รหัสโครงการ SUT7-705-56-12-40

# รายงานการวิจัย

# คอมไพเลอร์ที่มีเป้าหมายเป็นจาวาสคริปต์โดยใช้ตัวแจงส่วนเชิงทำนาย

# A compiler targeting JavaScript using predictive parser

ได้รับทุนอุดหนุนการวิจัยจาก
มหาวิทยาลัยเทคโนโลยีสุรนารี

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

**รหัสโครงการ** SUT7-705-56-12-40

**รายงานการวิจัย**

# คอมไพเลอร์ที่มีเป้าหมายเป็นจาวาสคริปต์โดยใช้ตัวแจงส่วนเชิงทำนาย

# A compiler targeting JavaScript using predictive parser

**คณะผู้วิจัย**

**หัวหน้าโครงการ**
**ผู้ช่วยศาสตราจารย์ ดร. ชาญวิทย์ แก้วกสิ**
สาขาวิชาวิศวกรรมคอมพิวเตอร์
สำนักวิชาวิศวกรรมศาสตร์



**ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ 2556**
**ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว**

**ตุลาคม 2562**

# กิตติกรรมประกาศ

งานวิจัยชิ้นนี้จะไม่สามารถสำเร็จลุล่วงไปได้ด้วยดีหากไม่ได้รับการสนับสนุนจากหน่วยงานในมหาวิทยาลัย ทั้งในระดับสาขาวิชา สำนักวิชา สถานวิจัยสำนักวิชาวิศวกรรมศาสตร์ และสถาบันวิจัยและพัฒนา และผู้ช่วยวิจัยทุกท่าน

ผู้วิจัยขอขอบคุณ Guillaume Laforge ผู้จัดการโครงการคอมไพเลอร์ภาษา Groovy และ Jochen Theodorou หัวหน้าทีมพัฒนาคอมไพเลอร์ภาษา Groovy สำหรับคำแนะนำในการแก้ปัญหาของระบบคอมไพเลอร์ของภาษา Groovy ในเชิงลึก ทำให้สามารถสร้าง GroovyJS ได้สำเร็จลุล่วงเป็นอย่างดี

## บทคัดย่อ

งานวิจัยชิ้นนี้ศึกษาและพัฒนา GroovyJS ซึ่งเป็นคอมไพเลอร์ลักษณะพิเศษที่แปลงภาษา Groovy ให้เป็น JavaScript ทำให้โปรแกรมผลลัพธ์สามารถทำงานได้บนเว็บเบราเซอร์ โดยทำการศึกษากฎการแปลงจากภาษา CoffeScript และนำตัวแจงส่วนจากคอมไพเลอร์ภาษา Groovy มาพัฒนาต่อโดยการเพิ่ม predicate เข้าไปในกลไกการแจงส่วนทำให้สามารถสร้างคอมไพเลอร์ที่ขึ้นกับบริบทและสามารถแปลงไวยากรณ์ภาษา Groovy ที่มีความกำกวมไปเป็น JavaScript ได้

จากผลงานวิจัย พบว่าสามารถพัฒนาคอมไพเลอร์ GroovyJS ที่สามารถแปลงไวยากรณ์ภาษา Groovy ได้ถูกต้องตามข้อกำหนด 100% บนฐานคอมไพเลอร์ของ Groovy 2.1.9 และ Groovy 2.2.2 จากฐานคอมไพเลอร์ทั้งหมด 4 รุ่นที่นำมาทดสอบ โดยตัวคอมไพเลอร์ GroovyJS ทำงานได้ดีที่สุดบนระบบรันไทม์ของ Groovy 2.2.2

## Abstract

The research work described in this report presents the study and development of GroovyJS, a compiler that transpiles the Groovy language into JavaScript. The resulting codes will be able to run normally on Web Browsers. Compiler rules are studied from the CoffeeScript compiler. The base compiler is the modification of the Groovy compilers which were added predictive predicates to make GroovyJS context-sensitive and able to translate ambiguous syntax of the Groovy language into the proper JavaScript.

From the experimental results, the compilers, based on Groovy 2.1.9 and Groovy 2.2.2, can successfully compile the Groovy language in the 100% correction according to the specification. The experiments were conducted on 4 different versions of the base Groovy compilers. The result also found that GroovyJS compilers work best on the runtime system of Groovy 2.2.2.

# สารบัญ

**เรื่อง** **หน้า**

**เรื่อง** **หน้า**

# บทที่ 1
# บทนำ

## 1.1 ความสำคัญ ที่มาของปัญหาที่ทำการวิจัย

ภาษาจาวาสคริปต์ [4] (JavaScript) เป็นภาษาหลักที่ใช้ร่วมกับภาษา HTML ในเว็บเบราเซอร์เพื่อสร้างเว็บแอพพลิเคชัน ด้วยจาวาสคริปต์เป็นภาษาที่มีความสำคัญต่อการใช้งานแอพพลิเคชันทำให้มีทั้งการศึกษาเชิงการปรับปรุงประสิทธิภาพของเครื่องจักรเสมือนที่ใช้รันจาวาสคริปต์ [6] และการศึกษาแคลคูลัสเชิงความหมายของจาวาสคริปต์ [7, 9, 11] อย่างกว้างขวาง

เฟรมเวิร์คหลายกลุ่มที่ทำงานบนเว็บเบราเซอร์เป็นหลักนั้นต้องการภาษาที่มีความกระชับและสนับสนุนเป้าหมายที่เป็นจาวาสคริปต์ ในปัจจุบันมีภาษาจำนวนเพิ่มขึ้นที่มีเป้าหมายเป็นจาวาสคริปต์เพื่อเพิ่มแนวทางการพัฒนาเว็บแอพพลิเคชันให้หลากหลายขึ้น เช่น คอมไพเลอร์ที่แปลงภาษาจาวาเป็นจาวาสคริปต์ใน Google Web Toolkit [13], คอมไพเลอร์ภาษา Objective-J [1] และคอมไพเลอร์ภาษา CoffeeScript [3] เป็นต้น

อย่างไรก็ตามกลุ่มนักพัฒนาอีกหลายกลุ่มยังยึดติดกับภาษาที่เคยใช้อยู่กับเฟรมเวิร์คที่ทำงานฝั่งแม่ข่าย และเพื่อให้สามารถนำทักษะเชิงภาษาโปรแกรมที่ใช้กับเฟรมเวิร์คฝั่งแม่ข่ายไปพัฒนาแอพพลิเคชั่นบนเว็บเบราเซอร์เป็นหลักได้นั้นจำเป็นต้องมีภาษาโปรแกรมที่ 1. มีไวยากรณ์คล้ายภาษาที่นักพัฒนาคุ้นเคย 2. แปลงเป็นจาวาสคริปต์และทำงานกับจาวาสคริปต์ได้โดยตรง 3. ตัวคอมไพเลอร์มีความเร็วในระดับที่ไม่ลด Developer Usability จึงมีความจำเป็นในการคิดค้นภาษาโปรแกรมใหม่ขึ้นเพื่อให้ได้คุณสมบัติดังกล่าวครบถ้วน

ในงานวิจัยนี้เสนอภาษา Groovy.JS โดยเลือกภาษา Groovy [8] มาเป็นภาษาต้นแบบสำหรับสร้าง Groovy.JS ซึ่งจะเป็นภาษาที่มีคอมไพเลอร์เพื่อสร้างรหัสเป้าหมายเป็นจาวาสคริปต์ เนื่องจากผลการสำรวจโดย eWeek [14] พบว่าภาษา Groovy มีการเติบโตมากที่สุดในช่วงปี 2009-2011 และมีการใช้งานในเฟรมเวิร์คฝั่งแม่ข่ายอย่างแพร่หลาย รวมทั้ง Groovy มีลักษณะเป็นภาษาเชิงพลวัตร จึงเป็นภาษาต้นแบบที่เหมาะสมสำหรับการสร้าง Groovy.JS

## 1.2 ทฤษฎี สมมุติฐาน และกรอบแนวความคิดของโครงการวิจัย

งานวิจัยนี้จะสร้างอยู่บนพื้นฐานของการสร้างคอมไพเลอร์ชนิด LL(k) ที่มีเป้าหมายเป็นจาวาสคริปต์ โดยที่ k มีค่าเป็นจำนวนเต็ม และจะหลีกเลี่ยงการใช้ LL(*) และอัลกอริทึมการย้อนรอยในตัวแจงส่วน (parser) แต่จะใช้ตัวทำนาย (predicate) เพื่อเพิ่มประสิทธิภาพการเลือกกฎของคอมไพเลอร์แทน โดยมีสมมติฐานเบื้องต้นว่าคอมไพเลอร์ LL(k) ที่มีค่า k เป็นจำนวนเต็มร่วมกับตัวแจงส่วนเชิงทำนาย (predictive parser) จะสามารถใช้สร้างคอมไพเลอร์ที่มีเป้าหมายเป็นจาวาสคริปต์ได้ประสิทธิภาพใกล้เคียงหรือเทียบเท่า LALR(k) ที่สร้างด้วยตัวสร้างคอมไพเลอร์ (compiler generator)

## 1.3 วัตถุประสงค์ของโครงการวิจัย

1. เพื่อศึกษาเชิงความหมายและคิดค้นกฎการแปลงจากภาษา Groovy.JS ให้เป็นภาษาจาวาสคริปต์
2. เพื่อพัฒนาคอมไพเลอร์ต้นแบบภาษา Groovy.JS

## 1.4 ขอบเขตของโครงการ

1. คิดค้นและพัฒนาคอมไพเลอร์ต้นแบบที่ทำงานได้บนเครื่องจักรเสมือนจาวาสคริปต์ V8 ของ Google
2. เปรียบเทียบคอมไพเลอร์ที่สร้างขึ้นกับภาษา JavaScript ที่สร้างด้วย CoffeeScript
3. วัดประสิทธิภาพของคอมไพเลอร์

## 1.5 ระเบียบวิธีวิจัย

1. ศึกษาโครงสร้างภาษา Groovy เพื่อใช้เป็นไวยากรณ์ต้นแบบ
2. สร้างไวยากรณ์ภาษา Groovy.JS
3. สร้างกฎการแปลงภาษา Groovy.JS ไปเป็นภาษาจาวาสคริปต์
4. สร้างคอมไพเลอร์ภาษา Groovy.JS
5. ทดลองวัดประสิทธิภาพของคอมไพเลอร์บนระบบรันไทม์ที่แตกต่างกัน

## 1.6 ประโยชน์ที่คาดว่าจะได้รับ

1. เผยแพร่เป็นบทความทางวิชาการ
2. ได้คอมไพเลอร์ที่มีเป้าหมายเป็นจาวาสคริปต์ เพื่อใช้พัฒนาแอพพลิเคชันและเฟรมเวิร์คสำหรับเว็บ

## 1.7 บทสรุป

สำหรับบทที่ 1 นี้เป็นเนื้อหาสำหรับบทนำ ส่วนในบทที่ 2 จะเป็นเนื้อหาเกี่ยวกับการทบทวนวรรณกรรม สวนในบทที่ 3 จะกล่าวถึงกฎและอัลกอริทึมสำหรับการแปลงภาษา Groovy เป็น JavaScript ในบทที่ 4 จะกล่าวถึงกระบวนการทดสอบและผลการทดสอบ สำหรับบทที่ 5 เป็นบทสรุปและอภิปรายงานในอนาคต

# บทที่ 2
## ทบทวนวรรณกรรม

ในบทนี้จะเป็นการทบทวนวรรณกรรมของระบบคอมไพเลอร์ที่เกี่ยวข้องการแปลงภาษาตั้งต้นให้เป็นภาษา JavaScript ซึ่งการออกแบบภาษาเพื่อใช้งานในเว็บเบราเซอร์มีประเด็นสำคัญที่ต้องพิจารณาดังต่อไปนี้คือ 1. ภาษาดังกล่าวต้องสามารถใช้ร่วมกับโปรแกรมภาษาจาวาสคริปต์เดิมได้ทันที 2. คอมไพเลอร์ของภาษาดังกล่าวต้องมีความเร็วอยู่ในระดับที่สามารถใช้งานทั่วไปได้โดยไม่ทำให้ Developer Usability ลดลง

การทบทวนวรรณกรรมจะแบ่งออกเป็น 2 กลุ่มคือ กลุ่มภาษาปกติ กลุ่มภาษาที่เกี่ยวข้องกับ JavaScript โดยตรง และกลุ่มเครื่องมือ

### 2.1 กลุ่มภาษาปกติ

ภาษา Java [2] เป็นภาษาที่เริ่มต้นพัฒนาโดย James Gosling ที่ Sun Microsystems เริ่มใช้ครั้งแรกในปี ค.ศ. 1995 โดยลักษณะไวยากรณ์หลักนั้นมีลักษณะคล้ายภาษา C และ C++ ภาษา Java ทำงานบนเครื่องจักรเสมือนที่เรียกว่า Java Virtual Machine (JVM) ซึ่งสามารถทำงานบนสถาปัตยกรรมของฮาร์ดแวร์ต่างๆ ได้หลายชนิด ในปี ค.ศ. 2007 ได้มีการเปิดต้นรหัส Java Development Kit (JDK) ภายใต้สัญญาอนุญาต GNU GPL เรียกว่า OpenJDK เทคโนโลยีภาษา Java ครอบคลุมหลายพื้นที่ที่สำคัญ เช่น Web Application, Mobile Application เป็นต้น อีกทั้งยังมีการปรับใช้ภาษา Java ในคอมไพเลอร์อื่นๆ นอกเหนือจาก JDK เช่น คอมไพเลอร์ของ Google Web Toolkit

ภาษา Groovy [8] เป็นภาษาโปรแกรมที่มีความหลากหลายทางมโนทัศน์และเป็นภาษาประเภทไดนามิคที่ทำงานบนแพล็ตฟอร์ม Java มีจุดเด่นเรื่องความสามารถในการช่วยสร้างภาษาเฉพาะทาง (Domain Specific Language) โดยภาษา Groovy ได้รับอิทธิพลส่วนใหญ่จากภาษา Java (รวมทั้ง Ruby และ Perl) Groovy 1.0 ปรากฏขึ้นครั้งแรกในปี ค.ศ. 2007 และ Groovy 2.0 ออกสู่สาธารณะในปี ค.ศ. 2012 โดยมีการเพิ่มความสามารถในการคอมไพล์รหัสแบบสแตติกเข้ามาในชุดของคอมไพเลอร์ ในงานวิจัยนี้ใช้โครงสร้างไวยากรณ์ ANTLR ของ Groovy และระบบพาร์สใน Groovy คอมไพเลอร์ซึ่งมีลักษณะเป็น LL(k) เป็นตัวตั้งตนสำหรับพัฒนา GroovyJS โดยทำการเพิ่มการพาร์สที่สนับสนุนความกำกวมเข้าไปเพื่อให้สามารถสร้างโค้ดที่เป็น JavaScript ได้มีประสิทธิภาพมากขึ้น

## 2.2 กลุ่มภาษาที่เกี่ยวข้องกับ JavaScript

CoffeeScript [2] เป็นภาษาโปรแกรมที่แปลงเป็นภาษาจาวาสคริปต์ ตัวภาษาเพิ่มลักษณะเด่นหลายประการจากภาษา Ruby, Python, Haskell และ Groovy เพื่อให้ภาษาอ่านง่ายขึ้นและใช้งานได้สะดวกมากกว่าภาษาจาวาสคริปต์ จุดเด่นของ CoffeeScript คือ 1. การแปลงเป็นภาษาจาวาสคริปต์ที่ยังมีโครงสร้างชัดเจนและสามารถทำความเข้าใจได้ง่าย 2. โปรแกรมที่เขียนด้วยภาษา CoffeeScript สามารถใช้งานโปรแกรมภาษาจาวาสคริปต์อื่นได้โดยตรงและ 3. รหัสที่สร้างขึ้นไม่มีขอขวดเรื่องประสิทธิภาพในการใช้งาน จุดด้อยของภาษา CoffeeScript คือ 1. ไวยากรณ์ภาษา CoffeeScript เป็นไวยากรณ์ที่สร้างขึ้นใหม่ ทำให้นักพัฒนาที่คุ้นเคยกับภาษาเดิมจำเป็นต้องเรียนรู้เชิงไวยากรณ์ใหม่ทั้งหมด 2. ความยากในการระบุบรรทัดการทำงานที่เกิดข้อผิดพลาดขึ้นในขณะรัน เนื่องจากเมื่อคอมไพล์โปรแกรมไปเป็นภาษาจาวาสคริปต์แล้วจะไม่มีข้อมูลเชิงบรรทัดของต้นรหัสเดิมเหลืออยู่ในรหัสปลายทาง ในส่วนของคอมไพเลอร์นั้น คอมไพเลอร์ภาษา CoffeeScript เป็นประเภท LALR และไม่มีการตรวจสอบระบบไทพ์เพิ่มเติม ทำให้การคอมไพล์สามารถทำได้อย่างรวดเร็ว

Objective-J [1] เป็นภาษาโปรแกรมที่พัฒนาขึ้นตามรูปแบบภาษา Objective-C โดยคอมไพเลอร์จะแปลงภาษาของ Objective-J เป็นจาวาสคริปต์ จุดเด่นของ Objective-J คือความคล้ายกับภาษา Objective-C และการมีการนิยามโครงสร้างการสืบทอดเชิงวัตถุให้กับรหัสที่ตัวคอมไพเลอร์สร้างขึ้น ตัวคอมไพเลอร์ภาษา Objective-J พัฒนาด้วยจาวาสคริปต์ทำให้สามารถใช้งานได้โดยตรงบนเว็บเบราเซอร์ Objective-J มีจุดด้อยคือยึดติดกับเฟรมเวิร์คสำหรับสร้างแอพพลิเคชันมากเกินไป เนื่องจากเป็นความตั้งใจในการออกแบบตั้งแต่แรกของตัวภาษาให้เป็นส่วนหนึ่งของเฟรมเวิร์คมากกว่าการใช้งานเป็นคอมไพเลอร์เดี่ยว

TypeScript [12] เป็นภาษาที่พัฒนาขึ้นในปี 2012 มีลักษณะของภาษาครอบคลุมไวยากรณ์ของภาษา JavaScript โดยถูกออกแบบขึ้นให้สามารถพัฒนาแอพพลิเคชันขนาดใหญ่ได้สะดวกและแปลงเป็น JavaScript เมื่อจะใช้งานจริงบนเว็บเบราเซอร์ ภาษา TypeScript สนับสนุนการประกาศข้อมูลไทพ์เพื่อให้สามารถใช้งานร่วมกับไลบรารี่ปัจจุบันเช่น JQuery หรือ Node.JS ได้

Grooscript [5] เป็น library สำหรับแปลง Groovy code เป็น JavaScript โดยเป็นงานที่คล้ายกันกับงานวิจัยชิ้นนี้มากที่สุด ข้อแตกต่างคือ GrooScript ใช้โครงสร้างที่พัฒนาขึ้นต่างหากและเน้นไปที่การสร้างภาษาเฉพาะทาง ส่วน GroovyJS นั้นเน้นการแปลงคลาสในลักษณะที่ต้องใช้ predicate เพื่อลดความกำกวม และให้สามารถใช้สำหรับเฟรมเวิร์คที่มากกว่า 1 ตัวได้

ในปัจจุบัน ภาษาที่มีความสามารถในการแปลงเป็น JavaScript ได้เช่น CoffeeScript [2], TypeScript [12], Grooscript [5] หรือแม้แต่ GroovyJS ในรายงานวิจัยฉบับนี้ สามารถทำงานได้ทั้งบนเว็บเบราเซอร์ และใช้เขียนเป็นโปรแกรมที่สามารถรันบน Server ได้ด้วยการใช้ระบบไลบรารี่ของ Node.JS

## 2.3 กลุ่มเครื่องมือ

Google Web Toolkit (GWT) [13] เป็นชุดเครื่องมือที่มีคอมไพเลอร์ที่แปลงภาษาจาวา (Java) เป็นภาษาจาวาสคริปต์ โดยมีกลไกการปรับปรุงประสิทธิภาพในระดับสูงทำให้สร้างรหัสภาษาจาวาสคริปต์ได้อย่างมีประสิทธิภาพ อย่างไรก็ตาม GWT มีจุดด้อยสองประเด็นหลักคือ 1. รหัสที่สร้างด้วย GWT ไม่สามารถใช้งานร่วมกับโปรแกรมจาวาสคริปต์อื่นได้ทันทีโดยต้องการการปรับแต่งก่อนจึงจะสามารถใช้งานร่วมกับโปรแกรมทั่ว ๆ ไปเช่น jQuery ได้ 2. คอมไพเลอร์ของ GWT พัฒนาด้วยภาษาจาวาจึงมี startup-time ที่ลด Developer Usability 3. ภาษาจาวาที่เป็นภาษาตั้งต้นของ GWT ไม่ได้รับการออกแบบให้เป็นภาษาเชิงพลวัตร ทำให้ขัดกับหลักการพื้นฐานของการสร้างภาษาจาวาสคริปต์ที่สร้างมาเพื่อแพล็ตฟอร์มเว็บในทางกลับกัน จุดเด่นที่สำคัญของการใช้ภาษาจาวาเป็นภาษาตั้งต้นคือมีเครื่องมือสนับสนุนการพัฒนาได้อย่างกว้างขวาง

ANTLR [10] เป็นเฟรมเวิร์คสำหรับพัฒนาคอมไพเลอร์ โดยในรุ่น 4.0 และ 4.5 มีการสนับสนุน การสร้างคอมไพเลอร์ด้วย JavaScript สำหรับระบบคอมไพเลอร์ของ GroovyJS ที่พัฒนาขึ้นใช้ ANTLR จากคอมไพเลอร์ภาษา Groovy [8] หลายรุ่น ดังนี้ Groovy 2.19, 2.2.2, 2.3.11 และ Groovy 2.4.7

## 2.4 บทสรุป

ในบทนี้ได้กล่าวถึงระบบคอมไพเลอร์และเครื่องมือที่เกี่ยวข้องกับการพัฒนาภาษา GroovyJS ทั้งทางตรงและทางอ้อม โดยสามารถแบ่งได้เป็นภาษาปกติคือ Java และ Groovy ภาษาที่เกี่ยวข้องกับ JavaScript คือ CoffeeScript, Objective-J, TypeScript และ Grooscript สำหรับกลุ่มที่สามคือเครื่องมือ โดน GWT เป็นตัวอย่างเครื่องมือที่ใช้แปลงภาษา Java เป็น JavaScript และสำหรับ ANTLR นั้นเป็นเฟรมเวิร์คสำหรับสร้างคอมไพเลอร์ ในบทถัดไปจะกล่าวถึงวิธีการสร้างระบบคอมไพเลอร์ของ GroovyJS

# บทที่ 3
# การพัฒนา GroovyJS

ในบทที่ 3 นี้เป็นการอธิบายโครงสร้างของคอมไพเลอร์ และกฎในการแปลงจากภาษา Groovy เป็นภาษา JavaScript โดยยืมโครงสร้างการแปลงค่ามาจากกลไกของภาษา CoffeScript [2]

## 3.1 ระบบ Build

ในส่วนของระบบ Build จะเป็นการใช้ระบบของ Gradle

```
apply plugin: 'groovy'

sourceCompatibility  = 1.7
targetCompatibility  = 1.7

repositories {
    mavenCentral()
}

dependencies {
    compile     'org.codehaus.groovy:groovy-all:2.2.2'

    testCompile 'org.spockframework:spock-core:1.0-groovy-2.0'
    testCompile 'org.mozilla:rhino:1.7R4'
}
```

โดยภาษาที่ใช้พัฒนาระบบเป็นภาษา Groovy ซึ่งมีความเข้ากันได้กับระบบคอมไพเลอร์ภาษา Java 7 (JDK 1.7) และมีการประกาศ Dependencies หลักดังต่อไปนี้

1. Groovy-all สำหรับใช้เป็นระบบคอมไพเลอร์หลักของ GroovyJS
2. Spock-Core 1.0 for Groovy 2.0 สำหรับใช้ตรวจสอบคุณภาพซอฟต์แวร์
3. Rhino 1.7R4 สำหรับตรวจสอบต้นไม้เชิงไวยากรณ์ระหว่างการพัฒนา

## 3.2 การใช้ Rhino Parser สำหรับตรวจสอบความถูกต้องของต้นไม้เชิงไวยากรณ์

ในการสร้าง parser เพื่อแปลงโปรแกรมภาษา Groovy ไปเป็น JavaScript นั้น จำเป็นต้องมีกลไกสำหรับตรวจสอบความถูกต้องของการแปลง โดยกลไกดังกล่าวจะต้องสามารถเปรียบเทียบความถูกต้องของต้นไม้เชิงไวยากรณ์ได้ โดยไม่ขึ้นกับการมี white space ปน ยกตัวอย่างเช่น

```
function A() { }
```

จะต้องเทียบเท่ากับ

```
function A() {

}
```

เป็นต้น

วิธีการที่ใช้ในงานวิจัยนี้คือจะต้องนำโปรแกรม JavaScript ทั้งสองเข้าไปพาร์สเพื่อเปลี่ยนเป็นต้นไม้เชิงไวยากรณ์ จากนั้นจึงนำต้นไม้ทั้งสองแปลงกับเป็นข้อความ (String) ที่ใช้จำนวน white space ที่เหมาะสม แล้วจึงนำมาเปรียบเทียบกัน

จากตัวอย่างข้างต้น โปรแกรมทั้งสองจะถูกพาร์สด้วย Rhino และเปลี่ยนกับเป็นข้อความจะเห็นได้ว่าโปรแกรมหลังพาร์สด้วย Rhino มีลักษณะที่เหมือนกันทุกประการและสามารถเทียบเท่ากันได้ด้วยการเทียบข้อความ ในขณะที่โปรแกรมต้นฉบับเป็นโปรแกรมที่เทียบเท่ากัน แต่ไม่เหมือนกันทุกประการในเชิงข้อความ

| **โปรแกรมต้นฉบับ** | **โปรแกรมหลังพาร์สด้วย Rhino** |
|---|---|
| `function A() { }` | `function A() {`<br>`}` |
| `function A(){`<br><br>`}` | `function A() {`<br>`}` |

ตามตัวอย่าง จะเป็นการตั้งค่า DEBUG_INDENT = 2 เพื่อให้การเปรียบเทียบเหมาะสมทั้งขนาด String และการให้ผู้พัฒนาสามารถอ่านได้ (human readable) เพื่อใช้ในการแก้ไขข้อผิดพลาดของตัวคอมไพเลอร์

```
class DebugPrintVisitor implements NodeVisitor {

    private String makeIndent(int depth) {
        StringBuilder sb = new StringBuilder(DEBUG_INDENT * depth);
        for (int i = 0; i < (DEBUG_INDENT * depth); i++) {
            sb.append(" ");
        }
        return sb.toString();
    }

    public boolean visit(AstNode node) {
        int tt = node.getType();
        String name = Token.typeToName(tt);
        buffer.append(makeIndent(node.depth()));
        buffer.append(name).append(" ");
        if (tt == Token.NAME) {
            buffer.append(" ").append(((Name)node).getIdentifier());
        }
        buffer.append("\n");
        return true;
    }

    public static String tree(String source) {
        def ce = new CompilerEnvirons()
        def node = new Parser(ce, ce.getErrorReporter())
                    .parse(source, '', 1)
        def dpv = new DebugPrintVisitor(new StringBuilder(1000))
        node.visit(dpv)
        return dpv.toString()
```

```
    }
}
```

โดยตัว Visitor จะทำการ visit ไปทุก ๆ โหนดของต้นไม้ AST เพื่อนำข้อมูลเข้ามาใส่ใน buffer ก่อนที่จะแปลงเป็นข้อความเพื่อใช้เปรียบเทียบกับต้นไม้ AST อีกต้น

**3.3 การแปลงคลาส**

โดยทั่วไปเราจะแปลงคลาสภาษา Groovy ไปเป็น JavaScript โดยใช้กฎการแปลงต่อไปนี้

คลาสในภาษา Groovy

```
class A {
}
```

ฟังก์ชันที่เทียบเท่าคลาสในภาษา JavaScript

```
var A;

A = (function() {

  function A() {}

  return A;

})();
```

โดยเมื่อตรวจสอบเจอชื่อคลาสในภาษา Groovy แล้วจะใช้ชื่อคลาสตั้งเป็นตัวแปรในภาษา JavaScript (เช่น var A;) จากนั้นจะสร้างฟังก์ชันชื่อเดียวกับคลาสนั้นเพื่อใช้เป็นตัวจำลองคอนสตรักเตอร์ในภาษา JavaScript แล้วคืนค่าให้เป็นตัวแปรดังกล่าวเพื่อใช้เป็นตัวจำลองคลาสในภาษา JavaScript ต่อไป

**3.4 การแปลงคลาสที่บรรจุ Closure**

คลาสที่บรรจุด Closure มีลักษณะเฉพาะคือ Closure ในภาษา Groovy จะถูกจำลองเป็น prototype-based function ในภาษา JavaScript

คลาสในภาษา Groovy

```
class A {
    def hello = { ->
        println "Hello World"
    }
}
```

ฟังก์ชันที่เทียบเท่าคลาสในภาษา JavaScript

```
var A;

A = (function() {

  function A() {}
```

```
    A.prototype.hello = function() {
      return println("Hello World");
    };

    return A;

  })();
```

จากโปรแกรมจะเห็นว่า Closure hello ซึ่งประกาศเป็นฟังก์ชันในภาษา Groovy แปงเป็น A.prototype.hello ซึ่งเป็นฟังก์ชันแบบ prototype-based ในภาษา JavaScript โดยจะอยู่ในรูปแบบที่ไม่มีชื่อ และมีการพิจารณาว่านิพจน์ println(“Hello World”) นั้นเป็นนิพจน์สุดท้ายในฟังก์ชัน ดังนั้นค่าของนิพจน์ดังกล่าวจะเป็นค่าที่คืนออกจากฟังก์ชันไม่ว่าจะมีหรือไม่ก็ตาม

โดยความหมายดังกล่าวเป็นความหมายของนิพจน์ในภาษา Groovy จึงทำให้เมื่อแปลงเป็นภาษา JavaScript แล้วนั้นจะมีคำสั่ง return กำกับไว้หน้า println(“Hello World”) เพื่อคืนค่าออกจากฟังก์ชัน สำหรับ Closure ของภาษา Groovy ในตัวอย่างของการแปลงนี้ประกาศในลักษณะ { -> } หมายถึงเป็น Closure ที่ไม่มีพารามิเตอร์ เทียบเท่า function() { } ในภาษา Groovy โดย Closure ที่มีจำนวนพารามิเตอร์ต่าง ๆ กันในภาษา Groovy จะสามารถแปลงเป็นภาษา JavaScript ได้ดังต่อไปนี้

1. Closure ที่ไม่มีพารามิเตอร์

```
{ -> }
```

Closure ที่ไม่มีพารามิเตอร์ สามารถแปลงเป็น JavaScript ได้ดังนี้

```
function() { }
```

2. Closure ที่มีพารามิเตอร์ 1 ตัว

```
{ it -> }
```

Closure ที่มีพารามิเตอร์ 1 ตัว สามารถแปลงเป็น JavaScript ได้ดังนี้

```
function(it) { }
```

3. Closure ที่มีพารามิเตอร์มากกว่า 1 ตัว

```
{ a, b, c -> }
```

Closure ที่มีพารามิเตอร์มากกว่า 1 ตัว สามารถแปลงเป็น JavaScript ได้ดังนี้

```
function(a, b, c) { }
```

4. Closure ที่มีพารามิเตอร์โดยปริยาย

```
{ }
```

Closure ที่มีพารามิเตอร์โดยปริยาย สามารถแปลงเป็น JavaScript ได้ดังนี้

```
function(it) { }
```

ในกรณีของ Closure ที่มีพารามิเตอร์โดยปริยายนั้นเป็นไวยากรณ์ของภาษา Groovy เพื่อทำให้ไวยากรณ์กระชับในการใช้งานการออกแบบภาษาเฉพาะทาง (Domain-Specific Language) จึงออกแบบให้ Closure ที่มีพารามิเตอร์โดยปริยายมีจำนวนพารามิเตอร์ 1 ตัว

### 3.5 การแปลงระบบแพ็คเกจ

แพ็คเกจเป็นกลไกการจัดกลุ่มคลาสที่มีเฉพาะในภาษาระดับสูง ในภาษา Groovy ใช้รูปแบบการจัดการแพ็คเกจแบบเดียวกับภาษา Java โดยใช้คีย์เวิร์ด package เป็นตัวประกาศ ในตัวอย่างต่อไปนี้เป็นการประกาศคลาส A อยู่ในแพ็คเกจ th.ac.sut

```
package th.ac.sut

class A {

  def hello = { v ->
  }

}
```

ในการแปลงเป็นภาษา JavaScript นั้นจำเป็นต้องตั้งกลไกของระบบแพ็คเกจขึ้นมาใหม่ โดยในงานวิจัยนี้จะจำลองระบบแพ็คเกจด้วยการใช้ JavaScript dictionary [2]

```
var th = th || {};
th.ac = th.ac || {};
th.ac.sut = th.ac.sut || {};

th.ac.sut.A = (function() {

  function A() {}

  A.prototype.hello = function(v) {
  };

  return A;

})();
```

ในการจำลองแพ็คเกจของภาษา Groovy ด้วย JavaScript นั้นทำโดยใช้อัลกอริทึมต่อไปนี้

```
if (hasPackage) {
    if (packageName.indexOf(".") != -1) {
        def packageSplit = packageName.split("\\.")
        String preResult = ""
        for (i in 1..<packageSplit.size()) {
            packageSplit[i] = packageSplit[i - 1]
                              + '.' + packageSplit[i]
        }
        for (pack in packageSplit) {
            preResult += pack + ' = ' + pack + ' || {};\n'
        }
        return 'var ' + preResult + result
```

```
    } else {
        return 'var ' + packageName + " = "
                  + packageName + " || {};"
                  + result
    }
}
```

### 3.6 การแปลงการสืบทอด

กลไกการสืบทอด (Inheritance) เป็นสมบัติปกติที่พบได้ในภาษาเชิงวัตถุรวมทั้งภาษา Groovy

```
package th.ac.sut

class A {
}

class B extends A {
}
```

โดยระบบกลไกการสืบทอดนั้นต้องการคุณสมบัติการจัดการแพ็คเกจก่อนจึงจะสามารถทำงานได้อย่างสมบูรณ์ และใช้ฟังก์ชัน __extends ที่เตรียมขึ้นเพื่อทำการโยงความสัมพันธ์แบบ inheritance ระหว่างคลาส

```
var th = th || {};
th.ac = th.ac || {};
th.ac.sut = th.ac.sut || {};

var __hasProp = {}.hasOwnProperty,
  __extends = function(child, parent) { for (var key in
parent) { if (__hasProp.call(parent, key)) child[key] =
parent[key]; } function ctor() { this.constructor = child; }
ctor.prototype = parent.prototype; child.prototype = new ctor();
child.__super__ = parent.prototype; return child; };

th.ac.sut.A = (function() {

  function A() {}

  return A;

})();

th.ac.sut.B = (function(_super) {

  __extends(B, _super);

  function B() {
    return B.__super__.constructor.apply(this, arguments);
  }

  return B;

})(th.ac.sut.A);
```

โดยประโยค __extends จะถูกใช้ในคลาสลูกดังตัวอย่างเพื่อโยงคลาส B และ th.ac.sut.A เข้าด้วยกัน

### 3.7 การแปลงสคริปต์

การแปลงสคริปต์เป็นรูปแบบการแปลงที่ง่ายที่สุด โดยไม่มีโครงสร้างคลาสเข้ามาเกี่ยวข้อง ซึ่งจะเป็นการประกาศฟังก์ชันขึ้นมาโดยตรงในไฟล์ต้นฉบับและสั่งให้ทำงานเลย

ตัวอย่างในส่วนนี้จะเป็นการแปลงการประกาศฟังก์ชันในสคริปต์ โดยประกอบไปด้วย 2 ฟังก์ชันคือ ฟังก์ชัน square และ cube

```
def square = { x -> x * x }
def cube =   { x -> square(x) * x }
```

โดยสริปต์ดังกล่าวจะแปลงไปเป็นภาษา JavaScript ในลักษณะต่อไปนี้

```
var cube, square;

square = function(x) {
    return x * x;
};

cube = function(x) {
    return square(x) * x;
};
```

### 3.8 การแปลงกระทำนิพจน์

การกระทำนิพจน์เป็นกลไกที่เกิดขึ้นโดยปกติในภาษาโปรแกรมโดยในตัวอย่างจะเป็นการแปลงนิพจน์การบวกจากภาษา Groovy ไปเป็นภาษา JavaScript จะสังเกตได้ว่ากลไกการประมวลผลดังกล่าวอยู่ในรูปแบบปกติเนื่องจากระบบ Type System ของ Groovy มีลักษณะกึ่งพลวัตรซึ่งสามารถแปลงไปเป็นการกระทำของภาษา JavaScript ได้ในลักษณะ 1:1

ตัวอย่างการบวกนิพจน์ในภาษา Groovy

```
class A {
  def add = {i, j ->
    return i + j
  }
}
```

ผลการแปลงการบวกนิพจน์ไปเป็นภาษา JavaScript

```
var A;
```

```
A = (function() {

  function A() {}

  A.prototype.add = function(i,j) {
    return i + j;
  };

  return A;

})();
```

### 3.9 การแปลงค่าพารามิเตอร์โดยปริยาย

ค่าพารามิเตอร์โดยปริยายเป็นสมบัติแบบหนึ่งในภาษา Groovy ที่จะสามารถระบุค่าโดยปริยายให้พารามิเตอร์แต่ละตัวสามารถมีค่าตั้งต้นได้โดยไม่ต้องระบุในฟังก์ชัน ยกตัวอย่างเช่น ฟังก์ชัน fill ระบุค่า c เป็น 1 และ q เป็น “coffee” ดังนั้นเมื่อเรียกใช้ fill() แล้วจะได้การเรียกที่เทียบเท่า fill(1, “coffee”) เป็นต้น

สำหรับในภาษา Groovy นั้น ค่าโดยปริยายจะเรียกว่า Initial Expression

ตัวอย่างฟังก์ชันภาษา Groovy ที่มีค่าพารามิเตอร์โดยปริยาย

```
def fill = { c = 1, q = "coffee" ->
    "Filling the ${c} with ${q}..."
}
```

และการแปลงเป็นฟังก์ชันในภาษา JavaScript โดยเพิ่มการตรวจค่า null ให้กับพารามิเตอร์แต่ละตัว

```
var fill;

fill = function(c, q) {
    if (c == null) {
        c = 1;
    }
    if (q == null) {
        q = "coffee";
    }
    return "Filling the " + c + " with " + q + "...";
};
```

โดยมีอัลกอริทึมดังต่อไปนี้

```
if(it.hasInitialExpression()){
    space()
    out "if("; out it.name; outln " == null){"
    space()
    space()
    final Expression initialExpr = it.getInitialExpression()
    final String typeName = initialExpr.type.name
    if (typeName == "java.lang.String") {
```

```
        out it.name; out "= \""; out initialExpr.text; outln "\";"
    } else {
        if (typeName == "java.util.Map") {
            out it.name; out "= "; outln "{};"
        } else {
            out it.name; out "= "; out initialExpr.text; outln ";"
        }
    }
    space()
    out "}"
}
```

### 3.10 การแปลงการวนรอบด้วย FOR

การวนรอบด้วย FOR ในลักษณะของภาษา Groovy จะเป็นการใช้ไวยากรณ์ตามรูปแบบ for-each นั่นคือจะใช้การประกาศตัวแปร และ คีย์เวิร์ด in สำหรับวนรอบค่าประเภท collection

```
class A {
  def hello = {i, v ->
    for(s in ['a','b','c']) {
      alert(s)
    }
    return null
  }
}
```

วนรอบ for-each เมื่อแปลงเป็นภาษา JavaScript จะมีลักษณะดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i,v) {
    var s, _i, _len, _ref;
    _ref = ['a', 'b', 'c'];
    for (_i = 0, _len = _ref.length; _i < _len; _i++) {
      s = _ref[_i];
      alert(s);
    }
    return null;
  };

  return A;

})();
```

โดยการวนรอบ for-each แปลงเป็นการวนรอบปกติโดยใช้อัลกอริทึมต่อไปนี้เป็นตัวแปลง ในกรณีที่ colExpr ไม่เป็น ClosureListExpression

```
outln "var ${argument}, _i, _len, _ref;"
sp(); out "_ref = "
colExpr.visit(this); outln ";"
sp(); outln "for (_i = 0, _len = _ref.length; _i < _len; _i++) {"
sp();sp(); outln "${argument} = _ref[_i];"
```

### 3.11 การแปลงการประกาศตัวแปรภายใน

ตัวแปรภายในหรือ local variable นั้นเป็นตัวแปรที่ประกาศเพื่อใช้ภายในฟังก์ชันหรือเมธอด ยกตัวอย่างเช่น

```
def i = 1
```

คือการประกาศ ตัวแปร i ขึ้นมาเพื่อใช้เฉพาะในฟังก์ชันหรือเมธอดที่กำหนดเท่านั้น

```
class MyClass {
    def func = {
        def i = 1
        return null
    }
}
```

โดยโปรแกรมภาษา Groovy ในตัวอย่างนี้จะสามารถแปลงเป็นโปรแกรมเป็นภาษา JavaScript ได้ดังต่อไปนี้ จะสังเกตได้ว่าการประกาศตัวแปรโดนแยกเป็น 2 ประโยคคือ var i; และ i = 1; เนื่องจากกลไกใน parser ของ Groovy ได้ทำการแยกความหมายของการประกาศไว้เรียบร้อยแล้ว

```
var MyClass;

MyClass = (function() {

  function MyClass() {}

  MyClass.prototype.func = function(it) {
    var i;
    i = 1;
    return null;
  };

  return MyClass;

})();
```

### 3.12 การแปลงโครงสร้างข้อมูลชนิด Map

ข้อมูลชนิด Map หรือ HashMap เป็นโครงสร้างที่ใช้เก็บข้อมูลในลักษณะ key และ value โดยเทียบได้กับ Dictionary ในภาษา JavaScript ยกตัวอย่างเช่นข้อมูล Map ต่อไปนี้

```
def singers = [Jagger: "Rock", Elvis:  "Roll"]
```

จะสามารถเปลี่ยนเป็นภาษา JavaScript ได้ดังนี้

```
var singers;

singers = {
  Jagger: "Rock",
  Elvis: "Roll"
};
```

โดยจะมีข้อจำกันคือค่า key จะต้องเป็น String เท่านั้น และมีกลไกการ visit ที่มีลักษณะเฉพาะกล่าวคือเป็น context-sensitive หมายถึงเป็นไวยากรณ์ขึ้นกับบริบท และใช้ KEY_EXPR เป็น guard โดยมี predicate อยู่ข้างใน

```
public void visitMapExpression(MapExpression e) {
    out " {"
    visitListOfExpressions(e.getMapEntryExpressions())
    outln()
    sp(); out "}"
}

public void visitMapEntryExpression(MapEntryExpression e) {
    indent()
    outln(); space()
    ctx.leftSide()
    ctx.enter(org.codehaus.groovejs.Context.KEY_EXPR)
    e.getKeyExpression().visit(this)
    out ": "
    ctx.rightSide()
    ctx.exit(org.codehaus.groovejs.Context.KEY_EXPR)
    e.getValueExpression().visit(this)
    dedent()
}
```

ซึ่ง predicate นั้นถูกใช้งานในขั้นตอนการเลือกประเภทของค่าคงที่ซึ่งกำกวมหากไม่มีการใช้ predicate

```
if(ctx.isIn(Context.KEY_EXPR)){
    out "${expr.value}"
} else if(ctx.isIn(Context.USE_UNDERSCORE)){
    out "_(${expr.value})"
} else if(expr.value instanceof String && !isPropertyName) {
    out("\"${expr.value}\"")
} else {
    out(expr.value)
}
```

### 3.13 การแปลงข้อมูลประเภทข้อความ (String)

ข้อความในภาษา Groovy นั้นสามารถเข้าถึงด้วยดัชนีในลักษณะคล้ายอะเรย์ได้ ในโปรแกรมต่อไปมีการเข้าถึงสมาชิกของอะเรย์ได้ด้วยตัวเลขค่าบวก และ ค่าลบ โดยเมื่อดัชนีเป็นค่าลบจะหมายถึงการเข้าถึงจากสมาชิกทางขวาสุด เช่น -1 คือตัวสุดท้าย และ -2 คือสมาชิกตัวรองสุดท้าย

```
class A {
  def hello = { i ->
    String s = "abcd"
    console.debug(s[0])
    console.debug(s[1])
    console.debug(s[-1])
    console.debug(s[-2])
  }
}
```

โปรแกรมภาษา JavaScript ที่แปลงได้จะมีลักษณะดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    var s;
    s = "abcd";
    console.debug(s.charAt(0));
    console.debug(s.charAt(1));
    console.debug(s.charAt(s.length-1));
    return console.debug(s.charAt(s.length-2));
  };

  return A;

})();
```

โดยจะเป็นการใช้เมธอด charAt แทนการใช้ String accessor และถ้าดัชนีเป็นค่าลบ จะหมายถึงการคำนวณโดยใช้ length เป็นตัวตั้ง

### 3.14 การเรียกใช้เมธอดที่รับพารามิเตอร์เป็น Closure

การเรียกใช้เมธอดที่รับพารามิเตอร์เป็น Closure มีลักษณะพิเศษคือสามารถใช้สร้างภาษาเฉพาะทาง ในลักษณะดังต่อไปนี้

```
3.times { n ->
  alert(n)
}
```

โดย Closure { n -> } ถูกส่งให้เป็นพารามิเตอร์ของเมธอด times() นั่นก็คือฟังก์ชัน function(n){ } ในภาษา JavaScript ซึ่งโปรแกรมภาษา Groovy จะแปลงเป็น JavaScript ได้ดังนี้

```
_(3).times(function(n){
  return alert(n);
});
```

ในตัวอย่างที่ 2 เป็นการแปลงการเรียกใช้เมธอด collect บน List

```
[1,2,3].collect { it * 2 }
```

และสามารถแปลงเป็น JavaScript ได้ดังนี้

```
_([1,2,3]).collect(function(it){
    return it *2;
});
```

ในตัวอย่างที่ 3 เป็นตัวอย่างการมีตัวแปรมารับค่าการเรียกใช้เมธอด collect

```
def formValues = $('.input').collect {
    it.value
}
```

ซึ่งสามารถแปลงเป็นภาษา JavaScript ได้ดังนี้

```
var formValues;

formValues = _($('.input')).collect(function(it) {
    return it.value;
});
```

โดยเมธอดที่รับพารามิเตอร์เป็นกลุ่มที่นำเข้ามาจาก Underscore library นั่นคือจะมีการใช้ predicate เลือกว่าเมธอดกลุ่มนี้อยู่ใน context ของ Underscore หรือไม่

```
if(ctx.isIn(Context.KEY_EXPR)){
    out "${expr.value}"
} else if(ctx.isIn(Context.USE_UNDERSCORE)){
    out "_(${expr.value})"
} else if(expr.value instanceof String && !isPropertyName) {
    out("\"${expr.value}\"")
} else {
    out(expr.value)
}
```

### 3.15 การแปลงคำสั่ง switch

คำสั่ง switch สามารถแปลงจากภาษา Groovy ไปเป็นภาษา JavaScript ได้แบบ 1:1 เนื่องจากมี Type System ในลักษณะกึ่งพลวัตรเหมือนกัน

โดยโปรแกรมในภาษา Groovy ต่อไปนี้

```
class A {
  def hello = { i ->
    switch(i) {
        case 10:
            console.debug("ten")
            break
        case 20:
            console.debug("twenty")
            break
        case true:
            console.debug("true")
            break
        case "hello":
            console.debug("hello")
            break
        default:
            console.debug("this is default")
    }
  }
}
```

สามารถเปลี่ยนเป็นโปรแกรมภาษา JavaScript ได้ดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    switch(i) {
        case 10:
            console.debug("ten")
            break
        case 20:
            console.debug("twenty")
            break
        case true:
            console.debug("true")
            break
        case "hello":
            console.debug("hello")
            break
        default:
            console.debug("this is default")
    }
  };

  return A;

})();
```

### 3.16 การแปลงประโยค try-catch

การแปลงประโยค try-catch จากภาษา Groovy ไปเป็นภาษา JavaScript สามารถทำได้โดยลดรูประบบ type ที่ใช้อยู่ในประโยค catch

```
class A {
  def hello = { i ->
    try {
      println "test"
    } catch(Exception e) {
      alert(e)
    }
  }
}
```

และสามารถแปลงเป็นภาษา JavaScript ได้ดังนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    try {
      println("test");
```

```
    } catch(e) {
      alert(e);
    }

  };

  return A;

})();
```

### 3.17 การแปลงการวนรอบแบบ while

การแปลงการวนรอบแบบ while จากภาษา Groovy ไปเป็น JavaScript มีลักษณะที่เทียบเท่ากัน กล่าวคือสามารถแปลงได้ในรูปแบบ 1:1 โดยแบ่งกรณีทดสอบออกเป็น 3 รูปแบบดังนี้

การแปลงการวนรอบแบบ while ที่ใช้ค่าคงที่

```
class A {
  def hello = {i ->
    while(true) {
      alert(i)
    }
    return null
  }
}
```

การใช้ค่าคงที่เป็นเงื่อนไขของการวนรอบจะเป็นรูปแบบที่ง่ายที่สุด โดยโปรแกรมภาษา Groovy ข้างต้นสามารถแปลงเป็นภาษา JavaScript ได้ดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    while(true) {
      alert(i);
    }
    return null;
  }

  return A;

})();
```

สำหรับกรณีทดสอบที่สองเป็นการแปลงการวนรอบแบบ while ที่ใช้ operator แบบ > หรือ <

```
class A {
  def hello = { i ->
    while(i > 10) {
      alert(i)
    }
  }
}
```

ซึ่งจะได้ผลลัพธ์ของการแปลงเป็นดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    while(i > 10) {
      alert(i);
    }
  };

  return A;

})();
```

ในกรณีที่ 3 การแปลงการวนรอบแบบ while ที่ใช้ operator แบบ !=

```
class A {
  def hello = { i ->
    int a = 1
    while(a != 10) {
      a = a + 1
    }
  }
}
```

จะได้ผลลัพธ์ในลักษณะเดียวกันในภาษา JavaScript ดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    var a;
    a = 1;
    while(a != 10) {
      a = a + 1;
    }
  };

  return A;

})();
```

### 3.18 การแปลงรูปแบบเงื่อนไข if

รูปแบบการแปลงเงื่อนไข if จากภาษา Groovy ไปเป็นภาษา JavaScript มีลักษณะเหมือนกันโดยทั่วไป แต่จะแตกต่างกันที่รูปแบบไวยากรณ์ภาษา Groovy จะพิจารณา if อยู่ในลักษณะกึ่งนิพจน์

นั่นคือ ถ้า if เป็นประโยคสุดท้ายในฟังก์ชันหรือเมธอดแล้วนั้น เราสามารถให้ if คืนค่าได้ ด้วยกฎพิเศษดังกล่าวทำให้สามารถแปลงจากภาษา Groovy ไปเป็นภาษา JavaScript ได้ดังนี้

โปรแกรมภาษา Groovy ที่มี if เป็นประโยคสุดท้าย และสามารถใช้แทนการคืนค่าได้

```
class A {
  def hello = { i ->
    if(i == 10) {
      true
    } else {
      false
    }
  }
}
```

โปรแกรมข้างต้นสามารถแปลงให้อยู่ในรูปแบบภาษา JavaScript ได้ดังต่อไปนี้

```
var A;

A = (function() {

  function A() {}

  A.prototype.hello = function(i) {
    if (i === 10) {
      return true;
    } else {
      return false;
    }
  };

  return A;

})();
```

โดยสามารถสังเกตได้ว่าประโยค if ในฟังก์ชันดังกล่าวมีการคืนค่า true / false ออกจากตัวฟังก์ชัน

### 3.19 การสนับสนุนการแปลงคลาสที่ใช้ในเฟรมเวิร์คอื่น ๆ

GroovyJS ออกแบบมาเพื่อสนับสนุนการสร้างโค้ดที่สามารถใช้งานได้กับเฟรมเวิร์คอื่น ๆ ของภาษา JavaScript โดยใช้แนวคิดของ annotation เข้ามาช่วย โดย annotation เป็นฟีเจอร์ในภาษา Java และ Groovy ที่ใช้เพื่อกำกับ metadata ลงบนโครงสร้างของภาษาโปรแกรม โดน annotation ที่ออกแบบขึ้นจะใช้กำกับคลาส

GroovyJS มี annotation สำหรับสร้างคลาสของ 3 เฟรมเวิร์ค

1. AngularJS
2. Backbone.JS
3. ASM.js

1. การแปลงโค้ดสำหรับเฟรมเวิร์ค AngularJS ทำได้โดยใช้ annotation ชื่อ groove.js.Angular กำกับคลาสภาษา Groovy ดังต่อไปนี้

```
@groove.js.Angular
class A {
    def hello = { ->
        println "Hello World"
    }
}
```

โดยจะได้การจำลองคลาสภาษา JavaScript ในลักษณะต่อไปนี้ ซึ่งเป็นรูปแบบคลาสที่จะสามารถใช้ได้ในเฟรมเวิร์ค AngularJS

```
var A;

A = function A($scope) {
    $scope.hello = function() {
        return println("Hello World");
    }
}
```

สำหรับการจำลองคลาสในเฟรมเวิร์ค AngularJS จะใช้ตัวแปร $scope แทนวัตถุในระบบ

2. การแปลงโค้ดสำหรับเฟรมเวิร์ค Backbone.JS สามารถทำได้โดยให้คลาสที่ต้องการสืบทอดจากคลาส groovy.js.Backbone.Model ดังนี้

```
class A extends groove.js.Backbone.Model {

}
```

โดยระบบคลาสของ Groove.JS นั้นจะใช้คำสั่ง extend() ของ Backbone.JS ในการจำลองกลไกการสืบทอดของคลาส และใช้ฟังก์ชัน initialize แทนการจำลองคอนสตรักเตอร์

```
(function () {
  'use strict';

  A = Backbone.Model.extend({
    initialize: function() { }
  });

}());
```

3. การแปลงโค้ดสำหรับเฟรมเวิร์ค ASM.js เป็นการแปลงโค้ดที่สนับสนุนระบบ type แบบ static โดยเราสามารถระบุ type ให้ตัวแปรและพารามิเตอร์ของฟังก์ชันได้ ซึ่งต่างกับระบบ type แบบพลวัตรที่ไม่ต้องการการระบุ type โดยข้อมูล type ที่ระบุให้ฟังก์ชันจะทำให้ฟังก์ชันทำงานได้เร็วขึ้น โดยเราสามารถกำกับคลาสเพื่อเพิ่มความเร็วของโปรแกรมที่สร้างขึ้นโดนใช้ annotation groovy.transform.CompileStatic

เมื่อ GrooveJS ตรวจสอบเจอ annotation จะทำการเปลี่ยนโหมดการทำงานให้สามารถตรวจสอบ type ได้ ตามตัวอย่างต่อไปนี้

```
@groovy.transform.CompileStatic
class A {
    int add(int i, int j) {
        return i + j
    }
}
```

และจะมีกลไกการ mask ตัวแปรเพื่อระบุว่าสามารถปรับปรุงประสิทธิภาพในเชิงตัวเลขได้ แทนการประมวลปกติซึ่งช้ากว่า โดย ASM.js นั้นเป็นฟีเจอร์พิเศษใน Mozilla Firefox

```
var A;

A = (function() {
    'use asm';

    function A() {}

    A.prototype.add = function(i, j) {
        i = i|0;
        j = j|0;
        return (i+j)|0;
    };

    return A;

})();
```

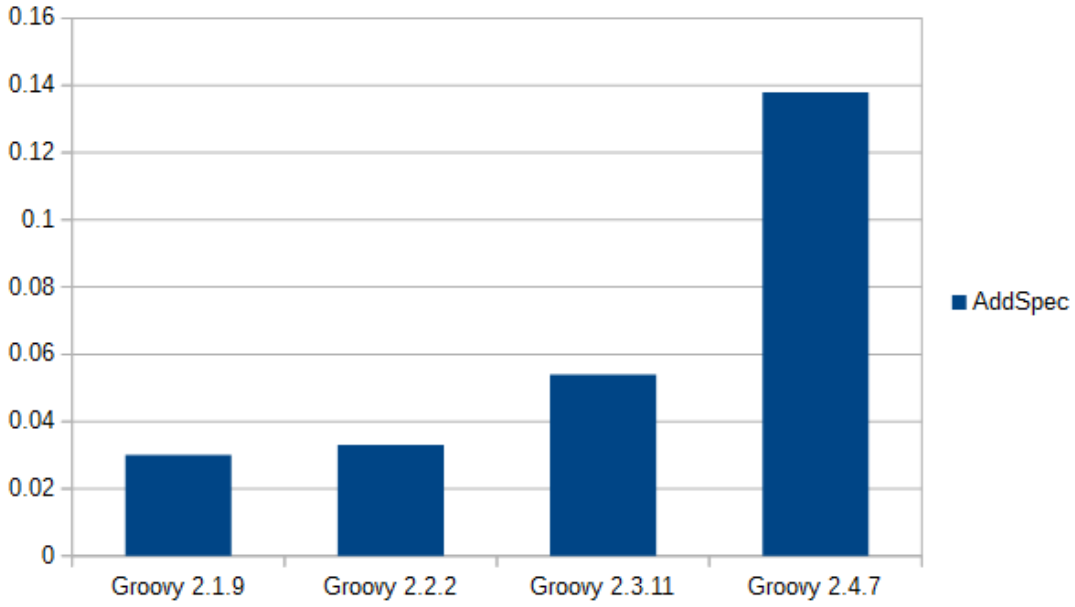
## 3.20 บทสรุป

ในบทที่ 3 นี้ได้นำเสนอกลไกการแปลงจากภาษา Groovy ไปเป็นภาษา JavaScript ในรูปแบบต่าง ๆ ตั้งแต่การสร้างคลาส คอนสตรักเชิงภาษาในรูปแบบต่าง ๆ รวมทั้งการแปลงคลาสสำหรับเฟรมเวิร์คต่าง ๆ เช่น AngularJS, Backbone.JS และ ASM.js

# บทที่ 4
## การทดสอบ

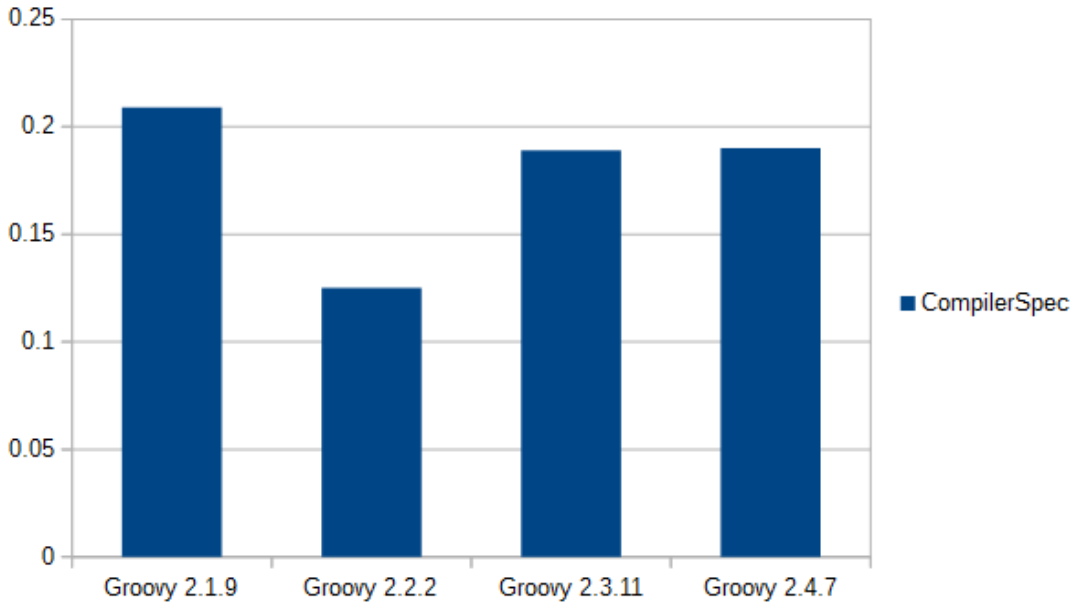
ในบทที่จะกล่าวถึงการทดสอบประสิทธิภาพของคอมไพเลอร์ GrooveJS ที่พัฒนาขึ้นโดนใช้เทคนิคแบบ predicate เพื่อลดความกำกวมในการสร้างโค้ดภาษา JavaScript โดยการทดสอบจะเป็นการวัดประสิทธิภาพของกรณีทดสอบทั้งหมด 8 กรณีทดสอบ บนระบบคอมไพเลอร์และรันไทม์ของ Groovy จำนวน 4 รุ่น ได้แก่ Groovy 2.1.9, Groovy 2.2.2, Groovy 2.3.11 และ Groovy 2.4.7 โดยทั้งหมดจะรันอยู่บน Java Virtual Machine 1.7.0_75 ชนิด Client

### 4.1 ผลการทดสอบกรณีทดสอบ AddSpec




กรณีทดสอบ AddSpec เป็นการพาร์สตัวกระทำบวก โดยพบว่า GrooveJS ทำงานได้เร็วที่สุดบน Groovy 2.1.9 และ Groovy 2.4.7 นั้นช้าที่สุดเนื่องจากโครงสร้างของรันไทม์มีความซับซ้อนมากขึ้น ในขณะที่ GrooveJS บน Groovy 2.2.2 ให้ค่าความเร็วใกล้เคียงกับ 2.1.9 เนื่องจากโครงสร้างภายในยังเปลี่ยนแปลงไปไม่มาก

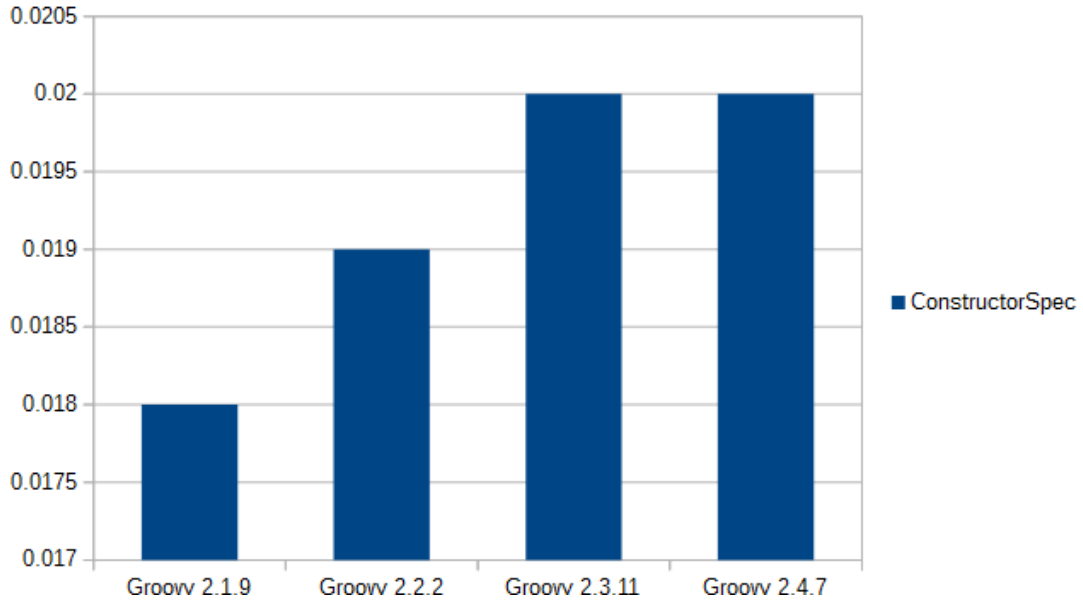
### 4.2 ผลการทดสอบกรณีทดสอบ CompilerSpec




กรณีทดสอบ CompilerSpec เป็นการทดสอบการสร้างโค้ดภาษา JavaScript จากคลาสที่มีลักษณะต่าง ๆ กัน โดยแต่ละคลาสมีเมธอดและจำนวนของพารามิเตอร์ไม่เท่ากัน พบว่า GrooveJS บน Groovy 2.2.2 นั้นมีความเร็วสูงสุดในขณะที่ GrooveJS บน Groovy 2.3.11 และ Groovy 2.4.7 มีความเร็วพอ ๆ กัน ส่วนการทำงานบน Groovy 2.19 นั้นช้าที่สุด

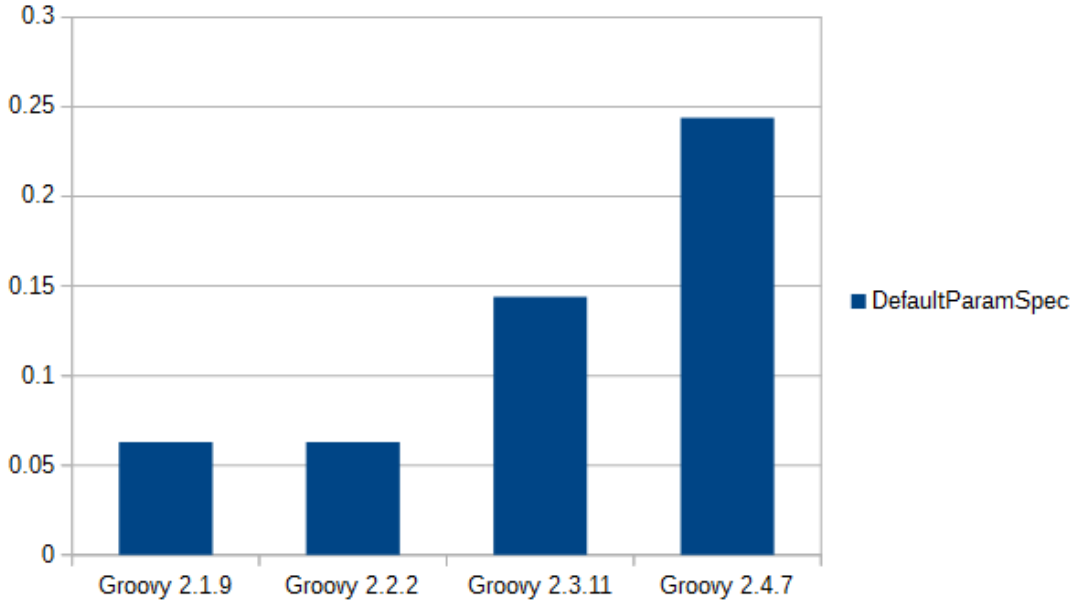
ความช้าบน Groovy 2.3.11 และ Groovy 2.4.7 นั้นชัดเจนว่ามาจากระบบคอมไพเลอร์และรันไทม์ที่ซับซ้อนขึ้น ในขณะที่ความช้าของ Groovy 2.1.9 มาจากระบบรันไทม์ที่ยังไม่ได้ปรับปรุงประสิทธิภาพอย่างเต็มที่

### 4.3 ผลการทดสอบกรณีทดสอบ ConstructorSpec




กรณีทดสอบ ConstructorSpec เป็นการทดสอบการสร้างโค้ดของคอนสตรักเตอร์ที่รับพารามิเตอร์ โดย GrooveJS บน Groovy 2.1.9 สามารถสร้างโค้ดของคอนสตรักเตอร์ได้เร็วที่สุด ในขณะที่ GrooveJS บน Groovy 2.3.11 และ 2.4.7 ทำงานได้ช้าที่สุดเท่ากัน

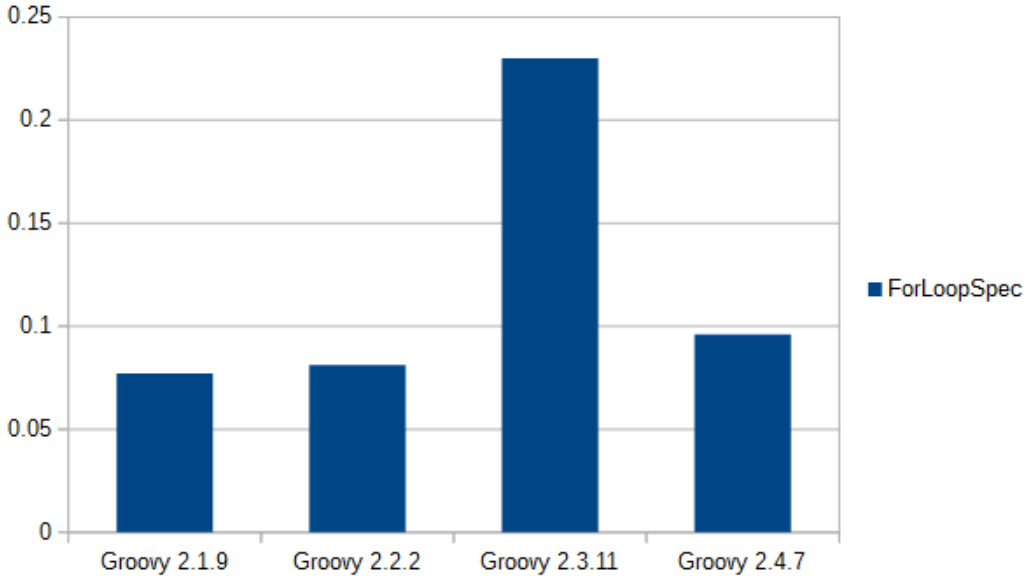
### 4.4 ผลการทดสอบกรณีทดสอบ DefaultParamSpec




กรณีทดสอบ DefaultParamSpec เป็นการทดสอบการสร้างโค้ดของฟังก์ชันที่รับพารามิเตอร์โดยปริยาย สำหรับกรณีทดสอบนี้ GrooveJS บน Groovy 2.1.9 และ Groovy 2.2.2

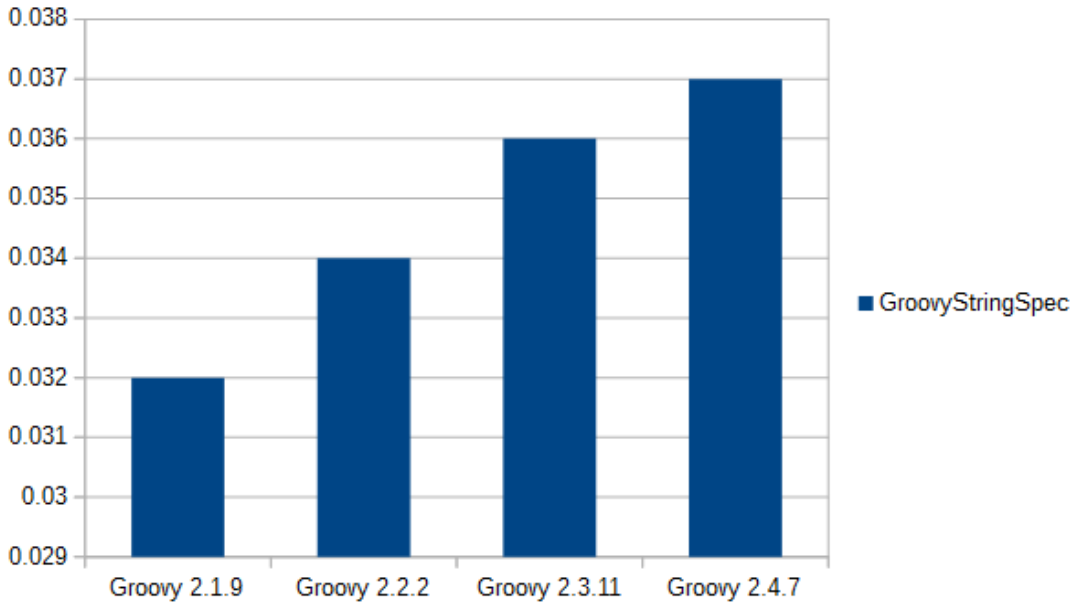
ทำงานได้เร็วที่สุดเท่ากัน สำหรับ GrooveJS บน Groovy 2.3.11 ใช้เวลาเกือบ 3 เท่า ส่วน GrooveJS บน Groovy 2.4.7 ใช้เวลาเกือบ 5 เท่าของเวลาที่เร็วที่สุด เหตุผลที่ช้ากว่าคือระบบรันไทม์ที่ซับซ้อนขึ้นของ Groovy 2.3 และ 2.4 ตามลำดับ

### 4.5 ผลการทดสอบกรณีทดสอบ ForLoopSpec



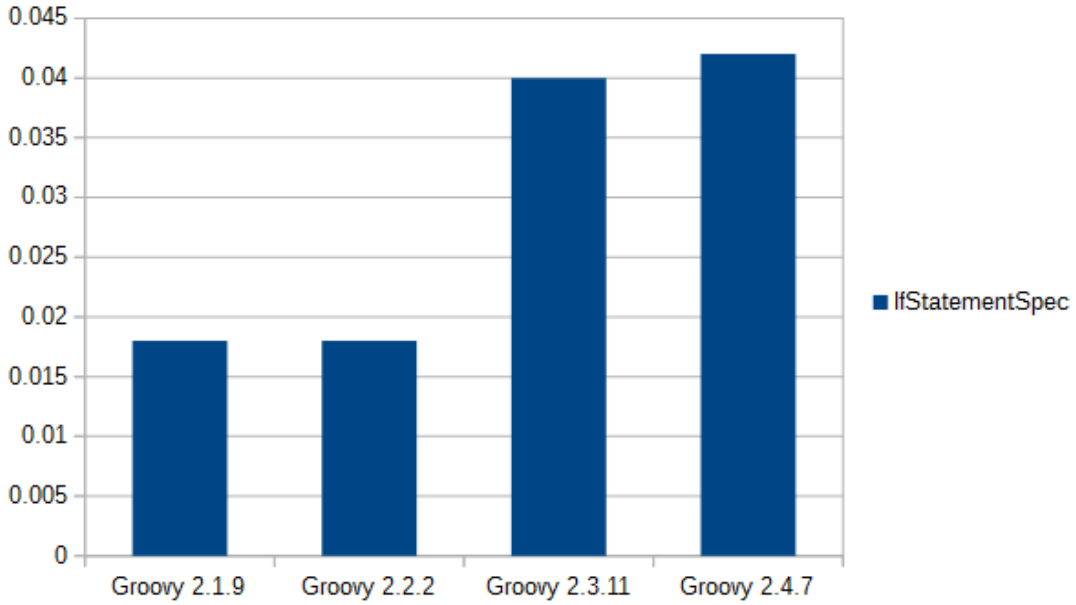

กรณีทดสอบ ForLoopSpec เป็นการสร้างโค้ดสำหรับการวนรอบที่ใช้ for แบบ for-each โดยเวลาของ GrooveJS ที่ทำงานบน Groovy 2.1.9 เร็วที่สุดและบน Groovy 2.2.2 ก็ใช้เวลาใกล้เคียงกัน สำหรับ GrooveJS บน Groovy 2.4.7 ช้ากว่าบนระบบที่ดีที่สุดเพียงเล็กน้อย ส่วนบน Groovy 2.3.11 ใช้เวลามากที่สุด

### 4.6 ผลการทดสอบกรณีทดสอบ GroovyStringSpec




กรณีทดสอบ GroovyStringSpec ทำงานได้เร็วที่สุดเมื่อทดสอบ GrooveJS บน Groovy 2.1.9 และช้าที่สุดบน Groovy 2.4.7 โดยประสิทธิภาพค่อย ๆ ช้าลงเมื่อเลขเวอร์ชันเพิ่มขึ้น แสดงให้เห็นว่าการสร้างโค้ดที่เกี่ยวข้องกับ String ทำงานได้ช้าในเวอร์ชันใหม่ ๆ ของ Groovy

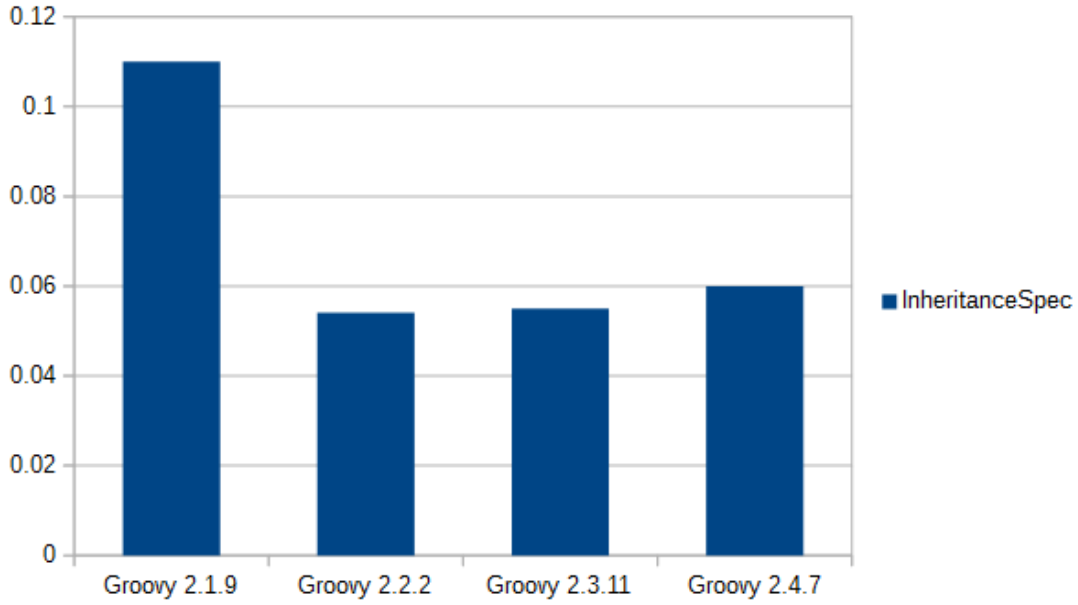
### 4.7 ผลการทดสอบกรณีทดสอบ IfStatementSpec




กรณีทดสอบ IfStatementSpec ทำงานได้เร็วที่สุดสำหรับ GrooveJS ที่รันบน Groovy 2.1.9 และ Groovy 2.2.2 โดยบน Groovy 2.3.11 และ 2.4.7 นั้นมีประสิทธิภาพใกล้เคียงกัน ความ

แตกต่างของประสิทธิภาพที่ชัดเจนนี้มาจากการเปลี่ยนระบบคอมไพเลอร์และระบบรันไทม์ของ Groovy 2.3 และ 2.4

### 4.8 ผลการทดสอบกรณีทดสอบ InheritanceSpec




กรณีทดสอบ InheritanceSpec ทำงานได้ช้าที่สุดด้วย GrooveJS ที่รันบน Groovy 2.1.9 สำหรับในเวอร์ชั่นอื่น ๆ มีความเร็วใกล้เคียงกัน โดยจุดนี้เป็นผลมาจากการเช็คเงื่อนไขการสืบทอดและใช้งาน String จำนวนมากในกลไกการสร้างโค้ดที่เกียวข้องกับการสืบทอด โดยแสดงให้เห็นว่า Groovy 2.1.9 ไม่ได้ optimize ในประเด็นนี้มากเมื่อเทียบกับรุ่นอื่น ๆ

### 4.9 สรุปผล

ในกรณีส่วนใหญ่ GrooveJS ที่ทำงานบน Groovy 2.1.9 และ 2.2.2 จะทำงานได้เร็วกว่า 2.3.11 และ 2.4.7 เนื่องจากระบบจัดการไทพ์ของคอมไพเลอร์และระบบรันไทม์มีความซับซ้อนน้อยกว่า อย่างไรก็ตามเมื่อเทียบระหว่าง Groovy 2.1.9 และ Groovy 2.2.2 จะเห็นได้ชัดเจนว่า Groovy 2.2.2 มีความเหมาะสมมากกว่าที่จะใช้เป็นระบบสำหรับตัวแปลภาษาของ GrooveJS ที่ใช้ predicate สำหรับบทถัดไปจะเป็นบทที่ 5 บทสรุปและอภิปรายงานในอนาคต

# บทที่ 5
# บทสรุปและข้อเสนอแนะ

## 5.1 สรุปผลการทดสอบ

การพัฒนา GroovyJS ใช้ตัวแจงส่วนแบบ LL(k) ของคอมไพเลอร์ภาษา Groovy ประกอบกับการใช้ตัวทำนาย (predicate) ทำให้ตัวแปลงภาษามีประสิทธิภาพและทนทานต่อความกำกวม เพื่อให้สามารถสร้างโค้ดภาษา JavaScript ได้ตามการออกแบบ

ในบทที่ 3 ได้ทำการอธิบายกฎการแปลงและตัวอย่างรวมถึงอัลกอริทึมที่เกี่ยวข้องและเทคนิคการตรวจสอบความถูกต้องโดยใช้การสร้างโค้ดของภาษา CoffeeScript [2] มาเป็นตัวอ้างอิง เพื่อให้สามารถพัฒนา GroovyJS ให้บรรลุวัตถุประสงค์ สำหรับบทที่ 4 ได้อภิปรายการทดสอบวัดประสิทธิภาพของระบบคอมไพเลอร์ GroovyJS บนฐานคอมไพเลอร์ภาษา Groovy [6] ที่แตกต่างกันจำนวน 4 เวอร์ชัน ได้แก่ Groovy 2.1.9, 2.2.2, 2.3.11 และ Groovy 2.4.7 โดยใช้กรณีทดสอบ 8 แบบเป็นตัววัดประสิทธิภาพ พบว่า Groovy 2.19 และ 2.2.2 จัดอยู่ในกลุ่มเร็ว สำหรับ Groovy 2.3.11 และ Groovy 2.4.7 จัดอยู่ในกลุ่มช้าเนื่องจากระบบการคอมไพล์ได้ถูกพัฒนาเพิ่มขึ้นจึงมีความซับซ้อนที่เพิ่มขึ้น

จากการทดสอบสรุปได้ว่าฐานคอมไพเลอร์ที่เหมาะสมสำหรับใช้สร้าง GroovyJS คือ Groovy 2.2.2 โดยมีความเร็วใกล้เคียงกับ Groovy 2.1.9 แต่ได้รับการแก้ไขข้อผิดพลาดต่อเนื่องมาจากรุ่น 2.1.9

## 5.2 ข้อเสนอแนะ

ภาษา JavaScript [3] ได้รับการพัฒนาอย่างต่อเนื่องทำให้เกิดภาษาใหม่ ๆ ที่เป็นซูเปอร์เซ็ตของภาษา JavaScript ที่ช่วยให้การพัฒนาเว็บแอพพลิเคชันเป็นไปได้สะดวกขึ้น การปรับปรุงให้ GroovyJS สามารถสนับสนุนภาษาที่เป็นซูเปอร์เซ็ตเหล่านั้นจะสามารถทำให้นำฟีเจอร์สำคัญ ๆ ของ Groovy มาใช้บนเว็บเบราเซอร์ได้สะดวกยิ่งขึ้น

เป็นที่น่าเสียดายที่การเปรียบเทียบ GroovyJS กับคอมไพเลอร์ Grooscript [5] ไม่ได้อยู่ในขอบเขตของงานวิจัยที่นำเสนอในรายงานฉบับนี้ จึงเป็นเรื่องที่น่าสนใจในการเปรียบเทียบเชิงประสิทธิภาพระหว่าง GroovyJS และ Grooscript ในอนาคต

การออบแบบภาษาด้วย ANTLR ให้มีประสิทธิภาพยังจำเป็นต้องศึกษากันต่อไปโดยเฉพาะการทำให้ ANTLR สามารถใช้กับ JavaScript ได้อย่างเต็มที่เป็นต้น ซึ่งจะยังเป็นประเด็นที่สำคัญสำหรับการค้นคว้าเพื่อสร้างคอมไพเลอร์ที่มีประสิทธิภาพสำหรับใช้งานบนเว็บเบราเซอร์ในอนาคต

# บรรณานุกรม


**[1]** 280 North Inc., **Cappuccino Web Framework - Objective-J Tutorial.** [Online]. Available: http://cappuccino.org/learn/tutorials/objective-j-tutorial.php. [Accessed: 15-Sep-2011].

**[2]** K. Arnold, J. Gosling and D. Holmes. **The Java Programming Language.** Boston: Addison-Wesley, 2000.

**[3]** T. Burnham, **CoffeeScript: Accelerated JavaScript Development**. Pragmatic Bookshelf, 2011.

**[4]** D. Flanagan, JavaScript: **The Definitive Guide: Activate Your Web Pages**. O'Reilly Media, Inc., 2011.

**[5]** J. Franco. **Grooscript documentation**. [Online]. Available: http://grooscript.org/doc.html. [Accessed: 30-Oct-2015].

**[6]** A. Gal et al., **Trace-based just-in-time type specialization for dynamic languages,** in ACM SIGPLAN Notices, 2009, vol. 44, pp. 465–478.

**[7]** A. Guha, C. Saftoiu, and S. Krishnamurthi, **The Essence of JavaScript,** in ECOOP 2010 – Object-Oriented Programming, vol. 6183, T. D'Hondt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 126-150.

**[8]** D. König, G. Laforge, P. King, J. Skeet, and H. D'Arcy, **Groovy in Action**, Second Edition (Early Access). Manning Publications, 2009.

**[9]** S. Maffeis, J. C. Mitchell, and A. Taly, **An Operational Semantics for JavaScript,** in Programming Languages and Systems, vol. 5356, G. Ramalingam, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 307-325.

**[10]** T. Parr, **The Definitive ANTLR 4 Reference**, Second Edition. Pragmatic Bookself, 2013.

**[11]** G. Richards, S. Lebresne, B. Burg, and J. Vitek, **An analysis of the dynamic behavior of JavaScript programs,** in ACM SIGPLAN Notices, 2010, vol. 45, pp. 1–12.

**[12]** N. Rozentals. **Mastering TypeScript.** Packt Publishing, April 2015.

**[13]** A. Tacy, R. Hanson, J. Essington, I. Bambury, and C. Ramsdale, **GWT in Action**, Second Edition. Manning Publications, 2011.

**[14]** D. Taft, **Groovy, JavaScript, Ruby Among Fastest Growing Programming Languages.** [Online]. Available: http://www.eweek.com/c/a/Application-Development/Groovy-JavaScript-Ruby-Among-Fastest-Growing-Programming-Languages-505803/. [Accessed: 15-Sep-2011].

## ประวัติผู้วิจัย

1. ชื่อ (ภาษาไทย) ดร. ชาญวิทย์ แก้วกสิ
   (ภาษาอังกฤษ) Dr. Chanwit Kaewkasi
2. เลขหมายบัตรประจำตัวประชาชน 3840100382000
3. ตำแหน่งปัจจุบัน ผู้ช่วยศาสตราจารย์
4. หน่วยงานที่อยู่ที่ติดต่อ
   สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์
   มหาวิทยาลัยเทคโนโลยีสุรนารี
   044-224224
   chanwit@sut.ac.th
5. ประวัติการศึกษา

| ปี | ระดับการศึกษา | อักษรย่อปริญญาและชื่อเต็ม | สาขาวิชา | สถาบัน |
|---|---|---|---|---|
| 2539-2542 | ปริญญาตรี | วศ.บ. / วิศวกรรมศาสตรบัณฑิต (เกียรตินิยมอันดับ 1) | วิศวกรรมคอมพิวเตอร์ | มหาวิทยาลัยเทคโนโลยีสุรนารี |
| 2544-2546 | ปริญญาโท | วศ.ม. / วิศวกรรมศาสตรมหาบัณฑิต | วิศวกรรมคอมพิวเตอร์ | จุฬาลงกรณ์มหาวิทยาลัย |
| 2549-2553 | ปริญญาเอก | Ph.D. / Computer Science | Computer Science | The University of Manchester, UK |

6. สาขาวิชาการที่มีความชำนาญพิเศษ
   เทคโนโลยีเชิงวัตถุ, เทคโนโลยีเชิงลักษณะ, วิศวกรรมซอฟต์แวร์
7. ประสบการณ์ที่เกี่ยวข้องกับการบริหารงานวิจัยทั้งภายในและภายนอกประเทศ :
   1. หัวหน้าโครงการวิจัย: โครงการวิจัยการออกแบบระบบคลัสเตอร์สำหรับกลุ่มเครื่องแม่ข่ายโปรแกรมประยุกต์ ทุนอุดหนุนการวิจัยเพื่อสนับสนุนการสร้างและพัฒนานักวิจัยรุ่นใหม่ ปี พ.ศ. 2547
   2. หัวหน้าโครงการวิจัย: โครงการวิจัยการออกแบบและพัฒนาเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ ทุนอุดหนุนการวิจัย มหาวิทยาลัยเทคโนโลยีสุรนารี ปี พ.ศ. 2548

3. ผู้ร่วมวิจัย: โครงการวิจัยการพัฒนาซอฟต์แวร์ตัวอย่างด้านความปลอดภัยอาหาร ทุนนวัตกรรมและสิ่งประดิษฐ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี ปีพ.ศ. 2547

8. งานวิจัยที่ดำเนินการเสร็จแล้ว :
    1. โครงการวิจัยการออกแบบระบบคลัสเตอร์สำหรับกลุ่มเครื่องแม่ข่ายโปรแกรมประยุกต์ ปีที่พิมพ์ 2548
       แหล่งทุน มหาวิทยาลัยเทคโนโลยีสุรนารี
    2. โครงการวิจัยการพัฒนาซอฟต์แวร์ตัวอย่างด้านความปลอดภัยอาหาร ปีที่พิมพ์ 2548
       แหล่งทุน กองทุนนวัตกรรมและสิ่งประดิษฐ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี มหาวิทยาลัยเทคโนโลยีสุรนารี
    3. โครงการวิจัยการออกแบบและพัฒนาเฟรมเวิร์คการเชื่อมต่อข้อมูลสำหรับลูกข่ายแบบบางของระบบกลุ่มแม่ข่ายโปรแกรมประยุกต์ ปีที่พิมพ์ 2553
       แหล่งทุน มหาวิทยาลัยเทคโนโลยีสุรนารี
    4. โครงการการพัฒนาซอฟตแวร์สำหรับจัดการและรายงานตำแหน่งพิกัดยานพาหนะที่สามารถปรับแต่งได้ด้วยภาษาเฉพาะทางภาษาไทย
       แหล่งทุน กองทุนนวัตกรรมและสิ่งประดิษฐ์ สมเด็จพระเทพรัตนราชสุดาฯ สยามบรมราชกุมารี มหาวิทยาลัยเทคโนโลยีสุรนารี

9. งานวิจัยระหว่างดำเนินการ:
    1. โครงการพัฒนาระบบปฏิบัติการเฉพาะทางสำหรับการศึกษาโครงสร้างผลึกโปรตีน สถานภาพ อยู่ระหว่างดำเนินการ คิดเป็นร้อยละ 90
       แหล่งทุน สำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ