บทที่ 3

การแบ่งคำ

ในการแปลงประโยคจากอีกภาษาหนึ่งให้เป็นอีกภาษาหนึ่ง ระบบหรือโปรแกรมที่ทำการแปลงประโยคต้องสามารถทำความเข้าใจกับประโยคนั้น ซึ่งการที่จะเข้าใจประโยคได้จะต้องสามารถแบ่งประโยคออกมาเป็นคำได้เสียก่อน ดังนั้นจะเห็นได้ว่าการแบ่งคำนั้นเป็นพื้นฐานของความเข้าใจในประโยค จึงจำเป็นที่จะต้องศึกษาและนำการแบ่งคำมาใช้

ภาษาไทยมีลักษณะการเขียนซึ่งเป็นเอกลักษณ์เฉพาะตัวผิดแผกจากภาษาตะวันตกอยู่หลายประการทั้งนี้เนื่องจากวัฒนธรรมอันเก่าแก่เป็นตัวของตัวเอง และจากการพัฒนาผ่านช่วงเวลาอันยาวนานลักษณะที่ว่าก็เช่นการมีตัวอักษรหลายระดับ และการเขียนไม่เว้นระยะระหว่างคำเป็นต้น ทั้งนี้จึงเป็นปัญหาที่นักคอมพิวเตอร์ต้องหาวิธีจัดการเพื่อที่จะพัฒนาคอมพิวเตอร์ซึ่งเป็นเทคโนโลยีโลกตะวันตกให้สามารถนำมาใช้งานกับข้อมูลภาษาไทย

มีงานวิจัยเกี่ยวกับภาษาไทยบนคอมพิวเตอร์อยู่หลายชิ้นตั้งแต่เมื่อเกือบสามสิบปีที่ผ่านมา ผลงานเหล่านี้ทำให้สามารถแก้ปัญหาภาษาไทยบนคอมพิวเตอร์ไปได้เป็นส่วนมาก ปัญหาบางเรื่องซึ่งก่อนนี้เคยเป็นเรื่องยุ่งยากก็กลายเป็นเรื่องปกติไป เช่น การจัดการกับตัวอักษรหลายระดับทั้งการแสดงผลทางจอภาพหรือเครื่องพิมพ์ และการรับข้อมูลจากแป้นพิมพ์ ปัจจุบันมีระบบซึ่งให้บริการงานพื้นฐานเหล่านี้อยู่บนเครื่องคอมพิวเตอร์แทบทุกระบบ

ปัญหาถัดมาคือ คำในประโยคภาษาไทยจะเขียนติดกันโดยไม่เว้นระยะระหว่างคำ เช่นภาษาตะวันตก การเขียนแบบนี้คงไม่มีปัญหาอะไรสำหรับคนอ่าน เพราะเคยชินและมีความสามารถที่จะรู้ขอบเขตของคำได้โดยไม่ลำบากอยู่แล้ว ซึ่งอันที่จริงแล้ว ก็ต้องยอมรับว่าแม้แต่มนุษย์บางครั้งเวลาอ่านหนังสือก็ยังมีการแบ่งคำพลาดได้บ้าง แต่ก็สามารถสามารถแก้ไขได้

อย่างรวดเร็วจนแทบไม่ทันสังเกต แต่เมื่อต้องการจะประมวลผลข้อความภาษาไทยเหล่านั้นด้วยคอมพิวเตอร์ จึงมีความจำเป็นที่จะต้องกำหนดวิธีการให้คอมพิวเตอร์มีความสามารถที่จะรู้ขอบเขตของคำได้เช่นเดียวกับมนุษย์ โดยที่ไม่จำเป็นต้องแบ่งคำให้กับระบบในระหว่างการพิมพ์ดังที่ยังคงเป็นอยู่ในหลายๆระบบ

พยางค์ และ คำ

ในงานวิจัยการแบ่งคำที่เกิดขึ้น เริ่มจากอัลกอริทึมที่ใช้กฎเป็นพื้นฐานนั้น หน่วยของสิ่งที่ได้มาจากอัลกอริทึมไม่ใช่คำ หากแต่เป็นสิ่งที่เรียกว่า พยางค์ เนื่องจากการตัดคำประเภทนี้เป็นสิ่งที่ก้ำกึ่งระหว่างพยางค์กับคำแต่ใกล้เคียงกับพยางค์มากกว่า ตัวอย่างเช่น เรียกคำว่า "นอน" ว่าเป็นหนึ่งพยางค์ และยังคงเรียกคำว่า "มหา" ว่าเป็นหนึ่งพยางค์ ทั้งๆที่ตามหลักไวยากรณ์แล้วไม่ใช่ เหตุที่ต้องทำเช่นนี้มาจากการใช้งานของอัลกอริทึมในชุดคำสั่งประมวลคำเนื่องจากคงไม่อยากเห็นคำดังกล่าวแยกออกเป็น "ม" กับ "หา" ไว้คนละบรรทัดอย่างแน่นอน

สำหรับ คำ นั้นจะใช้ตรงกับความหมายของคำในไวยากรณ์ไทย แต่เนื่องจากความหมายของคำในไวยากรณ์ไทยค่อนข้างจะไม่มีขอบเขต จึงต้องนิยามของคำให้เหมาะสมกับงานที่จะใช้ด้วย

ยุทธวิธีพื้นฐานของการแบ่งคำ

วิธีการแบ่งคำในปัจจุบันมีอยู่หลากหลายวิธี ซึ่งแต่ละวิธีมีจุดเด่นและจุดด้อยที่แตกต่างกันออกไปขึ้นอยู่กับวัตถุประสงค์ที่ต้องการใช้งานในแต่ละประเภท ส่วนการแบ่งคำภาษาไทยมักใช้กับงานด้านตัวประมวลผลคำเป็นส่วนใหญ่ โดยมีพื้นฐานจากแบ่งคำด้วยกฎ และการแบ่งคำด้วยพจนานุกรม ซึ่งมีลักษณะดังนี้

การแบ่งคำด้วยกฎ

แนวทางการแบ่งคำด้วยกฎเริ่มต้นจากทฤษฎีทางภาษาศาสตร์ และได้มีการพัฒนากฎที่ใช้ในการแบ่งคำให้มีเปอร์เซ็นต์ความถูกต้องตั้งแต่ 85% ในงานของ ยุพิน ไทยรัตนานนท์ (1981) ไปจนถึง 98% ในงานของ วิไล ธนประกอบ (1984) กฎที่ได้จากการพัฒนาเหล่านี้

ยังคงเป็นพื้นฐานของกฎที่ใช้กันอยู่ในปัจจุบัน สิ่งที่แตกต่างออกไปในงานชิ้นหลังๆจะเป็นในแง่ของวิธีการแทนกฎ การจัดการกับคำยกเว้น และวิธีการนำมาใช้

ลักษณะการทำงานของอัลกอริทึมแบ่งได้ออกเป็นสองลักษณะคือ การทำงานแบบขวาไปซ้าย และการทำงานแบบซ้ายไปขวา ซึ่งการแบ่งคำแบบซ้ายไปขวาเหมาะสำหรับการใช้ในงานที่เป็นลักษณะของการแบ่งข้อความออกเป็นคำๆ ในขณะที่การแบ่งคำแบบขวาไปซ้ายนั้นเหมาะสำหรับการตัดคำในชุดคำสั่งประมวลคำมากกว่า เพราะในกรณีหลังนั้นจะทำงานได้เร็วกว่าในกรณีแรกมาก เนื่องจากการที่อัลกอริทึมสามารถเริ่มต้นค้นหาจุดตัดจากขอบขวาที่ให้มาได้เลย ไม่ต้องไปเริ่มต้นค้นหามาจากต้นวรรคเหมือนการทำงานแบบซ้ายไปขวา

สำหรับวิธีการแทนกฎปกติมักแทนด้วยการเขียนลงไปเป็นตัวโปรแกรมเลย โดยนำเสนอวิธีการแทนกฎแบ่งพยางค์ด้วยโครงสร้างข้อมูลแบบต้นไม้ เพื่อที่จะสามารถแยกกฎเหล่านั้นออกจากตัวโปรแกรม ทำให้สามารถแก้ไขเพิ่มเติมและทดสอบได้สะดวก

<u>การแบ่งคำด้วยพจนานุกรม</u>

การแบ่งคำด้วยพจนานุกรมเริ่มต้นจากงานวิจัยของ ยืน ภู่วรวรรณ (2529) เป็นผลงานชิ้นแรกทำให้การแบ่งคำด้วยพจนานุกรมเป็นไปได้ โดยได้เสนออัลกอริทึมแบ่งคำด้วยพจนานุกรมแบบเปรียบเทียบกับคำที่ยาวที่สุด และค้นคว้าวิจัยหาโครงสร้างข้อมูลที่เหมาะสมสำหรับการแทนพจนานุกรมในหน่วยความจำหลัก โดยยึดหลักที่จะลดขนาดของพจนานุกรมลงให้มากที่สุดและสามารถค้นหาได้อย่างรวดเร็ว หลักการคือการแบ่งคำโดยการตรวจข้อความที่ได้กับพจนานุกรม ทำเครื่องหมายข้อความที่รวมเป็นคำได้แต่ยังไม่ยาวที่สุดเอาไว้ และเลือกแบ่งเป็นคำที่ยาวที่สุดแล้วกราดตรวจต่อ ทำเช่นนี้ไปจนกว่าจะไปต่อไม่ได้จึงทำการย้อนรอย (Backtrack) กลับไปยังจุดที่ทำเครื่องหมายไว้ล่าสุดแล้วทำการกราดตรวจเช่นเดิม ในกรณีที่พบกับคำที่ไม่ปรากฏในพจนานุกรม อัลกอริทึมจะอาศัยการแบ่งพยางค์ด้วยกฎในการค้นหาจุดที่เริ่มต้นแบ่งคำต่อไปได้แล้วจึงทำการกราดตรวจต่อไป

จุดประสงค์ของการหันมาใช้พจนานุกรมในการแบ่งคำในงานของ ยืน ภู่วรวรรณ (2529) คือเพื่อที่จะสามารถแบ่งคำได้อย่างรวดเร็วขึ้น ซึ่งผลจากการทดลองในงานวิจัยชิ้นดังกล่าวสรุปว่าอัลกอริทึมสามารถแบ่งคำได้อย่างรวดเร็ว ยกเว้นเวลาที่จำเป็นต้องส่วนแบ่งพยางค์ด้วยกฎซึ่งเกิดขึ้นน้อยกว่า 5% อีกทั้งยังมีความถูกต้องสูงขึ้นจนเป็นเปอร์เซ็นต์ความถูกต้องมากกว่า 99% โดยใช้พจนานุกรมซึ่งมีจำนวนคำ 5400 คำ ข้อเสียคือการเสียเนื้อที่

ในหน่วยความจำสำหรับเก็บพจนานุกรมไปประมาณ 50 กิโลไบต์ แต่ไม่น่าจะเป็นปัญหาเนื่องด้วยคอมพิวเตอร์ในปัจจุบันมีหน่วยความจำขนาดใหญ่

ถัดมาคือผลงานของ ดวงแก้ว สวามิภักดิ์ (2533) ซึ่งอาศัยอัลกอริทึมที่พัฒนามาจากผลงานข้างต้น แต่ใช้พจนานุกรมซึ่งอยู่ในหน่วยความจำสำรองโดยใช้โครงสร้างข้อมูลคือรูปต้นไม้แบบบี (B-tree) ผ่านทางซอฟต์แวร์ที่เป็นระบบการจัดการฐานข้อมูลซึ่งมีอยู่แล้วในระบบ แนวทางที่แตกต่างออกไปจากงานชิ้นแรกคือ เลือกที่จะแบ่งข้อความออกเป็นโทเค็น (Token) เสียก่อน โดยอาศัยตัวกราดตรวจที่เขียนด้วย Lex แล้วจึงใช้อัลกอริทึมแบ่งคำในการรวมโทเค็นเหล่านั้นเป็นคำเท่าที่ทำได้ โทเค็นที่เหลือซึ่งไม่สามารถรวมได้จะมาจากคำที่ไม่ปรากฏในพจนานุกรม พจนานุกรมที่ใช้มีขนาด 4500 คำ และในผลงานฉบับนี้ได้สรุปผลการทำงานว่าได้ความถูกต้อง 98.11% ตัวเลขที่ได้เป็นผลมาจากพจนานุกรมที่ใช้ ซึ่งถ้าเพิ่มคำที่เป็นปัญหาไปในพจนานุกรมทุกคำจะทำให้ความถูกต้องเพิ่มเป็น 100% ทันที จากเอกสารตัวอย่าง 12000 คำที่ใช้ในการวัดผล

นอกจากนี้ยังมีงานเชิงพาณิชย์ซึ่งมีการพัฒนาอัลกอริทึมแบ่งคำด้วยพจนานุกรมเป็นโปรแกรมใช้งานเกี่ยวกับตัวจัดพิมพ์ด้วยคอมพิวเตอร์แบบตั้งโต๊ะ (Desktop Publisher) และเป็นโปรแกรมที่เกี่ยวกับฐานข้อมูลข้อความ (Text Database) อีกด้วย ซึ่งแสดงถึงความสำเร็จของการใช้งานการแบ่งคำด้วยพจนานุกรมในทางปฏิบัติ อย่างไรก็ดีนั่นไม่ได้หมายความว่างานวิจัยการแบ่งคำด้วยพจนานุกรมจะสิ้นสุดลงแล้ว เนื่องจากยังมีจุดต่างๆให้พัฒนาต่อไปได้อีกมาก ทั้งในแง่โครงสร้างข้อมูลพจนานุกรมและตัวอัลกอริทึมรวมทั้งการจัดการกับความผิดพลาดต่างๆ สรุปก็คือยังต้องการที่จะแบ่งคำให้ได้เร็วขึ้น ถูกต้องมากขึ้น และใช้หน่วยความจำน้อยลงไปอีก

<u>ทฤษฎีการแบ่งคำด้วยพจนานุกรม</u>

การแบ่งคำที่ใช้กับงานวิเคราะห์ประโยคจะมุ่งเน้นเฉพาะการแบ่งคำด้วยพจนานุกรมเท่านั้น เนื่องจากสิ่งที่ต้องการได้จากการแบ่งคำคือ คำ ดังนั้นจึงต้องพิจารณาส่วนต่างๆของการแบ่งคำว่ามีลักษณะอย่างไร จะนำมาใช้ในงานวิเคราะห์ประโยคได้อย่างไร ซึ่งในส่วนนี้จะอธิบายถึงลักษณะต่างๆของการแบ่งคำด้วยพจนานุกรมไว้ดังนี้

พจนานุกรม วรรค และการแบ่งคำ

ในการแบ่งคำในที่นี้ได้พัฒนามาจากหลักการทางภาษาฟอร์มัล และตัวแปลชุดคำสั่งเพื่อใช้ในการนิยามและอธิบายการแบ่งคำได้อย่างชัดเจน โดยอาจจะอธิบายการแบ่งคำด้วยพจนานุกรมได้ดังนี้

ให้ Z เป็นเซ็ตของตัวอักษรไทย

$$Z = \{ก \ldots ฮ ะ า เ แ โ ไ \ldots\}$$

ให้ D เป็นเซ็ตของคำทั้งหมดในภาษาไทย เรียกอีกอย่างหนึ่งว่า พจนานุกรม ของคำในภาษาไทย

สมมติว่าคำในภาษาไทยมีอยู่เพียง 20 คำ คือ กก กร กล กลม กลับ ตา ตาก นม นา นารี มหา รก รี ลม ลับ หา หาก โค โคก โคน ดังนั้นเขียนเป็นนิพจน์ปกติ (Regular Expression) ได้ดังนี้

$$D = กก|กร|กล|กลม|กลับ|ตา|ตาก|นม|นา|นารี|มหา|รก|รี|ลม|ลับ|หา|หาก|โค|โคก|โคน \quad (1)$$

ให้

$$S = D^* \quad (2)$$

คำในภาษา S คือสายอักขระที่มีความยาวจำกัดซึ่งเกิดจากการนำคำในภาษา D มาต่อกัน โดยสามารถเลือกใช้คำใดกี่ครั้งก็ได้ หรือพูดอีกนัยหนึ่ง S คือเซ็ตของวรรคในภาษาไทย โดยให้นิยามว่าวรรคคือสายอักขระที่เกิดจากการนำคำในภาษาไทยมาเรียงต่อกันโดยไม่คำนึงถึงความหมาย และไม่มีตัวอักษรอื่นคั่น จะเห็นว่าวรรคนั่นเองที่เป็นสิ่งที่ใช้ในการนำไปแบ่งคำ ตัวอย่างเช่น ประโยค "โคนมหนองโพ" สิ่งที่ต้องการได้จากการแบ่งคำคือแยกวรรคนั้นออกมาเป็น (โค)(นม)(หนอง)(โพ)

เพราะฉะนั้นจากนิยามของ D และ S ข้างต้น สามารถสรุปได้ว่า การแบ่งคำคือการแยกตัวประกอบของสายอักขระในภาษา S ให้อยู่ในรูปของการแยกตัวประกอบที่อยู่ในภาษา D

การแยกตัวประกอบตัวอย่างแรกเป็น unique factoring ซึ่งต่างจากอีกตัวอย่างหนึ่ง ดังนี้ ให้ "ตากลม" มา คราวนี้จะเห็นว่าสามารถแยกตัวประกอบได้สองแบบคือ (ตา)(กลม) และ (ตาก)(ลม) ซึ่งเป็น non-unique factoring นั่นหมายความว่า S เป็นภาษาที่ไม่จำเป็นต้องได้ unique factoring เสมอไป ในการใช้งานทั่วไปกรณีเช่นนี้เกิดขึ้นน้อยมาก แต่เมื่อเกิดขึ้นก็ค่อนข้างจะเป็นปัญหาที่จะต้องจัดการต่อไป

ตัววิเคราะห์ศัพท์ (Lexical Analyzer)

สิ่งที่สรุปได้ในขั้นนี้คือปัญหาของการแบ่งคำเป็นปัญหาระดับนิพจน์ปกติเท่านั้น เพราะสามารถเขียนนิยามวรรคได้ในรูปของระดับนิพจน์ปกติ และการสร้างรูธีนแบ่งคำก็คือการสร้างตัววิเคราะห์ศัพท์สำหรับแบ่งวรรคภาษาไทยออกเป็นคำ

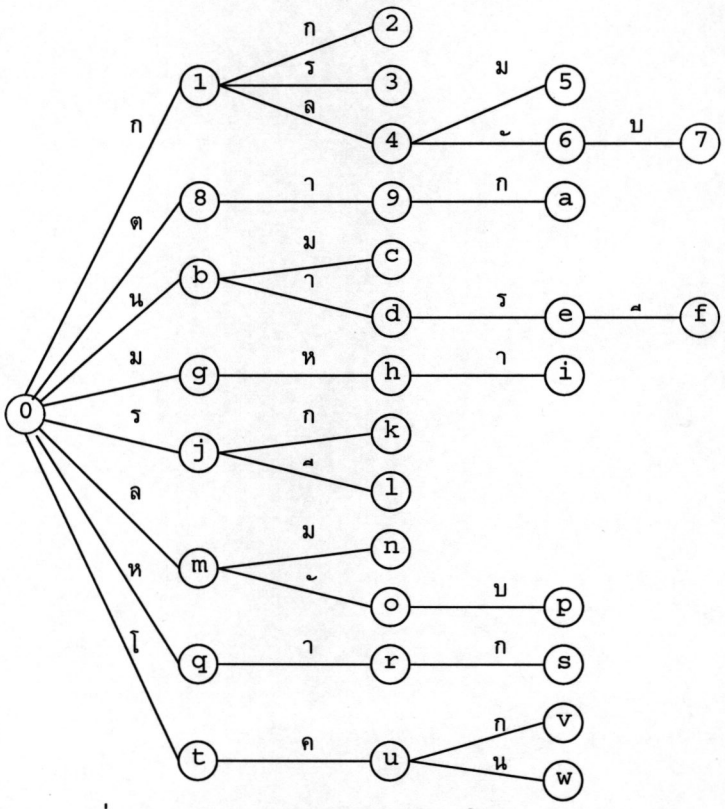
ถึงแม้สรุปว่ารูธีนแบ่งคำคือตัววิเคราะห์ศัพท์ แต่จะสรุปว่าคือสิ่งเดียวกันกับตัววิเคราะห์ศัพท์ที่ใช้ในตัวแปลภาษาที่ใช้กับภาษาคอมพิวเตอร์ต่างๆไม่ได้ สิ่งที่ต่างกันออกไปคือความซับซ้อนของการแบ่งคำ เนื่องด้วยในภาษาคอมพิวเตอร์นั้นโทเค็นจะมีอักขระคั่น (Delimiter) ที่แน่นอนเช่น ช่องว่าง หรือเครื่องหมายอื่นๆเสมอ แต่ในภาษาไทยไม่มี

ในบางกรณี ภาษาคอมพิวเตอร์จะประสบปัญหาความกำกวมเช่นเดียวกันกับภาษาไทย แต่เนื่องจากลักษณะเช่นนั้นคงไม่มีใครเขียนกันจึงถูกละเลยไปในการสร้างตัววิเคราะห์ศัพท์ เช่นในภาษา C ในนิพจน์ "+++++" ซึ่งผู้เขียนหมายถึง "++ + ++" แต่ตัววิเคราะห์ศัพท์จะถือว่าเป็น "++ ++ +" ตามหลักของ longest matching จากนั้นตัวแปลภาษาก็จะรายงานว่าผิดไวยากรณ์ ทั้งนี้เนื่องจากว่าตัววิเคราะห์ศัพท์ที่ใช้ในตัวแปลภาษานั้นถูกสร้างขึ้นเป็นกรณีพิเศษของระดับนิพจน์ปกติที่ออกแบบนิยามให้มีความกำกวมต่ำ และความกำกวมที่เกิดขึ้นก็สามารถตัดสินใจได้ด้วยหลักการง่ายๆ โดยไม่ต้องอาศัยการทำการย้อนรอย

ตัวรู้จำสถานะจำกัด (Finite State Recognizer)

ก่อนที่จะพิจารณาปัญหาการสร้างตัววิเคราะห์ศัพท์สำหรับภาษา S จะเริ่มต้นจากพิจารณาปัญหาที่ง่ายกว่าก่อน นั่นคือการสร้างตัวรู้จำ (Recognizer) สำหรับภาษา S และสิ่งที่กำลังพูดถึงคือ finite automaton (FA)

จากพจนานุกรม D ถ้านำพจน์ปกติจาก (1) มาแปลงเป็น deterministic finite automaton (DFA) จะได้ DFA ของ D ดังรูปที่ 3.1 ซึ่งมีอีกชื่อหนึ่งคือ trie

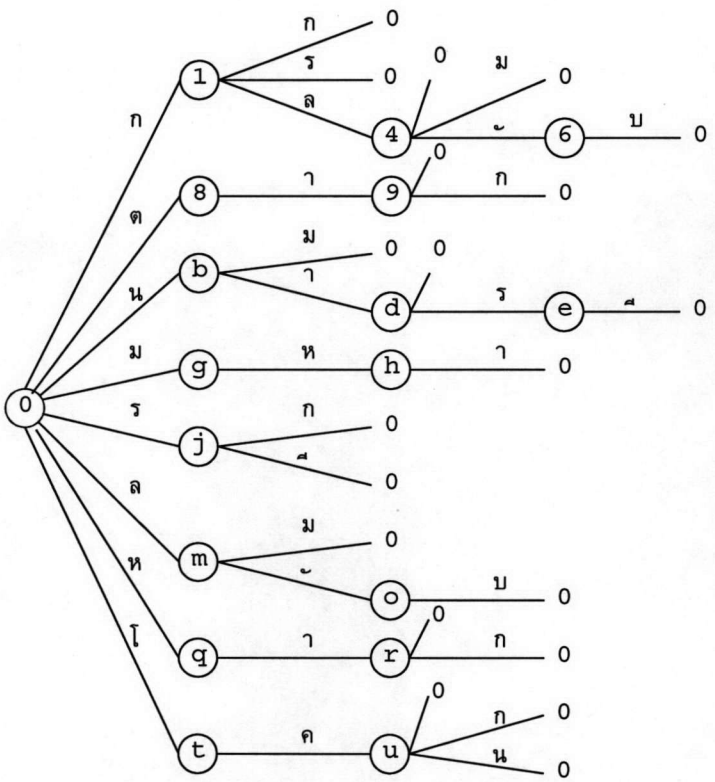


รูปที่ 3.1 DFA ของพจนานุกรม D ในรูปของ trie

คำในภาษา D คือคำในภาษาไทย ดังนั้น DFA นี้จึงเป็นตัวรู้จำสำหรับคำในภาษาไทย ลักษณะการทำงานของ DFA ตัวนี้คือเมื่อทำการป้อนสายอักขระเข้าไป จะสามารถบอกได้ว่าสายอักขระนั้นเป็นคำในพจนานุกรมหรือไม่ โดยดูจากสถานะสุดท้ายหลังจากการรับเข้ามาทีละตัวอักษรจากสายอักขระ โดยเริ่มต้นที่สถานะ 0 และเดินไปตามสถานะต่างๆตามตัวอักษรที่ได้รับ ถ้าสถานะสุดท้ายหลังจากจบสายอักขระเป็นหนึ่งในสถานะสุดท้าย แสดงว่าสายอักขระที่รับเข้ามาเป็นคำในภาษาไทย แต่ถ้าไม่ใช่สถานะสุดท้ายหรือเกิดการขัดข้องเสียก่อน แสดงว่าสายอักขระที่รับเข้ามาไม่ใช่คำในภาษาไทย

ปัญหาของการรู้จำคำดังที่กล่าวไว้ข้างต้น คือ สิ่งเดียวกับอัลกอริทึมค้นหาคำในพจนานุกรม ซึ่งเป็นงานของตัวตรวจสอบการสะกด (Spelling Checker) ที่มีอยู่หลายวิธีที่จะสร้างตัวรู้จำนี้นอกจากการใช้ trie

ต่อไปคือการสร้างตัวรู้จำสำหรับภาษา S โดยการแปลงพจน์ปกติจาก (2) ให้เป็น non-deterministic finite automation (NFA) จะได้ NFA ของ S ดังรูป 3.2



รูปที่ 3.2 NFA ของภาษา S

NFA ที่ได้ยังคงมีลักษณะคล้ายคลึงกับ DFA ของ D ต่างกันตรงที่ว่าสถานะสุดท้ายที่ไม่มีการผ่านต่อไปจะถูกยุบกลายเป็นสถานะ 0 และจะมีการผ่านแบบอี (e-transition) จากทุกสถานะสุดท้ายที่เหลือไปยังสถานะ 0 ลักษณะแบบนี้เกิดมาจากการนำคำมาต่อกันในการสร้างวรรค เนื่องจากเมื่อถึงสถานะสุดท้ายที่เป็นใบของรูปต้นไม้ มีเพียงทางเลือกที่จะจบวรรคลงที่ตรงนั้น หรือไม่ก็เริ่มต้นคำใหม่ แต่เมื่อถึงสถานะสุดท้ายที่ไม่ใช่ใบของรูปต้นไม้มีทางเลือกอยู่สามทางคือ จบวรรค เริ่มคำใหม่ หรืออยู่ที่เดิมรอตัวอักษรตัวต่อไปเพื่อประกอบเป็นคำที่ยาวขึ้น เช่น "กล" -> "กลม"

หน้าที่ของ NFA ของ S คล้ายคลึงกับ DFA ของ D คือเป็นตัวรู้จำที่ทำหน้าที่ตัดสินใจว่าสายอักขระที่รับเข้ามาอยู่ในภาษานั้นๆ หรือไม่ แต่การทำงานของตัวรู้จำตัวนี้แตกต่างจากตัวรู้จำของ D เนื่องจากเป็น NFA ไม่ใช่ DFA สาเหตุของความเป็น non-deterministic ของ FA ตัวนี้ก็เนื่องมาจากการมีการผ่านแบบอีนั่นเอง

ความยุ่งยากของการมีการผ่านแบบอีคือทำให้เกิดความจำเป็นที่จะต้องตัดสินใจเมื่อพบสถานะที่มีการผ่านแบบอีว่าจะไปตามการผ่านแบบอีหรือไม่และการตัดสินใจนี้มีความสำคัญในการรู้จำวรรคได้ถูกต้อง เมื่อต้องมีการตัดสินใจทำให้การเดินไปตาม NFA สามารถมีวิธีการเดินหรือเส้นทาง (Path) ได้หลายแบบ แต่ไม่จำเป็นว่าทุกเส้นทางจะประสบความสำเร็จตามทฤษฎีทางภาษา ซึ่งจะยอมรับสายอักขระหนึ่งด้วย NFA ก็ต่อเมื่อมีเส้นทางอย่างน้อยหนึ่งเส้นทางที่เรียกว่าเส้นทางสัมฤทธิ์ผล (Successful Path) จากสถานะเริ่มต้นไปยังสถานะสุดท้ายอันใดอันหนึ่ง ตัวอย่างเช่น จากวรรค "หากโคกล" เมื่อนำสายอักขระนี้มาใส่เข้าไปใน NFA ของ S จะมีเส้นทางที่เป็นไปได้อยู่สามเส้นทาง คือ

$$_0\underline{ห}_q\underline{า}_0\underline{ก}_1\underline{โ}_*$$
$$_0\underline{ห}_q\underline{า}_r\underline{ก}_0\underline{โ}_t\underline{ค}_0\underline{ก}_1\underline{ล}_0$$
$$_0\underline{ห}_q\underline{า}_r\underline{ก}_0\underline{โ}_t\underline{ค}_u\underline{ก}_0\underline{ล}_i$$

และเส้นทางที่เป็นเส้นทางสัมฤทธิ์ผลคือ เส้นทางที่สองเพราะจบลงที่สถานะสุดท้าย ในการสร้างเส้นทางสัมฤทธิ์ผลนี้ โดยผ่านจุดที่มีทางเลือกสองทางอยู่สองจุดคือ สถานะ q และสถานะ t เมื่อถึง สถานะ q และรับ "า" ซึ่งตัดสินใจที่จะเดินต่อไปยังสถานะ r เพื่อที่จะรวมกับตัวถัดมาเป็นคำว่า "หาก" แทนที่จะเป็น "หา" แต่เมื่อถึงสถานะ t และรับ "ค" จึงตัดสินใจที่จะกลับไปที่สถานะ 0 เลยทันที โดยถือเป็นคำว่า "โค" แทนที่จะปล่อยให้รวมกับ "ก" ตัวถัดมาเป็นคำว่า "โคก" เนื่องจากว่าตัวอักษร "ก" นั้นจริงๆ แล้วเป็นตัวอักษรตัวแรกของคำว่า "กล"

NFA กับการแบ่งคำ

เมื่อเสร็จเรื่องการรู้จำวรรคแล้วขั้นต่อไปคือการแบ่งวรรคออกเป็นคำๆ เนื่องจากเส้นทางสัมฤทธิ์ผลเป็นตัวบ่งบอกถึงการตัดสินใจที่จุดต่างๆระหว่างการทำการรู้จำ จึงแสดงถึงการแบ่งของวรรคด้วย เพราะฉะนั้นหน้าที่ของอัลกอริทึมแบ่งคำก็คือการรู้จำและเก็บเส้นทางสัมฤทธิ์ผลไว้ด้วยนั่นเอง สำหรับวรรค "หากโคกล" จะเห็นว่าสถานะสุดท้ายทำหน้าที่เดียวกับจุดแบ่งคำ ดังนั้นเส้นทางสัมฤทธิ์ผลนี้จะได้ผลของการแบ่งคือ (หาก)(โค)(กล) ส่วนในกรณีของวรรค "ตากลม" เส้นทางสัมฤทธิ์ผลทั้งสองคือจะเป็นตัวแทนของการแบ่งเป็น (ตา)(กลม) และ (ตาก)(ลม) ตามลำดับ

ที่ผ่านมาเป็นการวิเคราะห์ถึงการแบ่งคำเริ่มต้นด้วยนิยามของพจนานุกรมจนจบลงด้วยกลไกทางทฤษฎีที่ทำหน้าที่นี้คือ NFA ข้างต้น ขั้นถัดจากนี้ไปคือการสร้างอัลกอริทึมแบ่งคำด้วยพจนานุกรมซึ่งก็คือการจำลองการทำงานของ NFA ตัวนี้นั่นเอง

การค้นหาเส้นทางสัมฤทธิ์ผล

ดังที่ได้กล่าวมาแล้ว การเดินใน NFA จะมีเส้นทางที่เกิดขึ้นได้หลายเส้นทาง ซึ่งมาจากความเป็น non-deterministic และนำไปสู่ความยุ่งยากในการจำลองการทำงานของ NFA เมื่อวิเคราะห์ถึงลักษณะของการเกิดปัญหานี้ พบว่าปัญหานี้จะเกิดขึ้นก็ต่อเมื่อมีการใช้คำกำกวมจำนวนหนึ่ง ซึ่งแบ่งออกได้เป็น 2 ลักษณะคือ

1. การใช้คำที่มีคำนำหน้าก็เป็นคำด้วย เช่น "หาก" "การ" "ของ" "สามารถ"
2. การใช้คำที่สามารถเป็นคำนำหน้าของคำที่ยาวกว่าวางเรียงต่อกับอีกคำหนึ่งแล้วสามารถดูเป็นคำที่ยาวกว่านั้นได้ เช่น "โคนม" "ขอดม" "มารวม" "มีดาว" "มากว่า" "มากวาด" "ตากลับ"

ลักษณะแรกจะเกิดขึ้นบ่อยกว่าเพราะเกิดในคำที่ใช้ทั่วไป ในขณะที่กรณีที่สองมีโอกาสเกิดขึ้นค่อนข้างน้อย จุดที่สนใจคือกรณีเช่นนี้มีมากน้อยแค่ไหน และซับซ้อนแค่ไหน ทั้งนี้โดยพิจารณาถึงอัลกอริทึมแบ่งคำซึ่งทำงานจากซ้ายไปขวา

ถ้าลองมองการทำงานของอัลกอริทึมแบ่งคำจากซ้ายไปขวาอย่างคร่าว ๆจะสามารถแบ่งการทำงานได้ออกเป็นสองช่วงคือ เริ่มต้นด้วยเส้นทางเดียว อัลกอริทึมจะกราดตรวจสายอักขระไปเรื่อย ๆเป็นที่มีเส้นทางเดียว จนกระทั่งพบกับคำกำกวมซึ่งจะทำให้เข้าสู่ช่วงที่มีหลายเส้นทาง ขณะนี้อัลกอริทึมยังไม่รู้ว่าเส้นทางใดคือเส้นทางที่ถูกต้องเนื่องจากเพิ่งกราดตรวจมาถึงจุดนั้น เมื่ออัลกอริทึมกราดตรวจต่อไปจะทำให้เส้นทางที่ผิดขัดข้องไปทีละอัน จนกระทั่งความกำกวมหมดไปกลับไปเป็นช่วงเส้นทางเดียวอีกครั้ง วนสลับไปมาเช่นนี้

ความซับซ้อนของปัญหาความกำกวมเกิดขึ้นมาจากการที่ เมื่อคำกำกวมเกิดขึ้นในสายอักขระ บางครั้งจะมีผลต่อสายอักขระที่เหลือทางขวาไปได้อีกไกล และอาจทำให้เกิดการแตกเส้นทางต่อไปได้อีกเมื่อพบกับคำกำกวมต่อไปอีก

อัลกอริทึมแบ่งคำด้วยพจนานุกรม

อัลกอริทึมแบ่งคำด้วยพจนานุกรมจะทำการหาจุดแบ่งคำโดยใช้วิธีการสร้างต้นไม้แบบคั่น (Separation Tree) ซึ่งเป็นโครงสร้างข้อมูลแบบต้นไม้แบบทวิภาคที่มีอันดับ (Ordered Binary Tree) เพื่อแสดงการตัดสินใจเลือกจุดแบ่งคำทั้งหมดของข้อความที่ได้รับ อัลกอริทึมจะเริ่มทำงานโดยสร้างต้นไม้จากอักษรที่ได้รับทีละตัว หากถึงจุดที่สามารถแบ่งคำได้ก็จะเกิดการตัดสินใจว่าจะแบ่งคำที่จุดนั้นหรือไม่ ทำให้ต้องการสร้างต้นไม้ย่อย (Sub-Tree) ขึ้นมาสองต้น โดยต้นหนึ่งจะแทนการตัดสินใจว่าจะตัดคำ ส่วนอีกต้นหนึ่งจะแทนการไม่ตัดคำ เมื่อทำเช่นนี้ไปเรื่อยๆจนจบข้อความที่ต้องการตัดคำก็จะได้จุดแบ่งคำที่เป็นไปได้ทั้งหมด แต่จะพบว่าหากต้องทำการสร้างต้นไม้ทั้งหมดตามข้อมูลที่ได้มาจะเสียเวลาและสิ้นเปลืองโดยไม่จำเป็น เพราะมักละจุดที่ตัดสินใจแบ่งบางจุดเป็นการตัดสินใจที่ผิด เนื่องจากเมื่อเลยจุดนั้นไปแล้วไม่สามารถตัดคำที่เหลืออยู่ได้ ดังนั้น เพื่อประหยัดเวลาและหน่วยความจำที่ต้องใช้ในการทำงาน จึงต้องใช้อัลกอริทึมย้อนรอยในการสร้างต้นไม้แบบคั่น อัลกอริทึมจะทำงานโดยอ่านตัวอักษรเข้ามาทีละตัวจากซ้ายไปขวา แล้วทำการสร้างต้นไม้ เมื่อถึงจุดที่ต้องตัดสินใจว่าจะแบ่งคำหรือไม่ จากการทดลองและดูจากการแจกแจงความถี่ของคำในพจนานุกรม พบว่าหากเลือกที่จะไม่แบ่งจะมีโอกาสที่อัลกอริทึมจะหาจุดตัดคำได้ถูกต้องจะมีมากกว่าการเลือกที่จะตัดคำในจุดนี้ ดังนั้นอัลกอริทึมจะตัดสินใจที่จะไม่แบ่งก่อน แล้วจะทำการเรียกซ้ำ (Recursive) เพื่อทำการแบ่งคำส่วนที่เหลือต่อไป ถ้าการแบ่งคำส่วนที่เหลือประสบผลสำเร็จ แสดงว่าเป็นการตัดสินใจที่ถูกต้อง แต่ถ้าการแบ่งคำที่เหลือไม่ประสบผลสำเร็จ อัลกอริทึมจะกลับมาเลือกอีกทางหนึ่ง นั่นคือกลับมาแบ่งคำที่จุดนั้นนั่นเอง

การแบ่งคำด้วยพจนานุกรมกับการวิเคราะห์กระจายประโยค

สิ่งที่ตัววิเคราะห์ประโยคต้องการจากการแบ่งคำคือ การแบ่งคำที่อยู่ในประโยคออกมาเป็นคำที่มีความหมายของคำครบถ้วนอยู่ในคำนั้น โดยคำที่ได้ไม่ใช่เป็นส่วนประกอบของคำ ทั้งนี้เนื่องจากตัววิเคราะห์ประโยคไม่สามารถที่จะวิเคราะห์ส่วนประกอบของคำได้ สิ่งที่ตัววิเคราะห์จะสามารถทำได้คือ วิเคราะห์ส่วนประกอบของประโยคว่าประโยคที่เข้ามาถูกต้องตามกฎไวยากรณ์และความหมายหรือไม่เท่านั้น ดังนั้นหน้าที่ที่สำคัญของตัวแบ่งคำคือการหาจุดแบ่งคำในประโยคให้ถูกต้องโดยไม่มีคำใดเป็นส่วนประกอบของคำอื่นในประโยคให้ได้ การที่จะหาจุดแบ่งคำเช่นนั้น ตัวพจนานุกรมจะต้องบรรจุคำต่างๆที่ต้องใช้ไว้ทั้งหมด และจะต้องมีอัลกอริทึมในการที่จะตัดสินใจได้ว่าจุดใดจะเป็นจุดแบ่งคำที่เหมาะสมที่สุด เมื่อเกิดกรณีที่มีคำ

กำกวม หรือ non-unique factoring ฉะนั้นอัลกอริทึมดังที่กล่าวไว้ข้างต้นจะไม่เพียงพอแก่การทำการแบ่งคำเมื่อเกิดเหตุการณ์ดังกล่าว

แต่เนื่องจากในงานการค้นคืนข้อมูลด้วยประโยคนั้นจะมีคำกำกวม หรือคำที่เป็น non-unique factoring มีไม่มากนักหรือไม่มีเลย การแบ่งคำด้วยพจนานุกรมดังกล่าวจึงเพียงพอในระดับหนึ่ง อย่างไรก็ตามงานด้านแบ่งคำยังต้องมีการวิจัยเพิ่มขึ้นเพื่อให้ได้จุดแบ่งคำที่ถูกต้อง สำหรับใช้เป็นข้อมูลที่มีความถูกต้องต่องานวิเคราะห์กระจายประโยค