การสร้างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจากโปรแกรมภาษาซี

นาย เรืองยศ วรเจนวณิชย์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2543
ISBN 974-346-883-8

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

MR. REUNGYOS VORAJENWANICH

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 974-346-883-8

หัวข้อวิทยานิพนธ์	การสร้างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจากโปรแกรมภาษาซี
โดย	นายเรืองยศ วรเจนวณิชย์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(อาจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์)

.....อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์วิวัฒน์ วัฒนาวุฒิ)

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

.....กรรมการ
(อาจารย์ ดร.พรศิริ หมื่นไชยศรี)

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

เรืองยศ วรเจนวณิชย์ : การสร้างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจากโปรแกรมภาษาซี (A CONSTRUCTION OF FORMAL SPECIFICATION IN THE Z NOTATION FROM C PROGRAMS) อ.ที่ปรึกษา : ผศ.วิวัฒน์ วัฒนาวุฒิ, 142 หน้า. ISBN 974-346-883-8.

วิทยานิพนธ์นี้ได้กำหนดกฎและออกแบบขั้นตอนวิธีในการสร้างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจากโปรแกรมภาษาซี กฎที่ได้แบ่งเป็น 2 ส่วนคือกฎสำหรับการแปลงส่วนการประกาศ และกฎสำหรับการแปลงส่วนการทำงาน เนื่องจากพบว่ามีความคล้ายคลึงระหว่างโครงสร้างของโปรแกรมภาษาซีและข้อกำหนดเซด

งานวิจัยนี้ครอบคลุมถึงการแปลงชนิดข้อมูลพื้นฐานในภาษาโปรแกรมซี 7 ชนิด และข้อความสั่งหลัก 5 ประเภทในภาษาโปรแกรมซีได้แก่ ข้อความสั่งกำหนดค่า ข้อความสั่งเลือกทางเดิน ข้อความสั่งตามลำดับ ข้อความสั่งวนซ้ำ และข้อความสั่งเรียกใช้ฟังก์ชัน

นอกจากนั้นงานวิจัยนี้ยังได้นำเอาขั้นตอนวิธีที่ได้มาทำการพัฒนาเป็นเครื่องมือในการแปลงโปรแกรมภาษาซีเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ซึ่งโปรแกรมที่พัฒนาได้รับการทดสอบโดยใช้กรณีทดสอบ 9 กรณีและผลลัพธ์ได้รับการตรวจสอบในส่วนวากยสัมพันธ์และการพิสูจน์ทางคณิตศาสตร์จากโปรแกรม Z/EVES

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต เรืองยศ วรเจนวณิชย์

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา วิวัฒน์ วัฒนาวุฒิ

ปีการศึกษา 2543

##4170485421 : MAJOR COMPUTER SCIENCE

KEYWORD : FORMAL SPECIFICATION / Z NOTATION / REVERSE ENGINEERING

REUNGYOS VORAJENWANICH : A CONSTRUCTION OF FORMAL SPECIFICATION IN THE Z NOTATION FROM C PROGRAMS. THESIS ADVISOR : ASSIST. PROF. WIWAT VATANAWOOD, 142 pp. ISBN 974-346-883-8.

This thesis defines rules and designs algorithms for constructing formal specification in Z notation from C programs. Because of the similarity between C program structure and Z specification structure, the rules are categorized into two parts - the declaration-part translating rules and operation-part translating rules.

Seven primitive data types in C and five kinds of statement in C - assignment statement, alternative statement, sequence statement, iterative statement and function call statement, are covered in our approach.

Moreover, a tool for translating C programs to formal specification in Z notation is developed by using our algorithm. The software tool is tested to satisfy 9 test cases and the results are syntactically checked and theoretically proved using the program Z/EVES.

Department Computer Engineering ..... Student's signature .....

Fields of study Computer Science ..... Advisor's signature .....

Academic year 2000 .....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของ ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ อาจารย์ที่ปรึกษา ซึ่งได้ให้คำปรึกษาและคำแนะนำตลอดระยะเวลาการทำวิจัย และขอขอบคุณ อาจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์ อาจารย์ ดร. พรศิริ หมื่นไชยศรี รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์ และ อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา กรรมการวิทยานิพนธ์ที่กรุณาเสียสละเวลาให้คำแนะนำ ตรวจและแก้ไขต้นฉบับวิทยานิพนธ์

ขอขอบคุณ เพื่อน ๆ ทุกคน โดยเฉพาะเพื่อนในกลุ่ม Formal Method และสมาชิกห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ที่เสียสละเวลาในการให้คำปรึกษา และช่วยตรวจสอบผลการวิจัยที่ได้ ขอขอบคุณทุก ๆ ท่านที่ให้กำลังใจและข้อเสนอแนะต่าง ๆ ณ โอกาสนี้ด้วย

สุดท้ายนี้ ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา มารดา และพี่ ๆ ทุกท่านที่ให้การสนับสนุนในด้านต่าง ๆ และให้กำลังใจแก่ผู้วิจัยเสมอมา

เรืองยศ วรเจนวณิชย์

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในการพัฒนาซอฟต์แวร์คอมพิวเตอร์นั้นจะต้องทำการเขียนข้อกำหนดต่าง ๆ ไม่ว่าจะเป็น ข้อกำหนดของระบบ (System Specification) หรือ ข้อกำหนดของโปรแกรม (Program Specification) เพื่อทำการอธิบายพฤติกรรมของระบบซึ่งวิธีการเขียนที่ได้รับความนิยมในปัจจุบันจะเป็น วิธีอรูปนัย (Informal Method) และ วิธีกึ่งรูปนัย (Semi-Formal Method) ซึ่งจะใช้ ภาษาธรรมชาติ (Natural Language) หรือ แผนภาพ (Diagram) ในการอธิบายพฤติกรรมของระบบซึ่งทั้งสองวิธีประสบกับปัญหาในการตีความและทำความเข้าใจระหว่างกลุ่มผู้พัฒนา เนื่องจากตัวภาษาธรรมชาติเองมีความกำกวมดังนั้นจึงได้มีการเสนอ วิธีรูปนัย (Formal Method) ขึ้นมา โดยวิธีรูปนัยนี้พยายามที่จะลดความกำกวมของภาษาธรรมชาติลงโดยการนำเอา สัญกรณ์ (Notation) และทฤษฎีทางคณิตศาสตร์มาช่วยในการเขียนข้อกำหนดให้ชัดเจนและเป็นที่เข้าใจแบบเดียวกันโดยจะเรียกข้อกำหนดที่ได้ว่า ข้อกำหนดรูปนัย (Formal Specification) เนื่องจากทฤษฎีและสัญกรณ์ต่าง ๆ ทางคณิตศาสตร์มีการกำหนดความหมายไว้อย่างชัดเจนแล้ว และการที่ทำการเขียนข้อกำหนดของระบบด้วยวิธีรูปนัยยังสามารถที่จะทำการพิสูจน์ได้ว่าข้อกำหนดที่ได้ทำการเขียนขึ้นนั้นมีความถูกต้องมากน้อยเพียงใดด้วยการพิสูจน์ทางคณิตศาสตร์ และท้ายที่สุดของการนำเอาวิธีรูปนัยเข้ามาใช้งานก็คือสามารถที่จะนำเอาวิธีรูปนัยไปใช้ใน กระบวนการพัฒนาซอฟต์แวร์ (Software Development Process) เพื่อทำให้เป็นระบบอัตโนมัติ

ข้อกำหนดที่ได้ทำการเขียนขึ้นนั้นจะนำมาใช้เป็นเอกสารประกอบซอฟต์แวร์ซึ่งจะถูกนำไปใช้ในขั้นตอนการบำรุงรักษาซอฟต์แวร์ (Software Maintenence) ซึ่งเป็นขั้นตอนที่ทำการแก้ไขซอฟต์แวร์ระหว่างการใช้งานให้เหมาะสมกับงานปัจจุบันโดยในซอฟต์แวร์ที่มีอายุการใช้งานที่ยาวนานนั้นมักพบว่าเอกสารประกอบซอฟต์แวร์กับซอฟต์แวร์นั้นไม่สอดคล้องกัน เนื่องจากเมื่อมีการแก้ไขซอฟต์แวร์มักจะไม่มีการแก้ไขตัวเอกสารตามไปด้วย เป็นผลให้การบำรุงรักษาซอฟต์แวร์กระทำได้ค่อนข้างยากดังนั้นเพื่อให้สามารถทำการบำรุงรักษาต่อไปได้จึงต้องทำการสร้างเอกสารประกอบขึ้นมาใหม่จากซอฟต์แวร์ที่มีอยู่โดยการวิเคราะห์จากตัวโปรแกรม ดังนั้นงานวิจัยในครั้งนี้ได้ทำการกำหนดกฎและออกแบบขั้นตอนวิธี (Algorithm) ในการสร้างข้อกำหนดของซอฟต์แวร์จากโปรแกรมที่มีอยู่แล้วโดยแสดงข้อกำหนดอยู่ในรูปของข้อกำหนดรูปนัยและเลือกเอาภาษาซีเป็นกรณีศึกษา

1.2 วัตถุประสงค์

1.2.1) เพื่อกำหนดกฎและออกแบบขั้นตอนวิธีในการสร้างข้อกำหนดรูปนัยจากโปรแกรมต้นฉบับภาษาซีโดยใช้สัญกรณ์เซด

1.2.2) เพื่อพัฒนาเครื่องมือซอฟต์แวร์ในการสร้างข้อกำหนดรูปนัยจากโปรแกรมต้นฉบับภาษาซีโดยใช้สัญกรณ์เซด

1.3 ขอบเขตงานวิจัย

1.3.1) โปรแกรมต้นฉบับภาษาซีที่เป็นข้อมูลนำเข้าจะยึดตามมาตรฐาน ANSI C ซึ่งจะต้องถูกต้องตาม วากยสัมพันธ์ (Syntax) ของภาษา และจะต้องรวมการประกาศทุกอย่างไว้ภายในไฟล์เดียวกันเท่านั้น

1.3.2) โปรแกรมต้นฉบับภาษาซีจะต้องไม่มีการใช้คำสั่ง goto, break, continue ยกเว้นกรณีการใช้คำสั่ง break กับข้อความสั่ง switch

1.3.3) โปรแกรมต้นฉบับภาษาซีจะต้องอยู่ในรูปของคำสั่งเดี่ยว (Single Statement) เท่านั้นกล่าวคือจะต้องไม่มีข้อคำสั่งใดที่เกิดจากการนำเอาข้อความสั่งมากกว่า 1 ข้อความสั่งมารวมกันหรือหากข้อความสั่งสามารถแตกออกเป็นข้อความสั่งมากกว่า 1 ข้อความสั่งยกตัวอย่างเช่น

```
while ( (c = getch()) != 'C')
  { ..... }
```

จะเห็นว่าเงื่อนไขของลูปเกิดจากการนำเอาข้อความสั่ง c = getch() มารวมกับข้อความสั่ง c != 'C' เพราะฉะนั้นข้อความสั่งประเภทนี้จะไม่รวมอยู่ในงานวิจัยนี้

1.3.4) งานวิจัยจะครอบคลุมในส่วนของข้อมูลชนิดพื้นฐานได้แก่ int, float, double, short, long, unsigned, char และข้อความสั่งทั้ง 5 ประเภทได้แก่

- ข้อความสั่งกำหนดค่า
- ข้อความสั่งเลือกทางเดิน
- ข้อความสั่งตามลำดับ
- ข้อความสั่งวนซ้ำ
- ข้อความสั่งเรียกใช้ฟังก์ชัน

1.3.5) การแสดงข้อกำหนดรูปนัยของสัญกรณ์เซตจะอยู่ในรูปแบบของเอกสาร ลาเท็กซ์ (Latex)

1.3.6) งานวิจัยนี้จะทำการตรวจสอบผลลัพธ์ที่ได้โดย

- ตรวจสอบวากยสัมพันธ์ของสัญกรณ์เซต (Syntax Checking)
- ตรวจสอบโดยอาศัยการพิสูจน์ทางคณิตศาสตร์ (Theorem Proving)
- ตรวจสอบโดยอาศัยตัวอย่าง (Checking by example) ซึ่งตัวอย่างในการตรวจสอบจะครอบคลุมทุกกฎที่ทำการสร้างขึ้น

1.3.7) ซอฟต์แวร์ที่ทำการพัฒนาขึ้นจะทำงานบนระบบปฏิบัติการวินโดว์ส 95 ขึ้นไป

1.4 ขั้นตอนการดำเนินงาน

1.4.1) ศึกษางานวิจัยที่เกี่ยวข้อง

1.4.2) ศึกษาวิธีรูปนัยและสัญกรณ์เซต

1.4.3) ศึกษาโครงสร้างและวากยสัมพันธ์ของภาษาโปรแกรมซี

1.4.4) ออกแบบขั้นตอนวิธีในการแปลง

1.4.5) พัฒนาเครื่องมือซอฟต์แวร์

1.4.6) ตรวจสอบผลการวิจัย

1.4.7) สรุปผลการวิจัยและจัดทำเอกสาร

1.5 ประโยชน์ที่จะได้รับ

1.5.1) เป็นทางเลือกในการแปลงโปรแกรมต้นฉบับภาษาซีให้เป็นข้อกำหนดรูปนัย

1.5.2) แสดงให้เห็นถึงความสัมพันธ์ระหว่างสัญกรณ์เซตและภาษาโปรแกรมซี

1.5.3) เป็นเครื่องมือในการสร้างข้อกำหนดรูปนัยจากโปรแกรมภาษาซีที่มีอยู่แล้ว

บทที่ 2
งานวิจัยและทฤษฎีที่เกี่ยวข้อง

2.1 เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1.1) The Application of Formal Methods to the Reverse Engineering of Imperative Program Code ([1],[2],[3],[4],[5],[6]) โดย Gerald C. Gannod และ Betty H.C. Cheng

งานวิจัยนี้ได้ทำการออกแบบแนวคิดในการนำเอาวิธีรูปนัยมาใช้ในการทำวิศวกรรมย้อนกลับโปรแกรมเชิงคำสั่ง (Imperative Program) โดยใช้แนวคิดทางด้าน ความหมายรูปนัย (Formal Semantic) ของโปรแกรมเชิงคำสั่งซึ่งจะกล่าวถึง ตัวแปลงภาคแสดง (Predicate Tranformer) เรียกว่า เงื่อนไขก่อนการทำงานแบบอ่อนที่สุด (Weakest Pre-condition) [7],[8],[9] มาเป็นแนวทางในการสร้างตัวแปลงภาคแสดงเงื่อนไขหลังการทำงานแบบแข็งที่สุด (Strongest Post-condition) [4] เพื่อใช้เป็นกลไกในการแปลงข้อความสั่งในโปรแกรมเชิงคำสั่งให้อยู่ในรูป ภาคแสดง (Predicate) แล้วนำเอาภาคแสดงที่ได้มาทำการเชื่อมต่อกันด้วย ตัวดำเนินการ (Operator) ทางตรรกศาสตร์

รูปที่ 2.1 แสดงตัวอย่างผลลัพธ์ที่ได้จากงานวิจัยนี้โดยบรรทัดที่เป็นตัวอักษรเอียงจะแสดงภาคแสดงของแต่ละคำสั่งซึ่งจะเห็นว่าข้อกำหนดรูปนัยที่ได้จะอยู่ในรูปของภาคแสดงที่ยาวต่อเนื่องกัน ถึงแม้ว่าโปรแกรมจะแบ่งออกเป็นฟังก์ชันข้อกำหนดที่ได้ก็ยังคงไม่มีการแบ่งเป็นส่วน ๆ ทำให้มีความซับซ้อนในการอ่านและทำความเข้าใจโดยเฉพาะเมื่อข้อมูลนำเข้ามีขนาดใหญ่จะยิ่งทำให้ข้อกำหนดรูปนัยที่ได้มีความซับซ้อนมากยิ่งขึ้น

2.1.2) Design Recovery through Formal Specification [10] โดย Wie Ming Lim, John V. Harrison, Paul A. Bailes และ Anthony Berglas

งานวิจัยนี้มีแนวคิดมาจากความพยายามที่จะทำการแปลงระบบที่ถูกออกแบบบนฐานข้อมูล Ingres มาใช้เป็นฐานข้อมูล Oracle โดยเลือกเอาวิธีรูปนัยมาเป็นแนวทางในการแปลงซึ่งงานวิจัยนี้ได้ทำการสร้างภาษารูปนัยหนึ่งเรียกว่า ภาษาซีเอ็มแอล (Concept Mapping Language) ซึ่งพัฒนามาจากทฤษฎีตรรกศาสตร์ภาคแสดงลำดับแรก (First Order Predicate Logic) และเทคนิคทางด้าน เชิงวัตถุ (Object Oriented) เพื่อสะดวกในการที่จะเป็นภาษากลางในการแปลงระหว่างสองระบบ

ภาษาซีเอ็มแอลจะประกอบด้วยสองส่วนคือส่วนหนึ่งเรียกว่า นิยามคลาส (Class Definition) จะใช้ในการแปลงโครงสร้างข้อมูลและอีกส่วนเรียกว่า นิยามฟังก์ชัน (Function Definition) ใช้ในการแปลงส่วนของการทำงาน ซึ่งจะเห็นว่าภาษานี้ถูกออกแบบมาเฉพาะเพื่อใช้เป็นตัวกลางในการแปลงระบบที่เกี่ยวข้องกับฐานข้อมูลทำให้การนำมาใช้กับงานในลักษณะอื่นนั้นไม่ค่อยเหมาะสมนัก

```
1. if (a = b) then
        2. if (c = d) then
          3. begin
                4. x := p;
                4.1 (* (x{3}1 = p0) & U *)
                5. x := q;
                5.1 (* (x{3}2 = q0) & U *)
                6. end
                6.1 (* ((x{3}1 = p0) & (x{3}2 = q0)) & U *)
          7. end
        8. else
          9. begin
                9. x := r;
                9.1 (* (x{3}1 = r0) & U *)
                10. x := (x + q);
                10.1 (* (x{3}2 = r0 + q0) & U *)
                11. x := (x - p);
                11.1 (* (x{3}3 = ((r0 + q0) - p0)) & U *)
          12. end
        12.1 (* ((x{3}1 = r0) & (x{3}2 = r0 + q0) & (x{3}3 = r0 + q0 - p0)) & U *)
        12.2 (* (((c0 = d0) & ((x{3}1 = p0) & (x{1}1 = q0))) |
        12.3 ( not(c0 = d0) & ((x{3}1 = r0) & (x{3}2 = r0 + q0)
        12.4 & (x{1}1 = r0 + q0 - p0)))) & U *)
        13. else
        14. x := s;
        14.1 (* (x{1}1 = s0) & U *)
15. (* (((a0 = b0) & (((c0 = d0) & ((x{3}1 = p0) & (x{0}1 = q0))) |
15.1 (not(c0 = d0) & ((x{3}1 = r0) & (x{3}2 = r0 + q0) &
15.2 (x{0}1 = (r0 + q0 - p0)))))) |
15.3 ((not(a0 = b0)) & (x{0}1 = s0))) & U *)
```

รูปที่ 2.1 ผลลัพธ์ที่ได้จากงานวิจัย [1] โดยเป็นการแปลงโปรแกรมภาษาปาสคาล

2.2 ทฤษฎีที่เกี่ยวข้อง

2.2.1) เทคนิควิศวกรรมย้อนกลับซอฟต์แวร์ (Software Reverse Engineering Technique) [11]

เทคนิคการทำวิศวกรรมย้อนกลับเป็นขั้นตอนหนึ่งของการทำ วิศวกรรมซ้ำซอฟต์แวร์ (Software Re-Engineering) เพื่อช่วยให้ผู้ดูแลระบบสามารถทำความเข้าใจซอฟต์แวร์ที่ไม่ได้มีการทำเอกสารไว้หรือถูกพัฒนามาเป็นเวลานานโดยการวิเคราะห์ โปรแกรมต้นฉบับ (Source Code) เพื่อสร้าง การแทน (Representation) ที่มี ระดับภาวะนามธรรม (Abstraction Level) มากกว่าโปรแกรมต้นฉบับ

โดยทั่วไปสาเหตุส่วนใหญ่ของการทำวิศวกรรมย้อนกลับนั้นเนื่องมาจากไม่ได้มีการทำเอกสารประกอบซอฟต์แวร์หรือในซอฟต์แวร์ที่ได้ทำการพัฒนามายาวนานกว่า 10 ถึง 15 ปีนั้นจะพบว่าระหว่างช่วงเวลาดังกล่าว จะมีการเปลี่ยนแปลงแก้ไขไปจากระบบเดิมที่ได้ทำการออกแบบมา ซึ่งในการแก้ไขบางครั้งไม่ได้มีการตามไปปรับปรุงเอกสารประกอบระบบด้วยทำให้เมื่อเวลาผ่านไปเอกสารกับระบบนั้นไม่ตรงกันทำให้การดูแลรักษาระบบนั้นกระทำได้ยากหรืออาจทำไม่ได้เลย โดยเราสามารถที่จะอาศัยเทคนิควิศวกรรมย้อนกลับซอฟต์แวร์เพื่อสร้างเอกสารประกอบระบบใหม่ (Redocumentation) เนื่องจากเอกสารเดิมไม่สามารถจะนำมาใช้ในการบำรุงรักษาระบบได้หรือเพื่อทำ การเรียกคืนการออกแบบ (Design Recovery)

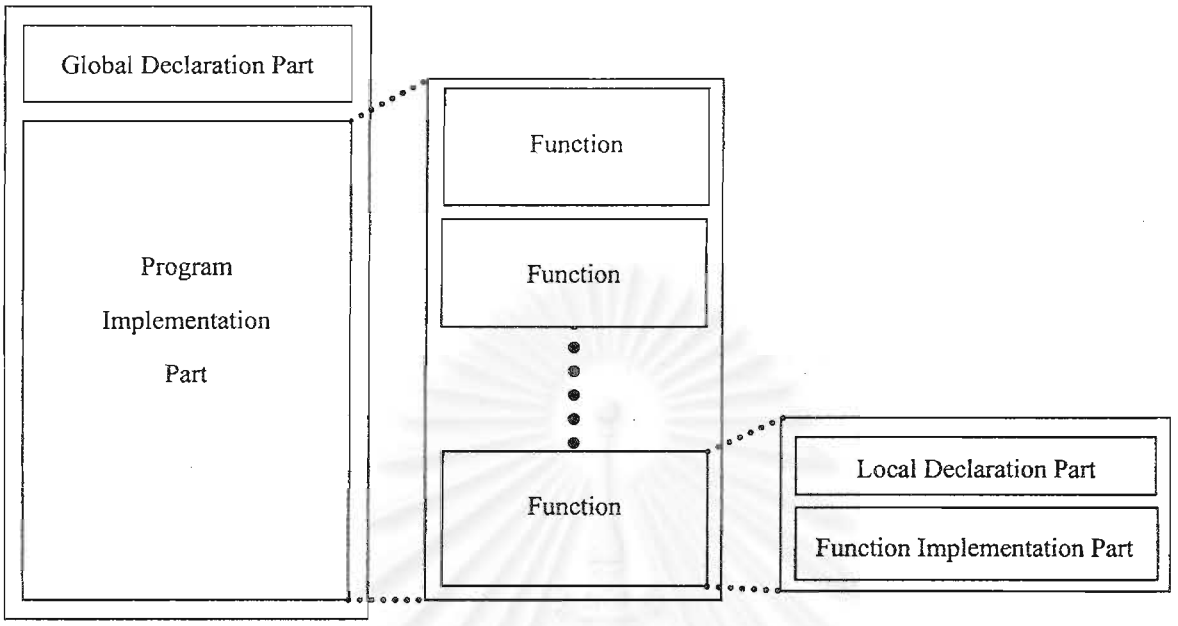
2.2.2) ภาษาโปรแกรมซี (C Programming Language)

ภาษาโปรแกรมซีเป็นภาษาโปรแกรมในกลุ่มของภาษาโปรแกรมเชิงคำสั่งซึ่งในโปรแกรมหนึ่ง ๆ จะประกอบไปด้วย ฟังก์ชัน (Function) และ ตัวแปร (Variable) ในแต่ละฟังก์ชันจะประกอบไปด้วย ข้อความสั่ง (Statement) ซึ่งคอมพิวเตอร์ใช้ในการปฏิบัติตามและตัวแปรใช้ในการเก็บค่าระหว่างการคำนวณ โดยแต่ละฟังก์ชันสามารถที่จะตั้งชื่ออะไรก็ได้ตามกฎการตั้งชื่อแต่จะมีฟังก์ชันชื่อ main ซึ่งมีความหมายพิเศษโปรแกรมจะเริ่มทำงานที่ฟังก์ชัน main เสมอนั่นหมายความว่าโปรแกรมที่สามารถดำเนินการได้จะต้องมีฟังก์ชัน main อยู่ด้วยและฟังก์ชัน main จะเรียกฟังก์ชันอื่นเพื่อทำงานต่อไป

ในงานวิจัยนี้จะพิจารณาโครงสร้างภาษาซีดังแสดงในรูปที่ 2.2 โดยในโปรแกรมหนึ่ง ๆ จะประกอบด้วย 2 ส่วนคือ ส่วนการประกาศส่วนกลาง (Global Declaration Part) และ ส่วนการทำงานของโปรแกรม (Program Implemantation Part) ซึ่งในส่วนการประกาศส่วนกลางจะประกอบด้วยการประกาศตัวแปร หรือ ต้นแบบฟังก์ชัน (Function Prototype) ต่าง ๆ เป็นต้น ส่วนการทำงานจะแบ่งออกเป็นฟังก์ชันย่อย ๆ โดยแต่ละฟังก์ชันจะประกอบด้วยส่วนการประกาศท้องถิ่น (Local Declaration Part) และส่วนการทำงานของฟังก์ชัน (Function Implementation Part) ซึ่งจะประกอบด้วยข้อความสั่งของภาษาโปรแกรมซี โดยสามารถจัดกลุ่มข้อความสั่งเหล่านั้นออกเป็นข้อความสั่งหลัก ๆ ด้วยกัน 5 ประเภท [7],[9]คือ

1) ข้อความสั่งกำหนดค่า (Assignment Statement)
2) ข้อความสั่งเลือกทางเดิน (Alternative Statement)
3) ข้อความสั่งวนซ้ำ (Iterative Statement)
4) ข้อความสั่งเรียงลำดับ (Sequence Statement)

5) ข้อความสั่งเรียกใช้ฟังก์ชัน (Function Call Statement)



รูปที่ 2.2 แสดงโครงสร้างโปรแกรมภาษาซี

2.2.2.1) กฎเกณฑ์ขอบเขต (Scope Rule)

ในการเขียนโปรแกรมภาษาซีนั้นเมื่อมีการประกาศตัวแปรขึ้นเมื่อใช้งานนั้นตัวแปรจะถูกจำกัดขอบเขตของการใช้งานซึ่งขอบเขตของตัวแปรใด ๆ ก็คือส่วนของโปรแกรมที่สามารถอ้างอิงตัวแปรนั้นได้ สำหรับตัวแปรที่ได้ประกาศไว้ตอนต้นของฟังก์ชัน ขอบเขตของตัวแปรเหล่านั้นก็คือภายในฟังก์ชันที่ประกาศมันไว้ ดังนั้นตัวแปรท้องถิ่น (Local Variable) ที่มีชื่อเหมือนกันในคนละฟังก์ชันจึงไม่มีความเกี่ยวข้องกัน หลักการนี้เป็นจริงสำหรับพารามิเตอร์ของฟังก์ชันด้วย (พารามิเตอร์จึงทำหน้าที่เป็นตัวแปรท้องถิ่น)

ส่วนขอบเขตของตัวแปรภายนอกหรือฟังก์ชันเริ่มต้น ณ จุดที่ประกาศจนสิ้นสุดแฟ้มต้นฉบับที่มีการประกาศนั้น

2.2.2.2) โครงสร้างบล็อก (Block-Structure)

ภาษาโปรแกรมซีมิใช่ภาษาซึ่งมีโครงสร้างบล็อกเช่นเดียวกับภาษาโปรแกรมป่าสคาล เพราะไม่สามารถกำหนดฟังก์ชันภายในฟังก์ชันได้ อย่างไรก็ตาม การประกาศตัวแปรในภาษาโปรแกรมซีสามารถกระทำแบบโครงสร้างบล็อกได้ภายในฟังก์ชัน การประกาศตัวแปรอาจกระทำได้ภายในเครื่องหมายวงเล็บปีกกาซึ่งไม่จำเป็นต้องมีอยู่เฉพาะตอนเริ่มต้นฟังก์ชัน ตัวแปรซึ่งประกาศแบบนี้จะเป็นตัวแปรคนละตัวกับตัวแปรที่มีชื่อเหมือนกันในบล็อกนอกออกมา ขอบเขตตัวแปรจะอยู่เฉพาะภายในบล็อกที่ประกาศตัวแปรนั้น ดังแสดงในรูปที่ 2.3

```
if (n > 0) {
        int i;      /*การประกาศตัวแปรใหม่ */

        for (i = 0; i  < n; I++)
                .....
}
```

รูปที่ 2.3 แสดงขอบเขตของตัวแปร

จากรูปที่ 2.3 ขอบเขตของ i นั้นอยู่เฉพาะในกรณีที่ if เป็นจริงเท่านั้น ดังนั้น i จึงไม่เกี่ยวข้องกับ i อื่น ๆ ในบล็อกนอกการประกาศ

ตัวแปรท้องถิ่นและพารามิเตอร์จะบดบังตัวแปรภายนอกและฟังก์ชันซึ่งมีชื่อเหมือนกันดังแสดงในรูป 2.4

```
int x;
int y;

f(double  x)
{
        double y;
        .....
}
```

รูปที่ 2.4 แสดงการบดบังของตัวแปรที่มีชื่อเหมือนกัน

จากรูปที่ 2.4 การอ้างอิง x ภายในฟังก์ชัน f จึงหมายถึงพารามิเตอร์ซึ่งเป็น double ส่วนการอ้างอิง x ภายนอกฟังก์ชัน f จะหมายถึงค่า int การทำงานของ y ก็เช่นเดียวกัน

2.3) วิธีรูปนัย (Formal Method)

วิธีรูปนัยเป็นวิธีการที่พัฒนาขึ้นโดยการนำเอาสัญกรณ์และทฤษฎีทางคณิตศาสตร์ที่ได้มีการกำหนดความหมายไว้อย่างดีแล้วมาเป็นเครื่องมือในการอธิบายถึงพฤติกรรมของระบบเพื่อไม่ให้เกิดความกำกวมเหมือนกับที่เกิดขึ้นจากการใช้ภาษาธรรมชาติและเรายังสามารถที่จะตรวจสอบกระบวนการพัฒนาซอฟต์แวร์ได้ตั้งแต่ขั้นตอนของการออกแบบโดยอาศัยการพิสูจน์ทางคณิตศาสตร์ในขณะที่วิธีอรูปนัยและวิธีกึ่งรูปนัยจะสามารถทำได้ก็ต่อเมื่อได้ทำการพัฒนาซอฟต์แวร์ออกมาเป็นที่เรียบร้อยแล้ว

วิธีรูปนัยสามารถแบ่งออกเป็น 2 กลุ่มใหญ่ ๆ [12] ด้วยกันคือ แบบโมเดลเบส (Model Based) และ แบบพร็อบเพอร์ตี้เบส (Property Based) ซึ่งถึงแม้ว่าทั้งสองแบบจะมีพื้นฐานมาจาก ภินทนคณิตศาสตร์ (Discrete Mathematics) เหมือนกัน แต่จุดที่แตกต่างของทั้งสองแบบคือรูปแบบและวิธีการในการ

เขียนข้อกำหนดและแต่ละแบบก็เหมาะสมในการที่จะนำไปใช้ในระบบคอมพิวเตอร์ที่แตกต่างกัน รูปที่ 2.5 จะแสดงให้เห็นถึงการจัดกลุ่มของการเขียนข้อกำหนดตามรูปแบบการเขียนข้อกำหนด

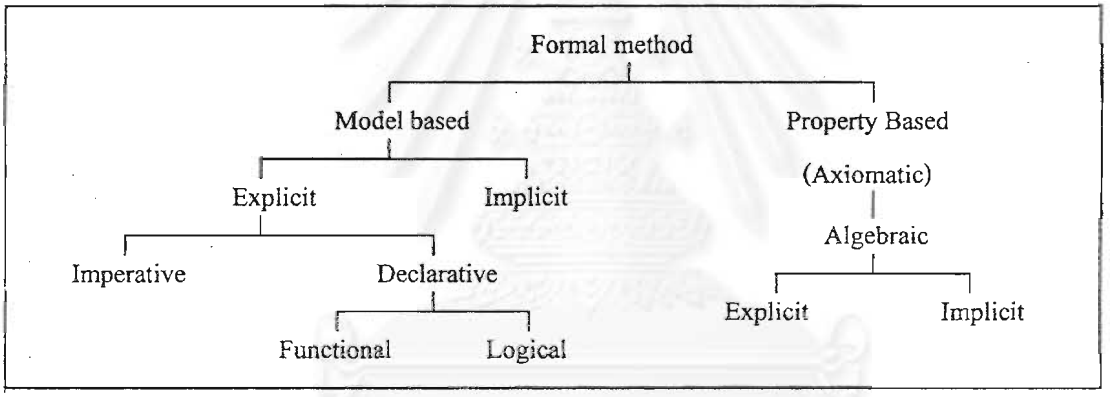
2.3.1) ข้อกำหนดแบบโมเดลเบส (Model Based Specification) เป็นการเขียนในลักษณะที่จะอธิบายรายละเอียดของการทำงานไว้ในข้อกำหนดด้วยซึ่งในบางครั้งจะถูกมองว่าเป็นการลงในรายละเอียดมากเกินไปแต่อย่างไรก็ตามวิธีการเขียนแบบนี้ถือได้ว่าเหมาะที่จะใช้ในการเขียนข้อกำหนดเมื่อเปรียบเทียบกับวิธีอื่น

2.3.1.1) ข้อกำหนดโดยปริยาย (Implicit Specification)

ข้อกำหนดโดยปริยายนั้นจะบอกรายละเอียดของการทำงานเพียงแค่ว่าต้องการที่จะทำอะไรแต่จะไม่บอกว่าจะทำได้อย่างไรโดยแสดงในรูปของ เงื่อนไขก่อนและหลังการทำงาน (Pre- , Post - condition)

2.3.1.2) ข้อกำหนดชัดแจ้ง (Explicit Specification)

ข้อกำหนดชัดแจ้งนอกจากที่จะบอกว่าต้องการจะทำอะไรแล้วยังมีการบอกด้วยว่าจะทำได้อย่างไรโดยอาศัย การทำให้ละเอียดขึ้น (Refinement) ซึ่งข้อกำหนดแบบนี้ยังถูกแบ่งออกเป็นกลุ่มย่อยอีกคือ



รูปที่ 2.5 การจัดกลุ่มของวิธีรูปนัยตามรูปแบบการเขียนข้อกำหนด [12]

1) ข้อกำหนดเชิงคำสั่ง (Imperative or State-Based Specification)

ข้อกำหนดเชิงคำสั่งจะเป็นข้อกำหนดที่แสดงในรูปของการเปลี่ยนแปลงค่าของตัวแปรเป็นหลักซึ่งลำดับภายในข้อกำหนดเชิงคำสั่งนั้นมีความสำคัญเนื่องจากการเปลี่ยนลำดับของข้อกำหนดจะก่อให้เกิดผลลัพธ์ที่แตกต่างกัน

2) ข้อกำหนดเชิงประกาศ (Declarative Specification)

ข้อกำหนดแบบเชิงประกาศจะเป็นข้อกำหนดที่จะแสดงในรูปของกลุ่มของสมการหรือความสัมพันธ์ของข้อมูลเป็นหลักโดยถูกแบ่งออกเป็น

- ข้อกำหนดเชิงฟังก์ชัน (Functional Specification) ข้อกำหนดที่ได้จะแสดงในรูปของฟังก์ชันไม่มีการใช้ตัวแปรในลักษณะของ ตัวแปรส่วนกลาง (Global Variable) หรือ ตัวแปรภายนอก (External Variable) การทำงานจะใช้เพียงการส่งผ่านค่าไปทาง พารามิเตอร์ (Parameter) ของฟังก์ชันเท่านั้น

- ข้อกำหนดเชิงตรรกะ (Logical Specification) ข้อกำหนดที่ได้จะแสดงอยู่ในรูปของนิพจน์ที่สามารถหาค่าความเป็นจริงได้โดยทั่วไปจะใช้ทฤษฎีตรรกศาสตร์ภาคแสดงในการเขียน

2.3.2) ข้อกำหนดแบบพร็อบเพอร์ตี้เบส (Property Based or Axiomatic Specification)

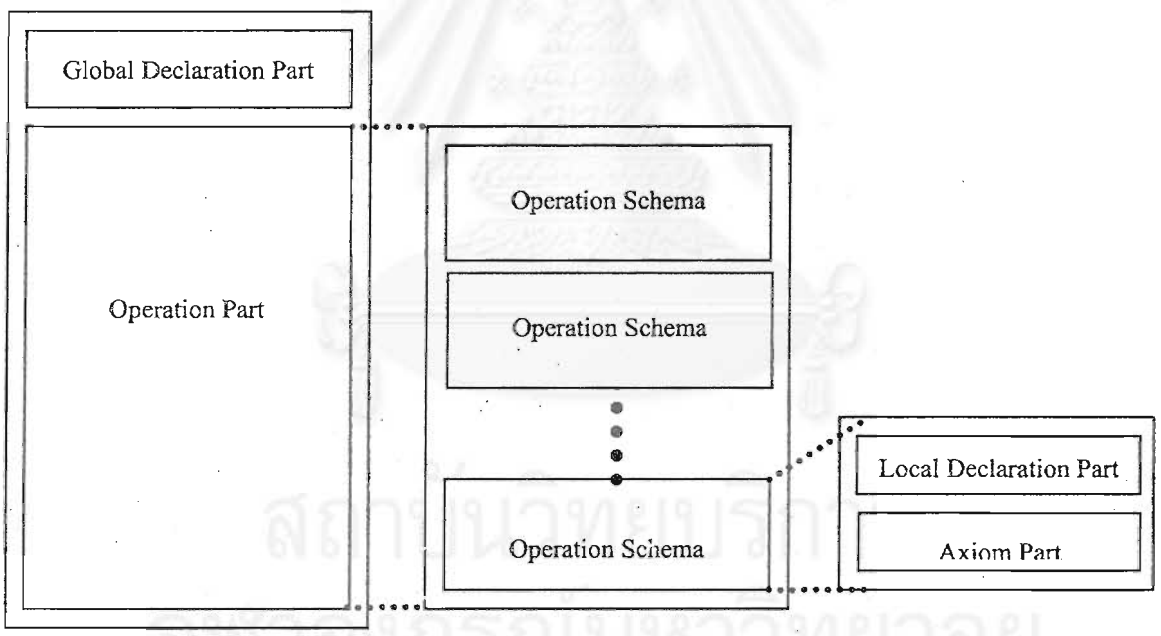
ข้อกำหนดแบบพร็อบเพอร์ตี้เบสจะทำการอธิบายระบบโดยการบอกถึง คุณสมบัติ (Property) ของระบบที่จะต้องรักษาไว้ตลอดการทำงานโดยใช้ทฤษฎีตรรกศาสตร์ภาคแสดงลำดับแรกบนพื้นฐานของ ตรรกศาสตร์ของฮอร์(Hoare Logic) ในการบอกถึงเงื่อนไขก่อนและหลังการทำงานในรูปของสัจพจน์

2.3.2.1) ข้อกำหนดพีชคณิต (Algebraic Specification)

ในกรณีที่สัจพจน์ถูกเขียนในรูปของสมการจะเรียกวิธีการแบบนี้ว่าพีชคณิตซึ่งการอธิบายการทำงานหนึ่ง ๆ นั้นจะทำโดยการอ้างถึงการทำงานอื่น

2.4) สัญกรณ์เซด (Z Notation)

สัญกรณ์เซดเป็น ภาษารูปนัย (Formal Language) ภาษาหนึ่งจัดอยู่ในกลุ่มของโมเดลเบสประเภทเชิงคำสั่งทำให้ข้อกำหนดที่ได้จากสัญกรณ์เซดมีโครงสร้างที่คล้ายคลึงกับภาษาโปรแกรมเชิงคำสั่งดังแสดงในรูปที่ 2.6

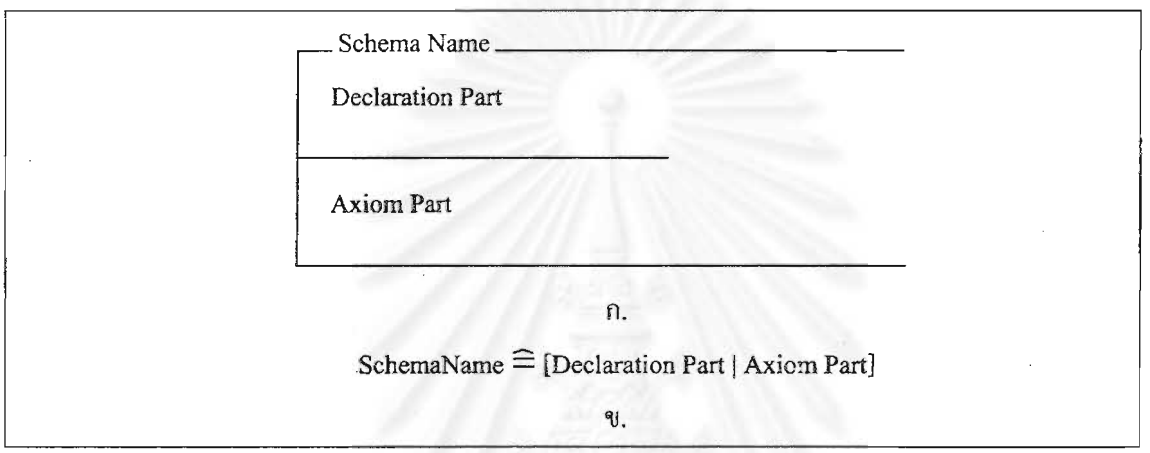


รูปที่ 2.6 แสดงโครงสร้างของข้อกำหนดรูปนัยที่เขียนด้วยสัญกรณ์เซด

จากรูปที่ 2.6 จะเห็นว่าข้อกำหนดรูปนัยจะประกอบไปด้วย 2 ส่วน ส่วนแรกคือส่วนการประกาศ (Declaration Part) ได้แก่ การประกาศส่วนกลาง (Global Declaration Part) และการประกาศส่วนท้องถิ่น (Local Declaration Part) ส่วนที่สองคือส่วนการทำงาน (Operation Part) โดยแต่ละส่วนจะถูกเขียนอยู่ในรูปของ เค้าร่างแบบกล่อง (Box Like Schema) ดังแสดงในรูปที่ 2.7ก หรือ เค้าร่างตามขวาง (Horizontal Schema) ดังแสดงในรูปที่ 2.7ข และสามารถทำการเชื่อมต่อแต่ละเค้าร่างเข้าด้วยกันโดยอาศัย แคลคูลัสเค้าร่าง (Schema Calculus) [13]

2.4.1) ส่วนของการประกาศ (Declaration Part) จะเป็นส่วนที่ใช้ในการกำหนดข้อมูลที่มีการใช้งานร่วมกันภายในข้อกำหนดซึ่งสามารถเขียนได้อยู่ในรูปของ

- เค้าร่างสถานะ (State Schema)
- เค้าร่างทั่วไป (Generic Schema)
- นิยามสัจพจน์ (Axiom Definition)
- ชนิดอิสระ (Free Type)
- กีเวนเซต (Given Set)



รูปที่ 2.7 แสดงสัญลักษณ์เค้าร่างในสัญกรณ์เซด [14] ก.เค้าร่างแบบกล่อง ข.เค้าร่างตามขวาง

2.4.2) ส่วนของการทำงาน (Operation Part) เป็นส่วนที่จะอธิบายถึงการทำงานต่าง ๆ ภายในข้อกำหนดซึ่งจะถูกเขียนอยู่ในรูปของ เค้าร่างการทำงาน (Operation Schema) ซึ่งภายในเค้าร่างการทำงานหนึ่ง ๆ ก็จะประกอบไปด้วยส่วนของการประกาศซึ่งจะเป็นการประกาศตัวแปรที่จะต้องใช้ภายในเค้าร่างนั้น ๆ และส่วนของสัจพจน์ซึ่งจะเป็นส่วนที่กำหนดเงื่อนไขการทำงานในรูปของเงื่อนไขก่อนและหลังการทำงานซึ่งในส่วนของสัจพจน์นั้นจะใช้ทฤษฎีและสัญกรณ์ทางคณิตศาสตร์ในการแสดงซึ่งทฤษฎีหลัก ๆ จะประกอบด้วย 4 ทฤษฎี [15]คือ

2.4.2.1) ทฤษฎีเซต (Set Theory) ในการกล่าวอ้างถึงกลุ่มของสิ่งต่าง ๆ ที่เกี่ยวข้องกันนั้นเราสามารถอาศัยการจัดกลุ่มสิ่งเหล่านั้นรวมไว้ภายในเซตเดียวกันเพื่อสะดวกในการที่จะกล่าวอ้างถึงต่อ ๆ ไปได้โดยภายในเซตหนึ่ง ๆ นั้นจะประกอบไปด้วยข้อมูลที่เกี่ยวข้องกันเท่านั้นซึ่งการกำหนดเซตขึ้นมานั้นสามารถกระทำได้ 3 แบบคือ

1) การกำหนดเซตโดยการแจกแจงสมาชิก
2) การกำหนดเซตโดยการระบุเงื่อนไขของสมาชิก
3) การกำหนดเซตโดยการระบุลักษณะของสมาชิก

2.4.2.2) ตรรกศาสตร์ (Logic) แบ่งออกเป็น 2 กลุ่มด้วยกันคือ ตรรกศาสตร์ประพจน์ (Propositional Logic) และ ตรรกศาสตร์ภาคแสดง (Predicate Logic)

ประพจน์ (Propositional) คือประโยคที่สามารถจะหาค่าความเป็นจริงได้ว่าเป็นจริงหรือเท็จโดยประพจน์หนึ่ง ๆ จะเป็นจริงหรือเท็จได้อย่างใดอย่างหนึ่งเท่านั้นและสามารถที่จะทำการเชื่อมประพจน์เหล่านี้ได้ด้วย ตัวเชื่อม (Connection) ดังต่อไปนี้

- และ (and: $\wedge$)
- หรือ (or: $\vee$)
- ถ้า ... แล้ว (implies: $\Rightarrow$)
- ก็ต่อเมื่อ (if and only if: $\Leftrightarrow$)

ตรรกศาสตร์ภาคแสดงเป็นการขยายขีดความสามารถทางด้านตรรกศาสตร์เนื่องจากในบางครั้งเราไม่สามารถที่จะใช้ประพจน์ในการหาค่าความจริงได้เช่นหากเราต้องการหาค่าความจริงของประโยค "เขาเป็นนักปราชญ์" นั้นด้วยคุณสมบัติของประพจน์ไม่สามารถที่จะทำได้จึงต้องอาศัยตรรกศาสตร์ภาคแสดงเข้ามาช่วยโดยการใช้ ตัวบ่งปริมาณ (Quantifier) ได้แก่

- สำหรับทุกตัว (Universal Quantifier: $\forall$)
- สำหรับบางตัว (Existential Quantifier : $\exists$)

2.4.2.3) ความสัมพันธ์ (Relation) ในการเขียนข้อกำหนดของระบบเราจำเป็นอย่างยิ่งที่จะต้องกล่าวอ้างถึงความสัมพันธ์ของข้อมูลโดยทฤษฎีความสัมพันธ์จะแสดงให้เห็นถึงการที่จับคู่ค่าของ เซตต้นทาง (Source Set) กับ เซตปลายทาง (Target Set) ในลักษณะของ หลายต่อหลาย (many to many)

ภายในความสัมพันธ์หนึ่ง ๆ จะประกอบไปด้วยข้อมูล 2 ส่วนคือ โดเมน (Domain) และ พิสัย (Range) โดยโดเมนจะเป็น ซับเซต (Subset) ของเซตต้นทางส่วนพิสัยจะเป็นซับเซตของเซตปลายทาง

2.4.2.4) ฟังก์ชัน (Function) เป็นความสัมพันธ์ชนิดหนึ่งที่มีลักษณะเป็นแบบ หลายต่อหนึ่ง (many to one) หรือ หนึ่งต่อหนึ่ง (one to one) เท่านั้น เนื่องจากความสัมพันธ์ในลักษณะของหลายต่อหลายนั้นไม่เหมาะสมในการที่จะนำมาใช้งานจริงโดยฟังก์ชันจะถูกจัดออกเป็น 2 กลุ่มใหญ่ ๆ คือ

- ฟังก์ชันบางส่วน (Partial Function) เป็นฟังก์ชันที่โดเมนของฟังก์ชันเป็นซับเซตของเซตต้นทาง
- ฟังก์ชันทั้งหมด (Totally Function) เป็นฟังก์ชันที่โดเมนของฟังก์ชันคือเซตต้นทาง

2.5) Z/EVES

Z/EVES เป็นเครื่องมือที่ถูกพัฒนาขึ้นโดยสถาบันโออาร์เอ (ORA) ประเทศแคนาดา ซึ่งเป็นเครื่องมือที่มีความสามารถในการตรวจสอบข้อกำหนดรูปนัยที่เขียนอยู่ในรูปของสัญกรณ์เซดซึ่งสามารถทำงานได้ทั้งในภาวะเชิงโต้ตอบ (Interactive Mode) และภาวะประมวลผลแบบกลุ่ม (Batch Mode) โดยข้อมูลนำเข้าของเครื่องมือจะต้องเขียนอยู่ในรูปแบบของแท็ก (Tag) ของลาเท็กซ์ (Latex) ดังแสดงรายละเอียดอยู่ในภาคผนวก ข

ความสามารถในการตรวจสอบข้อกำหนดของ Z/EVES นั้นประกอบไปด้วยการตรวจสอบวากยสัมพันธ์ของข้อกำหนด การตรวจสอบทางทฤษฎี และการตรวจสอบด้วยตัวอย่าง

2.5.1) การตรวจสอบวากยสัมพันธ์

เมื่อมีการป้อนข้อกำหนดให้กับ Z/EVES นั้น Z/EVES ก็จะทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดในระหว่างที่ทำการอ่านข้อกำหนดนั้นเข้าไปทันที ซึ่งหากข้อกำหนดนั้นถูกต้องก็จะทำการสร้างทฤษฎีสำหรับข้อกำหนดนั้นขึ้นมาเพื่อเป็นข้อมูลที่จะนำไปใช้ในขั้นตอนการตรวจสอบทฤษฎี

2.5.2) การตรวจสอบทางทฤษฎี

เป็นขั้นตอนที่ Z/EVES จะอาศัยทฤษฎีทางคณิตศาสตร์มาเป็นกลไกในการตรวจสอบโดยในขั้นตอนนี้ Z/EVES จะทำการลดรูปของข้อกำหนดที่จะทำการตรวจสอบลงเพื่อหาว่ามีสถานะอย่างน้อยหนึ่งสถานะที่เป็นไปตามข้อกำหนดที่ได้เขียนไว้

2.5.3) การตรวจสอบด้วยตัวอย่าง

เพื่อเพิ่มความสามารถในการตรวจสอบข้อกำหนดของ Z/EVES จึงมีการเพิ่มการตรวจสอบโดยการระบุข้อมูลสำหรับการตรวจสอบเป็นกรณีไปเพื่อดูผลลัพธ์ของการทำงานของข้อกำหนดว่าตรงตามความต้องการหรือไม่ ซึ่งการตรวจสอบในลักษณะนี้จะช่วยให้เห็นภาพการทำงานของข้อกำหนดได้ดีขึ้น

บทที่ 3
กฎและขั้นตอนวิธีการแปลงโปรแกรมภาษาซีเป็นข้อกำหนดเซต

ในการทำการแปลงโปรแกรมต้นฉบับภาษาซีมาเป็นข้อกำหนดโดยใช้สัญกรณ์เซตลำดับแรกจะต้องทำการหาความสัมพันธ์ระหว่างโปรแกรมภาษาซีและข้อกำหนดเซตเสียก่อนเพื่อทำการกำหนดกฎในการแปลงโดยจะทำการแบ่งกฎที่ได้ออกเป็น 2 ส่วนก็คือ กฎสำหรับการแปลงส่วนการประกาศและกฎสำหรับการแปลงส่วนการทำงาน

จากนั้นจะต้องทำการออกแบบขั้นตอนวิธีที่นำเอากฎที่ได้ทำการกำหนดไว้มาใช้ในการพัฒนาเป็นเครื่องมือสำหรับการแปลง

3.1 กฎสำหรับการแปลง

พิจารณาจากโครงสร้างของโปรแกรมภาษาซีและข้อกำหนดรูปนัยในรูปสัญกรณ์เซตจะพบว่าในการแปลงนั้นเราสามารถทำการแปลงส่วนของการประกาศของโปรแกรมภาษาซีมาเป็นส่วนของการประกาศของข้อกำหนดรูปนัยในรูปสัญกรณ์เซต และทำการแปลงส่วนของการทำงานของโปรแกรมภาษาซีมาเป็นส่วนสัจพจน์ของข้อกำหนดรูปนัยในรูปสัญกรณ์เซต ดังนั้นกฎที่ทำการกำหนดขึ้นจึงแบ่งออกเป็นสองส่วนด้วยกันคือ กฎสำหรับการแปลงส่วนของการประกาศ และกฎสำหรับการแปลงส่วนการทำงาน

3.1.1) กฎสำหรับการแปลงส่วนการประกาศ (Declarative Translation Rules)

ส่วนการประกาศในโปรแกรมภาษาซีเป็นส่วนที่ใช้ในการประกาศตัวแปรเพื่อใช้งานภายในโปรแกรมโดยชนิดของตัวแปรในภาษาโปรแกรมซีแบ่งออกเป็น

- ชนิดข้อมูลพื้นฐาน (Primitive Data Type) ได้แก่ int, short, long, double, float, unsigned และ char
- ชนิดข้อมูลไม่พื้นฐาน (Non-Primitive Data Type) ได้แก่ struct, pointer และ union เป็นต้น

โดยในงานวิจัยนี้จะพิจารณาเฉพาะชนิดข้อมูลพื้นฐาน เมื่อทำการแปลงให้เป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซตนั้นอันดับแรกจะต้องทำการสร้างชนิดของข้อมูลสำหรับข้อกำหนดก่อนเนื่องจากในสัญกรณ์เซตนั้นมีชนิดข้อมูลเพียงชนิดเดียวก็คือชนิดจำนวนเต็ม ดังนั้นในการแปลงจำเป็นจะต้องทำการกำหนดชนิดข้อมูลอื่นขึ้นมาจากชนิดจำนวนเต็มโดยในส่วนนี้จะถูกกำหนดไว้ในลักษณะข้อกำหนดล่วงหน้า (Predefined Specification) ซึ่งจะถูกผนวกรวมไปกับผลลัพธ์ที่ได้จากการแปลงโปรแกรมภาษาซี โดยข้อกำหนดล่วงหน้าทั้งหมดแสดงอยู่ในภาคผนวก ก

ชนิดข้อมูลพื้นฐานในภาษาโปรแกรมซีทั้ง 7 ชนิดนั้นเมื่อทำการแปลงให้เป็นข้อกำหนดรูปนัยนั้นไม่จำเป็นที่จะต้องสร้างชนิดข้อมูลในสัญกรณ์เซตทั้ง 7 ชนิดเนื่องจากข้อแตกต่างระหว่างแต่ละชนิดข้อมูลคือเนื้อที่ในหน่วยความจำที่ต้องการในการเก็บ แต่ในระดับของข้อกำหนดนั้นไม่จำเป็นต้องคำนึงถึงเนื้อที่ในหน่วยความจำที่จะต้องสูญเสียไปในการจัดเก็บ ดังนั้นงานวิจัยนี้จึงทำการจัดกลุ่มชนิดข้อมูลทั้ง 7 ชนิด

โดยพิจารณาจากลักษณะของชนิดข้อมูลซึ่งสามารถแบ่งออกได้เป็น 3 กลุ่มคือชนิดข้อมูลจำนวนเต็ม ชนิดข้อมูลจำนวนจริง และชนิดข้อมูลตัวอักขระ

1) ชนิดข้อมูลจำนวนเต็ม (Integer Data Type)

ชนิดข้อมูลจำนวนเต็มจะประกอบไปด้วยชนิดข้อมูลในภาษาโปรแกรมซี 4 ชนิดด้วยกันคือ int, short, long และ unsigned เนื่องจากในสัญกรณ์เซดมีชนิดข้อมูลจำนวนเต็มอยู่แล้วดังนั้นในการแปลงสามารถที่จะทำการแปลงได้โดยตรงไม่จำเป็นต้องกำหนดชนิดข้อมูลขึ้นมาใหม่ โดยการแปลงจะแบ่งออกเป็น 2 กรณีด้วยกันคือ

กรณีที่ 1 ถ้าทำการประกาศตัวแปรเป็นชนิดจำนวนเต็มแบบมีเครื่องหมาย (int, short, และ long) ในกรณีนี้เมื่อทำการแปลงจะให้ตัวแปรมีชนิดข้อมูลเป็นจำนวนเต็มในสัญกรณ์เซด ($\mathbb{Z}$)

กรณีที่ 2 ถ้าทำการประกาศตัวแปรเป็นชนิดจำนวนเต็มแบบไม่มีเครื่องหมาย (unsigned) ในกรณีนี้เมื่อทำการแปลงจะให้ตัวแปรมีชนิดข้อมูลเป็นจำนวนธรรมชาติ (Natural Number)

ตัวอย่างเช่นถ้าในโปรแกรมภาษาซีมีการประกาศตัวแปรเป็น

```
int counter;
```

เมื่อทำการแปลงจะได้เป็น

counter : $\mathbb{Z}$

แต่ถ้าภายในโปรแกรมมีการประกาศตัวแปร

```
unsigned int nonzero;
```

เมื่อทำการแปลงจะได้เป็น

nonzero : $\mathbb{N}$

2) ชนิดข้อมูลจำนวนจริง (Real Data Type)

ชนิดข้อมูลชนิดที่สองที่จะกล่าวถึงคือชนิดข้อมูลจำนวนจริงซึ่งประกอบไปด้วยชนิดข้อมูลในภาษาโปรแกรมซี 2 ชนิดด้วยกันคือ float และ double โดยการแปลงชนิดข้อมูลนี้ไม่สามารถทำการแปลงได้โดยตรงเนื่องจากในสัญกรณ์เซดไม่มีชนิดข้อมูลนี้ ดังนั้นในการแปลงจำเป็นที่จะต้องทำการแทนชนิดข้อมูลจำนวนจริงโดยอาศัยชนิดข้อมูลจำนวนเต็มที่มีอยู่ซึ่งวิธีการที่เลือกนำมาใช้ในการแทนชนิดข้อมูลจำนวนจริงจะอาศัยแนวคิดในการแทนค่าจำนวนจริงในระบบฮาร์ดแวร์ ซึ่งก็คือการแทนด้วยคู่ลำดับของจำนวนเต็มโดยจำนวนเต็มแรกในคู่ลำดับจะแทนลักษณะเฉพาะ (Characteristic) ของจำนวนจริง และจำนวนเต็มที่สองจะแทนเลขชี้กำลัง (Exponential) ของจำนวนจริง โดยนำมาเขียนให้อยู่ในรูปของสัญกรณ์เซดได้เป็น

$REAL == \{x, y : \mathbb{Z} \bullet (x, y)\}$

โดยที่ x แทน ลักษณะเฉพาะ ของจำนวนจริง

y แทน เลขชี้กำลังของจำนวนจริง

ในขั้นตอนของการแปลงโปรแกรมเมื่อมีการใช้งานชนิดข้อมูลจำนวนเต็มตัวอย่างเช่นถ้าในโปรแกรมมีการใช้งานจำนวนจริงที่มีค่า 5.25 เมื่อทำการแปลงให้อยู่ในรูปของข้อกำหนดทำการแปลงให้

อยู่ในรูปของคู่ลำดับ (525, -2) ส่วนกรณีที่มีการประกาศตัวแปรที่มีชนิดข้อมูลเป็น float หรือ double เมื่อทำการแปลงก็จะกำหนดให้มีชนิดข้อมูลเป็น REAL

ตัวอย่างเช่นถ้าในโปรแกรมมีการประกาศตัวแปรเป็น

float i;

เมื่อทำการแปลงก็จะได้เป็น

i : REAL

นอกจากการกำหนดการแทนชนิดข้อมูลจำนวนจริงแล้วต้องมีการกำหนดตัวดำเนินการต่าง ๆ ที่จะทำงานร่วมกับชนิดข้อมูลจำนวนจริงที่ได้ทำการกำหนดขึ้นเนื่องจากตัวดำเนินการที่มีอยู่ในสัญกรณ์เซตได้ถูกกำหนดขึ้นเพื่อใช้งานร่วมกับชนิดข้อมูลจำนวนเต็มเท่านั้น โดยตัวดำเนินการที่จะต้องทำการกำหนดขึ้นมาจะประกอบด้วยสองกลุ่มคือ

- ตัวดำเนินการทางคณิตศาสตร์ ได้แก่ ตัวดำเนินการบวก ลบ คูณ และหาร ดังแสดงรายละเอียดในรูปที่ 3.1
- ตัวดำเนินการทางตรรกศาสตร์ ได้แก่ ตัวดำเนินการมากกว่า น้อยกว่า มากกว่าหรือเท่ากับ และน้อยกว่าหรือเท่ากับ สำหรับตัวดำเนินการเท่ากับ และไม่เท่ากับ ไม่ต้องกำหนดใหม่เนื่องจากที่มีอยู่นั้นสามารถใช้ได้อยู่แล้ว ดังแสดงในรูปที่ 3.2

$$\_\ RPlus\ \_ : REAL \times REAL \longrightarrow REAL$$
$$\_\ RMinus\ \_ : REAL \times REAL \longrightarrow REAL$$
$$\_\ RMul\ \_ : REAL \times REAL \longrightarrow REAL$$
$$\_\ RDiv\ \_ : REAL \times REAL \longrightarrow REAL$$

$$\forall x,y : REAL \mid second(x) = second(y) \bullet x\ RPlus\ y = (first(x) + first(y), second(x))$$
$$\forall x,y : REAL \mid second(x) = second(y) \bullet x\ RMinus\ y = (first(x) - first(y), second(x))$$
$$\forall x,y : REAL \bullet x\ RMul\ y = (first(x) * first(y), second(x) + second(y))$$
$$\forall x,y : REAL \bullet x\ RDiv\ y = (first(x)\ div\ first(y), second(x) - second(y))$$

รูปที่ 3.1ข้อกำหนดรูปนัยของตัวดำเนินการทางคณิตศาสตร์สำหรับชนิดข้อมูลจำนวนจริง

พิจารณาส่วนสัจพจน์ของข้อกำหนดของตัวดำเนินการทางคณิตศาสตร์ที่แสดงอยู่ในรูปที่ 3.1 ซึ่งแสดงการทำงานของแต่ละตัวดำเนินการโดยจะถูกเขียนอยู่ในรูปของฟังก์ชันซึ่งรับพารามิเตอร์สองตัวและจะคำนวณค่าผลลัพธ์เป็นจำนวนจริงกลับมา สำหรับตัวดำเนินการบวกและลบจะมีข้อจำกัดของพารามิเตอร์เนื่องจากการบวกและการลบจะกระทำได้ก็ต่อเมื่อเลขชี้กำลังของจำนวนจริงทั้งสองจำนวนมีค่าเท่ากัน ส่วนการคูณและการหารไม่มีข้อจำกัดในการทำงาน

สำหรับรูปที่ 3.2 อธิบายรายละเอียดการทำงานของตัวดำเนินการทางตรรกศาสตร์โดยตัวดำเนินการทางตรรกศาสตร์นั้นเมื่อนำมาเขียนในรูปของสัญกรณ์เซตจะอยู่ในรูปของความสัมพันธ์ระหว่างจำนวนจริงสองจำนวนเนื่องจากการทำงานจะเพียงหาว่าจำนวนจริงทั้งสองจำนวนมีความสัมพันธ์ตามที่กำหนดไว้หรือไม่

$$\_\ RG\ \_ : REAL \leftrightarrow REAL$$
$$\_\ RL\ \_ : REAL \leftrightarrow REAL$$
$$\_\ RGE\ \_ : REAL \leftrightarrow REAL$$
$$\_\ RLE\ \_ : REAL \leftrightarrow REAL$$

$$\forall x,y : REAL \bullet (x\ RG\ y) \Leftrightarrow ((second(x) > second(y)) \vee (second(x) = second(y) \wedge first(x) > first(y)))$$
$$\forall x,y : REAL \bullet (x\ RL\ y) \Leftrightarrow ((second(x) < second(y)) \vee (second(x) = second(y) \wedge first(x) < first(y)))$$
$$\forall x,y : REAL \bullet (x\ RG\ y) \Leftrightarrow ((second(x) \geq second(y)) \vee (second(x) = second(y) \wedge first(x) \geq first(y)))$$
$$\forall x,y : REAL \bullet (x\ RL\ y) \Leftrightarrow ((second(x) \leq second(y)) \vee (second(x) = second(y) \wedge first(x) \leq first(y)))$$

รูปที่ 3.2 ข้อกำหนดรูปนัยของตัวดำเนินการทางตรรกศาสตร์สำหรับชนิดข้อมูลจำนวนจริง

พิจารณาข้อกำหนดของตัวดำเนินการในรูปที่ 3.1 และรูปที่ 3.2 จะพบว่าตัวดำเนินการทั้งหมดถูกกำหนดให้ทำงานกับชนิดข้อมูลจำนวนจริงเท่านั้นแต่ในความเป็นจริงมีโอกาสที่จะมีการทำงานร่วมกันระหว่างชนิดข้อมูลจำนวนจริง และชนิดข้อมูลจำนวนเต็มซึ่งเรียกว่านิพจน์ผสม [16] (Mixed-Mode Expression) ในกรณีนี้ตัวดำเนินการที่กำหนดขึ้นไม่สามารถรองรับการทำงานลักษณะนี้ได้ ด้วยเหตุนี้เพื่อให้รองรับการทำงานในลักษณะนี้ได้จึงต้องมีการกำหนดตัวดำเนินการพิเศษอีกหนึ่งตัวดำเนินการที่จะคอยทำหน้าที่เป็นตัวแปลงชนิดข้อมูลจำนวนเต็มให้เป็นชนิดข้อมูลจำนวนจริงซึ่งเรียกว่า ตัวดำเนินการ "i2r" โดยแสดงรายละเอียดในรูปที่ 3.3

$$i2r : \mathbb{Z} \longrightarrow REAL$$

$$\forall x : \mathbb{Z} \bullet i2r(x) = (x, 0)$$

รูปที่ 3.3 ข้อกำหนดรูปนัยของตัวดำเนินการ "i2r"

3) ชนิดข้อมูลอักขระ (Character Data Type)

ชนิดข้อมูลชนิดสุดท้ายที่จะกล่าวถึงก็คือชนิดข้อมูลอักขระซึ่งประกอบด้วยชนิดข้อมูลในภาษาโปรแกรมซีชนิดเดียวคือ char ซึ่งในภาษาโปรแกรมซีนั้นพิจารณาอักขระเป็นเสมือนกับเป็นชนิดข้อมูลจำนวนเต็มที่มีค่าระหว่าง 0 ถึง 255 ทั้งยังสามารถที่จะใช้ตัวดำเนินการทางคณิตศาสตร์ร่วมกับชนิดข้อมูลอักขระได้อีกด้วยดังนั้นในการแปลงชนิดข้อมูลอักขระนั้นจึงทำการกำหนดชนิดข้อมูลอักขระในสัญกรณ์เซตด้วยการกำหนดเป็นจำนวนเต็มที่มีค่าระหว่าง 0 ถึง 255 โดยสามารถเขียนในรูปสัญกรณ์เซตได้เป็น

CHAR == 0..255

หลังจากนั้นเมื่อในโปรแกรมภาษาซีมีข้อความสั่งประกาศตัวแปรที่มีชนิดข้อมูลเป็นชนิดข้อมูลอักขระก็จะทำการแปลงให้เป็นชนิดข้อมูล "CHAR"

ตัวอย่างเช่นถ้าในโปรแกรมมีข้อความสั่ง

char chr;

เมื่อทำการแปลงเป็นข้อกำหนดรูปนัยจะได้เป็น

chr : CHAR

สำหรับในขั้นตอนการแปลงส่วนการทำงานสามารถพิจารณาได้เป็น 2 กรณีคือ

กรณีที่ 1 ถ้าในโปรแกรมมีการเรียกใช้งานในแบบที่เป็นตัวแปรชนิดอักขระ ในกรณีนี้เมื่อทำการแปลงจะต้องทำการแปลงอักขระนั้นให้อยู่ในรูปของรหัสแอสกี (ASCII Code) ของตัวอักขระนั้นเสียก่อนจึงนำเอารหัสแอสกี นั้นไปใช้งานต่อ

ตัวอย่างเช่นถ้าในโปรแกรมมีข้อความสั่ง

chr = 'a';

เมื่อทำการแปลงจะต้องทำการหารหัสแอสกีของอักขระ 'a' ก่อนหลังจากนั้นจึงทำการกำหนดค่าตัวแปรด้วยรหัสแอสกีนั้นดังนั้นข้อความสั่งนี้จะถูกแปลงเป็น

chr = 97

กรณีที่ 2 ถ้าในโปรแกรมมีการเรียกใช้งานในแบบที่เป็นตัวแปรชนิดจำนวนเต็ม ในกรณีนี้เราสามารถทำการแปลงมาเป็นข้อกำหนดได้โดยตรง

ตัวอย่างเช่นถ้าในโปรแกรมมีข้อความสั่ง

chr = 97;

เมื่อทำการแปลงจะได้เป็น

chr = 97

3.1.2) กฎสำหรับการแปลงส่วนการทำงาน (Operative Translation Rules)

ภาษาโปรแกรมซีเป็นภาษาโปรแกรมในกลุ่มของภาษาโปรแกรมเชิงคำสั่งซึ่งข้อความสั่งทั้งหมดสามารถแบ่งได้เป็น 5 ประเภทด้วยกันคือ

1) ข้อความสั่งกำหนดค่า
2) ข้อความสั่งเลือกทางเดิน

3) ข้อความสั่งตามลำดับ

4) ข้อความสั่งวนซ้ำ

5) ข้อความสั่งเรียกใช้ฟังก์ชัน

สำหรับกฎในการแปลงข้อความสั่งทั้ง 5 ประเภทนั้นจะทำหน้าที่ในการที่จะแปลงข้อความสั่งทั้งหมดให้อยู่ในรูปของภาคแสดง โดยภาคแสดงที่ได้นั้นจะได้มาจากการพิจารณาถึงสถานะของตัวแปรหรืออีกนัยหนึ่งก็คือค่าของตัวแปรหลังการทำงานของแต่ละข้อความสั่ง เนื่องจากภาษาโปรแกรมซีนั้นเป็นภาษาโปรแกรมเชิงคำสั่งซึ่งแนวคิดการทำงานนั้นจะอาศัยการเปลี่ยนค่าของตัวแปรไปทีละขั้นจนกระทั่งได้ผลลัพธ์สุดท้ายที่ต้องการ

1) ข้อความสั่งกำหนดค่า

ข้อความสั่งกำหนดค่าในภาษาโปรแกรมซีอยู่ในรูปแบบ x = e โดยที่ x คือตัวแปรที่จะทำการเก็บผลลัพธ์และ e คือค่าที่จะกำหนดให้กับตัวแปรซึ่งอาจจะเป็นได้ทั้งค่าคงที่ หรือนิพจน์คำนวณค่า นอกจากนี้ในภาษาโปรแกรมซีนั้นเราสามารถที่จะทำการเขียนได้ในแบบลดรูปซึ่งได้ทำการสรุปรวมไว้ในตารางที่ 3.1

ตาราง 3.1 ตัวดำเนินการกำหนดค่าในภาษาโปรแกรมซี

Operator	Syntax	Semantic
=	x = e	x = e
+=	x += e	x = x + e
-=	x -= e	x = x – e
*=	x *= e	x = x * e
/=	x /= e	x = x/e

พิจารณาข้อความสั่งกำหนดค่า x = e หลังจากที่ข้อความสั่งนี้ทำงานเสร็จสิ้นจะพบว่าสถานะหลังการทำงานก็คือค่าของตัวแปร x จะต้องมีค่าเท่ากับนิพจน์ e ดังนั้นในการทำการแปลงข้อความสั่งกำหนดค่านั้นสามารถที่จะทำการแปลงได้โดยการพิจารณาว่าข้อความสั่งนั้นเป็นภาคแสดงทางคณิตศาสตร์ได้โดยตรง แต่ในการเขียนข้อความสั่งกำหนดค่านั้นสามารถที่จะเขียนในลักษณะที่อ้างอิงถึงค่าเดิมของตัวแปรนั้นเองได้เช่นเราสามารถเขียนข้อความสั่ง x = x + 5 ได้ซึ่งในกรณีนี้ถ้าเราพิจารณาว่าเป็นภาคแสดงโดยตรงจะพบว่าไม่มีทางที่ภาคแสดงนี้จะเป็นจริงได้เนื่องจากค่าของตัวแปร x ไม่มีทางที่จะมีค่าเท่ากับตัวมันเองบวกกับค่าคงที่ 5 ดังนั้นเราจะต้องทำการเปลี่ยนตัวแปรฝั่งซ้ายและฝั่งขวาของข้อความสั่งเป็นคนละตัวแปรกันโดยการใส่เลขกำกับตัวแปร (Subscript) ให้แต่ละตัวแปร

จากที่ได้กล่าวมาทั้งหมดเราสามารถสรุปขั้นตอนในการแปลงข้อคำสั่งกำหนดค่าได้ดังนี้

1) ทำการแปลงข้อความสั่งกำหนดค่าที่อยู่ในรูปย่อให้อยู่ในรูปเต็ม

2) กำหนดให้ตัวแปรที่มีเลขกำกับเป็น 0 ให้มีค่าเท่ากับตัวแปรเริ่มต้น

3) ทำการใส่เลขกำกับตัวแปรโดยจะเพิ่มเลขกำกับของตัวแปรที่อยู่ฝั่งขวาขึ้นทีละ 1

4) ทำการแปลงเป็นภาคแสดง

ตัวอย่างเช่นถ้าเรามีข้อความสั่งกำหนดค่า

$x = x + 10$

เนื่องจากข้อความสั่งนี้อยู่ในรูปแบบเต็มอยู่แล้วเพราะฉะนั้นอันดับแรกทำการกำหนดให้ตัวแปร x_0,y_0มีค่าเท่ากับตัวแปร x, y ตามลำดับ และทำการแปลงข้อความสั่ง $x = y + 10$ โดยเพิ่มเลขกำกับตัวแปรขึ้น 1 เพราะฉะนั้นจะได้เป็น

$x_0 = x \wedge y_0 = y$

$x_1 = y_0 + 10$

2) ข้อความสั่งเลือกทางเดิน

ข้อความสั่งเลือกทางเดินในภาษาโปรแกรมซีสามารถที่จะแบ่งออกเป็น 3 กลุ่มด้วยกันคือ แบบหนึ่งทางเลือก (Single Branching) แบบสองทางเลือก (Double Branching) และแบบหลายทางเลือก (Multiple Branching)

2.1) แบบทางเลือกเดียว ข้อความสั่งเลือกทางเดินแบบทางเลือกเดียวได้แก่ข้อความสั่ง if แบบไม่มี else ซึ่งจะเขียนอยู่ในรูป

```
if (G) {
        S1
}
```

โดยที่ G คือเงื่อนไขของข้อความสั่ง

S_1 คือข้อความสั่งที่จะทำงานเมื่อเงื่อนไข G เป็นจริง

พิจารณาข้อความสั่งเลือกทางเดินข้างต้นสถานะหลังการทำงานนั้นเป็นไปได้ 2 ทางด้วยกันคือ กรณีแรกถ้าเงื่อนไขของข้อความสั่ง if เป็นจริง กรณีที่สองคือถ้าเงื่อนไขของข้อความสั่ง if เป็นเท็จ

กรณีแรกถ้าเงื่อนไขของข้อความสั่ง if เป็นจริงกรณีนี้สถานะหลังการทำงานก็คือเงื่อนไข G จะต้องเป็นจริงและสถานะหลังการทำงานของข้อความสั่ง S_1 จะต้องเป็นจริงด้วย ซึ่งเขียนในรูปภาคแสดงได้เป็น

$G \wedge S_1$

กรณีที่สองถ้าเงื่อนไขของข้อความสั่ง if เป็นเท็จกรณีสถานะหลังการทำงานก็คือเงื่อนไข G จะต้องเป็นเท็จด้วย ซึ่งเขียนในรูปภาคแสดงได้เป็น

$\neg G$

ในการทำงานแต่ละครั้งของข้อความสั่ง if นั้นจะเกิดสถานะหลังการทำงานได้เพียงอย่างใดอย่างหนึ่งเท่านั้นดังนั้นจากเงื่อนไขการทำงานทั้งสองกรณีเราจะทำการเชื่อมเข้าด้วยกันโดยการใช้ Disjunction จะได้สถานะหลังการทำงานของข้อความสั่ง if เป็น

$(G \wedge S_1) \vee \neg G$

พิจารณาภาคแสดงแสดงสถานะหลังการทำงานของข้อความสั่ง if จะเห็นว่าเราไม่สามารถตัดสถานะหลังการทำงานในกรณีที่เงื่อนไขเป็นเท็จได้เนื่องจากถ้าเราตัดในส่วนนี้ทิ้งเราจะไม่สามารถหาผลลัพธ์ได้ และการที่เงื่อนไขของข้อความสั่งเป็นเท็จนั้นไม่ได้หมายความโปรแกรมนั้นจะหยุดทำงานเพียงแต่โปรแกรมนั้นจะไม่มีการทำงานในส่วนที่อยู่ภายในเงื่อนไขเท่านั้น

2.2) แบบสองทางเลือก ข้อความสั่งเลือกทางเดินแบบสองทางเลือกได้แก่ข้อความสั่ง if ที่มีการทำงานทั้งในกรณีที่เงื่อนไขเป็นจริงและเงื่อนไขเป็นเท็จ โดยมีรูปแบบการเขียนดังนี้

```
if (G) {
        S1;
}else
{
        S2;
}
```

โดยที่ G คือเงื่อนไขของข้อความสั่ง

S_1 คือข้อความสั่งที่จะทำงานเมื่อเงื่อนไข G เป็นจริง

S_2 คือข้อความสั่งที่จะทำงานเมื่อเงื่อนไข G เป็นเท็จ

สำหรับการแปลงข้อความสั่งเลือกทางเดินแบบสองทางเลือกนั้นจะเหมือนกับการแปลงข้อความสั่งเลือกทางเดินแบบทางเลือกเดียว เพียงแต่ในกรณีที่เงื่อนไขของข้อความสั่งเป็นเท็จสถานะหลังการทำงานนอกจากเงื่อนไขข้อความสั่งจะต้องเป็นจริงแล้วสถานะหลังการทำงานของข้อความสั่ง S_2 จะต้องเป็นจริงด้วย ซึ่งเขียนในรูปภาคแสดงได้เป็น

$$(G \wedge S_1) \vee (\neg G \wedge S_2)$$

2.3) แบบหลายทางเดิน ข้อความสั่งเลือกทางเดินแบบหลายทางเดินในภาษาโปรแกรมซีได้แก่ข้อความสั่ง switch ซึ่งมีรูปแบบการเขียนดังนี้

```
switch (C) {
        case C1 : S1;break;
        case C2 : S2;break;
        ..................
        case Cn : Sn;break;
        default : Sd
}
```

โดยที่ C คือนิพจน์ที่จะใช้ในการตรวจสอบ

$C_1, C_2, C_3, \ldots, C_n$ คือค่าที่จะใช้ในการตรวจสอบ

$S_1, S_2, S_3, \ldots, S_n, S_d$ คือข้อความสั่งที่จะทำงานเมื่อเงื่อนไขเป็นจริง

สำหรับการแปลงข้อความสั่ง switch นั้นเราจะอาศัยกฎการแปลงสำหรับข้อความสั่ง if มาใช้ เนื่องจากเราสามารถที่จะทำการแปลงข้อความสั่ง switch ให้อยู่ในรูปข้อความสั่ง if แบบ nested ได้ จากข้อความสั่ง switch ข้างต้นเมื่อทำการแปลงเป็นข้อความสั่ง if ได้ดังนี้

```
if (C == C1) {
        S1;
}else if (C == C2) {
        S2;
}else if (...) {
        .....
}else if (C == Cn){
        Sn;
} else {
        Sd;
}
```

หลังจากนั้นด้วยกฎของการแปลงข้อความสั่งเลือกทางเดินเราจะสามารถทำการแปลงข้อความสั่ง if ที่ได้นี้ให้อยู่ในรูปของภาคแสดงได้เป็น

$$(C = C_1) \wedge (S_1) \vee$$
$$\neg(C = C_1) \wedge (C = C_2) \wedge (S_2)) \vee$$
$$(\neg(C = C_1) \wedge \neg(C = C_2) \wedge \ldots)) \vee$$
$$(\neg(C = C_1) \wedge \neg(C = C_2) \wedge \ldots \wedge (C = C_n) \wedge (S_n)) \vee$$
$$(\neg(C = C_1) \wedge \neg(C = C_2) \wedge \ldots \wedge \neg(C = C_n) \wedge (S_d))$$

3) ข้อความสั่งตามลำดับ

การทำงานของโปรแกรมเชิงคำสั่งจะทำงานในลักษณะเรียงตามลำดับกล่าวคือจะทำงานทีละคำสั่งเรียงลำดับกันไป พิจารณาข้อความสั่งตามลำดับ

$S_1;S_2;S_3;\ldots\ldots;S_n;$

โดยที่ $S_1,S_2,S_3,\ldots,S_n$ คือข้อความสั่งในภาษาโปรแกรมซี

หลังจากที่ข้อความสั่ง S1 ทำงานเสร็จจะเกิดสถานะหลังการทำงานของข้อความสั่ง S1 ซึ่งจะเปลี่ยนมาเป็นสถานะก่อนการทำงานของ S2 ในทำนองเดียวกันสถานะหลังการทำงานของ S2 จะเปลี่ยนเป็นสถานะก่อนการทำงานของ S3 และเป็นเช่นนี้เรียงตามลำดับไปดังแสดงในรูปที่ 3.4 และเนื่องจากสถานะหลังการทำงานจะต้องไม่ทำให้สถานะหลังก่อนการทำงานเสียไปซึ่งหมายถึงสถานะหลังการทำงานก็จะต้องเป็นจริงตามเงื่อนไขก่อนการทำงานด้วย ดังนั้นในการแปลงให้เป็นข้อกำหนดรูปนัยเราจะนำเอาสถานะหลังการทำงานของแต่ละข้อความสั่งมาทำการเชื่อมด้วยตัวดำเนินการและ ($\wedge$)

ตัวอย่างเช่นพิจารณาส่วนของโปรแกรม

$x = 5; y = x + 10; y = y - 1;$

จากส่วนของโปรแกรมข้างต้นเราจะทำการหาสถานะหลังการทำงานของแต่ละคำสั่งหลังจากนั้นจึงทำการเชื่อมเข้าด้วยกันด้วยตัวดำเนินการและ($\wedge$)ซึ่งจะได้เป็น

$(x_1 = 5) \wedge (y_1 = x_1 + 10) \wedge (y_2 = y_1 - 1)$

$\{pre\} S_1 \{pre \wedge post_1\}$

$\{pre \wedge post_1\} S_2 \{pre \wedge post_1 \wedge post_2\}$

$\{pre \wedge post_1 \wedge post_2\} S_3 \{pre \wedge post_1 \wedge post_2 \wedge post_3\}$

.....

$\{pre \wedge post_1 \wedge post_2 \wedge post_3 \wedge .... \wedge post_{n-1}\} S_n \{pre \wedge post_1 \wedge post_2 \wedge post_3 \wedge ... \wedge post_{n-1} \wedge post_n\}$

โดยที่ pre คือเงื่อนไขก่อนการทำงาน

post คือเงื่อนไขหลังการทำงาน

รูปที่ 3.4 ความสัมพันธ์ระหว่างสถานะก่อนการทำงานและหลังการทำงาน

4) ข้อความสั่งวนซ้ำ

ข้อความสั่งวนซ้ำในภาษาโปรแกรมซีนั้นประกอบไปด้วย 3 ข้อความสั่งคือ ข้อความสั่ง while ข้อความสั่ง do..while และข้อความสั่ง for

- ข้อความสั่ง while มีรูปแบบการเขียนดังนี้

```
while (G) {
    S1;
}
```

โดยที่ G คือ เงื่อนไขของข้อความสั่งโดยจะต้องเป็นจริงตลอดการทำงานของ while

S1 คือข้อความสั่งที่จะทำงานเมื่อเงื่อนไข G เป็นจริง

สำหรับการหาสถานะหลังการทำงานของข้อความสั่งวนซ้ำนั้นเราจะต้องทำการหาข้อมูลอย่างหนึ่งของข้อความสั่งวนซ้ำซึ่งเรียกว่าตัวยืนยงวนซ้ำ (Loop invariant) โดยนิยามของตัวยืนยงวนซ้ำใน [17] สามารถกล่าวได้ว่าตัวยืนยงวนซ้ำคือภาคแสดงที่จะต้องมีค่าเป็นจริงก่อนและหลังที่ส่วนการทำงานของข้อความสั่งวนซ้ำจะทำงานในแต่ละรอบการทำงาน โดยในข้อความสั่งวนซ้ำหนึ่ง ๆ สามารถจะมีตัวยืนยงวนซ้ำมากกว่าหนึ่ง แต่ตัวยืนยงวนซ้ำที่ต้องการจะนำมาใช้นั้นจะต้องเป็นตัวยืนยงวนซ้ำที่แสดงถึงสถานะสุดท้ายของตัวแปรที่ถูกเปลี่ยนแปลงค่าภายในข้อความสั่งวนซ้ำ

ตัวอย่างเช่นพิจารณาส่วนของโปรแกรม

```
a = 0;
z = 0;
while (a != y) {
        z = z + x;
        a = a + 1;
}
```

สมมติให้ตัวแปร x และ y เริ่มต้นมีค่าเท่ากับ 2 และ 5 ตามลำดับจากส่วนของโปรแกรมข้างต้น สามารถทำการสร้างตารางติดตามค่า (Trace Table) ของแต่ละตัวแปรได้ดังแสดงในตาราง 3.2 หากพิจารณาในตารางเพื่อหาตัวยืนยงวนซ้ำของข้อความสั่งวนซ้ำจะเห็นว่าทุก ๆ ครั้งที่ข้อความสั่งวนซ้ำก่อนและหลังการทำงานค่าของตัวแปร z จะต้องมีค่าเท่ากับ ค่าของตัวแปร x คูณกับค่าของตัวแปร a เสมอซึ่งสามารถที่จะสรุปได้ว่าตัวยืนยงของข้อความสั่งวนซ้ำนี้ก็คือภาคแสดง $z = x * a$

ตาราง 3.2 ตารางติดตามค่าของตัวแปร

x	y	z	a
2	5	0	0
2	5	2	1
2	5	4	2
2	5	6	3
2	5	8	4

สำหรับการแปลงข้อความสั่งวนซ้ำนั้นหลังจากการทำงานของข้อความสั่งสถานะหลังการทำงานก็คือ เงื่อนไขของข้อความสั่งวนซ้ำจะต้องเป็นเท็จ และสถานะหลังการทำงานของส่วนการทำงานได้มาจากตัวยืนยงวนซ้ำ ดังนั้นเมื่อนำเอาข้อมูลทั้งสองมาเชื่อมเข้าด้วยกันด้วยตัวดำเนินการและ จากส่วนของโปรแกรมข้างต้นเราจะสามารถหาสถานะหลังการทำงานของข้อความสั่งวนซ้ำได้เป็น

$$\neg(a \neq y) \wedge (z = x * a)$$

- ข้อความสั่ง do..while มีรูปแบบการเขียนดังนี้

```
do {
        S1;
}while (G)
```

สำหรับการแปลงข้อความสั่ง do..while นั้นจะอาศัยกฎที่ใช้ในการแปลงข้อความสั่ง while เนื่องจากเราสามารถที่จะทำการเปลี่ยนรูปของข้อความสั่ง do..while ให้อยู่ในรูปของข้อความสั่ง while ได้ ซึ่งจากรูปแบบการเขียนข้อความสั่ง do..while ข้างต้นสามารถที่จะทำการเปลี่ยนเป็นข้อความสั่ง while ได้เป็น

S_1;
while (G) {
S_1;
}

หลังจากนั้นก็จะทำการแปลงเช่นเดียวกับการแปลงข้อความสั่ง while

- ข้อความสั่ง for มีรูปแบบการเขียนดังนี้

for (S_1;G;S_2) {
S_3;
}

สำหรับการแปลงข้อความสั่ง for ก็เช่นเดียวกันคือจะทำการแปลงให้อยู่ในรูปของข้อความสั่ง while เสียก่อนหลังจากนั้นจึงทำการแปลงเช่นเดียวกับข้อความสั่ง while ซึ่งจากรูปแบบของข้อความสั่ง for ข้างต้นสามารถที่จะทำการแปลงให้อยู่ในรูปของข้อความสั่ง while ได้เป็น

S_1;
while (G) {
S_3;
S_2;
}

หลังจากนั้นจึงทำการแปลงข้อความสั่ง while ที่ได้ด้วยวิธีการที่ได้กล่าวมาแล้วข้างต้น

5) ข้อความสั่งเรียกใช้ฟังก์ชัน

ในการเขียนโปรแกรมด้วยภาษาโปรแกรมซีเราสามารถทำการแบ่งโปรแกรมเป็นส่วนย่อย ๆ เรียกว่าฟังก์ชัน หลังจากนั้นเราสามารถเรียกให้ฟังก์ชันนั้นขึ้นมาทำงานจากอีกฟังก์ชันหนึ่งได้และสามารถส่งผ่านข้อมูลระหว่างกันได้โดยผ่านทางพารามิเตอร์และการส่งคืนค่ากลับ โดยวิธีในการส่งผ่านพารามิเตอร์นั้นโดยทั่วไปจะแบ่งออกเป็น 3 วิธีด้วยกันคือ

- การส่งผ่านค่า (Pass by value) เป็นการส่งผ่านเฉพาะค่าของตัวแปรเข้าไปการแก้ไขภายในฟังก์ชันไม่ส่งผลกระทบต่อตัวแปรภายนอก
- การส่งผ่านแบบอ้างอิง (Pass by Reference)
- การส่งผ่านแบบคัดลอกเข้าและคัดลอกออก (Pass by copy-in copy-out)

ถึงแม้ว่าจะมีวิธีในการส่งผ่านพารามิเตอร์ถึง 3 วิธีด้วยกันแต่ในภาษาโปรแกรมซีนั้นจะมีเพียงการส่งผ่านแบบการส่งผ่านค่าเท่านั้น แต่ถ้าต้องการใช้การส่งผ่านพารามิเตอร์จะต้องอาศัยตัวแปรชนิดตัวชี้ (Pointer) มาเป็นตัวช่วยในการส่งผ่าน

หากพิจารณาการเขียนโปรแกรมในกรณีที่ไม่มีการแบ่งเป็นฟังก์ชันนั้น การเขียนก็จะทำโดยการรวมเอาการทำงานในส่วนฟังก์ชันย่อยเข้ามาไว้ภายในตัวมันเองและทำการกำหนดตัวแปรที่จะส่งผ่านค่า

ให้เท่ากับตัวแปรพารามิเตอร์ ด้วยสาเหตุนี้ในการแปลงข้อความสั่งเรียกใช้ฟังก์ชันนั้นจะทำโดยอาศัยการรวมเค้าร่าง (Schema Inclusion) เข้ามาเป็นตัวช่วยในการแปลง โดยแบ่งเป็นขั้นตอนดังนี้

1) รวมเอาเค้าร่างของฟังก์ชันที่ถูกเรียกเข้ามาในเค้าร่างฟังก์ชันที่เรียก โดยการใส่ชื่อของเค้าร่างที่ต้องการรวมไว้ในส่วนการประกาศของเค้าร่างหลัก
2) ทำการกำหนดค่าของตัวแปรที่ต้องการส่งผ่านไปในฟังก์ชันให้เท่ากับตัวแปรพารามิเตอร์
3) กำหนดค่าที่คืนกลับมาจากฟังก์ชันให้เท่ากับตัวแปรที่จะทำการรับค่า

ตัวอย่างเช่นพิจารณาโปรแกรมด้านล่าง

```
int findMax(int a,b) {
        int ret;
        if (a >= b)
                ret = a;
        else
                ret = b;
        return ret;
}
int main()
{
        int x,y,z;
        z = findMax(x,y);
}
```

จากโปรแกรมขั้นต้นเราจะทำการแปลงฟังก์ชันที่ถูกเรียกใช้ก่อนก็คือฟังก์ชัน findMax ให้เป็นเค้าร่างก่อน จะได้เป็น

findMax

$a?, b? : \mathbb{Z}$

$result! : \mathbb{Z}$

$ret : \mathbb{Z}$

$((a? \geq b?) \land (ret = a?)) \lor (\neg(a? \geq b?) \land (ret = b?))$

$result! = ret$

หากพิจารณาในเค้าร่าง findMax จะเห็นว่าพารามิเตอร์ของฟังก์ชัน findMax ได้แก่ ตัวแปร a และ b ถูกแปลงให้กลายมาเป็นตัวแปรเข้าของเค้าร่าง และมีตัวแปร result! เพิ่มขึ้นมาเพื่อทำหน้าที่ในการเก็บผลลัพธ์ของฟังก์ชัน หลังจากที่ได้ทำการแปลงฟังก์ชันที่ถูกเรียกใช้เป็นที่เรียบร้อยแล้วขั้นต่อไปคือทำการแปลงในส่วนของฟังก์ชันที่มีการเรียกใช้ฟังก์ชันอื่นซึ่งก็คือฟังก์ชัน main จะได้เป็น

main

$x,y,z : \mathbb{Z}$

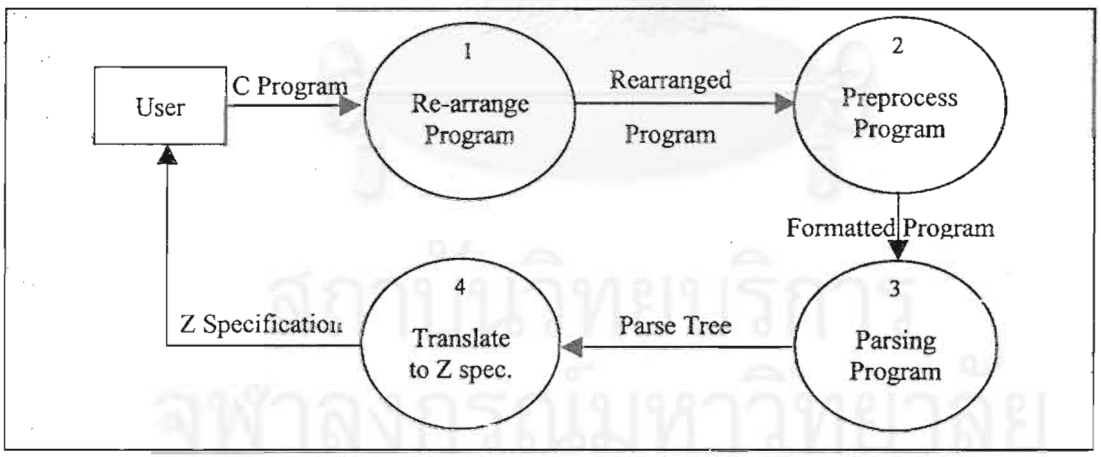
findMax

$x = a? \wedge y = b? \wedge z = result!$

จากเค้าร่าง main จะเห็นว่าในส่วนของการประกาศมีการรวมเอาเค้าร่าง findMax เข้ามาซึ่งจะหมายถึงว่าการทำงานของ main นั้นนอกจากจะต้องเป็นไปตามข้อกำหนดที่ระบุไว้ภายใน main แล้วจะต้องเป็นไปตามข้อกำหนดที่ระบุไว้ภายใน findMax อีกด้วย และในส่วนของสัจพจน์จะเห็นว่ามีการระบุไว้ว่าค่าของพารามิเตอร์ของฟังก์ชัน findMax จะต้องมีค่าเท่ากับค่าของตัวแปรที่ได้ทำการส่งผ่านเข้าไปซึ่งได้แก่ตัวแปร x, y และสำหรับตัวแปรที่รับค่าผลลัพธ์ที่ได้จากฟังก์ชันในที่นี้คือตัวแปร z จะต้องมีค่าเท่ากับตัวแปรผลลัพธ์ของเค้าร่าง findMax ซึ่งก็คือตัวแปร result!

3.2 ขั้นตอนวิธีการแปลง

หลังจากทำการกำหนดกฎสำหรับการแปลงโปรแกรมภาษาซีมาเป็นข้อกำหนดรูปนัยแล้ว ในหัวข้อนี้จะพิจารณาในส่วนของขั้นตอนการแปลงเริ่มตั้งแต่ได้รับโปรแกรมต้นฉบับภาษาซีมาจนกระทั่งสุดท้ายได้เป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ซึ่งในรูปที่ 3.5 แสดงขั้นตอนต่าง ๆ ในการแปลง โดยแบ่งเป็น 4 ขั้นตอนด้วยกันคือ



รูปที่ 3.5 ขั้นตอนการแปลงโปรแกรมภาษาซีเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

3.2.1) ส่วนการจัดเรียงโปรแกรมต้นฉบับ (Re-arrange Program)

ในการแปลงโปรแกรมต้นฉบับภาษาซีเป็นข้อกำหนดเซดนั้นอันดับแรกจะต้องทำการเรียงลำดับฟังก์ชันภายในโปรแกรมต้นฉบับตามลำดับการเรียกใช้งาน เนื่องจากในข้อกำหนดเซดหนึ่ง ๆ นั้นจะสามารถอ้างถึงข้อกำหนดในส่วนอื่นได้ก็ต่อเมื่อมีการกำหนดไว้ก่อนล่วงหน้าเท่านั้น เพราะการอ่านข้อกำหนดจะทำการอ่านจากบนลงล่าง แต่ในโปรแกรมภาษาซีนั้นสามารถเขียนฟังก์ชันที่ถูกเรียกไว้หลัง

ฟังก์ชันที่เรียกได้ ดังนั้นก่อนอื่นจะต้องมีการเรียงลำดับฟังก์ชันต่าง ๆ ตามลำดับการใช้งานโดยฟังก์ชันที่ไม่มีการเรียกใช้ฟังก์ชันอื่นจะถูกนำมาไว้ส่วนต้นของโปรแกรม

ตัวอย่างเช่นพิจารณาโปรแกรมภาษาซี

```
int FunctionA() {
        ……..
        int FunctionB();
        int FunctionC();
        FunctionC();
        FunctionB();
        ……
}
int FunctionB() {
        ……
}
int FunctionC() {
        ……
}
```

จากโปรแกรมภาษาซีข้างต้นจะเห็นว่าฟังก์ชัน FunctionA นั้นมีการเรียกใช้งานฟังก์ชัน FunctionB และ FunctionC แต่ว่าทั้งสองฟังก์ชันนั้นถูกประกาศไว้หลังฟังก์ชัน FunctionA ซึ่งถ้าเราทำการแปลงโดยไม่ได้ทำการเรียงเสียก่อนเราจะไม่สามารถทำการอ้างถึงเค้าร่างของฟังก์ชัน FunctionB และ FunctionC ดังนั้นจากโปรแกรมข้างต้นจะต้องทำการเรียงฟังก์ชันทั้งสามดังนี้

```
int FunctionB() {
        ……
}
int FunctionC() {
        ……
}
int FunctionA(){
        int FunctionB();
        int FunctionC();
        FunctionC();
        FunctionB();
        ……
}
```

3.2.2) ส่วนการจัดรูปแบบโปรแกรมต้นฉบับ (Preprocess Program)

สำหรับขั้นตอนนี้จะทำหน้าที่ในการปรับรูปแบบของคำสั่งที่จำเป็นต้องมีการเปลี่ยนรูปไปเป็นข้อความสั่งอื่นก่อนที่จะสามารถทำการแปลงได้ซึ่งจะประกอบไปด้วย

- ข้อความสั่งกำหนดค่าที่อยู่ในรูปอย่างย่อ อย่างที่ได้กล่าวไว้ในหัวข้อที่ 3.1.2 ว่าข้อความสั่งกำหนดค่านั้นเราสามารถที่จะเขียนอยู่ในรูปอย่างย่อได้ดังสรุปไว้ในตารางที่ 3.1 ซึ่งในการแปลงนั้นจะต้องทำการแปลงข้อความสั่งนี้ให้อยู่ในรูปแบบเต็มก่อนทำการแปลง
- ข้อความสั่ง switch สำหรับการแปลงข้อความสั่ง switch นั้นจะต้องทำการแปลงให้อยู่ในรูปของคำสั่ง if เสียก่อนเพื่อที่จะนำเอากฎสำหรับการแปลงข้อความสั่ง if มาใช้ในการแปลง
- ข้อความสั่ง do..while และข้อความสั่ง for สำหรับข้อความสั่งทั้งสองนี้จะถูกเปลี่ยนรูปให้เป็นข้อความสั่ง while ก่อนที่จะทำการแปลงซึ่งรายละเอียดได้กล่าวไว้ในหัวข้อ 3.1

3.2.3) ส่วนการสร้างต้นไม้วิภัช (Parse Program)

หลังจากขั้นตอนการเตรียมโปรแกรมต้นฉบับเป็นที่เรียบร้อยแล้วขั้นต่อไปจะนำเอาโปรแกรมต้นฉบับที่ได้มาทำการสร้างเป็นต้นไม้วิภัช โดยในขั้นตอนนี้จะอาศัยเครื่องมือชื่อ Ctree รุ่น 1.4 มาเป็นตัวสร้างต้นไม้วิภัชโดยเครื่องมือนี้จะรับเอาโปรแกรมต้นฉบับภาษาซีเป็นข้อมูลเข้าหลังจากนั้นจะสร้างต้นไม้วิภัชเป็นผลลัพธ์โดยต้นไม้วิภัชที่ได้จะประกอบด้วยโหนด (Node) ภายในจะประกอบไปด้วยโหนด 3 ประเภทคือ

- โหนด Tree จะเป็นโหนดที่สามารถจะมีลูกแตกออกไปได้ 2 โหนดด้วยกันคือโหนดซ้าย (Left Node) และโหนดขวา (Right Node) โดยโหนดประเภทนี้จะใช้แทนข้อความสั่งที่ต้องการแตกสาขาแค่เพียงสองสาขาได้แก่ ข้อความสั่งกำหนดค่า ข้อความสั่งวนซ้ำ while หรือนิพจน์ทั่ว ๆ ไป เป็นต้น
- โหนด If จะเป็นโหนดที่ใช้ในการแทนข้อความสั่งเลือกทางเดิน if โดยสามารถจะมีลูกได้ทั้งหมด 3 โหนดด้วยกันคือ
    1) โหนดเงื่อนไข (Condition Node) จะเก็บเงื่อนไขของข้อความสั่ง if โหนด
    2) โหนด Then (Then Node) จะเก็บข้อความสั่งที่จะทำงานเมื่อเงื่อนไขของข้อความสั่ง if เป็นจริง
    3) โหนด Else (Else Node) จะเก็บข้อความสั่งที่จะทำงานเมื่อเงื่อนไขของข้อความสั่ง if เป็นเท็จ
- โหนด For จะเป็นโหนดที่ใช้ในการแทนข้อความสั่งวนซ้ำ for และข้อความสั่งแสดงตนของฟังก์ชัน (Function Signature) โดยสามารถจะมีลูกได้ทั้งหมด 4 โหนดคือ
    1) โหนด init (Init Node) จะเก็บข้อความสั่งกำหนดค่าเริ่มต้นของข้อความสั่ง for หรือเก็บชนิดของข้อมูลที่จะคืนกลับมาจากฟังก์ชัน
    2) โหนดเงื่อนไข (Condition Node) จะเก็บเงื่อนไขของข้อความสั่ง for หรือเก็บชื่อของฟังก์ชันในกรณีที่นำไปใช้ในการแทนข้อความสั่งแสดงตนของฟังก์ชัน

3) โหนด incr (Incr Node) จะเก็บข้อความสั่งที่จะทำงานทุกครั้งในแต่ละรอบที่ข้อความสั่ง for ทำงานเสร็จ หรือใช้เก็บพารามิเตอร์ของฟังก์ชันทั้งชนิดข้อมูลของพารามิเตอร์และชื่อของพารามิเตอร์

4) โหนดข้อความสั่ง (Statement Node) จะเก็บข้อความสั่งที่จะทำงานในขณะที่เงื่อนไขของข้อความสั่ง for เป็นจริง หรือใช้เก็บข้อความสั่งภายในฟังก์ชัน

- โหนดใบ (Leaf Node) จะเป็นโหนดที่ใช้ในการแทนโหนดที่ไม่มีลูกต่อไปอีกซึ่งก็คือ Token แต่ละ Token ที่ถูกแบ่งออกในช่วงของการทำ lexical analysis โดยภายในจะมีเขตข้อมูล (Field) ที่จะเก็บค่าของโหนดเอาไว้โดยค่าภายในอาจจะเป็นตัวเลขหรือตัวอักษรก็ได้

นอกจากนี้ในโหนดแต่ละประเภทที่กล่าวมาข้างต้นนั้นจะมีเขตข้อมูลที่จะบอกว่าโหนดนั้นมีประเภทอะไรและอยู่ตรงตำแหน่งใดของโปรแกรมต้นฉบับ

3.2.4) ส่วนการแปลงเป็นข้อกำหนดเซด (Translate to Z Specification)

ส่วนการแปลงเป็นข้อกำหนดเซดเป็นส่วนที่ทำการแปลงฟังก์ชันต่าง ๆ ในโปรแกรมต้นฉบับภาษาซีให้เป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด โดยจะทำการแปลงทีละฟังก์ชันซึ่งการแปลง 1 ฟังก์ชันในสัญกรณ์เซดจะได้ 1 เค้าร่างในข้อกำหนดเซด

เนื่องจากภาษาโปรแกรมซีเป็นภาษาที่มีโครงสร้างเป็นบล็อกซึ่งทำให้สามารถที่จะประกาศตัวแปรที่มีชื่อเหมือนกันแต่อยู่คนละบล็อกได้ดังที่ได้กล่าวมาแล้ว แต่ในสัญกรณ์เซดนั้นถึงแม้จะสามารถแบ่งข้อกำหนดออกเป็นเค้าร่างย่อย ๆ ได้แต่ก็ไม่สามารถที่จะมีตัวแปรที่มีชื่อเหมือนกันภายในเค้าร่างเดียวกันได้ ดังนั้นในขั้นตอนการแปลงจำเป็นจะต้องทำการเปลี่ยนชื่อตัวแปรเดิมโดยการนำเอาชื่อฟังก์ชันและบล็อกที่ตัวแปรนั้นปรากฏอยู่มาเป็นตัวกำกับแล้วคั่นแต่ละส่วนด้วยเครื่องหมาย “_”

ตัวอย่างเช่น ถ้าในโปรแกรมประกอบด้วย 2 ฟังก์ชันคือฟังก์ชัน A และ ฟังก์ชัน B ซึ่งฟังก์ชัน A มีการเรียกใช้ฟังก์ชัน B และทั้งสองฟังก์ชันมีตัวแปรชื่อ counter เหมือนกันซึ่งถูกประกาศอยู่ที่บล็อกนอกสุดในกรณีนี้ตัวแปร counter ของฟังก์ชัน A จะต้องถูกเปลี่ยนชื่อเป็น A_0_counter โดยที่ A คือชื่อของฟังก์ชัน สำหรับ 0 คือเป็นตัวแปรที่ประกาศไว้บล็อกที่ 0 และ counter คือชื่อตัวแปร ส่วนตัวแปร counter ของฟังก์ชัน B จะต้องถูกเปลี่ยนชื่อเป็น B_0_counter โดยที่ B, 0 และ counter คือชื่อฟังก์ชัน บล็อกที่ตัวแปรถูกประกาศและ ชื่อของตัวแปรตามลำดับ

เนื่องจากในโปรแกรมภาษาซีแบ่งออกเป็นสองส่วนนั่นคือส่วนประกาศส่วนกลาง และส่วนการทำงานของโปรแกรมซึ่งจะประกอบด้วยฟังก์ชันต่าง ๆ โดยในการแปลงส่วนการประกาศส่วนกลางจะถูกแปลงให้อยู่ในรูปของนิยามสัจพจน์ และฟังก์ชันต่าง ๆ จะถูกแปลงให้อยู่ในรูปของเค้าร่างแบบกล่อง

1) การแปลงส่วนการประกาศส่วนกลาง

การแปลงส่วนประกาศส่วนกลางจะเป็นขั้นที่จะทำการแปลงตัวแปรที่ถูกประกาศไว้เพื่อที่จะใช้งานตลอดการทำงานของโปรแกรมซึ่งในการแปลงเป็นข้อกำหนดเราจะทำการแปลงให้อยู่ในรูปของนิยามสัจพจน์ซึ่งเมื่อแปลงให้อยู่ในรูปของนิยามสัจพจน์ก็จะสามารถที่จะอ้างถึงได้จากส่วนไหนของข้อกำหนดก็ได้

อย่างที่กล่าวไว้ข้างต้นว่าชื่อตัวแปรจะต้องมีการเปลี่ยนชื่อเพื่อป้องกันการซ้ำกันของชื่อตัวแปรเช่นเดียวกันกับตัวแปรที่ถูกประกาศเป็นส่วนกลางก็จะต้องถูกเปลี่ยนชื่อด้วย แต่สำหรับชื่อของตัวแปรที่ถูกประกาศเป็นส่วนกลางจะไม่มีชื่อฟังก์ชันดังนั้นจึงทำการตั้งคำสำคัญ (Keyword) เพื่อใช้แทนชื่อฟังก์ชันซึ่งในงานวิจัยนี้เลือกใช้คำว่า "Global" สำหรับหมายเลขบล็อกนั้นก็จะกำหนดให้เป็น 0

ตัวอย่างเช่นถ้ามีการประกาศตัวแปร share เป็นตัวแปรส่วนกลางโดยมีชนิดเป็น int เมื่อทำการแปลงตัวแปรนี้จะถูกเปลี่ยนชื่อเป็น Global_0_share โดยจะถูกบรรจุอยู่ในนิยามสัจพจน์คือ

$$\mid \text{Global\_0\_share} : \mathbb{Z}$$

2) การแปลงฟังก์ชัน

สำหรับการแปลงในส่วนของฟังก์ชันนั้นฟังก์ชันหนึ่งฟังก์ชันจะถูกแปลงให้เป็นเค้าร่างหนึ่งเค้าร่าง โดยในขั้นตอนการแปลงเราจะนำเอาต้นไม้วิภัชที่ได้จากขั้นตอนที่ 3 มาทำการท่อง (Traverse) ไปในแต่ละโหนดซึ่งขั้นตอนวิธีการแปลงโปรแกรมภาษาซีเป็นข้อกำหนดเซตคือ

กำหนดให้

FnName เป็นชื่อของฟังก์ชัน

$V_1, V_2, V_3, \ldots, V_n$ เป็นตัวแปรที่ถูกประกาศในฟังก์ชัน FnName

Stm เป็นข้อความสั่งในฟังก์ชัน FnName

$P_1, P_2, P_3, \ldots, P_n$ เป็นพารามิเตอร์ของของฟังก์ชัน FnName

Fn_Type เป็นชนิดของข้อมูลที่ส่งคืนจากฟังก์ชัน FnName

ในการแปลงมีขั้นตอนการแปลงดังนี้

- ให้ FnName เป็นชื่อของเค้าร่าง
- แปลง $P_1, P_2, P_3, \ldots, P_n$ เป็นตัวแปรเข้าของเค้าร่างโดยการใส่เครื่องหมาย "?" ตามหลังชื่อตัวแปร
- แปลง $V_1, V_2, V_3, \ldots, V_n$ เป็นตัวแปรของเค้าร่าง
- ใช้กฎที่ได้กำหนดไว้ในหัวข้อ 3.1.1 ทำการแปลง Stm ให้อยู่ในรูปภาคแสดงแล้วนำมาใส่ในส่วนสัจพจน์ของเค้าร่าง
- ในกรณีที่ฟังก์ชัน FnName มีการส่งค่าคืนกลับให้ทำการสร้างตัวแปรชื่อ "Result" ให้เป็นตัวแปรออกของเค้าร่างโดยการเติมเครื่องหมาย "!" ต่อท้ายชื่อตัวแปร โดยให้มีชนิดข้อมูลเป็น Fn_Type

พิจารณาตัวอย่างโปรแกรมต้นฉบับภาษาซีในรูปที่ 3.6

```
int findMax(int,int);
void main() {
  int x,y,Max,Min;
  x = 5;y = 10;

  Max = findMax(x,y);
}
int findMax(int a,int b) {
  if (a > b)
    return a;
  else
    return b;
}
```

รูปที่ 3.6 ตัวอย่างโปรแกรมต้นฉบับภาษาซี

จากโปรแกรมต้นฉบับภาษาซีในรูปที่ 3.6 สามารถที่จะทำการแปลงให้เป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซตได้ในรูปที่ 3.7

findMax

$a?,b?,result! : \mathbb{Z}$

$(a? > b?) \land (result! = a?)) \lor$

$(\neg(a? > b?) \land (result! = b?))$

main

$x,y,Max,Min : \mathbb{Z}$

findMax

$x = 5 \land y = 10$

$findMax.a? = x \land findMax.b? = y$

$Max = findMax.result!$

รูปที่ 3.7 ตัวอย่างผลลัพธ์ข้อกำหนดรูปนัยในรูปสัญกรณ์เซต

บทที่ 4

การพัฒนาเครื่องมือซอฟต์แวร์แปลงโปรแกรมภาษาซีเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด

สำหรับเครื่องมือที่ทำการพัฒนาขึ้นนั้นจะแบ่งการทำงานออกเป็น 3 ส่วนหลัก ๆ ด้วยกันคือ ส่วนรับข้อมูลเข้า ส่วนสร้างข้อกำหนด และส่วนบันทึกผลลัพธ์

4.1 ส่วนรับข้อมูลเข้า

ส่วนรับข้อมูลเข้าเป็นส่วนเริ่มต้นการทำงานโดย การรับชื่อของแฟ้มข้อมูลซึ่งจะใช้ในการทำงาน โดยลักษณะของข้อมูลนำเข้าจะอยู่ในรูปของแฟ้มข้อความ (Text File) รูปแบบแฟ้มข้อมูลจะเป็นไปตามมาตรฐาน ANSI C และต้องถูกต้องตามวากยสัมพันธ์ของภาษาซีโดยแฟ้มข้อมูลจะมีนามสกุลเป็น .C และโปรแกรมทั้งหมดจะต้องถูกรวมอยู่ภายในไฟล์เพียงไฟล์เดียวเท่านั้น โดยรูปที่ 4.1 เป็นตัวอย่างแฟ้มโปรแกรมต้นฉบับภาษาซี

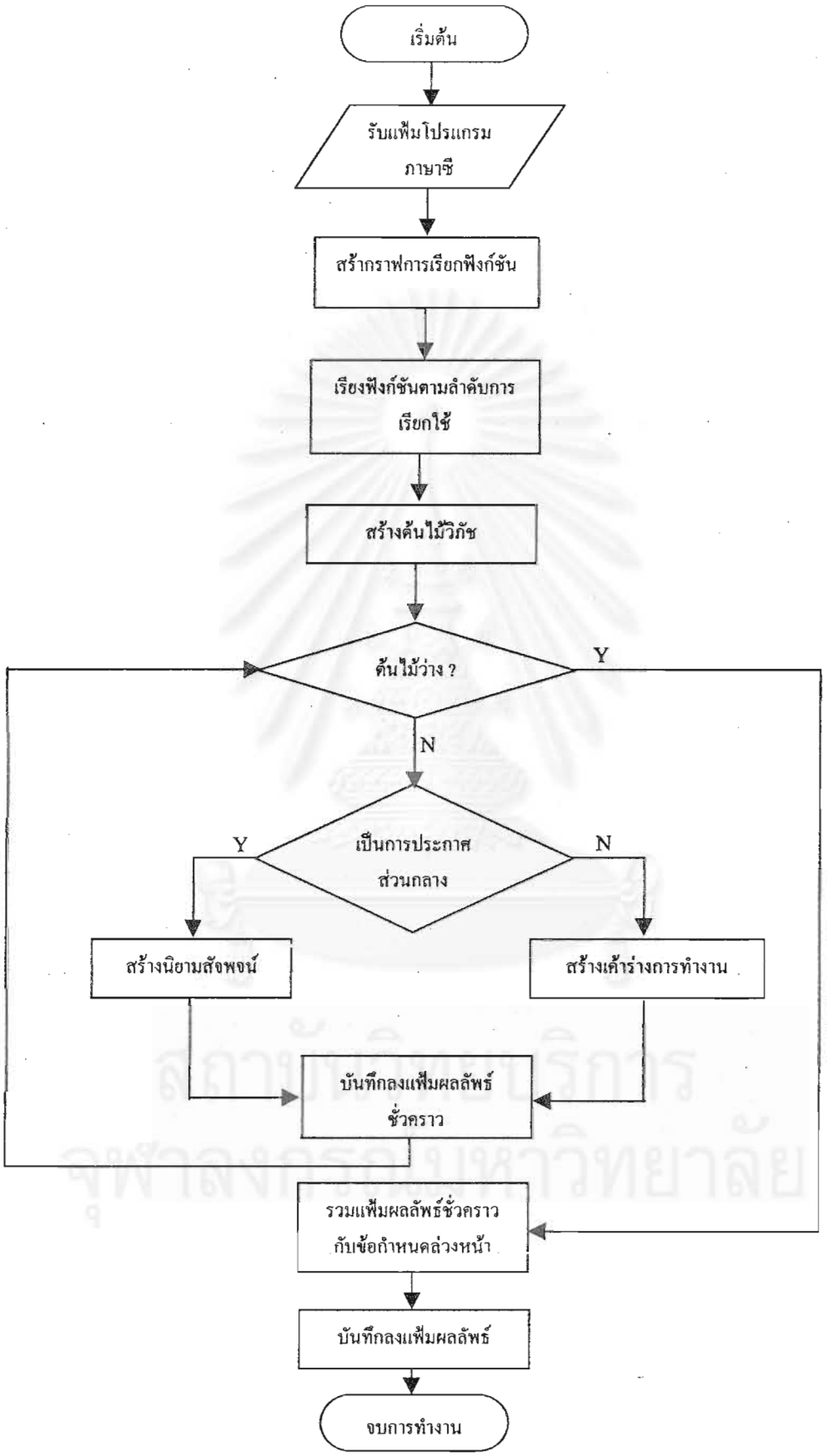
```
int findMax(int,int);
int main() {
  int x,y,Max,Min;
  x = 5;y = 10;

  Max = findMax(x,y);
}
int findMax(int a,int b) {
  if (a > b)
    return a;
  else
    return b;
}
```

รูปที่ 4.1ตัวอย่างแฟ้มต้นฉบับภาษาซี

4.2 ส่วนสร้างข้อกำหนด

ส่วนสร้างข้อกำหนดเป็นส่วนที่ทำหน้าที่ในการนำเอาข้อมูลเข้าที่ได้จากส่วนรับข้อมูลเข้ามาทำการสร้างเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด ซึ่งรูปที่ 4.2 เป็นแผนภาพแสดงขั้นตอนการทำงานของส่วนการสร้างข้อกำหนด โดยแบ่งออกเป็น

1. รับชื่อแฟ้มโปรแกรมต้นฉบับภาษาซี
2. อ่านข้อมูลจากแฟ้มโปรแกรมต้นฉบับภาษาซี



รูปที่ 4.2 ขั้นตอนการทำงานส่วนการสร้างข้อกำหนด

3. ทำการสร้างกราฟการเรียกใช้ฟังก์ชัน (Function Call Graph)
4. เรียงฟังก์ชันโดยอาศัยกราฟการเรียกใช้ฟังก์ชัน โดยฟังก์ชันที่ไม่มีการเรียกใช้ฟังก์ชันอื่นจะถูกนำมาไว้ในส่วนบนของโปรแกรมต้นฉบับ
5. ทำการประมวลผลก่อน (Preprocess) การแปลงโดยจะทำการเปลี่ยนรูปข้อความสั่งให้อยู่ในรูปแบบที่เหมาะสม เช่นการเปลี่ยนข้อความสั่งกำหนดค่าที่ถูกลดรูปให้อยู่ในรูปแบบเต็ม หรือการเปลี่ยนข้อความสั่ง switch ให้อยู่ในรูปของข้อความสั่ง if เป็นต้น
6. ทำการสร้างต้นไม้วิภัช
7. ทำการแปลงต้นไม้วิภัชที่ได้ให้เป็นข้อกำหนดเซต โดยต้องทำการตรวจสอบประเภทของโหนดว่าเป็นโหนดของข้อความสั่งประเภทไหนเพื่อที่จะทำการเลือกกฎที่จะนำมาใช้ในการแปลง เช่นถ้าเป็นโหนดที่เกิดจากข้อความสั่งกำหนดค่าก็จะใช้กฎสำหรับข้อความสั่งกำหนดค่า เป็นต้น
8. ทำการรวมข้อกำหนดที่ได้จากขั้นตอนที่ 7 เข้ากับข้อกำหนดล่วงหน้าซึ่งเป็นข้อกำหนดที่ได้ทำการกำหนดชนิดข้อมูลและตัวดำเนินการที่จำเป็นต้องใช้ ซึ่งได้กล่าวรายละเอียดไว้ในบทที่ 3
9. ทำการบันทึกผลลัพธ์ที่ได้ลงในแฟ้มข้อมูลผลลัพธ์

4.3 ส่วนบันทึกผลลัพธ์

ส่วนบันทึกข้อมูลเป็นส่วนที่ทำการบันทึกผลลัพธ์ข้อกำหนดเซตที่ได้จากส่วนการสร้างข้อกำหนดลงในแฟ้มข้อมูลที่ต้องการ โดยแฟ้มผลลัพธ์จะอยู่ในรูปแบบแฟ้มตัวอักษรโดยภายในจะประกอบด้วยข้อกำหนดเซตซึ่งถูกเขียนในรูปแท็กของลาเท็กซ์ของ Z/EVES ซึ่งรายละเอียดของแท็กทั้งหมดจะแสดงอยู่ในภาคผนวก ข โดยแฟ้มข้อมูลผลลัพธ์ที่ได้จะมีนามสกุลเป็น .ZED

จากตัวอย่างโปรแกรมต้นฉบับในรูปที่ 4.1 เมื่อทำการแปลงแฟ้มผลลัพธ์ที่จะได้จากการแปลงจะประกอบด้วยข้อกำหนดซึ่งอยู่ในรูปแท็กของลาเท็กซ์ดังแสดงในรูปที่ 4.3

ผลลัพธ์ที่แสดงอยู่ในรูปที่ 4.3 อยู่ในรูปของแท็กของลาเท็กซ์ซึ่งสามารถแสดงอยู่ในรูปของสัญกรณ์เซตได้ดังแสดงในรูปที่ 4.4

```
\begin{schema}{findMax}
findMax\_0\_a? : \num      \\
findMax\_0\_b? : \num      \\
findMax\_result! : \num     \\
findMax\_0\_b_0 : \num \\
findMax\_0\_a_0 : \num \\
```

รูปที่ 4.3 ผลลัพธ์ที่ได้จากการแปลงตัวอย่างโปรแกรมต้นฉบับภาษาซี

```
\where
findMax\_0\_a_0 = findMax\_0\_a? \also
 findMax\_0\_b_0 = findMax\_0\_b? \also
(findMax\_0\_a_0 > findMax\_0\_b_0)        \land
findMax\_result! = findMax\_0\_a_0 \lor
(\lnot (findMax\_0\_a_0 > findMax\_0\_b_0) )          \land
findMax\_result! = findMax\_0\_b_0
\end{schema}
\begin{schema}{main}
main\_result! : \num          \\
main\_0\_x, main\_0\_y, main\_0\_Max, main\_0\_Min : \num \\
findMax             \\
main\_0\_Min_0 : \num \\
main\_0\_Max_0 : \num \\
main\_0\_y_0, main\_0\_y_1 : \num \\
main\_0\_x_0, main\_0\_x_1 : \num \\
\where
main\_0\_x_0 = main\_0\_x          \also
main\_0\_y_0 = main\_0\_y          \also
main\_0\_Max_0 = main\_0\_Max  \also
main\_0\_Min_0 = main\_0\_Min   \also
main\_0\_x_1 = 5 \also
main\_0\_y_1 = 10          \also
findMax\_0\_a? = main\_0\_x_1                \also
findMax\_0\_b? = main\_0\_y_1                \also
main\_0\_Max_0 = findMax\_result!
\end{schema}
```

รูปที่ 4.3 ผลลัพธ์ที่ได้จากการแปลงตัวอย่างโปรแกรมต้นฉบับภาษาซี (ต่อ)

findMax

$findMax\_0\_a? : \mathbb{Z}$
$findMax\_0\_b? : \mathbb{Z}$
$findMax\_result! : \mathbb{Z}$
$findMax\_0\_b_0 : \mathbb{Z}$
$findMax\_0\_a_0 : \mathbb{Z}$

$findMax\_0\_a_0 = findMax\_0\_a?$
$findMax\_0\_b_0 = findMax\_0\_b?$
$findMax\_0\_a_0 > findMax\_0\_b_0 \land findMax\_result! = findMax\_0\_a_0$
$\lor \neg\, findMax\_0\_a_0 > findMax\_0\_b_0 \land findMax\_result! = findMax\_0\_b_0$

main

$main\_result! : \mathbb{Z}$
$main\_0\_x, main\_0\_y, main\_0\_Max, main\_0\_Min : \mathbb{Z}$
$findMax$
$main\_0\_Min_0, main\_0\_Max_0, main\_0\_y_0, main\_0\_y_1, main\_0\_x_0, main\_0\_x_1 : \mathbb{Z}$

$main\_0\_x_0 = main\_0\_x$
$main\_0\_y_0 = main\_0\_y$
$main\_0\_Max_0 = main\_0\_Max$
$main\_0\_Min_0 = main\_0\_Min$
$main\_0\_x_1 = 5$
$main\_0\_y_1 = 10$
$findMax\_0\_a? = main\_0\_x_1$
$findMax\_0\_b? = main\_0\_y_1$
$main\_0\_Max_0 = findMax\_result!$

รูปที่ 4.4 ผลลัพธ์ที่ได้จากการแปลงตัวอย่างโปรแกรมต้นฉบับภาษาซีในรูปสัญกรณ์เซด

4.4 ผังโครงสร้างของเครื่องมือ

จากที่ได้กล่าวมาเราจะเห็นถึงภาพรวมของเครื่องมือที่ได้ทำการพัฒนาขึ้น ตั้งแต่เริ่มรับโปรแกรมต้นฉบับจนกระทั่งได้ข้อกำหนดเซดออกมา โดยโครงสร้างของเครื่องมือที่ได้ทำการพัฒนาขึ้นนั้นแสดงอยู่ในรูปที่ 4.5 ซึ่งประกอบด้วย

โมดูล 1 รับแฟ้มโปรแกรมภาษาซี

โมดูลรับโปรแกรมต้นฉบับต้นฉบับภาษาซีเป็นโมดูลที่ทำหน้าที่ในการรับเอาข้อมูลจากผู้ใช้ซึ่งผู้ใช้ก็จะระบุชื่อแฟ้มโปรแกรมต้นฉบับภาษาซี

โมดูล 2 อ่านข้อมูล

โมดูลอ่านข้อมูลเป็นโมดูลที่ทำหน้าที่ในการอ่านแฟ้มข้อมูลที่ผู้ใช้ระบุเข้ามาในระบบ เพื่อที่จะนำไปใช้งานต่อไป

โมดูล 3 เรียงลำดับฟังก์ชัน

โมดูลเรียงลำดับฟังก์ชันเป็นโมดูลที่ทำหน้าที่ในการเรียงฟังก์ชันต่าง ๆ ในแฟ้มโปรแกรมต้นฉบับก่อนที่จะทำการแปลงซึ่งจะทำการจัดเรียงตามลำดับการเรียกใช้งาน โดยฟังก์ชันที่ไม่มีการเรียกใช้งานจะถูกนำมาไว้ในส่วนต้นของแฟ้ม

โมดูล 3.1 สร้างกราฟการเรียกฟังก์ชัน

โมดูลสร้างกราฟการเรียกใช้ฟังก์ชันเป็นโมดูลที่ทำหน้าที่ในการสร้างกราฟการเรียกใช้ฟังก์ชันซึ่งจะเป็นกราฟที่จะถูกนำไปใช้ในการเรียงฟังก์ชัน

โมดูล 4 ตัวประมวลก่อน

โมดูลตัวประมวลผลก่อนเป็นโมดูลที่ทำหน้าที่ในการเตรียมแฟ้มโปรแกรมต้นฉบับภาษาซีให้อยู่ในรูปแบบที่พร้อมจะนำไปทำการแปลงต่อ

โมดูล 4.1 แปลงข้อความสั่งกำหนดค่าแบบลดรูป

โมดูลแปลงข้อความสั่งกำหนดค่าแบบลดรูปเป็นโมดูลที่ทำหน้าที่ในการแปลงข้อความสั่งกำหนดค่าที่เขียนในแบบลดรูปให้อยู่ในรูปแบบเต็ม

โมดูล 4.2 แปลงข้อความสั่งวนซ้ำ

โมดูลแปลงข้อความสั่งวนซ้ำ เป็นโมดูลที่ทำหน้าที่ในการแปลงข้อความสั่ง do while และข้อความสั่ง for ให้อยู่ในรูปแบบของข้อความสั่ง while

โมดูล 4.3 แปลง switch เป็นข้อความสั่ง if

โมดูลแปลงข้อความสั่ง switch เป็นข้อความสั่ง if เป็นโมดูลที่ทำหน้าที่ในการแปลงข้อความสั่ง switch ให้อยู่ในรูปแบบของ nested-if เพื่อที่จะทำการแปลงเช่นเดียวกับการแปลงข้อความสั่ง if

โมดูล 5 สร้างต้นไม้วิภัช

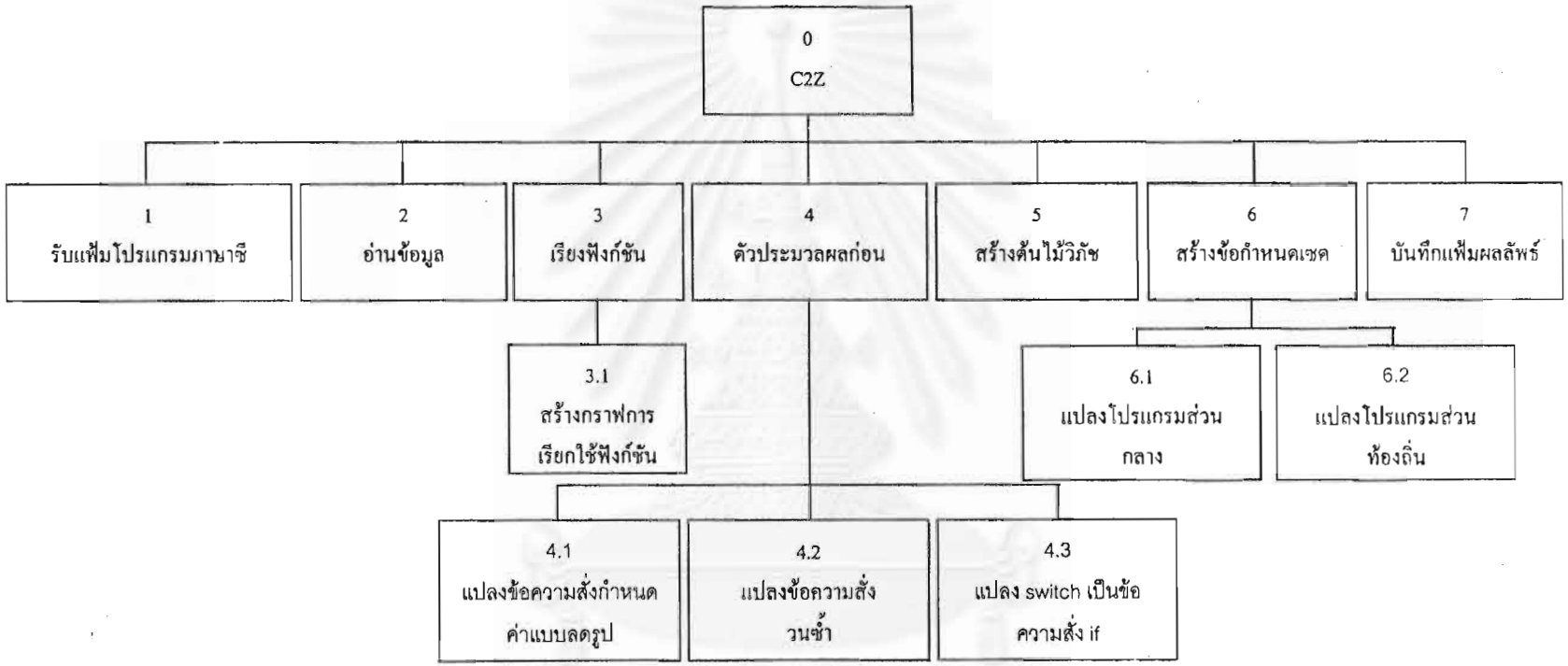
โมดูลสร้างต้นไม้วิภัชเป็นโมดูลที่ทำหน้าที่ในการนำเอาแฟ้มโปรแกรมต้นฉบับภาษาซีมาทำการสร้างเป็นต้นไม้วิภัชตามวากยสัมพันธ์ของภาษาโปรแกรมซี

โมดูล 6 สร้างข้อกำหนดเซต

โมดูลสร้างข้อกำหนดเซตเป็นโมดูลที่ทำหน้าที่ในการสร้างข้อกำหนดเซตโดยจะนำเอาต้นไม้วิภัชมาทำการท่องไปในแต่ละโหนดแล้วทำการแปลงให้เป็นภาคแสดง

โมดูล 6.1 แปลงโปรแกรมส่วนกลาง

โมดูลแปลงโปรแกรมส่วนกลางเป็นโมดูลที่ทำหน้าที่ในการแปลงโปรแกรมที่ถูกประกาศให้สามารถอ้างถึงได้ตลอดการทำงานของโปรแกรมโดยในการแปลงจะทำการสร้างเป็นนิยามสัจพจน์เพื่อจะสามารถอ้างถึงได้ทุกจุดในข้อกำหนด



รูปที่ 4.5 ผังโครงสร้างของเครื่องมือ

โมดูล 6.2 แปลงโปรแกรมส่วนท้องถิ่น

โมดูลแปลงโปรแกรมส่วนท้องถิ่นเป็นโมดูลที่ทำหน้าที่ในการแปลงฟังก์ชันการทำงานต่าง ๆ ที่ถูกประกาศในส่วนการทำงานของโปรแกรมโดยฟังก์ชันหนึ่ง ๆ จะถูกแปลงให้ได้ออกมาเป็นเค้าร่างในข้อกำหนดหนึ่งเค้าร่าง

โมดูล 6.2.1 รับตัวยืนยงวนซ้ำ

โมดูลรับตัวยืนยงวนซ้ำเป็นโมดูลที่ทำหน้าที่รับตัวยืนยงวนซ้ำจากผู้ใช้ในกรณีที่เป็นการแปลงข้อความสั่งวนซ้ำเพื่อนำมาใช้ในการแปลงโดยตัวยืนยงวนซ้ำจะเขียนในรูปของแท็คของลาเท็กซ์

โมดูล 7 บันทึกแฟ้มผลลัพธ์

โมดูลบันทึกแฟ้มผลลัพธ์เป็นโมดูลที่ทำหน้าที่ในการบันทึกผลลัพธ์ที่ได้จากการแปลงลงเก็บในแฟ้มข้อมูล

4.5 สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือซอฟต์แวร์

เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนามีรายละเอียดดังนี้

- คอมพิวเตอร์แบบพีซี PentiumIII 350 เมกะเฮิร์ส
- หน่วยความจำ 64 เมกะไบต์
- ฮาร์ดดิสต์ 6 กิกะไบต์

ซอฟต์แวร์ที่ใช้ในการพัฒนามีรายละเอียดดังนี้

- ระบบปฏิบัติการวินโดว์ส 98
- ภาษาโปรแกรมวิชวลซี รุ่น 6.0

บทที่ 5
การทดสอบโปรแกรมและตรวจสอบผลลัพธ์

5.1 ขั้นตอนการติดตั้ง

ทำการติดตั้งเครื่องมือนี้โดยการเรียก SETUP.EXE แล้วปฏิบัติตามขั้นตอนที่โปรแกรมแนะนำเพื่อเป็นการติดตั้งเครื่องมือ

5.2 สภาวะที่ใช้ทดสอบโปรแกรม

เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบมีรายละเอียดดังนี้

- คอมพิวเตอร์แบบพีซี Pentium III 350 เมกะเฮิร์ส
- หน่วยความจำ 64 เมกะไบต์
- ฮาร์ดดิสต์ 6 กิกะไบต์

5.3 กรณีทดสอบที่ใช้ทดสอบโปรแกรม

สำหรับการทดสอบงานวิจัยนี้มีวัตถุประสงค์หลักในการที่จะตรวจสอบว่ากฎที่ได้ทำการกำหนดขึ้นนั้นมีความถูกต้องหรือไม่มิได้เน้นที่จะตรวจสอบประสิทธิภาพของเครื่องมือ ดังนั้นกรณีทดสอบที่เลือกมาทำการทดสอบจึงมีขนาดที่ไม่ใหญ่แต่ให้ครอบคลุมทุก ๆ กฎที่ได้ทำการกำหนดขึ้นซึ่งประกอบด้วยกรณีทดสอบทั้งหมด 9 กรณีทดสอบได้แก่

<u>กรณีทดสอบที่ 1</u>

กฎที่ทำการทดสอบ

- กฎการแปลงตัวแปรที่มีชนิดข้อมูลเป็น int, short และ long
- กฎการแปลงข้อความสั่งแบบเต็มรูป
- กฎการแปลงข้อความสั่งเลือกทางเดินแบบสองทางเลือก
- กฎการแปลงข้อความสั่งตามลำดับ
- กฎการแปลงข้อความสั่งเรียกใช้ฟังก์ชัน

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน findMin เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะทำการหาค่าจำนวนเต็มที่น้อยกว่าส่งคืนกลับมา
- ฟังก์ชัน findMax เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะทำการหาค่าจำนวนเต็มที่มากกว่าส่งคืนกลับมา

- ฟังก์ชัน main เป็นฟังก์ชันหลักที่จะทำการเรียกใช้งานฟังก์ชัน findMin และฟังก์ชัน findMax

กรณีทดสอบที่ 2

กฎที่ทำการทดสอบ

- กฎการแปลงการประกาศส่วนกลาง
- กฎการแปลงข้อความสั่งเลือกทางเดินแบบหนึ่งทางเลือก
- กฎการแปลงข้อความสั่งเลือกทางเดินแบบสองทางเลือก
- กฎการแปลงข้อความสั่งตามลำดับ
- กฎการแปลงข้อความสั่งเรียกใช้ฟังก์ชัน

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน findMax เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะทำการหาค่าจำนวนเต็มที่มากกว่าส่งคืนกลับมา
- ฟังก์ชัน successor เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะคำนวณค่าตัวตามหลัง (Successor) ของจำนวนเต็มที่ส่งเข้ามา
- ฟังก์ชัน main เป็นฟังก์ชันหลักที่จะทำการเรียกใช้งานฟังก์ชัน findMax และฟังก์ชัน successor

กรณีทดสอบที่ 3

กฎที่ทำการทดสอบ

- กฎการแปลงข้อความสั่งกำหนดค่าแบบลดรูป
- กฎการแปลงข้อความสั่งเลือกทางเดินแบบ nested-if
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน Test เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะทำการหาค่าจำนวนเต็มที่ขึ้นอยู่กับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมา

กรณีทดสอบที่ 4

กฎที่ทำการทดสอบ

- กฎการแปลงตัวแปรที่มีชนิดข้อมูล unsigned และ char
- กฎการแปลงข้อความสั่งกำหนดค่าแบบเต็มรูป
- กฎการแปลงข้อความสั่งกำหนดค่าแบบลดรูป

- กฎการแปลงข้อความสั่ง switch แบบมี break ทุกทางเลือก
- กฎการแปลงข้อความสั่ง switch แบบไม่มี break ทุกทางเลือก
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน main เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นตัวอักขระ ฟังก์ชันจะทำการหาค่าจำนวนเต็มที่เท่ากับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมาโดยภายในฟังก์ชันจะมีการใช้งานข้อความสั่ง switch

<u>กรณีทดสอบที่ 5</u>

กฎที่ทำการทดสอบ

- กฎการแปลงข้อความสั่งกำหนดค่าแบบเต็มรูป
- กฎการแปลงข้อความสั่งกำหนดค่าแบบลดรูป
- กฎการแปลงข้อความสั่ง while
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน multi เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะนำเอาค่าจำนวนเต็มทั้งสองค่ามาคูณกันแล้วส่งคืนค่าผลคูณซึ่งเป็นจำนวนเต็มกลับมา

<u>กรณีทดสอบที่ 6</u>

กฎที่ทำการทดสอบ

- กฎการแปลงข้อความสั่งกำหนดค่าแบบเต็มรูป
- กฎการแปลงข้อความสั่งกำหนดค่าแบบลดรูป
- กฎการแปลงข้อความสั่ง for
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน sum เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะทำการหาค่าผลรวมตั้งแต่ 1 จนถึงค่าของพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมา

<u>กรณีทดสอบที่ 7</u>

กฎที่ทำการทดสอบ

- กฎการแปลงข้อความสั่งกำหนดค่าแบบเต็มรูป

- กฎการแปลงข้อความสั่ง do while
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน Copy เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะใช้ข้อความสั่งวนซ้ำ do while ทำการหาค่าจำนวนเต็มที่ขึ้นเท่ากับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มนั้นกลับมา

กรณีทดสอบที่ 8

กฎที่ทำการทดสอบ

- กฎการแปลงตัวแปรที่มีชนิดข้อมูลเป็น float และ double
- กฎการแปลงข้อความสั่งกำหนดค่าแบบเต็มรูป
- กฎการแปลงข้อความสั่ง while
- กฎการแปลงข้อความสั่งตามลำดับ

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชัน main เป็นฟังก์ชันที่ไม่มีการรับพารามิเตอร์จะทำการคำนวณค่าจำนวนจริงและการทำงานร่วมกันระหว่างจำนวนจริงและจำนวนเต็ม

กรณีทดสอบที่ 9

กฎที่ทำการทดสอบ

- ทำการทดสอบทุกกฎที่ได้ทำการกำหนดขึ้น

ภายในโปรแกรมประกอบด้วย

- ฟังก์ชันทั้งหมด 10 ฟังก์ชันได้แก่ ฟังก์ชัน main, compare, swap, div, multi, copy, IFTest, successor และ SwitchTest โดยรายละเอียดทั้งหมดแสดงอยู่ในภาคผนวก ค ซึ่งจะครอบคลุมกฎที่ได้ทำการกำหนดขึ้นภายในงานวิจัยนี้

โดยกรณีทดสอบทั้งหมดจะทำการตรวจสอบกฎต่อไปนี้

1) กฎการแปลงส่วนการประกาศ
   1.1) ตัวแปรที่มีชนิดข้อมูลเป็น int
   1.2) ตัวแปรที่มีชนิดข้อมูลเป็น short int
   1.3) ตัวแปรที่มีชนิดข้อมูลเป็น long int
   1.4) ตัวแปรที่มีชนิดข้อมูลเป็น float
   1.5) ตัวแปรที่มีชนิดข้อมูลเป็น double
   1.6) ตัวแปรที่มีชนิดข้อมูลเป็น unsigned
   1.7) ตัวแปรที่มีชนิดข้อมูลเป็น char

1.8) โปรแกรมที่มีการประกาศส่วนกลาง

2) กฎการแปลงส่วนการทำงาน

2.1) ข้อความสั่งกำหนดค่าแบบเต็มรูป

2.2) ข้อคำสั่งกำหนดค่าแบบลดรูป

2.3) ข้อความสั่งเลือกทางเดินแบบหนึ่งทางเลือก

2.4) ข้อความสั่งเลือกทางเดินแบบสองทางเลือก

2.5) ข้อความสั่งเลือกทางเดินแบบหลายทางเลือก

2.5.1) ข้อความสั่ง nested-if

2.5.2) ข้อความสั่ง switch แบบมีข้อความสั่ง break ทุกทางเลือก

2.5.3) ข้อความสั่ง switch แบบไม่มีข้อความสั่ง break ทุกทางเลือก

2.6) ข้อความสั่งเรียงลำดับ

2.7) ข้อความสั่งวนซ้ำ

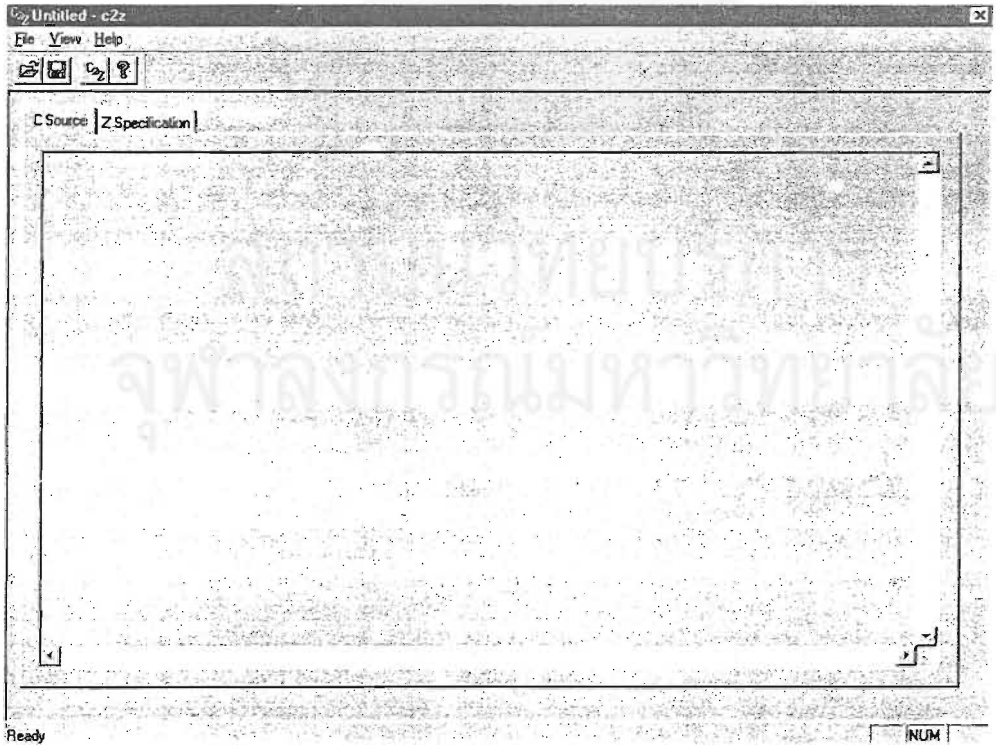
2.7.1) ข้อความสั่งวนซ้ำ while

2.7.2) ข้อความสั่งวนซ้ำ do while

2.7.3) ข้อความสั่งวนซ้ำ for

2.8) ข้อความสั่งเรียกใช้ฟังก์ชัน

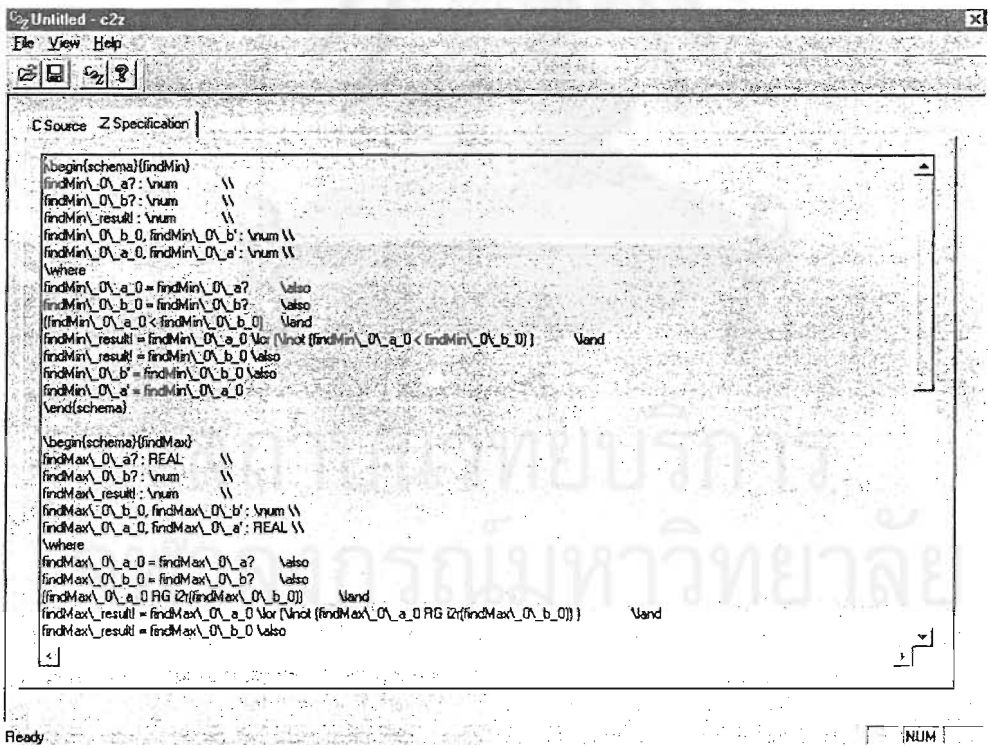
ในการทดสอบกฎทั้งหมดที่กล่าวมาจะใช้กรณีทดสอบทั้งหมด 9 กรณีทดสอบโดยรายละเอียดของกรณีทดสอบทั้งหมดแสดงอยู่ในภาคผนวก ค ซึ่งจะครอบคลุมกฎที่ต้องการทดสอบโดยที่หนึ่งกรณีทดสอบอาจจะทดสอบกฎได้มากกว่าหนึ่งกฎ



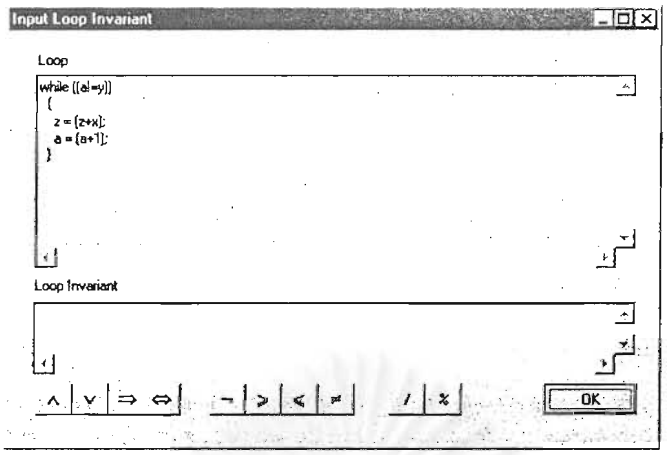
รูปที่ 5.1 หน้าจอหลักของโปรแกรม C2Z.EXE

5.4 ขั้นตอนการทดสอบ

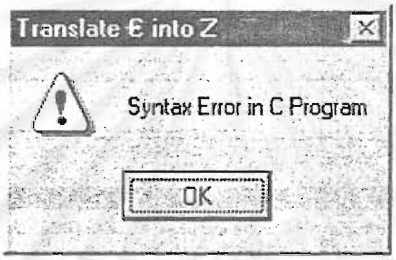
1) เรียกโปรแกรม C2Z.EXE เพื่อทำการทดสอบ ดังรูปที่ 5.1
2) ทำการเปิดแฟ้มโปรแกรมต้นฉบับภาษาซีโดยเลือกคำสั่ง Open ในเมนู File หรือกดปุ่ม  บนแถบเครื่องมือ ในกรณีที่สามารถเปิดแฟ้มได้จะแสดงแฟ้มโปรแกรมต้นฉบับบนหน้าจอ
3) เมื่อต้องการทำการแปลงเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซดให้ทำการเลือกคำสั่ง Translate จากเมนู File หรือกดปุ่ม  บนแถบเครื่องมือ
   - หากแฟ้มโปรแกรมต้นฉบับถูกต้องตามวากยสัมพันธ์ของภาษาโปรแกรมซีตามมาตรฐาน ANSI C โปรแกรมจะทำการสร้างข้อกำหนดเซดแสดงขึ้นบนหน้าจอดังแสดงในรูปที่ 5.2 แต่ในกรณีที่โปรแกรมต้นฉบับมีข้อความสั่งวนซ้ำโปรแกรมจะทำการเปิดหน้าจอในรูปที่ 5.3 ให้ผู้ใช้ระบุตัวบ่งชี้วนซ้ำ
   - หากโปรแกรมต้นฉบับผิดหลักวากยสัมพันธ์ของภาษาโปรแกรมซีโปรแกรมจะทำการแจ้งข้อความเตือนบนหน้าจอดังแสดงในรูปที่ 5.4
4) ผลลัพธ์ที่ได้สามารถบันทึกลงในแฟ้มข้อมูลได้โดยการเลือกคำสั่ง Save จากเมนู File หรือกดปุ่ม  จากแถบเครื่องมือ



รูปที่ 5.2 หน้าจอแสดงผลลัพธ์ของโปรแกรม C2Z.EXE



รูปที่ 5.3 หน้าจอสำหรับระบุตัวยืนยงวนซ้ำเมื่อทำการแปลงข้อความสั่งวนซ้ำ



รูปที่ 5.4 หน้าจอแสดงข้อความเตือนเมื่อโปรแกรมต้นฉบับผิดวากยสัมพันธ์

5.3 ผลการทดสอบโปรแกรม

จากการทดสอบพบว่าเครื่องมือนี้สามารถทำการแปลงโปรแกรมต้นฉบับภาษาซีให้อยู่ในรูปของข้อกำหนดรูปนัยในรูปสัญกรณ์เซดได้ โดยผลลัพธ์ที่ได้นั้นจะอยู่ในรูปของแท็คของลาท็กซ์ยึดตามมาตรฐานของ Z/EVES จากการทดสอบกับกรณีทดสอบทั้งหมดที่ได้กล่าวมาพบว่าสามารถทำการสร้างเป็นข้อกำหนดเซดได้โดยแต่ละฟังก์ชันจะถูกแปลงให้อยู่ในรูปของเค้าร่างหนึ่งเค้าร่างเสมอ ส่วนกรณีที่มีการประกาศส่วนกลางจะมีการสร้างนิยามสัจพจน์เพิ่มขึ้นจากเค้าร่าง โดยรายละเอียดของผลลัพธ์ที่ได้จากการแปลงนั้นแสดงอยู่ในภาคผนวก ค

5.3 ผลการตรวจสอบผลลัพธ์

ในการตรวจสอบผลลัพธ์ที่ได้จากการแปลงนั้นอาศัยเครื่องมือ Z/EVES มาช่วยในขั้นตอนนี้ โดยขั้นตอนการตรวจสอบแบ่งออกเป็น 3 ขั้นตอนด้วยกันคือ

1) การตรวจสอบวากยสัมพันธ์ สำหรับการตรวจสอบนี้จะทำการตรวจสอบว่าข้อกำหนดเซดที่ได้นั้นถูกต้องตามวากยสัมพันธ์ของสัญกรณ์เซดหรือไม่ซึ่งในกรณีที่ทำการทดสอบสำเร็จ Z/EVES จะทำการสร้างเป็นทฤษฎีของข้อกำหนดให้แต่ในกรณีที่ไม่ถูกต้องจะมีข้อความแสดงความผิดผลาดที่เกิดขึ้น

2) การตรวจสอบทางทฤษฎี เนื่องจากข้อกำหนดรูปนัยถูกเขียนอยู่ในรูปของสัญกรณ์ทางคณิตศาสตร์ดังนั้นในการตรวจสอบขั้นนี้จะอาศัยคำสั่ง prove by reduce; ของ Z/EVES เพื่อใช้ทฤษฎีทางคณิตศาสตร์เข้ามาทำการพิสูจน์เพื่อหาความขัดแย้งภายในข้อกำหนดซึ่งในกรณีที่ข้อกำหนดไม่มีความผิดพลาด Z/EVES จะทำการแสดงภาคแสดงสุดท้ายที่ถูกลดรูปลงแล้ว แต่กรณีที่มีความขัดแย้งเกิดขึ้นในข้อกำหนด Z/EVES จะทำการแจ้งผลลัพธ์เป็น false

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับการตรวจสอบขั้นนี้จะเป็นการตรวจสอบโดยการระบุค่าของตัวแปรภายในข้อกำหนดเพื่อดูว่าข้อกำหนดนั้นสามารถที่จะทำการคำนวณผลลัพธ์ได้ตรงกับโปรแกรมต้นฉบับภาษาซีหรือไม่ซึ่งจะใช้คำสั่งเดียวกับการทดสอบในขั้นตอนการตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ เพียงแต่ในขั้นตอนนี้เราจะเพิ่มคำสั่งที่จะกำหนดค่าให้กับตัวแปร

เพื่อความสะดวกในการอธิบายผลลัพธ์การตรวจสอบจะขออ้างถึงผลลัพธ์การตรวจสอบของกรณีทดสอบที่ 1 ในภาคผนวก ค มาประกอบการอธิบาย อย่างที่ได้กล่าวไว้ข้างต้นว่าการตรวจสอบจะกระทำใน 3 วิธีคือการตรวจสอบวากยสัมพันธ์ การตรวจสอบทางทฤษฎี และการตรวจสอบด้วยตัวอย่าง ซึ่งจากการที่นำเอาผลลัพธ์ที่ได้จากการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 1 มาทำการตรวจสอบผลที่ได้จากการตรวจสอบวากยสัมพันธ์แสดงอยู่ในรูปที่ ค-3

จากรูปที่ ค-3 จะเห็นว่าเครื่องมือ Z/EVES สามารถทำการสร้างทฤษฎีบทของข้อกำหนดขึ้นมาได้โดยไม่มีการแสดงข้อความแจ้งความผิดพลาดซึ่งก็หมายถึงข้อกำหนดที่ได้ถูกต้องตามวากยสัมพันธ์ของสัญกรณ์เซด

หลังจากนั้นเป็นขั้นตอนการตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์มาทำการตรวจสอบความขัดแย้งของข้อกำหนดผลลัพธ์ของกรณีทดสอบที่ 1 จะพบว่า Z/EVES สามารถทำการสร้างภาคแสดงใหม่จากข้อกำหนดได้โดยที่ไม่ส่งผลลัพธ์ออกมาเป็น faise นั่นก็หมายความว่าข้อกำหนดที่ได้นั้นไม่มีความขัดแย้งเกิดขึ้นอยู่ภายในข้อกำหนดซึ่งผลลัพธ์ของการตรวจสอบในขั้นตอนนี้แสดงอยู่ในรูปที่ ค-4

ขั้นตอนสุดท้ายของการตรวจสอบเป็นการตรวจสอบโดยอาศัยตัวอย่างซึ่งในขั้นตอนนี้จะทำการระบุค่าของตัวแปรในข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานเทียบกับผลลัพธ์ที่ได้จากการทำงานของโปรแกรมต้นฉบับโดยเมื่อการทดสอบกับกรณีทดสอบที่ 1 โดยการระบุค่าของตัวแปรซึ่งในที่นี้คือตัวแปร x และ y ให้มีค่าเป็น 4 และ 5 ตามลำดับ ซึ่งผลลัพธ์ที่จะได้จากการทำงานของโปรแกรมต้นฉบับก็คือตัวแปร max และ min จะต้องมีค่าเป็น 5 และ 4 ตามลำดับ ดังนั้นเมื่อเปรียบเทียบผลลัพธ์ที่ได้จากการตรวจสอบข้อกำหนดจะพบว่าผลลัพธ์ที่ได้นั้นสอดคล้องกับผลลัพธ์ของการทำงานของโปรแกรมนั้นคือตัวแปร main_0_max และ main_0_min เท่ากับ 5 และ 4 ตามลำดับดังแสดงในรูปที่ 5.5 ซึ่งเป็นส่วนตัดของผลลัพธ์ที่ได้จากการตรวจสอบซึ่งสามารถดูผลลัพธ์ทั้งหมดได้จากรูปที่ ค-5

```
\land main\_0\_Min_0 = 4 \\
\land main\_0\_Min' = 4 \\
\land main\_0\_Max_0 = main\_0\_Max \\
\land main\_0\_Max_0 = 5 \\
\land main\_0\_Max' = 5 \\
```

รูปที่ 5.5 ส่วนของผลลัพธ์ที่ได้จากการตรวจสอบโดยอาศัยตัวอย่าง

จากการทดสอบทั้งสามวิธีกับกรณีทดสอบทั้ง 8 กรณีทดสอบผลลัพธ์ที่ได้นั้นถูกต้องตามที่ต้องการทั้งหมดซึ่งตาราง 5.1 เป็นสรุปผลการทดสอบทั้ง 8 กรณีทดสอบโดยจะแสดงให้เห็นว่ากฎแต่ละกฎถูกตรวจสอบด้วยกรณีทดสอบใด ส่วนรายละเอียดผลลัพธ์ของการทดสอบนั้นแสดงอยู่ในภาคผนวก ค

ตาราง 5.1 สรุปผลการทดสอบข้อกำหนดเชิงด้วยเครื่องมือ Z/EVES

กฎที่ทำการทดสอบ	กรณีทดสอบ								
	1	2	3	4	5	6	7	8	9
1. กฎการแปลงส่วนการประกาศ									
1.1. ตัวแปรที่มีชนิดข้อมูลเป็น int	√	√	√						√
1.2. ตัวแปรที่มีชนิดข้อมูลเป็น short int	√								√
1.3. ตัวแปรที่มีชนิดข้อมูลเป็น long int	√								√
1.4. ตัวแปรที่มีชนิดข้อมูลเป็น float								√	√
1.5. ตัวแปรที่มีชนิดข้อมูลเป็น double								√	√
1.6. ตัวแปรที่มีชนิดข้อมูลเป็น unsigned				√					√
1.7. ตัวแปรที่มีชนิดข้อมูลเป็น char				√					√
1.8. การประกาศส่วนกลาง		√							√
2. กฎการแปลงส่วนการทำงาน									
2.1. ข้อความสั่งกำหนดค่าแบบเต็มรูป	√			√	√	√	√	√	√
2.2. ข้อคำสั่งกำหนดค่าแบบลดรูป			√	√	√	√			√

ตาราง 5.2 สรุปผลการทดสอบข้อกำหนดเชิงด้วยเครื่องมือ Z/EVES

กฎที่ทำการทดสอบ	กรณีทดสอบ								
	1	2	3	4	5	6	7	8	9
2.3. ข้อความสั่งเลือกทางเดินแบบหนึ่งทางเลือก		√							√
2.4. ข้อความสั่งเลือกทางเดินแบบสองทางเลือก	√	√							√
2.5. ข้อความสั่งเลือกทางเดินแบบหลายทางเลือก									
2.5.1. ข้อความสั่ง nested-if			√						√
2.5.2. ข้อความสั่ง switch แบบมี break ทุกทางเลือก				√					√
2.5.3. ข้อความสั่ง switch แบบมี case ที่ทำงานเหมือนกัน				√					√
2.6. ข้อความสั่งตามลำดับ	√	√	√	√	√	√	√	√	√
2.7. ข้อความสั่งวนซ้ำ									
2.7.1. ข้อความสั่งวนซ้ำ while					√				√
2.7.2. ข้อความสั่งวนซ้ำ do while							√		√
2.7.3. ข้อความสั่งวนซ้ำ for						√			√
2.8. ข้อความสั่งเรียกใช้ฟังก์ชัน	√	√							√

บทที่ 6
สรุปผลการวิจัย

จากการศึกษาและวิจัยสามารถสรุปผลที่ได้รับจากการวิจัย ปัญหา และข้อจำกัดที่พบได้ดังนี้

6.1 สรุปผลการวิจัย

จากงานวิจัยได้ทำการออกแบบกฎสำหรับการแปลงโปรแกรมภาษาซีตามมาตรฐาน ANSI C มาเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซด โดยกฎที่ทำการกำหนดขึ้นนั้นจะแบ่งออกเป็นสองส่วนด้วยกันคือ

1) กฎสำหรับการแปลงส่วนการประกาศ ซึ่งเป็นกฎที่ใช้ในการแปลงส่วนการประกาศในโปรแกรมภาษาซี โดยในงานวิจัยนี้ได้ทำการกำหนดกฎสำหรับชนิดข้อมูลพื้นฐานในภาษาโปรแกรมซีประกอบด้วยชนิดข้อมูลทั้งหมด 7 ชนิดด้วยกันได้แก่ int, short, long, float, double, unsigned, char

2) กฎสำหรับการแปลงส่วนการทำงาน ซึ่งเป็นกฎที่ใช้ในการแปลงข้อความสั่งภายในโปรแกรมภาษาซี โดยกฎที่ทำการกำหนดขึ้นนั้นประกอบด้วยกฎทั้งหมด 5 กฎครอบคลุมข้อความสั่ง 5 ประเภทในภาษาโปรแกรมซีได้แก่

- ข้อความสั่งกำหนดค่า
- ข้อความสั่งเลือกทางเดิน
- ข้อความสั่งเรียงลำดับ
- ข้อความสั่งวนซ้ำ
- ข้อความสั่งเรียกใช้ฟังก์ชั่น

นอกจากนี้ในงานวิจัยนี้ยังได้ทำการออกแบบขั้นตอนวิธีในการแปลงโปรแกรมภาษาซีมาเป็นข้อกำหนดรูปนัย โดยขั้นตอนวิธีที่ได้ทำการออกแบบจะอาศัยกฎที่ได้ทำการกำหนดขึ้นมาเป็นกลไกในการแปลง

ในขั้นตอนของการทดสอบ ผู้วิจัยได้ทำการทดสอบกับกรณีทดสอบทั้งหมด 8 กรณีทดสอบซึ่งครอบคลุมกฎที่ได้ทำการกำหนดขึ้นทั้งหมด และได้อาศัยเครื่องมือ Z/EVES มาช่วยในขั้นตอนการทดสอบซึ่งได้ทำแบ่งการทดสอบออกเป็น 3 ขั้นตอนด้วยกัน ขั้นตอนแรกจะเป็นการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ ขั้นที่สองจะเป็นการตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์เข้ามาช่วยในการตรวจสอบอันเป็นข้อดีของการทำข้อกำหนดรูปนัย และขั้นตอนสุดท้ายจะเป็นการตรวจสอบว่าข้อกำหนดที่ได้นั้นสามารถที่จะทำงานได้เช่นเดียวกับโปรแกรมต้นฉบับหรือไม่ซึ่งจากการตรวจสอบทั้งสามขั้นตอนกับกรณีทดสอบทั้ง 8 กรณีทดสอบนั้นผลลัพธ์ที่ได้ถูกต้องไม่มีข้อผิดพลาดใด

6.2 ประโยชน์ของเครื่องมือการสร้างข้อกำหนดรูปนัยในรูปสัญกรณ์เซดจากโปรแกรมภาษาซี

1) เป็นเครื่องมือที่ช่วยในการสร้างเอกสารประกอบระบบในกรณีที่ระบบนั้นไม่มีเอกสารประกอบระบบ หรือในกรณีที่เอกสารกับตัวระบบไม่ถูกต้องตรงกัน

2) เป็นเครื่องช่วยสนับสนุนการใช้ข้อกำหนดรูปนัยโดยผู้ใช้ไม่ต้องเริ่มเขียนข้อกำหนดตั้งแต่ต้น ในกรณีที่ต้องการจะเปลี่ยนการออกแบบระบบมาใช้วิธีการรูปนัย

6.3 ปัญหาและข้อจำกัดในงานวิจัย

6.3.1) ข้อความสั่งเลือกทางเดิน switch นั้นสามารถทำการเขียนได้ในสามรูปแบบได้แก่

- แบบที่ 1 ข้อความสั่ง switch แบบมีข้อความสั่ง break ทุกกรณี
- แบบที่ 2 ข้อความสั่ง switch แบบมี case ตั้งแต่สอง case ขึ้นไปที่ทำงานเหมือนกันทุกอย่าง
- แบบที่ 3 ข้อความสั่ง switch แบบมี case ที่ไม่มีข้อความสั่ง break

ซึ่งในงานวิจัยนี้จะครอบคลุมเฉพาะสองแบบเท่านั้น ส่วนแบบที่สามจะไม่สามารถทำการแปลงได้

6.3.2) ในการแปลงข้อความสั่งวนซ้ำนั้นจะต้องทำการหาตัวยืนยงวนซ้ำโดยการพิจารณาจากตารางติดตามค่าตัวแปรแล้วทำการหาความสัมพันธ์ของตัวแปรทั้งหมดถึงจะได้ตัวยืนยงวนซ้ำของข้อความสั่งวนซ้ำนั้นออกมาซึ่งกระบวนการทั้งหมดนั้นไม่สามารถที่จะทำแบบอัตโนมัติได้ ด้วยสาเหตุนี้ทำให้เครื่องมือที่ทำการพัฒนาขึ้นนั้นจะทำการร้องขอตัวยืนยงวนซ้ำในทุก ๆ ครั้งที่มีการแปลงข้อความสั่งวนซ้ำ ซึ่งผู้ใช้จะต้องเป็นผู้ทำการตัวยืนยงวนซ้ำแล้วระบุให้กับเครื่องมือ หลังจากนั้นเครื่องมือจึงจะสามารถทำการแปลงข้อความสั่งวนซ้ำนั้นได้

6.3.3) ตัวยืนยงวนซ้ำนั้นเป็นภาคแสดงที่จะแสดงให้เห็นถึงผลสุดท้ายที่ได้จากการทำงานของข้อความสั่งวนซ้ำเท่านั้น ดังนั้นเราไม่สามารถที่จะรู้ได้ว่าระหว่างการทำงานของข้อความสั่งวนซ้ำนั้นสถานะของตัวแปรจะเป็นอย่างไร

6.3.4) กรณีที่ข้อความสั่งวนซ้ำเป็นแบบซ้อนใน (nested loop) นั้นเครื่องมือจะถามตัวยืนยงวนซ้ำสำหรับข้อความสั่งวนซ้ำชั้นนอกสุดเท่านั้นเนื่องจากตัวยืนยงวนซ้ำของข้อความสั่งวนซ้ำนอกสุดนั้นจะต้องครอบคลุมถึงตัวยืนยงวนซ้ำที่อยู่ภายในด้วย

6.3.5) ชื่อตัวแปรในโปรแกรมภาษาซีนั้นจะต้องถูกเปลี่ยนชื่อโดยการนำเอาชื่อฟังก์ชันและบล็อกที่เป็นตัวประกาศต่อที่ด้านหน้าเพื่อป้องกันการเกิดปัญหาชื่อตัวแปรซ้ำในข้อกำหนด

6.4 แนวทางในการวิจัยต่อ

งานวิจัยชิ้นนี้สามารถทำการแปลงโปรแกรมภาษาซีให้อยู่ในรูปของข้อกำหนดเซตได้ซึ่งจากจุดนี้เราสามารถทำการวิจัยเพิ่มเติมเพื่อศึกษาว่าโปรแกรมภาษาซี 2 โปรแกรมสามารถทำงานแทนกันได้หรือไม่ โดยทำการพิสูจน์ทางคณิตศาสตร์ว่าข้อกำหนดเซตของทั้งสองโปรแกรมนั้นสมมูลกัน เพื่อเป็นประโยชน์ในการทำความเข้าใจโปรแกรม (Code Comprehension) ว่าโปรแกรมนั้นทำงานอะไรและมีพฤติกรรมเหมือนกันหรือไม่ได้

6.5 ผลงานตีพิมพ์

ผลงานวิจัยนี้ได้รับคัดเลือกให้ถูกตีพิมพ์ใน Proceeding งานประชุมวิชาการ The 4th National Computer Science and Engineering Conference (NCSEC 2000) ซึ่งได้จัดขึ้นที่ศูนย์การประชุมแห่งชาติสิริกิติ์ ระหว่างวันที่ 16-17 พฤศจิกายน 2543 โดยรายละเอียดแสดงอยู่ในภาคผนวก ง

รายการอ้างอิง

[1] Gerald C. Gannod. The Application of Formal Methods to the Reverse Engineering of Imperative Program Code, M.S. Thesis. Michigan State University, April 1994.

[2] Gerald C. Gannod and Betty Cheng. Abstraction of Formal Specifications from Program Code, Proceedings of the 3rd International Conference on Tools in Artificial Intelligence, November 1991.

[3] Gerald C. Gannod and Betty Cheng. A Two-Phase Approach to Reverse Engineering using Formal Methods, Lecture Notes in Computer Science: Proceedings of the Conference on Formal Methods in Programming and Their Applications, Vol. 735, Springer-Verlag, July 1993.

[4] Gerald C. Gannod and Betty Cheng. Strongest Postcondition Semantics as the Formal Basis for Reverse Engineering, Proceedings of the 2nd Working Conference on Reverse Engineering, IEEE, July 1995.

[5] Gerald C. Gannod and Betty Cheng. Using Informal and Formal Methods for the Reverse Engineering of C Programs, Proceedings of the 1996 IEEE International Conference on Software Maintenance, IEEE, November 1996.

[6] Gerald C. Gannod and Betty Cheng. A Formal Automated Approach for Reverse Engineering Programs with Pointers, in Proceeding of IEEE Automated Software Engineering, November 1997.

[7] Carlo Ghezzi and Mehdi Jazayeri. Programming Language Concept, New York: John Wiley & Sons, 1982.

[8] Ryan Stanfifer. The Study of Programming Language, International Edition, Englewood Cliffs N.J.: Prentice Hall, 1995.

[9] Ravi Sethi. Programming Language Concept & Construct. Second Edition. United States of America: Addison-Wesley, 1996

[10] Wie Ming Lim, John V. Harrison, Paul A. Bailes, Anthony Berglas. Design Recovery Throught Formal Specification, Software Engineering Conference, November 1998.

[11] Roger S. Pressman. Software Engineering: A Practitioner Approach. Fourth Edition. International Edition. Singapore: McGraw-Hill, 1997.

[12] Andrew Harry. Formal Method Fact File VDM and Z. Chichester: John Wiley and Sons, 1996

[13] Bryan Ratcliff. Introducing Specification Using Z: A Practical Case Study Approach. London: McGraw-Hill, 1994.

[14] J.M. Spivey. The Z Notation: Reference Manual. Second Edition. Oxford: Prentice Hall International (UK), 1998

รายการอ้างอิง

[15] Ben Potter. Jane Sinclair and David Till, An Introduction to formal Specification and Z. First Edition. Cambridge: Prentice Hall, 1991.

[16] Robert W.Sebesta. Concept of Programming Languages. Third Edition. California: Addison-Wesley, 1996

[17] Michael Huth and Mark Ryan. Logic in Computer Science: Modelling and Reasoning about Systems, First Edition, Cambridge: Cambridge University Press, 2000.

ภาคผนวก

ภาคผนวก ก. ข้อกำหนดล่วงหน้า (Predefined Specification)

ในการแปลงโปรแกรมโปรแกรมภาษาซีเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซดนั้นผลลัพธ์ที่ได้จากการแปลงจะต้องถูกผนวกรวมไปกับข้อกำหนดล่วงหน้าซึ่งเป็นส่วนทำการกำหนดชนิดข้อมูลที่ไม่มีอยู่ในสัญกรณ์เซดและตัวดำเนินการ ซึ่งข้อกำหนดล่วงหน้าที่ทำการกำหนดขึ้นทั้งหมดได้แสดงอยู่ในรูปที่

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus} {infun3} {RPlus}
\syndef{RMinus} {infun3} {RMinus}
\syndef{RMul} {infun3} {RMul}
\syndef{RDiv} {infun3} {RDiv}
\syndef{RG} {inrel} {RG}
\syndef{RL} {inrel} {RL}
\syndef{RGE} {inrel} {RGE}
\syndef{RLE} {inrel} {RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
```

รูปที่ ก-1 ข้อกำหนดล่วงหน้าที่จะผนวกรวมไปกับผลลัพธ์ของการแปลง

```
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
        (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
        (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
        (second(x) = second(y) \land first(x) \geq first(y)))              \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
        (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}
```

รูปที่ ก-1 ข้อกำหนดล่วงหน้าที่จะผนวกรวมไปกับผลลัพธ์ของการแปลง

ภาคผนวก ข. แท็คของ Z/EVES

การแปลงโปรแกรมภาษาซีเป็นข้อกำหนดรูปนัยในรูปสัญกรณ์เซตนั้นข้อกำหนดรูปนัยในรูปสัญกรณ์เซตที่ได้นั้นจะต้องอยู่ในรูปของแท็คของลาเท็กซ์ซึ่งยึดตามข้อกำหนดของเครื่องมือ Z/EVES เนื่องจากจะต้องอาศัย Z/EVES เป็นเครื่องมือในการตรวจสอบข้อกำหนด ซึ่งในตาราง ข-1 เป็นสรุปแท็คทั้งหมดที่กำหนดไว้ใน Z/EVES

ตาราง ข-1 แท็คของลาเท็กซ์ของสัญกรณ์เซตตามข้อกำหนดของ Z/EVES

Schema			**Relation**		
Name	**Z Notation**	**LATEX**	**Name**	**Z Notation**	**LATEX**
Axiomatic box	$\frac{D}{P}$	\begin{axdef} D \where P \end{schema}	maplet	$x \mapsto y$	x \mapsto y
			domain	dom x	\dom x
			range	ran x	\ran x
schema box	$\frac{S}{D \mid P}$	\begin{schema}{S} D \where P \end{schema}	composition	$x \fatsemi y$	x \comp y
			backward composition	$x \circ y$	x \circ y
horizontal schema	$S \mathrel{\widehat=} R$	S \defs R	domain restriction	$x \lhd y$	x \dres y
set comprehension	{D|P• E}	\{D|P@E\}	range restriction	$x \rhd y$	x \rres y
universal quantification	$\forall ST \bullet S$	\forall ST @ S	domain anti restriction	$x \lhd y$	x \ndres y
existential quantification	$\exists ST \bullet S$	\exists ST @ S	range anti restriction	$x \rhd y$	x \nrres y
Set			relational inverse	$x^\sim$	\inv x
inequality	$x \neq y$	x \neq y	relational image	$x(\!\|y\|\!)$	x\limg y \rimg
non-Membership	$x \notin y$	x \notin y	overriding	$x \oplus y$	x \oplus y
emptyset	$\varnothing$	\emptyset	**Bags**		
subset	$x \subset y$	x \subset y	multiplicity	$x \# y$	x \bcount y
set union	$x \cup y$	x \cup y	bags scaling	$x \otimes y$	x \otimes y
set intersection	$x \cap y$	x \cap y	bags membership	x in y	x \inbag y
set difference	$x \setminus y$	x \setminus y	sub bags	$x \sqsubset y$	x \subbag y
generalized union	$\bigcup x$	\bigcup x	bags union	$x \uplus y$	x \uplus y

ตาราง ข-1 แท็คของลาเท็กซ์ของสัญกรณ์เซตตามข้อกำหนดของ Z/EVES

Set			Bags		
Name	**Z Notation**	**LATEX**	**Name**	**Z Notation**	**LATEX**
generalized intersection	$\cap$ y	\bigcap y	bags difference	x $\uminus$ 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 $\frown$ 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 $\upharpoonright$ y	x \extract y			
filter	x $\upharpoonright$ y	x \filter y	number range	x..y	x \upto y
squash	squash x	squash~x	min	min x	min~x
distributed concatenation	$\frown$ / x	\dcat x	max	max x	max~x
			conjunction	x $\vee$ y	x \lor y
			disjunction	x $\wedge$ y	x \land y
			implication	x $\Rightarrow$ y	x \implies y
			equivalence	x $\Leftrightarrow$ y	x \iff y

ภาคผนวก ค. กรณีทดสอบ

กรณีทดสอบที่ 1

กรณีทดสอบที่ 1 เป็นโปรแกรมซึ่งประกอบด้วยฟังก์ชันทั้งหมด 3 ฟังก์ชันด้วยกันคือ

- ฟังก์ชัน findMin เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะทำการหาค่าจำนวนเต็มที่น้อยกว่าส่งคืนกลับมา
- ฟังก์ชัน findMax เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะทำการหาค่าจำนวนเต็มที่มากกว่าส่งคืนกลับมา
- ฟังก์ชัน main เป็นฟังก์ชันหลักที่จะทำการเรียกใช้งานฟังก์ชัน findMin และฟังก์ชัน findMax

โดยโปรแกรมต้นฉบับภาษาซีของกรณีทดสอบที่ 1 แสดงอยู่ในรูปที่ ค-1

```
int main()
{
  int x,y,Max,Min;
  int findMax(short,short);
  int findMin(int,int);

  Max = findMax(x,y);
  Min = findMin(x,y);
}

int findMax(short  a,short b)
{
  int ret;
  if (a > b)
    ret = a;
  else
    ret = b;
  return ret;
}
```

รูปที่ ค-1 โปรแกรมต้นฉบับภาษาซีของกรณีทดสอบที่ 1

```
int findMin(int a,int b)
{
  int ret;
  ret = b;
  if (a < b)
    ret = a;
  return ret;
}
```

รูปที่ ค-1 โปรแกรมต้นฉบับภาษาซีของกรณีทดสอบที่ 1

จากโปรแกรมต้นฉบับเมื่อนำมาทำการแปลงเป็นข้อกำหนดเซตผลลัพธ์ที่ได้จะประกอบด้วยเค้าร่างทั้งหมด 3 เค้าร่างตามจำนวนฟังก์ชันที่มีในโปรแกรมต้นฉบับดังแสดงอยู่ในรูปที่ ค-2

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus} {infun3} {RPlus}
\syndef{RMinus} {infun3} {RMinus}
\syndef{RMul} {infun3} {RMul}
\syndef{RDiv} {infun3} {RDiv}
\syndef{RG} {inrel} {RG}
\syndef{RL} {inrel} {RL}
\syndef{RGE} {inrel} {RGE}
\syndef{RLE} {inrel} {RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
```

รูปที่ ค-2 ข้อกำหนดที่ได้จากการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 1

```
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
                (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
                (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
                (second(x) = second(y) \land first(x) \geq first(y)))                \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
                (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{schema}{findMin}
findMin\_0\_a? : \num      \\
findMin\_0\_b? : \num      \\
findMin\_result! : \num    \\
findMin\_0\_ret : \num     \\
findMin\_0\_b_0, findMin\_0\_b' : \num \\
findMin\_0\_a_0, findMin\_0\_a' : \num \\
findMin\_0\_ret_0, findMin\_0\_ret_1, findMin\_0\_ret' : \num \\
\where
```

รูปที่ ค-2 ข้อกำหนดที่ได้จากการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 1

```
findMin\_0\_a_0 = findMin\_0\_a? \also
findMin\_0\_b_0 = findMin\_0\_b? \also
findMin\_0\_ret_0 = findMin\_0\_ret          \also
(findMin\_0\_a_0 < findMin\_0\_b_0)          \land
(findMin\_0\_ret_1 = findMin\_0\_a_0) \lor (\lnot (findMin\_0\_a_0 < findMin\_0\_b_0) ) \land
(findMin\_0\_ret_1 = findMin\_0\_b_0)        \also
findMin\_result! = findMin\_0\_ret_1 \also
findMin\_0\_b' = findMin\_0\_b_0 \also
findMin\_0\_a' = findMin\_0\_a_0 \also
findMin\_0\_ret' = findMin\_0\_ret_1
\end{schema}

\begin{schema}{findMax}
findMax\_0\_a? : \num      \\
findMax\_0\_b? : \num      \\
findMax\_result! : \num    \\
findMax\_0\_ret : \num     \\
findMax\_0\_b_0, findMax\_0\_b' : \num \\
findMax\_0\_a_0, findMax\_0\_a' : \num \\
findMax\_0\_ret_0, findMax\_0\_ret_1, findMax\_0\_ret' : \num \\
\where
findMax\_0\_a_0 = findMax\_0\_a? \also
findMax\_0\_b_0 = findMax\_0\_b? \also
findMax\_0\_ret_0 = findMax\_0\_ret          \also
(findMax\_0\_a_0 > findMax\_0\_b_0)          \land
(findMax\_0\_ret_1 = findMax\_0\_a_0) \lor (\lnot (findMax\_0\_a_0 > findMax\_0\_b_0) ) \land
(findMax\_0\_ret_1 = findMax\_0\_b_0)        \also
findMax\_result! = findMax\_0\_ret_1 \also
findMax\_0\_b' = findMax\_0\_b_0 \also
findMax\_0\_a' = findMax\_0\_a_0 \also
findMax\_0\_ret' = findMax\_0\_ret_1
```

รูปที่ ค-2 ข้อกำหนดที่ได้จากการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 1

```
\end{schema}

\begin{schema}{main}
main\_result! : \num          \\
main\_0\_x, main\_0\_y, main\_0\_Max, main\_0\_Min : \num \\
findMax              \\
findMin              \\
main\_0\_Min_0, main\_0\_Min' : \num \\
main\_0\_Max_0, main\_0\_Max' : \num \\
main\_0\_y_0, main\_0\_y' : \num \\
main\_0\_x_0, main\_0\_x' : \num \\
\where
main\_0\_x_0 = main\_0\_x            \also
main\_0\_y_0 = main\_0\_y            \also
main\_0\_Max_0 = main\_0\_Max  \also
main\_0\_Min_0 = main\_0\_Min   \also
findMax\_0\_a? = main\_0\_x_0                \also
findMax\_0\_b? = main\_0\_y_0     \also
main\_0\_Max_0 = findMax\_result!          \also
findMin\_0\_a? = main\_0\_x_0                \also
findMin\_0\_b? = main\_0\_y_0     \also
main\_0\_Min_0 = findMin\_result! \also
main\_0\_Min' = main\_0\_Min_0 \also
main\_0\_Max' = main\_0\_Max_0 \also
main\_0\_y' = main\_0\_y_0 \also
main\_0\_x' = main\_0\_x_0
\end{schema}
```

รูปที่ ค-2 ข้อกำหนดที่ได้จากการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 1

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-3

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema findMin
... axiom findMin\$declarationPart
schema findMax
... axiom findMax\$declarationPart
schema main
... axiom main\$declarationPart
Done.
```

รูปที่ ค-3 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 1

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-4

```
Proving gives ...
    main\_result! \in \num \\
\land main\_0\_x \in \num \\
\land main\_0\_y \in \num \\
\land main\_0\_Max \in \num \\
\land main\_0\_Min \in \num \\
\land main\_0\_x_0 = main\_0\_x \\
\land findMax\_0\_a? = main\_0\_x_0 \\
\land main\_0\_y_0 = main\_0\_y \\
\land findMax\_0\_b? = main\_0\_y_0 \\
\land findMax\_0\_ret_1 \in \num \\
\land findMax\_result! = findMax\_0\_ret_1 \\
\land findMax\_0\_ret \in \num \\
\land findMax\_0\_b_0 = findMax\_0\_b? \\
\land findMax\_0\_b' = findMax\_0\_b_0 \\
\land findMax\_0\_a_0 = findMax\_0\_a? \\
\land findMax\_0\_a' = findMax\_0\_a_0 \\
\land findMax\_0\_ret_0 = findMax\_0\_ret \\
\land findMax\_0\_ret' = findMax\_0\_ret_1 \\
\land findMin\_0\_a? = main\_0\_x_0 \\
\land findMin\_0\_b? = main\_0\_y_0 \\
\land findMin\_0\_ret_1 \in \num \\
\land findMin\_result! = findMin\_0\_ret_1 \\
\land findMin\_0\_ret \in \num \\
\land findMin\_0\_b_0 = findMin\_0\_b? \\
\land findMin\_0\_b' = findMin\_0\_b_0 \\
\land findMin\_0\_a_0 = findMin\_0\_a? \\
\land findMin\_0\_a' = findMin\_0\_a_0 \\
\land findMin\_0\_ret_0 = findMin\_0\_ret \\
\land findMin\_0\_ret' = findMin\_0\_ret_1 \\
\land main\_0\_Min_0 = main\_0\_Min \\
```

รูปที่ ค-4 ผลการตรวจสอบข้อกำหนดเชิดโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 1

```
\land main\_0\_Min_0 = findMin\_result! \\
\land main\_0\_Min' = main\_0\_Min_0 \\
\land main\_0\_Max_0 = main\_0\_Max \\
\land main\_0\_Max_0 = findMax\_result! \\
\land main\_0\_Max' = main\_0\_Max_0 \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x' = main\_0\_x_0 \\
\land (         findMax\_0\_ret_1 = findMax\_0\_a_0 \\
        \land findMax\_0\_a_0 > findMax\_0\_b_0 \\
    \lor      findMax\_0\_ret_1 = findMax\_0\_b_0 \\
        \land \lnot findMax\_0\_a_0 > findMax\_0\_b_0) \\
\land (         findMin\_0\_ret_1 = findMin\_0\_a_0 \\
        \land findMin\_0\_a_0 < findMin\_0\_b_0 \\
    \lor      findMin\_0\_ret_1 = findMin\_0\_b_0 \\
        \land \lnot findMin\_0\_a_0 < findMin\_0\_b_0)
```

รูปที่ ค-4 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 1

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร main_0_x เท่ากับ 4 และระบุค่า main_0_y เท่ากับ 5 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-5

```
Proving gives ...
    main\_result! \in \num \\
\land main\_0\_Max \in \num \\
\land main\_0\_Min \in \num \\
\land findMax\_0\_a? = 4 \\
\land findMax\_0\_b? = 5 \\
\land findMax\_0\_ret_1 = 5 \\
\land findMax\_result! = 5 \\
\land findMax\_0\_ret \in \num \\
\land findMax\_0\_b_0 = 5 \\
\land findMax\_0\_b' = 5 \\
\land findMax\_0\_a_0 = 4 \\
```

รูปที่ ค-5 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 1

```
\land findMax\_0\_a' = 4 \\
\land findMax\_0\_ret_0 = findMax\_0\_ret \\
\land findMax\_0\_ret' = 5 \\
\land findMin\_0\_a? = 4 \\
\land findMin\_0\_b? = 5 \\
\land findMin\_0\_ret_1 = 4 \\
\land findMin\_result! = 4 \\
\land findMin\_0\_ret \in \num \\
\land findMin\_0\_b_0 = 5 \\
\land findMin\_0\_b' = 5 \\
\land findMin\_0\_a_0 = 4 \\
\land findMin\_0\_a' = 4 \\
\land findMin\_0\_ret_0 = findMin\_0\_ret \\
\land findMin\_0\_ret' = 4 \\
\land main\_0\_Min_0 = main\_0\_Min \\
\land main\_0\_Min_0 = 4 \\
\land main\_0\_Min' = 4 \\
\land main\_0\_Max_0 = main\_0\_Max \\
\land main\_0\_Max_0 = 5 \\
\land main\_0\_Max' = 5 \\
\land main\_0\_y_0 = 5 \\
\land main\_0\_y' = 5 \\
\land main\_0\_x_0 = 4 \\
\land main\_0\_x' = 4
```

รูปที่ ค-5 ผลการตรวจสอบข้อกำหนดเชตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 1

กรณีทดสอบที่ 2

กรณีทดสอบที่ 2 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันทั้งหมด 3 ฟังก์ชันได้แก่

- ฟังก์ชัน findMax เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะทำการหาค่าจำนวนเต็มที่มากกว่าส่งคืนกลับมา
- ฟังก์ชัน successor เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะคำนวณค่าตัวตามหลัง (Successor) ของจำนวนเต็มที่ส่งเข้ามา

- ฟังก์ชัน main เป็นฟังก์ชันหลักที่จะทำการเรียกใช้งานฟังก์ชัน findMax และฟังก์ชัน successor

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่สองแสดงอยู่ในรูปที่ ค-6

```
int x,y,Max,s;
int main(int a, int b) {
  int findMax(int,int);
  int successor(int);
  x = a;
  y = b;
  Max = findMax(x,y);
  s = successor(Max);
  return s;
}

int findMax(int a,int b) {
  int ret;
  if (a > b)
    ret = a;
  else
    ret = b;
  return ret;
}
int successor(int x) {
  int a, y;
  a = x + 1;
  if (a - 1 == 0) {
    y = 1;
  } else {
    y = a;
  }
  return y;
}
```

รูปที่ ค-6 โปรแกรมต้นฉบับของกรณีทดสอบที่ 2

จากโปรแกรมต้นฉบับในรูปที่ ค-6 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี แต่เนื่องจากโปรแกรมมีการใช้งานการประกาศส่วนกลางดังนั้นจึงมีนิยามสัจพจน์เพิ่มขึ้นมา โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-7

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
```

รูปที่ ค-7 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 2

```
        (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
        (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
        (second(x) = second(y) \land first(x) \geq first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
        (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{axdef}
Global\_0\_s, Global\_0\_s_0 : \num \\
Global\_0\_Max, Global\_0\_Max_0 : \num \\
Global\_0\_y, Global\_0\_y_0, Global\_0\_y_1 : \num \\
Global\_0\_x, Global\_0\_x_0, Global\_0\_x_1 : \num \\
\where
Global\_0\_s_0 = Global\_0\_s      \also
Global\_0\_Max_0 = Global\_0\_Max      \also
Global\_0\_y_0 = Global\_0\_y      \also
Global\_0\_x_0 = Global\_0\_x      \also
\end{axdef}

\begin{schema}{successor}
successor\_0\_x? : \num    \\
successor\_result! : \num    \\
successor\_0\_a, successor\_0\_y : \num      \\
successor\_0\_x_0, successor\_0\_x' : \num \\
```

รูปที่ ค-7 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 2

```
successor\_0\_y_0, successor\_0\_y_1, successor\_0\_y' : \num \\
successor\_0\_a_0, successor\_0\_a_1, successor\_0\_a' : \num \\
\where
successor\_0\_x_0 = successor\_0\_x?          \also
successor\_0\_a_0 = successor\_0\_a           \also
successor\_0\_y_0 = successor\_0\_y           \also
successor\_0\_a_1 = (successor\_0\_x_0+1)  \also
((successor\_0\_a_1-1) = 0)           \land
(successor\_0\_y_1 = 1) \lor (\lnot ((successor\_0\_a_1-1) = 0) )          \land
(successor\_0\_y_1 = successor\_0\_a_1)     \also
successor\_result! = successor\_0\_y_1 \also
successor\_0\_x' = successor\_0\_x_0 \also
successor\_0\_y' = successor\_0\_y_1 \also
successor\_0\_a' = successor\_0\_a_1
\end{schema}

\begin{schema}{findMax}
findMax\_0\_a? : \num      \\
findMax\_0\_b? : \num      \\
findMax\_result! : \num     \\
findMax\_0\_ret : \num      \\
findMax\_0\_b_0, findMax\_0\_b' : \num \\
findMax\_0\_a_0, findMax\_0\_a' : \num \\
findMax\_0\_ret_0, findMax\_0\_ret_1, findMax\_0\_ret' : \num \\
\where
findMax\_0\_a_0 = findMax\_0\_a? \also
findMax\_0\_b_0 = findMax\_0\_b?\also
findMax\_0\_ret_0 = findMax\_0\_ret          \also
(findMax\_0\_a_0 > findMax\_0\_b_0)          \land
(findMax\_0\_ret_1 = findMax\_0\_a_0) \lor (\lnot (findMax\_0\_a_0 > findMax\_0\_b_0) ) \land
(findMax\_0\_ret_1 = findMax\_0\_b_0)       \also
findMax\_result! = findMax\_0\_ret_1 \also
```

รูปที่ ค-7 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 2

```
findMax\_0\_b' = findMax\_0\_b_0 \also
findMax\_0\_a' = findMax\_0\_a_0 \also
findMax\_0\_ret' = findMax\_0\_ret_1
\end{schema}

\begin{schema} {main}
main\_0\_a? : \num          \\
main\_0\_b? : \num          \\
main\_result! : \num        \\
findMax          \\
successor                   \\
main\_0\_b_0, main\_0\_b' : \num \\
main\_0\_a_0, main\_0\_a' : \num \\
\where
main\_0\_a_0 = main\_0\_a?          \also
main\_0\_b_0 = main\_0\_b?          \also
Global\_0\_x_1 = main\_0\_a_0       \also
Global\_0\_y_1 = main\_0\_b_0       \also
findMax\_0\_a? = Global\_0\_x_1             \also
findMax\_0\_b? = Global\_0\_y_1   \also
Global\_0\_Max_0 = findMax\_result!         \also
successor\_0\_x? = Global\_0\_Max_0         \also
Global\_0\_s_0 = successor\_result! \also
main\_result! = Global\_0\_s_0 \also
main\_0\_b' = main\_0\_b_0 \also
main\_0\_a' = main\_0\_a_0
\end{schema}
```

รูปที่ ค-7 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 2

ข้อกำหนดเชตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เชตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-8

definition of REAL

definition of CHAR

declaration of (_ RDiv _), (_ RG _), (_ RGE _), (_ RL _), (_ RLE _),

(_ RMinus _), (_ RMul _), (_ RPlus _)

... theorem unnamed\$n3\$domainCheck

... axiom RPlus\$declaration

... axiom RMinus\$declaration

... axiom RMul\$declaration

... axiom RDiv\$declaration

... axiom RG\$declaration

... axiom RL\$declaration

... axiom RGE\$declaration

... axiom RLE\$declaration

... axiom axiom\$4

... axiom axiom\$5

... axiom axiom\$6

... axiom axiom\$7

... axiom axiom\$8

... axiom axiom\$9

... axiom axiom\$10

... axiom axiom\$11

declaration of i2r

... axiom i2r\$declaration

... axiom axiom\$12

declaration of Global_0_Max, Global_0_Max_0, Global_0_s, Global_0_s_0,

Global_0_x, Global_0_x_0, Global_0_x_1, Global_0_y, Global_0_y_0,

Global_0_y_1

... axiom Global_0_s\$declaration

... axiom Global_0_s\$0\$declaration

... axiom Global_0_Max\$declaration

... axiom Global_0_Max\$0\$declaration

... axiom Global_0_y\$declaration

รูปที่ ค-8 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 2

```
... axiom Global\_0\_y\$0\$declaration
... axiom Global\_0\_y\$1\$declaration
... axiom Global\_0\_x\$declaration
... axiom Global\_0\_x\$0\$declaration
... axiom Global\_0\_x\$1\$declaration
... axiom axiom\$13
... axiom axiom\$14
... axiom axiom\$15
... axiom axiom\$16
schema successor
... axiom successor\$declarationPart
schema findMax
... axiom findMax\$declarationPart
schema main
... axiom main\$declarationPart
Done.
```

รูปที่ ค-8 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 2

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-9

```
Proving gives ...

    main\_0\_a? \in \num \\
\land main\_0\_b? \in \num \\
\land main\_result! = Global\_0\_s_0 \\
\land findMax\_0\_a? = Global\_0\_x_1 \\
\land findMax\_0\_b? = Global\_0\_y_1 \\
\land findMax\_0\_ret_1 \in \num \\
\land findMax\_result! = findMax\_0\_ret_1 \\
\land Global\_0\_Max_0 = findMax\_result! \\
\land findMax\_0\_ret \in \num \\
\land findMax\_0\_b_0 = Global\_0\_y_1 \\
\land findMax\_0\_b' = Global\_0\_y_1 \\
```

รูปที่ ค-9 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 2

```
\land findMax\_0\_a_0 = Global\_0\_x_1 \\
\land findMax\_0\_a' = Global\_0\_x_1 \\
\land findMax\_0\_ret_0 = findMax\_0\_ret \\
\land findMax\_0\_ret' = Global\_0\_Max_0 \\
\land successor\_0\_x? = Global\_0\_Max_0 \\
\land successor\_0\_y_1 \in \num \\
\land successor\_result! = successor\_0\_y_1 \\
\land Global\_0\_s_0 = successor\_result! \\
\land successor\_0\_a \in \num \\
\land successor\_0\_y \in \num \\
\land successor\_0\_x_0 = Global\_0\_Max_0 \\
\land successor\_0\_x' = Global\_0\_Max_0 \\
\land successor\_0\_y_0 = successor\_0\_y \\
\land successor\_0\_y' = Global\_0\_s_0 \\
\land successor\_0\_a_0 = successor\_0\_a \\
\land successor\_0\_a_1 = 1 + Global\_0\_Max_0 \\
\land successor\_0\_a' = 1 + Global\_0\_Max_0 \\
\land main\_0\_b_0 = main\_0\_b? \\
\land Global\_0\_y_1 = main\_0\_b_0 \\
\land main\_0\_b' = Global\_0\_y_1 \\
\land main\_0\_a_0 = main\_0\_a? \\
\land Global\_0\_x_1 = main\_0\_a_0 \\
\land main\_0\_a' = Global\_0\_x_1 \\
\land (     Global\_0\_Max_0 = Global\_0\_x_1 \\
    \land Global\_0\_x_1 > Global\_0\_y_1 \\
  \lor    Global\_0\_Max_0 = Global\_0\_y_1 \\
    \land \lnot Global\_0\_x_1 > Global\_0\_y_1) \\
\land (     Global\_0\_s_0 = 1 \\
    \land Global\_0\_Max_0 = 0 \\
  \lor Global\_0\_s_0 = 1 + Global\_0\_Max_0)
```

รูปที่ ค-9 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 2

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร main_0_a? เท่ากับ 5 และระบุค่า main_0_b? เท่ากับ 4 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-10

```
Proving gives ...

    main\_result! = Global\_0\_s_0 \\
\land findMax\_0\_a? = Global\_0\_x_1 \\
\land findMax\_0\_b? = Global\_0\_y_1 \\
\land findMax\_0\_ret_1 \in \num \\
\land findMax\_result! = findMax\_0\_ret_1 \\
\land Global\_0\_Max_0 = findMax\_result! \\
\land findMax\_0\_ret \in \num \\
\land findMax\_0\_b_0 = Global\_0\_y_1 \\
\land findMax\_0\_b' = Global\_0\_y_1 \\
\land findMax\_0\_a_0 = Global\_0\_x_1 \\
\land findMax\_0\_a' = Global\_0\_x_1 \\
\land findMax\_0\_ret_0 = findMax\_0\_ret \\
\land findMax\_0\_ret' = Global\_0\_Max_0 \\
\land successor\_0\_x? = Global\_0\_Max_0 \\
\land successor\_0\_y_1 \in \num \\
\land successor\_result! = successor\_0\_y_1 \\
\land Global\_0\_s_0 = successor\_result! \\
\land successor\_0\_a \in \num \\
\land successor\_0\_y \in \num \\
\land successor\_0\_x_0 = Global\_0\_Max_0 \\
\land successor\_0\_x' = Global\_0\_Max_0 \\
\land successor\_0\_y_0 = successor\_0\_y \\
\land successor\_0\_y' = Global\_0\_s_0 \\
\land successor\_0\_a_0 = successor\_0\_a \\
\land successor\_0\_a_1 = 1 + Global\_0\_Max_0 \\
\land successor\_0\_a' = 1 + Global\_0\_Max_0 \\
\land main\_0\_b_0 = 4 \\
```

รูปที่ ค-10 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 2

```
\land main\_0\_b' = 4 \\
\land main\_0\_a_0 = 5 \\
\land main\_0\_a' = 5 \\
\land Global\_0\_x_1 = 5 \\
\land Global\_0\_y_1 = 4 \\
\land Global\_0\_Max_0 = 5 \\
\land Global\_0\_s_0 = 6
```

รูปที่ ค-10 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 2

กรณีทดสอบที่ 3

กรณีทดสอบที่ 3 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน Test เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะทำการหาค่าจำนวนเต็มที่ขึ้นอยู่กับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมา

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 3 แสดงอยู่ในรูปที่ ค-11

```
int Test(int a)
{
  int y;
  y = 0;
  if (a < 0)
     y += 1;
  else if ( a >= 0 && a < 5)
     y += 2;
  else if ( a >= 5 && a < 10)
    y += 3;
  else if ( a >= 10 && a < 15)
  {
    y = a;
    y += 4;
  }
  return y;
}
```

รูปที่ ค-11 โปรแกรมต้นฉบับของกรณีทดสอบที่ 3

จากโปรแกรมต้นฉบับในรูปที่ ค-11 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 1 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-12

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
```

รูปที่ ค-12 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 3

```
        (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
        (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
        (second(x) = second(y) \land first(x) \geq first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
        (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{schema}{Test}
Test\_0\_a? : \num          \\
Test\_result! : \num        \\
Test\_0\_y : \num \\
Test\_0\_a_0, Test\_0\_a' : \num \\
Test\_0\_y_0, Test\_0\_y_1, Test\_0\_y_2, Test\_0\_y_3, Test\_0\_y' : \num \\
\where
Test\_0\_a_0 = Test\_0\_a?          \also
Test\_0\_y_0 = Test\_0\_y \also
Test\_0\_y_1 = 0  \also
(Test\_0\_a_0 < 0)          \land
(Test\_0\_y_2 = (Test\_0\_y_1+1)) \land (Test\_0\_y_3 = Test\_0\_y_2) \lor
(\lnot (Test\_0\_a_0 < 0) )  \land
((Test\_0\_a_0 \geq 0) \land (Test\_0\_a_0 < 5))        \land
(Test\_0\_y_2 = (Test\_0\_y_1+2)) \land (Test\_0\_y_3 = Test\_0\_y_2) \lor
(\lnot ((Test\_0\_a_0 \geq 0) \land (Test\_0\_a_0 < 5)) )          \land
((Test\_0\_a_0 \geq 5) \land (Test\_0\_a_0 < 10))       \land
```

รูปที่ ค-12 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 3

```
(Test\_0\_y_2 = (Test\_0\_y_1+3)) \land (Test\_0\_y_3 = Test\_0\_y_2) \lor
(\lnot ((Test\_0\_a_0 \geq 5) \land (Test\_0\_a_0 < 10)) )        \land
((Test\_0\_a_0 \geq 10) \land (Test\_0\_a_0 < 15))      \land
(Test\_0\_y_2 = Test\_0\_a_0)        \land
(Test\_0\_y_3 = (Test\_0\_y_2+4)) \lor
(\lnot((Test\_0\_a_0 \geq 10) \land (Test\_0\_a_0 < 15)))        \also
Test\_result! = Test\_0\_y_3 \also
Test\_0\_a' = Test\_0\_a_0 \also
Test\_0\_y' = Test\_0\_y_3
\end{schema}
```

รูปที่ ค-12 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 3

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-13

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
```

รูปที่ ค-13 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 3

```
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema Test
... axiom Test\$declarationPart
Done.
```

รูปที่ ค-13 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 3

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-14

```
Proving gives ...

    Test\_0\_a? \in \num \\
\land Test\_0\_y_3 \in \num \\
\land Test\_result! = Test\_0\_y_3 \\
\land Test\_0\_y \in \num \\
\land Test\_0\_a_0 = Test\_0\_a? \\
\land Test\_0\_a' = Test\_0\_a_0 \\
\land Test\_0\_y_0 = Test\_0\_y \\
\land Test\_0\_y_1 = 0 \\
\land Test\_0\_y_2 \in \num \\
\land Test\_0\_y' = Test\_0\_y_3 \\
\land (          Test\_0\_a_0 \geq 10 \\
        \land (          Test\_0\_y_2 = Test\_0\_a_0 \\
                \land Test\_0\_y_3 = 4 + Test\_0\_y_2 \\
            \implies \lnot Test\_0\_a_0 < 15) \\
    \implies \lnot Test\_0\_a_0 < 15)
```

รูปที่ ค-14 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 3

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร Test_0_a? เท่ากับ 2 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-15

```
Proving gives ...

    Test\_0\_y_3 \in \num \\
\land Test\_result! = Test\_0\_y_3 \\
\land Test\_0\_y \in \num \\
\land Test\_0\_a_0 = 2 \\
\land Test\_0\_a' = 2 \\
\land Test\_0\_y_0 = Test\_0\_y \\
\land Test\_0\_y_1 = 0 \\
\land Test\_0\_y_2 \in \num \\
\land Test\_0\_y' = Test\_0\_y_3
```

รูปที่ ค-15 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 3

กรณีทดสอบที่ 4

กรณีทดสอบที่ 4 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน main เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นตัวอักขระฟังก์ชันจะทำการหาค่าจำนวนเต็มที่ขึ้นอยู่กับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมาโดยภายในฟังก์ชันจะมีการใช้งานข้อความสั่ง switch

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 4 แสดงอยู่ในรูปที่ ค-16

```
int main(char ch)
{
  unsigned b,c;
  c = 5;
  switch (ch)
  {
    case 'a' :
          b = 1;
          b += 5;
          break;
```

รูปที่ ค-16 โปรแกรมต้นฉบับของกรณีทดสอบที่ 4

```
        case 'b' :
            b = 2;
            b += 5;
            b++;
            break;
        case 'c' :
        case 'd' :
            b = 3;
            break;
        defaults :
            b = c;
            break;
    }
    return b;
}
```

รูปที่ ค-16 โปรแกรมต้นฉบับของกรณีทดสอบที่ 4

จากโปรแกรมต้นฉบับในรูปที่ ค-16 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-17

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}
\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}
```

รูปที่ ค-17 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 4

```
\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
                (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
                (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
                (second(x) = second(y) \land first(x) \geq first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
                (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}


\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}
\begin{schema}{main}
main\_0\_ch? : CHAR      \\
main\_result! : \num      \\
```

รูปที่ ค-17 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 4

```
main\_0\_b, main\_0\_c : \nat          \\
main\_0\_ch_0, main\_0\_ch' : CHAR \\
main\_0\_c_0, main\_0\_c_1, main\_0\_c' : \nat \\
main\_0\_b_0, main\_0\_b_1, main\_0\_b_2, main\_0\_b_3, main\_0\_b' : \nat \\
\where
main\_0\_ch_0 = main\_0\_ch?       \also
main\_0\_b_0 = main\_0\_b          \also
main\_0\_c_0 = main\_0\_c          \also
main\_0\_c_1 = 5 \also
(main\_0\_ch_0 = 97)       \land
(main\_0\_b_1 = 1)         \land
(main\_0\_b_2 = (main\_0\_b_1+5)) \land (main\_0\_b_3 = main\_0\_b_2) \lor
(\lnot (main\_0\_ch_0 = 97) )        \land
(main\_0\_ch_0 = 98)       \land
(main\_0\_b_1 = 2)         \land
(main\_0\_b_2 = (main\_0\_b_1+5))          \land
(main\_0\_b_3 = (main\_0\_b_2+1)) \lor (\lnot (main\_0\_ch_0 = 98) )   \land
((main\_0\_ch_0 = 99) \lor (main\_0\_ch_0 = 100))    \land
(main\_0\_b_1 = 3) \lor (\lnot ((main\_0\_ch_0 = 99) \lor
(main\_0\_ch_0 = 100)) )   \land
(main\_0\_b_1 = main\_0\_c_1) \land (main\_0\_b_3 = main\_0\_b_1)   \also
main\_result! = main\_0\_b_3 \also
main\_0\_ch' = main\_0\_ch_0 \also
main\_0\_c' = main\_0\_c_1 \also
main\_0\_b' = main\_0\_b_3
\end{schema}
```

รูปที่ ค-17 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 4

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-18

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema main
... axiom main\$declarationPart
Done.
```

รูปที่ ค-18 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 4

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-19

```
Proving gives ...


    main\_0\_b_3 \in \num \\
\land main\_result! = main\_0\_b_3 \\
\land main\_0\_b \in \num \\
\land main\_0\_c \in \num \\
\land main\_0\_c_0 = main\_0\_c \\
\land main\_0\_c_1 = 5 \\
\land main\_0\_c' = 5 \\
\land main\_0\_b_0 = main\_0\_b \\
\land main\_0\_b_1 \in \num \\
\land main\_0\_b_2 \in \num \\
\land main\_0\_b' = main\_0\_b_3 \\
\land main\_0\_ch_0 = main\_0\_ch? \\
\land main\_0\_ch' = main\_0\_ch_0 \\
\land 0 \leq main\_0\_ch? \\
\land main\_0\_ch? \leq 255 \\
\land main\_0\_b \geq 0 \\
\land main\_0\_c \geq 0 \\
\land main\_0\_b_1 \geq 0 \\
\land main\_0\_b_2 \geq 0 \\
\land main\_0\_b_3 \geq 0 \\
\land (         main\_0\_ch_0 = 97 \\
        \land main\_0\_b_1 = 1 \\
        \land main\_0\_b_2 = 6 \\
        \land main\_0\_b_3 = 6 \\
    \lor      main\_0\_ch_0 = 98 \\
        \land main\_0\_b_1 = 2 \\
        \land main\_0\_b_2 = 7 \\
        \land main\_0\_b_3 = 8 \\
    \lor      main\_0\_b_1 = 3 \\
```

รูปที่ ค-19 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 4

```
            \land \lnot main\_0\_ch_0 = 98 \\
    \land (     main\_0\_ch_0 = 99 \\
                \lor main\_0\_ch_0 = 100) \\
        \lor      main\_0\_b_1 = 5 \\
            \land main\_0\_b_3 = 5 \\
            \land \lnot main\_0\_ch_0 = 99 \\
            \land \lnot main\_0\_ch_0 = 100)
```

รูปที่ ค-19 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 4

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร Test_0_a? เท่ากับ 2 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-20

```
Proving gives ...
    main\_0\_b_3 \in \num \land main\_result! = main\_0\_b_3 \\
\land main\_0\_b \in \num \\
\land main\_0\_c \in \num \\
\land main\_0\_c_0 = main\_0\_c \\
\land main\_0\_c_1 = 5 \\
\land main\_0\_c' = 5 \\
\land main\_0\_b_0 = main\_0\_b \\
\land main\_0\_b_1 \in \num \\
\land main\_0\_b_2 \in \num \\
\land main\_0\_b' = main\_0\_b_3 \\
\land main\_0\_ch_0 = 98 \\
\land main\_0\_ch' = 98 \\
\land main\_0\_b \geq 0 \\
\land main\_0\_c \geq 0 \\
\land main\_0\_b_1 \geq 0 \\
\land main\_0\_b_2 \geq 0 \\
\land main\_0\_b_3 \geq 0 \\
\land main\_0\_b_1 = 2 \\
\land main\_0\_b_2 = 7 \\
\land main\_0\_b_3 = 8 \\
```

รูปที่ ค-20 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 4

กรณีทดสอบที่ 5

กรณีทดสอบที่ 5 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน multi เป็นฟังก์ชันที่รับพารามิเตอร์สองตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะนำเอาค่าจำนวนเต็มทั้งสองค่ามาคูณกันแล้วส่งคืนค่าผลคูณซึ่งเป็นจำนวนเต็มกลับมา

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 5 แสดงอยู่ในรูปที่ ค-21

```
int multi(int x, int y) {
        int a, z;

        a = 0;
        z = 0;
        while (a != y) {
                z += x;
                a++;
        }
        return z;
}
```

รูปที่ ค-21 โปรแกรมต้นฉบับของกรณีทดสอบที่ 5

จากโปรแกรมต้นฉบับในรูปที่ ค-21 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-22

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}
\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
```

รูปที่ ค-22 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 5

```
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
            (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
            (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
            (second(x) = second(y) \land first(x) \geq first(y)))                \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
            (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}
\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}
```

รูปที่ ค-22 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 5

```
\begin{schema} {multi}
multi\_0\_x? : \num          \\
multi\_0\_y? : \num          \\
multi\_result! : \num        \\
multi\_0\_a, multi\_0\_z : \num       \\
multi\_0\_y_0, multi\_0\_y_1, multi\_0\_y' : \num \\
multi\_0\_x_0, multi\_0\_x' : \num \\
multi\_0\_z_0, multi\_0\_z_1, multi\_0\_z_2, multi\_0\_z' : \num \\
multi\_0\_a_0, multi\_0\_a_1, multi\_0\_a_2, multi\_0\_a' : \num \\
\where
multi\_0\_x_0 = multi\_0\_x?        \also
multi\_0\_y_0 = multi\_0\_y?        \also
multi\_0\_a_0 = multi\_0\_a         \also
multi\_0\_z_0 = multi\_0\_z         \also
multi\_0\_a_1 = 0 \also
multi\_0\_z_1 = 0 \also
\lnot(multi\_0\_a_2 \neq multi\_0\_y_1) \land
( ( multi\_0\_z_2 = multi\_0\_a_2 * multi\_0\_x_0 ) )  \also
multi\_result! = multi\_0\_z_2 \also
multi\_0\_y' = multi\_0\_y_1 \also
multi\_0\_x' = multi\_0\_x_0 \also
multi\_0\_z' = multi\_0\_z_2 \also
multi\_0\_a' = multi\_0\_a_2
\end{schema}
```

รูปที่ ค-22 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 5

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-23

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema multi
... axiom multi\$declarationPart
Done.
```

รูปที่ ค-23 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเชตของกรณีทดสอบที่ 5

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-24

```
Proving gives ...

    multi\_0\_x? \in \num \\
\land multi\_0\_y? \in \num \\
\land multi\_0\_x_0 = multi\_0\_x? \\
\land multi\_0\_y_1 \in \num \\
\land multi\_0\_a_2 = multi\_0\_y_1 \\
\land multi\_0\_z_2 = multi\_0\_a_2 * multi\_0\_x_0 \\
\land multi\_result! = multi\_0\_a_2 * multi\_0\_x_0 \\
\land multi\_0\_a \in \num \\
\land multi\_0\_z \in \num \\
\land multi\_0\_y_0 = multi\_0\_y? \\
\land multi\_0\_y' = multi\_0\_y_1 \\
\land multi\_0\_x' = multi\_0\_x_0 \\
\land multi\_0\_z_0 = multi\_0\_z \\
\land multi\_0\_z_1 = 0 \\
\land multi\_0\_z' = multi\_0\_a_2 * multi\_0\_x_0 \\
\land multi\_0\_a_0 = multi\_0\_a \\
\land multi\_0\_a_1 = 0 \\
\land multi\_0\_a' = multi\_0\_a_2
```

รูปที่ ค-24 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 5

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร multi_0_x? เท่ากับ 10 และ ระบุค่าตัวแปร multi_0_y? เท่ากับ 5 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-25

```
Proving gives ...

    multi\_0\_y_1 \in \num \\
\land multi\_0\_a_2 = multi\_0\_y_1 \\
\land multi\_0\_z_2 = 10 * multi\_0\_a_2 \\
\land multi\_result! = 10 * multi\_0\_a_2 \\
```

รูปที่ ค-25 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 5

```
\land multi\_0\_a \in \num \\
\land multi\_0\_z \in \num \\
\land multi\_0\_y_0 = 5 \\
\land multi\_0\_y' = multi\_0\_y_1 \\
\land multi\_0\_x_0 = 10 \\
\land multi\_0\_x' = 10 \\
\land multi\_0\_z_0 = multi\_0\_z \\
\land multi\_0\_z_1 = 0 \\
\land multi\_0\_z' = 10 * multi\_0\_a_2 \\
\land multi\_0\_a_0 = multi\_0\_a \\
\land multi\_0\_a_1 = 0 \\
\land multi\_0\_a' = multi\_0\_a_2
```

รูปที่ ค-25 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 5

กรณีทดสอบที่ 6

กรณีทดสอบที่ 6 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน sum เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็มฟังก์ชันจะทำการหาค่าผลรวมตั้งแต่ 1 จนถึงค่าของพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มกลับมา

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 6 แสดงอยู่ในรูปที่ ค-26

```
int sum(int x) {
        int z, y;

        y = x;
        for (z = 0;y > 0;y--) {
                z = z + y;
        }
        return z;
}
```

รูปที่ ค-26 โปรแกรมต้นฉบับของกรณีทดสอบที่ 6

จากโปรแกรมต้นฉบับในรูปที่ ค-26 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-27

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
                (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
                (second(x) = second(y) \land first(x) < first(y)))          \\
```

รูปที่ ค-27 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 6

```
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
                (second(x) = second(y) \land first(x) \geq first(y)))                    \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
                (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{schema}{sum}
sum\_0\_x? : \num          \\
sum\_result! : \num        \\
sum\_0\_z, sum\_0\_y : \num        \\
sum\_0\_x_0, sum\_0\_x' : \num \\
sum\_0\_y_0, sum\_0\_y_1, sum\_0\_y_2, sum\_0\_y' : \num \\
sum\_0\_z_0, sum\_0\_z_1, sum\_0\_z' : \num \\
\where
sum\_0\_x_0 = sum\_0\_x?          \also
sum\_0\_z_0 = sum\_0\_z  \also
sum\_0\_y_0 = sum\_0\_y  \also
sum\_0\_y_1 = sum\_0\_x_0         \also
\lnot(sum\_0\_y_2 > 0) \land
( ( sum\_0\_z_1 = ( ( sum\_0\_x_0 * ( sum\_0\_x_0 + 1 ) ) \div 2 ) ) )        \also
sum\_result! = sum\_0\_z_1 \also
sum\_0\_x' = sum\_0\_x_0 \also
sum\_0\_y' = sum\_0\_y_2 \also
sum\_0\_z' = sum\_0\_z_1
\end{schema}
```

รูปที่ ค-27 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 6

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-28

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema sum
... theorem sum\$domainCheck
... axiom sum\$declarationPart
Done.
```

รูปที่ ค-28 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 6

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-29

```
Proving gives ...

    sum\_0\_x? \in \num \\
\land sum\_0\_x_0 = sum\_0\_x? \\
\land sum\_0\_z_1 = sum\_0\_x_0 * (1 + sum\_0\_x_0) \div 2 \\
\land sum\_result! = sum\_0\_x_0 * (1 + sum\_0\_x_0) \div 2 \\
\land sum\_0\_z \in \num \\
\land sum\_0\_y \in \num \\
\land sum\_0\_x' = sum\_0\_x_0 \\
\land sum\_0\_y_0 = sum\_0\_y \\
\land sum\_0\_y_1 = sum\_0\_x_0 \\
\land sum\_0\_y_2 \in \num \\
\land sum\_0\_y' = sum\_0\_y_2 \\
\land sum\_0\_z_0 = sum\_0\_z \\
\land sum\_0\_z' = sum\_0\_x_0 * (1 + sum\_0\_x_0) \div 2 \\
\land \lnot sum\_0\_y_2 > 0
```

รูปที่ ค-29 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 6

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร Test_0_x? เท่ากับ 10 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-30

```
Proving gives ...

    sum\_result! = 55 \\
\land sum\_0\_z \in \num \\
\land sum\_0\_y \in \num \\
\land sum\_0\_x_0 = 10 \\
\land sum\_0\_x' = 10 \\
\land sum\_0\_y_0 = sum\_0\_y \\
\land sum\_0\_y_1 = 10 \\
\land sum\_0\_y_2 \in \num \\
```

รูปที่ ค-30 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 6

```
\land sum\_0\_y' = sum\_0\_y_2 \\
\land sum\_0\_z_0 = sum\_0\_z \\
\land sum\_0\_z_1 = 55 \\
\land sum\_0\_z' = 55 \\
\land \lnot sum\_0\_y_2 > 0
```

รูปที่ ค-30 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 6

กรณีทดสอบที่ 7

กรณีทดสอบที่ 7 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน Copy เป็นฟังก์ชันที่รับพารามิเตอร์หนึ่งตัวมีชนิดข้อมูลเป็นจำนวนเต็ม ฟังก์ชันจะใช้ข้อความสั่งวนซ้ำ do while ทำการหาค่าจำนวนเต็มที่เท่ากับพารามิเตอร์แล้วส่งคืนค่าจำนวนเต็มนั้นกลับมา

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 7 แสดงอยู่ในรูปที่ ค-31

```
int copy(int x) {
        int a,y;
        a = x;
        y = 0;
        do
        {
                y = y + 1;
                a = a -1;
        }while (a != 0);
        return y;
}
```

รูปที่ ค-31 โปรแกรมต้นฉบับของกรณีทดสอบที่ 7

จากโปรแกรมต้นฉบับในรูปที่ ค-31 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-32

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus} {infun3} {RPlus}
\syndef{RMinus} {infun3} {RMinus}
\syndef{RMul} {infun3} {RMul}
\syndef{RDiv} {infun3} {RDiv}
\syndef{RG} {inrel} {RG}
\syndef{RL} {inrel} {RL}
\syndef{RGE} {inrel} {RGE}
\syndef{RLE} {inrel} {RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
                (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
                (second(x) = second(y) \land first(x) < first(y)))          \\
```

รูปที่ ค-32 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 7

```
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
                (second(x) = second(y) \land first(x) \geq first(y)))                    \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
                (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{schema}{copy}
copy\_0\_x? : \num          \\
copy\_result! : \num        \\
copy\_0\_a, copy\_0\_y : \num       \\
copy\_0\_x_0, copy\_0\_x' : \num \\
copy\_0\_y_0, copy\_0\_y_1, copy\_0\_y_2, copy\_0\_y' : \num \\
copy\_0\_a_0, copy\_0\_a_1, copy\_0\_a_2, copy\_0\_a' : \num \\
\where
copy\_0\_x_0 = copy\_0\_x?          \also
copy\_0\_a_0 = copy\_0\_a           \also
copy\_0\_y_0 = copy\_0\_y           \also
copy\_0\_a_1 = copy\_0\_x_0         \also
copy\_0\_y_1 = 0 \also
 \lnot(copy\_0\_a_2 \neq 0) \land ( ( copy\_0\_y_2 = copy\_0\_x_0 - copy\_0\_a_2 ) )       \also
copy\_result! = copy\_0\_y_2 \also
copy\_0\_x' = copy\_0\_x_0 \also
copy\_0\_y' = copy\_0\_y_2 \also
copy\_0\_a' = copy\_0\_a_2
\end{schema}
```

รูปที่ ค-32 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 7

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-33

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema copy
... axiom copy\$declarationPart
Done.
```

รูปที่ ค-33 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 7

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-34

```
Proving gives ...

    copy\_0\_x? \in \num \\
\land copy\_0\_x_0 = copy\_0\_x? \\
\land copy\_0\_y_2 = copy\_0\_x_0 \\
\land copy\_result! = copy\_0\_y_2 \\
\land copy\_0\_a \in \num \\
\land copy\_0\_y \in \num \\
\land copy\_0\_x' = copy\_0\_x_0 \\
\land copy\_0\_y_0 = copy\_0\_y \\
\land copy\_0\_y_1 = 0 \\
\land copy\_0\_y' = copy\_0\_y_2 \\
\land copy\_0\_a_0 = copy\_0\_a \\
\land copy\_0\_a_1 = copy\_0\_x_0 \\
\land copy\_0\_a_2 = 0 \\
\land copy\_0\_a' = 0
```

รูปที่ ค-34 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 7

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร copy_0_x? เท่ากับ 20 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-35

```
Proving gives ...

    copy\_result! = 20 \\
\land copy\_0\_a \in \num \\
\land copy\_0\_y \in \num \\
\land copy\_0\_x_0 = 20 \\
\land copy\_0\_x' = 20 \\
\land copy\_0\_y_0 = copy\_0\_y \\
\land copy\_0\_y_1 = 0 \\
\land copy\_0\_y_2 = 20 \\
\land copy\_0\_y' = 20 \\
```

รูปที่ ค-35 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยตัวอย่างของกรณีทดสอบที่ 7

```
\land copy\_0\_a_0 = copy\_0\_a \\
\land copy\_0\_a_1 = 20 \\
\land copy\_0\_a_2 = 0 \\
\land copy\_0\_a' = 0
```

รูปที่ ค-35 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 7

กรณีทดสอบที่ 8

กรณีทดสอบที่ 8 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันเพียงฟังก์ชันเดียวได้แก่

- ฟังก์ชัน main เป็นฟังก์ชันที่ไม่มีการรับพารามิเตอร์จะทำการคำนวณค่าจำนวนจริงและการทำงานร่วมกันระหว่างจำนวนจริงและจำนวนเต็ม

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 8 แสดงอยู่ในรูปที่ ค-36

```
int main()
{
  int a,b;
  float e,f;
  double i;

  e = (b * 10)/5;
  a = (e + f);
  i = a + e + 5  ;
}
```

รูปที่ ค-36 โปรแกรมต้นฉบับของกรณีทดสอบที่ 8

จากโปรแกรมต้นฉบับในรูปที่ ค-36 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 3 เค้าร่างตามจำนวนของฟังก์ชันที่มี โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-37

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus}{infun3}{RPlus}
```

รูปที่ ค-37 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 8

```
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
                (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
                (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
                (second(x) = second(y) \land first(x) \geq first(y)))                    \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
                (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}
```

รูปที่ ค-37 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 8

```
\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
\end{axdef}

\begin{schema}{main}
main\_result! : \num          \\
main\_0\_a, main\_0\_b : \num       \\
main\_0\_e, main\_0\_f : REAL       \\
main\_0\_i : REAL           \\
main\_0\_i_0, main\_0\_i_1, main\_0\_i' : REAL \\
main\_0\_f_0, main\_0\_f' : REAL \\
main\_0\_e_0, main\_0\_e_1, main\_0\_e' : REAL \\
main\_0\_b_0, main\_0\_b' : \num \\
main\_0\_a_0, main\_0\_a_1, main\_0\_a' : \num \\
\where
main\_0\_a_0 = main\_0\_a          \also
main\_0\_b_0 = main\_0\_b          \also
main\_0\_e_0 = main\_0\_e          \also
main\_0\_f_0 = main\_0\_f\also
main\_0\_i_0 = main\_0\_i \also
main\_0\_e_1 = i2r(((main\_0\_b_0*10) \div 5))        \also
main\_0\_a_1 = i2r \inv ((main\_0\_e_1 RPlus main\_0\_f_0))  \also
main\_0\_i_1 = ((i2r(main\_0\_a_1) RPlus main\_0\_e_1) RPlus i2r(5)) \also
main\_0\_i' = main\_0\_i_1 \also
main\_0\_f' = main\_0\_f_0 \also
main\_0\_e' = main\_0\_e_1 \also
main\_0\_b' = main\_0\_b_0 \also
main\_0\_a' = main\_0\_a_1
\end{schema}
```

รูปที่ ค-37 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 8

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-38

```
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
... axiom axiom\$6
... axiom axiom\$7
... axiom axiom\$8
... axiom axiom\$9
... axiom axiom\$10
... axiom axiom\$11
declaration of i2r
... axiom i2r\$declaration
... axiom axiom\$12
schema main
... theorem main\$domainCheck
... axiom main\$declarationPart
Done.
```

รูปที่ ค-38 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 8

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-39

```
Proving gives ...


    main\_result! \in \num \\
\land main\_0\_a \in \num \\
\land main\_0\_b \in \num \\
\land main\_0\_b_0 = main\_0\_b \\
\land main\_0\_b' = main\_0\_b_0 \\
\land main\_0\_a_0 = main\_0\_a \\
\land main\_0\_f_0 = main\_0\_f \\
\land main\_0\_e_1 = i2r (10 * main\_0\_b_0 \div 5) \\
\land
 main\_0\_a_1 = i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0) \\
\land
 main\_0\_a' = i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0) \\
\land main\_0\_e_0 = main\_0\_e \\
\land main\_0\_i_0 = main\_0\_i \\
\land main\_0\_i_1
    = i2r (i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0))
      RPlus i2r (10 * main\_0\_b_0 \div 5) RPlus i2r 5 \\
\land main\_0\_i'
    = i2r (i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0))
      RPlus i2r (10 * main\_0\_b_0 \div 5) RPlus i2r 5 \\
\land main\_0\_f' = main\_0\_f_0 \\
\land main\_0\_e' = i2r (10 * main\_0\_b_0 \div 5) \\
\land i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0) \in \num \\
\land (\exists x: \num @ (\exists y: \num @ main\_0\_e = (x, y))) \\
\land (\exists x\_\_0: \num
      @ (\exists y\_\_0: \num @ main\_0\_f = (x\_\_0, y\_\_0))) \\
\land (\exists x\_\_1: \num
      @ (\exists y\_\_1: \num @ main\_0\_i = (x\_\_1, y\_\_1))) \\
\land (\exists x\_\_2: \num
```

รูปที่ ค-39 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 8

```
      @ (\exists y\_\_2: \num @ main\_0\_i_0 = (x\_\_2, y\_\_2))) \\
\land
 (\exists x\_\_3: \num
   @ (\exists y\_\_3: \num
      @ i2r (i2r \inv (i2r (10 * main\_0\_b_0 \div 5) RPlus main\_0\_f_0))
        RPlus i2r (10 * main\_0\_b_0 \div 5) RPlus i2r 5
       = (x\_\_3, y\_\_3))) \\
\land (\exists x\_\_4: \num
      @ (\exists y\_\_4: \num @ main\_0\_f_0 = (x\_\_4, y\_\_4))) \\
\land (\exists x\_\_5: \num
      @ (\exists y\_\_5: \num @ main\_0\_f' = (x\_\_5, y\_\_5))) \\
\land (\exists x\_\_6: \num
      @ (\exists y\_\_6: \num @ main\_0\_e_0 = (x\_\_6, y\_\_6))) \\
\land (\exists x\_\_7: \num
      @ (\exists y\_\_7: \num
         @ i2r (10 * main\_0\_b_0 \div 5) = (x\_\_7, y\_\_7)))
```

รูปที่ ค-39 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 8

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร main_0_b เท่ากับ 2 และ main_0_f เท่ากับ 5.25 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-40

```
Proving gives ...


    main\_result! \in \num \\
\land main\_0\_a \in \num \\
\land main\_0\_b_0 = 2 \\
\land main\_0\_b' = 2 \\
\land main\_0\_a_0 = main\_0\_a \\
\land main\_0\_f_0 = (525, 2) \\
\land main\_0\_e_1 = i2r 4 \\
\land main\_0\_a_1 = i2r \inv (i2r 4 RPlus (525, 2)) \\
\land main\_0\_a' = i2r \inv (i2r 4 RPlus (525, 2)) \\
```

รูปที่ ค-40 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 8

```
\land main\_0\_e_0 = main\_0\_e \\
\land main\_0\_i_0 = main\_0\_i \\
\land
main\_0\_i_1 = i2r (i2r \inv (i2r 4 RPlus (525, 2))) RPlus i2r 4 RPlus i2r 5 \\
\land
 main\_0\_i' = i2r (i2r \inv (i2r 4 RPlus (525, 2))) RPlus i2r 4 RPlus i2r 5 \\
\land main\_0\_f' = (525, 2) \\
\land main\_0\_e' = i2r 4 \\
\land i2r \inv (i2r 4 RPlus (525, 2)) \in \num \\
\land (\exists x: \num @ (\exists y: \num @ main\_0\_e = (x, y))) \\
\land (\exists x\_\_0: \num
      @ (\exists y\_\_0: \num @ main\_0\_i = (x\_\_0, y\_\_0))) \\
\land (\exists x\_\_1: \num
      @ (\exists y\_\_1: \num @ main\_0\_i_0 = (x\_\_1, y\_\_1))) \\
\land (\exists x\_\_2: \num
      @ (\exists y\_\_2: \num
         @ i2r (i2r \inv (i2r 4 RPlus (525, 2))) RPlus i2r 4 RPlus i2r 5
          = (x\_\_2, y\_\_2))) \\
\land (\exists x\_\_3: \num
      @ (\exists y\_\_3: \num @ main\_0\_e_0 = (x\_\_3, y\_\_3))) \\
\land
 (\exists x\_\_4: \num @ (\exists y\_\_4: \num @ i2r 4 = (x\_\_4, y\_\_4)))
```

รูปที่ ค-40 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 8

กรณีทดสอบที่ 9

กรณีทดสอบที่ 9 เป็นโปรแกรมที่ประกอบด้วยฟังก์ชันทั้งหมด 10 ฟังก์ชันเพื่อทำการทดสอบกฎทั้งหมดที่ได้ทำการกำหนดขึ้น

โดยโปรแกรมต้นฉบับของกรณีทดสอบที่ 9 แสดงอยู่ในรูปที่ ค-41

```
int i,j,k,l,m;
int main(int x, int y) {
        int ret;
        i = compare(x,y);
        j = mid(x,y);
        k = div(x,y);
        k = multi(k,5);
        l = copy(k);
        k = IFTest(l);
        m = SwitchTest('b');
        ret = i+j+k+l+m;
        return ret;
}

int compare(int a, int b) {
        int ret;
        if (a == b)
                ret = 0;
        else if (a < b)
                ret = -1;
        else
                ret = 1;
        return ret;
}

short swap(float a, float b) {
        float temp;
        temp = a;
        a = b;
        b = temp;
        return 0;
}
```

รูปที่ ค-41 โปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
unsigned mid(unsigned a,unsigned b) {
        int i;
        i = (a + b)/2;
        return i;
}

int div(int x, int y) {
        int r,d;
        r = x;
        d = 0;
        while(r >= 0) {
                r = r-y;
                d = d+1;
        }
        return d;
}

int multi(int x, int y) {
        int a, z;
        z = 0;
        for (a = 0;a != y;a++) {
                z += x;
        }
        return z;
}

int copy(int x) {
        int a,y;
        a = x;
        y = 0;
        do
        {
```

รูปที่ ค-41 โปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
y = y + 1;
a = a - 1;
        }while (a != 0);
        return y;
}

int IFTest(int a) {
        int y;
        if (a < 0)
                y = 0;
        else
                y = 1;
        return y;
}

int successor(int x) {
        int a, y;
        a = x + 1;
        y = a;
        if (a - 1 == 0) {
                y = 1;
        }
        return y;
}

int SwitchTest(char ch) {
        unsigned b,c;
        c = 5;
        switch (ch)
        {
                case 'a' :
                        b = 1;
```

รูปที่ ค-41 โปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
                b += 5;
        break;
                case 'b' :
                        b = 2;
                        b += 5;
                        b++;
                        break;
                case 'c' :
                case 'd' :
                        b = 3;
                        break;
                default:
                        b = 4;
                        break;
        }
        return b;
}
```

รูปที่ ค-41 โปรแกรมต้นฉบับของกรณีทดสอบที่ 9

จากโปรแกรมต้นฉบับในรูปที่ ค-41 เมื่อนำมาทำการแปลงเป็นข้อกำหนดจะได้ข้อกำหนดเซตทั้งหมด 10 เค้าร่างตามจำนวนของฟังก์ชันที่มี แต่เนื่องจากโปรแกรมมีการใช้งานการประกาศส่วนกลางดังนั้นจึงมีนิยามสัจพจน์เพิ่มขึ้นมา โดยผลลัพธ์ของการแปลงแสดงในรูปที่ ค-42

```
\begin{zed}
REAL == \{ x,y : \num @ (x,y) \} \\
CHAR == 0 \upto 255
\end{zed}

\syndef{RPlus}{infun3}{RPlus}
\syndef{RMinus}{infun3}{RMinus}
\syndef{RMul}{infun3}{RMul}
\syndef{RDiv}{infun3}{RDiv}
\syndef{RG}{inrel}{RG}
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
\syndef{RL}{inrel}{RL}
\syndef{RGE}{inrel}{RGE}
\syndef{RLE}{inrel}{RLE}

\begin{axdef}
\_ RPlus \_ : REAL \cross REAL \fun REAL \\
\_ RMinus \_ : REAL \cross REAL \fun REAL \\
\_ RMul \_ : REAL \cross REAL \fun REAL \\
\_ RDiv \_ : REAL \cross REAL \fun REAL \\
\_ RG \_ : REAL \rel REAL \\
\_ RL \_ : REAL \rel REAL \\
\_ RGE \_ : REAL \rel REAL \\
\_ RLE \_ : REAL \rel REAL \\
\where
\forall x,y : REAL @ x RPlus y = (first(x) + first(y),second(x)) \iff  second(x) = second(y) \\
\forall x,y : REAL | second(x) = second(y) @ x RMinus y = (first(x) - first(y),second(x)) \\
\forall x,y : REAL @ x RMul y = (first(x) * first(y),second(x) + second(y)) \\
\forall x,y : REAL @ x RDiv y = (first(x) \div first(y),second(x) - second(y)) \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) > second(y)) \lor
        (second(x) = second(y) \land first(x) > first(y)))          \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) < second(y)) \lor
        (second(x) = second(y) \land first(x) < first(y)))          \\
\forall x,y : REAL @ (x RG y) \iff ((second(x) \geq second(y)) \lor
        (second(x) = second(y) \land first(x) \geq first(y)))       \\
\forall x,y : REAL @ (x RL y) \iff ((second(x) \leq second(y)) \lor
        (second(x) = second(y) \land first(x) \leq first(y)))
\end{axdef}

\begin{axdef}
i2r : \num \fun REAL
\where
\forall x : \num @ i2r(x) = (x,0)
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
\end{axdef}

\begin{axdef}
Global\_0\_m, Global\_0\_m_0 : \num \\
Global\_0\_l, Global\_0\_l_0 : \num \\
Global\_0\_k, Global\_0\_k_0 : \num \\
Global\_0\_j, Global\_0\_j_0 : \num \\
Global\_0\_i, Global\_0\_i_0 : \num \\
\where
Global\_0\_m_0 = Global\_0\_m    \also
Global\_0\_l_0 = Global\_0\_l    \also
Global\_0\_k_0 = Global\_0\_k    \also
Global\_0\_j_0 = Global\_0\_j    \also
Global\_0\_i_0 = Global\_0\_i    \also
\end{axdef}

\begin{schema}{SwitchTest}
SwitchTest\_0\_ch? : CHAR        \\
SwitchTest\_result! : \num \\
SwitchTest\_0\_b, SwitchTest\_0\_c : \nat    \\
SwitchTest\_0\_ch_0, SwitchTest\_0\_ch' : CHAR \\
SwitchTest\_0\_c_0, SwitchTest\_0\_c_1, SwitchTest\_0\_c' : \nat \\
SwitchTest\_0\_b_0, SwitchTest\_0\_b_1, SwitchTest\_0\_b_2, SwitchTest\_0\_b_3, SwitchTest\_0\_b'
: \nat \\
\where
SwitchTest\_0\_ch_0 = SwitchTest\_0\_ch?  \also
SwitchTest\_0\_b_0 = SwitchTest\_0\_b     \also
SwitchTest\_0\_c_0 = SwitchTest\_0\_c     \also
SwitchTest\_0\_c_1 = 5    \also
(SwitchTest\_0\_ch_0 = 97)        \land
(SwitchTest\_0\_b_1 = 1)  \land
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
(SwitchTest\_0\_b_2 = (SwitchTest\_0\_b_1+5)) \land (SwitchTest\_0\_b_3 = SwitchTest\_0\_b_2) \lor
(\lnot (SwitchTest\_0\_ch_0 = 97)   \land
((SwitchTest\_0\_ch_0 = 98)         \land
(SwitchTest\_0\_b_1 = 2)  \land
(SwitchTest\_0\_b_2 = (SwitchTest\_0\_b_1+5))       \land
(SwitchTest\_0\_b_3 = (SwitchTest\_0\_b_2+1)) \lor (\lnot (SwitchTest\_0\_ch_0 = 98)    \land
(((SwitchTest\_0\_ch_0 = 99) \lor (SwitchTest\_0\_ch_0 = 100))          \land
(SwitchTest\_0\_b_1 = 3) \lor (\lnot ((SwitchTest\_0\_ch_0 = 99) \lor (SwitchTest\_0\_ch_0 = 100))
\land
((SwitchTest\_0\_b_1 = 4))) \land (SwitchTest\_0\_b_3 = SwitchTest\_0\_b_1)))))          \also
SwitchTest\_result! = SwitchTest\_0\_b_3 \also
SwitchTest\_0\_ch' = SwitchTest\_0\_ch_0 \also
SwitchTest\_0\_c' = SwitchTest\_0\_c_1 \also
SwitchTest\_0\_b' = SwitchTest\_0\_b_3
\end{schema}

\begin{schema} {IFTest}
IFTest\_0\_a? : \num        \\
IFTest\_result! : \num      \\
IFTest\_0\_y : \num         \\
IFTest\_0\_a_0, IFTest\_0\_a' : \num \\
IFTest\_0\_y_0, IFTest\_0\_y_1, IFTest\_0\_y' : \num \\
\where
IFTest\_0\_a_0 = IFTest\_0\_a?      \also
IFTest\_0\_y_0 = IFTest\_0\_y       \also
(IFTest\_0\_a_0 < 0)        \land
(IFTest\_0\_y_1 = 0) \lor (\lnot (IFTest\_0\_a_0 < 0)   \land
((IFTest\_0\_y_1 = 1)))     \also
IFTest\_result! = IFTest\_0\_y_1 \also
IFTest\_0\_a' = IFTest\_0\_a_0 \also
IFTest\_0\_y' = IFTest\_0\_y_1
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
\end{schema}

\begin{schema}{copy}
copy\_0\_x? : \num          \\
copy\_result! : \num        \\
copy\_0\_a, copy\_0\_y : \num       \\
copy\_0\_x_0, copy\_0\_x' : \num \\
copy\_0\_y_0, copy\_0\_y_1, copy\_0\_y_2, copy\_0\_y_3, copy\_0\_y' : \num \\
copy\_0\_a_0, copy\_0\_a_1, copy\_0\_a_2, copy\_0\_a_3, copy\_0\_a' : \num \\
\where
copy\_0\_x_0 = copy\_0\_x?          \also
copy\_0\_a_0 = copy\_0\_a           \also
copy\_0\_y_0 = copy\_0\_y           \also
copy\_0\_a_1 = copy\_0\_x_0         \also
copy\_0\_y_1 = 0 \also
copy\_0\_y_2 = (copy\_0\_y_1+1)   \also
copy\_0\_a_2 = (copy\_0\_a_1-1)   \also
\lnot(copy\_0\_a_3 \neq 0) \land ( ( copy\_0\_y_3 = copy\_0\_x_0 - copy\_0\_a_3 ) )        \also
copy\_result! = copy\_0\_y_3 \also
copy\_0\_x' = copy\_0\_x_0 \also
copy\_0\_y' = copy\_0\_y_3 \also
copy\_0\_a' = copy\_0\_a_3
\end{schema}

\begin{schema}{multi}
multi\_0\_x? : \num         \\
multi\_0\_y? : \num         \\
multi\_result! : \num       \\
multi\_0\_a, multi\_0\_z : \num      \\
multi\_0\_y_0, multi\_0\_y_1, multi\_0\_y' : \num \\
multi\_0\_x_0, multi\_0\_x' : \num \\
multi\_0\_z_0, multi\_0\_z_1, multi\_0\_z_2, multi\_0\_z' : \num \\
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
multi\_0\_a_0, multi\_0\_a_1, multi\_0\_a_2, multi\_0\_a' : \num \\
\where
multi\_0\_x_0 = multi\_0\_x?        \also
multi\_0\_y_0 = multi\_0\_y?        \also
multi\_0\_a_0 = multi\_0\_a         \also
multi\_0\_z_0 = multi\_0\_z         \also
multi\_0\_z_1 = 0 \also
multi\_0\_a_1 = 0 \also
\not(multi\_0\_a_2 \neq multi\_0\_y_1) \land ( ( multi\_0\_z_2 = multi\_0\_x_0 * multi\_0\_a_2 ) ) \also
multi\_result! = multi\_0\_z_2 \also
multi\_0\_y' = multi\_0\_y_1 \also
multi\_0\_x' = multi\_0\_x_0 \also
multi\_0\_z' = multi\_0\_z_2 \also
multi\_0\_a' = multi\_0\_a_2
\end{schema}

\begin{schema}{div}
div\_0\_x? : \num \\
div\_0\_y? : \num \\
div\_result! : \num          \\
div\_0\_r, div\_0\_d : \num \\
div\_0\_y_0, div\_0\_y' : \num \\
div\_0\_x_0, div\_0\_x' : \num \\
div\_0\_d_0, div\_0\_d_1, div\_0\_d_2, div\_0\_d' : \num \\
div\_0\_r_0, div\_0\_r_1, div\_0\_r_2, div\_0\_r' : \num \\
\where
div\_0\_x_0 = div\_0\_x?  \also
div\_0\_y_0 = div\_0\_y?  \also
div\_0\_r_0 = div\_0\_r   \also
div\_0\_d_0 = div\_0\_d   \also
div\_0\_r_1 = div\_0\_x_0 \also
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
div\_0\_d_1 = 0  \also
\lnot(div\_0\_r_2 \geq 0) \land ( ( div\_0\_d_2 * div\_0\_y_0 + div\_0\_r_2 = div\_0\_x_0 ) )  \also
div\_result! = div\_0\_d_2 \also
div\_0\_y' = div\_0\_y_0 \also
div\_0\_x' = div\_0\_x_0 \also
div\_0\_d' = div\_0\_d_2 \also
div\_0\_r' = div\_0\_r_2
\end{schema}

\begin{schema}{mid}
mid\_0\_a? : \nat \\
mid\_0\_b? : \nat \\
mid\_result! : \nat \\
mid\_0\_i : \num \\
mid\_0\_b_0, mid\_0\_b' : \nat \\
mid\_0\_a_0, mid\_0\_a' : \nat \\
mid\_0\_i_0, mid\_0\_i_1, mid\_0\_i' : \num \\
\where
mid\_0\_a_0 = mid\_0\_a? \also
mid\_0\_b_0 = mid\_0\_b? \also
mid\_0\_i_0 = mid\_0\_i  \also
mid\_0\_i_1 = ((mid\_0\_a_0+mid\_0\_b_0) \div 2)  \also
mid\_result! = mid\_0\_i_1 \also
mid\_0\_b' = mid\_0\_b_0 \also
mid\_0\_a' = mid\_0\_a_0 \also
mid\_0\_i' = mid\_0\_i_1
\end{schema}

\begin{schema}{compare}
compare\_0\_a? : \num  \\
compare\_0\_b? : \num  \\
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
compare\_result! : \num    \\
compare\_0\_ret : \num     \\
compare\_0\_b_0, compare\_0\_b' : \num \\
compare\_0\_a_0, compare\_0\_a' : \num \\
compare\_0\_ret_0, compare\_0\_ret_1, compare\_0\_ret' : \num \\
\where
compare\_0\_a_0 = compare\_0\_a? \also
compare\_0\_b_0 = compare\_0\_b?\also
compare\_0\_ret_0 = compare\_0\_ret          \also
(compare\_0\_a_0 = compare\_0\_b_0)         \land
(compare\_0\_ret_1 = 0) \lor (\lnot (compare\_0\_a_0 = compare\_0\_b_0)          \land
((compare\_0\_a_0 < compare\_0\_b_0)        \land
(compare\_0\_ret_1 = (-1)) \lor (\lnot (compare\_0\_a_0 < compare\_0\_b_0)       \land
((compare\_0\_ret_1 = 1)))))          \also
compare\_result! = compare\_0\_ret_1 \also
compare\_0\_b' = compare\_0\_b_0 \also
compare\_0\_a' = compare\_0\_a_0 \also
compare\_0\_ret' = compare\_0\_ret_1
\end{schema}

\begin{schema}{main}
main\_0\_x? : \num        \\
main\_0\_y? : \num        \\
main\_result! : \num      \\
main\_0\_ret : \num       \\
compare          \\
mid              \\
div              \\
multi            \\
copy             \\
IFTest           \\
SwitchTest               \\
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
main\_0\_y_0, main\_0\_y' : \num \\
main\_0\_x_0, main\_0\_x' : \num \\
main\_0\_ret_0, main\_0\_ret_1, main\_0\_ret' : \num \\
\where
main\_0\_x_0 = main\_0\_x?          \also
main\_0\_y_0 = main\_0\_y?          \also
main\_0\_ret_0 = main\_0\_ret       \also
compare\_0\_a? = main\_0\_x_0               \also
compare\_0\_b? = main\_0\_y_0       \also
Global\_0\_i_0 = compare\_result!   \also
mid\_0\_a? = main\_0\_x_0                   \also
mid\_0\_b? = main\_0\_y_0           \also
Global\_0\_j_0 = mid\_result!       \also
div\_0\_x? = main\_0\_x_0           \also
div\_0\_y? = main\_0\_y_0\also
Global\_0\_k_0 = div\_result!       \also
multi\_0\_x? = Global\_0\_k_0               \also
multi\_0\_y? = 5   \also
Global\_0\_k_0 = multi\_result!     \also
copy\_0\_x? = Global\_0\_k_0        \also
Global\_0\_l_0 = copy\_result!      \also
IFTest\_0\_a? = Global\_0\_l_0      \also
Global\_0\_k_0 = IFTest\_result!    \also
SwitchTest\_0\_ch? = 98    \also
Global\_0\_m_0 = SwitchTest\_result!         \also
main\_0\_ret_1 =
((((Global\_0\_i_0+Global\_0\_j_0)+Global\_0\_k_0)+Global\_0\_l_0)+Global\_0\_m_0)  \also
main\_result! = main\_0\_ret_1 \also
main\_0\_y' = main\_0\_y_0 \also
main\_0\_x' = main\_0\_x_0 \also
main\_0\_ret' = main\_0\_ret_1
\end{schema}
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
\begin{schema}{swap}
swap\_0\_a? : REAL        \\
swap\_0\_b? : REAL        \\
swap\_result! : \num      \\
swap\_0\_temp : REAL      \\
swap\_0\_b_0, swap\_0\_b_1, swap\_0\_b' : REAL \\
swap\_0\_a_0, swap\_0\_a_1, swap\_0\_a' : REAL \\
swap\_0\_temp_0, swap\_0\_temp_1, swap\_0\_temp' : REAL \\
\where
swap\_0\_a_0 = swap\_0\_a?          \also
swap\_0\_b_0 = swap\_0\_b?          \also
swap\_0\_temp_0 = swap\_0\_temp \also
swap\_0\_temp_1 = swap\_0\_a_0   \also
swap\_0\_a_1 = swap\_0\_b_0        \also
swap\_0\_b_1 = swap\_0\_temp_1   \also
swap\_result! = 0 \also
swap\_0\_b' = swap\_0\_b_1 \also
swap\_0\_a' = swap\_0\_a_1 \also
swap\_0\_temp' = swap\_0\_temp_1
\end{schema}

\begin{schema}{successor}
successor\_0\_x? : \num    \\
successor\_result! : \num   \\
successor\_0\_a, successor\_0\_y : \num      \\
successor\_0\_x_0, successor\_0\_x' : \num \\
successor\_0\_y_0, successor\_0\_y_1, successor\_0\_y_2, successor\_0\_y' : \num \\
successor\_0\_a_0, successor\_0\_a_1, successor\_0\_a' : \num \\
\where
successor\_0\_x_0 = successor\_0\_x?          \also
successor\_0\_a_0 = successor\_0\_a           \also
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

```
successor\_0\_y_0 = successor\_0\_y          \also
successor\_0\_a_1 = (successor\_0\_x_0+1)  \also
successor\_0\_y_1 = successor\_0\_a_1       \also
((successor\_0\_a_1-1) = 0)          \land
(successor\_0\_y_2 = 1) \lor (\lnot((successor\_0\_a_1-1) = 0)) \also
successor\_result! = successor\_0\_y_2 \also
successor\_0\_x' = successor\_0\_x_0 \also
successor\_0\_y' = successor\_0\_y_2 \also
successor\_0\_a' = successor\_0\_a_1
\end{schema}
```

รูปที่ ค-42 ผลลัพธ์ของการแปลงโปรแกรมต้นฉบับของกรณีทดสอบที่ 9

ข้อกำหนดเซตที่ได้จากการแปลงจะถูกนำไปตรวจสอบด้วยเครื่องมือ Z/EVES โดยจะทำการตรวจสอบใน 3 ลักษณะคือ

1) ทำการตรวจสอบวากยสัมพันธ์ของข้อกำหนดที่ได้ว่าถูกต้องตามหลักวากยสัมพันธ์ของสัญกรณ์เซตหรือไม่ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-43

```
Reading file D:\Thesis\Application\c2z\Data\Case9.zed
definition of REAL
definition of CHAR
declaration of (\_ RDiv \_), (\_ RG \_), (\_ RGE \_), (\_ RL \_), (\_ RLE \_),
(\_ RMinus \_), (\_ RMul \_), (\_ RPlus \_)
... theorem unnamed\$n3\$domainCheck
... axiom RPlus\$declaration
... axiom RMinus\$declaration
... axiom RMul\$declaration
... axiom RDiv\$declaration
... axiom RG\$declaration
... axiom RL\$declaration
... axiom RGE\$declaration
... axiom RLE\$declaration
... axiom axiom\$4
... axiom axiom\$5
```

รูปที่ ค-43 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 9

... axiom axiom\$6

... axiom axiom\$7

... axiom axiom\$8

... axiom axiom\$9

... axiom axiom\$10

... axiom axiom\$11

declaration of i2r

... axiom i2r\$declaration

... axiom axiom\$12

declaration of Global_0_i, Global_0_i_0, Global_0_j, Global_0_j_0,
Global_0_k, Global_0_k_0, Global_0_l, Global_0_l_0, Global_0_m,
Global_0_m_0

... axiom Global_0_m\$declaration

... axiom Global_0_m\$0\$declaration

... axiom Global_0_l\$declaration

... axiom Global_0_l\$0\$declaration

... axiom Global_0_k\$declaration

... axiom Global_0_k\$0\$declaration

... axiom Global_0_j\$declaration

... axiom Global_0_j\$0\$declaration

... axiom Global_0_i\$declaration

... axiom Global_0_i\$0\$declaration

... axiom axiom\$13

... axiom axiom\$14

... axiom axiom\$15

... axiom axiom\$16

... axiom axiom\$17

schema SwitchTest

... axiom SwitchTest\$declarationPart

schema IFTest

... axiom IFTest\$declarationPart

schema copy

รูปที่ ค-43 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 9

```
... axiom copy\$declarationPart
schema multi
... axiom multi\$declarationPart
schema div
... axiom div\$declarationPart
schema mid
... theorem mid\$domainCheck
... axiom mid\$declarationPart
schema compare
... axiom compare\$declarationPart
schema main
... axiom main\$declarationPart
schema swap
... axiom swap\$declarationPart
schema successor
... axiom successor\$declarationPart
Done.
```

รูปที่ ค-43 ผลการตรวจสอบวากยสัมพันธ์ข้อกำหนดเซตของกรณีทดสอบที่ 9

2) การตรวจสอบโดยอาศัยทฤษฎีทางคณิตศาสตร์ซึ่งผลลัพธ์ที่ได้นั้นแสดงอยู่ในรูปที่ ค-44

```
Proving gives ...

    main\_0\_x? \in \num \\
\land main\_0\_y? \in \num \\
\land main\_0\_ret \in \num \\
\land main\_0\_ret_0 = main\_0\_ret \\
\land main\_result! = main\_0\_ret_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land compare\_0\_a? = main\_0\_x_0 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land compare\_0\_b? = main\_0\_y_0 \\
\land compare\_0\_ret_1 \in \num \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land compare\_result! = compare\_0\_ret_1 \\
\land Global\_0\_i_0 = compare\_result! \\
\land compare\_0\_ret \in \num \\
\land compare\_0\_b_0 = compare\_0\_b? \\
\land compare\_0\_b' = compare\_0\_b_0 \\
\land compare\_0\_a_0 = compare\_0\_a? \\
\land compare\_0\_a' = compare\_0\_a_0 \\
\land compare\_0\_ret_0 = compare\_0\_ret \\
\land compare\_0\_ret' = Global\_0\_i_0 \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land (          compare\_0\_a_0 = compare\_0\_b_0 \\
        \land Global\_0\_i_0 = 0 \\
    \lor      Global\_0\_i_0 = \neg 1 \\
        \land \lnot compare\_0\_a_0 = compare\_0\_b_0 \\
        \land compare\_0\_a_0 < compare\_0\_b_0 \\
    \lor      Global\_0\_i_0 = 1 \\
        \land \lnot compare\_0\_a_0 < compare\_0\_b_0)
\land mid\_0\_a? = main\_0\_x_0 \\
\land mid\_0\_b? = main\_0\_y_0 \\
\land mid\_0\_b_0 = mid\_0\_b? \\
\land mid\_0\_a_0 = mid\_0\_a? \\
\land mid\_0\_i_1 = (mid\_0\_a_0 + mid\_0\_b_0) \div 2 \\
\land mid\_result! = (mid\_0\_a_0 + mid\_0\_b_0) \div 2 \\
\land mid\_0\_i \in \num \\
\land mid\_0\_b' = mid\_0\_b_0 \\
\land mid\_0\_a' = mid\_0\_a_0 \\
\land mid\_0\_i_0 = mid\_0\_i \\
\land mid\_0\_i' = Global\_0\_j_0 \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x' = main\_0\_x_0 \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land main\_0\_ret' = main\_0\_ret_0 \\
\land Global\_0\_j_0 = (mid\_0\_a_0 + mid\_0\_b_0) \div 2 \\
\land mid\_0\_a? \geq 0 \\
\land mid\_0\_b? \geq 0
\land main\_0\_x_0 = main\_0\_x? \\
\land div\_0\_x? = main\_0\_x_0 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land div\_0\_y? = main\_0\_y_0 \\
\land div\_0\_d_2 \in \num \\
\land div\_result! = div\_0\_d_2 \\
\land Global\_0\_k_0 = div\_result! \\
\land div\_0\_r \in \num \\
\land div\_0\_d \in \num \\
\land div\_0\_y_0 = div\_0\_y? \\
\land div\_0\_y' = div\_0\_y_0 \\
\land div\_0\_x_0 = div\_0\_x? \\
\land div\_0\_r_2 \in \num \\
\land div\_0\_r_2 + div\_0\_y_0 * Global\_0\_k_0 = div\_0\_x_0 \\
\land div\_0\_x' = div\_0\_r_2 + div\_0\_y_0 * Global\_0\_k_0 \\
\land div\_0\_d_0 = div\_0\_d \\
\land div\_0\_d_1 = 0 \\
\land div\_0\_d' = Global\_0\_k_0 \\
\land div\_0\_r_0 = div\_0\_r \\
\land div\_0\_r_1 = div\_0\_r_2 + div\_0\_y_0 * Global\_0\_k_0 \\
\land div\_0\_r' = div\_0\_r_2 \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land \lnot div\_0\_r_2 \geq 0
\land multi\_0\_x? = Global\_0\_k_0 \\
\land multi\_0\_y? = 5 \\
\land multi\_0\_x_0 = Global\_0\_k_0 \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land multi\_0\_y_1 \in \num \\
\land multi\_0\_a_2 = multi\_0\_y_1 \\
\land multi\_0\_z_2 = Global\_0\_k_0 * multi\_0\_a_2 \\
\land multi\_result! = Global\_0\_k_0 * multi\_0\_a_2 \\
\land multi\_0\_a \in \num \\
\land multi\_0\_z \in \num \\
\land multi\_0\_y_0 = 5 \\
\land multi\_0\_y' = multi\_0\_y_1 \\
\land multi\_0\_x' = Global\_0\_k_0 \\
\land multi\_0\_z_0 = multi\_0\_z \\
\land multi\_0\_z_1 = 0 \\
\land multi\_0\_z' = Global\_0\_k_0 \\
\land multi\_0\_a_0 = multi\_0\_a \\
\land multi\_0\_a_1 = 0 \\
\land multi\_0\_a' = multi\_0\_a_2 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land Global\_0\_k_0 = Global\_0\_k_0 * multi\_0\_a_2
\land copy\_0\_x? = Global\_0\_k_0 \\
\land copy\_0\_x_0 = Global\_0\_k_0 \\
\land copy\_0\_y_3 = Global\_0\_k_0 \\
\land copy\_result! = Global\_0\_k_0 \\
\land copy\_0\_a \in \num \\
\land copy\_0\_y \in \num \\
\land copy\_0\_x' = Global\_0\_k_0 \\
\land copy\_0\_y_0 = copy\_0\_y \\
\land copy\_0\_y_1 = 0 \\
\land copy\_0\_y_2 = 1 \\
\land copy\_0\_y' = Global\_0\_k_0 \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land copy\_0\_a_0 = copy\_0\_a \\
\land copy\_0\_a_1 = Global\_0\_k_0 \\
\land copy\_0\_a_2 = \neg 1 + Global\_0\_k_0 \\
\land copy\_0\_a_3 = 0 \\
\land copy\_0\_a' = 0 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land Global\_0\_1_0 = Global\_0\_k_0
\land IFTest\_0\_a? = Global\_0\_1_0 \\
\land IFTest\_0\_y_1 \in \num \\
\land IFTest\_result! = IFTest\_0\_y_1 \\
\land Global\_0\_k_0 = IFTest\_result! \\
\land IFTest\_0\_y \in \num \\
\land IFTest\_0\_a_0 = Global\_0\_1_0 \\
\land IFTest\_0\_a' = Global\_0\_1_0 \\
\land IFTest\_0\_y_0 = IFTest\_0\_y \\
\land IFTest\_0\_y' = Global\_0\_k_0 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land (         Global\_0\_k_0 = 0 \\
        \land Global\_0\_1_0 < 0 \\
    \lor      Global\_0\_k_0 = 1 \\
        \land \lnot Global\_0\_1_0 < 0)
\land SwitchTest\_0\_b_3 \in \num \\
\land SwitchTest\_result! = SwitchTest\_0\_b_3 \\
\land Global\_0\_m_0 = SwitchTest\_result! \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land SwitchTest\_0\_b \in \num \\
\land SwitchTest\_0\_c \in \num \\
\land SwitchTest\_0\_c_0 = SwitchTest\_0\_c \\
\land SwitchTest\_0\_c_1 = 5 \\
\land SwitchTest\_0\_c' = 5 \\
\land SwitchTest\_0\_b_0 = SwitchTest\_0\_b \\
\land SwitchTest\_0\_b_1 \in \num \\
\land SwitchTest\_0\_b_2 \in \num \\
\land SwitchTest\_0\_b' = Global\_0\_m_0 \\
\land SwitchTest\_0\_ch? = 98 \\
\land SwitchTest\_0\_ch_0 = 98 \\
\land SwitchTest\_0\_ch' = 98 \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land SwitchTest\_0\_b \geq 0 \\
\land SwitchTest\_0\_c \geq 0 \\
\land SwitchTest\_0\_b_1 \geq 0 \\
\land SwitchTest\_0\_b_2 \geq 0 \\
\land Global\_0\_m_0 \geq 0 \\
\land (          SwitchTest\_0\_b_1 = 2 \\
        \land SwitchTest\_0\_b_2 = 7 \\
        \land Global\_0\_m_0 = 8 \\
    \lor      SwitchTest\_0\_b_1 = 4 \\
        \land Global\_0\_m_0 = 4)
\land main\_result! = Global\_0\_m_1 \\
\land main\_0\_ret \in \num \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

```
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret_0 = main\_0\_ret \\
\land main\_0\_ret_1 = Global\_0\_m_1 \\
\land main\_0\_ret' = Global\_0\_m_1
```

รูปที่ ค-44 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยทฤษฎีทางคณิตศาสตร์ของกรณีทดสอบที่ 9

3) การตรวจสอบโดยอาศัยตัวอย่าง สำหรับขั้นตอนนี้จะทำการระบุค่าให้กับตัวแปรของข้อกำหนดเพื่อดูผลลัพธ์ของการทำงานโดยในที่นี้ทำการระบุค่าตัวแปร Test_0_a? เท่ากับ 2 ซึ่งผลลัพธ์ที่ได้จากการตรวจสอบแสดงอยู่ในรูปที่ ค-45

```
Proving gives ...

    main\_0\_ret \in \num \\
\land main\_0\_ret_0 = main\_0\_ret \\
\land main\_result! = main\_0\_ret_0 \\
\land compare\_0\_a? = 5 \\
\land compare\_0\_b? = 10 \\
\land compare\_0\_ret_1 \in \num \\
\land compare\_result! = compare\_0\_ret_1 \\
\land Global\_0\_i_0 = compare\_result! \\
\land compare\_0\_ret \in \num \\
\land compare\_0\_b_0 = 10 \\
\land compare\_0\_b' = 10 \\
\land compare\_0\_a_0 = 5 \\
\land compare\_0\_a' = 5 \\
\land compare\_0\_ret_0 = compare\_0\_ret \\
\land compare\_0\_ret' = Global\_0\_i_0 \\
\land main\_0\_y_0 = 10 \\
\land main\_0\_y' = 10 \\
\land main\_0\_x_0 = 5 \\
\land main\_0\_x' = 5 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land Global\_0\_i_0 = \neg 1
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

```
\land mid\_0\_a? = 5 \\
\land mid\_0\_b? = 10 \\
\land mid\_0\_b_0 = 10 \\
\land mid\_0\_a_0 = 5 \\
\land mid\_0\_i_1 = 7 \\
\land mid\_result! = 7 \\
\land mid\_0\_i \in \num \\
\land mid\_0\_b' = 10 \\
\land mid\_0\_a' = 5 \\
\land mid\_0\_i_0 = mid\_0\_i \\
\land mid\_0\_i' = Global\_0\_j_0 \\
\land Global\_0\_j_0 = 7
\land div\_0\_x? = 5 \\
\land div\_0\_y? = 10 \\
\land div\_0\_d_2 \in \num \\
\land div\_result! = div\_0\_d_2 \\
\land Global\_0\_k_0 = div\_result! \\
\land div\_0\_r \in \num \\
\land div\_0\_d \in \num \\
\land div\_0\_y_0 = 10 \\
\land div\_0\_y' = 10 \\
\land div\_0\_x_0 = 5 \\
\land div\_0\_r_2 \in \num \\
\land div\_0\_x' = 5 \\
\land div\_0\_d_0 = div\_0\_d \\
\land div\_0\_d_1 = 0 \\
\land div\_0\_d' = Global\_0\_k_0 \\
\land div\_0\_r_0 = div\_0\_r \\
\land div\_0\_r_1 = 5 \\
\land div\_0\_r' = div\_0\_r_2 \\
\land div\_0\_r_2 + 10 * Global\_0\_k_0 = 5 \\
\land \lnot div\_0\_r_2 \geq 0
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

```
\land mid\_0\_a? = 5 \\
\land mid\_0\_b? = 10 \\
\land mid\_0\_b_0 = 10 \\
\land mid\_0\_a_0 = 5 \\
\land mid\_0\_i_1 = 7 \\
\land mid\_result! = 7 \\
\land mid\_0\_i \in \num \\
\land mid\_0\_b' = 10 \\
\land mid\_0\_a' = 5 \\
\land mid\_0\_i_0 = mid\_0\_i \\
\land mid\_0\_i' = Global\_0\_j_0 \\
\land Global\_0\_j_0 = 7
\land div\_0\_x? = 5 \\
\land div\_0\_y? = 10 \\
\land div\_0\_d_2 \in \num \\
\land div\_result! = div\_0\_d_2 \\
\land Global\_0\_k_0 = div\_result! \\
\land div\_0\_r \in \num \\
\land div\_0\_d \in \num \\
\land div\_0\_y_0 = 10 \\
\land div\_0\_y' = 10 \\
\land div\_0\_x_0 = 5 \\
\land div\_0\_r_2 \in \num \\
\land div\_0\_x' = 5 \\
\land div\_0\_d_0 = div\_0\_d \\
\land div\_0\_d_1 = 0 \\
\land div\_0\_d' = Global\_0\_k_0 \\
\land div\_0\_r_0 = div\_0\_r \\
\land div\_0\_r_1 = 5 \\
\land div\_0\_r' = div\_0\_r_2 \\
\land div\_0\_r_2 + 10 * Global\_0\_k_0 = 5 \\
\land \lnot div\_0\_r_2 \geq 0
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

```
\land      multi\_result! = 0 \\
\land multi\_0\_a \in \num \\
\land multi\_0\_z \in \num \\
\land multi\_0\_y_0 = 5 \\
\land multi\_0\_y_1 \in \num \\
\land multi\_0\_y' = multi\_0\_y_1 \\
\land multi\_0\_x_0 = 0 \\
\land multi\_0\_x' = 0 \\
\land multi\_0\_z_0 = multi\_0\_z \\
\land multi\_0\_z_1 = 0 \\
\land multi\_0\_z_2 = 0 \\
\land multi\_0\_z' = 0 \\
\land multi\_0\_a_0 = multi\_0\_a \\
\land multi\_0\_a_1 = 0 \\
\land multi\_0\_a_2 = multi\_0\_y_1 \\
\land multi\_0\_a' = multi\_0\_a_2
\land copy\_0\_x? = Global\_0\_k_0 \\
\land copy\_0\_x_0 = Global\_0\_k_0 \\
\land copy\_0\_y_3 = Global\_0\_k_0 \\
\land copy\_result! = Global\_0\_k_0 \\
\land copy\_0\_a \in \num \\
\land copy\_0\_y \in \num \\
\land copy\_0\_x' = Global\_0\_k_0 \\
\land copy\_0\_y_0 = copy\_0\_y \\
\land copy\_0\_y_1 = 0 \\
\land copy\_0\_y_2 = 1 \\
\land copy\_0\_y' = Global\_0\_k_0 \\
\land copy\_0\_a_0 = copy\_0\_a \\
\land copy\_0\_a_1 = Global\_0\_k_0 \\
\land copy\_0\_a_2 = \neg 1 + Global\_0\_k_0 \\
\land copy\_0\_a_3 = 0 \\
\land copy\_0\_a' = 0 \\
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเซตโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

```
\land main\_0\_y_0 = 10 \\
\land main\_0\_y' = 10 \\
\land main\_0\_x_0 = 5 \\
\land main\_0\_x' = 5 \\
\land main\_0\_ret' = main\_0\_ret_0 \\
\land Global\_0\_l_0 = Global\_0\_k_0
\land IFTest\_result! = 1 \\
\land IFTest\_0\_y \in \num \\
\land IFTest\_0\_a_0 = 0 \\
\land IFTest\_0\_a' = 0 \\
\land IFTest\_0\_y_0 = IFTest\_0\_y \\
\land IFTest\_0\_y_1 = 1 \\
\land IFTest\_0\_y' = 1
\land Global\_0\_k_3 =  IFTest\_result!
\land SwitchTest\_0\_b_3 \in \num \\
\land SwitchTest\_result! = SwitchTest\_0\_b_3 \\
\land SwitchTest\_0\_b \in \num \\
\land SwitchTest\_0\_c \in \num \\
\land SwitchTest\_0\_c_0 = SwitchTest\_0\_c \\
\land SwitchTest\_0\_c_1 = 5 \\
\land SwitchTest\_0\_c' = 5 \\
\land SwitchTest\_0\_b_0 = SwitchTest\_0\_b \\
\land SwitchTest\_0\_b_1 \in \num \\
\land SwitchTest\_0\_b_2 \in \num \\
\land SwitchTest\_0\_b' = SwitchTest\_0\_b_3 \\
\land SwitchTest\_0\_ch_0 = 98 \\
\land SwitchTest\_0\_ch' = 98 \\
\land SwitchTest\_0\_b \geq 0 \\
\land SwitchTest\_0\_c \geq 0 \\
\land SwitchTest\_0\_b_1 \geq 0 \\
\land SwitchTest\_0\_b_2 \geq 0 \\
\land SwitchTest\_0\_b_3 \geq 0 \\
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

```
\land SwitchTest\_0\_b_3 = 8 \\
\land Global\_0\_m_1 = SwtichTest\_0\_result!
\land main\_result! = 15 \\
\land main\_0\_ret \in \num \\
\land main\_0\_y_0 = main\_0\_y? \\
\land main\_0\_y' = main\_0\_y_0 \\
\land main\_0\_x_0 = main\_0\_x? \\
\land main\_0\_x' = main\_0\_x_0 \\
\land main\_0\_ret_0 = main\_0\_ret \\
\land main\_0\_ret_1 = 15 \\
\land main\_0\_ret' = 15 \\
```

รูปที่ ค-45 ผลการตรวจสอบข้อกำหนดเชิงโดยอาศัยตัวอย่างของกรณีทดสอบที่ 9

ภาคผนวก ง. ผลงานตีพิมพ์

ผลงานนี้ได้ถูกตีพิมพ์ในงาน The 4th National Computer Science and Engineering Conference (NCSEC 2000) ซึ่งได้จัดขึ้นที่ศูนย์การประชุมแห่งชาติสิริกิติ์ ในวันที่ 16-17 พฤศจิกายน 2543

A Formal Approach to Recover Design Specification from C Program Code

Reungyos Vorajenwanich[1], Wiwat Vatanawood[2] and Wanchai Rivepiboon[3]
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Email: b0485421@student.chula.ac.th[1], wiwat@chula.ac.th[2], wanchai.r@chula.ac.th[3]

Abstract: In software evolution process the consistency of a changing system should be maintained. The usability of the existing system depends on its related up-to-date documents. In this paper is proposed a set of rules for regenerating or recovering design specification of the software system from C program code. Our approach is considered as an alternative approach to synchronize documentation and existing program code during the evolution process of software system.
Key words: Reverse Engineering, Formal Method

1. Introduction

Nowadays, while software requirements have been changing, the replacement of the old system with the new one is not recommended. Practically, software evolution process is preferably considered. The consistency of the changing system is one of the important factors in order to maintain the usability of the system [1]. Thus, generating new up-to-date design specification is an alternative to synchronize documentation and implementation.

The techniques are categorized into two approaches. One is based on semi formal model [2]. The graphical notations are used to represent software specification, which is easily read by human. Unfortunately, the graphical specification is ambiguous, unamenable and difficult to verify. The other is based on formal model [3] that using mathematical notations, which is amenable, unambiguous represent specification, and specification can be verified by mathematical theory that this approach is the more interesting approach.

This paper is organized as follow. Section 2 and 3 describe how to translate C program to Z specification. Section 4 is the conclusion.

2. Data declaration translation

C programming language categorizes data type of variable into two groups. One of them is primitive data type. The other is non-primitive data. In this paper will consider only primitive data type. Primitive data type in C can group into three groups, integer, real, character.

2.1 Integer

In C programming language, an integer variable may be defined as *int*, *short*, and *long*. When source code is translated to Z specification, an integer variable will be defined as type $\mathbb{Z}$ (built-in integer data type in Z notations).

In case of *unsigned int*, *unsigned short*, and *unsigned long*, these positive variables will be defined as type $\mathbb{N}$ (natural number in Z notation).

2.2 Real

Since there is no built-in real data type in Z notation, we decide to represent real number by a tuple of integer.

Let $REAL == \{ x, y : \mathbb{Z} \cdot (x, y) \}$
Where x is the characteristic of real number,
y is the exponential of real number.

In Z notation, numerical operator symbols such as '+' (addition), '-' (substraction), '*' (multiply), '/' (Division) are defined for integer data type. Thus a set of new operations is explicitly defined for REAL data.

In case of mix-mode expression, type conversion process from integer to real is needed. The integer value should be converted to a type REAL before the evaluation of the expression.

2.3 Character

In C programming language, character data type is represented as an integer, which is range between 0 to 255. To formalize character data type into Z specification, we defines a specific CHAR type as

$$CHAR == 0..255$$

3. Operation translation

There are five kinds of statement covered that is Assignment, Alternative, Iterative, Sequence and Function Call Statement. In this paper we use predicate transformer SP proposed in [3] as guideline for translating.

3.1 Assignment statement

Assignment statement in C program can be written in short form of syntax. In our approach, we assume that each assignment statement should be prepared in the full form. However, the statement like 'x = x + 5' is still not valid in the mathematical world because it is not possible that the value of x will be equal to the value of the same x plus 5. Thus a set of dummy variables x_0, x_1 should be defined as the temporary storage of the value x and its changes during expression evaluation.

3.2 Alternative statement

Alternative statement in C programming language has two forms as follows.

```
if (G) {                    if (G) {
    S1          or              S1
}                           } else {
                                S2
                            }
```

where G is guard command of statement such that S_1 is executed only if logical expression G is true. But in the second form if logical expression G is false then statement S_2 is executed.

In the first form of alternative statement, statement S_1 will be executed only if guard command G is true. Thus, this statement will be translated into predicate $(G \wedge S_1) \vee (\neg G)$.

The second form means statement S_1 will be executed only if guard command G is true. Otherwise, statement S_2 will be executed as the alternative. Thus, this statement will be translated into predicate $(G \wedge S_1) \vee (\neg G \wedge S_2)$.

Besides both forms of alternative statement, there is another form of alternative statement that is *switch* statement, which provided multiple branching. To formalize this statement, first of all this statement should be converted to *nested-if* statement then each *if* statement will be translated using *if* rules.

3.3 Sequence Statement

All operations in C program sequential act in statement-by-statement manner. For a given sequence of statements $S_1;S_2;S_3;\ldots;S_n$, the postcondition of statement S_i will be the precondition of statement S_{i+1}. Thus, all postcondition of each statement will be conjuncted.

3.4 Iterative Statement

C programming language has three kinds of iterative construction: *while* loop, *do-while* loop, and *for* loop construction. We assume that *do-while* loop and *for* loop construction can be changed to *while* loop construction. The syntax of *while* loop in C is as follow

```
while (G) { S }
```

Where G is Guard command of loop,
S is statement that executes only if guard command is true.

In translating process of iterative statement, one kind of information should be gathered: loop invariant. Loop invariant is a predicate that is true before and after each iteration of loop. When loop invariant is gathered then *while* loop construct can be translate to a predicate like this $(\neg G \wedge V)$ where G is guard command of loop and V is loop invariant

However when an automated approach is applied to loop construct that has too much complicated. So in this case the domain expert is needed to specify the proper specification of the statement.

In translating another two kinds of iterative statement that are *do-while* loop and *for* loop construct. These statement will be transform to *while* loop then the same translation rule for *while* loop are applied.

3.5. Function Call Statement

C source code can be divided into functions, which perform specific tasks. Each function can invoke other functions and also can communicate to each other by passing parameters. There are three in passing parameters: Call-by-value, Call-by-reference, Call-by-value result. We focus only call by value in this paper.

Each function in C can be translated to a schema one by one. After that if that function has function call statements, schema inclusion will be used by including the name of called schema into calling schema. Moreover, some predicates are needed to be added in calling schema. These predicates will state the parameter passing between the calling schema and called schema.

4.Conclusion

In this paper, we propose a set of rules for translating the C program code to Z specification. These rules help us to regenerate or recover total design specification from C program code. We consider that this technique will be an alternative to maintain the consistency of documents in software evolution process. The consistency of output Z specification has been checked by using automated tool called Z/EVE.

5.Reference

[1] P.O Brian Holt, "System Documentation and System Design: A Good Reason for Design the Manual First", The Institute of Electrical Engineers, Savoy Place, London, 1993

[2] L.Theodoros, H.M.Edwards, A.Bryant, and N.Willis, "ROMEO : Reverse Engineering from OO Source Code to OMT Design", Proceeding of Fifth Working Conference on Reverse Engineering, IEEE, October 1996

[3] G.C. Gannod, "The Application of Formal Methods to the Reverse Engineering of Imperative Program Code", M.S. Thesis. Michigan State University, April 1994.

ประวัติผู้เขียนวิทยานิพนธ์

นายเรืองยศ วรเจนวณิชย์ เกิดเมื่อวันที่ 17 พฤศจิกายน พ.ศ.2519 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิทยาศาสตร์บัณฑิต (วท.บ.) สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร เมื่อปีการศึกษา 2540 และเข้าศึกษาต่อหลักสูตรวิทยาศาสตร์มหาบัณฑิต (วท.ม.) ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2541