้าข้อมูลนำเข้าสมบูรณ์



รูปที่ 5.6 แสดงการทำงานของเครื่องมือถ้าข้อมูลนำเข้าไม่สมบูรณ์

5.5. ผลการทดสอบโปรแกรม

จากการทดสอบพบว่าเครื่องมือนี้สามารถทำการแปลงข้อมูลจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซตเป็นโปรแกรมโพรล็อก โดยอยู่ในรูปของกฎ ทั้งนี้ จากการทดสอบกับระบบ Birthday Book สามารถสร้างเป็นกฎต่าง ๆ ได้ทั้งหมด 12 กฎ ตัวอย่างโปรแกรมโพรล็อกของระบบ Birthday Book แสดงในรูปที่ 5.7 ซึ่งรายละเอียดของโปรแกรมทั้งหมดแสดงอยู่ในภาคผนวก ก.

```
1  :-use_module(library(z)).
2  birthdayBook(Birthday,Known):- dom(Birthday,Known).
3  addBirthday(Name,Date,Birthday,Known,Birthdayp,Knownp):- birthdayBook
     (Birthday,Known),notin(Name,Known),Temp1 = [(Name , Date)],cup
     (Birthday,Temp1,Birthdayp),dom(Birthdayp,Knownp).
4  findBirthday(Name,Birthday,Known,Date):- birthdayBook(Birthday,Known),in
     (Name,Known),function(Birthday,Name,Date).
5  remind(Today,Birthday,Known,Cards):- birthdayBook(Birthday,Known),Temp2 =
     Known,is2((function(Birthday,N,Temp4),Today = Temp4),Temp3),is2((Temp6 =
     N,Cards = Temp6),Temp5),land(Temp3,Temp5).
6  init_BirthdayBook(Birthday,Known,Knownp,Birthdayp):- birthdayBook
     (Birthday,Known),Knownp=[],Birthdayp=[].
```

รูปที่ 5.7 แสดงตัวอย่างโปรแกรมโพรล็อกของระบบ Birthday Book

จากตัวอย่างโปรแกรมโพรล็อกจากรูปที่ 5.7 มีกฎทั้งหมด 6 กฎ โดยกฎที่ 1 จะเป็นกฎที่เรียกใช้ไลบราลีภาษาโพรล็อกที่ได้สร้างขึ้น ส่วนกฎอื่น ๆ ที่เหลือนอกจากกฎที่ 1 จะเป็นการแสดงฟังก์ชัน (Function) การทำงานของโปรแกรมโพรล็อกของระบบ Birthday Book ซึ่งจะเป็นการแสดงฟังก์ชันการทำงานของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ Birthday Book เช่นกัน และตัวอย่างนี้โปรแกรมโพรล็อกสามารถทำ birthdayBook, addBirthday, findBirthday,remind และ init_BirthdayBook ได้ โดยในกฎ addBirthday นั้นจะมีลำดับการทำงานของเงื่อนไขที่ต่างจากต้นฉบับเค้าร่าง AddBirthday เพราะว่าลำดับภาคแสดงของต้นฉบับเค้าร่าง AddBirthday ไม่ถูกต้อง และหลังจากผ่านขั้นตอนการเรียงลำดับภาคแสดงแล้ว เมื่อทำการแปลงก็จะได้ผลลัพธ์โปรแกรมโพรล็อกที่สามารถทำงานได้

จากการทดสอบกับกรณีทดสอบที่ 2 ระบบ PDB สามารถสร้างเป็นกฎต่าง ๆ ได้ทั้งหมด 34 กฎ ตัวอย่างโปรแกรมโพรล็อกของระบบ PDB แสดงในรูป 5.8 ซึ่งรายละเอียดของโปรแกรมทั้งหมดแสดงอยู่ในภาคผนวก ข.

```
1   :-use_module(library(z)).
2   phoneDB(Members,Telephones):- dom(Telephones,Temp1),subset(Temp1,Members).
3   addEntry(Name,New_number,Members,Telephones,Telephonesp,Membersp):-
        phoneDB(Members,Telephones),Temp2 = [(Name , New_number)], notin
        (Temp2,Telephones), Temp3 = [(Name ,
        New_number)],cup(Telephones,Temp3,Telephonesp),Membersp = Members,dom
        (Telephonesp,Temp4),subset(Temp4,Membersp).
4   notMember(Name,Members,Telephones,Rep):-
        phoneDB(Members,Telephones),notin(Name,Members),Rep = not_a_member.
5   entryAlreadyExists(Name,New_number,Members,Telephones,Rep):- phoneDB
        (Members,Telephones),Temp5 = [(Name ,
        New_number)],in(Temp5,Telephones),Rep = entry_already_exists.
6   success(Rep):- Rep = okay.
```

รูปที่ 5.8 แสดงตัวอย่างโปรแกรมโพรล็อกของระบบ PDB

จากตัวอย่างโปรแกรมโพรล็อกจากรูปที่ 5.8 มีกฎทั้งหมด 6 กฎ โดยกฎที่ 1 จะเป็นกฎที่เรียกใช้ไลบราลีภาษาโพรล็อกที่ได้สร้างขึ้น ส่วนกฎอื่น ๆ ที่เหลือนอกจากกฎที่ 1 จะเป็นการแสดงฟังก์ชันการทำงานของโปรแกรมโพรล็อกของระบบ PDB ซึ่งจะเป็นการแสดงฟังก์ชันการทำงานของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB เช่นกัน และตัวอย่างนี้โปรแกรมโพรล็อกสามารถทำ phoneDB, addEntry, notMember, entryAlreadyExists และ success ได้ โดยในกฎ addEntry นั้นจะมีลำดับการทำงานของเงื่อนไขที่ต่างจากต้นฉบับเค้าร่าง AddEntry เพราะว่าลำดับภาคแสดงของต้นฉบับเค้าร่าง AddEntry

ไม่ถูกต้อง และหลังจากผ่านขั้นตอนการเรียงลำดับภาคแสดงแล้ว เมื่อทำการแปลงก็จะได้ผลลัพธ์โปรแกรมโพรล็อกที่สามารถทำงานได้

จากการทดสอบกับกรณีทดสอบที่ 3 ระบบห้องสมุด สามารถสร้างเป็นกฎต่าง ๆ ได้ทั้งหมด 38 กฎ ตัวอย่างโปรแกรมโพรล็อกของระบบห้องสมุด แสดงในรูปที่ 5.9 ซึ่งรายละเอียดของของโปรแกรมทั้งหมด แสดงอยู่ในภาคผนวก ค.

```
1  :-use_module(library(z)).
2  maxloans(Maxloans):- <(Maxloans,10).
3  reader(Reader).
4  copy(Copy).
5  libDB(Stock,Readers).
6  libLoans(Issued,Shelved,Maxloans,Reader):- maxloans(Maxloans),reader
       (Reader),Temp5=(Temp1 = Reader,member(R,Temp1),\+((Temp4 = [R],rres
       (Issued,Temp4,Temp3),#(Temp3,Temp2),leq(Temp2,Maxloans)))),\+(Temp5),dom
       (Issued,Temp7),cap(Shelved,Temp7,Temp6),Temp6=[].
7  library(Stock,Readers,Issued,Shelved,Maxloans,Reader):-
       libDB(Stock,Readers),libLoans(Issued,Shelved,Maxloans,Reader),dom
       (Stock,Temp8),dom(Issued,Temp10),cup(Temp10,Shelved,Temp9),Temp8 =
       Temp9,ran(Issued,Temp11),subset(Temp11,Readers).
8  init_Library(Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,S
       helvedp,Maxloansp,Readerp):- library(Stock,Readers,Issued,Shelved,Maxloans,
       Reader),Stockp=[],Readersp=[],Issuedp=[],Shelvedp=[].
```

รูปที่ 5.9 แสดงตัวอย่างโปรแกรมโพรล็อกของระบบห้องสมุด

จากตัวอย่างโปรแกรมโพรล็อกจากรูปที่ 5.9 มีกฎทั้งหมด 8 กฎ โดยกฎที่ 1 จะเป็นกฎที่เรียกใช้ไลบราลีภาษาโพรล็อกที่ได้สร้างขึ้น ส่วนกฎอื่น ๆ ที่เหลือนอกจากกฎที่ 1 จะเป็นการแสดงฟังก์ชันการทำงานของโปรแกรมโพรล็อกของระบบ PST ซึ่งจะเป็นการแสดงฟังก์ชันการทำงานของข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ PST เช่นกัน และตัวอย่างนี้โปรแกรมโพรล็อกสามารถทำ maxLoans, reader, copy, libDB, libLoans, library และ init_Library ได้

5.6. ผลการทดสอบโปรแกรมโพรล็อกด้วย SICStus™

ในการทดสอบความถูกต้องทางด้านวากยสัมพันธ์ของผลลัพธ์โปรแกรมโพรล็อกจะใช้โปรแกรม SICStus™ ในการทดสอบ โดย SICStus™ จะทำการตรวจสอบเงื่อนไขในโปรแกรมทีละเงื่อนไขเพื่อ

ตรวจสอบวากยสัมพันธ์ของเงื่อนไขนั้น ๆ หากเงื่อนไขใดวากยสัมพันธ์ไม่ถูกต้องก็จะแสดงข้อผิดพลาดออกมา ส่วนเงื่อนไขที่ถูกต้องแล้วสามารถใช้งานได้เฉพาะเงื่อนไขที่ไม่สัมพันธ์กับเงื่อนไขที่ไม่ถูกต้อง

ส่วนการทดสอบความหมายของผลลัพธ์โปรแกรมโพรล็อกจะมีการสร้างกรณีทดสอบเพื่อทดสอบผลลัพธ์ที่ได้ว่ามีความหมายตรงกับข้อกำหนดรูปนัยในรูปสัญกรณ์เซตที่ทำการแปลงหรือไม่ โดยกรณีทดสอบนี้จะทำการสร้างเพื่อทดสอบทุก ๆ กฎของแต่ละระบบและทำการดำเนินงานบนโปรแกรม SICStus™ เพื่อตรวจผลลัพธ์

5.7. การตรวจสอบความหมายของโปรแกรมโพรล็อก

ในการตรวจสอบความหมายของผลลัพธ์โปรแกรมโพรล็อก ได้ทำการตรวจสอบโดยการเปรียบเทียบความหมายของข้อกำหนดรูปนัยในรูปสัญกรณ์เซตและความสัมพันธ์กับผลลัพธ์โปรแกรมโพรล็อก การตรวจสอบจะแบ่งเป็นขั้นตอนย่อย ๆ ดังนี้

1) ขั้นตอนตรวจสอบความหมายของกฎในไลบราลี เป็นการตรวจสอบว่ากฎที่สร้างขึ้นแทนตัวดำเนินการ มีความหมายตรงกับความตัวดำเนินการนั้นหรือไม่ โดยจำนวนกฎจะมีมากกว่าจำนวนตัวดำเนินการ เนื่องจากตัวดำเนินการหนึ่ง ๆ อาจต้องใช้หลาย ๆ กฎประกอบกัน การตรวจสอบในขั้นตอนนี้ จะทำเพียงครั้งเดียว เพราะทุก ๆ ข้อกำหนดรูปนัยในรูปสัญกรณ์เซตสามารถใช้ไลบราลีเดียวกันได้
2) ขั้นตอนตรวจสอบการเรียงลำดับภาคแสดง เป็นการตรวจสอบว่า ในเค้าร่างที่ภาคแสดงมีการเรียงลำดับไม่ถูกต้องนั้น สามารถเรียงลำดับได้ถูกต้องหรือไม่ ซึ่งจะเป็นการตรวจสอบทั้งการแบ่งกลุ่มของตัวแปร และการแบ่งกลุ่มของภาคแสดง โดยจะมีเค้าร่างบางเค้าร่างที่ลำดับของภาคแสดงผิด เพื่อเป็นการทดสอบส่วนการเรียงลำดับภาคแสดงในเค้าร่าง
3) ขั้นตอนการตรวจสอบโครงสร้างของกฎ เป็นการตรวจสอบโครงสร้างของกฎที่ได้สร้างขึ้น ว่าตรงกับเค้าร่างที่ได้ทำการแปลงมาหรือไม่ โดยจะตรวจสอบในส่วนของชื่อ ตัวแปร และจำนวนเงื่อนไขของกฎนั้น ๆ
4) ขั้นตอนการตรวจสอบการรวมเค้าร่าง เป็นการตรวจสอบเค้าร่างที่มีการรวมเค้าร่างอื่นเข้ามา ทั้งในแบบการรวมเค้าร่างแบบมีการเปลี่ยนแปลงสถานะเค้าร่าง และการรวมแบบไม่มีการเปลี่ยนแปลงสถานะเค้าร่าง โดยในการรวมเค้าร่างแบบมีการเปลี่ยนแปลงสถานะเค้าร่างจะต้องมีการเพิ่มตัวแปรสถานะหลังของเค้าร่างที่รวมเข้ามาด้วย
5) ขั้นตอนการตรวจสอบเงื่อนไขของกฎ เป็นการตรวจสอบเงื่อนไขของกฎแต่ละกฎที่แปลงมาจากภาคแสดงของเค้าร่างว่าครบทุกภาคแสดง และมีความหมายตรงกันหรือไม่ โดยจะเปรียบเทียบภาคแสดงในเค้าร่าง กับเงื่อนไขในกฎนั้น ๆ รวมถึงตรวจสอบความหมายของเงื่อนไขแต่ละเงื่อนไขว่าตรงกับความหมายของภาคแสดงหรือไม่

6) ขั้นตอนการตรวจสอบความหมายของกฎ เป็นการตรวจสอบว่ากฎแต่ละกฎที่แปลงมาจากเค้าร่างใด ๆ มีความหมายตรงกับความหมายของเค้าร่างนั้น ๆ หรือไม่ โดยการดำเนินงานกฎทีละกฎ รวมถึงการทำงานของกฎหลาย ๆ กฎต่อเนื่องกัน

5.7.1. การตรวจสอบความหมายของผลลัพธ์โปรแกรมโพรล็อกของระบบ BirthdayBook

ระบบ Birthday Book ประกอบด้วยเค้าร่างทั้งหมด 12 เค้าร่าง ได้ทำการตรวจสอบตามขั้นตอนในการตรวจสอบ 6 ขั้นตอน ซึ่งได้ผลดังนี้

1) เมื่อตรวจสอบความหมายของกฎในไลบราลี พบว่าความหมายของกฎแต่ละกฎตรงกับความหมายของตัวดำเนินการนั้น ๆ ซึ่งในไลบราลีประกอบด้วยกฎซึ่งสร้างจากตัวดำเนินการทั้งหมด 50 ตัว
2) เมื่อตรวจสอบการเรียงลำดับภาคแสดง พบว่าต้นฉบับเค้าร่าง AddBirthday นั้นมีการเรียงลำดับภาคแสดงไม่ถูกต้อง ในต้นฉบับเค้าร่าง AddBirthday มีภาคแสดงทั้งหมด 3 ภาคแสดง โดยภาคแสดงที่ 2 เป็นภาคแสดงที่อยู่ในลำดับที่ผิด โดยเครื่องมือซอฟต์แวร์สามารถเรียงลำดับภาคแสดงในเค้าร่าง AddBirthday ใหม่ได้ถูกต้อง ซึ่งภาคแสดงที่ 2 ก็จะกลายเป็นภาคแสดงที่ 3 แทน
3) เมื่อตรวจสอบโครงสร้างของกฎพบว่า เครื่องมือซอฟต์แวร์สามารถสร้างกฎได้ทั้งหมด 12 กฎ แต่ละกฎมีชื่อ และตัวแปรตรงกับเค้าร่างที่ได้แปลงมา
4) เมื่อตรวจสอบการรวมเค้าร่าง พบว่าในระบบ Birthday Book มีเค้าร่างที่ทำการรวมเค้าร่างอื่นทั้งหมด 11 เค้าร่าง เป็นการรวมแบบมีการเปลี่ยนแปลงสถานะของเค้าร่าง 2 เค้าร่าง และเป็นการรวมแบบไม่มีการเปลี่ยนแปลงสถานะของเค้าร่างทั้งหมด 9 เค้าร่าง พบว่ากฎที่แปลงจากเค้าร่างดังกล่าวมีตัวแปรของเค้าร่างที่รวมอยู่ด้วย และสำหรับการรวมเค้าร่างแบบมีการเปลี่ยนแปลงสถานะหลังก็มีการเพิ่มตัวแปรสถานะหลังด้วย
5) เมื่อตรวจสอบเงื่อนไขของกฎ พบว่าแต่ละกฎมีเงื่อนไขครบตามภาคแสดงในเค้าร่าง และมีความหมายตรงกับภาคแสดงนั้น ๆ เมื่อมีการดำเนินงานทีละเงื่อนไข
6) เมื่อตรวจสอบความหมายของกฎ พบว่าแต่ละกฎสามารถทำงานได้ และมีผลลัพธ์ตรงกับผลลัพธ์ของเค้าร่างที่แปลงมา และเมื่อตรวจสอบการทำงานของกฎหลายกฎต่อเนื่องกันก็สามารถให้ผลลัพธ์ที่ถูกต้อง

5.7.2. การตรวจสอบความหมายของผลลัพธ์โปรแกรมโพรล็อกของระบบ PDB

ระบบ PDB ประกอบด้วยเค้าร่างทั้งหมด 34 เค้าร่าง ได้ทำการตรวจสอบตามขั้นตอนในการตรวจสอบ 6 ขั้นตอน ซึ่งได้ผลดังนี้

1) เมื่อตรวจสอบความหมายของกฎในไลบราลี พบว่าความหมายของกฎแต่ละกฎตรงกับความหมายของตัวดำเนินการนั้น ๆ ซึ่งในไลบราลีประกอบด้วยกฎซึ่งสร้างจากตัวดำเนินการทั้งหมด 50 ตัว

2) เมื่อตรวจสอบการเรียงลำดับภาคแสดง พบว่าต้นฉบับเค้าร่าง AddEntry และต้นฉบับเค้าร่าง RemoveEntry นั้นมีการเรียงลำดับภาคแสดงไม่ถูกต้อง ในต้นฉบับเค้าร่าง AddEntry และต้นฉบับเค้าร่าง RemoveEntry มีภาคแสดงทั้งหมด 4 ภาคแสดง โดยภาคแสดงที่ 2 เป็นภาคแสดงที่อยู่ในลำดับที่ผิด โดยเครื่องมือซอฟต์แวร์สามารถเรียงลำดับภาคแสดงในเค้าร่าง AddEntry และเค้าร่าง RemoveEntry ใหม่ได้ถูกต้อง ซึ่งภาคแสดงที่ 2 ก็จะกลายเป็นภาคแสดงที่ 4 แทน
3) เมื่อตรวจสอบโครงสร้างของกฎพบว่า เครื่องมือซอฟต์แวร์สามารถสร้างกฎได้ทั้งหมด 34 กฎ แต่ละกฎมีชื่อ และตัวแปรตรงกับเค้าร่างที่ได้แปลงมา
4) เมื่อตรวจสอบการรวมเค้าร่าง พบว่าในระบบ PDB มีเค้าร่างที่ทำการรวมเค้าร่างอื่นทั้งหมด 24 เค้าร่าง เป็นการรวมแบบมีการเปลี่ยนแปลงสถานะของเค้าร่าง 5 เค้าร่าง และเป็นการรวมแบบไม่มีการเปลี่ยนแปลงสถานะของเค้าร่างทั้งหมด 19 เค้าร่าง พบว่ากฎที่แปลงจากเค้าร่างดังกล่าวมีตัวแปรของเค้าร่างที่รวมอยู่ด้วย และสำหรับการรวมเค้าร่างแบบมีการเปลี่ยนแปลงสถานะหลังก็มีการเพิ่มตัวแปรสถานะหลังด้วย
5) เมื่อตรวจสอบเงื่อนไขของกฎ พบว่าแต่ละกฎมีเงื่อนไขครบตามภาคแสดงในเค้าร่าง และมีความหมายตรงกับภาคแสดงนั้น ๆ เมื่อมีการดำเนินงานทีละเงื่อนไข
6) เมื่อตรวจสอบความหมายของกฎ พบว่าแต่ละกฎสามารถทำงานได้ และมีผลลัพธ์ตรงกับผลลัพธ์ของเค้าร่างที่แปลงมา และเมื่อตรวจสอบการทำงานของกฎหลายกฎต่อเนื่องกันก็สามารถให้ผลลัพธ์ที่ถูกต้อง

5.7.3. การตรวจสอบความหมายของผลลัพธ์โปรแกรมโพรล็อกของระบบห้องสมุด

ระบบห้องสมุดประกอบด้วยเค้าร่างทั้งหมด 38 เค้าร่าง ได้ทำการตรวจสอบตามขั้นตอนในการตรวจสอบ 6 ขั้นตอน ซึ่งได้ผลดังนี้

1) เมื่อตรวจสอบความหมายของกฎในไลบราลี พบว่าความหมายของกฎแต่ละกฎตรงกับความหมายของตัวดำเนินการนั้น ๆ ซึ่งในไลบราลีประกอบด้วยกฎซึ่งสร้างจากตัวดำเนินการทั้งหมด 50 ตัว
2) เมื่อตรวจสอบการเรียงลำดับภาคแสดงพบว่าไม่มีเค้าร่างใดที่มีลำดับภาคแสดงไม่ถูกต้อง
3) เมื่อตรวจสอบโครงสร้างของกฎพบว่า เครื่องมือซอฟต์แวร์สามารถสร้างกฎได้ทั้งหมด 38 กฎ แต่ละกฎมีชื่อ และตัวแปรตรงกับเค้าร่างที่ได้แปลงมา
4) เมื่อตรวจสอบการรวมเค้าร่าง พบว่าในระบบห้องสมุดมีเค้าร่างที่ทำการรวมเค้าร่างอื่นทั้งหมด 26 เค้าร่าง เป็นการรวมแบบมีการเปลี่ยนแปลงสถานะของเค้าร่าง 5 เค้าร่าง และเป็นการรวมแบบไม่มีการเปลี่ยนแปลงสถานะของเค้าร่างทั้งหมด 21 เค้าร่าง พบว่ากฎที่แปลงจากเค้าร่างดังกล่าวมีตัวแปรของเค้าร่างที่รวมอยู่ด้วย และสำหรับการรวมเค้าร่างแบบมีการเปลี่ยนแปลงสถานะหลังก็มีการเพิ่มตัวแปรสถานะหลังด้วย

5) เมื่อตรวจสอบเงื่อนไขของกฎ พบว่าแต่ละกฎมีเงื่อนไขครบตามภาคแสดงในเค้าร่าง และมีความหมายตรงกับภาคแสดงนั้น ๆ เมื่อมีการดำเนินงานทีละเงื่อนไข
6) เมื่อตรวจสอบความหมายของกฎ พบว่าแต่ละกฎสามารถทำงานได้ และมีผลลัพธ์ตรงกับผลลัพธ์ของเค้าร่างที่แปลงมา และเมื่อตรวจสอบการทำงานของกฎหลายกฎต่อเนื่องกันก็สามารถให้ผลลัพธ์ที่ถูกต้อง

บทที่ 6
สรุปผลการวิจัย

จากการศึกษา และ วิจัยสามารถสรุปผลที่ได้รับจากการวิจัย ปัญหา และข้อจำกัดที่พบ ได้ดังนี้

6.1. สรุปผลการวิจัย

จากการวิจัย ได้ออกแบบขั้นตอนวิธีที่ใช้ในการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก โดยข้อมูลจากข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจะอยู่ในรูปแทคของลาเทกซ์ ทั้งนี้สามารถออกแบบขั้นตอนวิธีหลักได้ 2 ขั้นตอนวิธี ได้แก่

1) ขั้นตอนวิธีเรียงลำดับภาคแสดง เป็นส่วนที่ทำการเรียงลำดับภาคแสดงของแต่ละเค้าร่างให้อยู่ในลำดับที่ถูกต้อง เพื่อลดการเกิดข้อผิดพลาดของโปรแกรมโพรล็อก
2) ขั้นตอนวิธีแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก เป็นส่วนที่ทำสร้างกฎของภาษาโพรล็อกที่สอดคล้องกับการทำงานของตัวดำเนินการของสัญกรณ์เซด เพื่อความสะดวกในการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซด และเป็นส่วนที่ทำการแปลงภาคแสดงแต่ละภาคแสดงของแต่ละเค้าร่างเป็นเงื่อนไขของกฎ

ในการทดสอบ ผู้วิจัยได้ใช้กรณีทดสอบ 3 กรณีคือระบบ Birthday Book ระบบ PhoneDatabase และระบบห้องสมุดโดยเครื่องมือนี้สามารถแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่อยู่ในรูปแทคของลาเทกซ์จากทั้ง 3 ระบบนี้ ให้เป็นโปรแกรมโพรล็อกได้ และผลลัพธ์โปรแกรมโพรล็อกที่ได้จากการแปลงเมื่อนำไปทดสอบด้วย SICStus™ ปรากฏว่าไม่มีข้อผิดพลาดใด ๆ และได้ทำการทดสอบโดยสร้างกรณีทดสอบทั้งหมด 3 กรณีด้วย

6.2. ประโยชน์ของเครื่องมือแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดเป็นโปรแกรมโพรล็อก

1) เป็นเครื่องมือที่สาธิตการทำงานของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดได้ ทำให้ทราบว่าข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้น ๆ มีความสามารถอธิบายระบบอย่างสอดคล้องได้
2) เป็นเครื่องมือที่ช่วยในสนับสนุนการเขียนข้อกำหนดโดยใช้สัญกรณ์เซด

6.3. ปัญหา และข้อจำกัดที่ได้พบจากการวิจัย

1) ข้อมูลนำเข้าจะต้องอยู่ในรูปแทคของลาเทกซ์ตามแบบที่กำหนด จึงจะสามารถสร้างโปรแกรมโพรล็อกได้โดยสมบูรณ์

2) ข้อมูลนำเข้าจะต้องถูกต้องตามวากยสัมพันธ์ของสัญกรณ์เซต
3) ข้อมูลนำเข้าจะต้องอยู่ในลำดับที่เหมาะสมกับการอ้างอิง คือ ถ้าหากเค้าร่างใดถูกอ้างอิงโดยเค้าร่างอื่น เค้าร่างนั้น ก็จะต้องถูกสร้างก่อนที่จะถูกอ้างอิง
4) ตัวแปรแต่ละตัวในส่วนประกาศตัวแปรจะต้องมีสัญลักษณ์บอกว่า ตัวแปรใดเป็นข้อมูลเข้า ข้อมูลออก หรือตัวแปรสถานะหลัง เพื่อใช้ในการเรียงลำดับภาคแสดง
5) ภาคแสดงในแต่ละเค้าร่างถ้ามีตัวดำเนินการมากกว่า 1 ตัวจะต้องใส่วงเล็บเพื่อบอกว่าตัวดำเนินการใดจะถูกกระทำก่อน
6) ถ้าข้อกำหนดรูปนัยในรูปสัญกรณ์เซตใดมีเค้าร่างเชิงสัจพจน์ จะต้องทำการแปลงเค้าร่างเชิงสัจพจน์ให้เป็นเค้าร่างเชิงสคีมาก่อน

รายการอ้างอิง

[1] Spivey, J.M. “An introduction to Z and formal specification” Software Engineering Journal:4 (1 January 1989):40-50.

[2] Fuchs, N.E. “Specifications are (preferably) executable” Software Engineering Journal 7 (5 September 1992):323-333.

[3] Bowen, J.P. “Formal specification in Z as a design and document tool” Software Engineering, 1988 Software Engineering 88, Second IEE/BCS Conference (1988):164-168.

[4] Jia, X. “An approach to animating Z specification” Computer Software and applications Conference,COMPSAC’95. Proceedings, Nineteenth Annual International (1995):108-113.

[5] Utting, M. “Animating z: interactivity, transparency and equivalence” Software Engineering Conference, Proceedings, Asia Pacific (1995):297-303.

[6] Ishaq, K.P., Masterson, JJ., Rich, C.B. “The Z environment” Application of Computer Aided Software Engineering Tools, Iee Colloquium on (1989):4/1-4/5 .

[7] Ekambareshwar, S. “A suite of software tools for executing formal specification” TENCON’91, IEEE Region 10 International Conference onEC3-Energy, Computer, Communication and Control systems:2 (1991):322-329.

[8] Spivey, J.M. “The Z notation: a reference manual” Practice Hall (1989).

[9] West, M.M., Eaglestone, B.M. “Software development: two approaches to animation of Z specification using Prolog” Software Engineering Journal:7 (4 July 1992):264-267.

[10] Hewitt, M.A. “Automated animation of Z using Prolog” Department of computing, Lancaster university (1991):1-52.

[11] Bryan Ratcliff “Introducing Specification Using Z: A Pracical Case Study Approach” The McGraw-Hill International (UK) Limited (1994).

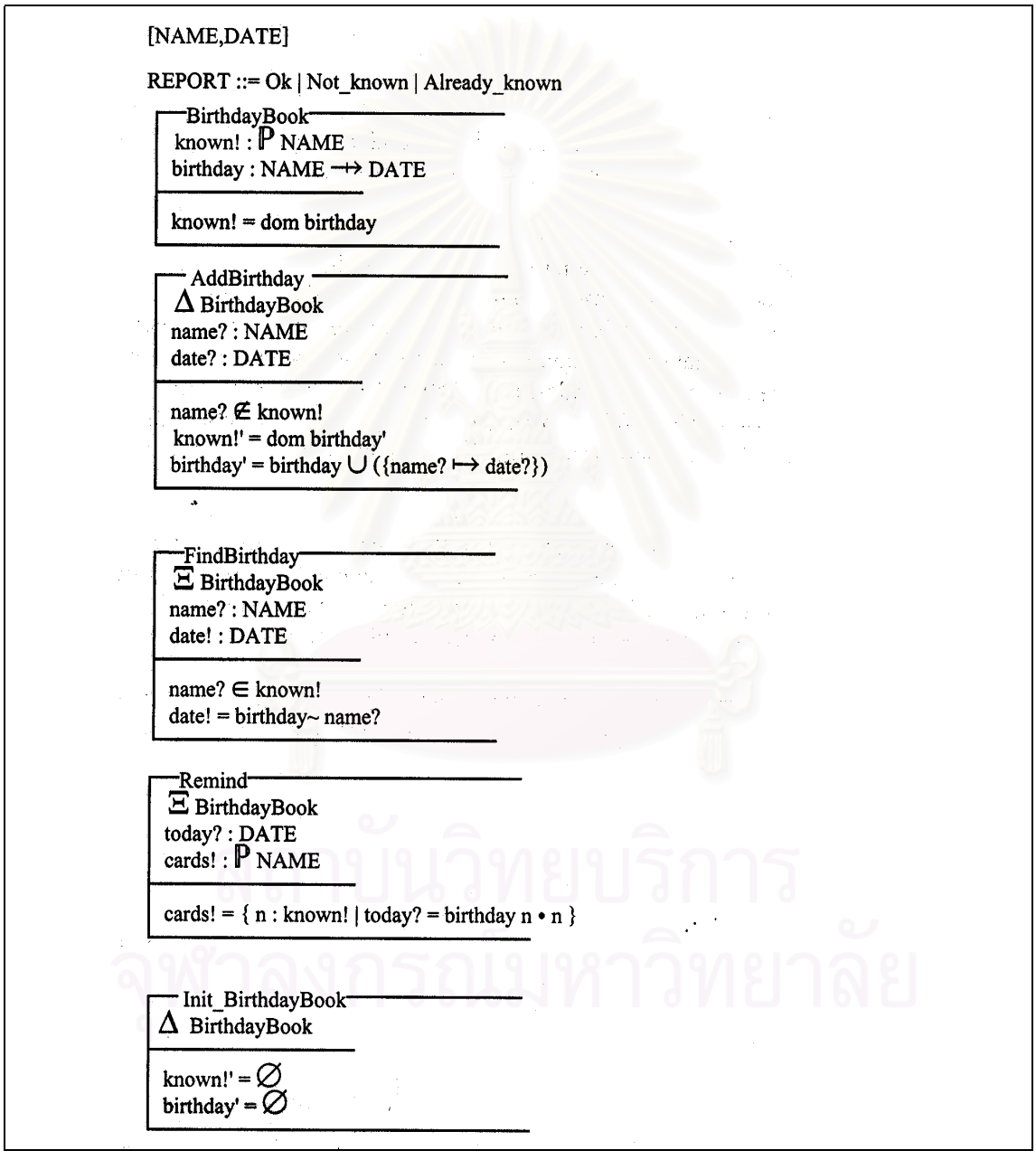
[12] Le, Tu Van “Techniques of Prolog programming” John Wiley&sons (1993).

[13] Meisels, Saaltink “Z/EVES reference manual” Developing Secure and Dependable Systems Using Advanced Software Technology (1999).

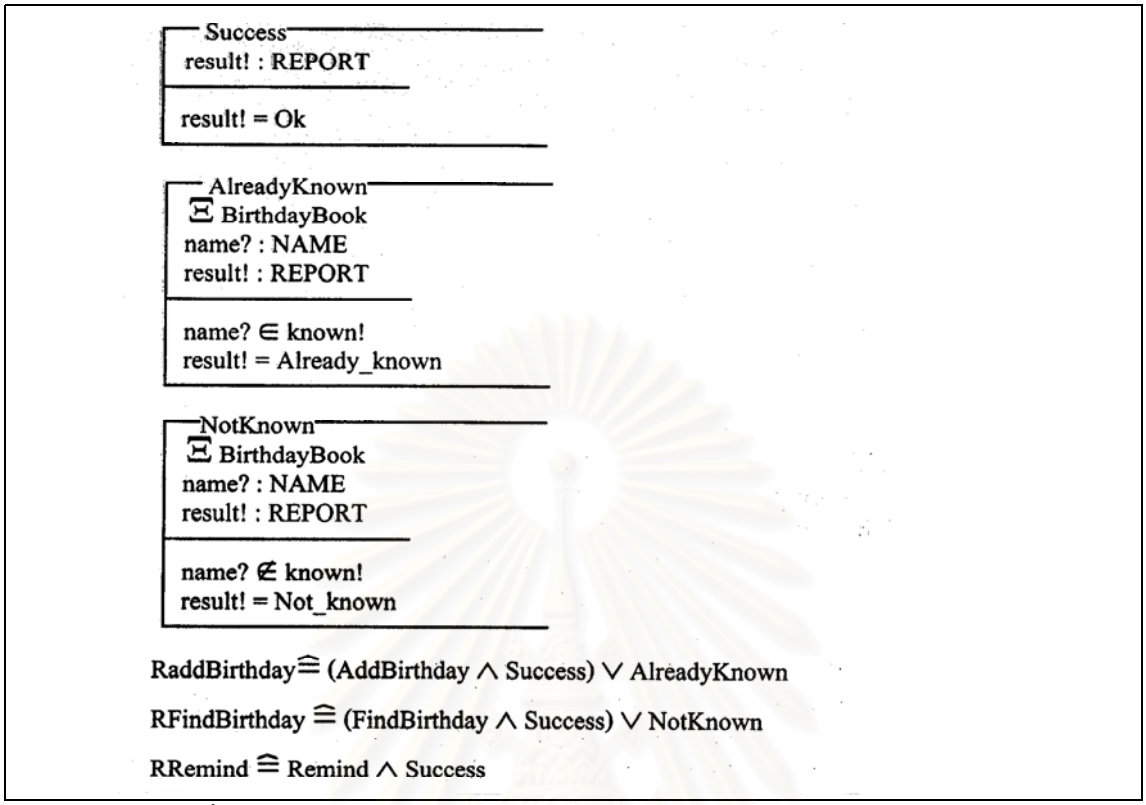
ภาคผนวก

ภาคผนวก ก. ระบบ Birthday Book

ระบบ Birthday Book เป็นระบบที่ใช้ในการบันทึกวันเกิดของบุคคล ซึ่งมีลักษณะคล้ายกับสมุดที่ใช้ในการบันทึกวันเกิด ในระบบจะมี Birthday ที่ใช้ในการเก็บชื่อและวันเกิดของบุคคลซึ่งชื่อคนที่เก็บได้จะต้องไม่ซ้ำกัน ผู้ใช้สามารถเพิ่มชื่อ ค้นหาชื่อจากระบบได้ และระบบสามารถบอกได้ว่าวันใดเป็นวันเกิดของคนไหน จากระบบ BirthdayBook นี้ สามารถเขียนเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ได้ดังรูปที่ ก-1



รูปที่ ก-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดระบบ Birthday Book



รูปที่ ก-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดระบบ Birthday Book (ต่อ)

จากระบบ Birthday Book ซึ่งอยู่ในรูปสัญกรณ์เซดสามารถแปลงให้อยู่ในรูปแทคของลาเทกซ์ได้ดังรูป ก-2

```
\begin{zed}
[NAME,DATE]
\end{zed}
\begin{zed}
REPORT ::= Ok | Not\_known | Already\_known
\end{zed}
\begin{schema}{BirthdayBook}
known! : \power NAME  \\
birthday : NAME \pfun DATE
\where
  known! = \dom birthday
\end{schema}
```

รูปที่ ก-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดระบบ Birthday Book ในรูปแทคของลาเทกซ์

```
\begin{schema}{AddBirthday}
\Delta BirthdayBook  \\
name? : NAME  \\
date? : DATE
\where
name? \notin known!  \\
known!' = \dom birthday'  \\
birthday' = birthday \cup (\{name? \mapsto date?\})
\end{schema}
\begin{schema}{FindBirthday}
\Xi BirthdayBook  \\
name? : NAME  \\
date! : DATE
\where
name? \in  known!  \\
date! = birthday~ name?
\end{schema}
\begin{schema}{Remind}
\Xi BirthdayBook  \\
today? : DATE  \\
cards! : \power NAME
\where
cards! = \{ n : known! | today? = birthday~ n @ n \}
\end{schema}
\begin{schema}{init\_BirthdayBook}
\Delta  BirthdayBook
\where
known!' = \emptyset  \\
birthday' = \emptyset
\end{schema}
\begin{schema}{Success}
  result! : REPORT
\where
```

รูปที่ ก-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดระบบ Birthday Book ในรูปแทคของลาเทกซ์ (ต่อ)

```
  result! = Ok
\end{schema}
\begin{schema}{AlreadyKnown}
\Xi BirthdayBook  \\
name? : NAME \\
result! : REPORT
\where
name? \in known! \\
result! = Already\_known
\end{schema}
\begin{schema}{NotKnown}
\Xi BirthdayBook  \\
name? : NAME \\
result! : REPORT
\where
name? \notin known! \\
result! = Not\_known  \\
\end{schema}
\begin{zed}
RAddBirthday \defs (AddBirthday \land Success) \lor AlreadyKnown
\end{zed}
\begin{zed}
RFindBirthday \defs (FindBirthday \land Success) \lor NotKnown
\end{zed}
\begin{zed}
RRemind \defs Remind \land Success
\end{zed}
```

รูปที่ ก-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตระบบ Birthday Book ในรูปแทคของลาเทกซ์ (ต่อ)

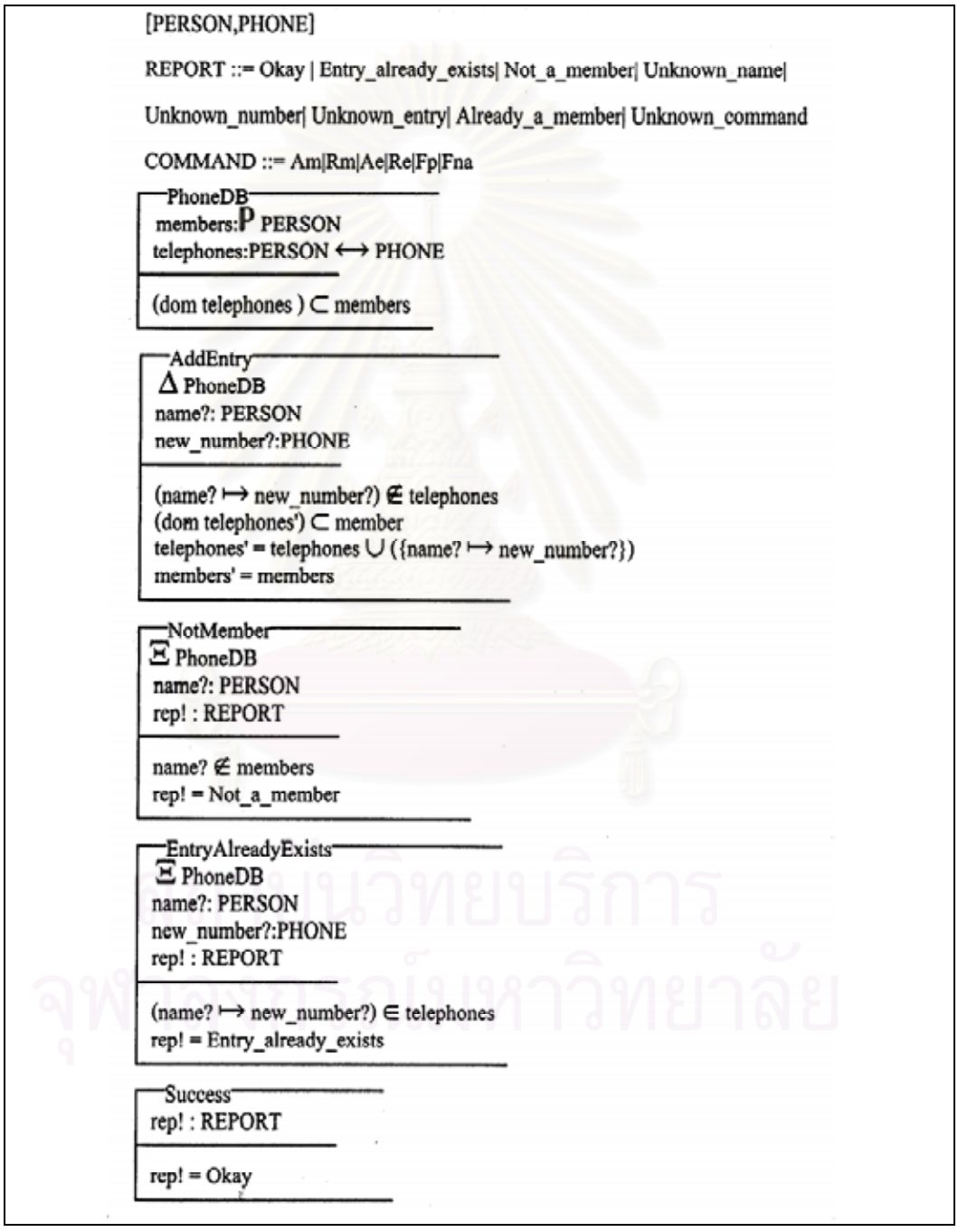
จากระบบ Birthday Book ซึ่งอยู่ในรูปแทคของลาเทกซ์ดังรูปที่ ก-2 สามารถใช้เครื่องมือซอฟต์แวร์แปลงให้อยู่ในรูปโปรแกรมโพรล็อกได้ดังรูป ก-3

```
:-use_module(library(z)).
birthdayBook(Birthday,Known):- dom(Birthday,Known).
addBirthday(Name,Date,Birthday,Known,Birthdayp,Knownp):- birthdayBook
    (Birthday,Known),notin(Name,Known),dom(Birthdayp,Knownp),Temp1 = [(Name ,
    Date)],cup(Birthday,Temp1,Birthdayp).
findBirthday(Name,Birthday,Known,Date):- birthdayBook(Birthday,Known),in
    (Name,Known),function(Birthday,Name,Date).
remind(Today,Birthday,Known,Cards):- birthdayBook(Birthday,Known),Temp2 = Known,
    is2((function(Birthday,N,Today)),Temp3),is2((Temp5 = N,Cards =
    Temp5),Temp4),land(Temp3,Temp4).
init_BirthdayBook(Birthday,Known,Birthdayp,Knownp):- birthdayBook(Birthday,Known),
    Knownp=[],Birthdayp=[].
success(Result):- Result = ok.
alreadyKnown(Name,Birthday,Known,Result):- birthdayBook(Birthday,Known),in
    (Name,Known),Result = already_known.
notKnown(Name,Birthday,Known,Result):- birthdayBook(Birthday,Known),notin
    (Name,Known),Result = not_known.
rAddBirthday(Name,Date,Birthday,Known,Result,Birthdayp,Knownp):-
    Schema1=addBirthday(Name,Date,Birthday,Known,Birthdayp,Knownp),Schema2=succ
    ess(Result),Schema3=alreadyKnown(Name,Birthday,Known,Result),land
    (Schema1,Schema2,Temp6),lor(Temp6,Schema3).
rFindBirthday(Name,Birthday,Known,Date,Result):- Schema1=findBirthday
    (Name,Birthday,Known,Date),Schema2=success(Result),Schema3=notKnown
    (Name,Birthday,Known,Result),land(Schema1,Schema2,Temp7),lor(Temp7,Schema3).
rRemind(Today,Birthday,Known,Result,Cards):-
    Schema1=remind(Today,Birthday,Known,Cards),Schema2=success(Result),land
    (Schema1,Schema2).
```

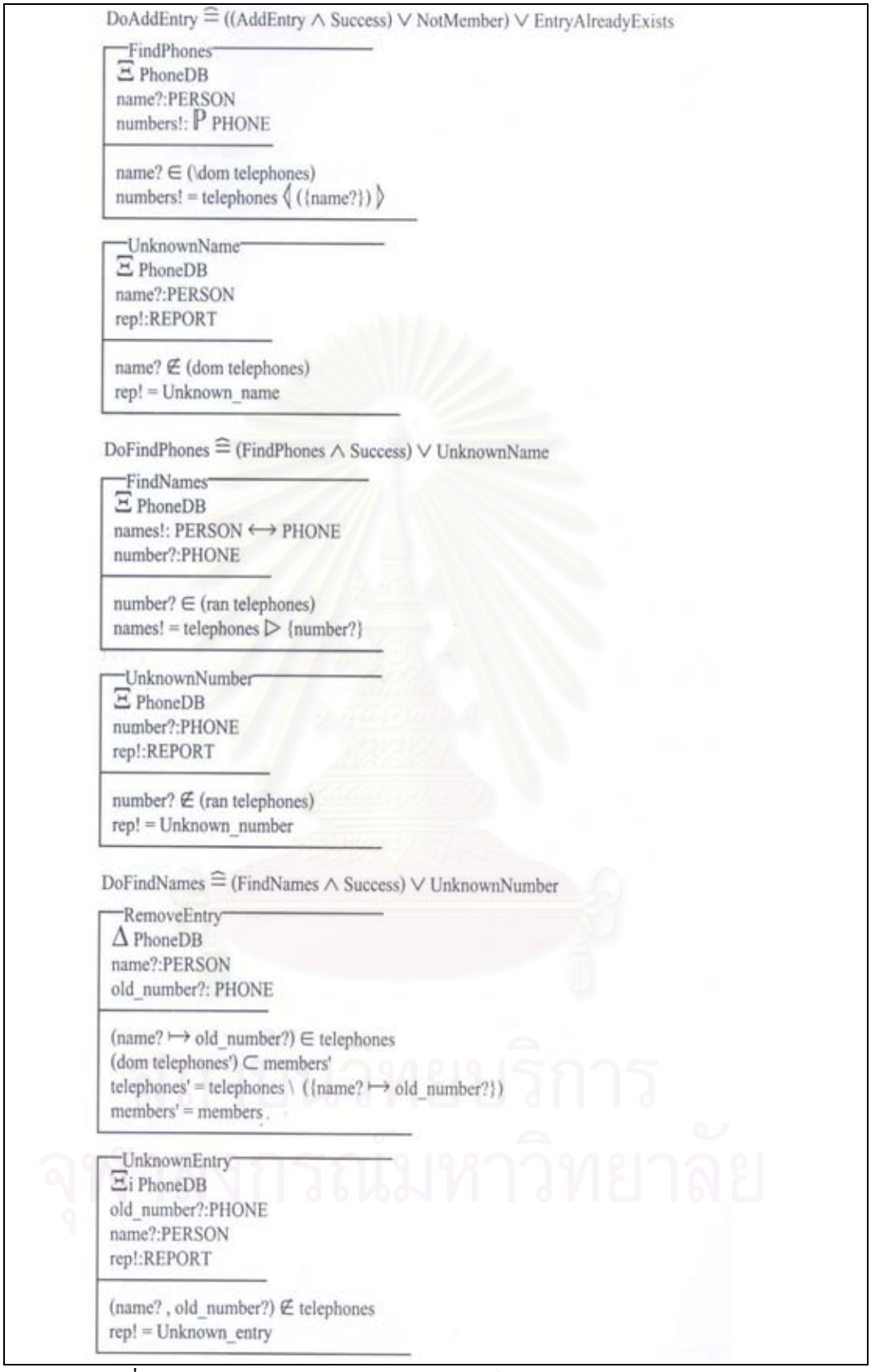
รูปที่ ก-3 แสดงโปรแกรมโพรล็อกของระบบ Birthday Book

ภาคผนวก ข. ระบบ PDB (Phone Database)

เป็นระบบที่ใช้ในการเก็บชื่อและเบอร์โทรศัพท์ของบุคคล มีลักษณะคล้ายกับสมุดโทรศัพท์ แต่ระบบนี้ ผู้ที่จะบันทึกเบอร์โทรศัพท์ได้จะต้องเป็นสมาชิกก่อน สมาชิก 1 คนอาจมีเบอร์โทรศัพท์ได้หลายเบอร์ และโทรศัพท์ 1 เบอร์อาจเป็นของสมาชิกหลายคน ระบบนี้สามารถเพิ่ม, ลบ, ค้นหาเบอร์โทรศัพท์ เพิ่ม, ลบ, ค้นหาสมาชิกได้ ระบบ PDB สามารถแสดงในรูปสัญกรณ์เซตได้ดังรูปที่ ข-1



รูปที่ ข–1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ PDB



รูปที่ ข–1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB (ต่อ)

$DoRemoveEntry \widehat{=} (RemoveEntry \land Success) \lor UnknownEntry$

AddMember

$\Delta PhoneDB$

$name?:PERSON$

$name? \notin members$

$members' = member \cup (\{name?\})$

$telephones' = telephones$

AlreadyMember

$\Xi PhoneDB$

$name?:PERSON$

$rep!:REPORT$

$name? \in members$

$rep! = Already\_a\_member$

$DoAddMember \widehat{=} (AddMember \land Success) \lor AlreadyMember$

RemoveMember

$\Delta PhoneDB$

$name?:PERSON$

$name? \in members$

$members' = members \setminus (\{name?\})$

$telephones' = (\{name?\}) \lhd telephones$

$DoRemoveMember \widehat{=} (RemoveMember \land Success) \lor NotMember$

Init_PhoneDB

$\Delta PhoneDB$

$members' = \varnothing$

$telephones' = \varnothing$

AddMemberCommand

$cmd?:COMMAND$

$cmd? = Am$

$ComAddMember \widehat{=} AddMemberCommand \land DoAddMember$

RemoveMemberCommand

$amd?: COMMAND$

$amd? = Rm$

$ComRemoveMember \widehat{=} RemoveMemberCommand \land DoRemoveMember$

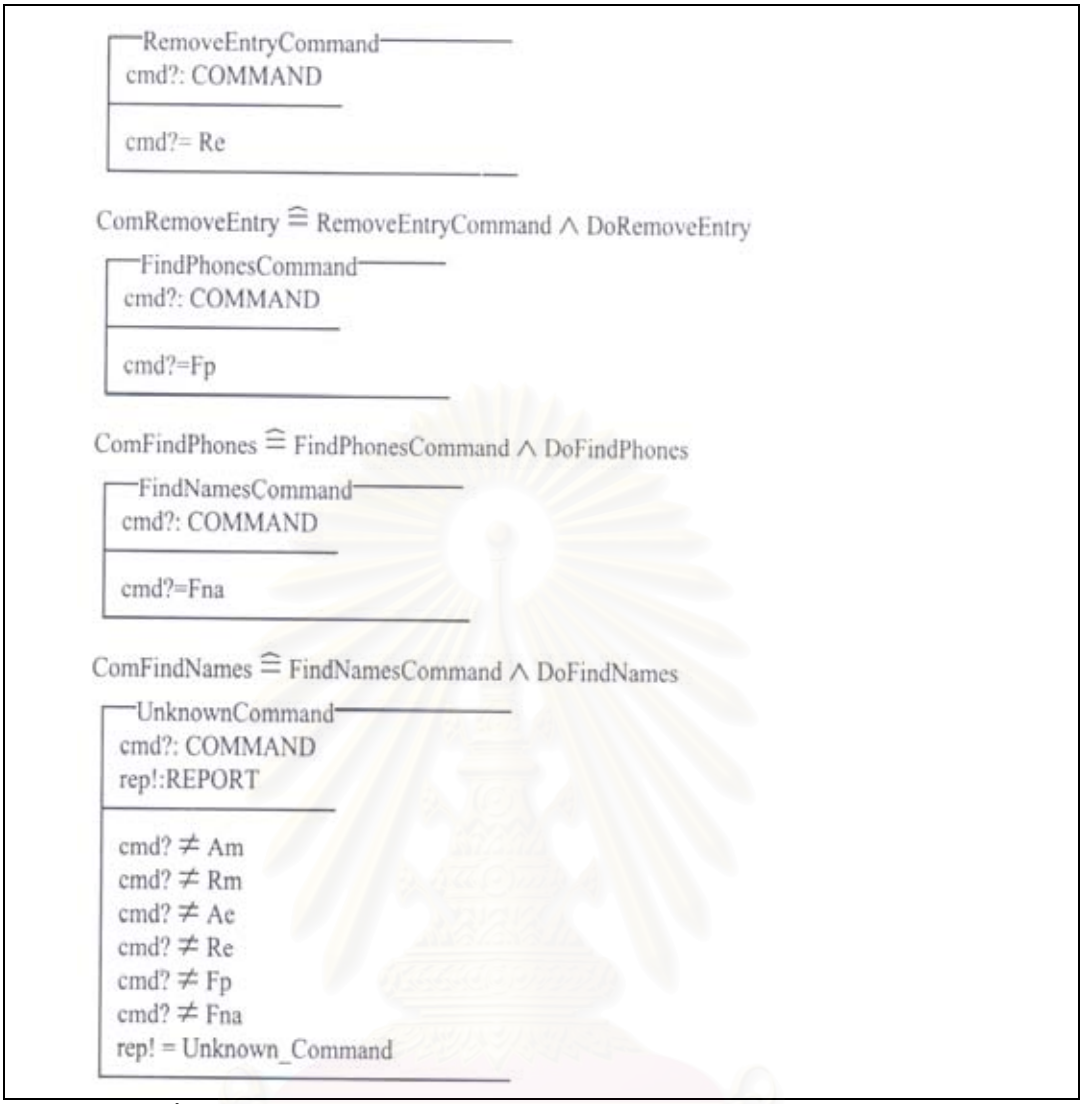
AddEntryCommand

$cmd?: COMMAND$

$cmd? = Ae$

$ComAddEntry \widehat{=} AddEntryCommand \land DoAddEntry$

รูปที่ ข–1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ PDB (ต่อ)



รูปที่ ข–1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB (ต่อ)

จากระบบ PDB ในรูปสัญกรณ์เซด สามารถแปลงให้อยู่ในรูปแทคของลาเทกซ์ได้ ดังรูปที่ ข-2

```
\begin{zed}
[PERSON,PHONE]
\end{zed}
\begin{zed}
REPORT ::= Okay | Entry\_already\_exists| Not\_a\_member| Unknown\_name|
    Unknown\_number| Unknown\_entry| Already\_a\_member| Unknown\_command
\end{zed}
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์

```
\begin{zed}
COMMAND ::= Am|Rm|Ae|Re|Fp|Fna
\end{zed}
\begin{schema}{PhoneDB}
members:\power PERSON \\
telephones:PERSON \rel PHONE
\where
(\dom telephones ) \subset members
\end{schema}
\begin{schema}{AddEntry}
\Delta PhoneDB \\
name?:PERSON \\
new\_number?:PHONE
\where
name? \in members     \\
(name? \mapsto new\_number?) \notin telephones \\
(\dom telephones' ) \subset members'  \\
telephones' = telephones \cup (\{name? \mapsto new\_number?\}) \\
members' = members
\end{schema}
\begin{schema}{NotMember}
\Xi PhoneDB \\
name?:PERSON \\
rep! : REPORT
\where
name? \notin members \\
rep! = Not\_a\_member
\end{schema}
\begin{schema}{EntryAlreadyExists}
\Xi PhoneDB \\
name?:PERSON \\
new\_number?:PHONE \\
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
rep!:REPORT
\where
(name? \mapsto new\_number?) \in telephones \\
rep! = Entry\_already\_exists
\end{schema}
\begin{schema}{Success}
rep!:REPORT
\where
rep! = Okay
\end{schema}
\begin{zed}
DoAddEntry \defs ((AddEntry \land Success) \lor NotMember)\lor EntryAlreadyExists
\end{zed}
\begin{schema}{FindPhones}
\Xi PhoneDB \\
name?:PERSON \\
numbers!: \power PHONE
\where
name? \in (\dom telephones) \\
numbers! = telephones \limg (\{name?\}) \rimg
\end{schema}
\begin{schema}{UnknownName}
\Xi PhoneDB \\
name?:PERSON \\
rep!:REPORT
\where
name? \notin (\dom telephones) \\
rep! = Unknown\_name
\end{schema}
\begin{zed}
DoFindPhones \defs (FindPhones \land Success) \lor UnknownName
\end{zed}
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
\begin{schema}{FindNames}
\Xi PhoneDB \\
names!: PERSON \rel PHONE \\
number?:PHONE
\where
number? \in (\ran telephones) \\
names! = telephones \rres (\{number?\})
\end{schema}
\begin{schema}{UnknownNumber}
\Xi PhoneDB \\
number?:PHONE \\
rep!:REPORT
\where
number? \notin (\ran telephones) \\
rep! = Unknown\_number
\end{schema}
\begin{zed}
DoFindNames \defs (FindNames \land Success) \lor UnknownNumber
\end{zed}
\begin{schema}{RemoveEntry}
\Delta PhoneDB \\
name?:PERSON \\
old\_number?: PHONE
\where
(name? \mapsto old\_number?) \in telephones \\
(\dom telephones' ) \subset members'  \\
telephones' = telephones \setminus  (\{name? \mapsto old\_number? \}) \\
members' = members
\end{schema}
\begin{schema}{UnknownEntry}
\Xi PhoneDB \\
old\_number?:PHONE \\
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
name?:PERSON \\
rep!:REPORT
\where
(name? ,old\_number?) \notin telephones \\
rep! = Unknown\_entry
\end{schema}
\begin{zed}
DoRemoveEntry \defs (RemoveEntry \land Success) \lor UnknownEntry
\end{zed}
\begin{schema}{AddMember}
\Delta PhoneDB \\
name?:PERSON
\where
name? \notin members? \\
members' = members \cup (\{name?\}) \\
telephones' = telephones
\end{schema}
\begin{schema}{AlreadyMember}
\Xi PhoneDB \\
name?:PERSON \\
rep!:REPORT
\where
name? \in members?  \\
rep! = Already\_a\_member
\end{schema}
\begin{zed}
DoAddMember \defs (AddMember \land Success) \lor AlreadyMember
\end{zed}
\begin{schema}{RemoveMember}
\Delta PhoneDB \\
name?:PERSON
\where
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
name? \in members? \\
members'= members \setminus (\{name?\}) \\
telephones' = (\{name?\}) \ndres telephones
\end{schema}
\begin{zed}
DoRemoveMember \defs (RemoveMember \land Success) \lor NotMember
\end{zed}
\begin{schema}{init\_PhoneDB}
\Delta PhoneDB
\where
members' = \emptyset \\
telephones' = \emptyset
\end{schema}
\begin{schema}{AddMemberCommand}
cmd?:COMMAND
\where
cmd? = Am
\end{schema}
\begin{zed}
ComAddMember \defs AddMember \land DoAddMember
\end{zed}
\begin{schema}{RemoveMemberCommand}
amd?:COMMAND
\where
amd?= Rm
\end{schema}
\begin{zed}
ComRemoveMember \defs RemoveMemberCommand \land DoRemoveMember
\end{zed}
\begin{schema}{AddEntryCommand}
cmd?:COMMAND
\where
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
cmd? = Ae
\end{schema}
\begin{zed}
ComAddEntry \defs AddEntryCommand \land DoAddEntry
\end{zed}
\begin{schema}{RemoveEntryCommand}
cmd?:COMMAND
\where
cmd?= Re
\end{schema}
\begin{zed}
ComRemoveEntry \defs RemoveEntryCommand \land DoRemoveEntry
\end{zed}
\begin{schema}{FindPhonesCommand}
cmd?:COMMAND
\where
cmd?=Fp
\end{schema}
\begin{zed}
ComFindPhones \defs FindPhonesCommand \land DoFindPhones
\end{zed}
\begin{schema}{FindNamesCommand}
cmd!:COMMAND
\where
cmd!=Fna
\end{schema}
\begin{zed}
ComFindNames \defs FindNamesCommand \land DoFindNames
\end{zed}
\begin{schema}{UnknownCommand}
cmd?:COMMAND \\
rep!:REPORT
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

```
\where
cmd? \neq Am \\
cmd? \neq Rm \\
cmd? \neq Ae \\
cmd? \neq Re \\
cmd? \neq Fp \\
cmd? \neq Fna \\
rep! = Unknown\_command
\end{schema}
```

รูปที่ ข-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบ PDB ในรูปแทคของลาเทกซ์ (ต่อ)

จากระบบ PDB ซึ่งอยู่ในรูปแทคของลาเทกซ์ดังรูปที่ ข-2 สามารถใช้เครื่องมือซอฟต์แวร์แปลงให้อยู่ในรูปโปรแกรมโพรล็อกได้ดังรูป ข-3

```
:-use_module(library(z)).
phoneDB(Members,Telephones):- dom(Telephones,Temp1),subset(Temp1,Members).
addEntry(Name,New_number,Members,Telephones,Membersp,Telephonesp):- phoneDB
    (Members,Telephones),in(Name,Members),mapsto(Name,New_number,Temp2),notin
    (Temp2,Telephones),dom(Telephonesp,Temp3),subset(Temp3,Membersp),Temp4 =
    [(Name , New_number)],cup(Telephones,Temp4,Telephonesp),Membersp = Members.
notMember(Name,Members,Telephones,Rep):- phoneDB(Members,Telephones),notin
    (Name,Members),Rep = not_a_member.
entryAlreadyExists(Name,New_number,Members,Telephones,Rep):- phoneDB
    (Members,Telephones),mapsto(Name,New_number,Temp5),in(Temp5,Telephones),Rep
    = entry_already_exists.
success(Rep):- Rep = okay.
doAddEntry(Name,New_number,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=addEntry(Name,New_number,Members,Telephones,Membersp,Telephonesp),S
    chema2=success(Rep),Schema3=notMember(Name,Members,Telephones,Rep),Schema4
    =entryAlreadyExists(Name,New_number,Members,Telephones,Rep),land
    (Schema1,Schema2,Temp6),lor(Temp6,Schema3,Temp5),lor(Temp5,Schema4).
findPhones(Name,Members,Telephones,Numbers):- phoneDB(Members,Telephones),dom
```

รูปที่ ข-3 แสดงโปรแกรมโพรล็อกของระบบ PDB

```
    (Telephones,Temp7),in(Name,Temp7),Temp8 = [Name],limg(Telephones,Temp8,
    Numbers).
unknownName(Name,Members,Telephones,Rep):- phoneDB(Members,Telephones),dom
    (Telephones,Temp9),notin(Name,Temp9),Rep = unknown_name.
doFindPhones(Name,Members,Telephones,Numbers,Rep):- Schema1=findPhones
    (Name,Members,Telephones,Numbers),Schema2=success(Rep),Schema3=unknownName
    (Name,Members,Telephones,Rep),land(Schema1,Schema2,Temp10),lor
    (Temp10,Schema3).
findNames(Number,Members,Telephones,Names):- phoneDB(Members,Telephones),ran
    (Telephones,Temp11),in(Number,Temp11),Temp12 = [Number],
    rres(Telephones,Temp12,Names).
unknownNumber(Number,Members,Telephones,Rep):- phoneDB(Members,Telephones),ran
    (Telephones,Temp13),notin(Number,Temp13),Rep = unknown_number.
doFindNames(Number,Members,Telephones,Names,Rep):- Schema1=findNames
    (Number,Members,Telephones,Names),Schema2=success(Rep),Schema3=unknownNum
    ber(Number,Members,Telephones,Rep),land(Schema1,Schema2,Temp14),lor
    (Temp14,Schema3).
removeEntry(Name,Old_number,Members,Telephones,Membersp,Telephonesp):- phoneDB
    (Members,Telephones),mapsto(Name,Old_number,Temp16),in(Temp16,Telephones),do
    m(Telephonesp,Temp17),subset(Temp17,Membersp),Temp18 = [(Name , Old_number
    )],setminus(Telephones,Temp18,Telephonesp),Membersp = Members.
unknownEntry(Old_number,Name,Members,Telephones,Rep):-
    phoneDB(Members,Telephones),mapsto(Name,Old_number,Temp19),notin
    (Temp19,Telephones),Rep = unknown_entry.
doRemoveEntry(Name,Old_number,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=removeEntry(Name,Old_number,Members,Telephones,Membersp,Telephonesp
    ),Schema2=success(Rep),Schema3=unknownEntry(Old_number,Name,Members,Telepho
    nes,Rep),land(Schema1,Schema2,Temp19),lor(Temp19,Schema3).
addMember(Name,Members,Telephones,Membersp,Telephonesp):- phoneDB
    (Members,Telephones),notin(Name,Members),Temp20 =
    [Name],cup(Members,Temp20,Membersp),Telephonesp = Telephones.
alreadyMember(Name,Members,Telephones,Rep):- phoneDB(Members,Telephones),in
    (Name,Members),Rep = already_a_member.
```

รูปที่ ข-3 แสดงโปรแกรมโพรล็อกของระบบ PDB(ต่อ)

```
doAddMember(Name,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=addMember(Name,Members,Telephones,Membersp,Telephonesp),Schema2=s
    uccess(Rep),Schema3=alreadyMember(Name,Members,Telephones,Rep),land
    (Schema1,Schema2,Temp21),lor(Temp21,Schema3).
removeMember(Name,Members,Telephones,Membersp,Telephonesp):- phoneDB
    (Members,Telephones),in(Name,Members),Temp22 = [Name],setminus
    (Members,Temp22,Membersp),Temp23 = [Name],
    ndres(Temp23,Telephones,Telephonesp).
doRemoveMember(Name,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=removeMember(Name,Members,Telephones,Membersp,Telephonesp),Schema
    2=success(Rep),Schema3=notMember(Name,Members,Telephones,Rep),land
    (Schema1,Schema2,Temp24),lor(Temp24,Schema3).
init_PhoneDB(Members,Telephones,Membersp,Telephonesp):-
    phoneDB(Members,Telephones), Membersp=[],Telephonesp=[].
addMemberCommand(Cmd):- Cmd = am.
comAddMember(Name,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=addMember(Name,Members,Telephones,Membersp,Telephonesp),Schema2=d
    oAddMember(Name,Members,Telephones,Rep,Membersp,Telephonesp),land
    (Schema1,Schema2).
removeMemberCommand(Amd):- Amd = rm.
comRemoveMember(Amd,Name,Members,Telephones,Rep,Membersp,Telephonesp):-
    Schema1=removeMemberCommand(Amd),Schema2=doRemoveMember
    (Name,Members,Telephones,Rep,Membersp,Telephonesp),land(Schema1,Schema2).
addEntryCommand(Cmd):- Cmd = ae.
comAddEntry(Cmd,Name,New_number,Members,Telephones,Rep,Membersp,Telephonesp):
    - Schema1=addEntryCommand(Cmd),Schema2=doAddEntry(Name,New_number,
    Members, Telephones,Rep,Membersp,Telephonesp),land(Schema1,Schema2).
removeEntryCommand(Cmd):- Cmd = re.
comRemoveEntry(Cmd,Name,Old_number,Members,Telephones,Rep,Membersp,Telephones
    p):- Schema1=removeEntryCommand(Cmd),Schema2=doRemoveEntry(Name, Old_
    number,Members,Telephones,Rep,Membersp,Telephonesp),land(Schema1,Schema2).
findPhonesCommand(Cmd):- Cmd = fp.
comFindPhones(Cmd,Name,Members,Telephones,Numbers,Rep):-
```

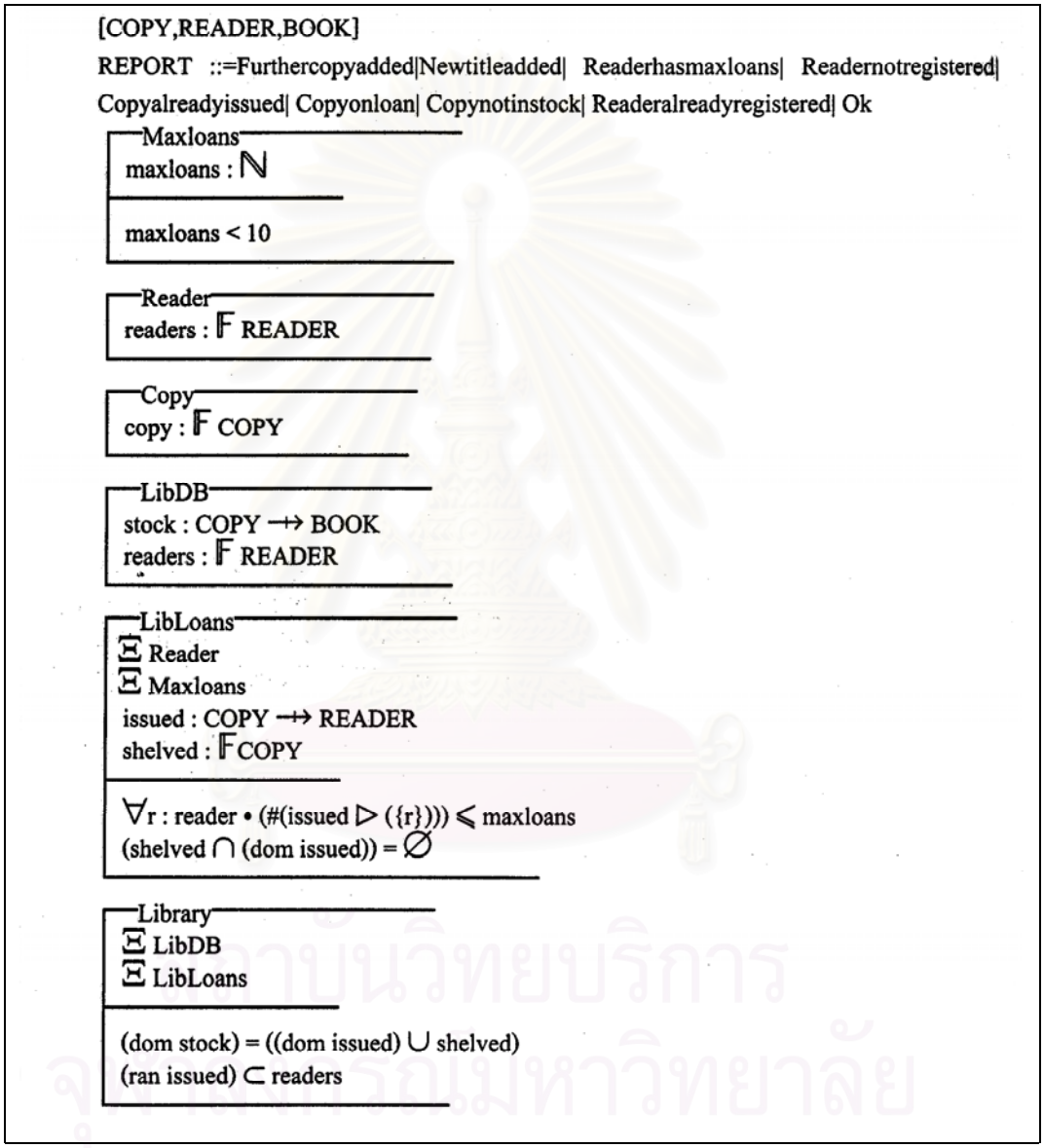
รูปที่ ข-3 แสดงโปรแกรมโพรล็อกของระบบ PDB(ต่อ)

```
    Schema1=findPhonesCommand(Cmd),Schema2=doFindPhones(Name,Members,Telepho
    nes,Numbers,Rep),land(Schema1,Schema2).
findNamesCommand(Cmd):- Cmd = fna.
comFindNames(Cmd,Number,Members,Telephones,Names,Rep):-
    Schema1=findNamesCommand(Cmd),Schema2=doFindNames(Number,Members,Teleph
    ones,Names,Rep),land(Schema1,Schema2).
unknownCommand(Cmd,Rep):- neq(Cmd,am),neq(Cmd,rm),neq(Cmd,ae),neq(Cmd,re),neq
    (Cmd,fp),neq(Cmd,fna),Rep = unknown_command.
```

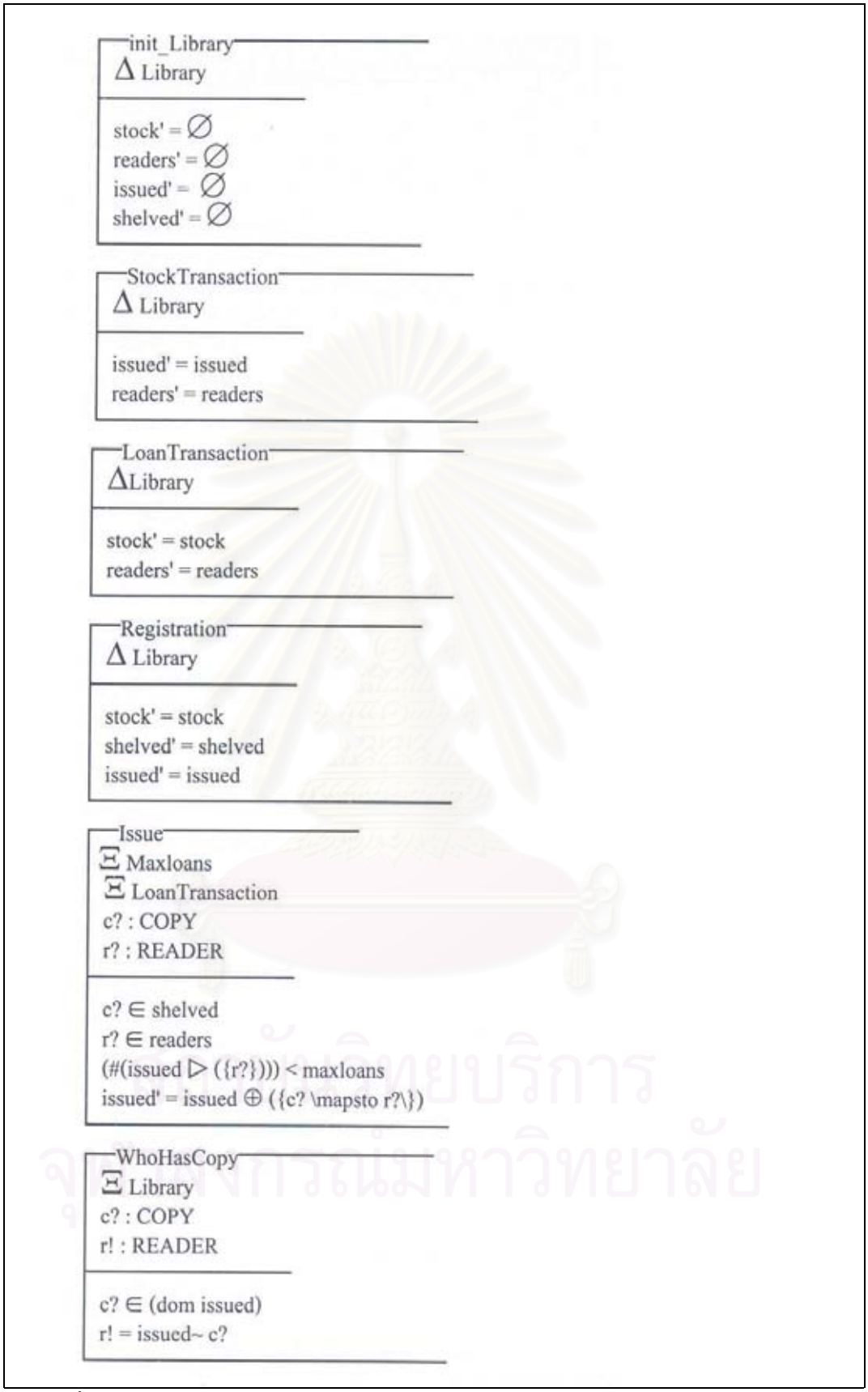
รูปที่ ข-3 แสดงโปรแกรมโพรล็อกของระบบ PDB (ต่อ)

ภาคผนวก ค. ระบบห้องสมุดขนาดเล็ก (Library)

เป็นระบบห้องสมุดขนาดเล็กที่ให้บริการยืมคืนหนังสือ ผู้ที่จะยืมหนังสือได้จะต้องสมัครเป็นสมาชิก มีการกำหนดจำนวนหนังสือสูงสุดที่สามารถยืมได้ ระบบนี้ สามารถทำการเพิ่ม ลบหนังสือ ตรวจสอบว่าใครยืมหนังสืออะไรไปบ้าง และหนังสือเล่มนี้ถูกใครยืมไป รวมทั้งสามารถเพิ่ม ลบสมาชิกได้ ระบบห้องสมุดขนาดเล็กสามารถแสดงอยู่ในรูปสัญกรณ์เซตได้ดังรูป ค-1

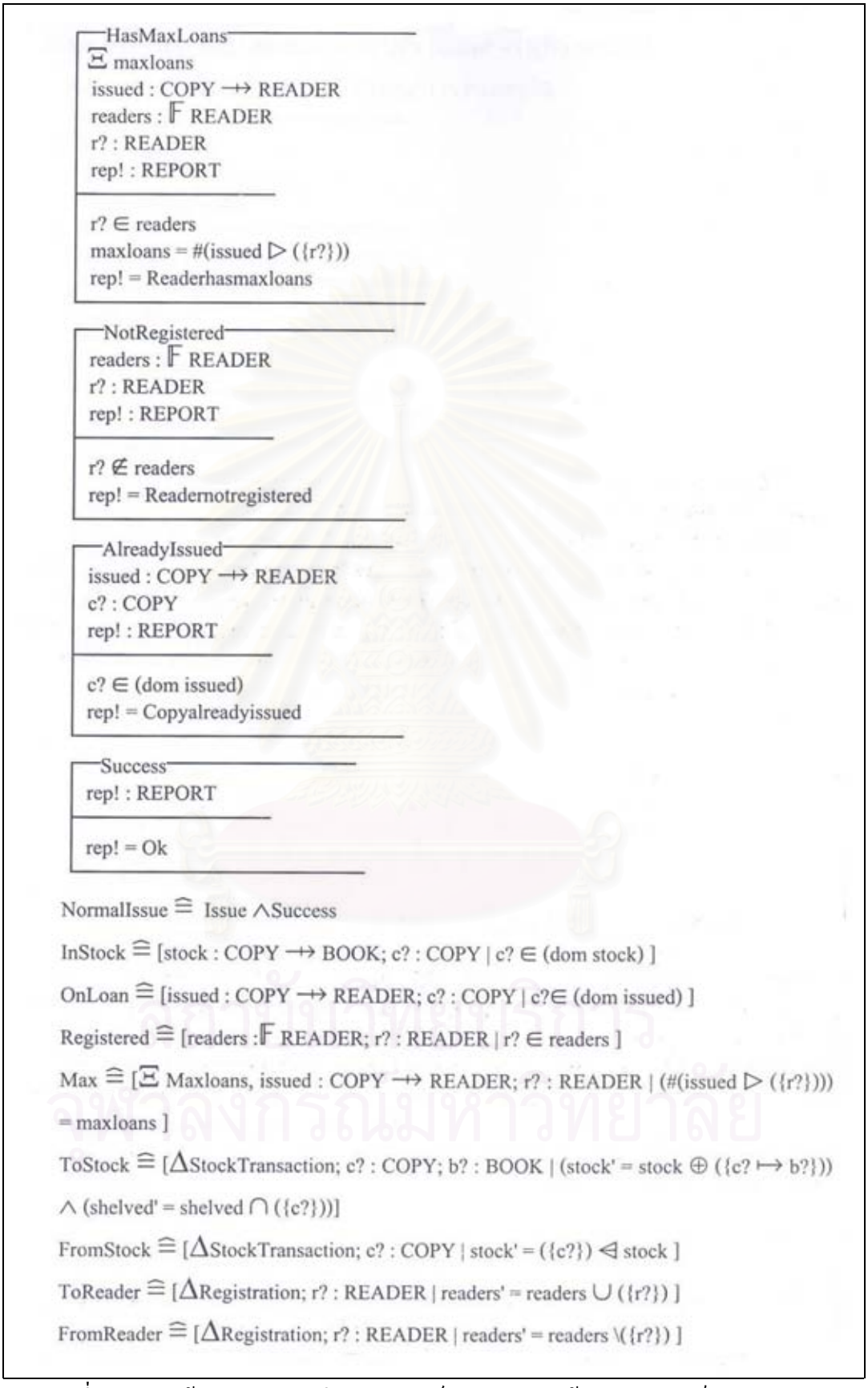


รูปที่ ค-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดขนาดเล็ก

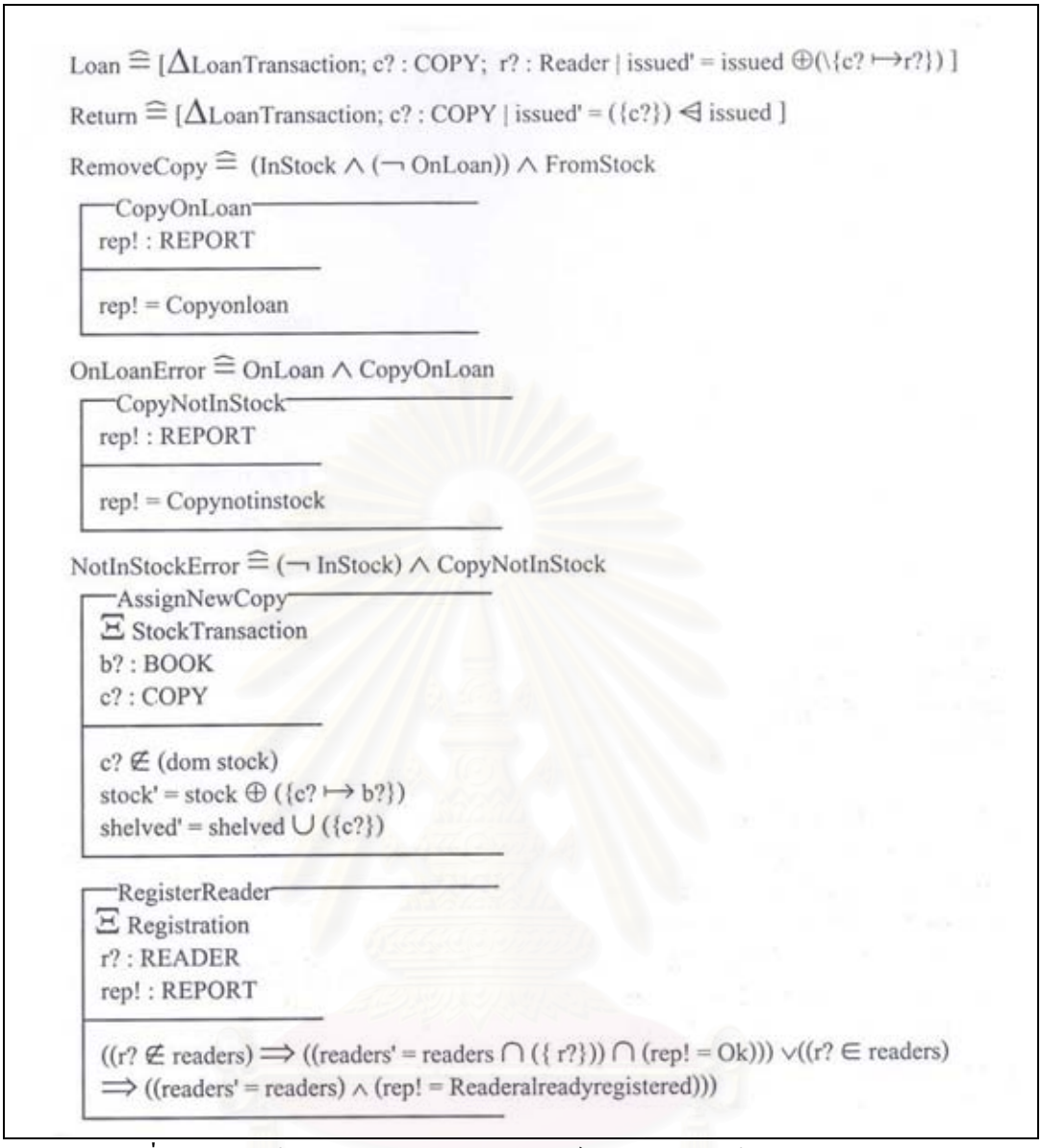


รูปที่ ค-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดขนาดเล็ก (ต่อ)

รูปที่ ค-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดขนาดเล็ก (ต่อ)



รูปที่ ค-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดขนาดเล็ก (ต่อ)



รูปที่ ค-1 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุด (ต่อ)

จากระบบห้องสมุด ในรูปสัญกรณ์เซด สามารถแปลงให้อยู่ในรูปแทคของลาเทกซ์ได้ ดังรูปที่ ค-2

```
\begin{zed}
[COPY,READER,BOOK]
\end{zed}
\begin{zed}
REPORT ::=Furthercopyadded|Newtitleadded| Readerhasmaxloans| Readernotregistered|
Copyalreadyissued| Copyonloan| Copynotinstock| Readeralreadyregistered| Ok
\end{zed}
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดในรูปแทคของลาเทกซ์

```
\begin{schema}{Maxloans}
maxloans:\nat
\where
maxloans<10
\end{schema}
\begin{schema}{Reader}
reader: \finset READER
\end{schema}
\begin{schema}{Copy}
copy: \finset COPY
\end{schema}
\begin{schema}{LibDB}
stock : COPY \pfun  BOOK  \\
readers : \finset READER
\end{schema}
\begin{schema}{LibLoans}
\Xi Maxloans \\
\Xi Reader \\
issued : COPY \pfun READER  \\
shelved : \finset COPY
\where
\forall r : reader @ (\#(issued \rres (\{r\})) \leq maxloans \\
(shelved \cap (\dom issued)) = \emptyset
\end{schema}
\begin{schema}{Library}
\Xi LibDB \\
\Xi LibLoans
\where
(\dom stock) = ((\dom issued) \cup shelved)  \\
(\ran issued) \subset readers
\end{schema}
\begin{schema}{init\_Library}
\Delta  Library
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
\where
stock' = \emptyset  \\
readers' = \emptyset  \\
issued' = \emptyset  \\
shelved' = \emptyset
\end{schema}
\begin{schema}{StockTransaction}
\Delta Library
\where
issued' = issued  \\
readers' = readers
\end{schema}
\begin{schema}{LoanTransaction}
\Delta Library
\where
stock' = stock  \\
readers' = readers
\end{schema}
\begin{schema}{Registration}
\Delta Library
\where
stock' = stock  \\
shelved' = shelved  \\
issued' = issued
\end{schema}
\begin{schema}{Issue}
\Xi Maxloans \\
\Delta  LoanTransaction \\
c? : COPY  \\
r? : READER
\where
c? \in shelved  \\
 r? \in readers \\
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
(\#(issued \rres (\{r?\})) < maxloans  \\
issued' = issued \oplus (\{c? \mapsto r?\})
\end{schema}
\begin{schema}{WhoHasCopy}
\Xi Library  \\
c? : COPY  \\
r! : READER
\where
c? \in (\dom issued)  \\
r! = issued~ c?
\end{schema}
\begin{schema}{AddKnownTitle}
\Xi Copy \\
\Delta Library  \\
b? : BOOK  \\
rep! : REPORT
\where
b? \in(\ran stock)  \\
\exists c:copy |(c \in copy) \land (c \notin (\dom stock)) @ (stock' = stock \oplus (\{c?
\mapsto b?\})) \land (shelved' = shelved\cup (\{c\}))  \\
issued' = issued \\
readers' = readers \\
rep! = Furthercopyadded
\end{schema}
\begin{schema}{AddNewTitle}
\Xi Copy \\
\Delta Library  \\
b? : BOOK  \\
rep! : REPORT
\where
b? \notin (\ran stock)  \\
\exists c:copy |(c \in copy) \land (c \notin (\dom stock)) @ (stock' = stock \oplus (\{c?
\mapsto b?\})) \land (shelved' = shelved\cup (\{c?\}))  \\
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
 issued' = issued \\
readers' = readers \\
rep! = Newtitleadded
\end{schema}
\begin{zed}
AddCopy1 \defs AddKnownTitle \lor AddNewTitle
\end{zed}
\begin{schema}{AddCopyReport}
stock : COPY \pfun BOOK \\
b? : BOOK \\
rep! : REPORT
\where
((b? \in (\ran stock)) \implies (rep! = Furthercopyadded))\lor((b? \notin (\ran stock)) \implies
(rep! = Newtitleadded))
\end{schema}
\begin{schema}{EnterNewCopy}
\Xi Copy \\
\Delta StockTransaction \\
b? : BOOK
\where
\exists c :copy | (c \in copy) \land (c \notin (\dom stock)) @ ((stock' = stock \oplus (\{c
\mapsto b?\}) \land (shelved' = shelved \cup (\{c\})))) \\
\end{schema}
\begin{zed}
AddCopy \defs EnterNewCopy \land AddCopyReport
\end{zed}
\begin{schema}{HasMaxLoans}
\Xi Maxloans \\
issued : COPY \pfun READER  \\
readers : \finset READER  \\
r? : READER  \\
rep! : REPORT
\where
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
r? \in readers  \\
maxloans = \#(issued \rres (\{r?\}))  \\
rep! = Readerhasmaxloans
\end{schema}
\begin{schema}{NotRegistered}
readers : \finset READER  \\
r? : READER \\
rep! : REPORT
\where
r? \notin readers  \\
rep! = Readernotregistered
\end{schema}
\begin{schema}{AlreadyIssued}
issued : COPY \pfun READER \\
c? : COPY \\
rep! : REPORT
\where
c? \in (\dom issued)  \\
rep! = Copyalreadyissued
\end{schema}
\begin{schema}{Success}
rep! : REPORT
\where
rep! = Ok
\end{schema}
\begin{schema}{NotRegistered}
readers : \finset READER  \\
r? : READER \\
rep! : REPORT
\where
r? \notin readers  \\
rep! = Readernotregistered
\end{schema}
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
\begin{schema}{AlreadyIssued}
issued : COPY \pfun READER \\
c? : COPY \\
rep! : REPORT
\where
c? \in (\dom issued)  \\
rep! = Copyalreadyissued
\end{schema}
\begin{schema}{Success}
rep! : REPORT
\where
rep! = Ok
\end{schema}
\begin{zed}
NormalIssue \defs  Issue \land Success
\end{zed}
\begin{zed}
InStock \defs [stock : COPY \pfun BOOK; c? : COPY | c? \in (\dom stock) ]
\end{zed}
\begin{zed}
OnLoan \defs [issued : COPY \pfun READER; c? : COPY | c? \in (\dom issued) ]
\end{zed}
\begin{zed}
Registered \defs [readers : \finset READER; r? : READER | r? \in readers ]
\end{zed}
\begin{zed}
Max \defs [\Xi Maxloans; issued : COPY \pfun READER; r? : READER | maxloans=(\#
(issued \rres (\{r?\}))) ]
\end{zed}
\begin{zed}
ToStock \defs [\Delta  StockTransaction; c? : COPY; b? : BOOK | (stock' = stock \oplus (\
{c? \mapsto b?\})) \land (shelved' = shelved \cup (\{c?\}))]
\end{zed}
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
\begin{zed}
FromStock \defs  [\Delta StockTransaction; c? : COPY | stock' = (\{c?\}) \ndres stock ]
\end{zed}
\begin{zed}
ToReader \defs [\Delta Registration; r? : READER | readers' = readers \cup (\{r?\}) ]
\end{zed}
\begin{zed}
FromReader \defs [\Delta Registration; r? : READER | readers' = readers? \setminus
(\{r?\}) ]
\end{zed}
\begin{zed}
Loan \defs [\Delta LoanTransaction; c? : COPY;  r? : READER | issued' = issued \oplus
(\{c? \mapsto r?\}) ]
\end{zed}
\begin{zed}
Return \defs [\Delta LoanTransaction; c? : COPY | issued' = (\{c?\}) \ndres issued ]
\end{zed}
\begin{zed}
RemoveCopy \defs  (InStock \land (\lnot OnLoan)) \land FromStock
\end{zed}
\begin{schema}{CopyOnLoan}
rep! : REPORT
\where
rep! = Copyonloan
\end{schema}
\begin{zed}
OnLoanError \defs OnLoan \land CopyOnLoan
\end{zed}
\begin{schema}{CopyNotInStock}
rep! : REPORT
\where
rep! = Copynotinstock
\end{schema}
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

```
\begin{zed}
NotInStockError \defs (\lnot InStock) \land CopyNotInStock
\end{zed}
\begin{schema}{AssignNewCopy}
\Delta StockTransaction \\
b? : BOOK \\
c? : COPY
\where
c? \notin (\dom stock) \\
stock' = stock \oplus (\{c? \mapsto b?\})  \\
shelved' = shelved \cup (\{c?\})
\end{schema}
\begin{schema}{RegisterReader}
\Xi Registration \\
r? : READER  \\
rep! : REPORT
\where
((r? \notin readers) \implies ((readers' = readers \cup (\{ r?\})) \land (rep! = Ok) ))\lor((r? \in
readers) \implies ((readers' = readers) \land (rep! = Readeralreadyregistered)))
\end{schema}
```

รูปที่ ค-2 แสดงข้อกำหนดรูปนัยในรูปสัญกรณ์เซตของระบบห้องสมุดในรูปแทคของลาเทกซ์ (ต่อ)

จากระบบห้องสมุด ซึ่งอยู่ในรูปแทคของลาเทกซ์ดังรูปที่ ค-2 สามารถใช้เครื่องมือซอฟต์แวร์แปลงให้อยู่ในรูปโปรแกรมโพรล็อกได้ดังรูป ค-3

```
:-use_module(library(z)).
maxloans(Maxloans):- <(Maxloans,10).
reader(Reader).
copy(Copy).
libDB(Stock,Readers).
libLoans(Issued,Shelved,Maxloans,Reader):- maxloans(Maxloans),reader(Reader),Temp5=
    (Temp1 = Reader,member(R,Temp1),\+((Temp4 = [R],rres(Issued,Temp4,Temp3),#(
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด

```
    Temp3,Temp2),leq(Temp2,Maxloans)))),\+(Temp5),dom(Issued,Temp7),cap
    (Shelved,Temp7,Temp6),Temp6=[].
library(Stock,Readers,Issued,Shelved,Maxloans,Reader):- libDB(Stock,Readers),libLoans
    (Issued,Shelved,Maxloans,Reader),dom(Stock,Temp8),dom(Issued,Temp10),cup
    (Temp10,Shelved,Temp9),Temp8 = Temp9,ran(Issued,Temp11),subset
    (Temp11,Readers).
init_Library(Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shel
    vedp,Maxloansp,Readerp):- library(Stock,Readers,Issued,Shelved,Maxloans,Reader),
    Stockp=[],Readersp=[],Issuedp=[],Shelvedp=[].
stockTransaction(Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp
    ,Shelvedp,Maxloansp,Readerp):- library(Stock,Readers,Issued,Shelved,Maxloans,
    Reader),Issuedp = Issued,Readersp = Readers.
loanTransaction(Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,
    Shelvedp,Maxloansp,Readerp):- library(Stock,Readers,Issued,Shelved,Maxloans,
    Reader),Stockp = Stock,Readersp = Readers.
registration(Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shel
    vedp,Maxloansp,Readerp):- library(Stock,Readers,Issued,Shelved,Maxloans,
    Reader),Stockp = Stock,Shelvedp = Shelved,Issuedp = Issued.
issue(C,R,Maxloans,Stock,Readers,Issued,Shelved,Reader,Stockp,Readersp,Issuedp,Shelve
    dp,Maxloansp,Readerp):- maxloans(Maxloans),loanTransaction(Stock,Readers,
    Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Reader
    p),in(C,Shelved),in(R,Readers),Temp14 = [R],rres(Issued,Temp14,Temp13),#
    (Temp13,Temp12),<(Temp12,Maxloans),Temp15 = [(C , R)],
    oplus(Issued,Temp15,Issuedp).
whoHasCopy(C,Stock,Readers,Issued,Shelved,Maxloans,Reader,R):- library
    (Stock,Readers,Issued,Shelved,Maxloans,Reader),dom(Issued,Temp16),in
    (C,Temp16),function(Issued,C,R).
addKnownTitle(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Reade
    rsp,Issuedp,Shelvedp,Maxloansp,Readerp):- copy(Copy),library(Stock,Readers,Issued,
    Shelved,Maxloans,Reader),ran(Stock,Temp17),in(B,Temp17),Temp18 = Copy,member
    (C,Temp18),in(C,Copy,Temp20),dom(Stock,Temp22),notin(C,Temp22,Temp21),land
    (Temp20,Temp21,Temp19),is2((Temp25 = [(C , B)],
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด (ต่อ)

```
    oplus(Stock,Temp25,Stockp)),Temp24),is2((Temp27 = [C],cup(Shelved,Temp27,
    Shelvedp)),Temp26),land(Temp24,Temp26,Temp23),land(Temp19,Temp23),Issuedp =
    Issued,Readersp = Readers,Rep = furthercopyadded.
addNewTitle(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Readers
    p,Issuedp,Shelvedp,Maxloansp,Readerp):- copy(Copy),library(Stock,Readers,
    Issued,Shelved,Maxloans,Reader),ran(Stock,Temp28),notin(B,Temp28),Temp29 =
    Copy,member(C,Temp29),in(C,Copy,Temp31),dom(Stock,Temp33),notin
    (C,Temp33,Temp32),land(Temp31,Temp32,Temp30),is2((Temp36 = [(C , B)],oplus
    (Stock,Temp36,Stockp)),Temp35),is2((Temp38 = [C],cup(Shelved,Temp38,
    Shelvedp)),Temp37),land(Temp35,Temp37,Temp34),land(Temp30,Temp34),Issuedp =
    Issued,Readersp = Readers,Rep = newtitleadded.
addCopy1(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Readersp,Is
    suedp,Shelvedp,Maxloansp,Readerp):-
    Schema1=addKnownTitle(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep
    ,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Schema2=addNewTitle
    (B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Readersp,Issuedp,
    Shelvedp,Maxloansp,Readerp),lor(Schema1,Schema2).
addCopyReport(Stock,B,Rep):- ran(Stock,Temp41),in(B,Temp41,Temp40),is2((Rep =
furthercopyadded),Temp42),implies(Temp40,Temp42,Temp39),ran(Stock,Temp45),notin
    (B,Temp45,Temp44),is2((Rep =
    newtitleadded),Temp46),implies(Temp44,Temp46,Temp43),lor(Temp39,Temp43).
enterNewCopy(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,I
    ssuedp,Shelvedp,Maxloansp,Readerp):- copy(Copy),stockTransaction(Stock,Readers,
    Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Reader
    p),Temp47 = Copy,member(C,Temp47),in(C,Copy,Temp49),dom(Stock,Temp51),notin
    (C,Temp51,Temp50),land(Temp49,Temp50,Temp48),is2((Temp55 = [(C , B)],oplus
    (Stock,Temp55,Stockp)), Temp54),Temp53 = Temp54,Temp52 = Temp53,land
    (Temp48,Temp52).
addCopy(B,Copy,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Readersp,Iss
    uedp,Shelvedp,Maxloansp,Readerp):- Schema1=enterNewCopy(B,Copy,Stock,
    Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloans
    p,Readerp),Schema2=addCopyReport(Stock,B,Rep),land(Schema1,Schema2).
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด (ต่อ)

```
hasMaxLoans(Issued,Readers,R,Maxloans,Rep):- maxloans(Maxloans),in(R,Readers),
    Temp57 = [R],rres(Issued,Temp57,Temp56),#(Temp56,Maxloans),Rep =
    readerhasmaxloans.
notRegistered(Readers,R,Rep):- notin(R,Readers),Rep = readernotregistered.
alreadyIssued(Issued,C,Rep):- dom(Issued,Temp58),in(C,Temp58),Rep =
    copyalreadyissued.
success(Rep):- Rep = ok.
normalIssue(C,R,Maxloans,Stock,Readers,Issued,Shelved,Reader,Rep,Stockp,Readersp,Iss
    uedp,Shelvedp,Maxloansp,Readerp):- Schema1=issue(C,R,Maxloans,Stock,Readers,
    Issued,Shelved,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Schema
    2=success(Rep),land(Schema1,Schema2).
inStock(Stock,C):- dom(Stock,Temp59),in(C,Temp59).
onLoan(Issued,C):- dom(Issued,Temp60),in(C,Temp60).
registered(Readers,R):- in(R,Readers).
max(Issued,R,Maxloans):- maxloans(Maxloans),Temp63 =
    [R],rres(Issued,Temp63,Temp62),#(Temp62,Temp61),Maxloans = Temp61.
toStock(C,B,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shel
    vedp,Maxloansp,Readerp):- stockTransaction(Stock,Readers,Issued,Shelved,Maxloans,
    Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),is2((Temp65=[(C, B)],
    oplus(Stock,Temp65,Stockp)),Temp64),is2((Temp67 = [C],cup(Shelved,Temp67,
    Shelvedp)),Temp66),land(Temp64,Temp66).
fromStock(C,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,She
    lvedp,Maxloansp,Readerp):- stockTransaction(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp68 =
    [C],ndres(Temp68,Stock,Stockp).
toReader(R,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shel
    vedp,Maxloansp,Readerp):- registration(Stock,Readers,Issued,Shelved,Maxloans,
    Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp69 = [R],cup
    (Readers,Temp69,Readersp).
fromReader(R,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,S
    helvedp,Maxloansp,Readerp):- registration(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp70 =
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด (ต่อ)

```
    fromReader(R,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issued
    p,Shelvedp,Maxloansp,Readerp):- registration(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp70 =
    [R],setminus(Readers,Temp70,Readersp).
loan(C,R,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shelve
    dp,Maxloansp,Readerp):- loanTransaction(Stock,Readers,Issued,Shelved,Maxloans,
    Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp71 = [(C ,
    R)],oplus(Issued,Temp71,Issuedp).
return(C,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,Shelved
    p,Maxloansp,Readerp):- loanTransaction(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),Temp72 =
    [C],ndres(Temp72,Issued,Issuedp).
removeCopy(Stock,C,Issued,Readers,Shelved,Maxloans,Reader,Stockp,Readersp,Issuedp,S
    helvedp,Maxloansp,Readerp):- Schema1=inStock(Stock,C),Schema2=onLoan
    (Issued,C),Schema3=fromStock(C,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stoc
    kp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),lnot(Schema2,Temp74),land
    (Schema1,Temp74,Temp73),land(Temp73,Schema3).
copyOnLoan(Rep):- Rep = copyonloan.
onLoanError(Issued,C,Rep):- Schema1=onLoan(Issued,C),Schema2=copyOnLoan
    (Rep),land(Schema1,Schema2).
copyNotInStock(Rep):- Rep = copynotinstock.
notInStockError(Stock,C,Rep):- Schema1=inStock(Stock,C),Schema2=copyNotInStock
    (Rep),lnot(Schema1,Temp75),land(Temp75,Schema2).
assignNewCopy(B,C,Stock,Readers,Issued,Shelved,Maxloans,Reader,Stockp,Readersp,Issu
    edp,Shelvedp,Maxloansp,Readerp):- stockTransaction(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),dom
    (Stock,Temp76),notin(C,Temp76),Temp77 = [(C,B)],oplus(Stock,Temp77,Stockp),
    Temp78 = [C],cup(Shelved,Temp78,Shelvedp).
registerReader(R,Stock,Readers,Issued,Shelved,Maxloans,Reader,Rep,Stockp,Readersp,Iss
    uedp,Shelvedp,Maxloansp,Readerp):- registration(Stock,Readers,Issued,Shelved,
    Maxloans,Reader,Stockp,Readersp,Issuedp,Shelvedp,Maxloansp,Readerp),notin
    (R,Readers,Temp80),is2((Temp83 = [ R],cup(Readers,Temp83,Readersp)),Temp82),is2
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด (ต่อ)

```
((Rep = ok),Temp84),land(Temp82,Temp84,Temp81),
implies(Temp80,Temp81,Temp79),in(R,Readers,Temp86),is2((Readersp =
Readers),Temp88),is2((Rep = readeralreadyregistered),Temp89),land
(Temp88,Temp89,Temp87),implies(Temp86,Temp87,Temp85),lor(Temp79,Temp85).
```

รูปที่ ค-3 แสดงโปรแกรมโพรล็อกของระบบห้องสมุด (ต่อ)

ภาคผนวก ง. แทคของ Z/EVES

เนื่องจากการแปลงข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้นข้อกำหนดรูปนัยในรูปสัญกรณ์เซดที่จะนำมาแปลงนั้นจะต้องมีความถูกต้องทางด้านวากยสัมพันธ์ก่อนจึงจะนำมาใช้กับเครื่องมือซอฟต์แวร์ได้ ดังนั้นผู้จัดทำจึงได้กำหนดเครื่องมือที่ใช้ในการตรวจสอบทางด้านวากยสัมพันธ์คือ Z/EVES ซึ่งก่อนที่จะนำข้อกำหนดรูปนัยในรูปสัญกรณ์เซดมาตรวจสอบวากยสัมพันธ์ต้องมีการแปลงให้อยู่ในรูปแทคของลาเทกซ์ของ Z/EVES ก่อนดังตารางที่ ง-1

ตารางที่ ง-1 แสดงแทคของลาเทกซ์ของสัญกรณ์เซด

Schema			**Relation**		
Name	**Z Notation**	**LATEX**	**Name**	**Z Notation**	**LATEX**
Axiomatic box	D P	\begin{axdef} D \where P \end{schema}	maplet	x ↦ y	x \mapsto y
			domain	dom x	\dom x
			range	ran x	\ran x
schema box	S D P	\begin{schema}{S} D \where P \end{schema}	composition	x ⨾ y	x \comp y
			backward composition	x ∘ y	x \circ y
horizontal schema	S ≙ R	S \defs R	domain restriction	x ◁ y	x \dres y
set comprehension	{D|P• E}	\{D|P@E\}	range restriction	x ▷ y	x \rres y
universal quantification	∀ ST• S	\forall ST @ S	domain anri restriction	x ⩤ y	x \ndres y
existential quantification	∃ ST• S	\exists ST @ S	range anti restriction	x ⩥ y	x \nrres y
Set			relational inverse	x~	\inv x
inequality	x ≠ y	x \neq y	relational image	x⦇y⦈	x\limg y \ring
non-Membership	x ∉ y	x \notin y	overriding	x ⊕ y	x \oplus y
emptyset	∅	\emptyset	**Bags**		
subset	x ⊂ y	x \subset y	multiplicity	x#y	x \bcount y
set union	x ∪ y	x \cup y	bags scaling	x⊗ y	x \otimes y
set intersection	x ∩ y	x \cap y	bags membership	x in y	x \inbag y
set difference	x \ y	x \setminus y	sub bags	x⊏y	x \subbag y
generalized union	∪ x	\bigcup x	bags union	x⊎y	x \uplus y

ตารางที่ ง-1 แสดงแทคของลาเทกซ์ของสัญกรณ์เซต(ต่อ)

Set			Bags		
Name	**Z Notation**	**LATEX**	**Name**	**Z Notation**	**LATEX**
generalized intersection	⋂ y	\bigcap y	bags difference	x ⨃ y	x \uminus y
			Other		
first	first x	first~x	addition	x + y	x + y
second	second x	second~x	substraction	x – y	x – y
Sequence			multiplication	x * y	x * y
concatenation	x ⌢ y	x \cat y	division	x div y	x \div y
reversal	rev x	rev~ x	modulus	x mod y	x \mod y
head	head x	head~x	less than	x < y	x < y
last	last s	last~x	less than or equal	x <= y	x \leq y
tail	tail x	tail~x	greater than	x > y	x > y
front	front x	front~x	greater than or equal	x >= y	x \geq y
extraction	x ↾ y	x \extract y			
filter	x ⇂ y	x \filter y	number range	x..y	x \upto y
squash	squash x	squash~x	min	min x	min~x
distributed concatenation	⌢/ x	\dcat x	max	max x	max~x
			conjunction	x ∨ y	x \lor y
			disjunction	x ∧ y	x \land y
			implication	x ⟹ y	x \implies y
			equivalence	x ⟛ y	x \iff y

ภาคผนวก จ. ผลงานตีพิมพ์

ผลงานนี้ได้ถูกตีพิมพ์ในงาน The 4th Annual National Symposium on Computational Science and Engineering ซึ่งได้จัดขึ้นที่มหาวิทยาลัยเกษตรศาสตร์ในวันที่ 27-29 มีนาคม 2543

Simulating Software Requirements Specification in Z

Nucharee Lapjitgkusol, Wiwat Vatanawood and Wanchai Rivepiboon
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand
+66 2 2186956, +66 2 2186957
Email: g41nlj@cp.eng.chula.ac.th, wiwat@chula.ac.th, wanchai.r@chula.ac.th

ABSTRACT
An alternative of simulating software requirements specification in Z is proposed using the modified procedural approach. We present several algorithms to prepare Z specification in appropriate order and to translate into Prolog program. In the early stage, the software developer will be provided with a scheme to demonstrate the functional behavior of the target software system from requirements specification. In addition, a set of Prolog rules is designed for each Z notation as to simplify the translating process.

1. INTRODUCTION

Formal specification becomes widely used in software engineering because of its advantages. Formal specification is not only precise and unambiguous but also easily found errors since discovering errors late takes high costs to correct them which is the problem of natural language. Formal specification defines all characteristics of system to be developed so that it can be as part of the contract between customer and developer as well[3]. Among these advantages, formal specification still has some problems. Using mathematical notation makes it difficult to understand especially for the beginner and customer[2].

One approach to solve this problem is simulating formal specification using executable language. The simulated specification can demonstrate all characteristics of systems including functionality[5]. The customer can understand specification easily and give feedback to developer immediately. Consequently, it helps developer to reduce errors in writing specification. Moreover, this approach can validate the specification before implementation[2].

In general, there are two research concepts to simulate Z specification: direct and indirect. The first concept such as Zans [4], does not use the other language to simulate specification. Specification is executed to demonstrate the result for users directly so there is no source code. The latter concept uses other executable language to simulate specification. Specification is transformed to other executable language, such as Prolog [18][9], Lisp [20], Ada [21], Mercury[22] and SQL [23], and executed.

This paper proposes an alternative to simulate Z specification using Prolog bases on [18][9]. While Z notation bases on mathematical and predicate logic, Prolog is executable programming language that based on first order predicate logic as well. Thus, using Prolog could reduce the complicated method in simulating.

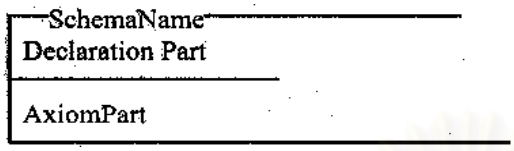
This paper is organized as follows. Section 2 introduces Z notation. Section 3 describes existing approaches to simulate Z specification using Prolog. Section 4 is birthday book case study. Section 5 describes method to simulate Z specification. Section 6 is conclusion.

2. Z NOTATION[11][25]

2.1 Z Schema

Using Z notation, specification is divided into modules called *schema*. Each schema has 3 parts, the schema name, declaration part and axiom part. The schema can be written in 2 formats, vertical and horizontal, as the following:

Vertical format:



Horizontal format:

SchemaName | [*Declaration Part* | *Axiom Part*]

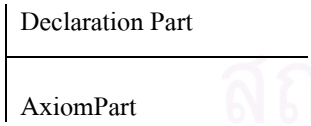
The schema name consists of letters and digits, starts with uppercase letter. It becomes global in the specification. The declaration part defines all variables and other inclusion schemas, which are used in this schema. Each variable has a name and type. The variable consists of letter and always starts with lowercase letter. The type of variable appears in the right side of ':'. It consists of letters and start with uppercase. Although, the variables are local, other schema can used them by schema inclusion. The axiom part indicates the operations of schema, which is an optional. Each operation is called predicate which is connected by operators 'and', 'or', 'imply' and 'equivalence'. Each predicate refers to one or more variables in declaration part.

2.2 Decoration

The variables can be decorated by the symbol ?(input), !(output) and ə(after state). The after state variable is used when the value of a variable is changed. In Z notation, the sentence 'x:=x+1' is wrong. It changes to 'xə=x+1', that xə is the after state of x.

2.3 Global variables

The variables in declaration part are local variables for each schema. In Z notation, the global variables are declared in the axiomatic box as the following:



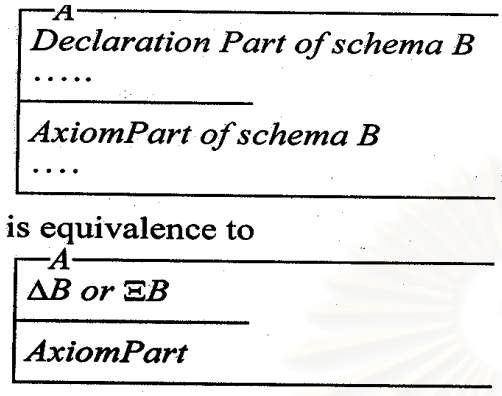
It looks like the Z schema but there are no line at the top and the bottom. The declaration part defines variables and type, which is similar to the Z schema. The Axiom part is the constraint of the variables such as 'date ⌉0'.

2.4 Types

The basic types that Z notation defines are integer(Z), natural number(N) and positive integer(N_1). Otherwise, user can define abstract data type as a given set. The given set name is uppercase letter. For example, the given set *Name* can defined as [NAME]. The declaration likes 'n:NAME'.

2.5 Schema inclusion

In Z notation, schema can reuse other schemas by referencing their names in the declaration part, which is called schema inclusion. There are two kinds of schema inclusion: state changed schema(Δ) and state unchanged schema(Ξ). The state changed schema has the after state variables of the included schema. For example, schema *A* includes schema *B*, then schema *A* is shown as follows:



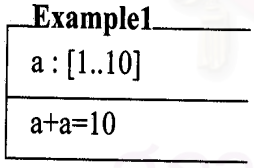
3. EXISTING APPROACHES

There are two approaches to simulating Z specification using Prolog: generate-and-test approach and procedural approach. The first approach is presented by Magaret M. West and Barry M. Eaglestone in [9] and the latter presented by M.A. Hewitt in [18]

3.1 The generate-and-test approach.

The generate-and-test approach transforms both declaration part and axiom part of Z schema to Prolog. The values of variables are trially generated from declaration part and tested by the predicate in axiom part. While the result of testing is false, other new values are generated again. See the example below.

Example1:



The schema Example1 can be translated to the following Prolog program:

example1(A):- givenset(A,[1..10]), A is 10.

From the Prolog program, the predicate ‘givenset(A,[1..10])’ trially generates the first value of given set, which is 1, to the variable *A*. Then the variable *A* is tested by the predicate ‘A+A is 10’ which is false. Consequently, the program backtracks to instantiate the next new value from given set and have it tested again. The program has to backtrack 10 times, variable A is finally instantiated to 10, the predicate will be true.

3.2 The procedural approach.

The procedural approach is similar to the generate-and-test approach but the procedural approach transforms only axiom part of Z schema. See example below:

Example2:

Example2

a,b : [1..10]

---

b=10 ∧ ----(1)
a=b+15 -----(2)

The schema Example2 can be translated to following Prolog program

example2(A,B):- B is 10, A is B+15.

From this example the program does not backtrack to find the result. Consequently, this approach takes time less than the generate-and-test approach. However, because of using only axiom part and Prolog processing is sequential, the sequence of predicate is important for transforming to Prolog. If the predicates in axiom part are in the wrong order, they cause errors in the program. See the example below:

Example3:

Example3

a,b : [1..10]

---

a=b+15 ∧ ----(1)
b=10 ----(2)

From the Example2, we change the order of predicate to the example3 and transform to following Prolog program

example3(A,B):- A is B+15, B is 10.

From the example3, the predicate 'A is B+15' causes an error because the variables *A* and *B* are not instantiated. So the program can not find the value of *A*.

4. BIRTHDAY BOOK:CASE STUDY

In this paper, we present birthday book[1][8] as a case study. The birthday book is the system that records the birthday of friends. It keeps the name of person and his birthday. The user can add and find friend's birthday from system. The system is able to remind the user that today is whose birthday. The birthday book system can be represented by the following Z specification:

Birthdaybook

birthday? : Name⇸Date
known! : ℙName

---

known!=dom(birthday?)

The schema *Birthdaybook* declares *birthday* as the one-to-one function from *Name* to *Date* and *known* as power set of *Name*. This schema finds domain of *birthday* and keeps in *known*.

InitBirthdayBook

ΞBirthdaybook

---

known=$\emptyset$

The schema *InitBirthdayBook* initializes birthday book by setting *known* to empty set, so *birthday* in schema *Birthdaybook* is empty too.

Addbirthday

ΔBirthdaybook
name? : Name
date? : Date

---

name?$\notin$ known! $\wedge$
knownp = dom(birthdayp) $\wedge$
birthday′=birthday? $\cup$ {name? $\rightarrowtail$date?}

The schema *Addbirthday* adds the new *name?* and *date?* to system. Before adding, the schema checks whether there is the same *name?* in *birthday* or not because *birthday* is one-to-one function.

Findbirthday

ΞBirthdaybook
name? : Name
date! : Date

---

name?$\in$ known! $\wedge$
date!= birthday?(name?)

The schema *Findbirthday* is used to find the *date!* of the input *name?*. Before finding, the schema checks whether there is the *name*? in *birthday* or not.

Remindbirthday

ΞBirthdaybook
today? : Date
cards! : $\mathbb{P}$Name

---

cards!={n:known | birthday(n)=today?}

The schema *Remindbirthday* reminds the user that *today*? is whose birthday because the person can have the same birthday.

5. SIMULATING Z SPECIFICATION USING PROLOG

In our approach, we divide the processes of simulating Z specification using Prolog into 3 subprocesses, 1) Create Prolog library, 2) Reorder Z specification and 3) Translate Z to Prolog as shown in figure 1.

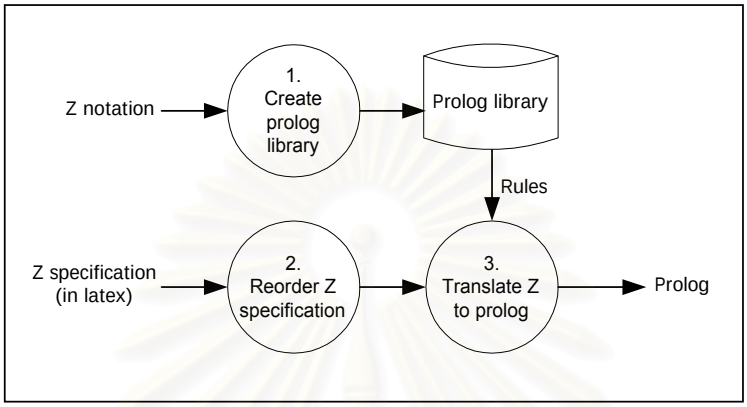


Figure 1: A scheme of simulating Z specification using Prolog.

5.1 Create Prolog library.

The first subprocess is to create library of Prolog. Each Z notation is assigned with a set of Prolog's rules. For example: The member(ε) symbol of Z notation can be translated to Prolog rules as follows:

```
member(X,[X|_]).
member(X,[_|L]):-member(X,L).
```

Consequently, we can translate predicate 'aεb' to 'member(A,B)' in Prolog.

5.2 Reorder Z specification.

In general, the order of predicates in Z schema is not significant. Unfortunately, since Prolog is sensitive to the order of predicate rules, reordering predicates is needed. The algorithm for reordering predicate is shown as follows:

Algorithm 1: Reorder Z specification

Let *Input* be the set of variables that are instantiated.
Output be the set of variables that are not instantiated.
P be the set of the predicates in axiom part.
Var be the set of variables in each predicate.
v? be input variable
v! be output variable
p be the predicate in axiom part

a) Consider the declaration part; then put all input variables *v*? into *Input* set and put all variables *v*! to *Output* set. Put all predicate *p* in axiom part to *P* set.
b) While (*P* is not empty set)
   If *p* is negative predicate

{ **If** *v* in *p* are in *Input* set **then**
{ put *p* in axiom part,
delete *p* from set *P*. }
}
else
{ **If** *v* in the right hand side of operators is in *Input* set **then**
{ put the *v* in left hand side of operators to *Input* set,
delete the *v* in left hand side of operators from *Output* set,
put *p* in axiom part,
delete *p* from set *P*. }
}

Example4:

Addbirthday1

known!,known′ : ℙName
birthday?,birthday′ : Name⇸Date
name? : Name
date? : Date

---

known!=dom(birthday?) ----------(1)
known′=dom(birthday′) ---------- (2)
name?∉ known! ---------- (3)
birthday′=birthday? ∪{name? ↦date?}-- (4)

Considering the schema *Addbirthday1*, the line(2) is in the inappropriate order because *birthday*ə and *known!* are output variables that are not instantiated. Thus, line (2) should be moved to the line after line(4) as shown below in the schema *Addbirthday2:*

Addbirthday2

known!,known′ : ℙName
birthday?,birthday′ : Name⇸Date
name? : Name
date? : Date

---

known!=dom(birthday?) ---------(1)
name?∉ known! ---------(3)
birthday′ =birthday?∪{name?↦date?}-----(4)
known′ =dom(birthday′) ---------(2)

5.3 Translate Z to Prolog

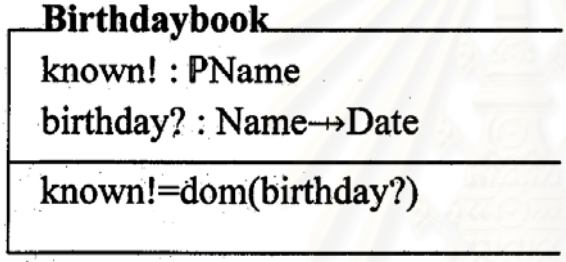
We modified the translating scheme of procedural approach in [18]. We still exploit the variables and the inclusion feature in the declaration part. Unlike procedural approach, during translating, inclusion schema will be performed before the predicate of schema. Moreover we use the variables in declaration part as arguments of Prolog rules. The algorithm for translating Z specification to Prolog is shown as follows:

Algorithm 2: Translate Z to Prolog

Let *SchName* be the name of schema
Var be the variables in declaration part of *SchName*
P be the predicate in axiom part of *SchName*
InSchChange be the changed state inclusion schema
InSchUn be the unchanged state inclusion schema
InChVar be all variables of *InSch*
InUnvar be all variables of *InSchUn*

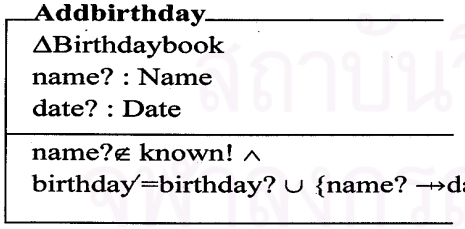
a) *SchName* will be name of rules.
b) *Var* will be arguments of rules.
c) For *InSchChange*, *InChVar* and after state of *InChVar* will be arguments of rules.
d) For *InSchUn*, *InUnVar* will be arguments of rules.
e) Name and arguments of *InSchChange* rules and *InSchUn* rules will be condition of this rule.
f) Each *P* will be condition of rules with some syntactical transformation.

Example5:



Considering the schema *Birthdaybook*, *Birthdaybook* becomes the name of rule,*birthdaybook*. The variables in declaration part, *known* and *birthday*, become arguments of rule *birthdaybook*, *Known* and *Birthday*. The predicate 'known!=dom (birthday?)' becomes 'dom(Birthday,Known)' as an condition of rule *birthdaybook*. The rule *birthdaybook* is shown as follows:

birthdaybook(Known,Birthday):- dom(Birthday,Known).



From schema *Addbirthday*, *Addbirthday* becomes the name of rule, *addbirthday*. The variables in declaration part, *name* and *date*, become arguments of rule *addbirthday*, *Name* and *Date*. The variables and the after state of variables of *birthdaybook* rule become arguments of *addbirthday* rule. The head of *birthdaybook* rule becomes condition of *addbirthday* rule. The predicates 'name?™known!', 'birthdayɜ=birthday?Y{name?♣date?}' become '/+(member(Name,Known))', 'union (Birthday,(Name,Date),Birthdayp)' as conditions of rule. The rule *addbirthday* is shown as follows:

```
addbirthday(Known,Birthday,Birthdayp,Name,Date,Knownp):-
        birthdaybook(Known.Birthday),
        \+(member(Known,Name)),
        union(Birthday,(Name,Known),Birthdayp).
```

From the algorithm1 and algorithm2, we can translate the birthday book system in Z specification to the Prolog program shown in the figure 2:

```
birthdaybook(Known,Birthday):- dom(Birthday,Known).
addbirthday(Known,Knownp,Birthday,Birthdayp,Name,date):-
        birthdaybook(Known,Birthday),
        /+(member(Known,Name)),
        union(Birthday,(Name,Date),Birthdayp),
        dom(Birthdayp,Knownp).
findbirthday(Known,Birthday,Knownp,Birthdayp,Name,Date):-
        birthdaybook(Known,Birthday),
        member(Known,Name),
        function(Birthday,Name,Date).
remindbirthday(Known,Birthday,Knownp,Birthdayp,Today,Cards):-
        birthdaybook(Known,Birthday),
        member(Known,N),
        function(Birthday,N,Today),
        append(N,[],Cards).
```

Figure2: The Prolog's rules for birthday book system.

6. CONCLUSION

In this paper, an alternative of simulating Z is proposed by using Prolog language. Each Z notation is assigned with set of Prolog rules. Several algorithms are proposed to prepare Z specification in appropriate order and translate into the final Prolog program. We expect that the validation of Z specifications can be conducted by execute the final Prolog program. The developer will be able to demonstrate the functional behavior of the target system in the early stage. Our future work is to enhance our algorithms to cope with compound expression in a predicate.

REFERENCE:

[14] Bowen, J.P., "Formal specification in Z as a design and document tool", Software Engineering,1988 Software Engineering 88., Second IEE/BCS Conference:1988, Page(s):164-168

[15] Fuchs, N.E., "Specifications are (preferably) executable", Software Engineering Journal Volume: 7 5 September 1992 Page(s):323-333

[16] Utting, M., "Animating Z: interactivity, transparency and equivalence", Software Engineering Conference,1995. Proceedings,1995 Asia Pacific,1995, Page(s):297-303

[17] Jia, X., "An approach to animating Z specification", Computer Software and applications conference, 1995. COMPSAC '95. Proceedings, Nineteenth Annual International, Page(s):108-113

[18] Hewitt, M.A., "Automated animation of Z using Prolog", Department of computing, Lancaster university, Page(s):1-52

[19] West, M.M., Eaglestone,B.M.,"Software development: two approaches to animation of Z specification using Prolog", Software Engineering Journal Volume: 7 4 July 1992, Page(s):264-267

[20] Morrey, I., Soddiqi, J., Briggs, J., Buckerry, G., "A lisp-based environment for animating Z specification", Automating Formal Methods for Computer Assisted Prototyping,IEE Colloquium on page(s): 1pp.

[21] Morrey, I., Soddiqi, J., Hibberd, R., Buckerry, G., "Towards case tools for prototyping Z specification", Computer-Aided Software Engineering 1993. CASE'93, Proceeding of the sixth International workshop on,1993, Page (s):199-173

[22] Winikoff, M, "Animating Z using logic programming techniques", Seminar at the University of Queensland's Software Verification Research Centre, 1998

[23] Love, M., "Using oralcle/SQL to animate Z specification", Automating Formal Methods for Computer Assisted Prototyping,IEE Colloquium on page (s): 4pp. 1992

[24] Ratclife, B., "Introducing specification using Z:A practical case study approach", The McGraw-Hill International (UK) Limited, 1994

[25] Barden, R., Stepney, S., Cooper, D., "Z in practice", Pentice-Hall International,1994

[26] Spivey, J.M., "An introduction to Z and formal specification", Software Engineering Journal Volume: 4 1 January 1989, Page(s):40-50

[27] Spivey, J.M., "The Z notation: a reference manual", Practice Hall, 1989

ประวัติผู้เขียน

นางสาวนุชรี ลาภจิตรกุศล เกิดเมื่อวันที่ 27 เมษายน พ.ศ. 2520 ที่จังหวัดนครราชสีมา สำเร็จการศึกษาหลักสูตรวิทยาศาสตร์บัณฑิต (วท.บ.) สาขาวิทยาศาสตร์คอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยหอการค้าไทย เมื่อปีการศึกษา 2540 และเข้าศึกษาต่อหลักสูตรวิทยาศาสตร์มหาบัณฑิต (วท.ม.) ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2541