การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี

โดย

ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2558

การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี

โดย

ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
ปีการศึกษา 2558

FORMAL VERIFICATION OF GO PROGRAMMING LANGUAGE USING CSP

BY

ACTING 2, LT. ANUCHIT PRASERTSANG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF SCIENCE (COMPUTER SCIENCE)

DEPARTMENT OF COMPUTER SCIENCE

FACULTY OF SCIENCE AND TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2015

มหาวิทยาลัยธรรมศาสตร์
คณะวิทยาศาสตร์และเทคโนโลยี

วิทยานิพนธ์

ของ

ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์

เรื่อง

การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี

ได้รับการตรวจสอบและอนุมัติ ให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)

เมื่อ วันที่ 7 กรกฎาคม พ.ศ. 2559

ประธานกรรมการสอบวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร.เสาวลักษณ์ วรรธนาภา)

กรรมการและอาจารย์ที่ปรึกษาวิทยานิพนธ์ _____
(อาจารย์ ดร.เด่นดวง ประดับสุวรรณ)

กรรมการสอบวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร.กษิดิศ ชาญเชี่ยว)

กรรมการสอบวิทยานิพนธ์ _____
(ผู้ช่วยศาสตราจารย์ ดร.จักริน สุขสวัสดิ์ชน)

คณบดี _____
(รองศาสตราจารย์ ปกรณ์ เสริมสุข)

หัวข้อวิทยานิพนธ์	การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี
ชื่อผู้เขียน	ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์
ชื่อปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา/คณะ/มหาวิทยาลัย	วิทยาการคอมพิวเตอร์ วิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์	อาจารย์ ดร.เด่นดวง ประดับสุวรรณ
ปีการศึกษา	2558

บทคัดย่อ

ภาษาโกเป็นภาษาโปรแกรมที่ถูกออกแบบมาเพื่อการพัฒนาซอฟต์แวร์ระบบเป็นหลัก ตัวภาษามีการสนับสนุนการทำงานแบบภาวะพร้อมกันและมีพฤติกรรมการทำงานเชิงไม่กำหนด ทำให้การทวนสอบโปรแกรมที่ถูกพัฒนาด้วยภาษาโกยาก ดังนั้นงานวิจัยนี้จึงได้นำเสนอการทวนสอบเชิงรูปนัยด้วยซีเอสพี เพื่อยืนยันความถูกต้องในการทำงานของโปรแกรมที่ถูกพัฒนาด้วยภาษาโก การทวนสอบแบ่งเป็น 2 ขั้นตอน ได้แก่ การสร้างโมเดลเชิงรูปนัยด้วยซีเอสพีสำหรับภาษาโก และการทวนสอบโมเดลเชิงรูปนัยของคุณลักษณะโปรแกรมและโปรแกรม เครื่องมือที่ใช้สำหรับการทวนสอบได้แก่ เครื่องมือเอฟดีอาร์ (University of Oxford, 2015) จากผลการทดลองกับกรณีศึกษาแสดงให้เห็นถึงประสิทธิภาพของวิธีการทวนสอบที่ได้นำเสนอ

คำสำคัญ: ภาษาโก, ซีเอสพี

Thesis Title	Formal Verification of GO Programming Language Using CSP
Author	Acting 2, Lt. Anuchit Prasertsang
Degree	Master of Science
Major Field/Faculty/University	Computer Science Faculty of Science and Technology Thammasat University
Thesis Advisor	Dr. Denduang Pradubsuwun
Academic Years	2015

ABSTRACT

Go is a programming language which is mainly designed for the concurrent programming. It supports concurrency derived from Hoare's Communicating Sequential Processes (CSP). Since a go-routine introduced by Go allows us to execute program simultaneously, a program behavior is asynchronous. It likely causes a failure. A formal verification is needed to assure the correctness of the program. Therefore, we have proposed a method to verify a program implemented by Go. It is divided into 2 steps. Firstly, a formal model of Go is constructed. It is represented in CSP. Then, a refinement checker for CSP or Failures Divergence Refinement (FDR) in (University of Oxford, 2015) is used as a verification tool to check the correctness between Go program and its specification. We also give some experimental results to show effectiveness of our method.

Keywords: Go programming language (Go), CSP, FDR

กิตติกรรมประกาศ

ขอขอบพระคุณ อาจารย์ ดร.เด่นดวง ประดับสุวรรณ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ให้ความสนับสนุน ให้คำปรึกษาแนะนำแนวทาง ส่งเสริมให้มีกำลังใจ สละเวลาอันมีค่าให้ความช่วยเหลือ ในการทำวิทยานิพนธ์ฉบับนี้ให้สำเร็จลุล่วงไปด้วยดี ขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณคณาจารย์ในภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ทุกท่านที่ประสิทธิ์ประสาทวิชาที่เป็นความรู้อันมีค่ายิ่งให้แก่ผู้วิจัย ช่วยเพิ่มคุณค่าให้แก่วิทยานิพนธ์ฉบับนี้มากยิ่งขึ้น

ขอขอบพระคุณคณาจารย์ในภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ทุกท่านที่ประสิทธิ์ประสาทวิชาที่เป็นความรู้อันมีค่ายิ่งให้แก่ผู้วิจัย ช่วยเพิ่มคุณค่าให้แก่วิทยานิพนธ์ฉบับนี้มากยิ่งขึ้น

ขอขอบพระคุณบิดาและมารดาเป็นอย่างสูง พร้อมทั้งญาติพี่น้อง ที่ส่งเสริมสนับสนุนการศึกษาเล่าเรียน เสริมสร้างกำลังใจตลอดมา จนงานวิจัยได้สำเร็จลุลวงไปได้ด้วยดี

ท้ายนี้ผู้วิจัยขอขอบพระคุณทุกท่านที่ได้เอ่ยนามข้างหรือที่ไม่ได้เอ่ยนามก็ดี และ ขออวยพระทุกท่าน กราบขอบุญบารมีรัศมีกำลังฤทธิอำนาจสิทธิเฉียบขาดแห่งองค์พระสัมมาสัมพุทธเจ้าทุกๆ พระองค์ในพระนิพพาน และบุญบารมีที่ท่านทั้งหลายได้กระทำไว้ ช่วยประคับประคองให้มีสุขภาพแข็งแรง อายุขัยยืนยาว ทุกข์โศกโรคภัยอย่าได้มาแผ่วพาน ให้มีแต่ความสุขสมหวังดังใจเทอญ

ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์

สารบัญ

สารบัญ

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

การทำงานแบบภาวะพร้อมกัน (concurrent) เป็นการทำงานที่ช่วยเพิ่มประสิทธิภาพการประมวลผลของเครื่องคอมพิวเตอร์ ดังนั้นภาษาที่นำมาใช้เพื่อพัฒนาซอฟต์แวร์สำหรับควบคุมให้เครื่องคอมพิวเตอร์ทำงาน ก็เป็นส่วนสำคัญที่จะทำให้สามารถสร้างซอฟต์แวร์ให้สามารถทำงานแบบภาวะพร้อมกันได้อย่างมีประสิทธิภาพ

ภาษาโก (GO) เป็นภาษาที่ถูกนำเสนอโดยบริษัท กูเกิล (Google) ซึ่งออกแบบมาให้สามารถรองรับการพัฒนาซอฟต์แวร์ระบบที่ทำงานแบบภาวะพร้อมกันได้เป็นหลัก (The Go Authors, 2011) โดยการทำงานแบบภาวะพร้อมของภาษาโกนั้น ถูกออกแบบและพัฒนาตามแนวคิดของ Hoare's Communicating Sequential Processes (Hoare C A R., 1987) ด้วยการนำเสนอคำสั่งที่สนับสนุนการทำงานแบบภาวะพร้อมกัน ได้แก่ คำสั่งโก (go) เมื่อระบุคำสั่งโกไว้ในภายในโปรแกรม การทำงานของโปรแกรมจะเป็นแบบภาวะพร้อมกัน ซึ่งจะมีผลทำให้เกิดพฤติกรรมเชิงไม่กำหนด และมีผลทำให้การทวนสอบความถูกต้องของโปรแกรมทำได้ยาก

ในปัจจุบันการทวนสอบซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโกจะใช้การจำลองพฤติกรรมของโปรแกรมเพื่อตรวจสอบความถูกต้องเท่านั้น (simulation verification) แต่การทวนสอบด้วยวิธีดังกล่าว ไม่สามารถรับประกันความถูกต้องในการทำงานของซอฟต์แวร์ได้ครอบคลุมครบทุกกรณี และความถูกต้องของโปรแกรมที่ถูกทวนสอบยังขึ้นอยู่กับเวกเตอร์ทดสอบอินพุท (input test vector) ที่ถูกสร้างโดยผู้พัฒนาหรือผู้ทดสอบโปรแกรมเองทำให้เกิดความอคติกับการทวนสอบซอฟต์แวร์

อย่างไรก็ตาม ในการทวนสอบซอฟต์แวร์ทั่วๆไป ยังมีวิธีการทวนสอบอีกรูปแบบหนึ่งได้แก่ วิธีการทวนสอบเชิงรูปนัย (formal verification) ซึ่งจะใช้วิธีรูปนัย (formal method) ในการทวนสอบซอฟต์แวร์ ทำให้สามารถรับประกันความถูกต้องในการทำงานของซอฟต์แวร์ได้ครอบคลุมทุกกรณี และแก้ไขจุดอ่อนของการทวนสอบด้วยวิธีการจำลองพฤติกรรมของโปรแกรม

ดังนั้นงานวิจัยนี้จึงนำเสนอวิธีการทวนสอบเชิงรูปนัยสำหรับทวนสอบซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโก โดยจะใช้วิธีรูปนัย ได้แก่ ซีเอสพี (CSP; Communication Sequential Processes) ทำให้สามารถรับประกันความถูกต้องในการทำงานของซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโกได้

1.2 วัตถุประสงค์การวิจัย

1. เพื่อออกแบบและพัฒนาอัลกอริทึมสำหรับสร้างแบบจำลองเชิงรูปนัย (Formal model) ของรหัสต้นฉบับ (Source code) ที่ถูกพัฒนาด้วยภาษาโกให้อยู่ในรูปของซีเอสพี
2. เพื่อทวนสอบความถูกต้องการทำงานแบบภาวะพร้อมกันของซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโก

1.3 ขอบเขตการวิจัย

1. การทวนสอบภาษาโกสำหรับงานวิจัยนี้จะครอบคลุมคำสั่งพื้นฐานทั้งหมด 6 คำสั่ง ได้แก่ คำสั่ง if-else คำสั่ง func คำสั่ง go คำสั่ง var คำสั่ง for และคำสั่ง channel
2. การทวนสอบจะใช้เครื่องมือเอฟดีอาร์ (FDR; Failures-Divergence Refinement) เป็นเครื่องมือทวนสอบความถูกต้องของซีเอสพี ซึ่งจะครอบคลุมการทวนสอบคุณสมบัติความปลอดภัย (Safety property) (Schneider S., 1999) เท่านั้น

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. เพื่อเพิ่มความน่าเชื่อถือและความถูกต้องให้กับซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโก
2. เพื่อลดความผิดพลาดที่อาจเกิดขึ้นกับซอฟต์แวร์ที่ถูกนำไปใช้งานจริง
3. เพื่อทำให้ต้นทุนในการทดสอบและพัฒนาซอฟต์แวร์ลดลง

บทที่ 2
วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะอธิบายทฤษฎีที่เกี่ยวข้องและงานวิจัยที่เกี่ยวข้องกับการทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี ทฤษฎีที่เกี่ยวข้องกับงานวิจัยนี้จะมีทั้งหมด 3 เรื่อง ได้แก่ ภาษาโก ซีเอสพี และการทวนสอบซอฟต์แวร์ด้วยเครื่องมือเอฟดีอาร์

2.1 ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องกับงานวิจัยเรื่องการทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี มีทั้งหมด 3 เรื่อง ได้แก่ ภาษาโก ซีเอสพี และการทวนสอบซอฟต์แวร์ด้วยเครื่องมือเอฟดีอาร์

ภาษาโก

ภาษาโก คือ ภาษาโปรแกรมที่เป็นโอเพนซอร์ส ถูกสร้างขึ้นจากบริษัทกูเกิล ในปี ค.ศ. 2007 โดย Robert Griesemer, Rob Pike, และ Ken Thompson เผยแพร่อย่างเป็นทางการ ในปี ค.ศ. 2009 ภาษาโกมีไวยากรณ์คล้ายกับภาษาซี (C) แต่ก็ยอมให้ผู้พัฒนาซอฟต์แวร์สามารถเขียนโปรแกรมในรูปแบบของภาษาสคริปต์ได้ (script language)

ภาษาโกถูกออกแบบมาสำหรับการเขียนซอฟต์แวร์ระบบเป็นหลัก มีลักษณะเป็นแบบ statically typed language หมายถึง ชนิดข้อมูลจะถูกกำหนดในช่วงเวลาของการคอมไพล์โปรแกรม (compile time) มี garbage collection เพื่อช่วยเพิ่มประสิทธิภาพการทำงานของโปรแกรม มีการสนับสนุนการคอมไพล์โปรแกรมเพื่อนำไปใช้บนแพลตฟอร์มอื่นๆ (cross-compiler) และมีการสนับสนุนการทำงานแบบภาวะพร้อมกัน ที่ถูกออกแบบขึ้นมาตามแนวคิดของ Hoare's Communicating Sequential Processes นอกจากนี้การเขียนโปรแกรมด้วยภาษาโกนั้นง่าย เพราะบางส่วนของภาษาโกมีลักษณะคล้ายคลึงกับภาษาสคริปต์

ชุดคำสั่งภาษาโก

คำสั่งในภาษาโก (The Go Authors, 2011) มีคำสั่งจำนวนมากให้เลือกใช้งานตามความเหมาะสม แต่ในงานวิจัยนี้ขอนำเสนอ 6 คำสั่งได้แก่ คำสั่ง if-else คำสั่ง func คำสั่ง go คำสั่ง var คำสั่ง for และคำสั่ง channel ซึ่งคำสั่งเหล่านี้เพียงพอต่อการพัฒนาโปรแกรมด้วยภาษาโก

(1) คำสั่ง if-else เป็นคำสั่งสำหรับเลือกเส้นทางการทำงานของโปรแกรมตามเงื่อนไขที่ได้กำหนดไว้ รูปแบบของคำสั่ง if-else แสดงดังภาพที่ 2.1 จากภาพถ้าเงื่อนไข exp เป็นจริง กลุ่มคำสั่ง

ที่อยู่ภายใต้ if จะทำงาน หากเงื่อนไขเป็นเท็จกลุ่มคำสั่งที่อยู่ภายใต้ else จะถูกทำงานแทน โครงสร้างของคำสั่ง if-else แสดงดังภาพที่ 2.1

```
if exp {
    //body if
} else{
  // body else
}
```

ภาพที่ 2.1 โครงสร้างคำสั่ง if-else

(2) คำสั่ง func เป็นคำสั่งสำหรับสร้างฟังก์ชันเพื่ออำนวยความสะดวกในการประมวลผลกลุ่มคำสั่งต่างๆ รูปแบบของคำสั่ง func แสดงดังภาพที 2.2 จากภาพจะเห็นได้ว่าฟังก์ชันชื่อว่า A จะสามารถรับค่าอยู่ในรูปแบบของ ชื่อว่าตัวแปร param1 ตามด้วย ptype1 คือชนิดของตัวแปร param1 และสามารถรับได้หลายๆค่าโดยคั่นด้วยเครื่องหมายจุลภาค (,) และสามารถคืนค่าได้โดยตัวแปร val1 ตามด้วยชนิดของตัวแปร vtype1 และสามารถคืนค่าได้หลายๆค่าโดยคั่นด้วยเครื่องหมายจุลภาค โครงสร้างคำสั่ง func แสดงดังภาพที่ 2.2

```
func A(param1 ptype1 | [param... ptype... ,] ) (val1 vtype1 | [val.. vtype... ,]) {
  //body func
}
```

ภาพที่ 2.2 โครงสร้างคำสั่ง func

(3) คำสั่ง go เป็นคำสั่งที่กำหนดให้โปรแกรมทำงานแบบภาวะพร้อมกันอย่างอัตโนมัติ ตัวอย่างรูปแบบของคำสั่งแสดงได้ในภาพที่ 2.3 การทำงานของคำสั่งโก โปรแกรมจะถูกแบ่งออกเป็นโพรเซสย่อยๆหลายอันทำงานแบบภาวะพร้อมกัน ซึ่งภาษาโกเรียกโพรเซสย่อยแต่ละอันว่า goroutines โปรแกรมเมอร์ไม่จำเป็นต้องรู้เรื่องของการแบ่งแยกโปรแกรมออกเป็นโพรเซสย่อยๆ ตัวภาษาโกจะเป็นตัวจัดการให้ โครงสร้างการใช้ของคำสั่ง go แสดงดังภาพที่ 2.3

```
func A() { //body func}
go A()
```

ภาพที่ 2.3 โครงสร้างคำสั่ง go

(4) คำสั่งที่ใช้สำหรับการประกาศตัวแปร แสดงดังภาพที่ 2.4 จากภาพเราสามารถประกาศตัวแปรได้ 2 รูปแบบ 1) การประกาศแบบใช้คำสั่ง var และ 2) การประกาศแบบย่อโดยใช้เครื่องหมาย := ในการประกาศตัวแปร ทั้งสองแบบมีความแตกต่างกันที่การประกาศในรูปแบบที่ 1 สามารถประกาศโดยระบุชนิดของข้อมูลหรือหากไม่ระบุชนิดข้อมูลภาษาโกจะทำการกำหนดชนิดข้อมูลให้ตามค่าที่อยู่ทางขาวมือของเครื่องหมายเท่ากับโดยอัตโนมัติและสามารถประกาศได้ทั้งภายในและภายนอกฟังก์ชัน ส่วนการประกาศในรูปแบบที่ 2 นั้นสามารถประกาศได้เฉพาะภายในฟังก์ชันเท่านั้นและไม่ต้องระบุชนิดของข้อมูลโดยภาษาโกจะกำหนดชนิดข้อมูลให้อัตโนมัติเช่นกัน โครงสร้างการประกาศตัวแปร แสดงดังภาพที่ 2.4

```
// การประกาศโดยใช้ var แบบไม่ระบุชนิด
var num = 10
// การประกาศโดยใช้ var แบบระบุชนิด
var num int = 10
// การประกาศแบบย่อ
num := 10
```

ภาพที่ 2.4 โครงสร้างการประกาศตัวแปร

(5) คำสั่ง for เป็นคำสั่งที่ทำงานแบบวนซ้ำ โดยจะทำการวนซ้ำถ้าหากเงื่อนไขเป็นจริง รูปแบบของคำสั่งแสดงดังภาพที่ 2.5 จากภาพถ้า condition เป็นจริงกลุ่มคำสั่งที่อยู่ภายใต้ for จะทำงานวนไปเรื่อยๆจนกว่าเงื่อนไขจะเป็นเท็จ โดยในภาษาโกนั้นจะไม่มีคำสั่ง while หรือ do-while ดังเช่นภาษาอื่น มีเพียงคำสั่ง for ที่สามารถเขียนแทนคำสั่งเหล่านั้นได้ โครงสร้างของคำสั่ง for แสดงดังภาพที่ 2.5

```
// คล้าย for ในภาษา C
for init; condition; post { }

// คล้าย while ในภาษา C
for condition { }

// คล้าย for(;;) ในภาษา C
for { }
```

ภาพที่ 2.5 โครงสร้างคำสั่ง for

(6) คำสั่ง channel เป็นคำสั่งที่ใช้สำหรับเป็นช่องทางในการติดต่อสื่อสารรับหรือส่งข้อมูลระหว่าง goroutines ที่ทำงานแบบภาวะพร้อมกันโดยจะใช้เครื่องหมาย “<-” ในการระบุว่าต้องการรับหรือส่งข้อมูล ดังแสดงในภาพที่ 2.6

```
// รับค่าจาก channel
<- channel

// ส่งค่าเข้า channel
channel <-
```

ภาพที่ 2.6 โครงสร้างคำสั่ง channel

ซีเอสพี

ซีเอสพี (CSP; Communicating Sequential Processes) เป็นวิธีรูปนัยซึ่งถูกออกแบบมาเพื่ออธิบายรูปแบบการปฏิสัมพันธ์กันของโพรเซส (process) ที่เป็นอิสระต่อกัน มีส่วนต่อประสาน (interface) หรือช่องสัญญาณ (channels) ในการติดต่อสื่อสารกันด้วยการส่งข้อความ(message) หรือการเกิดเหตุการณ์ (event) โดยทั่วไปซีเอสพีจะแสดงการทำงานของโพรเซสในรูปแบบของ event prefix (Schneider S., 1999) ซึ่งแสดงได้ดังนี้ “a → P” หมายถึง ถ้า P เป็นโพรเซสซีเอสพี และ a เป็นเหตุการณ์ที่เกิดขึ้นในช่องสัญญาณของโพรเซส P แล้ว เมื่อเหตุการณ์ a เกิดขึ้นแล้วจะทำให้เกิดโพรเซส P ขึ้น ซีเอสพีจะมีคำสั่งที่ใช้สำหรับอธิบายการทำงานของระบบหลายคำสั่ง ซึ่งจะนำเสนอในส่วนของเนื้อหาในตอนถัดไป

ชุดคำสั่งซีเอสพี

ซีเอสพีมีคำสั่งจำนวนมากให้ใช้อธิบายพฤติกรรมการทำงานของระบบ ในส่วนนี้จะขอนำเสนอคำสั่งที่นำมาใช้ในงานวิจัย ซึ่งมีทั้งหมด 6 ชุดคำสั่ง ได้แก่ คำสั่งอินพุท, คำสั่งเอาท์พุท, คำสั่งการเรียกซ้ำ, คำสั่งแบบวนซ้ำ, คำสั่งแบบเลือก, คำสั่งแบบภาวะพร้อมกัน ซึ่งคำสั่งทั้งหมดนี้เป็นชุดคำสั่งพื้นฐานที่สามารถนำมาใช้อธิบายพฤติกรรมการทำงานของระบบได้

(1) คำสั่งอินพุท (input) เป็นคำสั่งที่ใช้อธิบายการรับค่าเข้าสู่ระบบ ผ่านช่องสัญญาณของโพรเซส โดยจะใช้เครื่องหมาย “?” เป็นสัญลักษณ์แสดงเหตุการณ์อินพุท ตัวอย่างคำสั่งอินพุท เช่น “c?x → P” หมายถึง การรับค่าของตัวแปร x ผ่านช่องสัญญาณ c แล้วทำให้เกิดโพรเซส P

(2) คำสั่งเอาท์พุท (output) เป็นคำสั่งที่ใช้อธิบายการนำค่าออกจากระบบ ผ่านทางช่องสัญญาณ โดยจะใช้เครื่องหมาย “!” เป็นสัญลักษณ์แสดงเหตุการณ์เอาท์พุท ตัวอย่างคำสั่ง

เอาท์พุท เช่น “c!v → P” หมายถึง การแสดงค่าของตัวแปร x ผ่านช่องสัญญาณ c แล้วทำให้เกิดโพรเซส P

(3) คำสั่งการเรียกซ้ำ (recursion) เป็นคำสั่งที่ใช้อธิบายการทำงานแบบวนซ้ำ รูปแบบทั่วไปของคำสั่ง recursion เขียนได้ดังนี้ “N = P” หมายถึง การนิยามโพรเซส N ด้วยนิยามของโพรเซส P และในนิยามของโพรเซส P จะมีนิยามของโพรเซส N เป็นองค์ประกอบ ตัวอย่างเช่น

LIGHT = on → off → LIGHT

จากนิพจน์ข้างต้น จะเห็นได้ว่าโพรเซส LIGHT เทียบได้กับโพรเซส N ที่ถูกนิยามในโพรเซส P ที่เทียบได้กับ on → off → LIGHT ซึ่งจะทำให้พฤติกรรมของระบบเกิดการทำงานแบบวนซ้ำขึ้น

(4) คำสั่งแบบวนซ้ำ (mutual recursion) เป็นคำสั่งที่ใช้สำหรับอธิบายการทำงานแบบวนซ้ำเหมือนกับคำสั่ง recursion เพียงแต่ไม่แสดงชื่อของโพรเซส N ไว้ในนิยามของโพรเซส P อย่างชัดเจน ตัวอย่างเช่น

LIGHT = on → ON

ON = off → LIGHT

(5) คำสั่งแบบเลือก (choice) เป็นคำสั่งที่ใช้อธิบายการทำงานแบบเลือก หมายถึง ณ เวลาใดๆ ระบบอาจจะต้องตัดสินใจเลือกการทำงานของระบบจากทางเลือกที่กำหนดไว้ โดยคำสั่ง ซึ่งแบ่งเป็น 2 แบบได้แก่ คำสั่งแบบ internal choice และคำสั่งแบบ external choice ทั้งสองคำสั่งมีความแตกต่างกันตรงบุคคลที่ทำหน้าที่เป็นผู้ตัดสินใจเลือกการทำงานของระบบจากทางเลือกที่กำหนดไว้ กรณีที่คำสั่งแบบ internal choice บุคคลที่ทำหน้าที่ตัดสินใจได้แก่ ระบบ และกรณีที่คำสั่งแบบ external choice บุคคลที่ทำหน้าที่ตัดสินใจได้แก่ ผู้ใช้ รูปแบบทั่วไปของคำสั่ง choice เขียนแสดงได้ดังนี้

คำสั่งแบบ internal choice จะเขียนอยู่ในรูป “P1 ⊓ P2” ตัวอย่างเช่น ROUTER = VIA_A ⊓ VIA_B หมายถึง การเลือกเส้นทางให้ผ่านระหว่าง ROUTER A หรือ ROUTER B นั้นระบบจะทำหน้าที่เป็นผู้ตัดสินใจ

คำสั่งแบบ external choice จะเขียนอยู่ในรูป “P1 □ P2” ตัวอย่างเช่น SERVICE = BUS_29 □ BUS_510 หมายถึง การเลือกใช้บริการรถประจำทางสาย 29 หรือรถประจำทางสาย 510 นั้นผู้ใช้จะทำหน้าที่เป็นผู้ตัดสินใจ

(6) คำสั่งแบบภาวะพร้อมกัน (concurrency) เป็นคำสั่งที่ใช้อธิบายการทำงานแบบภาวะพร้อมกันซึ่งแบ่ง 3 แบบได้แก่ คำสั่งแบบ alphabetized parallel, interleaving, interface parallel

1) คำสั่งแบบ alphabetized parallel จะเขียนอธิบายอยู่ในรูปแบบของ $G \,{}_{\{A\}}\|_{\{B\}} H$ โดยที่ G และ H เป็นโพรเซสที่ทำงานแบบภาวะพร้อมกัน และมีการประสานจังหวะกัน (synchronize) ผ่านช่องสัญญาณ $\{A\} \cap \{B\}$

2) คำสั่งแบบ interleaving จะเขียนอธิบายอยู่ในรูปแบบของ G ||| H โดยที่ G และ H เป็นโพiเซสที่ทำงานแบบภาวะพร้อมกัน และแต่ละโพรเซสจะมีการทำงานที่เป็นอิสระต่อกันไม่มีการประสานจังหวะกัน

3) คำสั่งแบบ interface parallel จะเขียนอธิบายอยู่ในรูปแบบของ $G \,{}_{\{A\}}\|\, H$ โดยที่ G และ H เป็นโพรเซสที่ทำงานแบบภาวะพร้อมกัน และมีการประสานจังหวะกันผ่านช่องสัญญาณ {A}

การทวนสอบซอฟต์แวร์ด้วยเครื่องมือเอฟดีอาร์

เครื่องมือที่นำมาใช้ในการทวนสอบซอฟต์แวร์ ได้แก่ เครื่องมือเอฟดีอาร์ ซึ่งจะทำการทวนสอบระหว่างโปรแกรม (implementation) กับคุณลักษณะของโปรแกรม (specification) ที่แสดงในรูปของโมเดลเชิงรูปนัยที่ถูกเขียนด้วยซีเอสพี ซึ่งจะแสดงการทำงานของโปรแกรมและคุณลักษณะของโปรแกรมในรูปของโพรเซส

การทวนสอบความถูกต้องระหว่างโปรแกรมและคุณลักษณะของโปรแกรมด้วยการพิจารณาความสอดคล้องกัน (consistency) ระหว่าง trace ซึ่งใช้เป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของโปรแกรมและคุณลักษณะของโปรแกรม

กำหนดให้

P_1 เป็นโพรเซสของคุณลักษณะของโปรแกรม

P_2 เป็นโพรเซสของโปรแกรม

trace(P_1) เป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของคุณลักษณะของโปรแกรม

trace(P_2) เป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของโปรแกรม

ถ้า trace(P_1) มีความสัมพันธ์แบบ trace equivalent กับ trace(P_2) แสดงว่าพฤติกรรมที่เป็นไปได้ทั้งหมดของคุณลักษณะของโปรแกรมและโปรแกรมสอดคล้องกัน สามารถเขียนแทนความสัมพันธ์ได้ดังนี้คือ

$$P_1 =_T P_2 = \text{trace}(P_1) = \text{trace}(P_2) \qquad (2.1)$$

เครื่องมือเอฟดีอาร์เป็นซอฟต์แวร์สำหรับทวนสอบความสอดคล้องกันระหว่างคุณลักษณะของโปรแกรมและโปรแกรมอย่างอัตโนมัติ การทวนสอบจะทำด้วยการตรวจสอบความสัมพันธ์การแบ่งละเอียด (Refinement checking) ระหว่างโพรเซสที่เป็นตัวแทนของคุณลักษณะของโปรแกรมและโปรแกรม จากสมการ (2.1) โดยปกติแล้วเครื่องมือเอฟดีอาร์จะทำการตรวจสอบว่าโพรเซส P_2 ทำงานเป็นไปตามคุณลักษณะที่ได้กำหนดไว้ในโพรเซส P_1 หรือไม่ ถ้าโพรเซส P_2 ทำงานเป็นไปตามคุณลักษณะที่ได้กำหนดไว้ในโพรเซส P_1 แสดงว่า trace (P_2) $\subseteq$ trace (P_1) หมายถึงโปรแกรมทำงานถูกต้องตรงกับคุณลักษณะของโปรแกรมที่ได้กำหนดไว้

กำหนดให้

SPEC เป็นโพรเซสของคุณลักษณะของโปรแกรม

IMPL เป็นโพรเซสของโปรแกรม

trace(SPEC) เป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของคุณลักษณะของโปรแกรม

trace(IMP) เป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของโปรแกรม

ถ้าโปรแกรมทำงานถูกต้องตรงกับคุณลักษณะของโปรแกรม แสดงว่า โปรแกรมเป็นการแบ่งละเอียด(Refinement) ของคุณลักษณะของโปรแกรม สามารถเขียนแทนความสัมพันธ์ได้ดังนี้

$$\text{traces}(IMP) \subseteq \text{traces}(SPEC) \Leftrightarrow SPEC =_T SPEC \prod IMP \qquad (2.2)$$

เครื่องมือเอฟดีอาร์จะทำการตรวจสอบสมการ (2.2) ด้วยการอ่านโมเดลเชิงรูปนัยของคุณลักษณะของโปรแกรมและโปรแกรมที่เขียนด้วยซีเอสพีเข้ามาเป็นอินพุท หลังจากนั้นเครื่องมือเอฟดีอาร์จะสร้างปริภูมิสถานะ (state space) ซึ่งเป็นตัวแทนพฤติกรรมที่เป็นไปได้ทั้งหมดของคุณลักษณะของโปรแกรมและโปรแกรม แล้วนำปริภูมิสถานะของคุณลักษณะของโปรแกรมและโปรแกรมมาท่อง (traverse) ไปในทุกสถานะ (state) เพื่อทวนสอบความสอดคล้องกันของทั้งสองปริภูมิสถานะ หากมีความสอดคล้องกันก็จะแสดงว่า โปรแกรมทำงานได้ถูกต้องตรงตามคุณลักษณะของโปรแกรมที่กำหนดไว้ เครื่องมือเอฟดีอาร์สามารถทวนสอบคุณสมบัติได้หลากหลาย แต่ในงานวิจัยนี้จะนำเสนอการทวนสอบคุณสมบัติความปลอดภัยของซอฟต์แวร์เท่านั้น คุณสมบัติความปลอดภัยหรือการไม่มีสิ่งไม่ดีเกิดขึ้น (nothing bad happens.) ในที่นี้สิ่งไม่ดีหมายถึง การมีโพรเซสมากกว่าหนึ่งโพรเซสเข้าใช้ทรัพยากรร่วมของระบบแบบพร้อมกันในเวลาเดียวกัน เหตุการณ์ดังกล่าวก็จะเป็นหนึ่งในเงื่อนไขของการเกิดการติดตาย (Deadlock) ซึ่งเครื่องมือเอฟดีอาร์นั้นสามารถทวนสอบเหตุการณ์ติดตายได้ หรืออีกนัยหนึ่งเราสามารถใช้เครื่องมือเอฟดีอาร์ทวนสอบคุณสมบัติความปลอดภัยได้

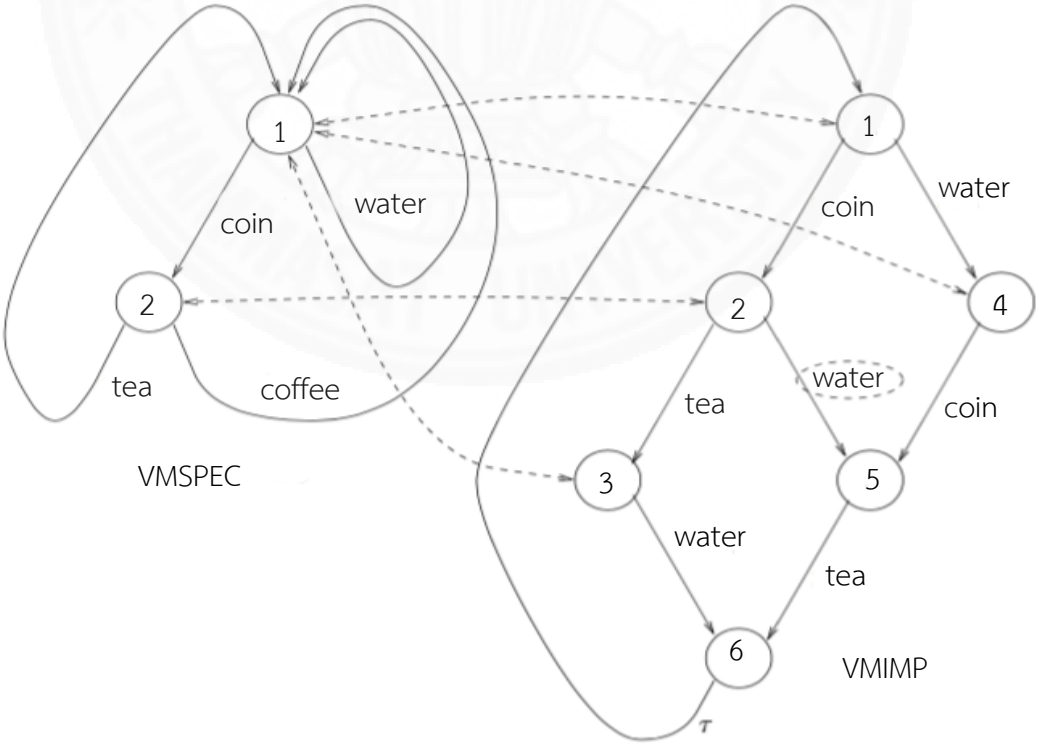
ในที่นี้จะขอนำเสนอตัวอย่างของ เครื่องจำหน่ายสินค้าแบบหยอดเหรียญ (vending machine) ซึ่งมีการกำหนดคุณลักษณะในรูปของโพรเซส VMSPEC และโปรแกรมของเครื่องจำหน่ายสินค้าแบบหยอดเหรียญนั้นแสดงในรูปของโพรเซส VMIMP (Schneider S., 1999) ตามลำดับดังนี้

VMSPEC = coin ((tea → VMSPEC) ⊓ (coffee → VMSPEC)) ⊓ water →VMSPEC

VMIMP = (coin → tea → SKIP ||| water → SKIP); VMIMP

ทั้งคุณลักษณะที่คาดหวังและโปรแกรมของเครื่องจำหน่ายสินค้าแบบหยอดเหรียญที่แสดงด้วยโพรเซส VMSPEC และ VMIMP นั้นจะถูกนำมาสร้างปริภูมิสถานะ เพื่อทวนสอบความสอดคล้องกันระหว่างโปรแกรมของเครื่องจำหน่ายสินค้าแบบหยอดเหรียญกับคุณลักษณะที่กำหนด แสดงดังภาพที่ 2.7

ภาพที่ 2.7 แสดงการทวนสอบความสอดคล้องกันระหว่างโปรแกรมกับคุณลักษณะของโปรแกรมของเครื่องจำหน่ายสินค้าแบบหยอดเหรียญ ในตัวอย่างนี้จะเห็นได้ว่า สถานะหมายเลข 2 ใน ปริภูมิสถานะของ VMSPEC มีทรานสิชันที่เป็นไปได้ 2 อันได้แก่ tea และ coffee ซึ่งไม่สอดคล้องกับสถานะหมายเลข 2 ในปริภูมิสถานะของ VMIMP ที่มีทรานสิชันที่เป็นไปได้ 2 อันได้แก่ tea และ water ดังนั้นโปรแกรมเครื่องจำหน่ายสินค้าแบบหยอดเหรียญนี้ทำงานไม่สอดคล้องกันกับคุณลักษณะของโปรแกรมที่กำหนดไว้



ภาพที่ 2.7 การทวนสอบความสอดคล้องกันระหว่างโปรแกรมกับคุณลักษณะของโปรแกรมของเครื่องจำหน่ายสินค้าแบบหยอดเหรียญ (Schneider S., 1999)

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับการทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพีนั้นได้แก่ งานวิจัยต่างๆที่ได้นำเอาซีเอสพีไปประยุกต์ใช้ในการอธิบายหรือการทวนสอบภาวะพร้อมกันในลักษณะงานต่างๆ ซึ่งแสดงได้ดังนี้

งานวิจัย (Michael Möller, 2002) นำเสนอวิธีการใช้ซีเอสพีเพื่อทวนสอบความถูกต้องของโปรแกรมภาษาจาวาที่ทำงานในระบบแบบกระจาย (Distributed system)

งานวิจัย (Murat Moran, James Heather and Steve Schneider, 2014) นำเสนอการทวนสอบระบบการลงคะแนนเสียงเชิงอิเล็คทรอนิกส์ด้วยเครื่องมือเอฟดีอาร์ โดยใช้ซีเอสพีอธิบายระบบและคุณลักษณะของระบบ

งานวิจัย (Gordon Thomas Rohrmair, 2005) นำเสนอแบบจำลองการวิเคราะห์การจู่โจมความปลอดภัยของโปรแกรมโดยใช้ซีเอสพีในการสร้างแบบจำลองความปลอดภัยและทำการวิเคราะห์อย่างละเอียดทุกการโจมตีที่เป็นไปได้ตามแบบจำลอง เพื่อนำไปสู่การหาเทคนิคในการป้องกันการโจมตี

งานวิจัย (Liyi Li, Elsa Gunter, and William Mansky, 2014) นำเสนอเครื่องมือในการตรวจสอบ trace ที่สร้างขึ้นตามพื้นฐานของซีเอสพี เพื่อทำให้สามารถจัดการระบบที่มีกลุ่มการทำงานเชิงไม่กำหนดได้ถูกต้องแม่นยำมากยิ่งขึ้น

งานวิจัย (A.W. Roscoe and Z. Wu, 2006) นำเสนอการทวนสอบแผนภูมิสถานะ (statechart) ด้วยการใช้เครื่องมือเอฟดีอาร์ โดยจะทำการแปลงแผนภูมิสถานะให้อยู่ในรูปของซีเอสพีก่อนแล้วจึงนำไปทวนสอบ

งานวิจัย (Mantas, J. M., & Palma, A, 1997) นำเสนอการออกแบบการลดทอนและการนำกลับมาใช้ใหม่ของส่วนประกอบของซอฟต์แวร์ในระบบแบบกระจาย ด้วยการสร้างโมเดลของซอฟต์แวร์ในรูปแบบซีเอสพี เพื่อนำโมเดลที่ได้ไปพัฒนาด้วยภาษาโปรแกรมได้ง่าย

งานวิจัย (Wang, X., Kwiatkowska, M., Theodoropoulos, G., & Zhang, Q., 2005) นำเสนอการทวนสอบการประสานจังหวะของฮาร์ดแวร์เพื่อทวนสอบคุณสมบัติต่างๆเช่นการติดตายและความหน่วงเวลาในการทำงานของส่วนประกอบต่างๆในฮาร์ดแวร์ด้วยการสร้างโมเดลเชิงรูปนัยด้วยซีเอสพี

จากงานวิจัยที่ได้กล่าวมาทั้งหมดสามารถเขียนสรุปเป็นตารางเปรียบเทียบคุณลักษณะของงานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ ได้แก่ ภาษาที่ใช้พัฒนซอฟต์แวร์ โหมดการทำงานของซอฟต์แวร์ โมเดลเชิงรูปนับของซอฟต์แวร์ และเครื่องมือทวนสอบซอฟต์แวร์ ดังแสดงภาพที่ 2.8

จากงานวิจัยที่ได้กล่าวมาทั้งหมดจะเห็นได้ว่าซีเอสพีถูกนำไปใช้สำหรับสร้างโมเดลเชิงรูปนัยเพื่อทวนสอบความถูกต้องของการทำงานในระบบต่างๆ แต่ยังไม่มีงานวิจัยใดที่นำซีเอสพีมาสร้างโมเดลเชิงรูปนัยสำหรับภาษาโก ดังนั้นงานวิจัยนี้จะได้นำซีเอสพีมาสร้างโมเดลเชิงรูปนัยสำหรับภาษาโกเพื่อนำไปใช้ทวนสอบความถูกต้องของโปรแกรมที่ถูกพัฒนาด้วยภาษาโกโดยใช้เครื่องมือเอฟดีอาร์เป็นเครื่องมือทวนสอบ

งานวิจัย	ภาษาที่ใช้พัฒนาซอฟต์แวร์	โหมดการทำงานของซอฟต์แวร์	โมเดลเชิงรูปนัยของซอฟต์แวร์	เครื่องมือทวนสอบซอฟต์แวร์
Michael Möller, 2002)	ภาษาจาวา	ระบบแบบกระจาย	ซีเอสพี	jassda
Murat Moran, James Heather and Steve Schneider, 2014	ไม่ระบุ	ระบบแบบเดี่ยว (Stand-alone system)	ซีเอสพี	เอฟดีอาร์
Gordon Thomas Rohrmair, 2005	ไม่ระบุ	ระบบแบบเดี่ยว	ซีเอสพี	เอฟดีอาร์
Liyi Li, Elsa Gunter, and William Mansky, 2014	Ocaml	ระบบแบบภาวะพร้อมกัน	ซีเอสพี	manual
A.W. Roscoe and Z. Wu, 2006	ไม่ระบุ	ระบบแบบภาวะพร้อมกัน	ซีเอสพี	เอฟดีอาร์
Mantas, J. M., & Palma, A, 1997	ไม่ระบุ	ระบบแบบกระจาย	ซีเอสพี	ไม่ทวนสอบ
Wang, X., Kwiatkowska, M., Theodoropoulos, G., & Zhang, Q., 2005	Stateful Timed CSP	ระบบแบบทันที (Real-time system)	ซีเอสพี	PAT
งานวิจัยที่นำเสนอ	ภาษาโก	ระบบแบบภาวะพร้อมกัน	ซีเอสพี	เอฟดีอาร์

ภาพที่ 2.8 ตารางเปรียบเทียบคุณลักษณะของงานวิจัยที่เกี่ยวข้อง

บทที่ 3
วิธีการวิจัย

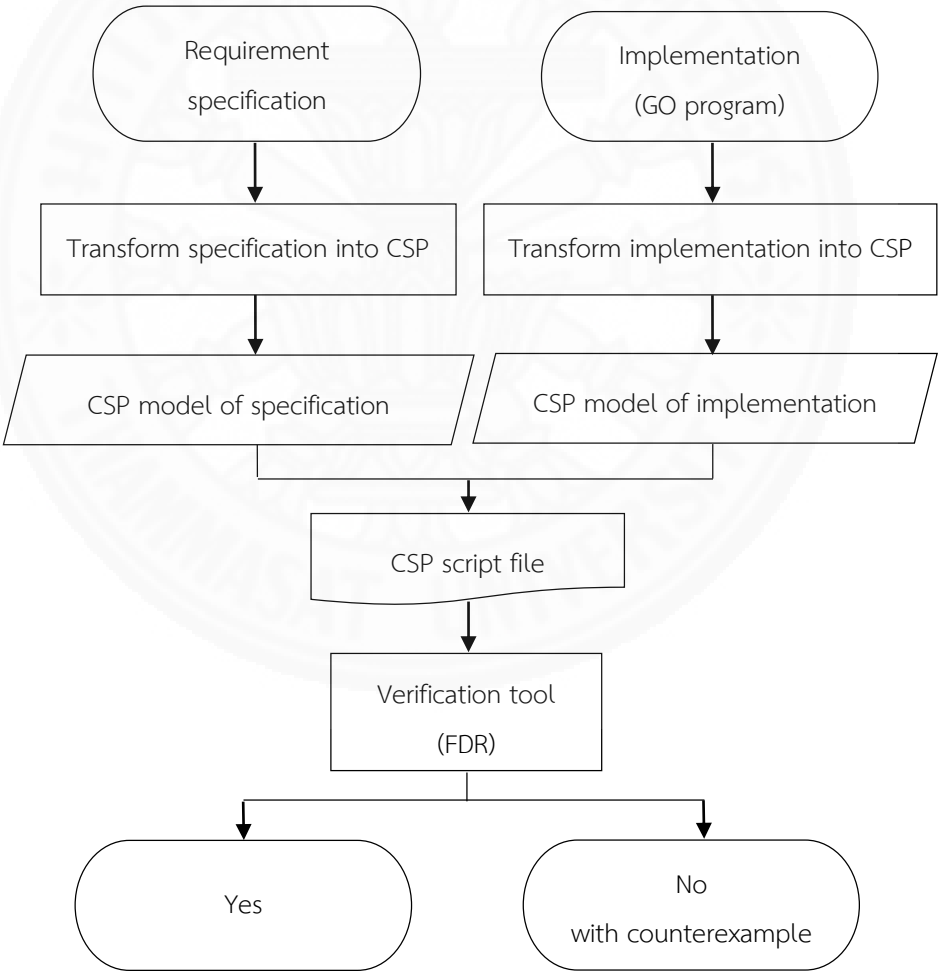
การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี ในบทนี้แสดงให้เห็นภาพรวมของการทวนสอบเพื่อยืนยันความถูกต้องของโปรแกรมที่ถูกพัฒนาด้วยภาษาโกและตัวอย่างของการทวนสอบภาษาโก

การทวนสอบเชิงรูปนัยภาษาโกด้วยซีเอสพี

ภาพรวมของการทวนสอบเชิงรูปนัยของโปรแกรมที่ถูกพัฒนาด้วยภาษาโกด้วยซีเอสพี แสดงดังภาพที่ 3.1 โดยสามารถแบ่งออกเป็น 2 ขั้นตอน ได้แก่

1) การสร้างโมเดลเชิงรูปนัยจากรหัสต้นฉบับภาษาโกให้อยู่ในรูปของซีเอสพี

2) การทวนสอบด้วยเครื่องมือเอฟดีอาร์



ภาพที่ 3.1 ภาพรวมการทวนสอบเชิงรูปนัยโปรแกรมภาษาโกด้วยซีเอสพี

3.1 การสร้างโมเดลเชิงรูปนัยจากรหัสต้นฉบับภาษาโกให้อยู่ในรูปของซีเอสพี

การแปลงรหัสต้นฉบับโปรแกรมที่ถูกพัฒนาด้วยภาษาโก ให้เป็นโมเดลเชิงรูปนัยในรูปแบบของซีเอสพีนั้นจะถูกแปลงด้วย Algorithm 1 ซึ่งรายละเอียดของอัลกอริทึมแสดงดังภาพที่ 3.2

Algorithm 1 Transform Go program into CSP model.

```
บรรทัด  Input:   a Go program
        Output: a CSP script
1:      Generate an abstract syntax tree from an input
2:      Initial a CSP script as an empty file.
3:      for each node n in the abstract syntax tree do
4:              if a node N is a variable then
5:                      CSP = CreateCSPModel (n,value)
6:              else if a node N is a function then
7:                      CSP = CreateCSPModel(n,null)
8:              endif
9:                      Append CSP to a CSP script.
10:     endfor
11:     return a CSP script
12:     CreateCSPModel (n, value)
13:     {
14:             Map each node n to CSP language.
15:             if value is not null then
16:             assign value to a variable
17:     return a CSP model
18:     }
```

ภาพที่ 3.2 อัลกอริทึมสำหรับแปลงรหัสต้นฉบับโปรแกรมภาษาโกให้อยู่ในรูปของซีเอสพี

รายละเอียดการทำงานของ Algorithm 1 ในภาพที่ 3.2 สามารถอธิบายได้ดังนี้

บรรทัดที่ 1:

อัลกอริทึมเริ่มการทำงานด้วยการอ่านไฟล์ภาษาโกเข้ามาให้อยู่ในรูปของโครงสร้างข้อมูล AST (Abstract syntax tree) ซึ่งเป็นโครงสร้างข้อมูลที่ใช้อธิบายไวยากรณ์ของรหัสต้นฉบับที่ถูกเขียนด้วยภาษาโก โดยที่แต่ละโหนด (node) ของโครงสร้างข้อมูล AST จะแทนคำสั่งที่เกิดขึ้นในรหัสต้นฉบับ โครงสร้างข้อมูล AST จะไม่แสดงทุกรายละเอียดที่มีอยู่ในรหัสต้นฉบับจริง ตัวอย่างเช่น เครื่องหมายปีกกา “{}“ จะไม่แสดงอยู่ในรายละเอียดของโหนด โครงสร้างข้อมูล AST ของภาษาโกจะถูกสร้างด้วยการเรียกใช้ฟังก์ชัน ParseFile ที่จัดสรรไว้ให้แล้วในตัวภาษาโก

บรรทัดที่ 2:

อัลกอริทึมจะสร้างไฟล์เริ่มต้นสำหรับบันทึกโมเดลเชิงรูปนัยของโปรแกรมในรูปแบบของสคริปต์ไฟล์

บรรทัดที่ 3-11:

อัลกอริทึมจะท่องในโครงสร้างข้อมูล AST แล้วทำการแปลงคำสั่งที่เจอในโครงสร้างข้อมูล AST ให้เป็นคำสั่งของภาษาซีเอสพี ด้วยการเรียกใช้ฟังก์ชัน CreateCSPModel ในบรรทัดที่ 12-18 ซึ่งมีการทำงานตามขั้นตอนดังนี้

1) แปลงฟังก์ชันของโปรแกรมภาษาโกให้เป็นโพรเซสในซีเอสพี
2) กำหนดตัวแปรที่ถูกประกาศในโปรแกรมภาษาโกให้เป็นเหตุการณ์ (event) ในซีเอสพี
3) แปลงคำสั่งในฟังก์ชันที่ได้นิยามไว้ในโปรแกรมภาษาโกให้เป็นคำสั่งซีเอสพี ซึ่งสามารถอ้างอิงการเทียบชุดคำสั่งของภาษาโกและซีเอสพีได้ดังภาพที่ 3.3

Go	CSP

if-else statement:
if x < y {
return x
} else {
return y
}
	[x < y -> x ∏ x>=y -> y]

function statement: func COPY{}	COPY::[]
interleaving: func read(){} func write(){} go read() go write()	read ||| write
assignment statement: var x = 1	x := integer; x:=1;

for statement:
for i:=0; i<10; i++ {
i++
}
	*[i:=0;i<10 -> i:=i+1; i:=i+1]

ภาพที่ 3.3 การเทียบชุดคำสั่งของภาษาโกและซีเอสพี

3.2 การทวนสอบด้วยเครื่องมือเอฟดีอาร์

ขั้นตอนนี้เป็นการทวนสอบความถูกต้องโปรแกรมภาษาโก โดยจะนำโมเดลเชิงรูปนัยของโปรแกรมและคุณลักษณะของโปรแกรมที่ถูกบันทึกไว้ในซีเอสพีสคริปต์ไฟล์ ซึ่งเป็นผลลัพธ์ที่ได้จากขั้นตอนก่อนหน้านี้มาเป็นอินพุทของเครื่องมือทวนสอบเอฟดีอาร์ เครื่องมือเอฟดีอาร์จะสร้างปริภูมิสถานะของโมเดลเชิงรูปนัยเพื่อตรวจสอบความสัมพันธ์การแบ่งละเอียดของซอฟต์แวร์

ในที่นี้จะขอแสดงตัวอย่างการทวนสอบโปรแกรมตามวิธีการที่ได้นำเสนอไปแล้วข้างต้น ตัวอย่างเช่น การทวนสอบโปรแกรมคำนวณหาค่าเลขยกกำลังสอง คุณลักษณะของโปรแกรมได้

กำหนดไว้ดังภาพที่ 3.4 assert CON_SQUARE [T = STOP หมายถึง โพรเซส CON_SQUARE ซึ่งเป็นโมเดลเชิงรูปนัยแสดงการทำงานของโปรแกรมคำนวณหาค่าเลขยกกำลังสองที่ถูกเขียนด้วยภาษาโกดังภาพที่ 3.5ในที่สุดแล้วจะต้องหยุด (termination) และให้ผลลัพธ์ออกมา

```
assert CON_SQUARE [T= STOP
```

ภาพที่ 3.4 คุณลักษณะของโปรแกรมคำนวณหาค่าเลขยกกำลังสอง

```
func SQUARE(x int) int {
        return Square(x)
}
func Square(x int) int {
        return x * x
}
func CON_SQUARE() {
        go SQUARE(3)
        go SQUARE(5)
}
```

ภาพที่ 3.5 โปรแกรมคำนวณหาค่าเลขยกกำลังสองที่ถูกเขียนด้วยภาษาโก

เริ่มต้นการทำงานโปรแกรมในภาพที่ 3.5 จะถูกแปลงเป็นโมเดลเชิงรูปนัยด้วย Algorithm 1 ในขั้นตอนแรกอัลกอริทึมจะต้องสร้างโครงสร้างข้อมูล AST ที่สอดคล้องกับโปรแกรม ตัวอย่างโครงสร้างข้อมูล AST ของฟังก์ชัน SQUARE แสดงดังภาพที่ 3.6

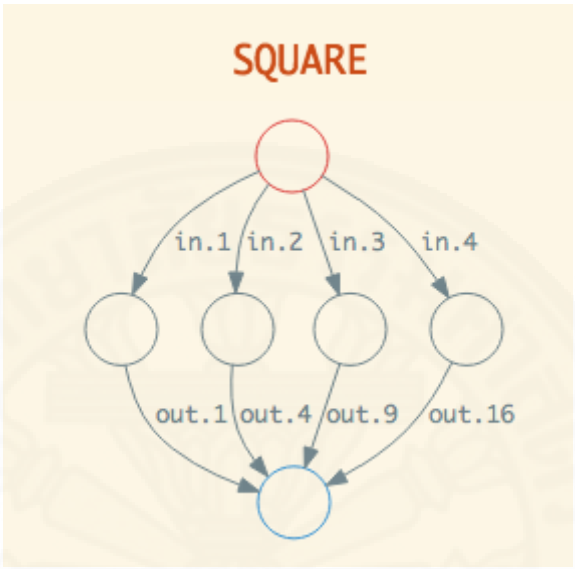
ภาพที่ 3.6 AST ของฟังก์ชัน SQUARE

หลังจากนั้นโครงสร้างข้อมูล AST จะถูกท่องเพื่อสร้างเป็นโมเดลเชิงรูปนัยที่อยู่ในรูปแบบของซีเอสพี แสดงดังภาพที่ 3.7

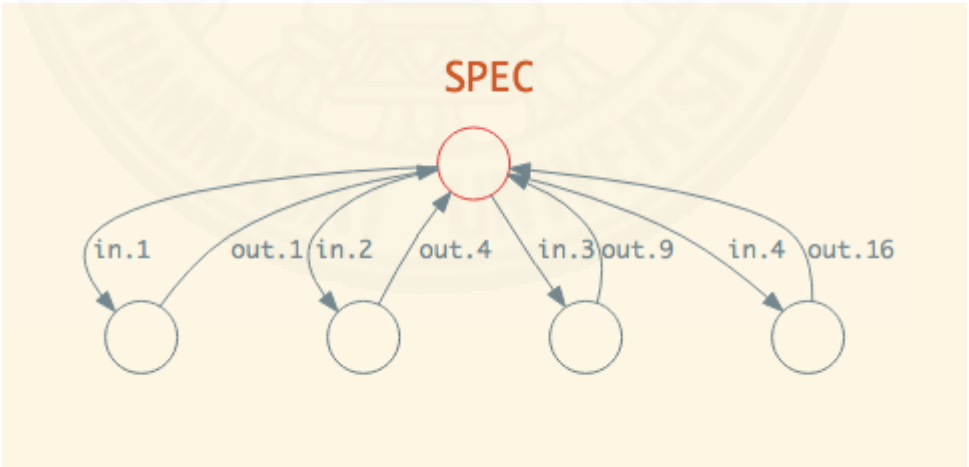
```
channel in : {1..4}
channel out : {x * x | x <- {1..4}}

Square(y) = y * y
SQUARE = in?x -> out!Square(x) -> STOP
CON_SQUARE = SQUARE ||| SQUARE
```

ภาพที่ 3.7 โมเดลเชิงรูปนัยที่อยู่ในรูปของซีเอสพี

โมเดลเชิงรูปนัยที่อยู่ในรูปของซีเอสพีในภาพที่ 3.7 จะถูกอ่านเป็นอินพุทเข้าสู่เครื่องมือเอฟดีอาร์(University of Oxford, 2015) พร้อมกับคุณลักษณะที่คาดหวัง แสดงดังภาพที่ 3.4 เครื่องมือเอฟดีอาร์จะทำการสร้างปริภูมิสถานะของโปรแกรมคำนวณหาค่าเลขยกกำลังสองและปริภูมิสถานะของคุณลักษณะของโปรแกรม แสดงดังภาพที่ 3.8 และภาพที่ 3.9ตามลำดับ

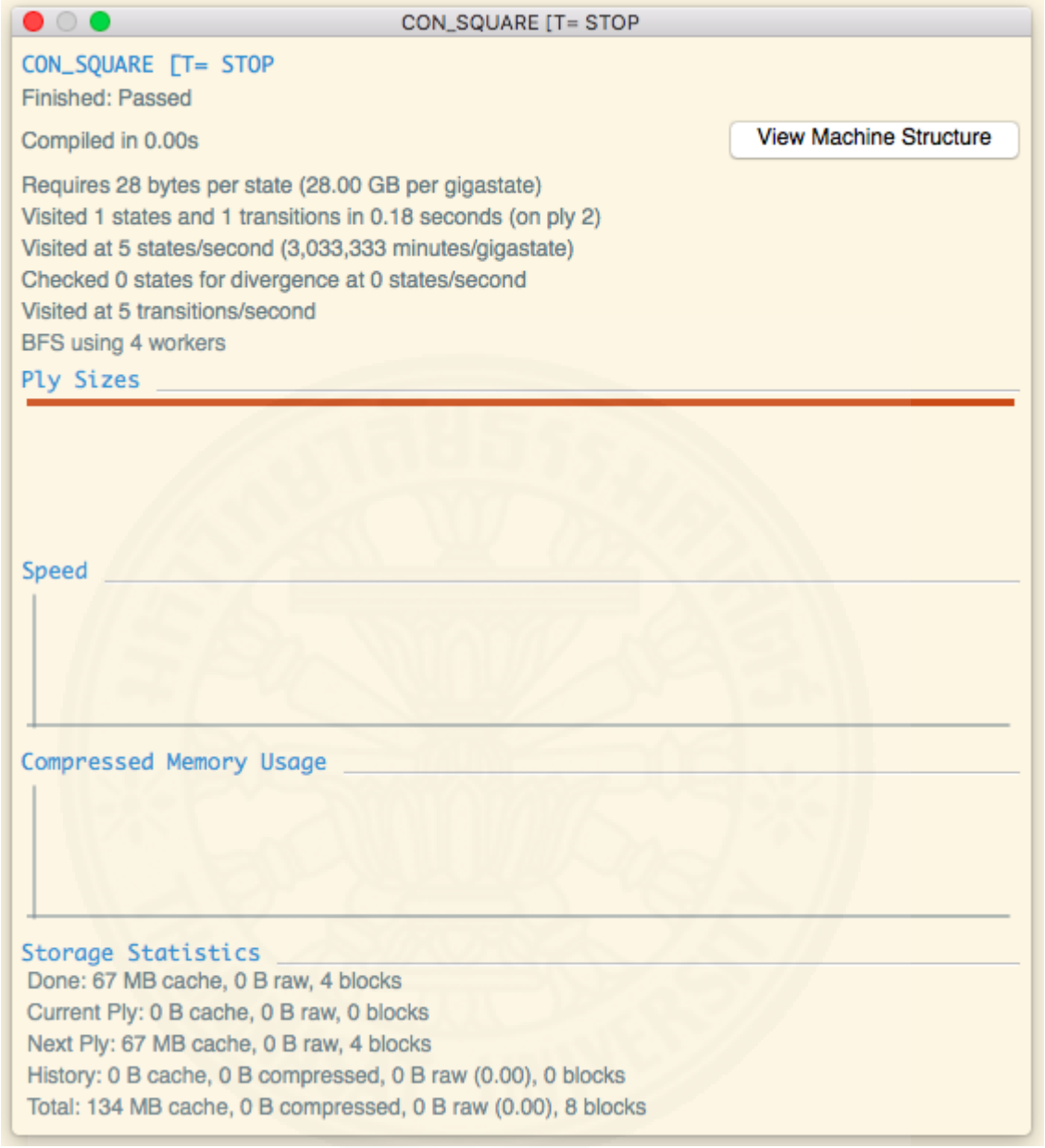


ภาพที่ 3.8 ปริภูมิสถานะของโปรแกรมคำนวณหาค่าเลขยกกำลังสอง



ภาพที่ 3.9 ปริภูมิสถานะคุณลักษณะของโปรแกรมคำนวณหาค่าเลขยกกำลังสอง

เมื่อทำการประมวลผลเพื่อทวนสอบความสอดคล้องกันของปริภูมิสถานะทั้งสองด้วยเครื่องมือเอฟดีอาร์ ผลลัพธ์ที่ได้แสดงให้เห็นว่าโปรแกรมคำนวณเลขยกกำลังสองนั้นมีความสอดคล้องกับคุณลักษณะที่คาดหวังไว้ในทุกกรณี ผลลัพธ์ของการประมวลผลแสดงดังภาพที่ 3.10



ภาพที่ 3.10 ผลลัพธ์การทวนสอบความถูกต้องในการทำงานแบบภาวะพร้อมกัน

บทที่ 4
ผลการวิจัยและอภิปรายผล

ในบทนี้จะกล่าวถึงผลการวิจัยที่ได้ดำเนินการตามวิธีการวิจัยที่ได้นำเสนอในบทที่ 3 กับโปรแกรมการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกันและนำเสนอการทวนสอบกับกรณีศึกษาซึ่งเป็นตัวอย่างที่ซับซ้อนขึ้นทั้งหมด 2 กรณีศึกษาได้แก่ ปัญหาอาหารเย็นของนักปรัชญา (Dining Philosopher's Problem) และ ปัญหาที่พักข้อมูลขนาดจำกัด (Bounded Buffer Problem) (Hoare, C. A. R., 1978) และในบทนี้จะแสดงให้เห็นผลลัพธ์ที่ได้จากการทวนสอบกับกรณีศึกษาแต่ละอันด้วย

4.1.กรณีศึกษาการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกัน

กรณีศึกษานี้จะทำการแสดงให้เห็นถึงการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกัน เราจะทำการทวนสอบว่าโปรแกรมมีความสอดคล้องกับคุณลักษณะของโปรแกรมที่คาดหวังไว้หรือไม่

4.1.1. คุณลักษณะของซอฟต์แวร์ที่ต้องการทวนสอบ

คุณลักษณะที่จะใช้สำหรับทวนสอบซอฟต์แวร์ได้แก่ การคัดลอกข้อมูลผ่านช่องสัญญาณสามารถทำงานแบบภาวะพร้อมกันได้อย่างถูกต้อง ซึ่งคุณลักษณะของซอฟต์แวร์ที่ต้องการทวนสอบนี้แสดงในรูปของซีเอสพีด้วยโพรเซส COPY_SPEC แสดงดังภาพที่ 4.1 โพรเซส COPY_SPEC จะประกอบด้วยช่องสัญญาณสองช่องที่คัดลอกข้อมูลแบบภาวะพร้อมกันอย่างอิสระ

```
COPY_SPEC = in?c -> out!c -> STOP ||| in?c -> out!c -> STOP
assert COPY_SPEC [T= COPY_IMPL
```

ภาพที่ 4.1 คุณลักษณะของซอฟต์แวร์การคัดลอกข้อมูลแบบภาวะพร้อมกัน

4.1.2. ชุดคำสั่งของซอฟต์แวร์ที่ใช้ในการทวนสอบ

โปรแกรมสำหรับการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกันแสดงดังภาพที่ 4.2 โปรแกรมถูกพัฒนาด้วยภาษาโกมีการเรียกใช้ชุดคำสั่ง go (The Go Authors, 2011) เพื่อให้โปรแกรมสามารถทำการคัดลอกข้อมูลแบบภาวะพร้อมกันได้ โมเดลเชิงรูปนัยของโปรแกรมสำหรับการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกันที่เขียนอยู่ในรูปของซีเอสพีแสดงดังภาพที่ 4.3

```
func Copy(in, out chan rune) {
    for r := range in {
```

```
                    out <- r
            }
            close(out)
    }
        func CopyImpl() {
            go Copy(in1, out1)
            go Copy(in2, out2)
    }
```

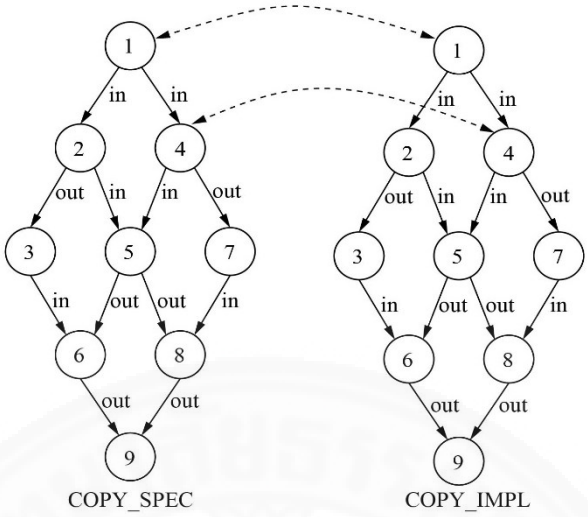
ภาพที่ 4.2 ชุดคำสั่งซอฟต์แวร์การคัดลอกข้อมูลแบบภาวะพร้อมกัน

```
COPY:: [c:integer;*[in?c -> out!c->STOP]]
 COPY_IMPL::COPY ||| COPY
```

ภาพที่ 4.3 โมเดลเชิงรูปนัยของซอฟต์แวร์การคัดลอกข้อมูลแบบภาวะพร้อมกัน

4.1.3. ผลการทวนสอบ

ในการทวนสอบโปรแกรมเครื่องมือเอฟดีอาร์จะทำการสร้างปริภูมิสถานะของโพรเซส COPY_SPEC และ COPY_IMPL ซึ่งเป็นโมเดลเชิงรูปนัยแทนคุณลักษณะโปรแกรมและโปรแกรมการคัดลอกข้อมูลผ่านช่องสัญญาณแบบภาวะพร้อมกัน แล้วทำการตรวสอบความสัมพันธ์การแบ่งละเอียด โดยการสร้างปริภูมิสถานะของโพรเซสทั้งสองและทำการตรวจสอบความสอดคล้องกันของสถานะทั้งหมดภายในปริภูมิสถานะ ดังแสดงในภาพที่ 4.4



ภาพที่ 4.4 ปริภูมิสถานะของโพรเซส COPY_SPEC และ COPY_IMPL

ผลลัพธ์ของการทดลองแสดงดังภาพที่ 4.5 และภาพที่ 4.6

ภาพที่ 4.5 ผลการทวนสอบโปรแกรมการคัดลอกข้อมูล

คุณสมบัติที่ทวนสอบ	ผลลัพธ์
COPY_SPEC [T= COPY_IMPL	Passed

ภาพที่ 4.6 ผลการทวนสอบโปรแกรมการคัดลอกข้อมูล

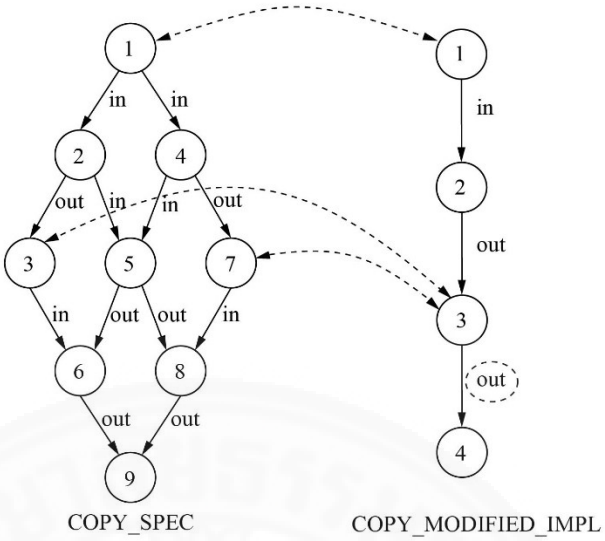
นอกจากนี้เราได้แก้ไขโปรแกรมในภาพที่ 4.7 เพื่อทำให้โปรแกรมการคัดลอกข้อมูลผ่านช่องสัญญาณไม่สามารถทำงานแบบภาวะพร้อมกันได้แสดงดังภาพที่ 4.8 ซึ่งจะไม่เป็นไปตามคุณลักษณะของโปรแกรมที่กำหนดไว้ ทั้งนี้เพื่อเป็นการตรวจสอบว่าเครื่องมือเอฟดีอาร์สามารถทวนสอบความไม่สอดคล้องกันระหว่างโปรแกรมกับคุณลักษณะของโปรแกรมได้ และผลการทวนสอบโปรแกรมดังกล่าวด้วยเครื่องมือเอฟดีอาร์พบว่าปริภูมิสถานะของโพรเซส COPY_SPEC และโพรเซส COPY_MODIFIED_IMPL ไม่สอดคล้องตรงกันดังแสดงในภาพที่ 4.9

```
func Copy(in, out chan rune) {
    for r := range in {
        out <- r
        out <- r
    }
    close(out)
}
func CopyModifiedImpl() {
    Copy(in, out)
}
```

ภาพที่ 4.7 ชุดคำสั่งซอฟต์แวร์การคัดลอกข้อมูลแบบภาวะไม่พร้อมกัน

```
COPY::*[c:integer;in?c -> out!c -> out!c->STOP]
COPY_MODIFIED_IMPL::COPY
```

ภาพที่ 4.8 โมเดลเชิงรูปนัยของซอฟต์แวร์การคัดลอกข้อมูลแบบภาวะไม่พร้อมกัน



ภาพที่ 4.9 ปริภูมิสถานะของโพรเซส COPY_SPEC และ COPY_MODIFIED_IMPL

4.2.กรณีศึกษาปัญหาอาหารเย็นของนักปรัชญา

ปัญหาอาหารเย็นของนักปรัชญาเป็นกรณีศึกษาที่กำหนดให้มีนักปราชญ์ 5 ท่านที่จะใช้ชีวิตในรูปแบบของการคิด และการกินโดยนักปราชญ์ทั้ง 5 ท่านนี้จะใช้โต๊ะรับประทานอาหารร่วมกันซึ่งโต๊ะรับประทานอาหารมีลักษณะเป็นวงกลมและมีเก้าอี้ 5 ตัวล้อมรอบโต๊ะ ซึ่งเก้าอี้แต่ละตัวจะเป็นตำแหน่งของนักปราชญ์แต่ละท่าน นอกจากนี้จะมีช้อนส้อมวางไว้บนโต๊ะเพื่อรับประทานอาหาร 5 อัน หากนักปราชญ์ต้องการรับประทานอาหารจะต้องหยิบช้อนส้อมทางด้านซ้ายมือและทางด้านขวามือของนักปราชญ์ หากนักปราชญ์ท่านใดไม่สามารถหยิบช้อนส้อมได้ 2 อันเพื่อที่จะรับประทานอาหารจะทำให้นักปราชญ์นั้นอดตาย (starvation) ในตัวอย่างนี้จะแสดงให้เห็นถึงการทวนสอบการติดตาย (deadlock) ของการหยิบช้อนส้อม

4.2.1.คุณลักษณะของซอฟต์แวร์ที่ต้องการทวนสอบ

คุณลักษณะที่จะใช้สำหรับทวนสอบซอฟต์แวร์ที่ถูกพัฒนาขึ้นตามปัญหาที่ได้กล่าวไปแล้วข้างต้นนั้น จะต้องสามารถทวนสอบคุณสมบัติเรื่องการติดตายได้ ซึ่งคุณลักษณะของซอฟต์แวร์แสดงดังภาพที่ 4.10

```
assert SYSTEM :[deadlock free [F]]
```

ภาพที่ 4.10 คุณลักษณะของซอฟต์แวร์

4.2.2.ชุดคำสั่งของซอฟต์แวร์ที่ใช้ในการทวนสอบ

ชุดคำสั่งที่ถูกเขียนด้วยภาษาโกสำหรับปัญหานี้ (The Go Authors, 2011) แสดงดังภาพที่ 4.11 และชุดคำสั่งนี้จะถูกแปลงให้อยู่ในรูปของโมเดลเชิงรูปนัยซีเอสพีด้วย Algorithm 1 ที่ได้นำเสนอไปแล้วในบทที่ 3 ผลลัพธ์ที่ได้จากการแปลงแสดงดังภาพที่ 4.12

```
func DiningPhilosophers(runFor time.Duration) {
        enterRoom := make(chan int)
        exitRoom := make(chan int)
        room := func() {
                occupancy := 0
                for {
                        select {
                        case i := <-enterRoom:
                                if occupancy < 4 {
                                        occupancy++
                                } else {
                                        // If all philosophers sit down to eat, they
starve.
                                        // Wait for someone to leave.
                                        fmt.Printf("%v wants to enter, but must
wait.\n", i)
                                        <-exitRoom
                                        // Enter the room, occupancy stays the
same.
                                        fmt.Printf("%v can finally enter!\n", i)
                                }
                        case <-exitRoom:
                                occupancy--
                        }
                }
```

```
}

// The forks are goroutines listening to pickup and putdown channels
// like the room, but we need one channel per philosopher to
// distinguish them so that we can match pickup and putdown of a fork.
pickup := make([]chan int, 5)
putdown := make([]chan int, 5)
for i := 0; i < 5; i++ {
        pickup[i] = make(chan int)
        putdown[i] = make(chan int)
}
fork := func(i int) {
        for {
                select {
                case <-pickup[i]:
                        <-putdown[i]
                case <-pickup[abs(i-1)%5]:
                        <-putdown[abs(i-1)%5]
                }
        }
}

// Thinking and eating are sleeps followed by a log message so we know
// what's going on.
think := func(i int) {
        fmt.Printf("%v thought.\n", i)
}
eat := func(i int) {s
        fmt.Printf("%v ate.\n", i)
}
```

```
        // A philosopher leads a simple life.
        philosopher := func(i int) {
                for {
                        think(i)
                        enterRoom <- i
                        pickup[i] <- i
                        pickup[(i+1)%5] <- i
                        eat(i)
                        putdown[i] <- i
                        putdown[(i+1)%5] <- i
                        exitRoom <- i
                }
        }

        // Launch the scenario.
        go room()
        for i := 0; i < 5; i++ {
                go fork(i)
        }
        for i := 0; i < 5; i++ {
                go philosopher(i)
        }

        time.Sleep(runFor)
}
```

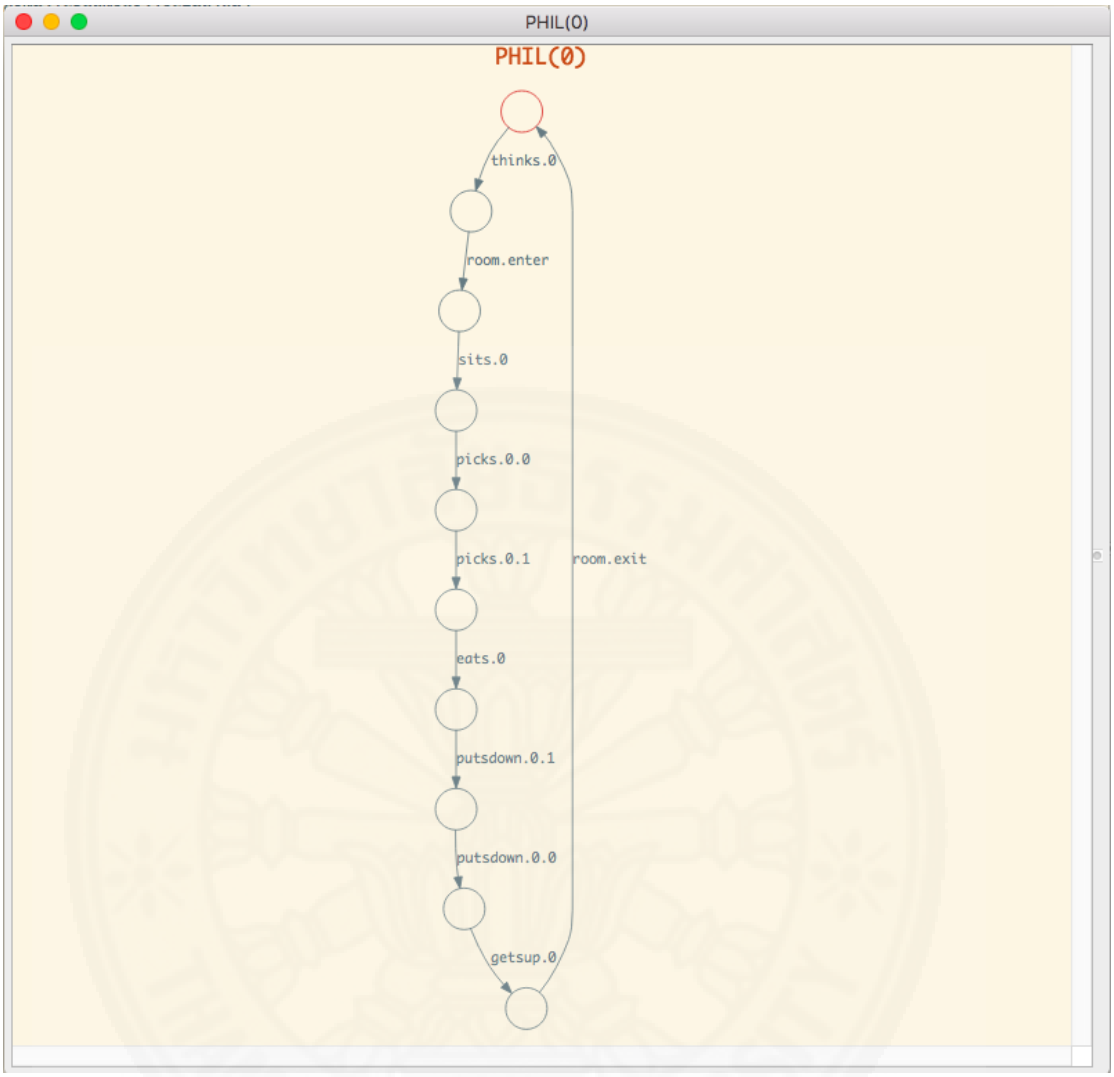
ภาพที่ 4.11 ชุดคำสั่งของซอฟต์แวร์ปัญหาอาหารเย็นของนักปราชญ์ในภาษาโก

```
PHIL:: *[THINK;
        room!enter();
        fork(i)!pickup();fork((i+1)%5)!pickup();
        EAT;
        fork(i)!putdown();fork((i+1)%5)!putdown();
        room!exit()]
FORK::[phil(i)?pickup()->phil(i)?putdown()
        []phil((i-1)%5)!pickup()->phil((i-1)%5)?putdown()]
ROOM::occupancy:interger; occupancy := 0;
        *[(i:0..4)phil(i)?enter()->occupancy:=occupancy+1
        [](i:0..4)phil(i)?exit()->occupancy:=occupancy-1]
// Launch the scenario
PHILS = ||| i:PHILNAMES@ PHIL(i)
FORKS = ||| i:FORKNAMES@ FORK(i)
```

ภาพที่ 4.12 โมเดลเชิงรูปนัยที่อยู่ในรูปของซีเอสพี

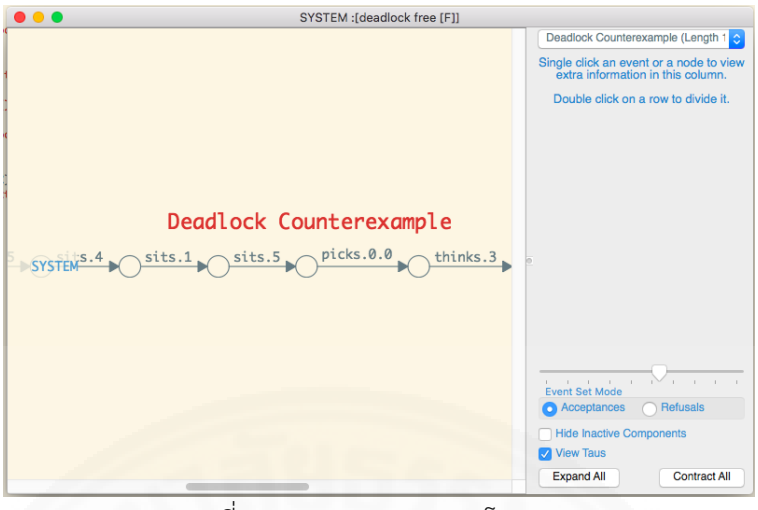
4.2.3.ผลการทวนสอบ

การทวนสอบซอฟต์แวร์นี้ไม่สามารถที่จะสร้างปริภูมิสถานะออกมาเป็นรูปภาพได้ทุกสถานะ เนื่องจากปริภูมิสถานะมีขนาดใหญ่มาก จึงได้นำตัวอย่างของปริภูมิสถานะของนักปราชญ์เพียง 1 ท่าน มาแสดงตัวอย่างการดำเนินชีวิตของนักปราชญ์แสดงดังภาพที่ 4.13

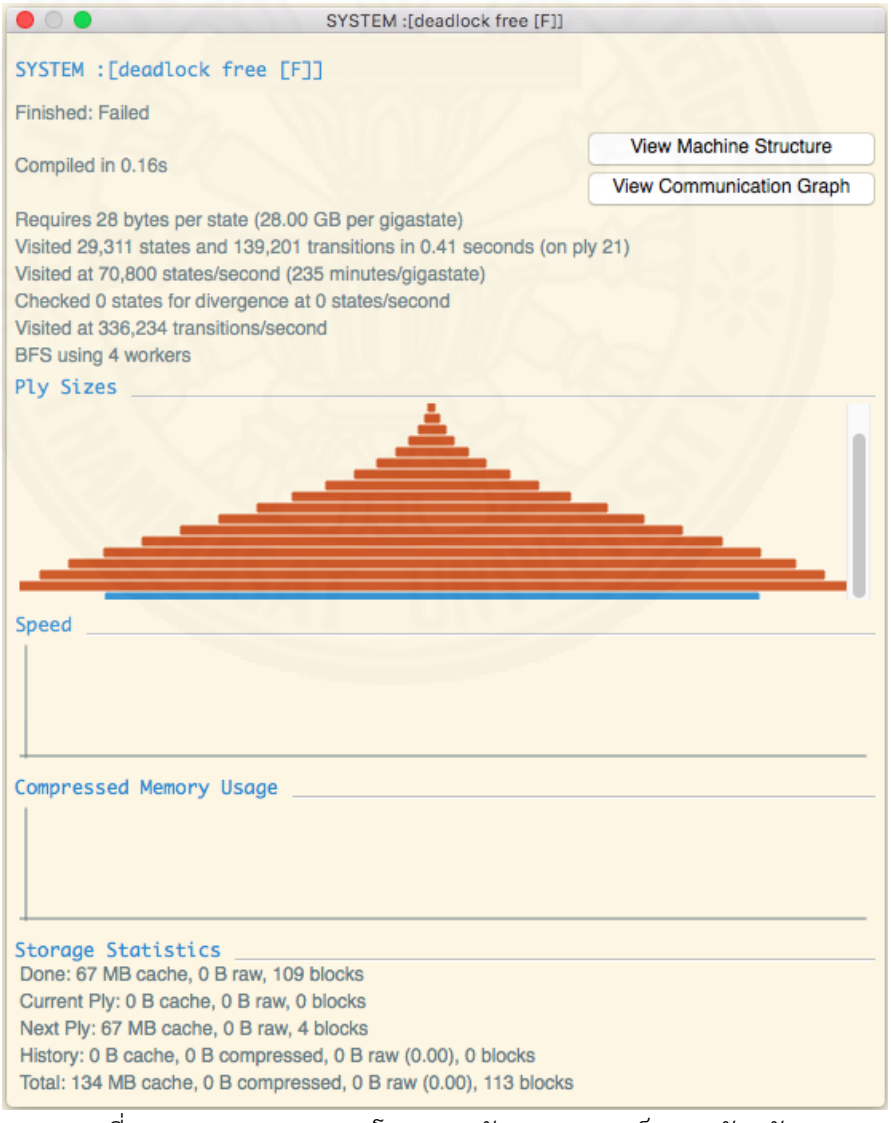


ภาพที่ 4.13 ตัวอย่างสถานะของนักปราชญ์

ผลลัพธ์ที่ได้แสดงให้เห็นว่ามีการหยุดชะงักตามที่เราได้คาดหวังตามคุณลักษณะของซอฟต์แวร์ข้างต้นทำให้เราสามารถทวนสอบหาข้อบกพร่องของซอฟต์แวร์ได้แสดงดังภาพที่ 4.14 และแสดงผลลัพธ์ของการทวนสอบดังภาพที่ 4.15 และ ภาพที่ 4.16



ภาพที่ 4.14 ผลการทวนสอบโปรแกรม



ภาพที่ 4.15 ผลการทวนสอบโปรแกรมปัญหาอาหารเย็นของนักปรัชญา

คุณสมบัติที่ทวนสอบ	ผลลัพธ์
SYSTEM :[deadlock free [F]]	Failed

ภาพที่ 4.16 ตารางผลการทวนสอบโปรแกรมปัญหาอาหารเย็นของนักปรัชญา

4.3.กรณีศึกษา Bounded Buffer Problem

ปัญหาที่พักข้อมูลขนาดจำกัดเป็นกรณีศึกษาเกี่ยวกับการใช้ทรัพยากรร่วมกันระหว่างสองโปรเซสในการเก็บข้อมูลที่มีขนาดจำกัด โดยจะมีโพรเซสที่ทำหน้าที่สร้างข้อมูลเรียกว่าผู้ผลิต (Producer) เพื่อนำไปใส่ไนบัพเฟอร์ (Buffer) ที่มีขนาดจำกัดและโพรเซสที่ทำหน้าที่นำข้อมูลออกเรียกว่าผู้บริโภค (Consumer) จากบัพเฟอร์ โดยทั้งสองโพรเซสต้องทำงานสอดคล้องกันเพื่อทำให้ไม่เกิดการติดตาย

4.3.1. คุณลักษณะของซอฟต์แวร์ที่ต้องการทวนสอบ

การทวนสอบปัญหาที่พักข้อมูลขนาดจำกัดมีคุณลักษณะที่คาดหวังในการทวนสอบคือต้องไม่มีการติดตายเกิดขึ้น แสดงดังภาพที่ 4.17

```
assert BUFFER(<>) :[deadlock free [F]]
```

ภาพที่ 4.17 คุณลักษณะของซอฟต์แวร์

4.3.2. ชุดคำสั่งของซอฟต์แวร์ที่ใช้ในการทวนสอบ

ชุดคำสั่งที่ถูกเขียนด้วยภาษาโกสำหรับปัญหาปัญหาที่พักข้อมูลขนาดจำกัด (The Go Authors, 2011) แสดงดังภาพที่ 4.18 และชุดคำสั่งนี้จะถูกแปลงให้อยู่ในรูปของโมเดลเชิงรูปนัยซีเอสพีด้วย Algorithm 1 ที่ได้นำเสนอไปแล้วในบทที่ 3 ผลลัพธ์ที่ได้จากการแปลงแสดงดังภาพที่ 4.19

```
func Buffer(bufSize int) (consumer chan int, producer chan int) {
        buffer := make([]int, bufSize)
        consumer = make(chan int)
        producer = make(chan int)
```

```
        in, out := 0, 0
        go func() {
                for {
                        if in < out+10 {
                                // We have room in the buffer, check the producer.
                                select {
                                case i := <-producer:
                                        buffer[in%bufSize] = i
                                        in++
                                default: // don't block
                                }
                        }

                        if out < in {
                                // We have something in the buffer, check the
consumer.
                                select {
                                case consumer <- buffer[out%bufSize]:
                                        out++
                                default: // don't block
                                }
                        }
                }
        }()
        return consumer, producer
}
```

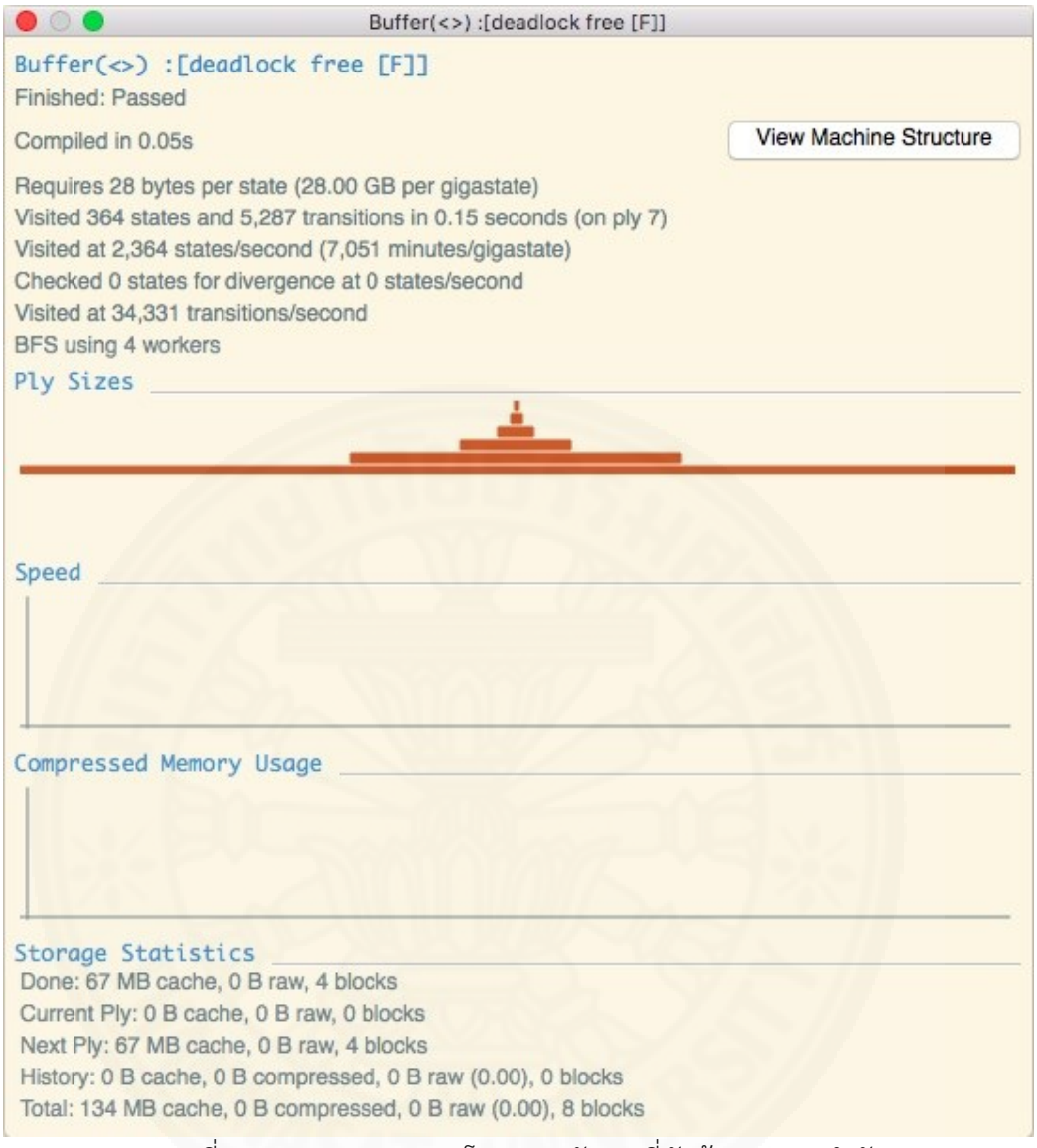
ภาพที่ 4.18 ชุดคำสั่งของซอฟต์แวร์ปัญหาที่พักข้อมูลขนาดจำกัดในภาษาโก

```
in,out:integer; in := 0; out := 0;
BUFFER::*[in<out +10; producer?buffer(in mod 10) -> in := in + 1
    [] out < in; consumer?more() -> consumer!buffer(out mod 10) ;
       out := out + 1
]
```

ภาพที่ 4.19 โมเดลเชิงรูปนัยที่อยู่ในรูปของซีเอสพี

4.3.3. ผลลัพธ์ของการทวนสอบ

การทวนสอบซอฟต์แวร์นี้ไม่สามารถที่จะสร้างปริภูมิสถานะออกมาเป็นรูปภาพได้ทุกสถานะ เนื่องจากปริภูมิสถานะมีขนาดใหญ่มาก แต่ผลลัพธ์ที่ได้จากการทวนสอบนั้นทำให้เห็นได้ว่าไม่เกิดสภาวะการติดตายของปัญหาปัญหาที่พักข้อมูลขนาดจำกัด ซึ่งเป็นไปตามคุณลักษณะที่ต้องการทวนสอบ ดังแสดงผลการทวนสอบในภาพที่ 4.20



ภาพที่ 4.20 ผลการทวนสอบโปรแกรมปัญหาที่พักข้อมูลขนาดจำกัด

คุณสมบัติที่ทวนสอบ	ผลลัพธ์
BUFFER(<>) :[deadlock free [F]]	Passed

ภาพที่ 4.21 ตารางผลการทวนสอบโปรแกรมปัญหาที่พักข้อมูลขนาดจำกัด

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

ในบทนี้จะกล่าวถึงสรุปผลการวิจัยและข้อเสนอแนะ หลังจากที่ผู้วิจัยได้นำเสนอวิธีการทวนสอบเชิงรูปนัยสำหรับซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโก และได้นำวิธีดังกล่าวไปทวนสอบกับกรณีศึกษา

5.1 สรุปผลการวิจัย

งานวิจัยนี้ได้เสนอวิธีการทวนสอบเชิงรูปนัยสำหรับซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโก ซึ่งเป็นภาษาที่สนับสนุนการทำงานแบบภาวะพร้อมกัน วิธีการทวนสอบเชิงรูปนัยสำหรับซอฟต์แวร์ที่ถูกพัฒนาด้วยภาษาโกแบ่งเป็น 2 ขั้นตอน ได้แก่ ขั้นตอนการสร้างโมเดลเชิงรูปนัยของรหัสต้นฉบับภาษาโกให้อยู่ในรูปซีเอสพี และขั้นตอนการทวนสอบความถูกต้องของโปรแกรมภาษาโกด้วยเครื่องมือดีเอฟอาร์

ในขั้นตอนแรก ผู้วิจัยได้ออกแบบและพัฒนาอัลกอริทึมสำหรับสร้างแบบจำลองเชิงรูปนัยของรหัสต้นฉบับที่ถูกพัฒนาด้วยภาษาโกให้อยู่ในรูปของซีเอสพี อัลกอริทึมดังกล่าวใช้เทคนิคการเทียบชุดคำสั่งระหว่างภาษาโกและซีเอสพี ทั้งนี้เพราะภาษาโกเป็นภาษาที่ถูกออกแบบและพัฒนาขึ้นมาตามแนวคิดของ Hoare's Communicating Sequential Processes อัลกอริทึมที่ถูกออกแบบและพัฒนาสำหรับงานวิจัยนี้จะครอบคลุมคำสั่งพื้นฐานทั้งหมดของภาษาโก 6 คำสั่งได้แก่ คำสั่ง if-else คำสั่ง func คำสั่ง go คำสั่ง var คำสั่ง for และคำสั่ง channel ทั้งนี้ผู้วิจัยได้ตรวจสอบความถูกต้องของอัลกอริทึมด้วยการเปรียบเทียบระหว่างโมเดลเชิงรูปนัยที่ถูกสร้างขึ้นด้วยอัลกอริทึมและโมเดลเชิงรูปนัยที่ถูกนำเสนอใน (Hoare, C. A. R., 1978) สำหรับปัญหาอาหารเย็นของนักปราชญ์และปัญหาที่พักข้อมูลขนาดจำกัดซึ่งเป็นกรณีศึกษาด้วย

สำหรับขั้นตอนการทวนสอบความถูกต้องของโปรแกรมภาษาโกด้วยเครื่องมือเอฟดีอาร์ ผู้วิจัยได้ทวนสอบโปรแกรมที่ถูกพัฒนาด้วยภาษาโกจาก (The Go Authors, 2011) ซึ่งเป็นกรณีศึกษา ก่อนที่จะทำการทวนสอบด้วยเครื่องมือเอฟดีอาร์ ผู้วิจัยจะต้องสร้างโมเดลเชิงรูปนัยที่อยู่ในรูปซีเอสพีตามขั้นตอนแรกที่ได้กล่าวไปแล้วข้างต้นก่อน แล้วจึงนำโมเดลเชิงรูปนัยมาทวนสอบกับเครื่องมือดีเอฟอาร์ ผลการทวนสอบกับกรณีศึกษาพบว่า เครื่องมือเอฟดีอาร์สามารถตรวจสอบความสอดคล้องกันและคุณสมบัติความปลอดภัยของโมเดลเชิงรูปนัยที่เป็นตัวแทนของคุณลักษณะของโปรแกรมและโปรแกรมได้อย่างถูกต้อง ทำให้เพิ่มความน่าเชื่อถือให้กับโปรแกรมที่ถูกพัฒนาด้วยภาษาโก ช่วยลดความผิดพลาดที่เกิดขึ้นกับโปรแกรมที่ถูกนำไปใช้งานจริง และทำให้ต้นทุนที่ใช้ในการพัฒนาซอฟต์แวร์ลดลง

5.2 ข้อเสนอแนะ

1.งานวิจัยนี้สามารถพัฒนาอัลกอริทึมการสร้างโมเดลเชิงรูปนัยของโปรแกรมภาษาโกให้ครอบคลุมได้มากกว่า 6 ชุดคำสั่งตามได้นำเสนอไว้ในขอบเขตงานวิจัยได้

2.เราสามารถใช้เครื่องมือเอฟดีอาร์ในการทวนสอบคุณสมบัติอื่นๆของโปรแกรมภาษาโก ซึ่งนอกเหนือจากการตรวจสอบคุณสมบัติความปลอดภัยของคุณลักษณะของโปรแกรมและโปรแกรมได้

3.งานวิจัยนี้ไม่ได้สนใจเรื่องประสิทธิภาพด้านเวลาและหน่วยความจำของการทวนสอบโปรแกรมภาษาโก แต่อย่างไรก็ตามโดยปกติในการทำงานแบบภาวะพร้อมกัน ผู้วิจัยอาจจะต้องเผชิญกับปัญหาการระเบิดของสถานะ (State Explosion Problem) ดังนั้นผู้วิจัยควรจะต้องตระหนักถึงประเด็นเรื่องประสิทธิภาพด้านเวลาและหน่วยความจำด้วย

4.จากผลการทดลองกับกรณีศึกษาปัญหาอาหารเย็นของปราชญ์ (Hoare, C. A. R., 1978) วิธีการทวนสอบที่ได้นำเสนอในงานวิจัยนี้สามารถสร้างโมเดลเชิงรูปนัยของภาษาโกสำหรับปัญหาดังกล่าวและทวนสอบคุณสมบัติการติดตายที่เกิดขึ้นได้ ในปัจจุบันมีการนำเสนอวิธีสำหรับแก้ปัญหาการติดตายหลายวิธี (Peterson, J. L., & Silberschatz, A., 1985) ดังนั้นเราสามารถประยุกต์ใช้วิธีการทวนสอบที่ได้นำเสนอกับวิธีแก้ปัญหาการติดตายอื่นๆ เพื่อทวนสอบความถูกต้องของวิธีการเหล่านั้นได้

รายการอ้างอิง

หนังสือและบทความในหนังสือ

Schneider, S. (1999). Concurrent and Real Time Systems: The CSP Approach. By Steve Schneider. Published by John Wiley and Sons Ltd., Chichester, UK.

Peterson, J. L., & Silberschatz, A. (1985). *Operating system concepts* (Vol. 2). Reading, MA: Addison-Wesley.

บทความวารสาร

Hoare, C. A. R. (1978). *Communicating sequential processes* (pp. 413-443). Springer New York.

Möller, M. (2002). Specifying and checking java using CSP. In Workshop on Formal Techniques for Java-like Programs-FTf-JP.

Moran, M., Heather, J., & Schneider, S. (2014). Verifying anonymity in voting systems using CSP. Formal Aspects of Computing, 26(1), 63-98.

Rohrmair, G. T. (2005). Using CSP to Verify Security-Critical Applications. PhD thesis, University of Oxford (Hilary 2005).

Li, Liyi, Elsa Gunter, and William Mansky. "Symbolic Analysis Tools for CSP."*International Colloquium on Theoretical Aspects of Computing*. Springer International Publishing, 2014.

Li L, Gunter E and Mansky W. “Symbolic Semantics for CSP”. Springer-Verlag ICC'14. (2014).

Roscoe, A. W., & Wu, Z. (2006). Verifying Statemate statecharts using CSP and FDR. In Formal Methods and Software Engineering (pp. 324-341). Springer Berlin Heidelberg.

Mantas, J. M., & Palma, A. (1997, May). Designing reusable software components following the CSP distributed programming model. In *Software Engineering for Parallel and Distributed Systems, 1997. Proceedings,, Second International Workshop on* (pp. 174-185). IEEE.

Wang, X., Kwiatkowska, M., Theodoropoulos, G., & Zhang, Q. (2005). Towards a unifying CSP approach to hierarchical verification of asynchronous hardware. *Electronic Notes in Theoretical Computer Science*, *128*(6), 231-246.

สื่ออิเล็กทรอนิกส์

University of Oxford. FDR Manual Release 3.3.1. 2015. Retrieved from https://www.cs.ox.ac.uk/projects/fdr/downloads/fdr-manual.pdf (accessed February 15, 2016)

The Go Authors. Effective go. 2011. Retrieved from http://golang.org/doc/effective_go.html (accessed January 15, 2016)

The Go Authors. How to Write Go Code. 2011. Retrieved from https://golang.org/doc/code.html (accessed January 15, 2016)

University of Oxford. FDR3 — The CSP Refinement Checker. 2015. Retrieved from https://www.cs.ox.ac.uk/projects/fdr/ (accessed February 15, 2016)

The Go Authors. Package csp. 2011. Retrieved from https://godoc.org/github.com/thomas11/csp (accessed January 30, 2016)

ประวัติผู้เขียน

ชื่อ	ว่าที่ร้อยตรีอนุชิต ประเสริฐสังข์
วันเดือนปีเกิด	8 กุมภาพันธ์ พ.ศ. 2535
ประสบการณ์ทำงาน	2558 – ปัจจุบัน: Software Engineer บริษัท ออดอี(ประเทศไทย) จำกัด 2557: Software Developer บริษัท ซอฟต์พลัสเทคโนโลยี จำกัด