การสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา

นายวิสุทธิ์ สุขศรีบางเตย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

Extracting Data Invariants from Promela

Mr. Wisut Suksribangteuy

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Software Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2018

หัวข้อวิทยานิพนธ์	การสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา
โดย	นายวิสุทธิ์ สุขศรีบางเตย
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ)

..... กรรมการ
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.บัณฑิต ฐานะโสภณ)

วิสุทธิ์ สุขศรีบางเตย : การสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา . (Extracting Data Invariants from Promela) อ.ที่ปรึกษาหลัก : รศ. ดร.วิวัฒน์ วัฒนาวุฒิ

ตัวยืนยงชนิดข้อมูลเป็นข้อมูลคงที่ซึ่งจะมีค่าคงอยู่ตลอดการทำงานของโปรแกรม ซึ่งอยู่ในรูปของตัวแปรหรือเงื่อนไข ที่อยู่ในความต้องการที่ได้กำหนดไว้แสดงด้วยสูตรแอลทีแอล ซึ่งระบุความถูกต้องของการทำงานของโปรแกรมที่เกิดขึ้นในปัจจุบัน

โปรแกรมซอฟต์แวร์จำนวนมากในปัจจุบันมุ่งเน้นเรื่องการปรับปรุงคุณภาพซอฟต์แวร์ ซึ่งการทวนสอบโปรแกรมซอฟต์แวร์ก็เป็นสิ่งสำคัญสำหรับองค์กร เนื่องจากการทวนสอบนั้นเป็นหนึ่งในวิธีที่ใช้ในการตรวจสอบความถูกต้องของอัลกอริทึมภายใต้การทำงานของซอฟต์แวร์

ในทางปฏิบัตินั้นการเขียนสูตรแอลทีแอลด้วยตนเองนั้นเป็นสิ่งที่ยาก และมีความซับซ้อนสำหรับโค้ดโปรแกรม โพรเมลาแตกต่างจากภาษาอื่นๆ เพราะโพรเมลาสนับสนุนโปรแกรมที่มีการทำงานหลายอย่างพร้อมกัน บางครั้งก็จะมีการทำงานที่สอดแทรกหรือคาบเกี่ยวกัน ซึ่งในการทำงานหลายอย่างพร้อมๆกัน ส่งผลให้กระทบกับตัวแปรหรือฟังก์ชันอื่นๆ ซึ่งอาจก่อให้เกิดข้อผิดพลาด ทำให้โปรแกรมทำงานได้ไม่ถูกต้อง และคนทั่วไปมักไม่ค่อยตระหนักถึงการทวนสอบความถูกต้องของค่าที่ใช้ในโปรแกรม งานวิจัยนี้จึงได้นำเสนอวิธีการแบ่งส่วนของข้อมูล เพื่อสร้างตัวยืนยงชนิดข้อมูลจากโพรเมลา โดยอยู่ในรูปของสูตรแอลทีแอลขึ้นมาช่วยในการตรวจสอบความถูกต้องในการทำงานของโปรแกรม

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อนิสิต .....

ปีการศึกษา 2561 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

5870962321 : MAJOR SOFTWARE ENGINEERING

KEYWORD: Data Invariants LTL Program SlicingPromela SPIN Formal

Wisut Suksribangteuy : Extracting Data Invariants from Promela. Advisor: Assoc. Prof. Wiwat Vatanawood

Data invariants are constant data that could persist data or control program integrity. They have to be true throughout the program execution. LTL formulae is referring to time for specifying correctness requirement of program, it is possible to prove the correctness of the function for executable program

Many software programs emphasize to improve software quality by verify software program to help organizations. Verification is one of most effective way that provides a correctness of intended algorithms underlying a software with respect to a certain formal specification or property

In practice, writing the LGL formula manually is so difficult and complicate for program source code. Promela is meant to be difference from other languages that it supports the concurrency. Conceptually a program may be thought of as a collection of threads. Several threads may compute values of the same variable. Many of these threads may interleaving others. Some people are not aware the correctness of the value to define program by LTL formulae. This thesis proposes the method to extracting data invariants for generate LTL formulae to verify correctness program

Field of Study: Software Engineering Student's Signature .....

Academic Year: 2018 Advisor's Signature .....

กิตติกรรมประกาศ

ขอกราบขอบพระคุณ รศ. ดร.วิวัฒน์ วัฒนาวุฒิ อาจารย์ที่ปรึกษาโครงงาน เป็นอย่างยิ่งที่ได้ช่วยเหลือในทุกๆด้าน ทั้งให้คำปรึกษา ให้คำแนะนำ และให้แนวทางสำหรับการทำโครงงานมหาบัณฑิต รวมทั้งเป็นผู้ประสานงานให้ความช่วยเหลือแก่นิสิตที่ทำโครงงานทุกคน

ขอกราบขอบพระคุณ รศ.ดร.สมชาย ประสิทธิ์จูตระกูล เป็นประธานกรรมการสอบโครงงานวิทยานิพนธ์และขอความอนุเคราะห์ รศ.ดร.พรศิริ หมื่นไชยศรี และ ผศ. ดร.บัณฑิต ฐานะโสภณ คณะกรรมการคุมสอบโครงงานมหาบัณฑิตเป็นอย่างยิ่ง ที่ได้กรุณาแนะนำแนวทาง รวมถึงการตรวจสอบและแก้ไขโครงงานมหาบัณฑิตนี้

ขอขอบคุณคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ที่ให้คำแนะนำ ความรู้และแนวทางการทำโครงงาน

ขอขอบคุณเพื่อนๆ หลักสูตรวิศวกรรมซอฟต์แวร์ สำหรับช่วยเหลือ และให้คำแนะนำในการจัดทำโครงงานมหาบัณฑิต

สุดท้ายนี้ ขอกราบขอบพระคุณ บิดา มารดา รวมถึงสมาชิกในครอบครัวที่ให้การสนับสนุนและให้กำลังใจที่ดีเสมอมา

วิสุทธิ์ สุขศรีบางเตย

สารบัญ

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ตัวยืนยง (Invariants) [1] คือ ค่าหรือเงื่อนไขที่ไม่เปลี่ยนแปลงตลอดการทำงานของโปรแกรม โดยอ้างอิงจากข้อกำหนดที่ได้ระบุไว้ ซึ่งแสดงถึงความคงที่ ไม่เปลี่ยนแปลง ในทางคอมพิวเตอร์ คือ เงื่อนไขที่จะต้องเป็นจริงเสมอตลอดการทำงานของโปรแกรม แบ่งออกเป็นหลายประเภท เช่น ตัวยืนยงประเภทชั้น (Class Invariants), ตัวยืนยงประเภทการวนซ้ำ (Loop Invariants) หรือตัวยืนยงชนิดข้อมูล (Data Invariants) โดยตัวยืนยงที่ใช้ในงานวิจัยนี้ได้แก่ ตัวยืนยงชนิดข้อมูล คือ ข้อมูลที่อยู่ในรูปขอบตัวแปร หรือเงื่อนไขที่ใช้ในการกำหนดการทำงานของซอฟต์แวร์ ซึ่งจะต้องเป็นจริงตลอดการทำงานของโปรแกรม เช่น การกำหนดค่าเวลา หรือการกำหนดการแสดงผลลัพธ์ในรูปแบบต่างๆ เช่น แสง หรือสี ที่จะแสดงผลเมื่อตรงตามเงื่อนไข ซึ่งการสร้างตัวยืนยงชนิดข้อมูลจากตรรกะเชิงกาลเวลาแบบเส้นตรง หรือเรียกสั้นๆว่าแอลทีแอล (LTL) ใช้ในการกำหนดความยืนยงของข้อมูลของโปรแกรมบนโพรเมลาได้ โดยแอลทีแอล เป็นภาษาแบบเป็นทางการ (Formal Language) มีการกำหนดรูปแบบสัญลักษณ์ที่เป็นลำดับ และสามารถอธิบายสิ่งต่างๆที่ขึ้นกับเวลา หรือมีเวลามาเกี่ยวข้อง ซึ่งจะนำมาใช้ในการกำหนดเงื่อนไขหรือตรวจสอบการทำงานของโปรแกรม เพื่อให้มีคุณสมบัติบางอย่าง บนภาษาประเภทการตรวจสอบโมเดล (Model Checking) เพื่อใช้ในการตรวจสอบความถูกต้องในการทำงานของโปรแกรม

ในการเขียนสูตรแอลทีแอลขึ้นมาด้วยตัวเองนั้นเป็นสิ่งที่ยาก เนื่องจากมีความซับซ้อน มักเกิดปัญหา เขียนได้ไม่ถูกต้อง หรือเขียนไม่ครบถ้วน มีคนจำนวนมากพยายามนำเสนอวิธีเขียนสูตรของแอลทีแอล (LTL Formulae) ขึ้นมา แต่ยังเกิดปัญหาความไม่สอดคล้องกันของโปรแกรม กับแอลทีแอลที่เขียนขึ้นมา และไม่สามารถเขียนแอลทีแอลได้อย่างถูกต้องครบถ้วน จากประเด็นดังกล่าวงานวิจัยนี้จึงได้เสนอวิธีการสกัด (Extracting) เพื่อนำข้อมูล เงื่อนไขต่างๆภายใต้การควบคุมที่มีความยืนยง คงที่ ไม่เปลี่ยนแปลง อยู่ในขอบเขตที่กำหนดให้อยู่ในรูปแบบสูตรแอลทีแอลแบบอัตโนมัติ โดยใช้เทคนิคการแบ่งส่วน (Slicing) ของโปรแกรมตามฟังก์ชัน หรือส่วนโมดูลต่างๆ เพื่อให้ผู้ใช้งานนำสูตรแอลทีแอลนี้ไปใช้ในการตรวจสอบอัลกอริทึมในภาษาประเภทการตรวจสอบโมเดล ได้แก่ โพรเมลา บนโปรแกรมสปิน (SPIN) ได้เพื่อตรวจสอบความยืนยงของการทำงานในโปรแกรม ตั้งแต่ข้อมูลและเงื่อนไข เพื่อป้องกันข้อผิดพลาดที่อาจเกิดขึ้น

1.2 วัตถุประสงค์ของโครงงาน

1) เสนอวิธีการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา ภายใต้กฎเงื่อนไขที่มีความสัมพันธ์กัน เพื่อให้ได้สูตรของแอลทีแอลที่เป็นตัวยืนยงชนิดข้อมูลของโปรแกรมที่มีการทำงานหลายอย่างพร้อมกัน

2) เสนอการออกแบบ และพัฒนาเครื่องมือที่ใช้ในการสร้างสูตรแอลทีแอลแบบอัตโนมัติจากการนำข้อมูลจากโค้ดโปรแกรมของโพรเมลาในรูปแบบของโพรเมลาไฟล์

1.3 ขอบเขตของโครงงาน

1) ไฟล์ข้อมูลที่นำเข้าโปรแกรมที่พัฒนานั้นเป็นไฟล์ชนิดโพรเมลาหรือไฟล์ข้อมูลตัวอักษร (Text File)

2) การสกัดนั้นจะมีการสกัดเฉพาะตัวยืนยงชนิดข้อมูล ที่เป็นข้อมูลและควบคุมได้จากตัวแปรและค่าที่เปลี่ยนไปของตัวแปรที่ประกาศไว้

3) ผลลัพธ์ที่ได้มาคือตัวยืนยงชนิดข้อมูลที่เขียนในรูปของสูตรแอลทีแอล

4) สูตรของแอลทีแอลสามารถนำมาใช้กับโพรเมลาที่ใช้สปินในการตรวจสอบการทำงานของโปรแกรม และตรวจสอบความยืนยงชนิดข้อมูลของแอลทีแอลนี้

5) แอลทีแอลต้องผ่านการตรวจสอบไวยากรณ์ (Syntax) ก่อนนำไปทวนสอบ(Verification)

6) ใช้เว็บแอปพลิเคชันในการพัฒนาเครื่องมือ

7) โปรแกรมที่นำมาใช้ในการหาตัวยืนยงต้องเป็นโปรแกรมที่มีการทำงานถูกต้อง

1.4 ขั้นตอนและวิธีการดำเนินโครงงาน

1) ศึกษาข้อมูลเกี่ยวกับการสกัดข้อมูล โดยใช้เทคนิคการแบ่งส่วนของโปรแกรม และการจัดเก็บข้อมูลของโปรแกรม เพื่อให้สามารถนำไปใช้ในการประมวลผลของเครื่องมือได้ง่ายขึ้น

2) ศึกษาข้อมูลเกี่ยวกับโพรเมลา และการนำสูตรของแอลทีแอลมาใช้ในโพรเมลาในโปรแกรมสปิน

3) ศึกษาการสร้างสูตรของแอลทีแอลให้มีความยืนยง และมีคุณสมบัติตรงตามที่ต้องการของโปรแกรม

4) ออกแบบการจัดเก็บข้อมูล หรือเงื่อนไขต่างๆภายใต้เงื่อนไขที่ผู้ใช้ต้องการ เพื่อนำมาใช้ในการประมวลผลได้

5) ออกแบบการสกัดข้อมูล เงื่อนไข ต่างๆของโปรแกรมเพื่อให้อยู่ภายใต้เงื่อนไขหรือภายใต้การควบคุม ผ่านเทคนิคการแบ่งส่วนของโปรแกรม

6) ออกแบบเครื่องมือที่ใช้ในการประมวลผลข้อมูลจากโค้ดโปรแกรมโพรเมลาให้อยู่ใน

รูปแบบสูตรของแอลทีแอล

7) ออกแบบการสร้างสูตรของแอลทีแอลให้มีความยืนยง และสามารถนำไปใช้ในการประมวลผลของโพรเมลาได้อย่างรวดเร็ว

8) การสกัดข้อมูลจากโค้ดโปรแกรมโพรเมลาจากข้อมูล เงื่อนไขการควบคุมชุดคำสั่งต่างๆ ผ่านเทคนิคการแบ่งส่วนของโปรแกรม

9) การพัฒนาโปรแกรมในการประมวลผลข้อมูลจากโปรแกรมโพรเมลาให้อยู่ในรูปของสูตรแอลทีแอล

10) การนำเครื่องมือที่พัฒนาขึ้นมาประมวลผลข้อมูลจากโปรแกรมโพรเมลาที่อยู่ในรูปไฟล์เท็กซ์เพื่อให้ได้สูตรของแอลทีแอลขึ้นมาแบบอัตโนมัติ

11) ทดสอบระบบโดยการนำสูตรของแอลทีแอลที่ได้มาจากการประมวลผลของเครื่องมือมาใช้ในการตรวจสอบการทำงานของโปรแกรมบนโพรเมลาได้

12) ตรวจสอบการทำงานของโปรแกรมเกี่ยวกับคุณสมบัติความปลอดภัย

13) ตรวจสอบข้อมูล หรือเงื่อนไขภายใต้ที่ผู้ใช้กำหนด ถึงความถูกต้องและตรงที่ตามที่ระบุไว้หรือไม่

14) การประเมินผลการทำงานของโปรแกรม คำนึงถึงความถูกต้อง และครบถ้วนตามความต้องการที่กำหนดไว้

15) จัดทำบทความวิชาการและนำเสนอ

16) การสรุปผลของการทำวิทยานิพนธ์ และจัดทำรูปเล่มวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) เพิ่มความสะดวกแก่ผู้ใช้งานในการได้สูตรของแอลทีแอล จากการได้สูตรแอลทีแอลแบบอัตโนมัติ

2) สูตรของแอลทีแอลที่ได้จากการประมวลผลของเครื่องมือมีความยืนยงชนิดข้อมูลมีคุณสมบัติความปลอดภัย และครอบคลุมถึงขอบเขตของค่าหรือเงื่อนไขที่เป็นไปได้

3) สูตรของแอลทีแอลสามารถนำมาใช้ในการตรวจสอบโปรแกรมได้อย่างถูกต้อง

1.6 ผลงานที่ตีพิมพ์จากงานวิจัย

ส่วนหนึ่งของวิทยานิพนธ์ฉบับนี้ได้รับการตอบรับเพื่อตีพิมพ์เป็นบทความวิจัยในหัวข้อเรื่อง “EXTRACTING DATA INVARIANTS FROM PROMELA” โดย Wisut Suksribangteuy and WiwatVattanawood ในงานการประชุมนานาชาติด้านวิศวกรรมซอฟต์แวร์ ครั้งที่ 3 (The 3th

International Conference on Software Engineering : ICOSE EiCompendex, and Scopus 2018) ซึ่งจัดขึ้นที่กรุงเทพมหานคร ประเทศไทย

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีและงานวิจัยที่เกี่ยวข้องกับการการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา โดยมีรายละเอียดดังนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 ตัวยืนยง (Invariants)

ตัวยืนยง คือ ค่าหรือเงื่อนไขที่ถูกอ้างถึงการทำงานที่ต้องเป็นจริงเสมอตลอดการทำงานของโปรแกรม ถูกใช้เป็นการยืนยันตรรกะ (Logical Assertion) ที่จะถูกยึดถือเอาไว้ตลอดการทำงานทุกขั้นตอนของโปรแกรม จะต้องเป็นจริงตลอด โดยตัวยืนยงมีอยู่หลายชนิด เช่น ตัวยืนยงชนิดการวนซ้ำ (Loop Invariants), ตัวยืนยงชนิดชั้น (Class Invariants) หรือตัวยืนยงชนิดข้อมูล (Data Invariants) ที่จะนำมาใช้ในงานวิจัยนี้ โดยตัวยืนยงแต่ละชนิดก็มีลักษณะที่ต่างกันดังนี้

ตัวยืนยงชนิดการวนซ้ำ [3] เป็นคุณสมบัติของการทำงานซ้ำของโปรแกรมที่จะต้องเป็นจริงก่อน ของแต่ละการวนซ้ำในแต่ละครั้ง จะเป็นการยืนยันตรรกะ บางครั้งจะถูกตรวจสอบภายในโปรแกรมด้วยการเรียกใช้การยืนยัน (Assert) ที่จะแสดงเป็นตรรกะที่แสดงแบบรูปนัย ใช้ในการพิสูจน์คุณสมบัติของการทำซ้า

ตัวยืนยงชนิดชั้น [4] เป็นตัวยืนยงที่ใช้จำกัดหรือบังคับวัตถุของชั้น โดยวิธีการของชั้นควรจะทำให้คงอยู่ของตัวยืนยง ตัวยืนยงชนิดชั้นเป็นสภาพที่ถูกจัดเก็บในวัตถุ ถูกนำมาใช้ในซอฟต์แวร์เชิงวัตถุที่จะถูกขยายในสืบทอดของข้อมูล

ตัวยืนยงชนิดข้อมูล เป็นข้อมูลที่อยู่ในรูปของเงื่อนไขการทำงานของโปรแกรมที่ถูกอ้างอิงจากข้อกำหนดที่ได้ระบุไว้ โดยจะต้องเป็นข้อมูลที่อยู่ในรูปของตัวแปร สมการ เงื่อนไขที่จะต้องเป็นจริงตลอดการทำงานของโปรแกรม จะช่วยให้โปรแกรมมีคุณสมบัติความปลอดภัย (Safety) คือ สิ่งที่ไม่ดีของการทำงานของโปรแกรมจะไม่เกิดขึ้น และเมื่อมีการทำงานของโปรแกรมตัวยืนยงชนิดข้อมูลจะต้องคงสภาพยืนยงเสมอ เช่น การกำหนดค่าเวลา หรือการกำหนดการแสดงผลลัพธ์ในรูปแบบต่างๆ เช่น แสง หรือสี ที่จะแสดงผลเมื่อตรงตามเงื่อนไข หรือในทางคณิตศาสตร์ ได้แก่ การหารของจำนวนตัวเลขซึ่งตัวหารหรือเรียกว่าต้นส่วนนั้นจะต้องมีค่าไม่เท่ากับศูนย์ โดยตัวยืนยงชนิดข้อมูลนี้เป็นตัวยืนยงที่จะนำมาใช้ในงานวิจัยนี้

2.1.2 ตรรกะเชิงกาลเวลาแบบเส้นตรง (Linear Temporal Logic)

ตรรกะเชิงกาลเวลาแบบเส้นตรง หรือเรียกสั้นๆว่าแอลทีแอล [5] เป็น ภาษาทางคณิตศาสตร์ที่ใช้แสดงตรรกะเวลาแบบเชิงเส้นที่มีการอ้างถึงเวลา มีการเข้ารหัส (Encode) อยู่ในรูปแบบสูตร ซึ่งสูตรนั้นจะมีความเกี่ยวข้องกับเส้นทาง (Paths) ที่เกิดขึ้นในปัจจุบัน หรืออนาคต เช่น เงื่อนไขนี้จะเป็นจริง เกิดขึ้นจริงในที่สุด เงื่อนไขนี้จะเป็นจริงก็ต่อเมื่ออีกเงื่อนไขหนึ่งกลายเป็นจริง แอลทีแอลเป็นส่วนเล็กๆที่เมื่อนำมารวมกันแล้วจะมีความซับซ้อนสูตรของแอลทีแอลนั้นมาจากการสัญลักษณ์ หรือตัวอักษรรวมเข้าด้วยกันต่อกันเป็นลำดับ

สัญลักษณ์ที่ใช้ในแอลทีแอล [6] มีอยู่ 3 ประเภท ได้แก่

1. สัญลักษณ์ประพจน์อะโตมิค (Atomic Proposition Symbols) เช่น p, q, r เป็นต้น

2. ตัวเชื่อมแบบชั่วคราว (Temporal Connectives) ประกอบด้วยสัญลักษณ์ดังตารางที่ 2-1

ตารางที่ 2-1 สัญลักษณ์ตัวเชื่อมแบบชั่วคราว

สัญลักษณ์	ความหมาย	ตัวอย่าง
G หรือ [] (Globally)	จะเป็นจริงตลอดไปในอนาคต	G(a => b)
F หรือ <> (Finally)	ในอนาคตจะเป็นจริงในที่สุด	Fl
X หรือ O (Next)	จะเป็นจริงในลำดับถัดไป	X(d V e -> f)
U (Until)	จะเป็นจริงจนกระทั่งอีกเหตุการณ์เป็นจริง	g U h

3. ตัวเชื่อมบูลีน (Boolean connectives) มีอยู่หลายสัญลักษณ์ ตามตารางที่ 2.2

ตารางที่ 2-2 สัญลักษณ์ตัวเชื่อมบูลีน

สัญลักษณ์	ความหมาย	ตัวอย่าง
∧ , && (Conjunction)	และ	l ∧ j
V , || (Disjunction)	หรือ	k V l
¬ , ! (Negation)	ไม่	¬ m
-> (Unless)	ถ้า... แล้ว	n -> p

2.1.3 การสกัดข้อมูล (Data Extraction)

การสกัดข้อมูลเป็นกระบวนการ (Process) แยกข้อมูลที่ผู้ใช้ต้องการจากข้อมูลที่มีขนาดใหญ่หรือข้อมูลจำนวนมากจากแหล่งข้อมูลเพื่อให้ได้ข้อมูลที่มีประโยชน์ หรือสามารถนำไปใช้ให้เกิดประโยชน์ได้ โดยเป็นข้อมูลที่มีเหตุผล และหลักฐานที่เชื่อถือได้ ซึ่งเป็นสิ่งสำคัญที่จะนำไปใช้ประโยชน์ต่อไป เช่น การจัดการข้อมูล การตัดสินใจในการทำบางสิ่ง โดยสิ่งที่ต้องคำนึงถึงการสกัดข้อมูลให้มี

การใช้งาน รบกวนการทำงานของแหล่งข้อมูล หรือระบบการดำเนินงานให้น้อยที่สุดเท่าที่จะเป็นไปได้ ซึ่งจะช่วยให้การทำงานของระบบการดำเนินงานนั้นยังคงดำเนินไปได้อย่างปกติ โดยการสกัดข้อมูลนั้นเราสามารถเรียกได้อีกอย่างหนึ่งว่าเป็น “ขั้นตอนการเลือกข้อมูล” ที่จะทำการเลือกข้อมูลตามความต้องการของผู้ใช้เพื่อที่จะนำข้อมูลเหล่านั้นไปเก็บไว้ในคลังข้อมูลต่อไป

2.1.4 การแบ่งส่วนของโปรแกรม (Program Slicing)

ในโปรแกรมคอมพิวเตอร์ การแบ่งส่วนของโปรแกรมเป็นเทคนิคของการแบ่งแยกส่วนประกอบต่างๆของโปรแกรมซึ่งอาจส่งผลกระทบกับจุดหรือค่าที่เราสนใจที่เกี่ยวข้องโดยตรง หรือทางอ้อมที่จะส่งผลต่อค่าหรือตัวแปรอื่นๆ ตามเงื่อนไขที่กำหนด โดยถูกวิเคราะห์จากการไหลของข้อมูล (Data Flow) หรือการไหลของการควบคุม (Control Flow) สำหรับการคำนวณประมวลผลของชุดคำสั่งของโปรแกรม โดยการแบ่งส่วนของโปรแกรมนั้นเป็นเทคนิคที่มักถูกนำมาใช้สนับสนุนในวิศวกรรมซอฟต์แวร์ ในการวิเคราะห์ความต้องการ ซึ่งสามารถวิเคราะห์ข้อกำหนดเชิงรูปนัยของซอฟต์แวร์ (Software Formal Specification) และค้นหาปัญหาที่เป็นไปได้ตั้งแต่เริ่มต้นพัฒนา เพื่อหลีกเลี่ยงการแก้ไขโค้ดที่มีความสัมพันธ์กัน หรือมีจำนวนมากขึ้นในภายหลัง การแบ่งส่วนของโปรแกรมถูกอ้างถึงจากเงื่อนไขของการแบ่งส่วน

การแบ่งส่วนของโปรแกรมเป็นการลดขนาดของโปรแกรมให้มีขนาดเล็กลงจนกระทั่งมีพฤติกรรม (Behavior) ที่ต้องการ การแบ่งส่วนของโปรแกรมนั้นสามารถนำไปใช้ได้รูปแบบ เช่น การหาจุดข้อผิดพลาดที่เกิดขึ้นของโปรแกรม (Error Positioning) ซึ่งช่วยให้หาจุดที่เกิดข้อผิดพลาดได้ว่าอยู่ที่ตำแหน่งใด และช่วยหาจุดที่เกิดข้อผิดพลาดได้ง่ายขึ้น การดูแลรักษาซอฟต์แวร์ (Software Maintenance), การได้มาของโค้ดที่เหมาะสมที่สุด (Code Optimization) , การวิเคราะห์โปรแกรม (Program Analysis), แบบจำลองการตรวจสอบ (Model Checking), การทดสอบระบบ (Testing) และการหาตรวจสอบข้อผิดพลาด (Debugging) ที่เกิดขึ้นในโปรแกรม ซึ่งรูปแบบการนำไปใช้ทั้งหมดนี้มีส่วนช่วยให้ซอฟต์แวร์นั้นมีประสิทธิภาพมากยิ่งขึ้น การแบ่งส่วนของโปรแกรมนั้นมีเทคนิคต่างๆ มีทั้งหมดหลายประเภท เช่น

1. การแบ่งส่วนเคลื่อนที่ไปด้านหน้า (Forward Slicing)

การแบ่งส่วนเคลื่อนที่ไปข้างหน้าเป็นการระบุชุดคำสั่งที่ถูกกระทบจากชุดคำสั่งเฉพาะ มีการคำนวณประมวลผลอีกครั้งจากกราฟการพึ่งพิงของโปรแกรม (PDG : Program Dependence Graph) ซึ่งถูกทำตามจากการพึ่งพิงของข้อมูลและการควบคุม เทคนิคนี้จะช่วยให้รู้ถึงผลกระทบต่อการทำงานหรือการแก้ไขในส่วนของโปรแกรมบนส่วนอื่นๆของโปรแกรม เช่น โปรแกรม p มีตัวแปร V ในชุดคำสั่งของโปรแกรมและแสดงในโปรแกรม ซึ่งอาจถูกส่งผลกระทบจากตัวแปรของ V อื่นของโปรแกรม p โดยขั้นตอนในการทำการแบ่งส่วนไปข้างหน้าจะมีขั้นตอนคือ ระบุถึงเกณฑ์ (Criterion)

ที่จะพิจารณา อ่านโปรแกรมจากส่วนบนของโปรแกรมไล่มาด้านล่างๆเรื่อยๆ และทำการสกัดข้อมูลส่วนที่เกี่ยวกับเกณฑ์ที่พิจารณาทั้งหมดและทำการตัดส่วนที่ไม่เกี่ยวข้องออกไปทั้งหมด เก็บแต่ข้อมูลที่เกี่ยวข้องกับเกณฑ์ที่พิจารณา หรือข้อมูลที่สามารถนำไปใช้ประโยชน์ต่อได้เอาไว้ ตัวอย่างของโปรแกรมการแบ่งส่วนเคลื่อนที่ไปข้างหน้า ดังรูปที่ 2-1 มีการพิจาณาที่ชุดคำสั่ง “int sum = 0;” โดยให้ sum เป็นเกณฑ์ในการพิจารณา ที่ตัวแปร sum โดยทำการอ่านโปรแกรมจากบนลงล่าง ไล่ลงมาถึงบรรทัดที่เกี่ยวข้องกับตัวแปร sum เจอชุดคำสั่ง “sum = sum + I;” และ “System.out.println(sum);” ตามลำดับ ทำให้การแบ่งส่วนของโปรแกรมดังรูปนี้ เหลือ 3 บรรทัดที่ได้มีการขีดเส้นใต้เอาไว้ ซึ่งจะนำมาใช้ประโยชน์ต่อๆไป

```
class Example{
  public static void main () {
     int sum = 0;
     int i = 1;
     while (i < 11) {
        sum = sum + i;
        i = i + 1;
     }
     System.out.println(sum);
     System.out.println(i);
  }
}
```

รูปที่ 2-1 ตัวอย่างโปรแกรมการแบ่งส่วนเคลื่อนที่ไปข้างหน้า

2. การแบ่งส่วนเคลื่อนที่ไปด้านหลัง (Backward Slicing)

การแบ่งส่วนเคลื่อนที่ไปข้างหลังเป็นการระบุชุดคำสั่งที่ส่งผลต่อการคำนวณการประมวลผลคำสั่งเฉพาะ และสามารถทำได้ง่ายในการประมวลผลจากโปรแกรมกราฟการพึ่งพิงและสามารถติดตามความเกี่ยวข้องสัมพันธ์กันของข้อมูลและการควบคุม เช่น เราพิจารณาที่โปรแกรม p และตัวแปร V ที่เกี่ยวข้องกับชุดคำสั่งและแสดงในโปรแกรมและอาจส่งผลต่อค่าของตัวแปร V ที่โปรแกรม p ตัวอย่างของโปรแกรมการแบ่งส่วนเคลื่อนที่ไปข้างหลังดังรูปที่ 2-2

```
class Example{
  public static void main () {
      int sum = 0;
      int i = 1;
      while (i < 11) {
         sum = sum + i;
         i = i + 1;
      }
      System.out.println(sum);
      System.out.println(i);
   }
}
```

รูปที่ 2-2 ตัวอย่างโปรแกรมการแบ่งส่วนเคลื่อนที่ไปข้างหลัง

3. การแบ่งส่วนแบบไม่เคลื่อนที่ (Static Slicing)

การแบ่งส่วนของโปรแกรมแบบไม่เคลื่อนที่ของชุดคำสั่งในโปรแกรมนั้นอาจส่งผลต่อตัวแปรที่จุดใดๆจุดหนึ่ง หรือมากกว่า 1 จุด การแบ่งส่วนนั้นจะถูกกำหนดด้วยกฎเกณฑ์ของการแบ่งส่วน เช่น C=(x,V) โดยที่ x คือชุดคำสั่งของโปรแกรม และ V เป็นชุดของตัวแปรของโปรแกรม การแบ่งส่วนแบบไม่เคลื่อนที่นั้นประกอบไปด้วยชุดคำสั่งที่จะส่งผลต่อตัวแปร V สำหรับข้อมูลนำเข้าทั้งหมดที่เป็นไปได้ในจุดที่เราสนใจ เช่น สนใจในชุดคำสั่ง x เป็นต้นการแบ่งส่วนแบบไม่เคลื่อนที่นั้นถูกคำนวณโดยการค้นหาชุดของคำสั่งที่เกี่ยวข้องในทางอ้อมของลำดับถัดไป การแบ่งส่วนแบบไม่เคลื่อนที่จะไม่มีการรับค่าตัวนำเข้า (Input) เข้ามาทดสอบ ตัวอย่างโปรแกรมที่ถูกนำมาพิจารณา ดังรูปที่ 2-3


```
int i;
int sum = 0;
int product = 1;
for(i = 1; i < N; ++i) {
 sum = sum + i;
 product = product * i;
}
write(sum);
write(product);
```

รูปที่ 2-3 ตัวอย่างโปรแกรมต้นฉบับที่ถูกนำมาพิจารณา

จากรูปที่ 2-3 ถ้าเราทำการแบ่งส่วนของโปรแกรมตามกฎเกณฑ์หากเราพิจารณาจุดที่เราสนใจ คือ (write(sum) , {sum}); นี้ จะได้ดังรูปที่ 2-4

```
int i;
int sum = 0;

for(i = 1; i < N; ++i) {
  sum = sum + i;

}
write(sum);
```

รูปที่ 2-4 ตัวอย่างโปรแกรมที่นำมาแบ่งส่วนจากจุดที่สนใจ

4. การแบ่งส่วนแบบเคลื่อนที่ (Dynamic Slicing)

การแบ่งส่วนของโปรแกรมแบบเคลื่อนที่ เป็นการใช้ข้อมูลเกี่ยวกับการประมวลผลเฉพาะของโปรแกรม ประกอบไปด้วยชุดคำสั่งทั้งหมดที่เกิดขึ้นจริงและส่งผลต่อค่าของตัวแปรที่จุดต่างๆในโปรแกรมสำหรับการประมลผลที่ไม่ได้ขึ้นกับกฎเกณฑ์ของโปรแกรม

ความแตกต่างระหว่างการแบ่งส่วนแบบไม่เคลื่อนที่ และการแบ่งส่วนแบบเคลื่อนที่จะพิจารณาจากส่วนของโปรแกรมในจุดเล็กๆ ส่วนไหนมีการทำซ้ำการทำงานของคำสั่งต่างๆที่ประกอบไปด้วยกล่อง (Block) ของ If-Else คำสั่ง If-Else ซึ่งมีผลต่อตัวแปร ในกรณีของการแบ่งส่วนแบบไม่เคลื่อนที่ โปรแกรมทั้งหมดจะถูกมองไปทั่วถึงจุดที่ไม่ได้คำนึงถึงของการประมวลผลเฉพาะจุดในโปรแกรม ซึ่งจะเป็นแบบไม่จำกัด แต่ในกรณีของการแบ่งส่วนแบบเคลื่อนที่จะพิจารณาการประมวลผลเฉพาะจุดของโปรแกรม ถ้าในกล่องของเงื่อนไข if ถูกประมวลผล และส่งผลต่อคำสั่ง ในกล่องของ Else นั้นจะไม่ถูกประมวลผล และการแบ่งส่วนแบบเคลื่อนนี้จะมีการรับค่าของตัวนำเข้าเข้ามาใช้ในการแบ่งส่วน

2.1.5 การพึ่งพิงกันของข้อมูลและการควบคุม (Data and control dependence)

การพึ่งพิงกันแบ่งออกเป็น 2 ประเภท [11] คือ

1. การพึ่งพิงกันของของข้อมูล (Data Dependence) เป็นการพึ่งพิงกันของข้อมูลระหว่างข้อมูล 2 ข้อมูลหรือมากกว่า ถ้าตัวแปรที่เราสนใจนั้นมีการเปลี่ยนแปลงค่าจากการกำหนดค่าใหม่หรือการคำนวณค่าจากเดิม และส่งผลต่อตัวแปรตัวอื่นๆตามมา ดังตัวอย่าง ระหว่างตัวแปร a, b, c โดยถ้า a ทำการกำหนดค่าให้กับ b และ c มีการอ้างถึง a ด้วย หมายความว่ามีอย่างน้อย 1 เส้นทางที่มีการประมวลผลหรือมี a มาเกี่ยวข้องนั้นจะถูกเปลี่ยนแปลงด้วย ตัวอย่างของการพึ่งพิงกันของข้อมูล ดังรูปที่ 2-5

```
int sum(int a,b,c){
    int v1;
    v1 = a + b;
    v2 = v1 + c;
    return v2;
}
```

รูปที่ 2-5 ตัวอย่างโปรแกรมที่มีการพึ่งพิงกันของข้อมูล

2. การพึ่งพิงกันของของการควบคุม (Control Dependence) เป็นการพึ่งพิงกันของการควบคุมของเงื่อนไขที่มีการควบคุมการทำงานและมีการทำงานย่อยอยู่ภายใน เช่น การทำซ้ำของลูป โดยเงื่อนไข if , do เป็นต้น ดังตัวอย่าง ในการทำงานของลูป d มีเงื่อนไขที่ถูกกำหนดอยู่ และมี e, f ทำงานอยู่ภายใน โดย e, f จะถูกแสดงและประมวลผล จาก d ที่เป็นตัวกำหนดโดยผลลัพธ์ของ e, f ตัวอย่างของการพึ่งพิงกันของข้อมูล ดังรูปที่ 2-6

```
int v1,v2,a,b;
a = 1;
v1 = 0;
v2 = 3;
If (v1<v2){
    v1=v1+2
}
```

รูปที่ 2-6 ตัวอย่างโปรแกรมที่มีการพึ่งพิงกันของการควบคุม

2.1.6 โพรเมลา (Promela)

โพรเมลา เป็นภาษาที่ใช้ในการสร้างแบบจำลอง หรือโมเดล สำหรับการทวนสอบการทำงานของระบบที่มีการทำงานหลายๆอย่างพร้อมกัน หรือเกิดขึ้นในเวลาเดียวกัน เช่น ระบบการกระจาย (Distributed Systems) ในโมเดลโพรเมลาจะติดต่อสื่อสารกันด้วยช่องทางข้อความที่ถูกกำหนดขึ้น อาจเป็นแบบเกิดขึ้นพร้อมกัน (Synchronous) หรือแบบไม่ได้เกิดขึ้นพร้อมกัน (Asynchronous) โพรเมลาเป็นโมเดลที่ถูกวิเคราะห์โดยสปิน (SPIN) ซึ่งสปินเป็นโมเดลที่ใช้ในการตรวจสอบความถูกต้องการทำงานแบบสุ่ม (Random) หรือการจำลองแบบวนซ้ำ (Iterate Simulation) ของการประมวลผลระบบ โดยสปินจะทำการตรวจสอบโมเดลของระบบและสร้างพฤติกรรมที่ต้องการ (Desired Behavior) ของระบบขึ้นมาได้

โมเดลโพรเมลา ประกอบไปด้วย 5 ส่วน คือ

1. การประกาศชนิด (Type Declarations) เช่น bit, byte, int, bool, mtype, typedefs, constantsเป็นต้น

2. การประกาศช่องทาง (Channel Declarations) เป็นการประกาศเริ่มต้น การส่งข้อมูลหรือรับข้อมูล เช่น Chan ch = [dim] of {type, ...} Asynchronous: dim>0เป็นต้น

3. การประกาศตัวแปร (Variable Declarations) สำหรับใช้การทำงานและการเข้าถึงของกระบวนการโดยมีการประกาศตัวแปร 2 แบบ คือการประกาศตัวแปรในฟังก์ชัน (Local variable) และการประกาศตัวแปรนอกฟังก์ชัน (Global variable)

4. การประกาศกระบวนการ (Process Declarations) เป็นกระบวนการที่มีคำสั่งและตัวแปรทำงานร่วมกันหรือที่เรียกกันว่า proctype ซึ่งมีลักษณะเป็นฟังก์ชัน ซึ่งสามารถประกาศตัวแปร หรือกำหนดการทำงานได้จากการเรียก Initหรือใช้ active proctype เพื่อเป็นการกำหนดเริ่มต้นการทำงานทันทีเมื่อเริ่มโปรแกรม โดยสามารถประกาศเองได้ โดยไม่ต้องประกาศใน init โดยสามารถใส่ตัวเลขกำหนดจำนวนครั้งของการทำงานของกระบวนการนี้ได้ โดยกำหนดไว้ในปีกกา “[]” ระหว่างคำว่า active proctype เช่น active [3] proctype

5. กระบวนการเริ่มต้น (Init Process) คือ การเริ่มต้นการทำงานของตัวแปร และกระบวนการกำหนดกระบวนการทำงานทั้งหมด หรือลำดับการทำงานต่างๆของกระบวนการทำงาน

การทำงานของโพรเมลานั้นอาจจะมีการทำงานหลายอย่างควบคู่ไปด้วยกัน ทำให้อาจมีการใช้ทรัพยากรร่วมกัน หรือใช้ตัวแปรเดียวกัน หรือมีการทำงานสลับไปมา ซึ่งอาจต้องมีการควบคุมที่ถูกต้อง ไม่ใช่นั้นอาจจะทำให้โปรแกรมทำงานไม่ถูกต้องตามความต้องการ หรือเกิดข้อผิดพลาดได้ โดยหากการทำงานนั้นสามารถเลือกทำงานได้มากกว่า 1 การทำงานจะมีการทำงานหนึ่งสามารถทำงานได้ และเงื่อนไขนั้นจะต้องเป็นจริง โดยจะแสดงด้วยเครื่องหมายโคลอนจำนวนสองตัว (::) และอีกการทำงานหนึ่งไม่ได้ทำงาน ซึ่งการทำงานที่สามารถทำงานได้เรียกว่าการคุ้มกัน (Guard)และเงื่อนไขอื่นที่ไม่ได้ถูกทำงานเรียกว่าการกีดกัน(Block)ซึ่งอาจโดยคำที่ใช้ในโพรเมลานั้นมีอยู่หลายประเภท เช่น

- ข้อความสั่ง (Statements) ที่ใช้โพรเมลา มีอยู่หลายคำสั่ง เช่น Skip, Assert (<expr>), expression, Assignment, if, do, break, send (ch!), receive (ch?) เป็นต้น

- คำหลัก (Keywords) เป็นคำเฉพาะที่ใช้ในโปรแกรม มีความหมายในตัวมัน ไม่สามารถตั้งชื่อตัวแปรได้ตรงกับคำเหล่านี้ได้ เช่น active, assert, atomic, bit, bool, break, byte, chan, d_step, d_proctype, do, else, empty, enabled, fi, full, goto, hidden, if, init, int, len, mtype, nempty, never, od, of, printf, priority, proctype, provided, run, short, skip, timeout, typedef, unless, unsigned เป็นต้น

- นิพจน์ (Expressions) ประกอบด้วยสัญลักษณ์ หรือฟังก์ชันที่ใช้ในการสร้างนิพจน์ เช่น +, -, *, /, %, >, >=, <=, ==, !=, !,&, ||, &&, |, ~, >>, <<, ++, --, len(), empty(), full(), run, enabled() เป็นต้น

เทคนิคการตรวจสอบความถูกต้องของภาษาโพรเมลาในงานวิจัยนี้มีการใช้คำสั่งการยืนยัน (Assertion) เงื่อนไขเพื่อใช้ประเมินถึงเงื่อนไข ถ้าเงื่อนไขนั้นเป็นจริง การทำงานของโปรแกรมจะยัง

ดำเนินต่อไป แต่หากไม่เป็นจริงโปรแกรมทั้งหมดจะถูกยกเลิก และแสดงข้อความข้อผิดพลาดออกมา (Error Message)

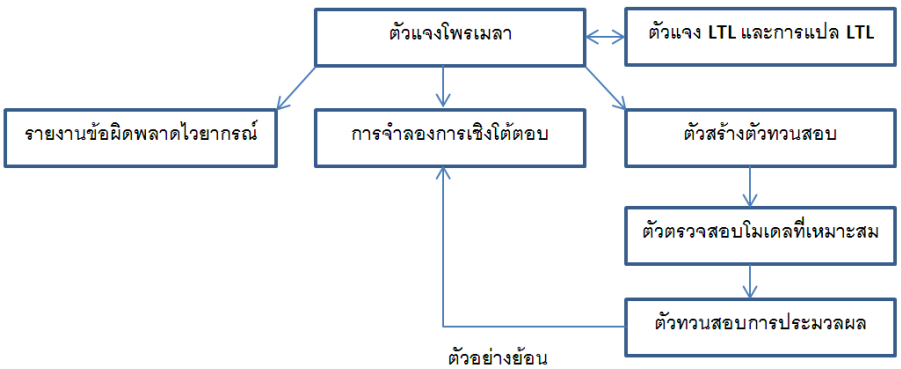
2.1.7 การทวนสอบเชิงรูปนัย (Formal Verification)

การทวนสอบเชิงรูปนัย เป็นการพิสูจน์ความถูกต้องของการออกแบบซอฟต์แวร์หรือฮาร์ดแวร์ ข้อกำหนดเชิงรูปนัยมีลำดับพื้นฐานสำหรับการทวนสอบเชิงรูปนัย คือ โมเดลของระบบเชิงรูปนัย และการพิสูจน์โมเดลความพึงพอใจเชิงรูปนัยนั้น สามารถใช้ในการค้นหาข้อผิดพลาดของการออกแบบ เช่น ความผิดพลาดในการออกแบบฮาร์ดแวร์ เพื่อหาว่าระบบมีพฤติกรรมตรงตามข้อกำหนดความต้องการหรือไม่ตรงตามข้อกำหนด

การทวนสอบเชิงรูปนัยทำได้ด้วยวิธี Deductive Methods และ Automatic Methods ซึ่งวิธี Deductive Methods คือการใช้หลักการพิสูจน์ทางคณิตศาสตร์เพื่อให้แน่ใจว่าสมการหรือสูตรตรรกะต่างๆที่แสดงถึงพฤติกรรมของระบบมีคุณสมบัติตรงตามต้องการหรือไม่ และวิธี Automatics methods ใช้หลักการแจกแจงสถานะที่เป็นไปได้ของระบบและตรวจสอบพฤติกรรมของระบบทุกกรณีที่เป็นไปได้ว่ามีคุณสมบัติความต้องการหรือไม่

2.1.8 สปิน(SPIN)

SPIN เป็นระบบที่สนับสนุนการออกแบบและการทวนสอบระบบของกระบวนการทำงาน SPIN เป็นเครื่องมือที่ใช้ตรวจสอบโมเดลการทวนสอบ (Verification model) โดยใช้ภาษาโพรเมลา SPIN ย่อมาจาก Simple Promela Interpreter ใช้ใน 2 โหมดที่แตกต่างกันได้แก่โหมดจำลอง (Simulator) และโหมดตัวทวนสอบ (Verifier) เพื่อพิสูจน์คุณสมบัติว่าตรงตามที่ต้องการหรือไม่ และใช้ในการหาข้อผิดพลาด (Debugging) รายงานวากยสัมพันธ์ (Syntax) ของโปรแกรมได้ และการระบุหาข้อผิดพลาดโดย SPIN มีโครงสร้างแบบพื้นฐานดังรูปที่ 2-7



รูปที่ 2-7 โครงสร้างพื้นฐานของ SPIN

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัย “Quantitative Program Slicing : Separating Statements by Relevance” โดย Raul Santelices, el al. ปี ค.ศ. 2013 [2]

งานวิจัยนี้ได้นำเสนอถึงวิธีการสกัดวิธีใหม่ คือ Quantitative Slicing (Q-Slicing) ซึ่งการสกัดด้วยวิธีนี้จะแสดงถึงจำนวนชุดคำสั่งที่เกี่ยวข้องกันทั้งหมดที่จะทำการสกัดออกมา การสกัดด้วยวิธี Q-Slicing นี้จะช่วยให้ผู้ใช้งานมุ่งเน้นไปสกัดที่ส่วนสำคัญของโปรแกรมก่อน แล้วจึงไปสกัดยังส่วนอื่นๆ วิธีการสกัดแบบ Q-Slicing นี้สามารถนำไปใช้ประโยชน์ได้มากกว่าการสกัดแบบทั่วไปเพราะการสกัดแบบทั่วไปนั้นจะเป็นการสกัดเฉพาะส่วนที่ต้องการ และส่วนที่ไม่ได้นำมาใช้งาน หรือไม่สำคัญนั้นจะตัดทิ้งทั้งหมด

งานวิจัยนี้จะแสดงให้ได้ทราบว่าชุดคำสั่งทั้งหมดที่มีในโปรแกรมนี้เราสามารถสกัดออกมาแล้วได้กี่ส่วนโดยจะพิจารณาถึงฟังก์ชันการทำงานใหญ่ๆหรือสำคัญก่อน ส่วนที่ไม่ได้ใช้ก็จะไม่มีการตัดทิ้งไป เพราะอาจมีการนำมาใช้ช่วยเหลือในการหาข้อมูลอื่นๆได้

2.2.2 งานวิจัย “Program Slicing” โดย Chen duanzhi ปี ค.ศ. 2010 [3]

งานวิจัยนี้ได้มีการรวบรวมงานวิจัยหรือความก้าวหน้าต่างๆเกี่ยวกับเทคโนโลยีการแบ่งส่วนและแนะนำถึงการจัดเรียงข้อมูลด้วยวิธีการหลากหลายที่แตกต่างกันของการแบ่งส่วนของโปรแกรม กฎเกณฑ์เงื่อนไขของการจัดแบ่งข้อมูลและอัลกอริทึมของการจัดแบ่งข้อมูล รวมถึงแนะนำเทคนิควิธีในการจัดแบ่งข้อมูลที่ก้าวหน้าล่าสุด โดยการเปลี่ยนแปลงแนวคิด การจัดหมวดหมู่ในการจัดแบ่งข้อมูล การพัฒนาอัลกอริทึมที่มีความแม่นยำและความซับซ้อนมากยิ่งขึ้น

การจัดเรียงการแบ่งส่วนของโปรแกรมมีหลายวิธีที่แตกต่างกันในการจัดแบ่งข้อมูลสำหรับการแบ่งส่วนของโปรแกรมจากมุมมองที่แตกต่างกันทั้งหมด 3 มุมมอง ได้แก่

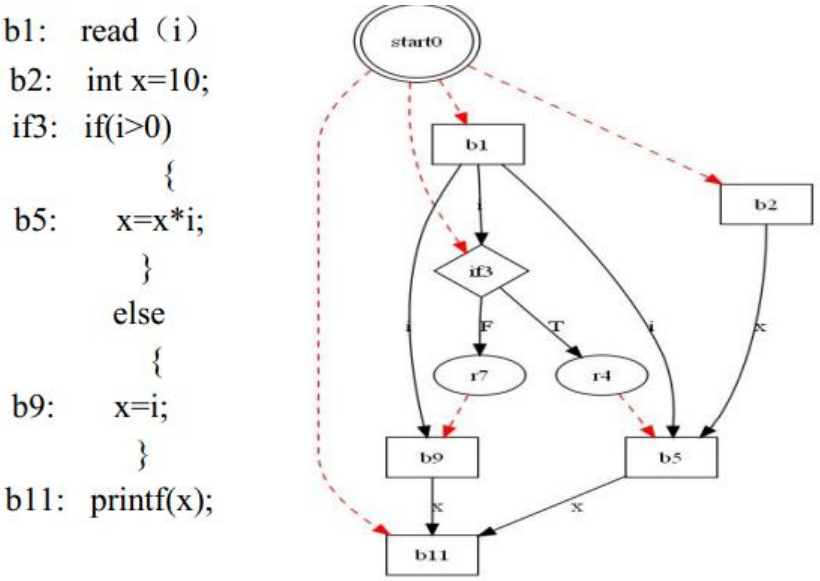
1. Static Slicing and Dynamic Slicing
2. Backward Slicing and Forward Slicing
3. Intra & Inter procedural Slicing

อัลกอริทึมของการแบ่งส่วนของโปรแกรมโดยหลักๆจะเป็นของ Weiser ซึ่งจะยึดสมการการไหลของข้อมูลเป็นหลัก และอัลกอริทึมของ K.J.Ottenstein และ L.M.Ottenstein จะยึดถึงกราฟการพึ่งพิงของโปรแกรมเป็นหลัก และอัลกอริทึมของ Horwitz จะยึดถึงกราฟการพึ่งพิงของระบบเป็นหลักและใช้การแบ่งส่วนการคำนวณของโปรแกรมระหว่างขั้นตอนการทำงาน

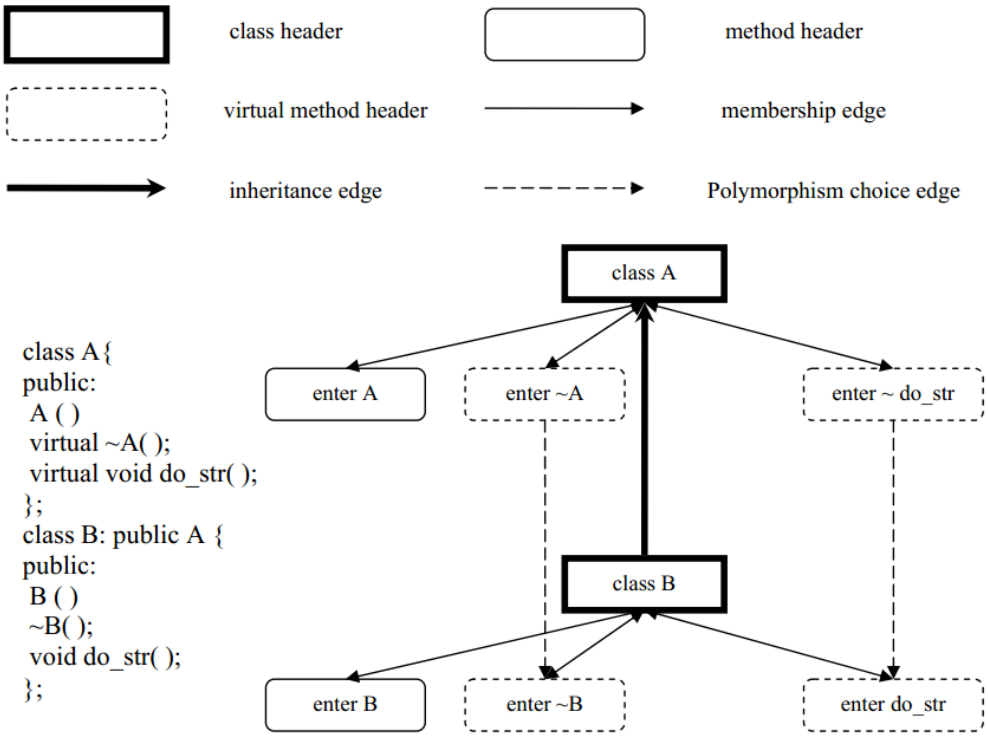
งานวิจัยนี้ได้แสดงให้เห็นถึงความแตกต่างของการแบ่งส่วนในมุมมองต่างๆเพื่อแสดงให้เห็นว่าการแบ่งส่วนแบบไหนเหมาะสมที่จะนำมาใช้การแบ่งส่วนของโปรแกรมนี้ และนำมาสร้างสูตรแอลทีแอลได้ง่ายที่สุด

2.2.3 งานวิจัย “A collection of Program Slicing” โดย Chen duanzhi ปี ค.ศ. 2010 [4]

งานวิจัยนี้ได้นำเสนอถึงแนวทางใหม่ในการดูแลรักษาโปรแกรมและความเข้าใจในโปรแกรม (Program Understanding) โดยได้มีการแนะนำถึงแนวคิดพื้นฐาน และการพัฒนาการแบ่งส่วนของโปรแกรมและแนะนำถึงชนิดของอัลกอริทึมในการแบ่งส่วนที่แตกต่างกัน รวมถึงวิธีการเปลี่ยนรูปแบบโปรแกรมที่มีขนาดใหญ่ให้มีขนาดเล็กลง โดยในงานวิจัยนี้จะแนะนำถึงอัลกอริทึมแบบพีดีจี (PDG : Program Dependence Graph) ดังรูปที่ 2-8 และแบบโอพีดีจี (OPDG :Object-oriented program dependence graph) ดังรูปที่ 2-9



รูปที่ 2-8 ตัวอย่างอัลกอริทึมแบบ PDG [4]



รูปที่ 2-9 ตัวอย่างอัลกอริทึมแบบ OPDG [4]

อัลกอริทึมแบบพีดีจีและโอพีดีจีจะแสดงให้เห็นเป็นกราฟที่แสดงลำดับขั้นตอนการทำงานแบบโปรแกรมแบบทั่วไปแบบอ่านทีละบรรทัด หรือมองเป็นวัตถุกลุ่มใหญ่ก่อนมีการเรียกใช้งาน ซึ่งงานวิจัยนี้ช่วยให้มองถึงการมองโปรแกรมรวมๆว่าเราจะมีการเข้าถึงแบบใดจึงจะเหมาะสมควรอ่านทีละบรรทัดลงมา หรือมองที่หัวข้อใหญ่ก่อนว่ามีความสัมพันธ์กันอย่างไร และจึงมองไปที่การทำงานด้านในอีกที

2.2.4 งานวิจัย “Forward Computation of Dynamic Program Slices” โดย Bogdan Korel and Satish Yalamanchili ปี ค.ศ. 1994 [5]

งานวิจัยนี้ได้นำเสนอถึงวิธีการแบ่งส่วนแบบเคลื่อนที่ (Dynamic Slice) ในการทำงานแบบเคลื่อนที่ไปข้างหน้า (Forward Slice) ซึ่งในวิธีการแบ่งส่วนแบบเคลื่อนที่เป็นการทำงานระหว่างที่โปรแกรมกำลังทำการประมวลผลโดยไม่ได้มีการบันทึกติดตามการทำงานการประมวลผลของโปรแกรม ข้อดีของการแบ่งส่วนการเคลื่อนที่ไปข้างหน้าคือ การจำกัดพื้นที่ที่มีความซับซ้อน ซึ่งตรงข้ามกับวิธีการแบ่งส่วนแบบเคลื่อนที่ไปด้านหลัง (Backward Slice) ดังตัวอย่างรูปที่ 2-10 แสดงโปรแกรมก่อนที่จะนำไปแบ่งส่วนแบบเคลื่อนที่ และแสดงถึงการตามรอยการทำงานของโปรแกรมตั้งแต่จนจบทีละบรรทัด รวมถึงการทำงานซ้ำของโปรแกรมเดิม เรียงตามลำดับโดยจะมีตัวเลขฐานซึ่ง

แสดงถึงลำดับบรรทัดของโปรแกรม และตัวเลขด้านบนแสดงถึงลำดับการทำงาน ดังรูปที่ 2-11 และตัวอย่างการแบ่งส่วนการเคลื่อนที่ไปข้างหน้าโดยแยกตามเกณฑ์ที่พิจารณา ดังรูปที่ 2-12

```
var
n,max,min,sum,i,k: integer;
a: array [1..100] of integer;
1       input(n,a,k) ;
2       max := a[1] ;
3       min := a[1] ;
4       sum := a[1] ;
5       i := k+1 ;
6       while i <= n do
          begin
7,8           if max < a[i] then sum := a[i];
9,10          if  min > a[i] then min := a[i];
11            sum := sum + a[i] ;
12            i := i + k ;
          end ;
13    output(max,min,sum) ;
```

รูปที่ 2-10 ตัวอย่างโปรแกรมโพรเมลาของการหาค่ามากสุด น้อยสุด หรือค่ารวม [5]

```
1^1     input (n,a,k)
2^2     max := a[1]
3^3     min := a[1]
4^4     sum := a[1]
5^5     i := k+1
6^6     i <= n
7^7     max < a[i]              {max < a[3]}
8^8     sum:=a[i]               {sum:=a[3]}
9^9     min > a[i]              {min > a[3]}
11^10   sum:=sum+a[i]           {sum:=sum+a[3]}
12^11   i := i + k
6^12    i <= n
7^13    max < a[i]              {max < a[5]}
8^14    sum:=a[i]               {sum:=a[5]}
9^15    min > a[i]              {min > a[5]}
11^16   sum:=sum+a[i]           {sum:=sum+a[5]}
12^17   i := i + k
6^18    i <= n
13^19   output (max,min,sum)
```

รูปที่ 2-11 การตามรอยโปรแกรมตามลำดับการทำงาน [5]

		SLICE(max)	*SLICE*(min)	*SLICE*(sum)	*SLICE*(i)
1^1	input (n,a,k)	Ø	Ø	Ø	Ø
2^2	max := a[1]	Ø	Ø	Ø	Ø
3^3	min := a[1]	{1,2}	Ø	Ø	Ø
4^4	sum := a[1]	{1,2}	{1,3}	Ø	Ø
5^5	i := k+1	{1,2}	{1,3}	{1,4}	Ø
6^6	i <= n	{1,2}	{1,3}	{1,4}	{1,5}
7^7	max < a[i]	{1,2,5,6}	{1,3,5,6}	{1,4,5,6}	{1,5,6}
8^8	sum:=a[i]	{1,2,5,6,7}	{1,2,3,5,6,7}	{1,2,4,5,6,7}	{1,2,5,6,7}
9^9	min > a[i]	{1,2,5,6}	{1,3,5,6}	{1,2,4,5,6,7,8}	{1,5,6}
11^{10}	sum:=sum+a[i]	{1,2,5,6}	{1,3,5,6}	{1,2,4,5,6,7,8}	1,5,6)
12^{11}	i := i + k	{1,2,5,6}	{1,3,5,6}	{1,2,4,5,6,7,8,11}	{1,5,6}
6^{12}	i <= n	{1,2,5,6}	{1,3,5,6}	{1,2,4,5,6,7,8,11}	{1,5,6,12}
7^{13}	max < a[i]	{1,2,5,6,12}	{1,3,5,6,12}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}
8^{14}	sum:=a[i]	{1,2,5,6,7,12}	{1,2,3,5,6,7,12}	{1,2,4,5,6,7,8,11,12}	{1,2,5,6,7,12}
9^{15}	min > a[i]	{1,2,5,6,12}	{1,3,5,6,12}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}
11^{16}	sum:=sum+a[i]	{1,2,5,6,12}	{1,3,5,6,12}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}
12^{17}	i := i + k	{1,2,5,6,12}	{1,3,5,6,12}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}
6^{18}	i <= n	{1,2,5,6,12}	{1,3,5,6,12}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}
13^{19}	output (max,min,sum)	{1,2}	{1,3}	{1,2,4,5,6,7,8,11,12}	{1,5,6,12}

รูปที่ 2-12 โปรแกรมหลังจากการแบ่งส่วนตามตัวแปรที่คำนึงถึง [5]

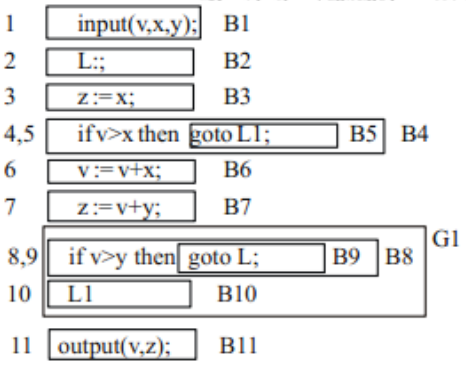
2.2.5 งานวิจัย “Dynamic Program Slicing Methods” โดย Bogdan Korel and Jurgen Rilling ปี ค.ศ. 1994 [6]

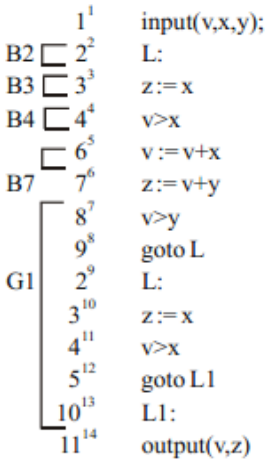
งานวิจัยนี้ได้นำเสนอถึงการแบ่งกลุ่มของวิธีการแบ่งส่วนแบบเคลื่อนที่ทั้งหมดที่มีอยู่ในตอนนี้ และมีการอธิบายถึงอังกอริทึม ที่ใช้ในการแบ่งส่วนแบบเคลื่อนที่ และมีการเปรียบเทียบวิธีการแบ่งส่วนแบบเคลื่อนที่แบบต่างๆดังตัวอย่างของโปรแกรมรูปที่ 2-13 รวมถึงการตัดออกของคำสั่งที่มีการควบคุมการทำงาน ดังรูปที่ 2-14 ซึ่งทำการแยกตามเกณฑ์ที่พิจารณา โดยมีตัวเลขด้านหน้าสองตัวตัวที่เป็นฐานจะเป็นลำดับบรรทัดของโปรแกรม และตัวเลขชี้กำลังแสดงถึงลำดับการทำงานของโปรแกรม

```
1     input (n,a);
2     max := a[1];
3     min  := a[1];
4     i := 2;
5     s:= 0;
6     while i ≤ n do
      begin
7         if max < a[i] then
          begin
8             max := a[i];
9             s := max;
          end;
10        if min > a[i] then
          begin
11            min := a[i];
12            s := min;
          end;
13        output (s);
14        i := i +2;
      end;
15    output (max, min)
```

```
1[1]    input (n,a);
2[2]    max := a[1];
3[3]    min  := a[1];
4[4]    i := 2;
5[5]    s := 0;
6[6]    i ≤ n
7[7]    max  <  a[i];
8[8]    max  := a[i];
9[9]    s := max;
10[10]  min > a[i];
13[11]  output(s);
14[12]  i := i +2;
6[13]   i ≤ n
15[14]  output (max, min)
```

รูปที่ 2-13 (ด้านซ้าย) แสดงตัวอย่างของโปรแกรม (ด้านขวา) แสดงการนำค่า n=3, a=(1,2,3) กำหนดให้กับตัวแปรในโปรแกรมโดยทำการแบ่งส่วนแบบเคลื่อนที่ [6]





รูปที่ 2-14 ตัวอย่างการตัดออกของคำสั่งที่มีการควบคุมการทำงาน [6]

บทที่ 3

การแบ่งส่วนและการสกัดตัวยืนยงชนิดข้อมูล

ในบทนี้จะครอบคลุมเนื้อหาที่เกี่ยวกับแนวคิดและวิธีการดำเนินงานของการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลาโปรแกรม ในรูปแบบของเว็บแอพพลิเคชัน ให้สามารถสร้างสูตรของแอลทีแอลของการทำงานของโปรแกรมภาษาโพรเมลา ที่สามารถสูตรของแอลทีแอลได้อย่างอัตโนมัติ เพื่อให้ได้สูตรแอลทีแอลที่ถูกต้องและครอบคลุมถึงการทำงานของโปรแกรม และโปรแกรมสามารถทำงานได้อย่างถูกต้อง

งานวิจัยนี้ต้องการที่จะช่วยให้ผู้ใช้งานที่ทำหน้าที่เกี่ยวกับข้อกำหนดเชิงรูปนัย ได้สามารถตรวจสอบความถูกต้องของการทำงานของโปรแกรม จากการนำข้อมูลสูตรแอลทีแอล จากฟังก์ชันต่างๆ ในโปรแกรมมาตรวจสอบการทำงานจากข้อมูลตัวแปร หรือการควบคุมเงื่อนไขต่างๆ ที่อยู่ในฟังก์ชัน หรือชุดคำสั่งใดๆของโปรแกรมหนึ่ง ซึ่งอาจมีความสัมพันธ์กันกับฟังก์ชันอื่นๆ หรือในฟังก์ชันตัวมันเอง ที่ผ่านเครื่องมือซึ่งจะทำการสกัดสูตรแอลทีแอลนี้มาจากฟังก์ชันต่างๆ จากเทคนิคการแบ่งส่วนของข้อมูล หรือการควบคุมของชุดคำสั่งต่างๆ เพื่อให้ได้สูตรแอลทีแอล สามารถนำสูตรแอลทีแอลเหล่านั้นที่มีการทำงานกระจายการทำงานแยกออกจากไป หรือมีการทำงานพร้อมกันของฟังก์ชันนำมาใช้ในโพรเมลาในโปรแกรมสปินอีกทีเพื่อทำการตรวจสอบความถูกต้องของการทำงานของโปรแกรม ดังภาพรวมในรูปที่ 3-1

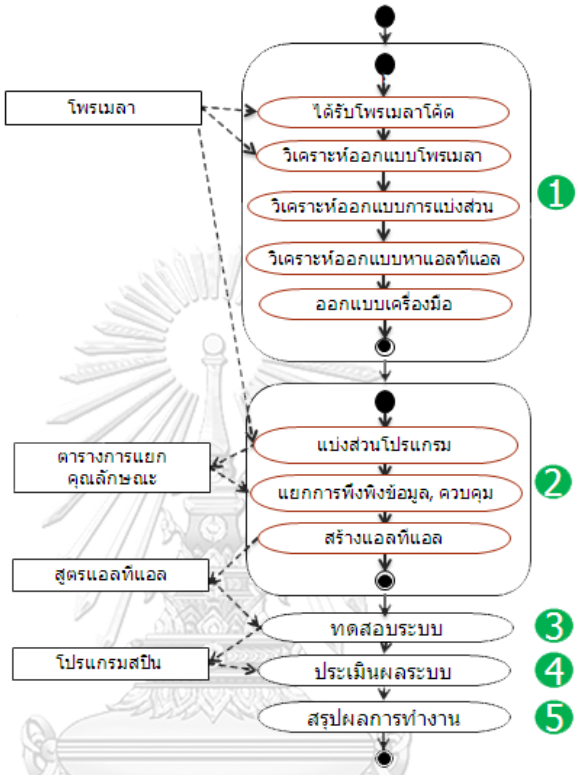
ซึ่งในรูปที่ 3-1 แสดงถึงภาพรวมของงานวิจัย แสดงถึงผู้ใช้งานที่เกี่ยวข้องกับข้อกำหนดเชิงรูปนัยทำการนำโพรเมลาเข้าสู่ระบบการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา โดยภายในระบบจะมีโปรแกรมที่รับตัวนำเข้าโพรเมลาเข้ามาและทำการแบ่งส่วนของโปรแกรมให้อยู่ในรูปแบบข้อมูลที่สามารถนำไปใช้งานต่อได้ และจากข้อมูลนี้ก็นำไปสู่การแปลงให้อยู่ในรูปแบบของสูตรแอลทีแอล หลังจากได้แอลทีแอลแล้วผู้ใช้งานก็จะนำสูตรของแอลทีแอลนี้ไปใช้ทดสอบ และประเมินผลในโปรแกรมบนสปินต่อไป ขั้นตอนในการดำเนินงานวิจัยประกอบไปด้วย 5 ขั้นตอนใหญ่ ได้แก่

3.1 การวิเคราะห์และออกแบบระบบ

ในขั้นตอนการวิเคราะห์และออกแบบระบบ เป็นการวิเคราะห์และออกแบบถึงการนำโปรแกรมที่อยู่ในรูปของไฟล์โพรเมลา มาผ่านการวิเคราะห์ถึงโค้ดโพรเมลา และการแบ่งส่วนเพื่อทำการสกัดข้อมูลในสิ่งที่เราสนใจ เพื่อให้ได้ของสูตรแอลทีแอล และเป็นการออกแบบถึงเทคนิคในการแบ่งส่วนของโปรแกรมว่าจะใช้เทคนิคใดในการแบ่งส่วนของโปรแกรม และทำการออกแบบโครงร่าง

ของเครื่องมือที่ใช้ในการแปลงโพรเมลาให้อยู่ในรูปของสูตรแอลทีแอลโดยขั้นตอนในการวิเคราะห์และออกแบบระบบแสดงได้เป็น 5 ขั้นตอน

3.1.1 การนำเข้าโปรแกรมโพรเมลาจากผู้ใช้งาน เป็นขั้นตอนของการอ่านไฟล์โปรแกรมโพรเมลาจากผู้ใช้งาน ตัวอย่างดังรูปที่ 3-2



รูปที่ 3-1 ขั้นตอนการดำเนินงานวิจัย


```
1 mtype = { Wakeme, Running }            // file ex_5.pml
2
3 bit       lk,     sleep_q
4 bit       r_lock,       r_want
5 mtype   State =     Running
6
7 active proctype client()
8 {
9 sleep:                                 // sleep routine
10   atomic { (lk == 0) -> lk = 1 }      // spinlock(&lk)
11   do                                  // while r->lock
12   :: (r_lock == 1) ->                 // r->lock == 1
13         r_want = 1
14         State = Wakeme
15         lk = 0                        // freelock(&lk)
16         (State == Running)            // wait for wakeup
17   :: else ->                          // r->lock == 0
18         break
19   od;
20 progress:
21   assert(r_lock == 0)                 // should still be true
22   r_lock = 1                          // consumed resource
23   lk = 0                              // freelock(&lk)
24   goto sleep
25 }
```

รูปที่ 3-2 ตัวอย่างโค้ดของโปรแกรมโพรเมลา [13]

3.1.2 วิเคราะห์โปรแกรมโพรเมลา และทำให้อยู่ในรูปของตารางแยกคุณลักษณะเป็นขั้นตอนการทำ Promela Code ให้อยู่ในรูปของตารางแยกคุณลักษณะ โดยคุณลักษณะนั้นมีหลายชนิด ดังตารางที่ 3-1

ตารางที่ 3-1 ตารางการแยกคุณลักษณะต่างๆของโปรแกรม

Statement	Function Name	Type Variable	Name Variable	Value	Condition	Reference Variable	Line Number
			,				

โดยตารางการแยกคุณลักษณะต่างๆ ของโปรแกรม จะแสดงข้อมูลคุณลักษณะสำหรับเก็บข้อมูลรายละเอียดของโค้ดโปรแกรม โดยมีรายละเอียด ดังนี้

1. ข้อความสั่ง (Statement) แสดงให้ทราบว่าในบรรทัดนั้นมีการทำงานอะไร ซึ่งคุณลักษณะชุดคำสั่งนี้ จะต้องมีค่าเสมอจะประกอบไปด้วยคำจำพวก Declare, Assign, Proctype, Function, Condition, Assert, Goto, Define, Print, Init, d_stepetc.

2. ชื่อฟังก์ชัน (Function Name) แสดงชื่อฟังก์ชันของการทำงานจะประกอบไปด้วยฟังก์ชันการทำงานด้านใน

3. ชนิดของตัวแปรที่ประกาศ (Variable Type) เช่น mtype, bit, byte, Chan, Boolean, Integer

4. ชื่อของตัวแปร (Variable Name)

5. ค่าของตัวแปร (Value)

6. เงื่อนไข (Condition) จะประกอบไปด้วยเงื่อนไขการทำงานว่ามีการนำตัวแปรไปทำงานอย่างไร หรือมีเงื่อนไขการ ทำงานอย่างไร

7. ตัวแปรที่ถูกอ้างถึง (Reference Variable) เป็นตัวแปรทั้งหมดที่ได้รับผลกระทบในบรรทัดนั้น

8. หมายเลขบรรทัดของโปรแกรม (Line Number)

โดยตารางจะทำการแยกถึงส่วนของโปรแกรมที่จะไม่นำเข้ามาด้วย คือ ข้อมูลที่เป็นการบันทึกไว้สำหรับอธิบายการทำงานโปรแกรมสั้นๆ (Comment) และบรรทัดที่ไม่มีข้อมูล

ตัวอย่างของตารางที่มีการแยกคุณลักษณะของโปรแกรมแสดงดังตารางที่ 3-2 จากการนำข้อมูลโปรแกรมจากรูปที่ 3-2

ตารางที่ 3-2 ตัวอย่างตารางการแยกคุณลักษณะต่างๆของโปรแกรม

Statement	Function Name	Variable Type	Variable Name	Value	Condition	Reference Variable	Line Number
Declare		mtype	Wakeme ,Running			Wakeme, Running	1
Declare		bit	lk,sleep_q			lk,sleep_q	3
Declare		bit	r_lock,r_want			r_lock, r_want	4
Declare, Assign		mtype	State	Running		State	5
Proctype	client						7
Condition					{		8
Function	sleep						9
Condition					(lk==0)->lk=1	lk	10
Condition					Do		11
Condition					(r_lock==1)->	r_lock	12
Assign			r_want	1		r_want	13
Assign			State	Wakeme		State, Wakeme	14
Assign			lk	0		lk	15
Assign			State	Running		State, Running	16
Condition					Else		17
Condition					Break		18
Condition					od		19
Function	progress						20
Assert					r_lock==0	r_lock	21
Assign			r_lock	1		r_lock	22
Assign			lk	0		lk	23
Goto	sleep						24
Condition					}		25

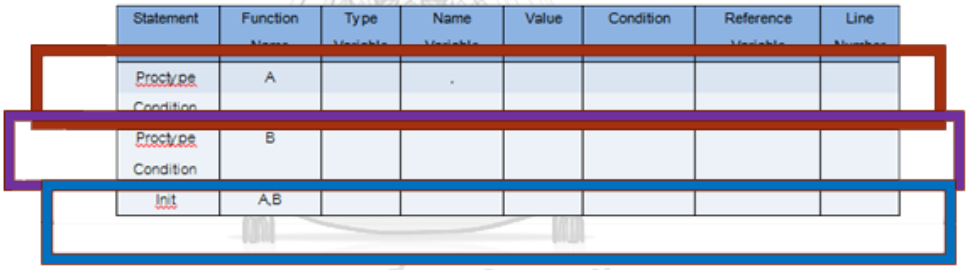
3.1.3 วิเคราะห์ออกแบบการทำการแบ่งส่วนของข้อมูล

เป็นขั้นตอนในการวิเคราะห์และการออกแบบถึงการแบ่งส่วนข้อมูลจากตารางการแยกคุณลักษณะของโปรแกรมโดยจะพิจารณาจากการพึ่งพิงกันของข้อมูล และการควบคุม โดยมีขั้นตอนดังนี้

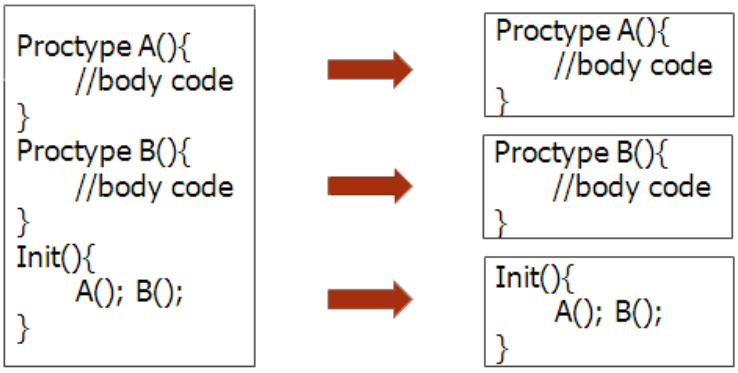
โดยขั้นตอนในการวิเคราะห์และออกแบบการแบ่งส่วนของข้อมูลแบ่งเป็น 8 ขั้นตอน ได้แก่

1. การหาค่าเกณฑ์ (Criterion) จากชื่อของตัวแปรอ่านค่าชื่อของตัวแปรทั้งหมดจากตารางแยกคุณลักษณะและทำการเก็บเอาไว้ในตัวจัดเก็บข้อมูลชั่วคราวเพื่อใช้ในการสกัดข้อมูล

2. การแบ่งส่วนข้อมูลจากโปรแกรมใหญ่ให้มีขนาดเล็กลงหลายๆตารางโดยพิจารณาจากคุณสมบัติชุดคำสั่ง (Statement) ที่เป็น Proctype, Active Proctype, Inline, Label, Function, Init ทำให้ได้ข้อมูลหลายส่วนตามจำนวน Proctype, Active, Proctype, Inlineทั้งหมดที่มีในโปรแกรมโดย Initจะเป็นคำสั่งเริ่มต้นของการทำงานจะบอกว่าโปรแกรมนี้มีการประมวลผลฟังก์ชันใดบ้าง โดยโปรแกรมจะมีคำสั่งนี้หรือไม่มีก็ได้ การแบ่งส่วนนี้ทำให้ได้ตารางย่อยหลายตาราง โดยการตรวจสอบชุดคำสั่งโดยการอ่านค่าจากตารางแยกคุณลักษณะจากบนลงล่างตั้งแต่บรรทัดแรกจนจบ เมื่ออ่านเจอคำที่กำหนดไว้จะทำให้ดูชื่อฟังก์ชันด้วยสำหรับตรวจสอบว่าฟังก์ชัน นี้จะหยุดเมื่อไรจากคำที่กำหนดด้านบน หรือไม่มีคำเพราะสิ้นสุดโปรแกรมแล้ว อ่านจนถึงเจอคำที่กำหนดไว้ใหม่อีกครั้งโดยบรรทัดแรกที่เจอคำกลุ่มนี้จนถึงบรรทัดล่าสุดที่เจอ -1 (บรรทัดก่อนหน้า) จะทำการแยกไปตารางใหม่ โดยการแยกส่วนของโปรแกรมจากตารางการแยกคุณลักษณะ และ การแยกส่วนของโปรแกรมจากโปรแกรมใหญ่ออกเป็นตารางย่อยหลายตาราง ดังรูปที่ 3-3 และ 3-4 ตามลำดับ



รูปที่ 3-3 การแยกส่วนโปรแกรมจากตารางแยกคุณลักษณะ



รูปที่ 3-4 การแยกส่วนโปรแกรมใหญ่ออกเป็นตารางย่อยหลายตาราง

3. ได้ตารางย่อยที่ประกอบไปด้วยคำว่า Proctype, Active Proctype, Inline, Initทั้งหมดที่มีในโปรแกรม ตัวอย่างดังรูปที่ 3-5

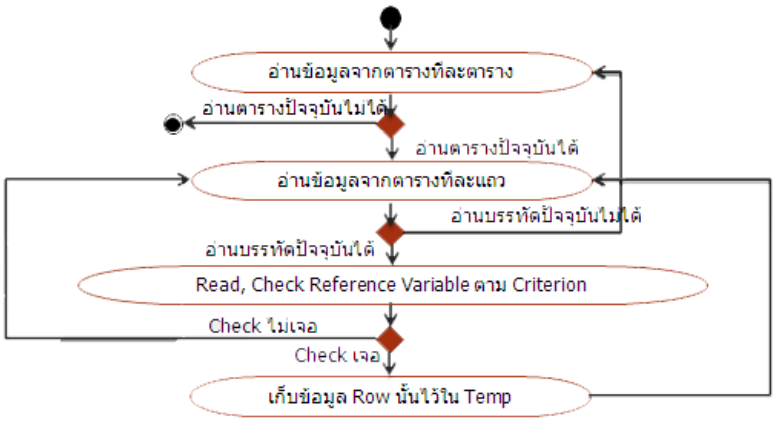
Statement	Function Name	Type Variable	Name Variable	Value	Condition	Reference Variable	Line Number
Proctype Condition	A		.				

Statement	Function Name	Type Variable	Name Variable	Value	Condition	Reference Variable	Line Number
Proctype Condition	B		.				

Statement	Function Name	Type Variable	Name Variable	Value	Condition	Reference Variable	Line Number
init	A,B		.				

รูปที่ 3-5 ตารางย่อยหลังจากแยกส่วนจากโปรแกรมขนาดใหญ่

4. ใช้การแบ่งส่วนแบบเคลื่อนที่ไปข้างหน้าตามเกณฑ์ที่พิจารณา โดยการใช้การแบ่งส่วนแบบเคลื่อนที่ไปข้างหน้าตามเกณฑ์ที่พิจารณา โดยการอ่านค่าเกณฑ์ที่ได้เก็บเอาไว้ทำการเช็คค่าตรวจสอบทีละตาราง และทีละแถวโดยใช้ตัวแปรที่ถูกอ้างถึงในการหาเพื่อเก็บตัวแปรชื่อตัวแปร ค่าของตัวแปรและเงื่อนไขที่เกี่ยวข้องของแต่ละเกณฑ์นอกจากนี้หากเจอชุดคำสั่งที่เป็นเงื่อนไขและเงื่อนไขที่เกี่ยวข้องกับการทำงานซ้ำ หรือเลือกการทำงาน เช่น Do, od, If, fi, Else, break รวมถึงเครื่องหมายปีกกา {, } จะต้องทำการเก็บข้อมูลด้วยเพื่อจะนำมาใช้ในการหาการพึ่งพิงกันของการควบคุม โดยขั้นตอนในการค้นหาตัวแปรที่เกี่ยวข้องแต่ละบรรทัดและตารางมีขั้นตอนดังรูปที่ 3-6



รูปที่ 3-6 ขั้นตอนการค้นหาตัวแปรที่เกี่ยวข้องจากตารางย่อย

5. การสกัดข้อมูลโดยอ้างอิงจากการพึ่งพิงกันของข้อมูลทำการเลือกข้อมูลจากที่เก็บข้อมูลชั่วคราวที่ได้เก็บข้อมูลมาทั้งตัวแปร ชื่อของตัวแปร ค่าของตัวแปรหรือเงื่อนจากการทำการแบ่งส่วนแบบเคลื่อนที่ไปข้างหน้าของแต่ละเกณฑ์มาตรวจสอบว่าข้อมูลที่ได้มามีผลต่อการควบคุมการ ทำงานของการทำซ้ำ หรือการเลือกทำเงื่อนไขหรือไม่ ถ้าไม่มีก็จะจัดอยู่ในการพึ่งพิงกันของข้อมูล โดยาจะประกอบไปด้วยคำที่เกี่ยวกับการนำมาใช้ซ้ำหรือการเลือก เช่น if, fi, do, od, break เป็นต้น

6. การสกัดข้อมูลโดยอ้างอิงจากการพึ่งพิงกันของการควบคุมคล้ายกับการสกัดข้อมูลโดยอ้างอิงจากการพึ่งพิงกันของข้อมูล จะดูกลุ่มคำในที่เก็บข้อมูลชั่วคราวว่ามีคำว่า if, fi, do, od, break หรือไม่ ถ้ามีแสดงว่าอยู่ในกลุ่มการสกัดข้อมูลโดยอ้างอิงจากการพึ่งพิงกันของการควบคุม จะมีการนำเงื่อนไขที่เกี่ยวข้องนั้นมาใช้ในการสร้างสูตรของแอลทีแอล

7. ตรวจสอบความสัมพันธ์กันของตัวแปร เงื่อนไขหรือสมการของการพึ่งพิงกันของข้อมูลและการพึ่งพิงกันของการควบคุมการพึ่งพิงกันนั้นจะพิจารณาถึงการพึ่งพิงชนิดเดียวกันและคนละการพึ่งพิงคนละชนิด หากมีตัวแปร เงื่อนไขหรือสมการซ้ำกันให้ทำการตัดออกให้เหลือเพียงชนิดเดียว และในกรณีที่มีเงื่อนไขต่างกันสามารถนำตัวแปรหรือเงื่อนไขมาเชื่อมต่อกันด้วยสัญลักษณ์ || (or), && (and)

3.1.4 วิเคราะห์ออกแบบการสร้างสูตรแอลทีแอล

สัญลักษณ์ของแอลทีแอลมีอยู่หลายสัญลักษณ์ แต่ที่นำมาใช้ในงานวิจัยนี้ สำหรับการสร้างตัวยืนยงชนิดข้อมูลสำหรับใช้ในการตรวจสอบโปรแกรมจึงมีการใช้เพียงสัญลักษณ์ [] คือคุณสมบัติความปลอดภัย มีความหมายว่าสิ่งที่ไม่ดีจะไม่เกิดขึ้นในการประมวลผลโปรแกรม และสัญลักษณ์ || (or), && (and) ที่ใช้ในการเชื่อมตัวแปร สมการหรือเงื่อนไข

การได้มาของแอลทีแอลนี้จะต้องมีการใส่สัญลักษณ์ [] ทุกครั้ง ตามด้วยตัวแปรหรือสมการคุณสมบัติความปลอดภัยสอดคล้องกับตัวยืนยงชนิดข้อมูล คือ จะต้องเป็นจริงตลอดการทำงานของโปรแกรม เราจะนำตัวแปร สมการ หรือเงื่อนไขสำหรับการทำงานมาใส่ด้านหลังของสัญลักษณ์ [] เช่น LTL([]p) หรือ LTL([]((s<10) || (t>0)))

โดยการสร้างสูตรแอลทีแอลนี้สร้างจากการหาพึ่งพิงกันของข้อมูลและการควบคุม เพื่อให้ได้ตัวแปร สมการหรือเงื่อนไข และนำข้อมูลเหล่านั้นมาใส่สัญลักษณ์ [] วางไว้ด้านหน้าและใส่ในฟังก์ชันแอลทีแอล

3.1.5 ออกแบบเครื่องมือที่ใช้ในการพัฒนาระบบ ที่อยู่ในรูปแบบของเว็บแอปพลิเคชัน

เครื่องมือที่ใช้ในการสร้างสูตรแอลทีแอลจากตัวยืนยงชนิดข้อมูล มีการพัฒนาเป็นเว็บแอปพลิเคชัน โดยมีการนำเข้าข้อมูลไฟล์โปรแกรมโพรเมลาที่เป็นชนิด .pmlและมีการส่งออกข้อมูลเป็นแอลทีแอลในรูปของเท็กซ์ไฟล์

3.2 การพัฒนาระบบ

ในขั้นตอนการพัฒนาระบบ เป็นขั้นตอนในการพัฒนาเครื่องมือสำหรับแปลงข้อมูลโค้ดของโพรเมลา ให้อยู่ในรูปของสูตรแอลทีแอล เพื่อนำไปใช้ในการตรวจสอบโปรแกรมภาษาโพรเมลาที่ได้มาจากโค้ดต้นฉบับเดิมอีกที โดยพัฒนาตั้งแต่การนำโพรเมลาเข้ามาทำการแบ่งส่วนของโปรแกรมเพื่อให้ได้ตารางการแยกคุณลักษณะของโปรแกรม และจากตารางนี้ก็นำมาสกัดข้อมูลโดยใช้การพึ่งพิงของข้อมูล และการควบคุม เพื่อนำสองส่วนนี้มาสร้างสูตรแอลทีแอล ทำให้ได้สูตรของแอลทีแอลที่สร้างขึ้นมาจากตัวยืนยงชนิดข้อมูลขึ้นมา

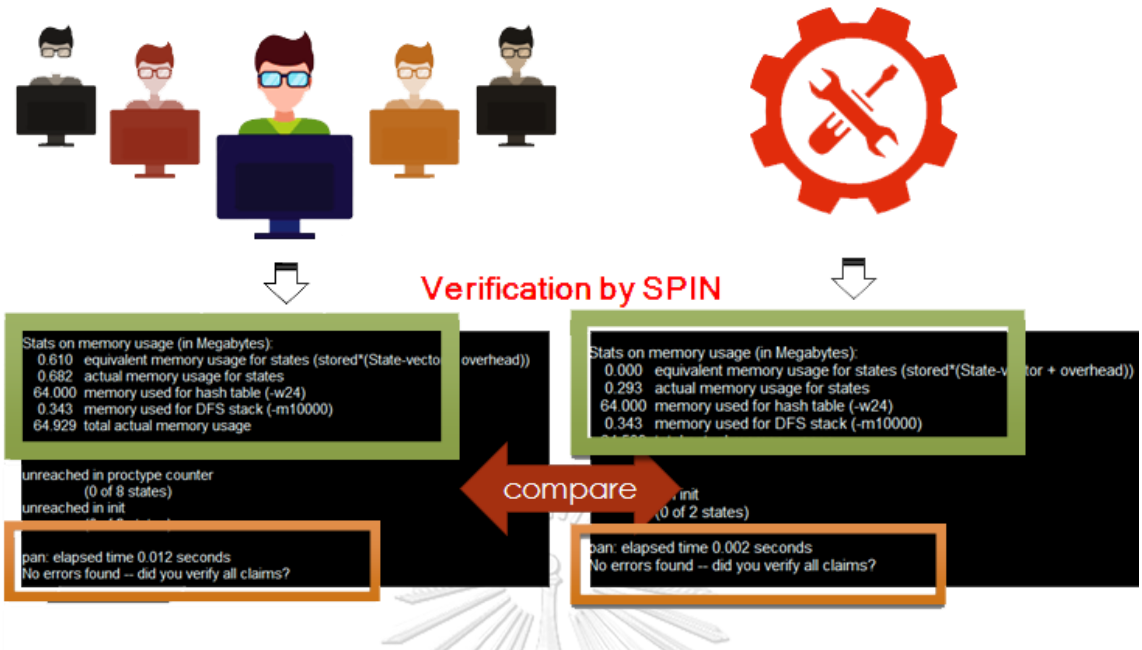
3.3 การทดสอบระบบ

ในขั้นตอนการทดสอบระบบ เป็นการทดสอบเครื่องมือที่พัฒนาขึ้นมานี้ โดยการรันเครื่องมือบนเว็บแอปพลิเคชันเพื่อให้เครื่องมือทำการประมวลผลตั้งแต่โหลดข้อมูลโค้ดโปรแกรมโพรเมลาสกัดข้อมูลจากตัวแปร หรือฟังก์ชัน ชุดคำสั่งต่างๆ ผ่านเทคนิคการแบ่งส่วนของโปรแกรม และนำข้อมูลมาหาการพึ่งพิงของข้อมูลและการควบคุมเพื่อให้ได้สูตรของแอลทีแอลที่อยู่ในรูปของไฟล์เท็กซ์ และนำสูตรของแอลทีแอลที่ได้นี้มาใช้ในคำสั่งแอลทีแอลในโพรเมลาบนโปรแกรมสปินเพื่อทดสอบการทำงานของโปรแกรม

3.4 การประเมินผลระบบ

ในขั้นตอนการประเมินผลระบบ เป็นขั้นตอนการประเมินผลของการทำงานของเครื่องมือว่ามีประสิทธิภาพอย่างไร กับการใช้ในสูตรของแอลทีแอลที่สร้างขึ้นมาแบบอัตโนมัติมาใช้กับโปรแกรมโดยนำมาเปรียบเทียบกับโปรแกรมที่มีการใช้แอลทีแอลที่มีการสร้างขึ้นมาด้วยผู้ใช้เอง หรือนำโปรแกรมของระบบที่มีการใช้งานอยู่จริง นำส่วนของสูตรแอลทีแอล หรือคำสั่งการยืนยันมาเปรียบเทียบกันโปรแกรมเดียวมาผ่านเครื่องมือนี้ และนำมาทดสอบบนโปรแกรมสปินดังรูปที่ 3-7

โดยทำการเปรียบเทียบด้านต่างๆ เช่น จำนวนข้อผิดพลาดที่เกิดขึ้น ขนาดของหน่วยความจำที่ใช้ในการประมวลผล หรือเวลาที่ผ่านพ้นไป (Elapse Time) เป็นต้น และยังสามารถดูได้โดยว่ามีตัวอย่างแย้ง (Counter Example) เกิดขึ้นในโปรแกรมหรือไม่



รูปที่ 3-7 การประเมินผลของแอลทีแอล

3.5 การสรุปผลการทำงาน

ขั้นตอนสรุปผลการทำงาน เป็นขั้นตอนการสรุปการทำงานทั้งหมดของทุกขั้นตอนตั้งแต่การวิเคราะห์และออกแบบระบบ การพัฒนาโปรแกรม การทดสอบระบบ และการประเมินผลระบบว่าผลที่ได้มานั้นเป็นอย่างไร เหมาะสมกับการนำระบบนี้ไปใช้งานจริงหรือไม่ และในการทำงานส่งผลให้เกิดข้อดี ข้อเสียในการทำงานอย่างไร และทำการบันทึกผลเพื่อใช้ในการปรับปรุงแก้ไข หรือพัฒนาต่อไป

บทที่ 4

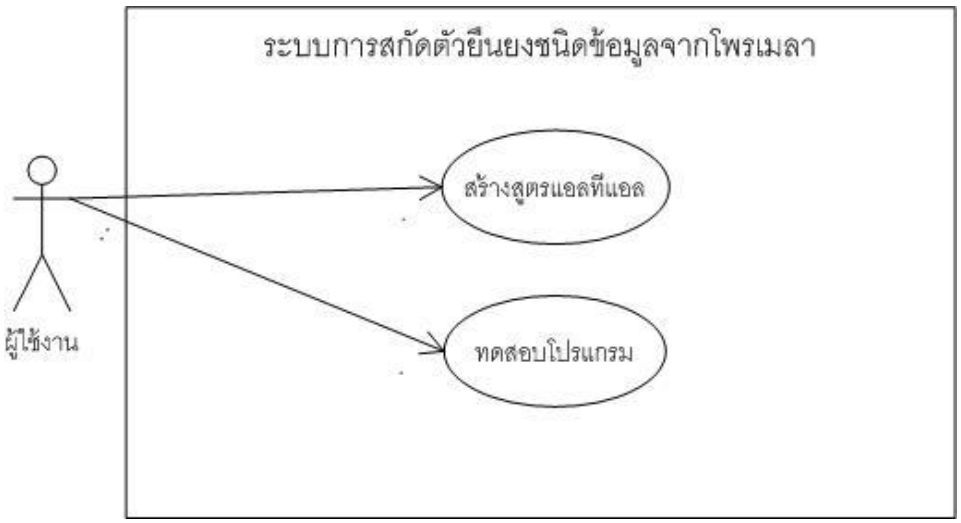
การออกแบบและพัฒนาเครื่องมือ

สำหรับในบทนี้จะนำเสนอการออกแบบเครื่องมือ และขั้นตอนการพัฒนาโปรแกรมสำหรับการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลาโปรแกรมซึ่งผู้วิจัยได้ออกแบบโปรแกรมเชิงวัตถุด้วยภาษายูเอ็มแอลสำหรับการอธิบายฟังก์ชนการทำงานของเครื่องมือ ประกอบไปด้วย แผนภาพยูสเคส(Use Case Diagram) แสดงถึงวิธีการทำงานระหว่างฟังก์ชันการทำงาน และปฏิสัมพันธ์กับผู้ใช้ แผนภาพกิจกรรม (Activity Diagram) อธิบายขั้นตอนการทำงานของแต่ละกิจกรรม แผนภาพคลาส (Class Diagram) แสดงโครงสร้างของเครื่องแผนภาพลำดับ (Sequence Diagram) แสดงลำดับขั้นตอนของการทำงานรวมถึงแสดงส่วนต่อประสานกับผู้ใช้งานด้วยเว็บแอพพลิเคชันดังรายละเอียดต่อไปนี้

4.1 การออกแบบเครื่องมือ

4.1.1 การออกแบบแผนภาพยูสเคส

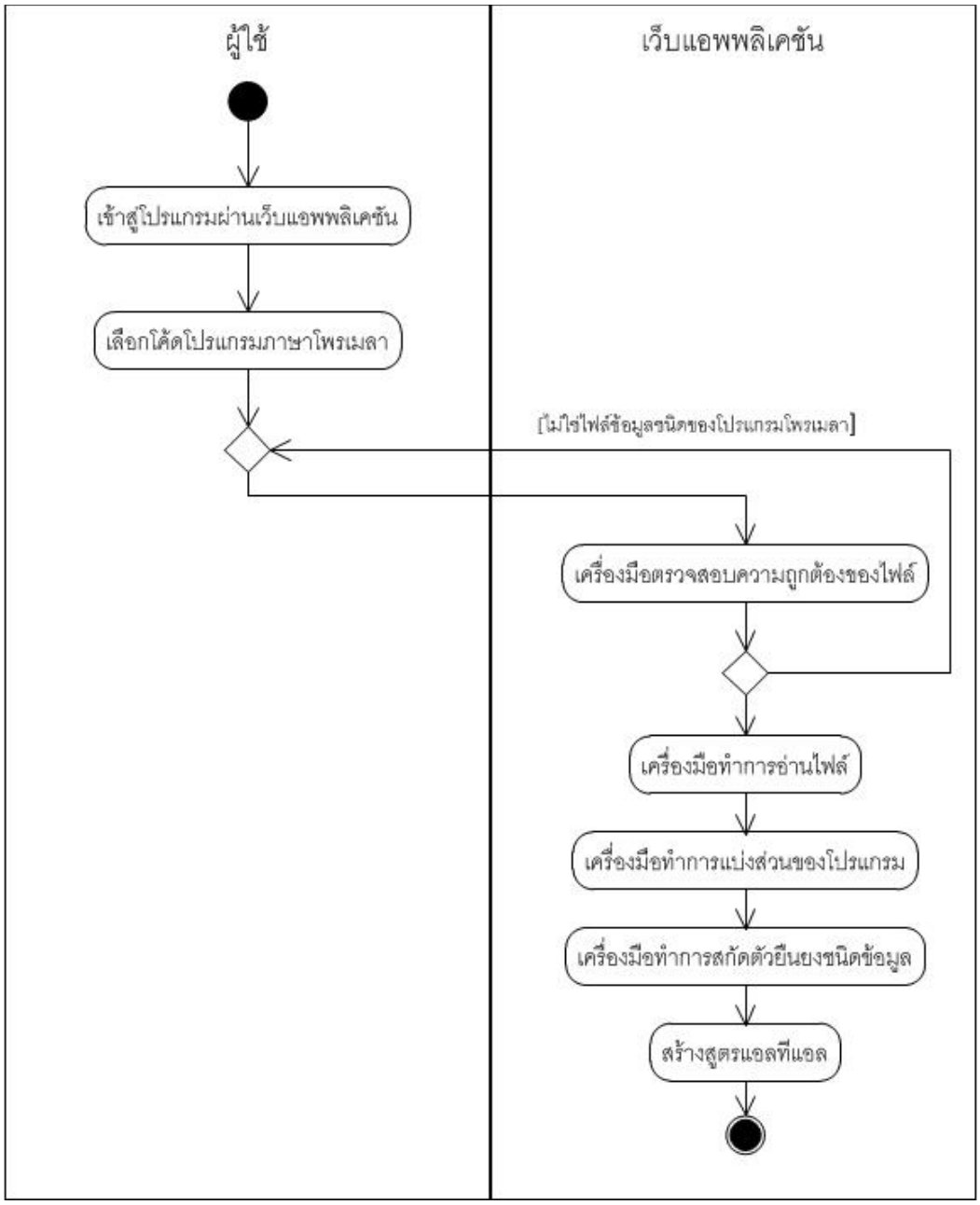
แผนภาพยูสเคสของเครื่องมือการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา ดังรูปที่ 4-1 ผู้ใช้สามารถทำกิจกรรมต่างๆได้บนเครื่องมือได้ 2 กิจกรรมเริ่มจากการนำเข้าของโปรแกรมโพรเมลาเพื่อทำการแบ่งส่วนการทำงานของโปรแกรมเป็นส่วนๆ และนำโปรแกรมที่แบ่งออกเป็นส่วนๆตามคุณลักษณะนั้น มาหาความสัมพันธ์ที่เกี่ยวข้องกันของข้อมูล หรือเงื่อนไขต่างๆ เพื่อนำมาหาตัวยืนยังของข้อมูล และนำตัวยืนยงของข้อมูลนี้มาสร้างสูตรของแอลทีแอลเพื่อนนำมาใช้ในโปรแกรมที่ผู้ใช้ได้ทำการนำเข้าในตอนแรก เพื่อนำโปรแกรมไปใช้งานต่อไป



รูป 4-1 แผนภาพยูสเคสแสดงขอบเขตการทำงานระบบ

4.1.2 การออกแบบแผนภาพกิจกรรม

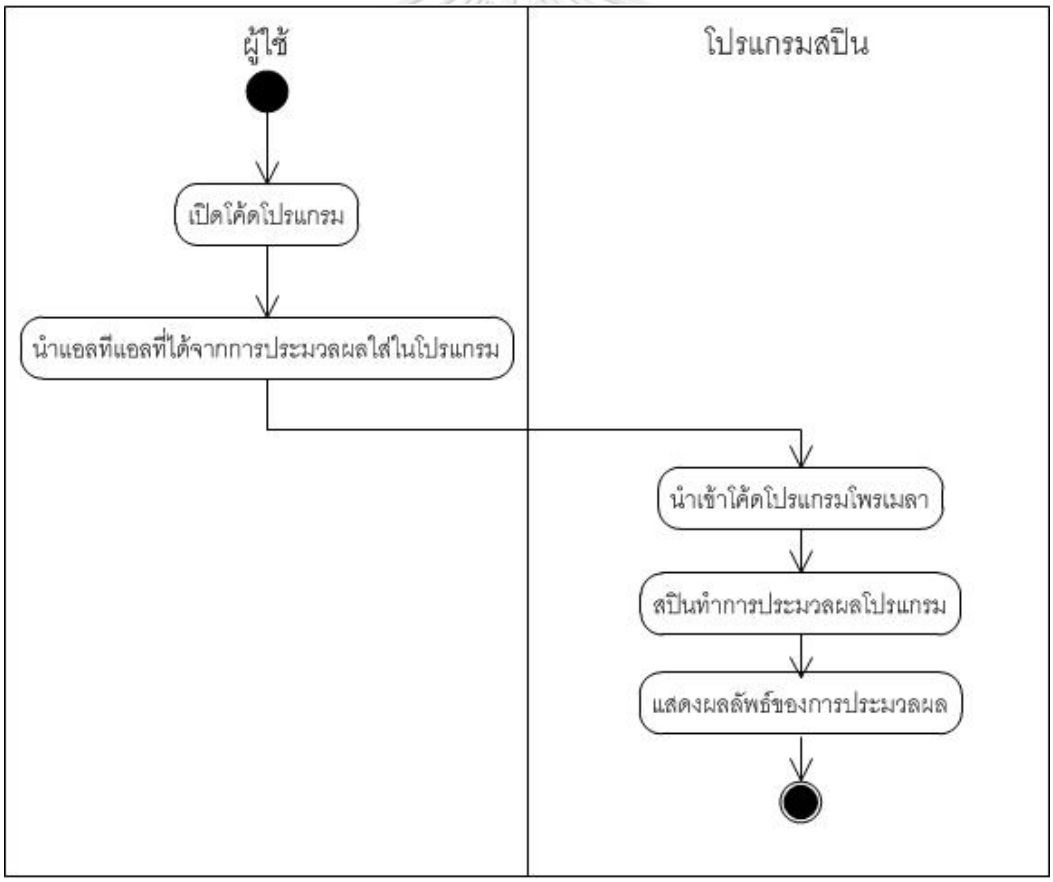
แผนภาพกิจกรรมของเครื่องมือนี้เป็นแผนภาพที่ใช้ในการแสดงขั้นตอนการทำงานของเครื่องมือ โดยแสดงให้เห็นว่าใคร หรือระบบมีหน้าที่รับผิดชอบใดๆในแต่ละกิจกรรม ประกอบไปด้วย 2กิจกรรมหลัก ได้แก่ แผนภาพกิจกรรมการสร้างสูตรแอลทีแอล ดังรูปที่ 4-2 และแผนภาพการทดสอบโปรแกรม ดังรูปที่ 4-3



รูป 4-2 แผนภาพกิจกรรมการสร้างสูตรแอลทีแอล

1. แผนภาพกิจกรรมการสร้างสูตรแอลทีแอล

สำหรับการสร้างแผนภาพนี้แสดงถึงกิจกรรมการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลาเพื่อนำไปสู่การสร้างสูตรของแอลทีแอล โดยเริ่มตั้งแต่ผู้ใช้งานทำการเขาเว็บแอพพลิเคชันผ่านเว็บเบราว์เซอร์เพื่อเริ่มต้นการทำงาน และเมื่อเข้ามาสู่หน้าจอหลัก ก็ทำการเลือกไฟล์โปรแกรมโพรเมลาที่ต้องการหาตัวยืนยงชนิดข้อมูลของโปรแกรม ระบบก็จะทำการตรวจสอบว่าเป็นไฟล์โปรแกรมชนิดของภาษาโพรเมลา (.pml) หรือไม่ หากไม่ใช่ก็ต้องทำการเลือกไฟล์ใหม่จนเป็นไฟล์ข้อมูลชนิดโพรเมลาไฟล์ และผู้ใช้ทำการยืนยันว่าต้องการหาตัวยืนยงชนิดข้อมูลจากไฟล์นี้ ระบบก็จะทำการอ่านไฟล์และทำการแบ่งส่วนของโปรแกรมตามคุณลักษณะต่างๆ และหาความสัมพันธ์ของแต่ละตัวแปร หรือฟังก์ชันในโปรแกรมเพื่อทำการหาตัวยืนยงชนิดข้อมูล และนำตัวยืนยงชนิดข้อมูลนั้นมาสร้างสูตรแอลทีแอล



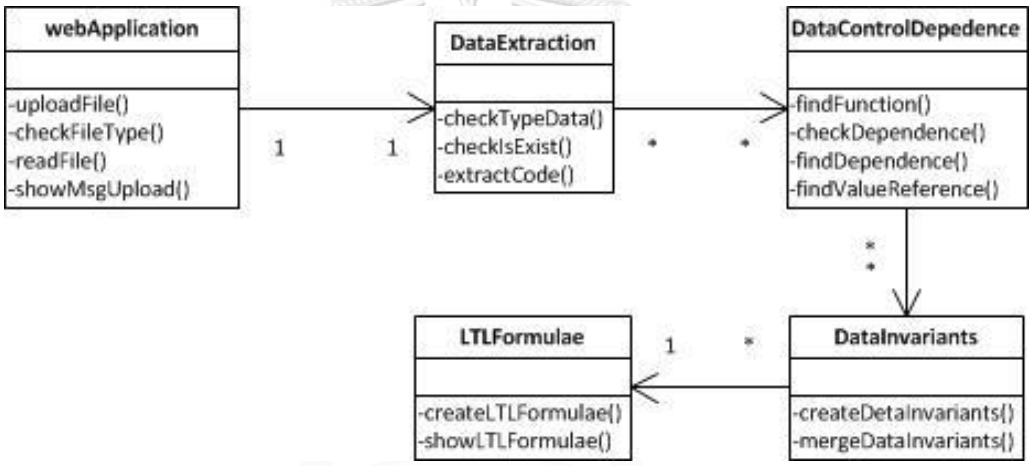
รูป 4-3 แผนภาพกิจกรรมการทดสอบโปรแกรม

2. แผนภาพกิจกรรมการทดสอบโปรแกรม

สำหรับแผนภาพนี้เป็นการนำสูตรของแอลทีแอลที่ได้มาจากการสกัดตัวยืนยงชนิดข้อมูลมาใส่ในโค้ดของโปรแกรมโพรเมลา และผู้ใช้นำโค้ดของโปรแกรมไปประมวลผลต่อบนโปรแกรมสปิน เพื่อทำการประมวลผลและทวนสอบโปรแกรม เพื่อให้สปินแสดงผลลัพธ์ของการทำงานของโพรเมลาที่ประกอบไปด้วยสูตรแอลทีแอลที่หาจากการสกัดตัวยืนยงชนิดข้อมูล

4.1.3 การออกแบบแผนภาพคลาส

แผนภาพคลาสแสดงโครงสร้างการทำงานและความสัมพันธ์ต่างๆทั้งหมดของการทำงานของเว็บแอพพลิเคชันสำหรับการสร้างตัวยืนยงชนิดข้อมูลจากโพรเมลา ซึ่งสามารถแสดงได้ดังรูป 4-4 มีรายละเอียดดังนี้



รูป 4-4 แผนภาพคลาสของระบบการสร้างตัวยืนยงชนิดข้อมูลจากโพรเมลา

จากรูปที่ 4-4 แสดงแผนภาพคลาสของระบบการสร้างตัวยืนยงชนิดข้อมูลจากโพรเมลา ประกอบไปด้วย คลาสของ webApplication, baseGenerator, diGenerator และ result ซึ่งมีรายละเอียดของแต่ละคลาสดังนี้

1) คลาส webApplication ทำหน้าที่เป็นตัวเริ่มต้นการทำงานของโปรแกรมโดยทำหน้าที่ในการนำเข้าไฟล์โปรแกรมและทำการเช็คชนิดไฟล์ว่าถูกต้องหรือไม่ และแสดงกล่องข้อความว่านำเข้าไฟล์ถูกต้องหรือไม่ หากถูกต้องสามารถอ่านไฟล์ได้ก็จะทำการอ่านไฟล์โปรแกรมนั้นๆ

2) คลาส dataExtraction ทำหน้าที่ในการแบ่งส่วนของโปรแกรมที่อ่านไฟล์โปรแกรมมาและทำการการแยกส่วนตามชนิดของข้อมูล และทำการเช็คว่าในแต่ละส่วนมีส่วนใดที่ซ้ำกันบ้าง และทำการจัดเก็บข้อมูลไว้เพื่อดูความสอดคล้องกันของตัวแปรหรือเงื่อนไข

3) คลาส dataControlDependence ทำหน้าที่ในค้นหาตัวแปรหรือสมการ ที่มีความเกี่ยวข้องกัน หรือพึงพิงกัน โดยทำการค้นหาทีละฟังก์ชันการทำงาน

4) คลาส dataInvariants ทำหน้าที่ในการรวมความสัมพันธ์ของตัวแปร หรือเงื่อนไขตามการพึ่งกันของข้อมูล หรือการควบคุม และนำมาสร้างตัวยืนยงชนิดข้อมูล

5) คลาส LTLformulae ทำหน้าที่ในการสร้างสูตรแอลทีแอลจากตัวยืนยงชนิดข้อมูลที่ได้มาจากคลาส dataInvariants และแสดงสูตรแอลทีแอล

4.2 การพัฒนาเครื่องมือ

ในหัวข้อการพัฒนาเครื่องมือนี้จะแสดงถึงสภาพแวดล้อมในการพัฒนาเครื่องมือสำหรับพัฒนาเว็บแอพพลิเคชันในการแบ่งส่วนของโปรแกรมในการสกัดหาตัวยืนยงชนิดข้อมูลจากโพรเมลาโปรแกรม และแสดงถึงการออกแบบโครงสร้างส่วนต่อประสานกับผู้ใช้

4.2.1 สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือ

สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือสามารถแบ่งได้เป็น 2 ประเภท ได้แก่ ฮาร์ดแวร์ และซอฟต์แวร์ ซึ่งมีรายละเอียดดังนี้

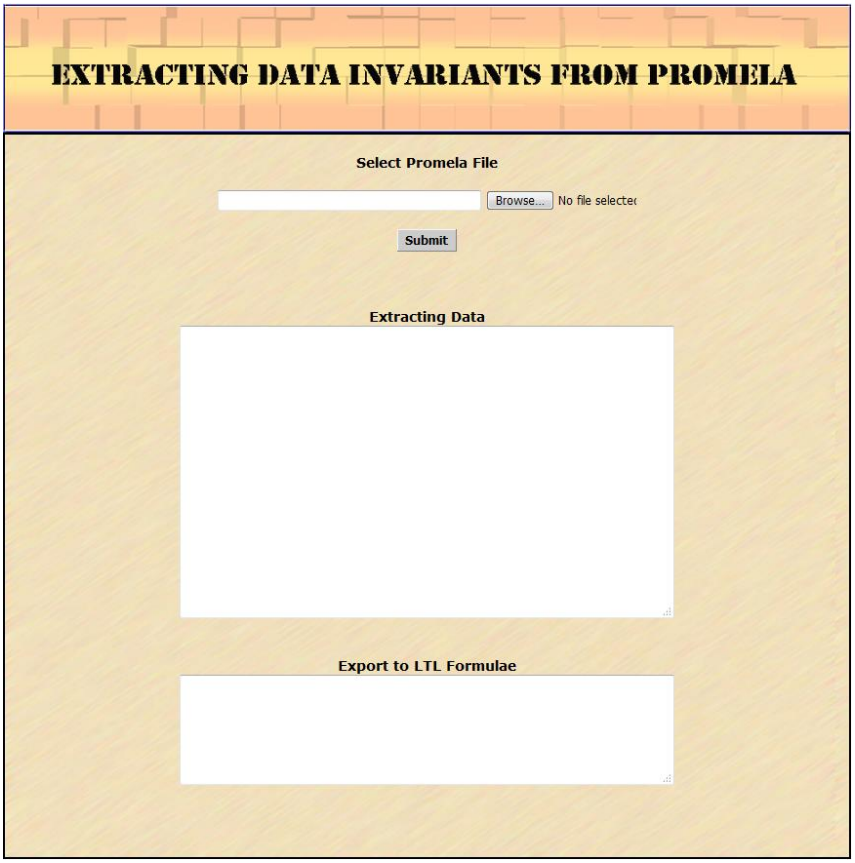
1) สภาพแวดล้อมในการพัฒนาเครื่องมือด้านฮาร์ดแวร์
   - เครื่องคอมพิวเตอร์แบบพกพา แม็คบุ๊คโปร หน่วยประมวลผล Intel Core™ i53210M CPU 2.5 GHz
   - หน่วยความจำของคอมพิวเตอร์ RAM 8 GB
   - ชนิดระบบของคอมพิวเตอร์ระบบปฏิบัติการ 64 bit
2) สภาพแวดล้อมในการพัฒนาเครื่องมือด้านซอฟต์แวร์
   - ระบบปฏิบัติการ MacOS
   - Microsoft Visio 2007
   - Microsoft Visual Studio Ultimate 2010

4.2.2 การออกแบบส่วนต่อประสานกับผู้ใช้

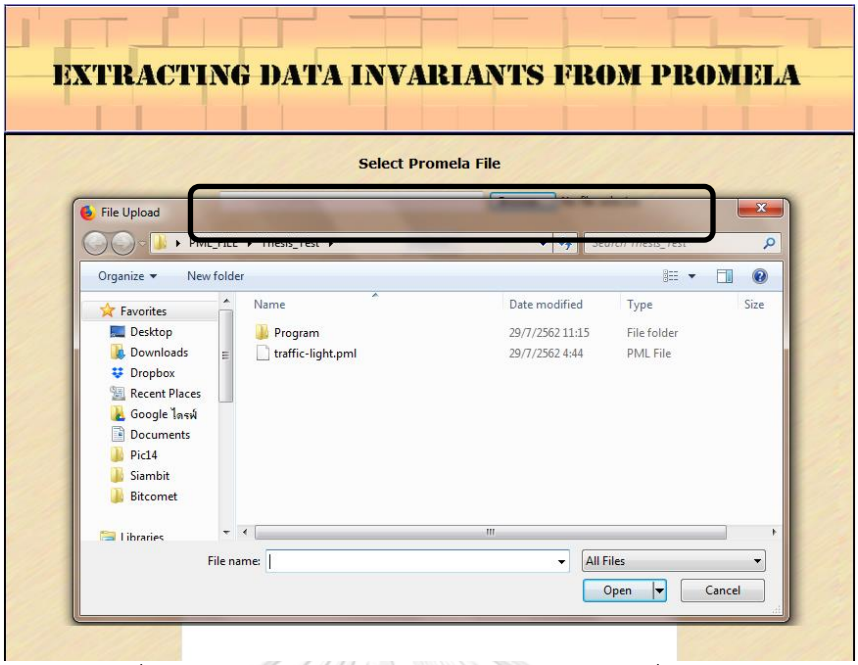
การออกแบบโครงสร้างส่วนต่อประสานของเครื่องมือในการสกัดตัวยืนยงชนิดข้อมูลของโพรเมลานั้นเป็นรูปแบบของเว็บแอพพลิเคชันดังรูปที่ 4-6 ซึ่งประกอบไปด้วย ปุ่มการเลือกไฟล์ข้อมูล ปุ่มการแปลงข้อมูล และหน้าต่างแสดงผลลัพธ์ของการแปลงข้อมูลซึ่งจะได้ตัวยืนยงชนิดข้อมูลในรูปแบบของสูตรแอลทีแอลขึ้นมา โดยแต่ละส่วนประกอบประกอบไปด้วย

1. ปุ่มการเลือกไฟล์ข้อมูล เป็นปุ่มที่ใช้สำหรับเรียกไฟล์ของโพรเมลาที่ต้องการสร้างตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลาขึ้นมาในรูปแบบของสูตรแอลทีแอลโดยไฟล์ข้อมูลจะเป็นไฟล์ .pml ดังรูปที่ 4-7

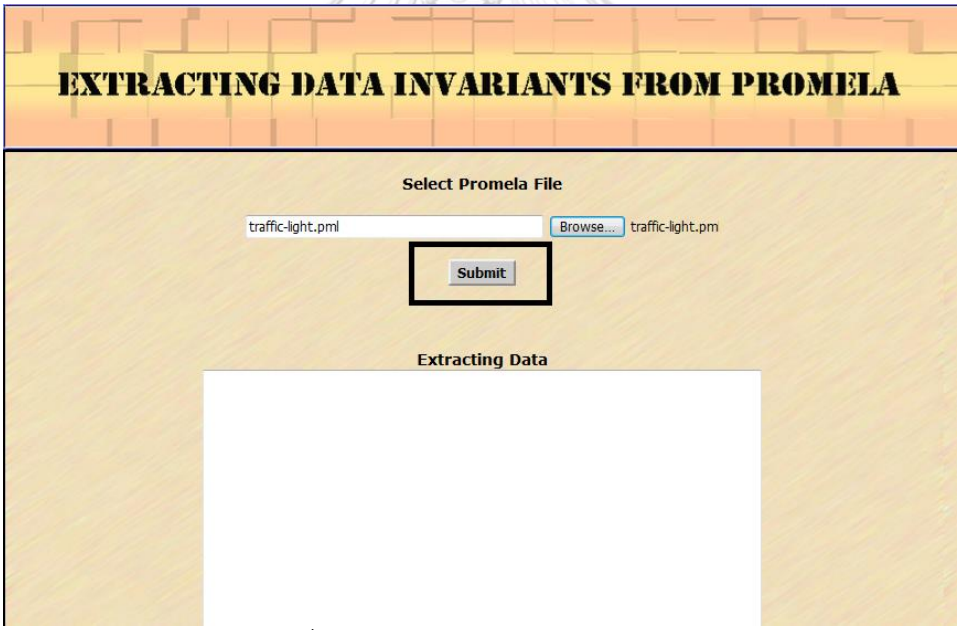
2 .ปุ่มการแปลงไฟล์ข้อมูลเป็นปุ่มที่ใช้สำหรับแปลงไฟล์โพรเมลาที่ต้องการสร้างตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลาขึ้นมาในรูปแบบของสูตรแอลทีแอลโดยจะทำการเลือกจากไฟล์ .pml ที่ผู้ใช้ทำการเลือกไว้ในขั้นตอนก่อนหน้า ดังรูปที่ 4-8



รูปที่ 4-6 รูปหน้าจอการสกัดตัวยืนยงชนิดข้อมูลของโพรเมลา



รูปที่ 4-7 หน้าจอแสดงการเลือกไฟล์ชนิดโพรเมลาที่ต้องการแปลง



รูปที่ 4-8 หน้าจอแสดงปุ่มในการแปลงไฟล์ข้อมูล

3. ปุ่มการแปลงไฟล์ข้อมูล “Submit” เป็นปุ่มที่ใช้สำหรับแปลงไฟล์โพรเมลาที่ต้องการสร้างตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลาขึ้นมาในรูปแบบของสูตรแอลทีแอลโดยจะทำการเลือกจากไฟล์ .pml ที่ผู้ใช้ทำการเลือกไว้ในขั้นตอนก่อนหน้า ดังรูปที่ 4-9

เมื่อผู้ใช้งานกดปุ่ม “Submit” เพื่อแปลงไฟล์โพรเมลาที่ได้นำเข้ามา หน้าจอด้านบนจะมีกล่องแสดงการแบ่งส่วนของข้อมูลตามคุณลักษณะ โดยแยกคุณลักษณะตามประเภทที่กำหนดไว้ก่อน

หน้า และโปรแกรมจะทำการสกัดข้อมูลส่วนที่เกี่ยวข้องกับเกณฑ์ที่กำหนดไว้เพื่อมาสร้างตัวยืนยงชนิดข้อมูลในรูปแบบของแอลทีแอลดังกล่องด้านล่างเพื่อให้ผู้ใช้นำสูตรแอลทีแอลด้านล่างนี้นำไปใส่ในโปรแกรมโพรเมลาต้นฉบับอีกที เพื่อนำไปทดสอบหรือประมวลผลต่อไป

รูปที่ 4-9 หน้าจอแสดงผลลัพธ์ตัวยืนยงชนิดข้อมูลในรูปแบบสูตรแอลทีแอล

บทที่ 5

การทดสอบเครื่องมือและกรณีศึกษา

สำหรับใบบทนี้จะกล่าวถึงการทดสอบเครื่องมือสนับสนุนการสกัดตัวยืนยงชนิดข้อมูลชนิดโพรเมลา กรณีทดสอบ การแบ่งส่วนของโปรแกรม ซึ่งจะกล่าวถึงสภาพแวดล้อมที่ใช้ในการทดสอบการทำงานของเครื่องมือ และผลการทดสอบเครื่องมือ ดังกรณีศึกษาดังต่อไปนี้

5.1 กรณีศึกษาที่ 1 ระบบไฟจราจร

กรณีศึกษาของระบบไฟจราจรเป็นการแสดงผลของระบบไฟจราจร เหลือง แดง และเขียว ซึ่งจะต้องแสดงผลเพียงทีละหนึ่งสีโดยไม่แสดงสีเกิน 1 สีในเวลาเดียวกัน และจะต้องแสดงตามสีไล่ไปตามลำดับ สีเขียว สีเหลือง และสีแดงวนไปตามลำดับ ซึ่งระบบไฟจราจรนี้สามารถแสดงได้ด้วยภาษาโพรเมลาบนโปรแกรมสปินดังตัวอย่างรูปที่ 5-1 ซึ่งจะใส่สูตรแอลทีแอลด้วยคำสั่งที่ผู้ใช้งานอาจคิดขึ้นเองเช่น Rlight + Ylight + Glight == 1

```
1   1int RLight = 0;
2   int YLight = 0;
3   int GLight = 0;
4
5   active proctype Trafic()
6   {
7    int t = 1;
8    S0 : goto SR;
9    SR: do
10           ::(t == 1) ->
11              t = 0;
12              atomic {
13                RLight = 1;
14                YLight = 0;
15                GLight = 0;
16              }
17              t = 1;
18              goto SG;
19         od;
20     SG:  do
21             ::(t == 1) ->
22                t = 0;
23                atomic {
24                  RLight = 0;
25                  YLight = 0;
26                  GLight = 1;
```

รูปที่ 5-1 แสดงโค้ดโปรแกรมระบบจราจรด้วยภาษาโพรเมลา

```
27                  }
28                  t = 1;
29                goto SY;
30              od;
31        SY:  do
32                ::(t == 1) ->
33                  t = 0;
34                  atomic {
35                    RLight = 0;
36                    YLight = 1;
37                    GLight = 0;
38                  }
39                  t = 1;
40                  goto SR;
41              od;
42      }
43
```

รูปที่ 5-1 แสดงโค้ดโปรแกรมระบบจราจรด้วยภาษาโพรเมลา (ต่อ)

จากโค้ดรูปที่ 5-1 เมื่อมีการนำมาแบ่งส่วนของโปรแกรมจะได้ดังตารางที่ 5-1

ตารางที่ 5.1 ตารางการแยกคุณลักษณะของโปรแกรมไฟจราจร

Statement	Function Name	Variable Type	Variable Name	Value	Condition	Reference Variable	Line Number
Declare		int	Rlight	1		Rlight	1
Declare		int	Ylight	0		Ylight	2
Declare		int	Glight	0		Glight	3
Proctype	Traffic						5
Condition					{		6
Declare		int	t	1		t	7
Label	S0				goto SR		8
Label	SR				do		9
Condition					(t==1)->	t	10
Condition			t	0		t	11
Atomic					{		12
Assign			Rlight	1		Rlight	13
Assign			Ylight	0		Ylight	14
Assign			Glight	0		Glight	15
Condition					}		16
Assign			t	1			17
Goto					goto SG		18
Condition					od		19
Label	SG				do		20
Condition					(t==1)->	t	21

ตารางที่ 5.1 ตารางการแยกคุณลักษณะของโปรแกรมไฟจราจร (ต่อ)

Statement	Function Name	Variable Type	Variable Name	Value	Condition	Reference Variable	Line Number
Condition			t	0		t	22
Atomic					{		23
Assign			Rlight	0		Rlight	24
Assign			Ylight	0		Ylight	25
Assign			Glight	1		Glight	26
Condition					}		27
Assign			t	1			28
Goto					goto SY		29
Condition					od		30
Label	SY				do		31
Condition					(t==1)->	t	32
Condition			t	0		t	33
Atomic					{		34
Assign			Rlight	0		Rlight	35
Assign			Ylight	1		Ylight	36
Assign			Glight	0		Glight	37
Condition					}		38
Assign			t	1			39
Goto					goto SR		40
Condition					od		41
Condition					}		42

ซึ่งเมื่อพิจารณาตามเกณฑ์ตามตัวแปรที่สนใจจะได้เกณฑ์จำนวน 3 จำนวน ได้แก่

1. Rlight ซึ่งเมื่อนำเกณฑ์มาสกัดตัวยืนยงจะได้บรรทัดที่ 13 Rlight=1 บรรทัดที่ 24 Rlight =0 และ บรรทัดที่ 35 Rlight=0
2. Ylight ซึ่งเมื่อนำเกณฑ์มาสกัดตัวยืนยงจะได้บรรทัดที่ 14 Ylight=0 บรรทัดที่ 25 Ylight =0 และ บรรทัดที่ 36 Ylight=1
3. Glight ซึ่งเมื่อนำเกณฑ์มาสกัดตัวยืนยงจะได้บรรทัดที่ 15 Glight=0 บรรทัดที่ 26 Glight =1 และ บรรทัดที่ 37 Rlight=0

ซึ่งเมื่อแบ่งตามฟังก์ชันการทำงานของ 3 ฟังก์ชันการทำงานที่มีเกณฑ์มาเกี่ยวข้องจะได้ 3 ฟังก์ชันการทำงานดังนี้

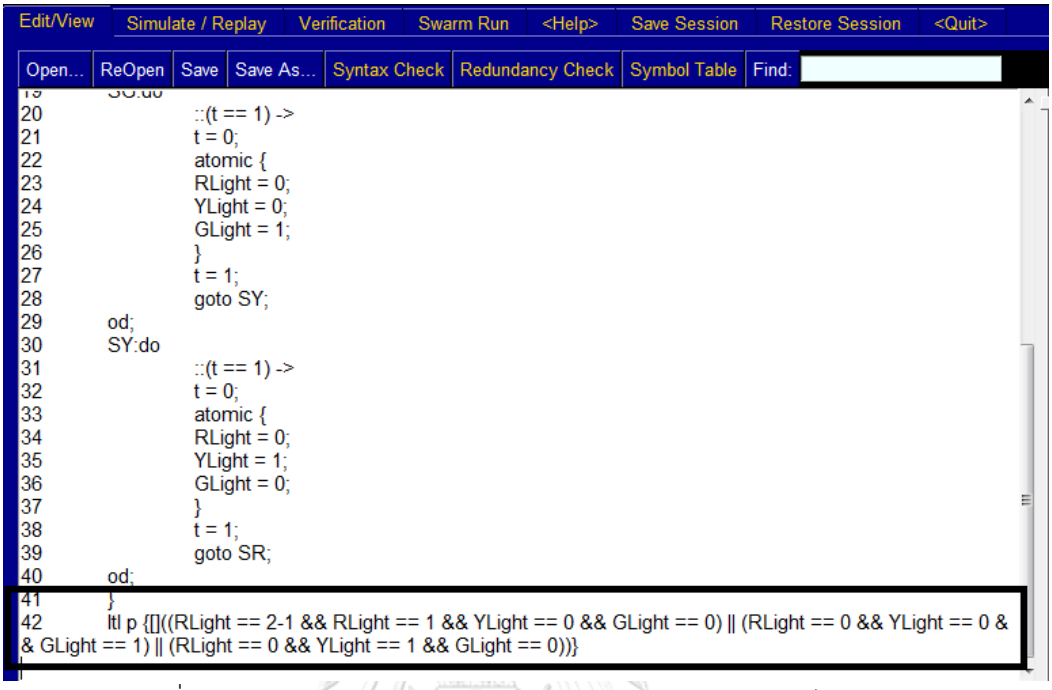
1. ฟังก์ชัน SR (Signal Red) จะได้ Rlight = 1, Glight = 0, Ylight = 0
2. ฟังก์ชัน SG (Signal Green) จะได้ Rlight = 0, Glight = 1, Ylight = 0

3. ฟังก์ชัน SY (Signal Yellow) จะได้ Rlight = 0, Glight = 0, Ylight = 1

เมื่อมีการนำความสัมพันธ์ ตัวพึงพิงกันของข้อมูลหรือการควบคุมทั้งหมดที่เกี่ยวข้องกันนำมาเข้ารวมเข้าด้วยกันดังสูตร “DataInvResult(n) = S_1 ∨ S_2 ∨ … ∨ S_n” ซึ่งมาจากการรวมหลายๆ ข้อมูลมาหาจุดที่เหมือนกันหรือเกี่ยวข้องกันมารวมเข้าด้วยกัน และนำสิ่งที่สนใจหลายๆสิ่งนั้นมารวมกันในภายหลังทำให้ได้ผลลัพธ์ของตัวยืนยงชนิดข้อมูลนี้จะได้ (Rlight == 1 && Ylight == 0 && Glight == 0) || (Rlight == 0 && Ylight == 0 && Glight == 1) || (Rlight==0 && Ylight == 1 && Glight == 0) ซึ่งเมื่อนำมาสร้างเป็นสูตรแอลทีแอลจะได้ LTL []((Rlight == 1 && Ylight == 0 && Glight == 0) || (Rlight == 0 && Ylight == 0 && Glight == 1) || (Rlight==0 && Ylight == 1 && Glight == 0) จากที่กล่าวมาเมื่อนำโปรแกรมมาประมวลผลบนเครื่องมือจะได้ดังรูปที่ 5-2

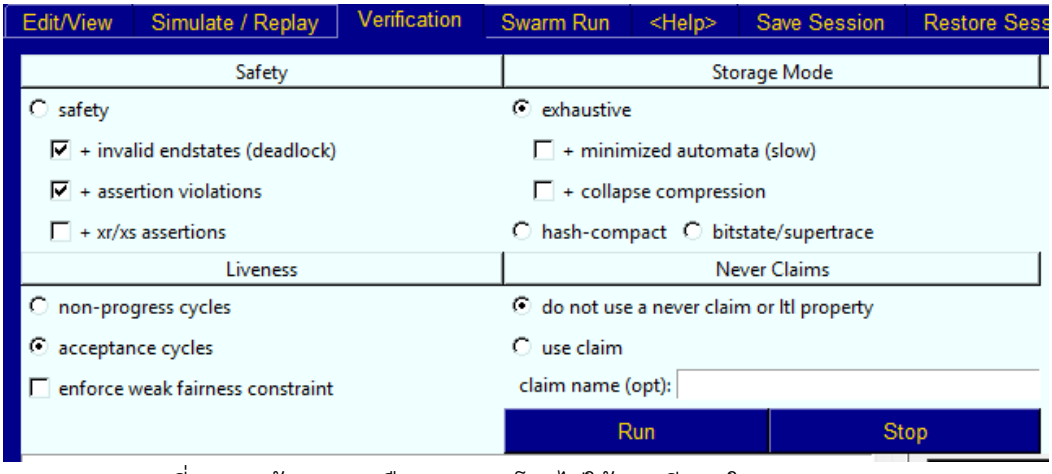
รูปที่ 5-2 หน้าจอแสดงตัวยืนยงชนิดข้อมูลในรูปแบบสูตรแอลทีแอลของระบบไฟจราจร

ซึ่งเมื่อนำผลลัพธ์ของสูตรแอลทีแอลที่ได้มาจากการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา นำมาใส่ในโค้ดต้นฉบับ และนำมาประมวลผลบนโปรแกรมสปิน ดังรูปที่ 5-3



รูปที่ 5-3 หน้าจอการนำแอลทีแอลมาใส่บนโพรเมลาของระบบไฟจราจร

หากไม่ใช้สูตรแอลทีแอลเมื่อใช้สปินตรวจสอบการทำงานโดยการเลือกคำสั่ง Verification ที่โปรแกรมสปินและเลือกคำสั่ง “do not use a never claim or ltl property” เพื่อไม่ใช้สูตรแอลทีแอลในการตรวจสอบ ดังรูปที่ 5-4 จะแสดงผลดังรูป 5-5 จะแสดงผลไม่เกิดข้อผิดพลาด เนื่องจากไม่ได้มีการตรวจสอบด้วยสูตรแอลทีแอล

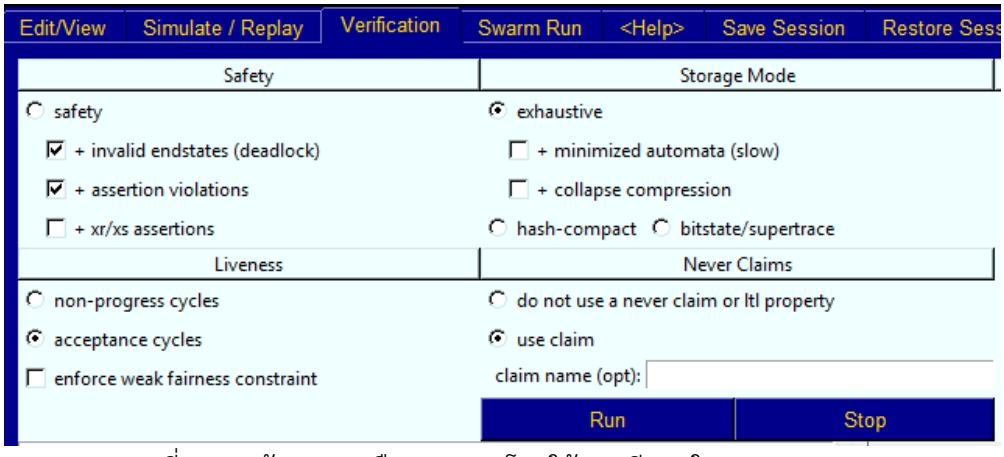


รูปที่ 5-4 หน้าจอการเลือกทดสอบโดยไม่ใช้แอลทีแอลในการตรวจสอบ

```
(Spin Version 6.2.7 -- 2 March 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim         - (not selected)
        assertion violations +
        cycle checks        - (disabled by -DSAFETY)
        invalid end states +

State-vector 16 byte, depth reached 5, errors: 0
       6 states, stored
       1 states, matched
       7 transitions (= stored+matched)
       0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
   0.000  equivalent memory usage for states (stored*(State-vector + overhead))
   0.290  actual memory usage for states
  64.000  memory used for hash table (-w24)
   0.343  memory used for DFS stack (-m10000)
  64.539  total actual memory usage

unreached in proctype Trafic
        traffic-light.pml:41, state 35, "-end-"
        (1 of 35 states)

pan: elapsed time 0.001 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-5 แสดงผลการตรวจสอบด้วยการไม่ใช้แอลทีแอลไฟจราจร

หากใช้สูตรแอลทีแอลและใช้สปินตรวจสอบการทำงานโดยการเลือกคำสั่ง Verification ที่โปรแกรมสปินและเลือกคำสั่ง “use claim” ดังรูปที่ 5-6 จะแสดงผลดังรูปที่ 5-7



รูปที่ 5-6 หน้าจอการเลือกทดสอบโดยใช้แอลทีแอลในการตรวจสอบ

```
(Spin Version 6.2.7 -- 2 March 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim         + (p)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 32 byte, depth reached 15, errors: 0
       8 states, stored
       1 states, matched
       9 transitions (= stored+matched)
       0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
   0.000   equivalent memory usage for states (stored*(State-vector + overhead))
   0.289   actual memory usage for states
  64.000   memory used for hash table (-w24)
   0.343   memory used for DFS stack (-m10000)
  64.539   total actual memory usage

unreached in proctype Trafic
        traffic-light.pml:41, state 35, "-end-"
        (1 of 35 states)
unreached in claim p
        _spin_nvr.tmp:8, state 10, "-end-"
        (1 of 10 states)

pan: elapsed time 0.002 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-7 แสดงผลการตรวจสอบด้วยการใช้แอลทีแอลระบบไฟจราจร

จากผลของการทดสอบการทำงานของโปรแกรมด้วยการนำตัวยืนยงชนิดโพรเมลาในรูปแบบสูตรแอลทีแอลมาใช้ในการตรวจสอบมาเปรียบเทียบกับการทำงานของโปรแกรมโดยไม่ได้ใช้แอลทีแอลในการตรวจสอบ โดยเปรียบเทียบความแตกต่างได้ดังตารางที่ 5-2

ตารางที่ 5.2 ตารางแสดงการเปรียบเทียบผลการทดสอบระบบไฟจราจรสำหรับการใช้สูตรแอลทีแอลกับไม่ใช้สูตรแอลทีแอล

ระบบไฟจราจร	ไม่ใช้สูตรแอลทีแอล	ใช้สูตรแอลทีแอล
State Vector	16 byte	32 byte
State Stored	6 state	8 state
States Match	1 state	1 state

ตารางที่ 5.2 ตารางแสดงการเปรียบเทียบผลการทดสอบระบบไฟจราจรสำหรับการใช้สูตรแอลทีแอล กับไม่ใช้สูตรแอลทีแอล (ต่อ)

ระบบไฟจราจร	ไม่ใช้สูตรแอลทีแอล	ใช้สูตรแอลทีแอล
Equivalent usage memory for states	0.000	0.000
Actual usage memory for states	0.290	0.289
Memory used for hash table	64.000	64.000
Memory used for DFS stack	0.343	0.343
Total actual memory usage	64.539	64.539
Elaspe Time	0.001 seconds	0.002 seconds

5.2 กรณีศึกษาที่ 2 ระบบผู้ใช้บริการ และเครื่องบริการ (Client – Server)

กรณีศึกษาของระบบผู้ใช้บริการ และเครื่องบริการ เป็นตัวอย่างสำหรับการใช้งานบริการของผู้ใช้งานคอมพิวเตอร์ 2 คน โดยมีเครื่องบริการที่รองรับการทำงานเพียงทีละเครื่องโดยผู้ใช้บริการก็จะทำงานได้เพียงทีละหนึ่งคน ตามลำดับโดยเครื่องบริการจะรอรับคำสั่งการร้องขอจากผู้ใช้บริการ และเมื่อทำงานเสร็จแล้วก็จะลบการทำงานปัจจุบัน และรอคำสั่งการทำงานถัดไปโดยเขียนเป็นภาษาโพรเมลาได้ดังรูปที่ 5-8


```
1   #define NUM_CLIENTS 1
2
3   mtype = {NONE, REQ1, REQ2};
4
5   show mtype request = NONE;
6   active proctype server()
7   {
8
9   endwait:
10  do
11  :: request == REQ1 ->
12          printf("Processing request type 1.\n");
13          request = NONE;
14  :: request == REQ2 ->
15          printf("Processing request type 2.\n");
16          request = NONE;
```

รูปที่ 5-8 แสดงโค้ดโปรแกรมระบบผู้ใช้บริการ และเครื่องบริการ

```
17      od;
18      }
19      active[NUM_CLIENTS] proctype client1()
20      {
21      atomic {
22              request == NONE ->
23              request = REQ1;
24              }
25      }
26      active[NUM_CLIENTS] proctype client2()
27      {
28      atomic {
29              request == NONE ->
30              request = REQ2;
31              }
32      }
```

รูปที่ 5-8 แสดงโค้ดโปรแกรมระบบผู้ใช้บริการ และเครื่องบริการ (ต่อ)

จากโค้ดรูปที่ 5-8 เมื่อมีการนำมาแบ่งส่วนของโปรแกรมจะได้ดังตารางที่ 5-3

ตารางที่ 5.3 ตารางการแยกคุณลักษณะของโปรแกรมผู้ใช้บริการ และเครื่องบริการ

Statement	Function Name	Variable Type	Variable Name	Value	Condition	Reference Variable	Line Number
Define			NUM_CLIENTS	1		NUM_CLIENTS	1
Declare		mtype	request	NONE		request	5
Proctype	server						6
Condition					{		7
Label	Endwait						9
Condition					Do		10
Condition					request == REQ1 ->	request	11
Condition					request = NONE	request	13
Condition					request == REQ2 ->	request	14
Condition					request = NONE	request	16
Condition					od		17
Condition					}		18
Proctype	client1						19
Condition					{		20

ตารางที่ 5.3 ตารางการแยกคุณลักษณะของโปรแกรมผู้ใช้บริการ และเครื่องบริการ (ต่อ)

Atomic					{		21
Condition					request == NONE ->	request	22
Condition					request = REQ1	request	23
Condition					}		24
Condition					}		25
Proctype	client2						26
Condition					{		27
Atomic					{		28
Condition					request == NONE ->	request	29
Condition					Request = REQ2	request	30
Condition					}		31
Condition					}		32

ซึ่งเมื่อพิจารณาตามเกณฑ์ตามตัวแปรที่สนใจจะได้เกณฑ์จำนวน 1 จำนวน ได้แก่

- request ทำการแบ่งส่วนตามฟังก์ชันการทำงานได้แก่ Proctype เป็น 3 ส่วน ได้แก่

1. Server พิจารณาเกณฑ์ request จะได้ request == REQ1 และ request == REQ2 ซึ่งเป็นตัวแปรตัวเดียวกัน จึงใช้คำสั่ง “ถ้า..แล้ว” (->) ในการดำเนินการ แทนจากเดิมกรณีที่เป็นตัวแปรต่างกันจะใช้ “และ” ในการดำเนินการ

2. Client1 พิจารณาเกณฑ์ request จะได้ request == NONE และ request == REQ1 ซึ่งเป็นตัวแปรตัวเดียวกัน จึงใช้คำสั่ง “ถ้า..แล้ว” (->) ในการดำเนินการ

3. Client2 พิจารณาเกณฑ์ request จะได้ request == NONE และ request == REQ2 ซึ่งเป็นตัวแปรตัวเดียวกัน จึงใช้คำสั่ง “ถ้า..แล้ว” (->) ในการดำเนินการ

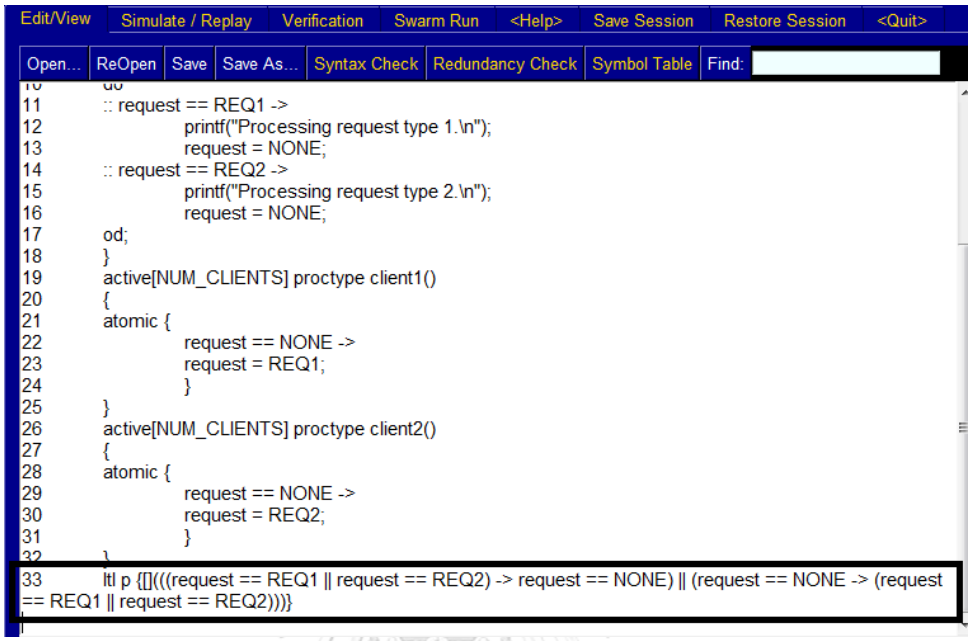
เมื่อการนำความสัมพันธ์ที่เกี่ยวข้องกันนำมาเข้ารวมเข้าด้วยกันดังสูตร “DataInvResult(n) = S_1 v S_2 v ... v S_n” ซึ่งมาจากการรวมหลายๆข้อมูลมาหาจุดที่เหมือนกันหรือเกี่ยวข้องกันมารวมเข้าด้วยกัน และนำสิ่งที่สนใจหลายๆสิ่งนั้นมารวมกันในภายหลังทำให้ได้ผลลัพธ์ของตัวยืนยงชนิดข้อมูลนี้จะได้ ((request == REQ1 || request == REQ2) -> request == NONE) และ (request == NONE -> (request == REQ1 || request == REQ2))

ซึ่งเมื่อนำมาสร้างเป็นสูตรแอลทีแอลจะได้ LTL [](((request == REQ1 || request == REQ2) -> request == NONE) || (request == NONE -> (request == REQ1 || request == REQ2)))

จากที่กล่าวมาข้างต้นเมื่อนำโปรแกรมมาประมวลผลบนเครื่องมือจะได้ดังรูปที่ 5-9

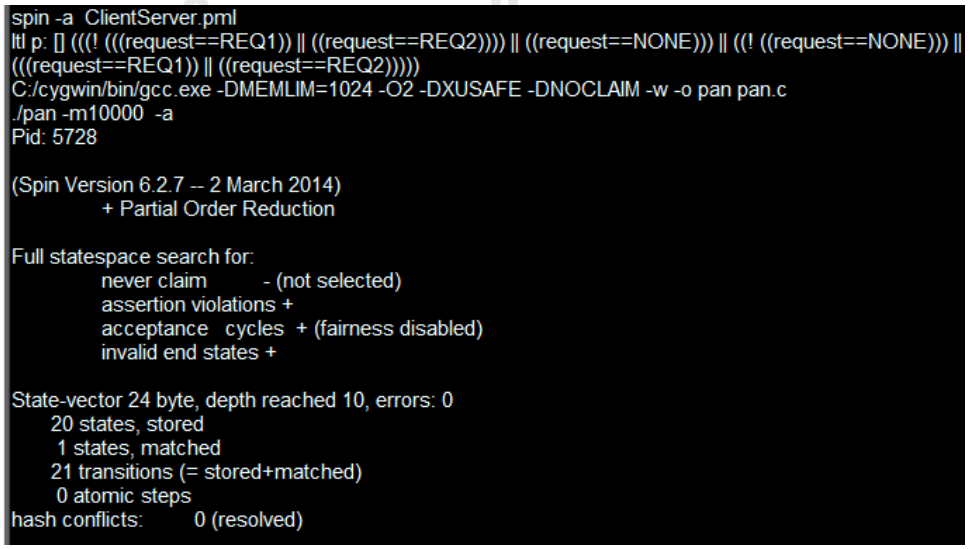
รูปที่ 5-9 หน้าจอแสดงตัวยืนยงชนิดข้อมูลในรูปแบบสูตรแอลทีแอลของระบบผู้ใช้งาน และเครื่องบริการ

ซึ่งเมื่อนำผลลัพธ์ของสูตรแอลทีแอลที่ได้มาจากการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา นำมาใส่ในโค้ดต้นฉบับ และนำมาประมวลผลบนโปรแกรมสปิน ดังรูปที่ 5-10



รูปที่ 5-10 หน้าจอการนำแอลทีแอลมาใส่บนโพรเมลาของระบบผู้ใช้งานและเครื่องบริการ

หากไม่ใช้สูตรแอลทีแอลในการตรวจสอบการทำงานเมื่อใช้สปินตรวจสอบการทำงานจะแสดงผลดังรูป 5-11 จะแสดงผลไม่เกิดข้อผิดพลาด เนื่องจากไม่ได้มีการตรวจสอบด้วยสูตรแอลทีแอล หากใช้สูตรแอลทีแอลและใช้สปินตรวจสอบการทำงานแสดงผลดังรูปที่ 5-12



รูปที่ 5-11 แสดงผลการตรวจสอบด้วยการไม่ใช้แอลทีแอลระบบผู้ใช้งานและเครื่องบริการ

```
Stats on memory usage (in Megabytes):
    0.001   equivalent memory usage for states (stored*(State-vector + overhead))
    0.290   actual memory usage for states
   64.000   memory used for hash table (-w24)
    0.343   memory used for DFS stack (-m10000)
   64.539   total actual memory usage


unreached in proctype server
            ClientServer.pml:18, state 10, "-end-"
            (1 of 10 states)
unreached in proctype client1
            (0 of 4 states)
unreached in proctype client2
            (0 of 4 states)
unreached in claim p
            _spin_nvr.tmp:3, state 6, "(!((((!(((request==REQ1)||(request==REQ2)))||(request==NONE))||(!(
(request==NONE))||((request==REQ1)||(request==REQ2))))))"
            _spin_nvr.tmp:3, state 6, "(1)"
            _spin_nvr.tmp:8, state 10, "-end-"
            (2 of 10 states)

pan: elapsed time 0.002 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-11 แสดงผลการตรวจสอบการไม่ใช้แอลทีแอลระบบผู้ใช้งานและเครื่องบริการ (ต่อ)

```
spin -a  ClientServer.pml
ltl p: [] (((! (((request==REQ1)) || ((request==REQ2)))) || ((request==NONE))) || ((! ((request==NONE)
(((request==REQ1)) || ((request==REQ2)))))
C:/cygwin/bin/gcc.exe -DMEMLIM=1024 -O2 -DXUSAFE -w -o pan pan.c
./pan -m10000  -a
Pid: 6632

(Spin Version 6.2.7 -- 2 March 2014)
            + Partial Order Reduction

Full statespace search for:
            never claim         + (p)
            assertion violations + (if within scope of claim)
            acceptance   cycles  + (fairness disabled)
            invalid end states - (disabled by never claim)

State-vector 28 byte, depth reached 21, errors: 0
       20 states, stored
        2 states, matched
       22 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001   equivalent memory usage for states (stored*(State-vector + overhead))
    0.290   actual memory usage for states
   64.000   memory used for hash table (-w24)
    0.343   memory used for DFS stack (-m10000)
   64.539   total actual memory usage


unreached in proctype server
            ClientServer.pml:18, state 10, "-end-"
            (1 of 10 states)
unreached in proctype client1
            (0 of 4 states)
unreached in proctype client2
            (0 of 4 states)
unreached in claim p
            _spin_nvr.tmp:8, state 10, "-end-"
            (1 of 10 states)

pan: elapsed time 0.001 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-12 แสดงผลการตรวจสอบด้วยการใช้แอลทีแอลระบบผู้ใช้งานและเครื่องบริการ

จากผลของการทดสอบการทำงานของโปรแกรมด้วยการนำตัวยืนยงชนิดโพรเมลาในรูปแบบสูตรแอลทีแอลมาใช้ในการตรวจสอบมาเปรียบเทียบกับการทำงานของโปรแกรมโดยไม่ได้ใช้แอลทีแอลในการตรวจสอบ โดยเปรียบเทียบความแตกต่างได้ดังตารางที่ 5-4

ตารางที่ 5.4 ตารางแสดงการเปรียบเทียบผลการทดสอบระบบผู้ใช้บริการ และเครื่องบริการสำหรับการใช้สูตรแอลทีแอล กับไม่ใช้สูตรแอลทีแอล

ระบบผู้ใช้งานและเครื่องบริการ	ไม่ใช้สูตรแอลทีแอล	ใช้สูตรแอลทีแอล
State Vector	24 byte	28 byte
State Stored	20 state	20 state
States Match	1 state	2 state
Equivalent usage memory for states	0.001	0.001
Actual usage memory for states	0.290	0.290
Memory used for hash table	64.000	64.000
Memory used for DFS stack	0.343	0.343
Total actual memory usage	64.539	64.539
Elaspe Time	0.002 seconds	0.001 seconds

5.3 กรณีศึกษาที่ 3 ระบบการเคลื่อนตำแหน่งสถานะ (Move State)

กรณีศึกษาของระบบการเคลื่อนตำแหน่งสถานะซึ่งแสดงถึงการเคลื่อนตำแหน่งสถานะไปด้านหน้า หรือไปด้านหลังโดยตรวจสอบว่าอยู่ในสถานะที่ต้องการหรือไม่ โดยมีการสั่งการทำงานพร้อมกัน 2 กระบวนการ ดังตัวอย่างโปรแกรมดังรูปที่ 5-13 และเมื่อมีการนำมาแบ่งส่วนของโปรแกรมจะได้ดังตารางที่ 5-5

```
1       byte state = 1;
2       proctype A()
3       {       (state == 1) ->
4               state = state + 1;
5       }
6       proctype B()
7       {       (state == 1) ->
8               state = state - 1;
9       }
10      init { run A(); run B() }
```

รูปที่ 5-13 แสดงโค้ดโปรแกรมระบบเคลื่อนตำแหน่งสถานะ

ตารางที่ 5.5 ตารางการแยกคุณลักษณะของโปรแกรมการเคลื่อนตำแหน่งสถานะ

Statement	Function Name	Variable Type	Variable Name	Value	Condition	Reference Variable	Line Number
Declare		byte	state	1		state	1
Proctype	A						2
Condition					{		3
Condition					(state == 1) ->	state	4
Condition					state = state + 1	state	5
Condition					}		6
Proctype	B						7
Condition					{		8
Condition					state == 1	state	9
Condition					state = state – 1	state	10
Condition					}		11
Init					RUN A		12
Init					RUN B		12

ซึ่งเมื่อพิจารณาตามเกณฑ์ตามตัวแปรที่สนใจจะได้เกณฑ์จำนวน 1 จำนวน ได้แก่

- state ทำการแบ่งส่วนตามฟังก์ชันการทำงานได้แก่ Proctype เป็น 2 ส่วน ได้แก่

1. Proctype A พิจารณาเกณฑ์ state จะได้ state == 1 และ state = state + 1 ซึ่งเป็นตัวแปรตัวเดียวกัน จึงใช้คำสั่ง “ถ้า..แล้ว” (->) ในการดำเนินการ จะได้ state == 1 -> state = state + 1 จากความสัมพันธ์การพึ่งพิงกันของข้อมูล จะทำให้ได้ state = (state == 1) + 1 จะได้ state == 2 เมื่อรวมแล้วจากฟังก์ชัน A นี้จะได้ state == 1 -> state == 2

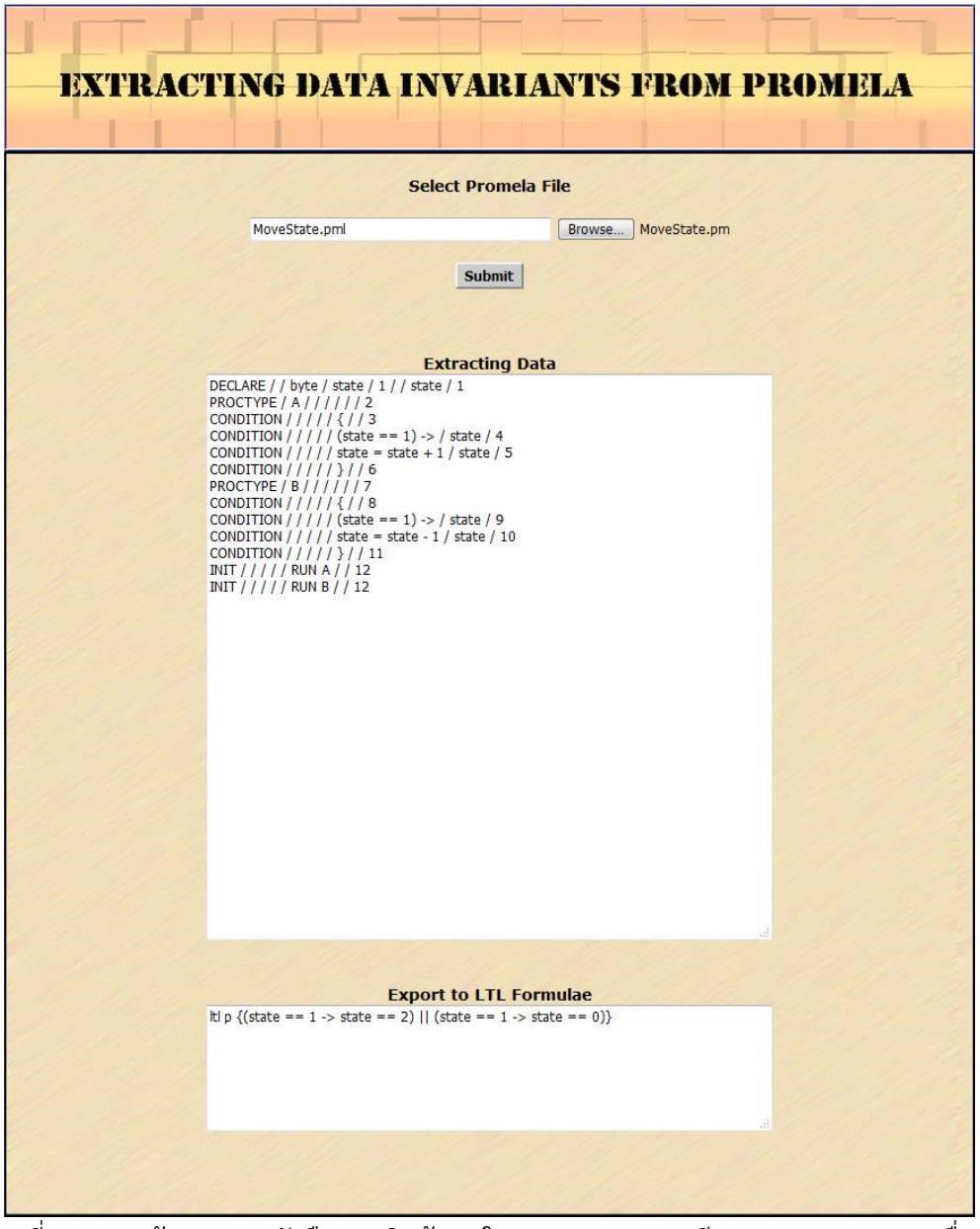
2. Proctype B พิจารณาเกณฑ์ state จะได้ state == 1 และ state = state – 1 ซึ่งเป็นตัวแปรตัวเดียวกัน จึงใช้คำสั่ง “ถ้า..แล้ว” (->) ในการดำเนินการ จะได้ state == 1 -> state =

state - 1 จากความสัมพันธ์การพึ่งพิงกันของข้อมูล จะทำให้ได้ state = (state == 1) - 1 จะได้ state == 0 เมื่อรวมแล้วจากฟังก์ชัน B นี้จะได้ state == 1 -> state == 0

เมื่อการนำความสัมพันธ์ที่เกี่ยวข้องกันนำมาเข้ารวมเข้าด้วยกันดังสูตร “DataInvResult(n) = S_1 v S_2 v ... v S_n” ซึ่งมาจากการรวมหลายๆข้อมูลมาหาจุดที่เหมือนกันหรือเกี่ยวข้องกันมารวมเข้าด้วยกันและนำสิ่งที่สนใจหลายๆสิ่งนั้นมารวมกันในภายหลังทำให้ได้ผลลัพธ์ของตัวยืนยงชนิดข้อมูลนี้จะได้ (state == 1 -> state == 2) || (state == 1 -> state == 0) ซึ่งจากโปรแกรมจะมีการตรวจสอบค่า state == 1 เสมอ แสดงให้เห็นว่าเมื่อเข้าที่ฟังก์ชันใดแล้วจะหยุดที่ฟังก์ชันนั้นๆเลย

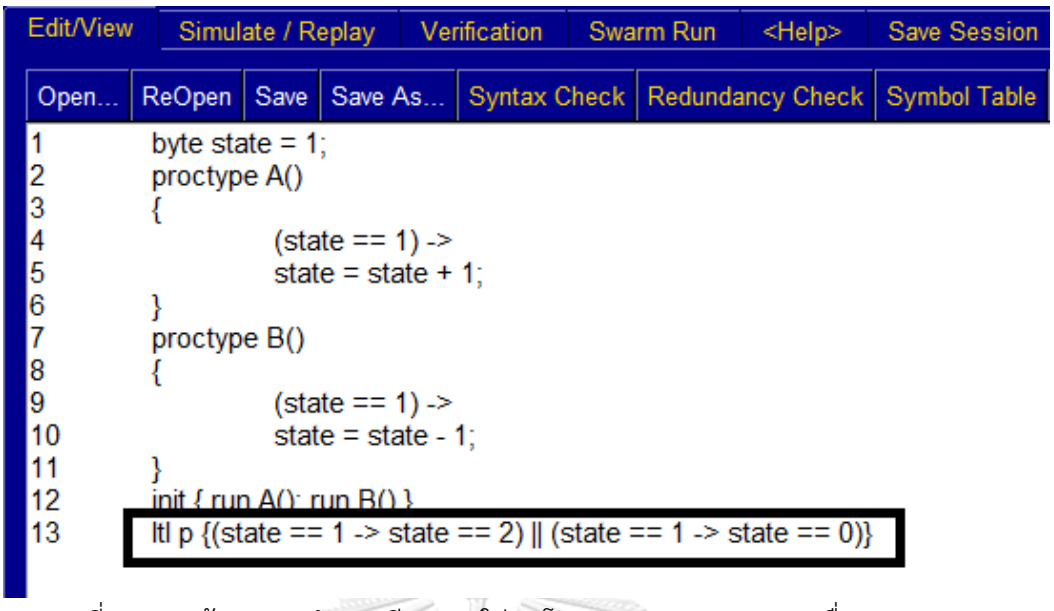
ซึ่งเมื่อนำมาสร้างเป็นสูตรแอลทีแอลจะได้ LTL []((state == 1 -> state == 2) || (state == 1 -> state == 0))

จากที่กล่าวมาข้างต้นเมื่อนำโปรแกรมมาประมวลผลบนเครื่องมือจะได้ดังรูปที่ 5-14



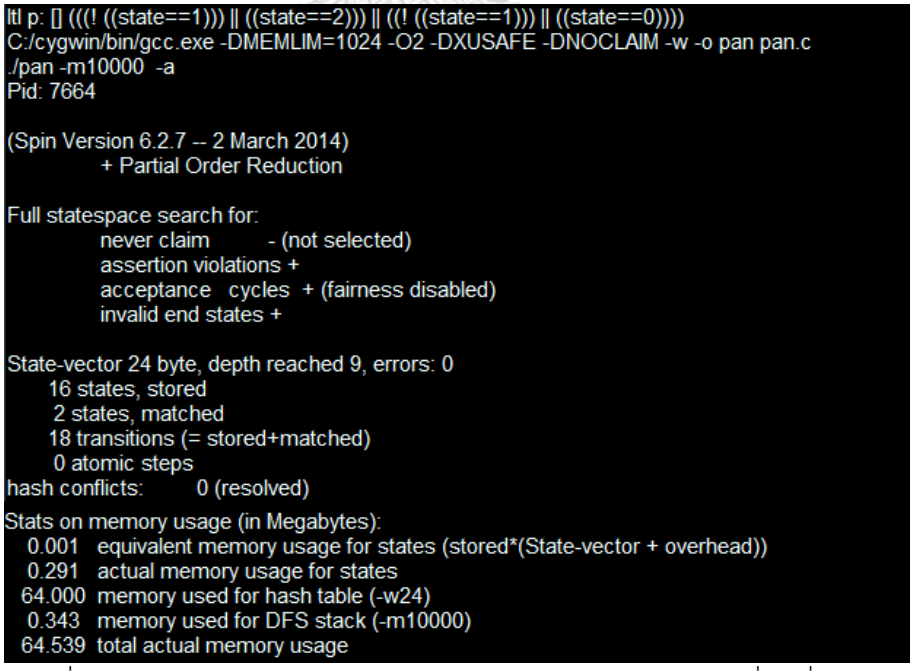
รูปที่ 5-14 หน้าจอแสดงตัวยืนยงชนิดข้อมูลในรูปแบบสูตรแอลทีแอลของระบบการเคลื่อนของสถานะ

ซึ่งเมื่อนำผลลัพธ์ของสูตรแอลทีแอลที่ได้มาจากการสกัดตัวยืนยงชนิดข้อมูลจากโพรเมลา นำมาใส่ในโค้ดต้นฉบับ และนำมาประมวลผลบนโปรแกรมสปิน ดังรูปที่ 5-15



รูปที่ 5-15 หน้าจอการนำแอลทีแอลมาใส่บนโพรเมลาของระบบการเคลื่อนของสถานะ

หากไม่ใช้สูตรแอลทีแอลในการตรวจสอบการทำงานเมื่อใช้สปินตรวจสอบการทำงานจะแสดงผลดังรูป 5-16 จะแสดงผลไม่เกิดข้อผิดพลาด เนื่องจากไม่ได้มีการตรวจสอบด้วยสูตรแอลทีแอล หากใช้สูตรแอลทีแอลและใช้สปินตรวจสอบการทำงานแสดงผลดังรูปที่ 5-17



รูปที่ 5-16 แสดงผลการตรวจสอบการไม่ใช้แอลทีแอลระบบการเคลื่อนที่ของสถานะ

```
unreached in proctype A
        (0 of 3 states)
unreached in proctype B
        (0 of 3 states)
unreached in init
        (0 of 3 states)
unreached in claim p
        _spin_nvr.tmp:3, state 6, "(!((((!((state==1))||(state==2))||(!((state==1))||(state==0)))))"
        _spin_nvr.tmp:3, state 6, "(1)"
        _spin_nvr.tmp:8, state 10, "-end-"
        (2 of 10 states)

pan: elapsed time 0.002 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-16 แสดงผลการตรวจสอบการไม่ใช้แอลทีแอลระบบการเคลื่อนที่ของสถานะ (ต่อ)

```
ltl p: [] (((! ((state==1))) || ((state==2))) || ((! ((state==1))) || ((state==0))))
C:/cygwin/bin/gcc.exe -DMEMLIM=1024 -O2 -DXUSAFE -w -o pan pan.c
./pan -m10000  -a
Pid: 7908

(Spin Version 6.2.7 -- 2 March 2014)
        + Partial Order Reduction

Full statespace search for:
        never claim         + (p)
        assertion violations + (if within scope of claim)
        acceptance   cycles  + (fairness disabled)
        invalid end states - (disabled by never claim)

State-vector 20 byte, depth reached 0, errors: 0
       1 states, stored
       0 states, matched
       1 transitions (= stored+matched)
       0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
   0.000  equivalent memory usage for states (stored*(State-vector + overhead))
   0.291  actual memory usage for states
  64.000  memory used for hash table (-w24)
   0.343  memory used for DFS stack (-m10000)
  64.539  total actual memory usage


unreached in proctype A
        MoveState.pml:4, state 1, "((state==1))"
        MoveState.pml:5, state 2, "state = (state+1)"
        MoveState.pml:6, state 3, "-end-"
        (3 of 3 states)
unreached in proctype B
        MoveState.pml:9, state 1, "((state==1))"
        MoveState.pml:10, state 2, "state = (state-1)"
        MoveState.pml:11, state 3, "-end-"
        (3 of 3 states)
unreached in init
        MoveState.pml:12, state 2, "(run B())"
        MoveState.pml:12, state 3, "-end-"
        (2 of 3 states)
unreached in claim p
        _spin_nvr.tmp:8, state 8, "-end-"
        (1 of 8 states)

pan: elapsed time 0.001 seconds
No errors found -- did you verify all claims?
```

รูปที่ 5-17 แสดงผลการตรวจสอบด้วยการใช้แอลทีแอลระบบการเคลื่อนที่ของสถานะ

จากผลของการทดสอบการทำงานของโปรแกรมด้วยการนำตัวยืนยงชนิดโพรเมลาในรูปแบบสูตรแอลทีแอลมาใช้ในการตรวจสอบมาเปรียบเทียบกับการทำงานของโปรแกรมโดยไม่ได้ใช้แอลทีแอลในการตรวจสอบ โดยเปรียบเทียบความแตกต่างได้ดังตารางที่ 5-6

ตารางที่ 5.6 ตารางแสดงการเปรียบเทียบผลการทดสอบระบบการเคลื่อนที่ของสถานะสำหรับการใช้สูตรแอลทีแอล กับไม่ใช้สูตรแอลทีแอล

ระบบผู้ใช้งานและเครื่องบริการ	ไม่ใช้สูตรแอลทีแอล	ใช้สูตรแอลทีแอล
State Vector	24 byte	20 byte
State Stored	16 state	1 state
States Match	2 state	0 state
Equivalent usage memory for states	0.001	0.000
Actual usage memory for states	0.291	0.291
Memory used for hash table	64.000	64.000
Memory used for DFS stack	0.343	0.343
Total actual memory usage	64.539	64.539
Elaspe Time	0.002 seconds	0.001 seconds

จะเห็นได้ว่ามีผลลัพธ์เหมือนกับที่ไม่ได้ใส่สูตรแอลทีแอลเข้าไป แสดงว่าโปรแกรมสามารถทำงานได้อย่างถูกต้องเหมือนเดิม และไม่มีข้อผิดพลาดเกิดขึ้น แต่จะเป็นการตรวจสอบหรือการป้องกันการเกิดข้อผิดพลาด เพื่อให้ทราบว่ามีที่ ณ เวลาใดๆ มีการทำงานก่อให้เกิดข้อผิดพลาดได้ แต่อาจเป็น ณ เวลาใดเวลาหนึ่งสั้นๆ ซึ่งผู้ใช้อาจมองไม่ทัน หรือมองไม่เห็น และจากตารางการเปรียบเทียบของกรณีศึกษาทั้ง 3 กรณีศึกษา จะเห็นได้ว่าในการทำงานของโปรแกรมที่มีแอลทีแอลจะมีสถานะการทำงานมากกว่า เนื่องจากมีการทำงานมากขึ้น เพราะมีการตรวจสอบข้อมูลเพิ่มขึ้นมาตามความใหญ่ และความซับซ้อนของโปรแกรม แต่จะมีการใช้ขนาดของความจำของสถานะน้อยลง

สำหรับโปรแกรมทั่วๆไป แต่กรณีที่โปรแกรมมีการสุ่มเลือกการทำงาน จะทำให้เป็นการป้องกัน หรือการบังคับไม่ให้เกิดคุณสมบัติความไม่ปลอดภัย ทำให้มีการทำงานที่สถานะน้อยลง ซึ่งทั้งหมดนี้เพื่อเป็นการป้องกัน และเพื่อความปลอดภัยของการทำงานไม่ให้เกิดข้อผิดพลาด และควรมีการหาตัวอย่างค้าน เพื่อหาตัวอย่างที่มีโอกาสทำให้โปรแกรมเกิดข้อผิดพลาดได้ การใส่สูตรแอลทีแอลเหมือนเป็นกฎเกณฑ์ที่ใช้กำหนดว่าโปรแกรมที่มีข้อจำกัดอะไรบ้าง ไม่ให้ค่าต่างๆเกินหรือต่ำกว่ากำหนด เพราะจะทำให้ไม่ตรงตามความต้องการ หรืออาจทำให้เกิดข้อผิดพลาดได้ โดยเป็นวิธีที่ใช้ในการตรวจสอบการทำงานของโปรแกรมก่อนนำไปใช้จริง เพื่อให้มั่นใจได้ว่าโปรแกรมจะทำงานได้อย่างถูกต้อง

บทที่ 6

สรุปผลงานวิจัยและข้อเสนอแนะ

จากการศึกษา วิจัยและพัฒนาเครื่องมือสำหรับสนันการสกัดตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลา โดยการแบ่งส่วนของโปรแกรมให้มีขนาดเล็กลง และกรณีทดสอบของงานวิจัยนี้สามารถสรุปผลการวิจัย ข้อจำกัดของเครื่องมือ และแนวทางในการพัฒนาต่อไปในอนาคต โดยมีรายละเอียดดังนี้

6.1 สรุปผลการวิจัย

งานวิจัยนี้นำเสนอการสกัดตัวยืนยงชนิดแบบข้อมูลสำหรับภาษาโพรเมลา โดยการแบ่งส่วนของข้อมูลของโค้ดของโปรแกรมก่อนเพื่อสามารถหาตัวยืนยงได้ง่ายขึ้น เพราะมีการแบ่งส่วนที่ชัดเจนขึ้น ลดความซับซ้อนของข้อมูล เนื่องจากบางโปรแกรมมีขนาดใหญ่ หรือมีความซับซ้อนมาก ทำให้การหาค่าของตัวยืนยงของข้อมูล หรือข้อจำกัดของโปรแกรมนั้นเป็นไปได้ยาก หรือหาได้ไม่ครบถ้วนจากเงื่อนไขต่างๆของความต้องการทั้งหมด ทำให้พลาดถึงบางจุดที่อาจก่อให้โปรแกรมทำงานผิดพลาดได้ จึงได้พัฒนาเครื่องมือนี้ขึ้นมาเพื่อช่วยเหลือนักพัฒนาโปรแกรม ในการหาตัวยืนยงชนิดข้อมูลนี้ เครื่องมือจัดอยู่ในรูปแบบของเว็บแอพพลิเคชันให้ผู้ใช้งานสามารถใช้งานได้ง่ายโดยการเลือกไฟล์โปรแกรม และกดปุ่มแปลงไฟล์เพื่อให้ได้ผลลัพธ์เป็นตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลาในรูปแบบของสูตรแอลทีแอล เพื่อให้ผู้ใช้งานนำแอลทีแอลนี้ไปใช้ต่อกับโปรแกรมโพรเมลานั้นๆ

6.2 ข้อจำกัดของงานวิจัย

เครื่องมือสำหรับการสกัดตัวยืนยงชนิดข้อมูลสำหรับภาษาโพรเมลานี้มีข้อจำกัดดังต่อไปนี้

1) เครื่องมือสามารถนำเข้าข้อมูลชนิดโพรเมลา หรือชนิดข้อมูลอักษรเท่านั้น

2) เครื่องมือนี้แสดงผลลัพธ์ในรูปแบบของสูตรแอลทีแอลที่มีคุณสมบัติความปลอดภัยเท่านั้น

3) เครื่องมือนี้แสดงผลลัพธ์ตัวยืนยงชนิดข้อมูลของโพรเมลาในรูปของแอลทีแอล ผู้ใช้จะต้องทำการคัดลอกและนำไปแปะใส่โค้ดของผู้ใช้งานอีกที

4) โปรแกรมโพรเมลาที่จะนำมาประมวลผลจะต้องเป็นโปรแกรมที่ประมวลได้ถูกต้องเท่านั้น

5) ไม่รองรับการทำงานโพรเมลาที่เป็นแบบสุ่มการทำงาน การทำงานแบบช่องทาง (Channel)

6.3 ข้อเสนอแนะ และแนวทางการพัฒนาต่อ

เครื่องมือการสกัดตัวยืนยงชนิดข้อมูลสำหรับโพรเมลานี้ มีการตรวจสอบความถูกต้องของการทำงานของข้อมูลเพื่อไม่ให้เกิดข้อผิดพลาดนี้ สามารถนำไปต่อยอดหรือพัฒนาเพิ่มเติมในอนาคตได้ดังนี้

1) สามารถนำแนวทางวิธีการสร้างสูตรสำหรับจัดการเงื่อนไข เพื่อป้องกันไม่ให้เกิดข้อผิดพลาดของการทำงานของโปรแกรมภาษาอื่นๆได้ ที่ไม่ได้อยู่ในรูปแบบของสูตรแอลทีแอลได้

2) การแบ่งส่วนโปรแกรมนี้สามารถนำไปประยุกต์ใช้กรณีผู้ใช้งานสามารถนำไปใช้ด้านอื่นๆ เช่น การตรวจสอบจุดที่ผิดพลาดของโปรแกรม หรือหาจุดที่ผู้ใช้สนใจหรือตัวแปรหรือเงื่อนไขที่เกี่ยวข้องทั้งหมด เพื่อลดความซับซ้อน และลดเวลาที่ใช้ในการค้นหาทั้งหมด

3) สามารถนำเข้าข้อมูลภาษาโพรเมลา ได้หลากหลายรูปแบบ ที่นอกเหนือรูปแบบที่กำหนดไว้เท่านั้น

รายการอ้างอิง

[1] Fogarty, J., Invariant theory, Lecture Notes in Mathematics, vol. 585, Springer-Verlag, New York–Heidelberg–Berlin, 1977

[2] Raul Santelices, et al., Quantitative Program Slicing : Separating Statements by Relevance , Proceedings of the 2013 International Conference on Software Engineering, 2013

[3] Chen duanzhi, Program Slicing, International Forum on Information Technology and Applications, 2010

[4] Chen duanzhi, A collection of Program Slicing, International Conference on Computer Application and System Modeling (ICCASM 2010), 2010

[5] BogdanKorel and SatishYalamanchili, Forward Computation of Dynamic Program Slices, ACM SIGSOFT international symposium on software testing and analysis, 1994

[6] BogdanKorel and JuergenRilling, Dynamic Program Slices Methods, Information and Software Technology, 1998

ประวัติผู้เขียน

นายวิสุทธิ์ สุขศรีบางเตย เกิดวันที่ 15 พฤษภาคม 2534 สำเร็จการศึกษาระดับปริญญาตรี หลักสูตรวิทยาศาสตร์บัณฑิต (วท.บ.) สาขาเทคโนโลยีสารสนเทศ คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2555ประสบการณ์ พนักงานองค์กรเอกชน บริษัทธนาคารกรุงเทพ จำกัด ตำแหน่งนักพัฒนาโปรแกรม เข้าศึกษาต่อระดับปริญญามหาบัณฑิต ปีการศึกษา 2558 หลักสูตรวิทยาศาสตรมหาบัณฑิต (วท.ม.) สาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

บรรณานุกรม

ประวัติผู้เขียน