ข้อกำหนดภาษา Z สำหรับการแปลงเอกสารเอชทีเอ็มแอลเป็นเอกสารเอ๊กซ์เอ็มแอล

นาย ประยุทธ ลิมปกานนท์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2545
ISBN 974-17-3134-5

Z SPECIFICATIONS FOR HTML TO XML DOCUMENT CONVERSION

Mr. Prayouth Limpakanon

A Thesis Submitted in partial Fulfillment of the Requirements
for the Degree of Master of Science in Computational Science

Department of Mathematics
Faculty of Science
Chulalongkorn University

Academic Year 2545

ISBN 974-17-3134-5

หัวข้อวิทยานิพนธ์	ข้อกำหนดภาษา Z สำหรับการแปลงเอกสารเอชทีเอ็มแอลเป็นเอกสารเอ๊กซ์เอ็มแอล
โดย	นาย ประยุทธ ลิมปกานนท์
สาขาวิชา	วิทยาการคณนา
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ ดร. พีระพนธ์ โสพัศสถิตย์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิทยาศาสตร์
(รองศาสตราจารย์ ดร. วันชัย โพธิ์พิจิตร)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ศาสตราจารย์ ดร. ชิดชนก เหลือสินทรัพย์)

..... อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์ ดร. พีระพนธ์ โสพัศสถิตย์)

..... กรรมการ
(อาจารย์ ดร. กรุง สินอภิรมย์สราญ)

ประยุทธ ลิมปกานนท์ : ข้อกำหนดภาษา Z สำหรับการแปลงเอกสารเอชทีเอ็มแอลเป็นเอกสารเอ๊กซ์เอ็มแอล. (Z SPECIFICATIONS FOR HTML TO XML DOCUMENT CONVERSION) อ. ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร. พีระพนธ์ โสพัศสถิตย์, จำนวนหน้า 62 หน้า. ISBN 974-17-3134-5.

ในปัจจุบันการแลกเปลี่ยนข้อมูลระหว่างองค์กรหรือหน่วยงานผ่านเครือข่ายคอมพิวเตอร์มีจำนวนเพิ่มขึ้นอย่างมาก เอกสาร HTML ที่ใช้กันอยู่ไม่สามารถตอบสนองต่อความต้องการในเรื่องการแลกเปลี่ยนข้อมูลได้อย่างสมบูรณ์เพราะ HTML เป็นภาษาที่เน้นการแสดงผลเป็นหลัก ไม่สามารถแยกข้อมูลออกมาเป็นโครงสร้างเพื่องานประเภทอื่นได้อย่างง่ายๆ ในขณะที่ XML เป็นภาษาที่เน้นเรื่องโครงสร้างของข้อมูล สามารถสร้าง tag เพิ่มและอธิบายความหมายได้ด้วยตัวเอง จึงทำให้ภาษา XML มีความยืดหยุ่นและสามารถนำไปใช้ในการติดต่อกับ application อื่นๆได้มากมาย งานวิทยานิพนธ์นี้จึงเป็นการกำหนดแนวทางเชิงรูปนัยของการแปลงเอกสาร HTML เป็นเอกสาร XML เพื่อตอบสนองต่อความต้องการข้างต้นที่จะเกิดขึ้นในอนาคต

เนื่องจากโปรแกรมที่พัฒนากันทั่วไปมักเกิดจากการลองผิดลองถูก จึงทำให้พบความผิดพลาด (bug)ขณะโปรแกรมทำงานอยู่เสมอ วิธีเชิงรูปนัยไม่เริ่มจากการลองผิดลองถูก แต่จะสร้างข้อกำหนดของกระบวนการดังกล่าวขึ้นมาก่อนด้วยภาษา Z เพราะเป็นภาษาที่มีระเบียบวิธีเชิงรูปนัยบนฐานทางคณิตศาสตร์ ทำให้โปรแกรมที่สร้างตามระเบียบวิธีที่ได้มีความถูกต้อง ชัดเจน และสะดวกต่อการเปลี่ยนแปลงแก้ไขในอนาคต

ภาควิชา คณิตศาสตร์ ลายมือชื่อนิสิต.....

สาขาวิชา วิทยาการคณนา ลายมือชื่ออาจารย์ที่ปรึกษา.....

ปีการศึกษา 2545

437 23246 23 : MAJOR COMPUTATIONAL SCIENCE
KEY WORD: HTML / XML / CONVERSION / SPECIFICATIONS / Z

PRAYOUTH LIMPAKANON : Z SPECIFICATIONS FOR HTML TO XML DOCUMENT CONVERSION. THESIS ADVISOR : ASST. PROF. PERAPHON SOPHATSATHIT, Ph. D., 62 pp. ISBN 974-17-3134-5.

Nowadays data interchange among organizations via computer network increases considerably. Use of HTML documents alone does not entirely serve the needs for data interchange in that HTML merely displays the outcome or contents of the document. It is unable to extract the structure of the HTML document for subsequence usage. XML, on the other hand, focuses on the structure of data in the document which allows new tags to be added for self-descriptive documentation. This renders XML to be highly flexible and adaptable to other applications. This research will concentrate on establishing a formal guideline for HTML to XML document transformation so as to fulfill the above requirements in the future.

The fact that computer programs are generally developed on trial and error basis makes program bugs inevitable as the programs are being executed. Formal approach, on the contrary, will employ Z specification, a formal language based on mathematics, that yields correct, workable, and modifiable output to suit any future change.

Department Mathematics
Field of study Computational Science
Academic year 2002

Student's signature .....
Advisor's signature .....

กิตติกรรมประกาศ

ผู้วิจัยขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร. พีระพนธ์ โสพัศสถิตย์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่กรุณาให้ความรู้และแนวทางในการทำวิจัยจนลุล่วงด้วยดี ตั้งแต่เริ่มแรกจนกระทั่งวิทยานิพนธ์เสร็จสมบูรณ์ รวมทั้งขอขอบพระคุณ ศาสตราจารย์ ดร. ชิดชนก เหลือสินทรัพย์ ประธานกรรมการสอบวิทยานิพนธ์ และ อาจารย์ ดร. กรุง สินอภิรมย์สราญ กรรมการสอบวิทยานิพนธ์ สำหรับคำแนะนำในการแก้ไขวิทยานิพนธ์ซึ่งทำให้วิทยานิพนธ์มีความสมบูรณ์ยิ่งขึ้น

ขอขอบคุณ ศูนย์วิจัย AVIC และ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ที่ได้อนุเคราะห์อุปกรณ์สำหรับใช้ในงานวิจัย

สุดท้ายขอขอบพระคุณ บิดา มารดา รวมถึงทุกคนในครอบครัวสำหรับกำลังใจที่มีให้ผู้วิจัยตลอดมา

สารบัญ

หน้า

หน้า

ตาราง หน้า

ภาพประกอบ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการแลกเปลี่ยนข้อมูลระหว่างองค์กรหรือหน่วยงานผ่านเครือข่าย WWW มีจำนวนเพิ่มขึ้นอย่างมาก เอกสาร HTML ที่ใช้กันอยู่ไม่สามารถตอบสนองต่อความต้องการในเรื่องการแลกเปลี่ยนข้อมูลได้อย่างสมบูรณ์เพราะ HTML เป็นภาษาที่เน้นการแสดงผลเป็นหลัก ไม่สามารถแยกข้อมูลออกมาเป็นโครงสร้างเพื่องานประเภทอื่นได้อย่างง่าย ในขณะที่ XML เป็นภาษาที่เน้นเรื่องโครงสร้างของข้อมูล สามารถสร้าง tag เพิ่มและอธิบายความหมายได้ด้วยตัวเอง จึงทำให้ภาษา XML มีความยืดหยุ่นและสามารถนำไปใช้ในการติดต่อกับ application อื่นๆได้มากมาย ดังนั้นจึงมีแนวคิดที่จะสร้างโปรแกรมแปลงเอกสาร HTML เป็นเอกสาร XML เพื่อตอบสนองต่อความต้องการที่อาจเกิดขึ้นในอนาคต

เนื่องจากโปรแกรมที่สร้างขึ้นโดยทั่วไปมักสร้างขึ้นจากการลองผิดลองถูก จึงทำให้พบความผิดพลาดขณะโปรแกรมทำงาน (bug) อยู่เสมอ แต่โปรแกรมที่เราจะสร้างขึ้นนี้จะไม่สร้างขึ้นจากการลองผิดลองถูกแต่จะสร้างข้อกำหนด (specifications) ก่อนด้วยภาษา Z เพราะเป็นภาษาที่มีระเบียบวิธีเชิงรูปนัยบนฐานทางคณิตศาสตร์ ทำให้โปรแกรมที่สร้างตามระเบียบวิธีที่ได้มีความถูกต้อง ชัดเจน และสะดวกต่อการเปลี่ยนแปลงแก้ไขในอนาคต

1.2 วัตถุประสงค์ของการวิจัย

วัตถุประสงค์สำหรับงานวิจัยประกอบด้วยวัตถุประสงค์หลักๆดังนี้

1.2.1 แสดงข้อกำหนดเชิงรูปนัยของการแปลงเอกสาร HTML เป็นเอกสาร XML โดยใช้วิธีการและภาษาทางคณิตศาสตร์

1.2.2 พัฒนาระเบียบวิธีการโปรแกรมเพื่อแปลงเอกสาร HTML เป็นเอกสาร XML ซึ่งจะให้ความถูกต้อง ชัดเจนในรายละเอียดจากข้อกำหนดที่สร้างในข้อที่ (1.2.1)

1.3 ขอบเขตของการวิจัย

ขอบเขตของการวิจัยการแปลงเอกสาร HTML เป็นเอกสาร XML มีดังนี้

1.3.1 เอกสาร HTML ที่จะนำใช้เป็นข้อมูลเข้า (input) ต้องเป็นเอกสารประเภท well-formed document ตามมาตรฐานของ HTML 2.0 [5]

1.3.2 ใช้ภาษา Z ในการอธิบายวิธีการแปลงเอกสาร HTML เป็นเอกสาร XML

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

เอกสาร HTML ที่ใช้แสดงผลบน web ในทุกวันนี้มีการใช้ปะปนกันระหว่าง tag ของ HTML (tag เปิดซึ่งได้แก่ข้อความที่ถูกปิดล้อมด้วยเครื่องหมาย ‘<’ และ ‘>’ หากมีข้อความต่อท้ายข้อความที่ถูกปิดล้อมด้วยเครื่องหมาย ‘<’ และ ‘>’ เราเรียกข้อความนั้นว่า attribute และ tag ปิดซึ่งได้แก่ข้อความที่ถูกปิดล้อมด้วยเครื่องหมาย ‘</’ และ ‘>’) และข้อมูล (ข้อความที่ปิดล้อมด้วย tag เปิดและ tag ปิด) เช่น <A HREF= “www.spa.com”>Spa Guide</A> เราเรียก <A HREF= “www.spa.com” > ว่าเป็น tag เปิด โดยมี HREF= “www.spa.com” เป็น attribute, Spa Guide เป็นข้อมูลและ </A> เป็น tag ปิด จึงทำให้ไม่สะดวกหากต้องการนำข้อมูลไปใช้ในการทำงานโดยเฉพาะในกรณีที่เอกสาร HTML มีขนาดใหญ่จึงมีแนวความคิดที่จะดึงข้อมูลของเอกสาร HTML ออกมาโดยยังคงรักษาโครงสร้างของเอกสารเดิมเอาไว้และกำจัด tag ที่อยู่ในเอกสารทิ้งไปเพราะว่า tag ของเอกสาร HTML ไม่ใช่ข้อมูลแต่ใช้สำหรับช่วยในการแสดงผลของข้อมูลที่ถูก tag นั้นล้อมอยู่ เช่น tag <B> ทำให้ข้อความที่ถูกปิดล้อมมีตัวอักษรหนา, tag <I> ทำให้ข้อความที่ถูกปิดล้อมมีตัวอักษรเอียง, tag <U> จะขีดเส้นใต้ข้อความที่ถูกปิดล้อมเป็นต้น และเนื่องจากในปัจจุบันเอกสาร XML เป็นที่นิยมใช้กันมากขึ้นเพราะเป็นมาตรฐานของการแลกเปลี่ยนข้อมูล รวมทั้งเทคโนโลยีใหม่ๆหลายชนิดตั้งอยู่บนพื้นฐานของ XML ดังนั้นจึงมีแนวความคิดว่าเมื่อดึงข้อมูลจากเอกสาร HTML เสร็จแล้วจะแปลงเป็นเอกสาร XML เพื่อความสะดวกต่อการนำไปใช้

แนวคิดที่จะนำมาใช้ในการแปลงเอกสาร HTML เป็นเอกสาร XML จะประยุกต์จากงานวิจัย [6] ซึ่งเริ่มจากการแปลงเอกสาร HTML ให้อยู่ในรูปของ tree โดย tree ที่ได้ในขั้นต้นนี้เราเรียกว่า Document Hierarchy (DOH) ซึ่ง DOH ที่ได้ยังคงรักษาโครงสร้างของเอกสาร HTML ต่อมาเราจะแปลง DOH ให้อยู่ในรูป tree ใหม่ซึ่งเราเรียกว่า Data Hierarchy (DAH) ซึ่งในขั้นตอนนี้จะมีการกำจัด tag ของ HTML ออกไปเนื่องจาก tag ดังกล่าวไม่ใช่ข้อมูลที่จะนำมาใช้ในการสร้างเอกสาร XML แต่เลือกเก็บเฉพาะข้อมูลและ attribute บางตัวเพื่อจะนำมาใช้เป็นข้อมูลในการสร้างเอกสาร XML ในขั้นตอนนี้จะมีการเปลี่ยนโครงสร้างบางส่วนของ DOH ด้วยและเมื่อผ่านขั้นตอนนี้โดยสมบูรณ์ tree ที่ได้จะไม่มี tag ของ HTML เหลืออยู่และในขั้นตอนสุดท้ายจึงนำ DAH ที่ได้มาสร้างเป็นเอกสาร XML

ความเป็นจริงจากแนวคิดดังกล่าวก็สามารถพัฒนาโปรแกรมในการแปลงเอกสาร HTML เป็นเอกสาร XML ได้แล้วแต่ในงานวิจัยนี้ต้องการจะสร้างข้อกำหนดของการแปลงนี้ทั้งหมดด้วยภาษา Z ก่อนที่จะนำไปพัฒนาโปรแกรมด้วยภาษาซีเนื่องจากภาษา Z เป็นภาษาที่ใช้เซต ฟังก์ชัน และตรรกะในการอธิบาย

1.3.3 พัฒนาระเบียบวิธีการโปรแกรมสำหรับการแปลงเอกสาร HTML เป็นเอกสาร XML

1.4 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้จากงานวิจัยชิ้นนี้คือ

1.4.1 ระเบียบวิธีการแปลงเอกสาร HTML เป็นเอกสาร XML ซึ่งมีข้อกำหนดวิธีการแปลงเอกสารเชิงรูปนัย มีความชัดเจน เป็นมาตรฐาน และมีความถูกต้องเชื่อถือได้

1.4.2 สามารถประยุกต์ระเบียบวิธีเชิงรูปนัยในการพัฒนาโปรแกรมลักษณะอื่นได้

1.5 วิธีดำเนินการวิจัย

สร้างโปรแกรมแยกโครงสร้างของเอกสาร HTML เป็นเอกสาร XML โดยมีรายละเอียดและขั้นตอนดังนี้

1.5.1 ศึกษาเกี่ยวกับภาษา Z

ขั้นตอนนี้เป็นการศึกษาและทำความเข้าใจเกี่ยวกับการใช้สัญลักษณ์ ข้อกำหนดและเกณฑ์ต่างๆในภาษา Z เพื่อใช้ในการนิยามระบบงานเชิงรูปนัย

1.5.2 กำหนดกฎสำหรับการแปลงและข้อกำหนด

ขั้นตอนนี้เป็นการแปลงเอกสาร HTML เป็นเอกสาร XML โดยการใช้โครงสร้างแบบ DOH และ DAH ดังรายละเอียดในงานวิจัย [6] พร้อมทั้งอธิบายการทำงานด้วยภาษา Z

1.5.3 พัฒนาโปรแกรม

ขั้นตอนนี้จะเป็นการนำข้อกำหนดที่ได้จากข้อที่ 1.5.2 มาพัฒนาเป็นโปรแกรมแยกโครงสร้างของเอกสาร HTML เพื่อสร้างเป็นเอกสาร XML โดยใช้ภาษาซี

ระบบงาน ดังนั้นจึงทำให้ข้อกำหนดที่ได้มีความชัดเจน ถูกต้องและไม่คลุมเครือ เมื่อนำข้อกำหนดดังกล่าวไปพัฒนาโปรแกรม จะช่วยให้เราไม่ต้องลองผิดลองถูก สะดวกในการแก้ไขและพัฒนาโปรแกรมในโอกาสต่อๆไป

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

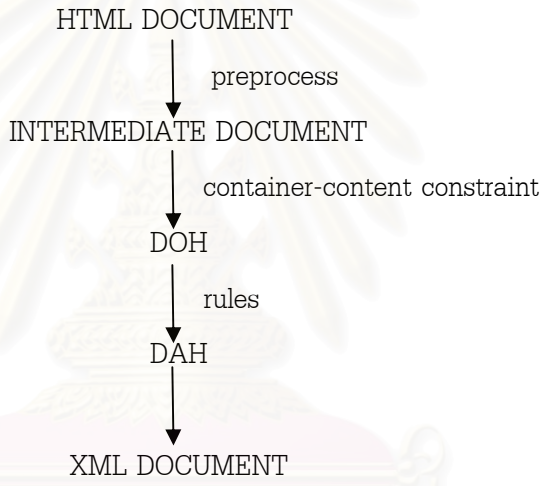
ในงานวิจัย [6] Lim และ Ng เสนอวิธีการแปลงเอกสาร HTML เป็นเอกสาร XML โดยใช้ idea ของ tree ในการแปลง โดยจะแปลงเอกสาร HTML ที่เป็น input file ไปเป็น tree ก่อนเข้าสู่กระบวนการแปลงเป็น เอกสาร XML โดยไม่จำกัดรูปแบบของเอกสาร HTML ดังนั้น HTML ที่จะนำมาแปลงจะมีรูปแบบอย่างไรก็ได้

ในงานวิจัย [8] Ouahid และ Karmouch เสนอวิธี Case-Based transformation ซึ่งเป็น การแปลงเอกสาร HTML ที่ค่อนข้างเฉพาะเจาะจงไปเป็นเอกสาร XML ดังนั้นเอกสาร HTML ที่จะนำมาแปลงจึงถูกจำกัดด้วยรูปแบบที่ถูกกำหนดไว้แล้ว ซึ่งเป็นข้อเสียของวิธีนี้

บทที่ 3
วิธีดำเนินการวิจัย

3.1 ภาพรวมของ process ทั้งหมด

กระบวนการในระบบนี้จะเริ่มด้วยการมี input file เป็นเอกสาร HTML และจะผ่านขั้นตอน preprocess เพื่อให้เอกสารดังกล่าวอยู่ในรูปแบบที่เหมาะสมสำหรับการแปลง กระบวนการต่อไปคือการเก็บข้อมูลในเอกสาร HTML ให้อยู่ในรูปของโครงสร้างต้นไม้ (tree) โดยต้นไม้ที่ได้นี้เราเรียกว่า Document Hierarchy (DOH) ต่อมาเราจะสร้างกฎ (rules) เพื่อมาแปลง DOH ที่ได้นี้ให้เป็น tree ใหม่เรียกว่า Data Hierarchy (DAH) ซึ่งต้นไม้ที่ได้ใหม่นี้จะมีเพียงข้อมูลเท่านั้น tag จะถูกกำจัดทิ้งไป แล้วจึงนำ DAH ที่ได้มาแปลงเป็นเอกสาร XML ต่อไป



รูปที่ 3.1 แสดงภาพรวมของกระบวนการทั้งหมด

3.2 ประเภทของ tag ที่เกี่ยวข้อง

tag ที่จะกล่าวถึงในงานวิจัยนี้แบ่งออกเป็น 2 ประเภทหลักๆคือ

3.2.1 tag ที่จะเก็บเอาไว้ (type 1) คือ tag ที่จะคงเอาไว้ไม่ถูกกำจัดในขั้นตอน preprocess เพราะเป็น tag ที่เก็บข้อมูลสำคัญที่จะนำไปใช้ในการแปลงเป็นเอกสาร XML

3.2.2 tag ที่จะตัดทิ้ง (type 2) คือ tag ที่จะถูกกำจัดในขั้นตอน preprocess เนื่องจากเป็น tag ที่เน้นการแสดงผล

ซึ่งชนิดของ tag ในแต่ละประเภทแสดงในตารางที่ 3.1

ตารางที่ 3.1 แสดงชนิดของ tag แต่ละประเภท

tag ที่จะเก็บเอาไว้ (type 1) (tag ที่เก็บ data)	tag ที่จะตัดทิ้ง (type 2) (tag ที่เก็บส่วน representation)
BODY, H1, H2, H3, H4, H5, H6, UL, OL, DIR, MENU, LI, DL, DT, DD, P, DIV, TR, TABLE, TH, TD, CAPTION, CENTER, BLOCKQUOTE, ADDRESS	HTML, HEAD, TITLE, META, ISINDEX, BASE, LINK, SCRIPT, STYLE, HR, XMP, LISTING, PLAINTEXT, TT, I, B, U, STRIKE, BIG, SMALL, SUB, SUP, EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, IMG, APPLET, FONT, BASEFONT, BR, MAP, FORM, INPUT, SELECT, OPTION, TEXTAREA, PARAM

3.3 ขั้นตอน preprocess

ก่อนที่จะเข้าสู่กระบวนการแปลง เราจำเป็นต้องปรับแต่งเอกสาร HTML ก่อนดังนี้

- สร้าง tag ปิดเพิ่มให้กับ tag ที่ไม่มี tag ปิด
- ตรวจสอบว่า tag ในเอกสาร HTML เป็น tag ประเภท type 1 หรือ type 2 หรือไม่ ถ้าไม่ใช่ให้กำจัดทิ้ง รวมทั้งกำจัด tag ประเภท type 2 ด้วย
- แยกเอกสาร HTML ออกเป็น 2 ส่วนคือ ส่วนแรกเก็บ title element ไว้ในตัวแปร TE ส่วนที่สองเก็บ body element ไว้ในตัวแปร BE ซึ่งจะนำมาใช้ในขั้นตอนของกระบวนการแปลง
- แยกเก็บ table element ไว้ในตัวแปร TE เพื่อเก็บไว้สำหรับการแปลง
- เติมค่า index ให้กับ tag ทุกตัวยกเว้น title กับ body
- ตัด extra spaces ภายในเอกสาร

ภายหลังจากขั้นตอน preprocess ก็คือการสร้าง DOH โดย DOH คือ tree ที่แสดงโครงสร้างความสัมพันธ์ระหว่าง element และ ข้อมูลที่อยู่ภายใน element นั้น (container-content) ตามไวยากรณ์ของ HTML ซึ่งจาก tree ดังกล่าวจะแสดงให้เราเห็นอย่างชัดเจนถึงโครงสร้างของข้อมูล

3.4 เงื่อนไข **container-content constraint**

เงื่อนไข container-content constraint คือเงื่อนไขในการสร้าง tree ที่กล่าวถึงความสัมพันธ์ระหว่าง tag และ data หรือ subelement ภายใน element หนึ่งๆ เช่น

ตัวอย่างที่ 1 ถ้า element ที่ต้องการแปลงเป็น tree คือ

<H1>MATH</H1>

จะได้ tree ดังรูป

รูปที่ 3.2 แสดง tree ที่ได้จาก element ของตัวอย่างที่ 1

ตัวอย่างที่ 2 ถ้า element ที่ต้องการแปลงเป็น tree คือ

<BODY>

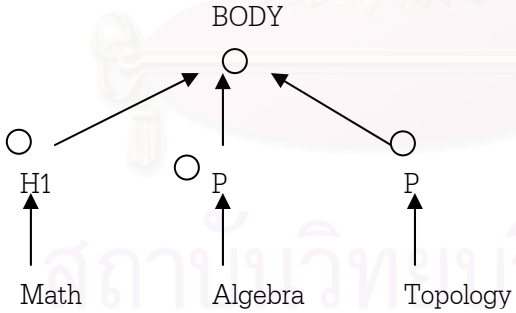
<H1>MATH</H1>

<P>Algebra</P>

<P>Topology</P>

</BODY>

จะได้ tree ดังรูป



รูปที่ 3.3 แสดง tree ที่ได้จาก element ของตัวอย่างที่ 2

เป็นต้น

แต่ก่อนที่จะกล่าวถึงข้อกำหนดของระบบ จะขอกำหนดเซตที่จะใช้ในระบบดังนี้

3.5 เซตที่ใช้ในระบบ

เนื่องจากการสร้างข้อกำหนดของระบบงานนี้จำเป็นต้องใช้สมาชิกของเซตบางเซตซึ่งไม่เป็นที่รู้จักสากลและจำเป็นต้องใช้บ่อยๆ จึงขอกำหนดไว้เพื่อความสะดวกในการนำไปใช้พร้อมคำอธิบายดังนี้

Definition 1. $\theta \in$ char , θ = " " หรือ \t หรือ \n

กำหนดให้ θ แทนตัวอักษรว่าง 1 ตัว (a space) หรือ ระยะ tab หรือ การขึ้นบรรทัดใหม่

Definition 2. G = { g $\in$ seq char | g = TITLE, BODY, H1, H2, ..., H6, P, ...}

เซต G คือ เซตที่ประกอบด้วย tag เปิดในภาษา HTML 2.0

Definition 3. V = { v $\in$ seq char | v = _00,_01,_02,...,_98, _99}

เซต V คือ เซตที่ประกอบด้วย เครื่องหมายขีดล่างและตามด้วยตัวเลขตั้งแต่ 00 ถึง 99

Definition 4. T = { t$\in$ seq char | $\exists$g $\in$ G, $\exists$v $\in$ V $\bullet$ t = g + v }

เซต T คือเซตที่ประกอบด้วย tag เปิดในภาษา HTML โดยเติมท้ายด้วยหมายเลขตั้งแต่ 00-99 เพื่อป้องกันความสับสนในขั้นตอนการสร้างและจัดการ DOH และ DAH

Definition 5. A = { a $\in$ seq char | $\forall$i : 1..j $\bullet$ $\exists atb_i \in$ seq char $\bullet$ a = (θ + atb_i)^j }

เซต A คือเซตที่ประกอบด้วยชุดของ attributes ที่ถูกคั่นด้วย space

Definition 6. O = { o $\in$ seq char | $\exists$t $\in$ T $\wedge$ $\exists$a $\in$ A $\bullet$ o = "<" + t + a + ">" }

เซต O คือเซตที่ประกอบด้วยสมาชิกที่อยู่ในเซต T ซึ่งเป็น tag และมี attribute อยู่ภายใน

Definition 7. C = { c$\in$ seq char | $\exists$t $\in$ T $\bullet$ c = "</" + t + ">" }

เซต C คือเซตที่ประกอบด้วย tag ปิดในภาษา HTML

Definition 8. I = { i $\in$ seq char | "<" , ">" $\notin$ I}

เซต I คือเซตของตัวอักษรที่ไม่มี "<" และ ">"

Definition 9. E = { e $\in$ seq char | $\exists$t$\in$T $\wedge$ $\exists$d $\in$ seq char $\wedge$ $\exists$a $\in$ A $\bullet$ e = "<" + t + a + ">" + d + "</" + t + “>” }

เซต E คือเซตเป็น element ใน HTML โดยที่ tag เปิดและ tag ปิดจะต้องสอดคล้องกัน

นิยามเพิ่มเติม

Definition 10. $(I+E)^1 = I+E$

Definition 11. $(I+E)^m = I+E+(I+E)^{m-1}$ โดยที่ m $\in \mathbb{N}$ และ m>=2

Definition 12. $front^n$ t = front (front (front (....(front t)))) จำนวน n ครั้ง โดยที่ n $\in \mathbb{N}$

Definition 13. $(x_i)^j = x_1 + x_2 + ... + x_{j-1} + x_j$ โดยที่ $\exists$i,j $\in \mathbb{N}$, i : 1..j

Definition 14. $((a_{q,r})^m)^n = (a_{1,1} + a_{1,2} +... + a_{1,n}) + (a_{2,1} + a_{2,2} +... + a_{2,n}) + ... + (a_{m,1} + a_{m,2} +... + a_{m,n})$ โดยที่ $\exists$m,n,p,q$\in \mathbb{N}$, q : 1..m, r : 1..n

3.6 กระบวนการสร้าง DOH

ในระบบของเราเอกสาร HTML ที่เป็น input file จะถูกเก็บอยู่ในรูปของ tree แต่เนื่องจากเราไม่ทราบจำนวน child node ของแต่ละ parent node เราจึงจะใช้การเก็บข้อมูลในรูปของ function ที่ map จาก element (E), ข้อมูล - information (I) หรือ tag (T) ไปยัง tag ที่ล้อมมันอยู่ในรูปของ schema ชื่อ DOH ดังนิยามต่อไปนี้

DOH

$d1 : \mathbf{P}\, E \mid \mathbf{P}\, I \mid \mathbf{P}\, T$

$parent : E \rightarrow I \mid I \rightarrow I \mid T \rightarrow I$

$d1 = dom\ parent$

ใน schema นี้ parent เป็น function ที่ map จาก child node ในที่นี้คือข้อมูล 3 ประเภท element, information และ tag ซึ่งได้แก่เซต E, เซต I และเซต T ตามลำดับ ไปยัง parent node ซึ่งในที่นี้คือเซต I ดังนั้น domain ของฟังก์ชัน parent จะเป็นสมาชิกของเซต E, เซต I และเซต T และ range เป็นเซต I ตามที่กล่าวข้างต้น

เมื่อระบบถูกสร้างขึ้นมาเป็นครั้งแรก แน่นอนว่าภายใน schema ย่อมไม่มีข้อมูลอยู่ ดังนั้นก่อนที่จะมีการกระทำใดๆเราจะกำหนดค่าว่างให้กับ schema เริ่มต้นก่อนโดยผ่าน schema InitDOH ซึ่งมีนิยามดังนี้

InitDOH

DOH

$d1 = \varnothing$

นอกจาก schema ที่ได้กล่าวไปแล้ว เราจะสร้าง schema เพิ่มขึ้นมาอีก 1 schema เพื่อเก็บ attributes ของแต่ละ tag ไว้สำหรับนำมาใช้ในภายหลัง เช่น เป็นข้อมูลสำหรับเอกสาร XSL เป็นต้น

ATB

$d2 : \mathbf{P}\, T$

$attrib : T \rightarrow A$

$d2 = dom\ attrib$

InitATB

ATB

$d2 = \varnothing$

schema ATB กล่าวถึงการสร้างระบบสำหรับเก็บ attributes มี attrib เป็นฟังก์ชันที่ map จาก tag ไปยัง attributes และ schema InitATB เป็นตัวกำหนดค่าเริ่มต้นซึ่งเป็นค่าว่างให้กับ schema ATB

ในการสร้าง tree ใดๆ root node จะมีเพียง node เดียวดังนั้นเมื่อเราอ่านข้อมูลจาก body element (BE) ซึ่งเป็น input เราจะได้ <body> เป็น root node ใน tree ที่จะสร้างขึ้นนี้ ส่วนข้อมูลในส่วนที่ body element ล้อมอยู่จะเป็น child node ดังรายละเอียดใน schema DefineRoot

DefineRoot

ΔDOH

ΔATB

BE : seq char

$\exists g \in G \land g = \text{"BODY"}, \exists a \in A, \exists o \in O \bullet o = \text{"<"} + g + a + \text{">"} \land$

$attrib' = attrib \cup \{ g \rightarrow a \} \land$

$\forall j : 1..m \bullet \exists i_j \in I, \exists e_j \in E, \exists c \in C \bullet c = \text{"</"} + g + \text{">"} \land$

$BE = o + (i_j + e_j)^m + c \land$

$i_j \neq <> \land parent' = parent \cup \{ i_j \rightarrow g \} \land$

$e_j \neq <> \land parent'' = parent' \cup \{ e_j \rightarrow g \}$

ในที่นี้ g มีค่าเป็น BODY, o เป็น tag เปิด BODY ที่อาจจะมีหรือไม่มี attribute, c เป็น tag ปิดของ g นั่นคือ c = </BODY> , i_j คือ information หรือ data ใดๆ และ e_j คือ element ใดๆ ที่ถูกล้อมด้วย body element (BE) ซึ่งเป็น input ผลลัพธ์ที่ได้จาก schema นี้คือ จะได้ BODY เป็น root node และ ข้อมูลที่อยู่ใน body element ซึ่งจะต้องไม่ใช่สายอักขระว่างในที่นี้แทนด้วยสัญลักษณ์ < > จะถูก map ไปยัง root node โดยเรียงตามลำดับ นอกจากนี้ยังมีการเก็บค่า attributes ของ tag BODY สำหรับการใช้ในภายหลังด้วย

ขั้นตอนต่อไปจะเป็นขั้นตอนสุดท้ายของการสร้าง DOH ดังรายละเอียดใน schema CompleteDOH วิธีการสร้างคือกระจายทุกๆ elements ที่ประกอบด้วย tag, data และ subelements จนกว่าจะไม่มี subelement เหลืออยู่นอกจาก data เท่านั้น ที่ระดับนี้เราเรียกว่า leaf node ความสัมพันธ์

แบบ tree เกิดขึ้นคือ tag ใน element ใดๆเมื่อถูกแปลงเป็น tree มันจะเป็น child node ที่ map ไปยัง parent node ของ element เดิมและเป็น parent node ของ data และ subelements ที่เกิดขึ้นใหม่ตามลำดับ การพิสูจน์ schema DefineRoot นี้ได้แสดงไว้เป็นตัวอย่างในภาคผนวก ง

CompleteDOH

ΔDOH

ΔATB

$\exists e \in E \bullet e \in \text{dom parent} \land \exists p \in \text{seq char} \bullet p = \text{parent}(e) \land$

$\exists g \in G, \exists v \in V, \exists t \in T \bullet t = g + v \land$

$\exists a \in A, \exists o \in O \bullet o = \text{"<"} + t + a + \text{">"} \land$

$\forall j : 1..m \bullet \exists i_j \in I, \exists e_j \in E, \exists c \in C \bullet c = \text{"</"} + t + \text{">"} \land$

$e = o + (i_j + e_j)^m + c \land$

$[\quad (\quad g = \text{"A"} \land$

$i_j \neq <> \land \text{parent}' = \text{parent} \cup \{ i_j \rightarrow \text{front}(\text{tail } o) \} \land$

$e_j \neq <> \land \text{parent}'' = \text{parent}' \cup \{ e_j \rightarrow \text{front}(\text{tail } o) \} \land$

$\text{parent}^3 = \text{parent}'' \setminus \{ e \rightarrow p \} \land$

$\text{parent}^4 = \text{parent}^3 \cup \{ \text{front}(\text{tail } o) \rightarrow p \} \quad)$

$\lor$

$(g \neq \text{"A"} \land$

$\text{attrib}' = \text{attrib} \cup \{ t \rightarrow a \} \land$

$i_j \neq <> \land \text{parent}' = \text{parent} \cup \{ i_j \rightarrow t \} \land$

$e_j \neq <> \land \text{parent}'' = \text{parent}' \cup \{ e_j \rightarrow t \} \land$

$\text{parent}^3 = \text{parent}'' \setminus \{ e \rightarrow p \} \land$

$\text{parent}^4 = \text{parent}^3 \cup \{ t \rightarrow p \} \quad) \quad]$

ผลที่เกิดขึ้นจาก schema นี้เป็นดังที่ได้กล่าวไปแล้วข้างต้น ในส่วนที่เพิ่มขึ้นคือ การจัดการกับ tag A โดย tag นี้จะเก็บ attribute ที่จะนำมาใช้ในขั้นตอนต่อไปเอาไว้ นั่นคือ HREF ดังนั้นเราจึงยังไม่ตัด attribute นี้ทิ้งแต่เก็บเอาไว้ในตัว tag ด้วยเลย และขั้นตอนของการสร้าง DOH ก็เสร็จสมบูรณ์ การพิสูจน์ schema CompleteDOH นี้ได้แสดงไว้เป็นตัวอย่างในภาคผนวก ง

ใน schema CompleteDOH นี้มีการแสดงตัวแปร parent ในหลายๆรูปแบบได้แก่ parent, parent', parent'', $parent^3$ และ $parent^4$ แต่ละรูปแบบคือความสัมพันธ์เดียวกันแต่ต่างกันที่ลำดับก่อนหลังของการทำงานในระบบ $parent^3$ และ $parent^4$ อาจเขียนแทนด้วย parent''' และ parent'''' ตามลำดับแต่การแสดงโดยใช้ $parent^3$ และ $parent^4$ ให้ความสะดวกและลดความสับสนในการใช้งาน และเราจะใช้การแสดงลำดับก่อนหลังของตัวแปรอื่นในลักษณะนี้ตลอดงานวิจัยนี้

เนื่องจากสิ่งที่เราต้องการจากเอกสาร HTML คือข้อมูลเท่านั้น ดังนั้นเมื่อสร้าง DOH เสร็จสมบูรณ์เราจึงต้องกำจัด tag ที่เก็บอยู่ใน node (node ที่มีแต่ data จะข้ามไป) ของ DOH tree กระบวนการดังกล่าวดำเนินไปด้วยกฎต่างๆ (rules) ผลที่ได้คือ tree ใหม่ที่ไม่มี node ใดเก็บ tag เอาไว้เลย เราเรียก tree ที่ได้จากขั้นตอนนี้ว่า Data Hierarchy (DAH)

3.7 กระบวนการสร้าง DAH

เราจะนำ tree ที่ได้จาก DOH มาแปลงเป็น DAH โดยใช้กฎต่างๆ ดังนี้

Rule1

ΔDOH

$\exists c_1, c_2, c_3 \in I \ \wedge \exists p \in seq\ char \bullet p = parent\ (c_1) \ \wedge$

$\exists g \in G, \exists v \in V, \exists t \in T \bullet t = g + v \wedge g = "A" \wedge \forall k : 1..l \bullet$

$\exists atb_k \in seq\ char \ \wedge \exists o \in O \bullet o = "<" + t + (\theta + atb_k)^1 + ">" \ \wedge$

$p = parent\ (front\ (tail\ o)) \ \wedge front\ (tail\ o) = parent\ (c_2) \ \wedge$

$p = parent\ (c_3) \ \wedge$

$\exists x \in seq\ char \wedge x = c_1 + \theta + c_2 + \theta + c_3 \ \wedge \exists m \in \mathbb{N}, \exists y \in seq\ char \bullet$

$atb_m = "HREF=" + y \wedge$

$parent' = parent \ \setminus \{ c_1 \rightarrow p, c_2 \rightarrow front\ (tail\ o), c_3 \rightarrow p, front\ (tail\ o) \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ x \rightarrow p, atb_m \rightarrow x)$

ใน schema Rule1 นี้มีสัญลักษณ์ใหม่คือ front และ tail โดย front ของสายอักขระใดหมายถึงสายอักขระนั้นที่ตัดอักษรตัวสุดท้ายทิ้งและ tail ของสายอักขระใดหมายถึงสายอักขระนั้นที่ตัดอักษรตัวแรกสุดทิ้ง

ใน schema Rule1 นี้เราพิจารณาความสัมพันธ์ $c_1 \rightarrow p$, $c_2 \rightarrow front\ (tail\ o)$, $c_3 \rightarrow p$, $front\ (tail\ o) \rightarrow p$ เมื่อ p เป็น parent node ของ c_1, c_3 และ $front\ (tail\ o)$ ส่วน $front\ (tail\ o)$ คือ anchor tag ที่ตัด “<”, “>” ออกเป็น parent node ของ c_2 ในที่นี้ c_1, c_2, c_3 คือ data ซึ่งจะแทนด้วย

$x \rightarrow p$, $atb_m \rightarrow x$ เมื่อ atb_m คือ HREF attribute และ x คือการเอา data c_1, c_2, c_3 มาเชื่อมต่อกันโดยคั่นกลางแต่ละตัวด้วย space

ใน schema Rule2 ที่จะกล่าวถัดไป จะมีการใช้สมบัติเกี่ยวกับค่า precedence ของ tag แต่ละ tag ที่อยู่ในประเภทที่ 1 ซึ่งในที่นี้จะแสดงสมบัติเกี่ยวกับค่า precedence ในรูปตัวเลข โดยที่เราจะสร้าง function ชื่อ pred ที่ map จาก tag ไปยังจำนวนเต็ม ถ้า precedence ของ tag ใดสูงกว่าก็จะมีค่ามากกว่า ดังข้อมูลใน schema Precedence

Precedence

$pred : T \rightarrow \mathbb{N}$

$\exists g \in G \ \wedge \ , \exists v \in V, \exists t \in T \bullet t = g + v \wedge$

[

(g = "H1" $\wedge$ pred(t) = 100) $\vee$

(g = "H2" $\wedge$ pred(t) = 90) $\vee$

(g = "H3" $\wedge$ pred(t) = 80) $\vee$

(g = "H4" $\wedge$ pred(t) = 70) $\vee$

(g = "H5" $\wedge$ pred(t) = 60) $\vee$

(g = "H6" $\wedge$ pred(t) = 50) $\vee$

(g $\in$ { "P", "UL", "OL", "DL", "DIR", "MENU", "ADDRESS", "DIV", "CENTER", "BLOCKQUOTE", "TABLE" } $\wedge$ pred(t) = 40) $\vee$

(g $\in$ { "LI", "DT", "CAPTION" } $\wedge$ pred(t) = 30) $\vee$

(g $\in$ { "DD", "TR" } $\wedge$ pred(t) = 20) $\vee$

(g $\in$ { "TH", "TD" } $\wedge$ pred(t) = 10)]

Rule2

ΔDOH

ΞPrecedence

$\forall m : 1..n \bullet t_m \in T, g_m \in G, v_m \in V, t_m = g_m + v_m \wedge$

$p = parent(t_m) \wedge \exists j \in \mathbb{N}, \exists k \in \mathbb{N} \bullet 1<=j<k<=n \wedge$

$\exists t_j, t_k \in T \wedge p = parent(t_j) \wedge p = parent(t_k) \wedge$

$\exists i_1, i_2 \in I \bullet parent(i_1) = t_j, parent(i_2) = t_k \wedge$

$[(g_k = "ADDRESS" \wedge pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j , t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_1 \rightarrow p, front^3 (tail\ t_k) \rightarrow i_1 \})$

$\vee$

$(g_k = "P" \wedge pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j , i_2 \rightarrow t_k , t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_1 \rightarrow p, "DESCRIPTION" \rightarrow i_1, i_k \rightarrow "DESCRIPTION" \})$

$\vee$

$(g_k \notin \{ "ADDRESS", "P", "TR" \} \wedge$

$pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j , i_2 \rightarrow t_k , t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_2 \rightarrow i_1, i_1 \rightarrow p \})]$

ในขั้นตอนของ Rule 2 จะทำการลด tag และสร้างความสัมพันธ์ใหม่เริ่มจากซ้ายไปขวา โดยเราจะสนใจ tag 2 ตัวที่อยู่ติดกันและมี child node เป็น leaf node ซึ่งประกอบไปด้วยการกระทำย่อยๆ 4 อย่าง ได้แก่

1. ในกรณีที่ tag ที่ 2 เป็น ADDRESS และ precedence ของ tag แรกมากกว่า tag ที่ 2
2. ในกรณีที่ tag ที่ 2 เป็น P และ precedence ของ tag แรกมากกว่า tag ที่ 2
3. ในกรณีที่ tag ที่ 2 ไม่ใช่ ADDRESS, P, TR และ precedence ของ tag แรกมากกว่า tag ที่ 2

ในกรณีที่เป็น tag เกี่ยวกับ table เช่น TABLE, TH, TR จะกล่าวถึงในภายหลัง

สำหรับ Rule 3 กล่าวถึง 3 node ใดๆ ที่ความสัมพันธ์เป็น $c_2 \longrightarrow t \longrightarrow c_1$ เมื่อ c_2, c_1 คือ data และ t_2 เป็น tag ที่ไม่ใช่ address หรือ tag ที่เกี่ยวกับตาราง หรือ DESCRIPTION ซึ่งจะได้ผลเป็น $c_2 \longrightarrow c_1$ เช่น เมื่อมี subtree ใน DOH เป็น fried rice $\longrightarrow$ li $\longrightarrow$ food เราจะได้ผลจาก schema Rule3 เป็น fried rice $\longrightarrow$ food เป็นต้น เราใช้นิยามดังนี้

Rule3

ΔDOH

$\exists c_1, c_2 \in I$, $\exists g \in G$, $\exists v \in V$, $\exists t \in T \bullet t = g + v \wedge$

$t = parent(c_2) \wedge c_1 = parent(t) \wedge$

$[\ g \neq$ "ADDRESS" $\vee\ g \notin \{$ "TH", "TD", "TR" $\} \vee g \neq$ "DESCRIPTION" $] \wedge$

$parent^1 = parent \setminus \{ c_2 \longrightarrow t\ ,\ t \longrightarrow c_1 \} \wedge$

$parent^2 = parent^1 \cup \{ c_2 \longrightarrow c_1 \}$

ใน Rule 4 จะเป็นการจัดการกับ DOH เมื่อ DOH ผ่านการแปลงโดย Rule ทั้งสามที่กล่าวถึงมาก่อนหน้านี้แล้ว และผลที่ได้ไม่เป็น tree แต่เป็น forest คือไม่มี root node เราจะแก้ปัญหานี้โดยการสร้าง dummy root node แล้วนำชื่อของ TITLE มาเป็น data เก็บใน dummy root node ดังแสดงใน schema Rule4

Rule4

ΔDOH

TE : seq char

$\forall p : 1..q\ , \forall m : 1..n \bullet \exists i_p, i_m \in I \bullet (i_p\ ,\ i_m) \in parent \wedge$

$\forall a \in I, (i_m\ , a) \notin parent \wedge$

$\exists s \in I \bullet TE =$ "<TITLE>" $+ s +$ "</TITLE>" $\wedge$

$\forall m{:}1..n, parent(i_m) = s$

ในที่นี้ s เป็นชื่อของ TITLE ดังนั้นเราจะให้ s เป็น root node ของ tree นี้

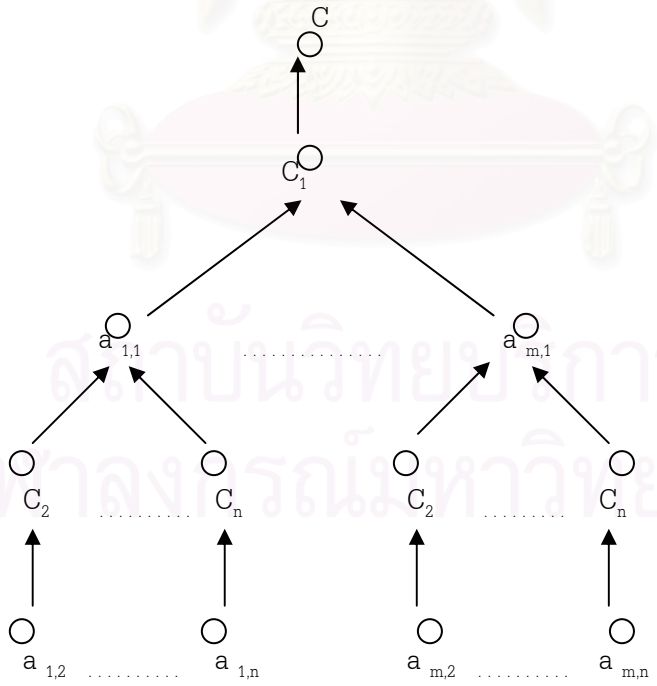
ใน Rule 5 นี้เป็นการจัดการกับ tag <TABLE> หรือ ตาราง โดยเราจะสนใจเป็นพิเศษกับกรณีที่ตารางมีลักษณะดังรูป

C

C_1	.....	C_n
$a_{1,1}$	.....	$a_{1,n}$
;	:	:
:	:	:
:	:	:
$a_{m,1}$	.....	$a_{m,n}$

รูปที่ 3.4 แสดงลักษณะของตารางที่เราสนใจ (โดย C เป็นชื่อของตารางดังกล่าว)

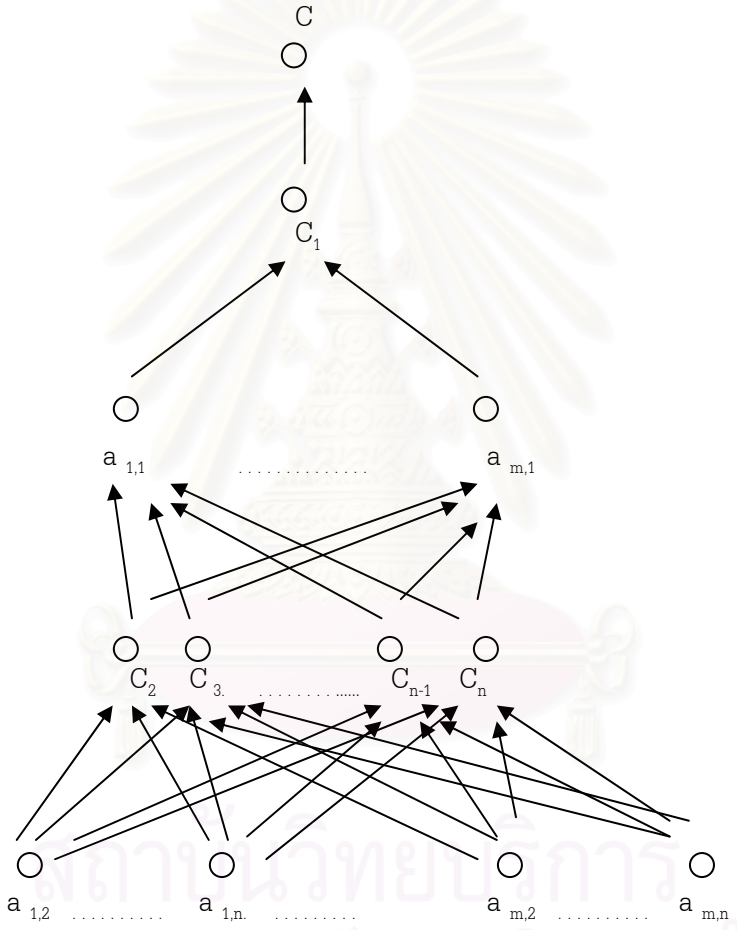
เมื่อ C เป็นชื่อ (caption) ของตาราง (table) C_j ที่ $1<=j<=n$ เป็น column heading และ $a_{i,j}$ เมื่อ $1<=i<=m$ และ $1<=j<=n$ เป็น data โครงสร้าง DAH ที่ได้จากการแปลงตาราง (table) จะเป็นดังรูป



รูปที่ 3.5 แสดงโครงสร้างของ DAH tree ที่ต้องการจากตารางในรูปที่ 3.4

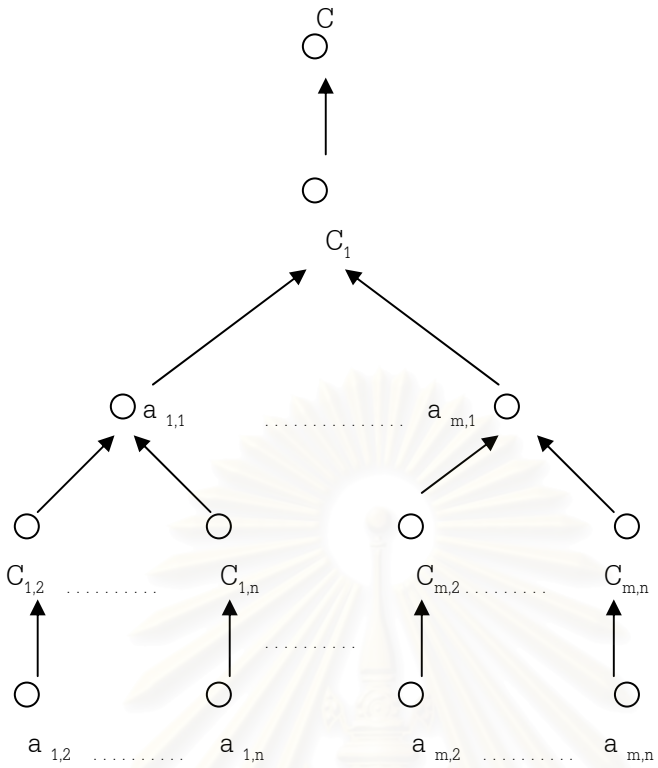
ใน Rule5 นี้จะมี process แยกจากการสร้าง DOH เพราะหลังจากการประยุกต์ใช้ Rule นี้กับ input ที่เป็น table element (ซึ่งใน schema Rule5 นี้เราจะใช้ตัวแปร Tb ดังที่กล่าวในหัวข้อ 3.3) ที่อยู่ภายใน body element ซึ่ง table element จะถูกแปลงเป็น DAH ทันทีโดยไม่ผ่านขั้นตอนของ DOH

นอกจากรายละเอียดที่ได้กล่าวข้างต้น ใน schema Rule5 ยังเพิ่มตัวแปรเพิ่มขึ้น 1 ตัวคือ $C_{i,j}$ โดยที่ 1<=i<=m, 2<=j<=n เมื่อ m เป็นจำนวนแถวและ n เป็นจำนวนคอลัมน์ ซึ่งเดิมคือ C_j แต่เนื่องจากการใช้ C_j ก่อให้เกิดความสับสนในการสร้าง tree เพราะ C_j ถูกนำมาใช้เป็น parent node ในหลายๆ leaf node จึงไม่ตรงกับผลที่เราต้องการดังรูปที่ 3.6



รูปที่ 3.6 แสดงโครงสร้างของ tree ที่ได้จากตารางในรูปที่ 3.4

แต่ถ้าเราใช้ $C_{i,j}$ แทน C_j ก็จะสามารถสร้าง tree ตามรูปที่ 3.5 ได้ตามต้องการ ดังรูปที่ 3.7



รูปที่ 3.7 แสดงโครงสร้างของ DAH tree ที่ปรับปรุงจาก tree ในรูปที่ 3.6

เมื่อสร้างข้อกำหนดเพื่อให้ได้โครงสร้างของ DAH tree ดังที่กล่าวข้างต้น ก็จะได้ข้อกำหนดที่แสดงใน schema Rule5

Rule5

ΔDOH

ΔATB

Tb? : E

$\exists p \in$ seq char, p = parent(Tb) $\wedge$

$\exists g_1 \in G, \exists v_1 \in V, \exists t_1 \in T \bullet t_1 = g_1 + v_1 \wedge g_1 =$ "TABLE" $\wedge$

$\exists d_1 \in C \bullet d_1 =$ "</" + t_1 + ">" $\wedge \exists b_1 \in A \wedge$

$\exists o_1 \in O \bullet o_1 =$ "<" + t_1 + b_1 + ">" $\wedge$ attrib' = attrib $\cup$ { $t_1 \rightarrow b_1$ } $\wedge$

$\forall i:0..m \bullet \exists e_i \in E \bullet Tb = o_1 + (e_i)^i + d_1 \wedge$

[

($\exists g_2 \in G, \exists v_2 \in V, \exists t_2 \in T \bullet t_2 = g_2 + v_2 \wedge g_2 =$ "CAPTION" $\wedge$

$\exists d_2 \in C \bullet d_2 =$ "</" + t_2 + ">" $\wedge \exists b_2 \in A \wedge$

$\exists o_2 \in O \bullet o_2 =$ "<" + t_2 + b_2 + ">" $\wedge$ attrib'' = attrib' $\cup$ { $t_2 \rightarrow b_2$ } $\wedge$

$\exists S \in I \bullet e_0 = o_2 + S + d_2 \wedge \forall r:1..n \bullet C_r \in I \bullet$

$e_1 =$ "<TR>" + ("<TH>" + C_r +"</TH>")^n + "</TR>" $\wedge$

$e_2 + .. + e_m =$ ("<TR>" + ("<TD>" + $a_{q,r}$ + "</TD >")^n + "</TR>")^m $\wedge$

parent(S) = P $\wedge$ parent(C_1) = S $\wedge \forall i:1..m,$ parent($a_{i,1}$) = $C_1 \wedge$

$\forall i:2..m, \forall j:2..n, C_{i,j} \in I \bullet C_{i,j} = C_j \wedge$ parent($C_{i,j}$) = $a_{i,1}$ $\wedge$

$\forall i:2..m, \forall j:2..n \bullet$ parent($a_{i,j}$) = $C_{i,j}$)

$\vee$

($\exists g_2 \in G, \exists v_2 \in V, \exists t_2 \in T \bullet t_2 = g_2 + v_2 \wedge g_2 =$ "CAPTION" $\wedge$

$\exists d_2 \in C \bullet d_2 =$ "</" + t_2 + ">" $\wedge \exists b_2 \in A \wedge$

$\exists o_2 \in O \bullet o_2 =$ "<" + t_2 + b_2 + ">" $\wedge \exists S \in I \bullet e_1 \neq o_2 + S + d_2 \wedge$

S = "TABLE" $\wedge$ parent(S) = p $\wedge \forall r:1..n \bullet C_r \in I \wedge$

$e_0 =$ "<TR>" + ("<TH>" + C_r +"</TH>")^r + "</TR>" $\wedge$

$e_1 + .. + e_m =$ ("<TR>" + ("<TD >" + $a_{q,r}$ + "</TD >")^n + "</TR>")^m $\wedge$

parent(C_1)=S $\wedge \forall i:1..m,$ parent($a_{i,1}$)= $C_1 \wedge$

$\forall i:2..m, \forall j:2..n, C_{i,j} \in I \bullet C_{i,j} = C_j \wedge$ parent($C_{i,j}$) = $a_{i,1} \wedge$

$\forall i:2..m, \forall j:2..n \bullet$ parent($a_{i,j}$) = $C_{i,j}$)]

หลังจาก input file ที่เป็นเอกสาร HTML ผ่านกระบวนการในหัวข้อ 3.6 และ 3.7 แล้วเราจะได้ DAH tree แต่เนื่องจากไวยากรณ์ของภาษา XML ไม่อนุญาตให้มีช่องว่าง (space) ในสายอักขระที่จะนำมาใช้เป็นชื่อของ tag ดังนั้นเมื่อได้ DAH tree แล้วเราจะสร้าง process ที่ทำหน้าที่เติม "_" (under score) ลงในตำแหน่งที่เป็นช่องว่างของข้อมูลใน node ที่ไม่ใช่ leaf node และเมื่อทำเสร็จแล้วเราก็จะได้ DAH tree ที่พร้อมสำหรับกระบวนการแปลงเป็นเอกสาร XML ดังจะได้กล่าวในบทต่อไป

บทที่ 4
การแปลงให้อยู่ในรูปแบบเอกสาร XML

จากบทที่ 3 เราได้กล่าวถึงการแปลงเอกสาร HTML ให้เป็น DAH ในบทนี้จะกล่าวถึงความสำคัญที่จะแปลงโครงสร้าง DAH ให้เป็นเอกสาร XML โดยในขั้นแรกเราจะสร้าง schema XML เพื่อเป็นระบบซึ่งใช้เก็บผลลัพธ์ที่ได้จากกระบวนการนี้

XML

X : seq char

$X = \varnothing$

schema ItoC, Xcorrespond, Dcorrespond ที่จะกล่าวถึงต่อไป จะนำไปใช้ในระหว่างกระบวนการ โดยที่

schema ItoC จะมีข้อมูลในระบบเป็น function ที่เป็นการแปลงเลขจำนวนเต็มระหว่าง 1 ถึง 10 เป็นตัวอักษร "1" ถึง "10" และที่เรากำหนดให้มีการแปลงเพียงเลข 1 ถึง เลข 10 เพราะเพื่อให้สอดคล้องกับ tree ที่เราสร้างตอนเขียนโปรแกรมให้แต่ละ node ที่มี child node จะมีได้ไม่เกิน 10 node

ItoC

itoc : $\mathbb{N} \rightarrow$ seq char

---

itoc = { (1,"1"), (2,"2"), (3,"3"),...(8,"8"), (9,"9"), (10,"10") }

schema Xcorrespond เป็นระบบที่เก็บความสัมพันธ์ระหว่าง ข้อมูลใน DAH และ XML

XCorrespond

xcrp : seq char $\rightarrow$ seq char

$xcrp = \varnothing$

schema Dcorrespond เป็นระบบที่เก็บความสัมพันธ์ระหว่าง ข้อมูลใน DAH และ DTD

DCorrespond

$dcrp : seq\ char \rightarrow seq\ char$

$dcrp = \varnothing$

เพราะว่าหลังจากกระบวนการแปลง tree จาก DOH เป็น DAH โดยอาศัย rules ต่างๆ ผลลัพธ์ที่ได้ยังคงอยู่ในระบบของ DOH แต่เราต้องการเก็บในระบบของ DAH ดังนั้น เราจะสร้างระบบของ DAH ซึ่งในที่นี้ก็คือ schema DAH เพื่อเก็บผลลัพธ์สุดท้ายที่ได้จาก DOH

DAH

ΞDOH

$parent2 : seq\ char \rightarrow seq\ char$

$parent2 = parent$

จากการทำงานของ schema DAH ฟังก์ชัน parent2 เก็บข้อมูลล่าสุดซึ่งเป็นความสัมพันธ์ที่อยู่ใน DOH

เนื่องจากระหว่างกระบวนแปลง tree ค่าของ root อาจมีการเปลี่ยนแปลงจาก DOH เมื่อเริ่มต้น ดังนั้นเราจะกำหนดค่าของ root ไว้ในตัวแปรชื่อ root ด้วย schema RootOfDAH เพื่อความสะดวกในการนำมาใช้ต่อไป

RootOfDAH

ΞDAH

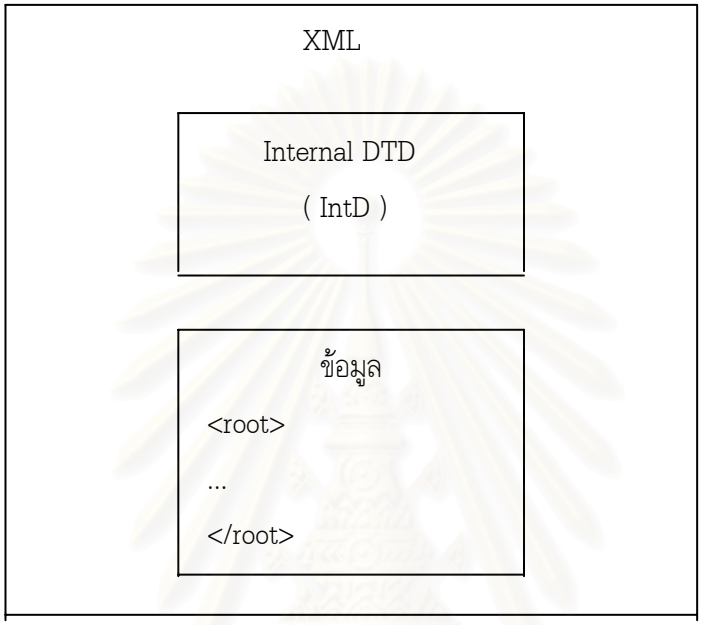
$root! : seq\ char$

$\forall i : 1..j \bullet \exists c_i \in seq\ char, \exists a \in seq\ char \bullet (c_i, a) \in parent \wedge$

$\forall b \in seq\ char \bullet (a,b) \notin parent2 \wedge$

$root! = a$

เอกสาร XML ที่เราจะสร้างนี้ นอกจากจะมีข้อมูลที่เป็นส่วนของข้อมูลของเอกสาร แล้วยังมีส่วนของ Document Type Definition (DTD) ซึ่งใช้บอกโครงสร้างของเอกสารด้วย เนื่องจากเป็น DTD ที่อยู่ภายในเอกสาร XML จึงเรียกว่า internal DTD โดยใน schema InitXML เราจะใช้ตัวแปร IntD และในส่วนของข้อมูล เราจะนำค่าของ root ใน DAH ที่ได้จาก schema RootOfDAH มาเป็นชื่อ tag ที่อยู่ใน element นอกสุด ดังรูป



InitXML

ΞDAH

ΞRootOfDAH

ΔXCorrespond

ΔXML

X! : seq char

X! = "<? xml version='1.0' encoding='windows-874'?>" $\wedge$

$\forall i : 1..j \bullet \exists c_i \in$ seq char $\bullet (c_i, root) \in$ parent2 $\wedge$

$\exists$IntD $\in$ seq char $\wedge$ IntD = $\varnothing \wedge$

$\forall i:1..j \bullet \exists S_i \in$ seq char $\wedge S_i = \varnothing \wedge$ xcrp' = xcrp $\cup$ { (c_i, S_i) } $\wedge$

X!' = X! + IntD + "<" + root + ">" + $S_1 + S_2 + ... + S_i + ... + S_j$ + "</" + root + ">"

ใน schema InitDTD นี้จะกล่าวถึงกระบวนการเริ่มต้นใน Internal DTD โดยใน schema นี้จะใช้ตัวแปร IntD ซึ่งจะแทนโครงสร้างโดยรวมของ Internal DTD และ D แทนโครงสร้างภายใน Internal DTD กำหนดให้ j แทนจำนวน child node ของ root ซึ่งทำให้จำนวน data และ element ใน Internal DTD มีจำนวนเท่ากับ j ด้วย ดังโครงสร้างของ D ใน schema InitDTD

InitDTD

ΞDAH

ΞRootOfDAH

ΔDCorrespond

ΔXML

$\forall i:1..j \bullet \exists M_i \in \text{seq char} \land \exists c_i \in \text{seq char} \bullet (c_i,\text{root}) \in \text{parent2} \land$

[

$\forall d \in \text{seq char} \bullet (d,c_i) \notin \text{parent2} \land M_i = \text{root}+\text{"data"}+\text{itoc}(i)$

$\lor$

[

$\forall i:1..j \ \exists M_i \in \text{seq char} \land \exists N_i \in \text{seq char} \land N_i = \varnothing \land$

$\forall i:1..j \ \text{dcrp}' = \text{dcrp} \cup \{ (c_i, N_i) \} \land$

$\forall m :1..n \bullet \exists c'_m \in \text{seq char} \bullet (c'_m, c_i) \in \text{parent2} \land M_i = c_i$

]

]

$\land$

$\exists D \in \text{seq char} \bullet D = \text{"<!ELEMENT "} + \text{root} + \text{" ("} + M_1 + M_2 + ... + M_i + ... + M_j + \text{")>"} + N_1 + N_2 + ... + N_i + ... + N_j \land$

$\text{IntD} = \text{"<!DOCTYPE "} + \text{root} + \text{" ["} + D + \text{"]>"}$

กระบวนการใน schema ProcessDTD นี้จะเป็นการทำในส่วนของ Internal DTD ที่เหลือให้เสร็จสมบูรณ์ โดยการเติมข้อมูลของ M_i และ N_i ที่ประกาศไว้ใน schema InitDTD

เนื่องจากการประกาศ element กับการประกาศข้อมูลใน DTD มีลักษณะแตกต่างกัน การพิจารณาว่า node ใดใน tree จะมีสถานะเป็น element หรือข้อมูลจะกระทำโดยยึดหลักดังนี้

1. node ใดที่เป็น leaf node จะถือว่า node นั้นเป็นข้อมูล

2. node ใดที่ child node เป็น leaf node จะพิจารณา node นั้นว่าเป็น element โดยมีข้อมูลใน element เป็นค่าของ leaf node นั้น โดยข้อมูลตัวที่ i หากมี parent node เป็น A แล้ว จะกำหนดให้ tag เปิดและ tag ปิดที่ครอบมันคือ "<" + A + "data" + i +">" และ "</" + A + "data" + i +">" ตามลำดับ

3. node ใดที่มี child node เป็น element และข้อมูลปนกัน จะเรียก node นั้นว่าเป็น element ที่มีข้อความใน element เป็น element (node ที่มีสมบัติดังในข้อ 2) และข้อมูล (node ที่มีสมบัติดังในข้อ 1)

เมื่อนำหลักดังที่กล่าวข้างต้นมาใช้ร่วมกับไวยากรณ์การสร้าง DTD ก็จะสามารถสร้างได้ดังข้อกำหนดใน schema ProcessDTD

ProcessDTD

ΞDAH

ΞItoC

ΔXCorrespond

ΔXML

$dcrp \neq \varnothing \land$

[

$\forall i : 1..j \bullet \exists c_i \in seq\ char, \exists N_i \in seq\ char \land (c_i, N_i) \in dcrp \land$

$\forall d \in seq\ char \bullet (d,c_i) \notin parent2 \land \exists e \in seq\ char \bullet (c_i,e) \in parent2 \land$

$N_i = "<!ELEMENT\ " + e + "data" + itoc(i) + "(\#PCDATA)>" \land$

$dcrp' = dcrp \setminus \{ (c_i, N_i) \}$

]

$\lor$

[

$\forall m : 1..n\ \exists c'_m \in seq\ char \bullet (c'_m, c_i) \in parent2 \land$

[

$\forall d \in seq\ char \bullet (d,c'_m) \notin parent2 \land M'_m = c_i + "data" + itoc(m)$

$\lor$

[

$\forall p : 1..q\ \exists c''_p \in seq\ char \bullet (c''_p, c'_m) \in parent2 \land$

$\exists M'_p \in seq\ char \land \exists N'_p \in seq\ char \land N'_p = \varnothing \land M'_m = c'_m$

]

] $\land$

$N_i = N_i + "<!ELEMENT\ " + c_i + "\ (" + M'_1 + M'_2 + ... + M'_m + ... + M'_n + ")>" +$

$N'_1 + N'_2 + ... + N'_m + ... + N'_n \land \forall m:1..n\ \ dcrp' = dcrp \cup \{ (c'_i, N'_i) \}$

]

กระบวนการใน schema ProcessXML นี้จะเป็นการทำในส่วนของข้อมูลให้เสร็จสมบูรณ์ โดยการนำข้อมูลที่อยู่ในฟังก์ชัน xcrp มาจัดการ

การสร้างข้อความในส่วน XML ยังคงใช้หลักการแปลงจาก tree เป็น element หรือ data ดังที่กล่าวในส่วนการสร้าง DTD แต่ข้อความใน DTD กับ XML มีความแตกต่างกันเพราะมีรูปแบบการสร้างไม่เหมือนกัน เมื่อนำหลักดังที่กล่าวถึงในส่วนของการสร้าง DTD มาใช้ร่วมกับไวยากรณ์การสร้าง XML ก็จะสามารถสร้างได้ดังข้อกำหนดใน schema ProcessXML

ProcessXML

ΞDAH

ΞItoC

ΔXCorrespond

ΔXML

$xcrp \neq \varnothing \land$

[

$\forall i : 1..j \bullet \exists c_i \in seq\ char, \exists S_i \in seq\ char \land (c_i, S_i) \in xcrp \land$

$\forall d \in seq\ char \bullet (d, c_i) \notin parent2 \land \exists e \in seq\ char \bullet (c_i, e) \in parent2 \land$

S_i = "<" + e + "data" + itoc(i) + ">" + c_i +"</"+ e + "data" + itoc(i) + ">"]

$\land$ $xcrp = xcrp \setminus \{ (c_i, S_i) \}$

]

$\lor$

[

$\forall m : 1..n \bullet \exists c'_m \in seq\ char \bullet (c'_m, c_i) \in parent2 \land$

$\forall S'_m \in seq\ char \land S'_m = \varnothing \land$

S_i = "<" + c_i+ ">" + S'_1 + S'_2 +...+ S'_m + .. + S'_n + "</" + c_i+ ">" $\land$

$\forall m:1..n, xcrp' = xcrp \cup \{ (c'_m, S'_i) \}$

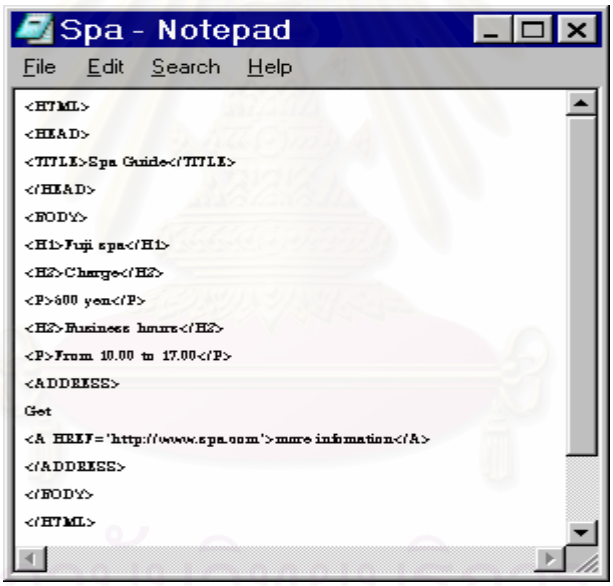
]

บทที่ 5
การทดลอง

ในบทนี้จะกล่าวถึงการทดลองโดยการนำกฎการแปลงดังที่กล่าวถึงในบทที่ 3 และ 4 มาใช้แปลงตัวอย่างเอกสาร HTML โดยจะแสดงขั้นตอนวิธีการแปลงจนได้ผลลัพธ์ที่เป็น DOH, DAH และเอกสาร XML ตามลำดับ ดังแสดงในตัวอย่างในหัวข้อที่ 5.1 และ 5.2

5.1 วิธีการทดลองการแปลงเอกสาร HTML อย่างง่ายไปเป็นเอกสาร XML

ในหัวข้อนี้จะแสดงวิธีการแปลงเอกสาร HTML อย่างง่ายในที่นี้อยู่ในแฟ้มเอกสารชื่อ Spa.html ไปเป็นเอกสาร XML โดยจะแสดงวิธีการแปลงทีละขั้นตอนเพื่อแสดงให้เห็นอย่างชัดเจนว่า เราสามารถนำกฎที่ได้สร้างไว้มาประยุกต์ใช้ได้อย่างไร เอกสาร HTML ที่จะนำมาเป็นตัวอย่างแสดงในรูปที่ 5.1



รูปที่ 5.1 แสดงเอกสาร HTML ที่ต้องการนำมาแปลงอยู่ในแฟ้มเอกสารชื่อ Spa.html

เมื่อนำเอกสาร HTML จากรูปที่ 5.1 มาแปลงเป็น DOH โดยใช้หลัก container-content constraint ดังที่ได้กล่าวไปแล้วในบทที่ 3 ตามขั้นตอนใน schema DefineRoot และ schema CompleteDOH ที่จะกล่าวถึง ก็จะได้โครงสร้าง DOH ดังรูปที่ 5.2

schema DefineRoot

DefineRoot

ΔDOH

ΔATB

BE : seq char

$\exists g \in G \land g = \text{"BODY"}, \exists a \in A, \exists o \in O \bullet o = \text{"<"} + g + a + \text{">"} \land$

$attrib' = attrib \cup \{ g \rightarrow a \} \land$

$\forall j : 1..m \bullet \exists i_j \in I, \exists e_j \in E, \exists c \in C \bullet c = \text{"</"} + g + \text{">"} \land$

$BE = o + (i_j + e_j)^m + c \land$

$i_j \neq <> \land parent' = parent \cup \{ i_j \rightarrow g \} \land$

$e_j \neq <> \land parent'' = parent' \cup \{ e_j \rightarrow g \}$

schema CompleteDOH

CompleteDOH

ΔDOH

ΔATB

$\exists e \in E \bullet e \in \text{dom parent} \land \exists p \in \text{seq char} \bullet p = \text{parent}(e) \land$

$\exists g \in G, \exists v \in V, \exists t \in T \bullet t = g + v \land$

$\exists a \in A, \exists o \in O \bullet o = \text{"<"} + t + a + \text{">"} \land$

$\forall j : 1..m \bullet \exists i_j \in I, \exists e_j \in E, \exists c \in C \bullet c = \text{"</"} + t + \text{">"} \land$

$e = o + (i_j + e_j)^m + c \land$

$[\quad (\quad g = \text{"A"} \land$

$i_j \neq <> \land \text{parent}' = \text{parent} \cup \{ i_j \rightarrow \text{front}(\text{tail } o) \} \land$

$e_j \neq <> \land \text{parent}'' = \text{parent}' \cup \{ e_j \rightarrow \text{front}(\text{tail } o) \} \land$

$\text{parent}^3 = \text{parent}'' \setminus \{ e \rightarrow p \} \land$

$\text{parent}^4 = \text{parent}^3 \cup \{ \text{front}(\text{tail } o) \rightarrow p \} \;)$

$\lor$

$(g \neq \text{"A"} \land$

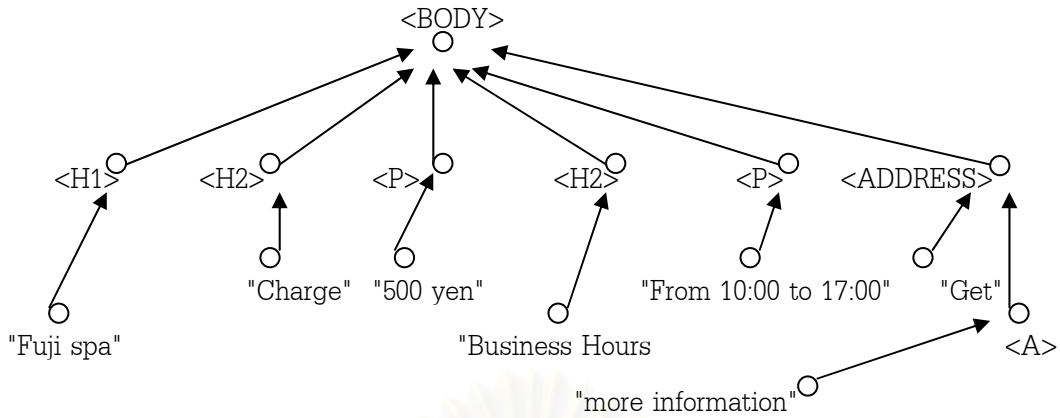
$\text{attrib}' = \text{attrib} \cup \{ t \rightarrow a \} \land$

$i_j \neq <> \land \text{parent}' = \text{parent} \cup \{ i_j \rightarrow t \} \land$

$e_j \neq <> \land \text{parent}'' = \text{parent}' \cup \{ e_j \rightarrow t \} \land$

$\text{parent}^3 = \text{parent}'' \setminus \{ e \rightarrow p \} \land$

$\text{parent}^4 = \text{parent}^3 \cup \{ t \rightarrow p \} \;) \quad]$



รูปที่ 5.2 โครงสร้าง DOH ที่ได้จาก Spa.html (รูปที่ 5.1)

หลังจากได้ DOH แล้วก็เข้าสู่กระบวนการแปลงเป็น DAH ในขั้นตอนแรกเราพบว่ามี node ที่เก็บ tag <A> เอาไว้เราจึงมาเข้ากระบวนการแปลงของ rule 1 ใน schema Rule1 ซึ่งกำลังกล่าวถึงถัดไปและได้ผลลัพธ์ดังรูปที่ 5.3

schema Rule1

Rule1

ΔDOH

$\exists c_1, c_2, c_3 \in I \ \wedge \exists p \in \text{seq char} \bullet p = \text{parent}(c_1) \ \wedge$

$\exists g \in G, \exists v \in V, \exists t \in T \bullet t = g + v \wedge g = \text{"A"} \wedge \forall k : 1..l \bullet$

$\exists atb_k \in \text{seq char} \ \wedge \exists o \in O \bullet o = \text{"<"} + t + (\theta + atb_k)^1 + \text{">"} \ \wedge$

$p = \text{parent}(\text{front}(\text{tail } o)) \ \wedge \text{front}(\text{tail } o) = \text{parent}(c_2) \ \wedge$

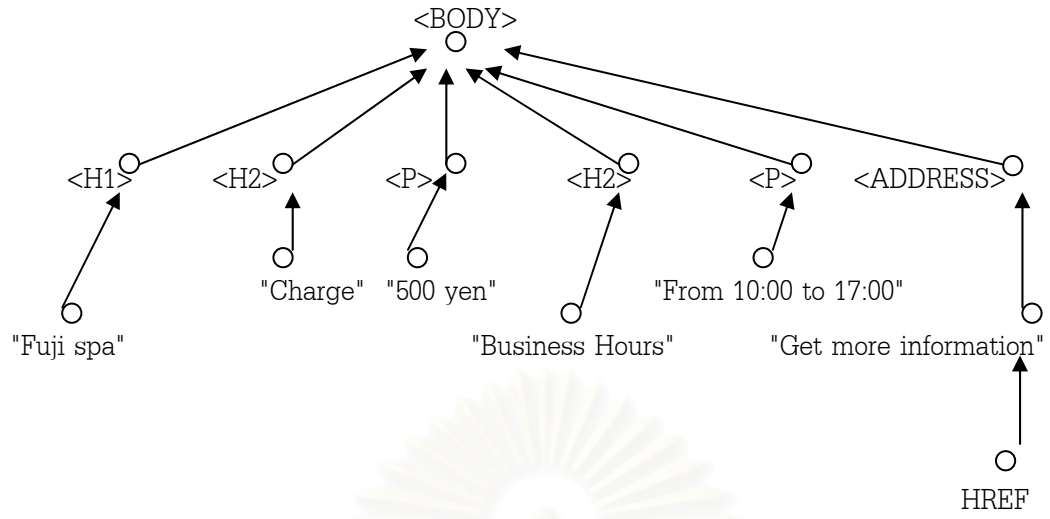
$p = \text{parent}(c_3) \ \wedge$

$\exists x \in \text{seq char} \wedge x = c_1 + \theta + c_2 + \theta + c_3 \ \wedge \exists m \in \mathbb{N}, \exists y \in \text{seq char} \bullet$

$atb_m = \text{"HREF="} + y \wedge$

$\text{parent}' = \text{parent} \setminus \{ c_1 \rightarrow p, c_2 \rightarrow \text{front}(\text{tail } o), c_3 \rightarrow p, \text{front}(\text{tail } o) \rightarrow p \} \wedge$

$\text{parent}'' = \text{parent}' \cup \{ x \rightarrow p, atb_m \rightarrow x \}$



รูปที่ 5.3 โครงสร้าง DOH ที่ได้จากการแปลงโดยใช้ rule 1

ในกระบวนการของ rule 2 ที่จะแสดงให้เห็นถัดไปใน schema Rule2 ได้กล่าวถึงการกระทำกับ tag หลายกรณีด้วยกันแต่ในที่นี้ส่วนที่จะนำมาใช้ได้แก่ส่วนที่กล่าวถึงกรณีที่มีบาง node เก็บค่า tag <P> เอาไว้และอีกข้อคือการแปลง DOH โดยคำนึงถึงค่าลำดับความสำคัญของ tag (precedence) ซึ่งแสดงใน schema Precedence เมื่อนำมาใช้แปลง DOH ในรูปที่ 5.3 ก็จะได้ DOH ดังในรูปที่ 5.4

schema Precedence

Precedence

$pred : T \rightarrow \mathbb{N}$

$\exists g \in G \ \wedge , \exists v \in V, \exists t \in T \bullet t = g + v \wedge$

[

(g = "H1" $\wedge$ pred(t) = 100) $\vee$

(g = "H2" $\wedge$ pred(t) = 90) $\vee$

(g = "H3" $\wedge$ pred(t) = 80) $\vee$

(g = "H4" $\wedge$ pred(t) = 70) $\vee$

(g = "H5" $\wedge$ pred(t) = 60) $\vee$

(g = "H6" $\wedge$ pred(t) = 50) $\vee$

(g $\in$ { "P", "UL", "OL", "DL", "DIR", "MENU", "ADDRESS", "DIV", "CENTER",
"BLOCKQUOTE", "TABLE" } $\wedge$ pred(t) = 40) $\vee$

(g $\in$ { "LI", "DT", "CAPTION" } $\wedge$ pred(t) = 30) $\vee$

(g $\in$ { "DD", "TR" } $\wedge$ pred(t) = 20) $\vee$

(g $\in$ { "TH", "TD" } $\wedge$ pred(t) = 10)]

schema Rule2

Rule2

ΔDOH

ΞPrecedence

$\forall m : 1..n \bullet t_m \in T, g_m \in G, v_m \in V, t_m = g_m + v_m \wedge$

$p = parent(t_m) \wedge \exists j \in \mathbb{N}, \exists k \in \mathbb{N} \bullet 1<=j<k<=n \wedge$

$\exists t_j, t_k \in T \wedge p = parent(t_j) \wedge p = parent(t_k) \wedge$

$\exists i_1, i_2 \in I \bullet parent(i_1) = t_j, parent(i_2) = t_k \wedge$

$[(g_k = "ADDRESS" \wedge pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j, t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_1 \rightarrow p, front^3(tail\ t_k) \rightarrow i_1 \})$

$\vee$

$(g_k = "P" \wedge pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j, i_2 \rightarrow t_k, t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_1 \rightarrow p, "DESCRIPTION" \rightarrow i_1, i_k \rightarrow "DESCRIPTION" \})$

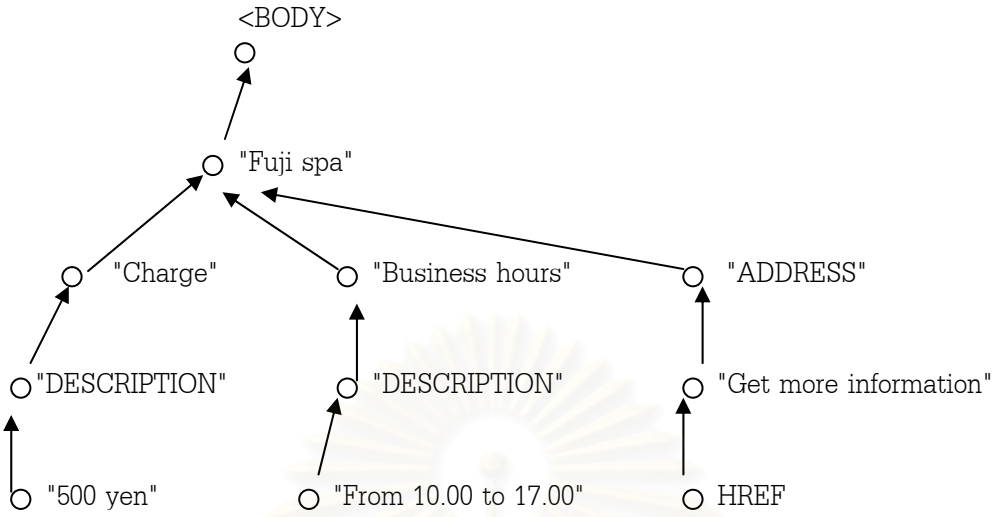
$\vee$

$(g_k \notin \{ "ADDRESS", "P", "TR" \} \wedge$

$pred(t_j) > pred(t_k) \wedge$

$parent' = parent \setminus \{ i_1 \rightarrow t_j, i_2 \rightarrow t_k, t_j \rightarrow p, t_k \rightarrow p \} \wedge$

$parent'' = parent' \cup \{ i_2 \rightarrow i_1, i_1 \rightarrow p \})]$



รูปที่ 5.4 โครงสร้าง DOH ที่ได้จากการแปลงโดยใช้ rule 2

ในขั้นตอนของ rule 3 ที่จะแสดงให้เห็นถัดไปใน schema Rule3 ซึ่งมีการทำงานคือ node ที่เก็บ tag ของ HTML จะถูกกำจัดจนหมดโดยคงไว้เฉพาะ node ที่เก็บข้อมูล ADDRESS หรือ DESCRIPTION เท่านั้น ซึ่งเมื่อแปลง DOH ในรูปที่ 5.4 โดยใช้ rule 3 โครงสร้างที่ได้ก็จะเป็น DAH ดังแสดงในรูปที่ 5.5

schema Rule3

Rule3

ΔDOH

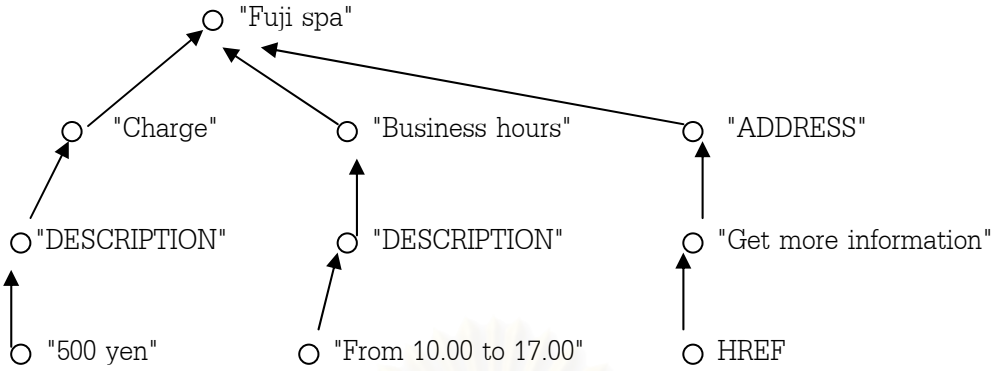
$\exists c_1, c_2 \in I , \exists g \in G, \exists v \in V, \exists t \in T \bullet t = g + v \wedge$

$t = parent (c_2) \wedge c_1 = parent (t) \wedge$

$[g \neq$ "ADDRESS" $\vee g \notin \{$ "TH", "TD", "TR" $\} \vee g \neq$ "DESCRIPTION" $] \wedge$

$parent^1 = parent \setminus \{ c_2 \rightarrow t , t \rightarrow c_1 \} \wedge$

$parent^2 = parent^1 \cup \{ c_2 \rightarrow c_1 \}$



รูปที่ 5.5 DOH ที่ผ่านกระบวนการ rule 3 และได้เป็น DAH

DOH ในรูปที่ 5.5 ไม่มี tag ของเอกสาร HTML เหลืออยู่ และโครงสร้างที่ได้ก็เป็น tree (ไม่ได้เป็น forest) ดังนั้นเราจึงไม่ต้องใช้ rule 4 ในการแปลง DOH ที่ได้ ดังนั้นผล DOH ที่ได้ดังแสดงในรูปที่ 5.5 จึงเป็น DAH ที่ต้องการ และเมื่อนำ DAH tree ที่ได้ไปสร้างเป็นเอกสาร XML ตามกระบวนการที่ได้กล่าวไปแล้วในบทที่ 4 และในที่นี้ก็ได้นำมาแสดงประกอบ 1 schema คือ schema ProcessXML ซึ่งเมื่อ DAH ได้ผ่านกระบวนการการแปลงจาก DAH ไปเป็นเอกสาร XML แล้วก็จะได้ผลลัพธ์ดังรูปที่ 5.6

schema ProcessXML

ProcessXML

ΞDAH

ΞItoC

ΔXCorrespond

ΔXML

$xcrp \neq \emptyset \land$

[

$\forall i : 1..j \bullet \exists c_i \in seq\ char, \exists S_i \in seq\ char \land (c_i, S_i) \in xcrp \land$

$\forall d \in seq\ char \bullet (d, c_i) \notin parent2 \land \exists e \in seq\ char \bullet (c_i, e) \in parent2 \land$

S_i = "<" + e + "data" + itoc(i) + ">" + c_i +"</"+ e + "data" + itoc(i) + ">"]

$\land$ $xcrp = xcrp \setminus \{ (c_i, S_i) \}$

]

$\lor$

[

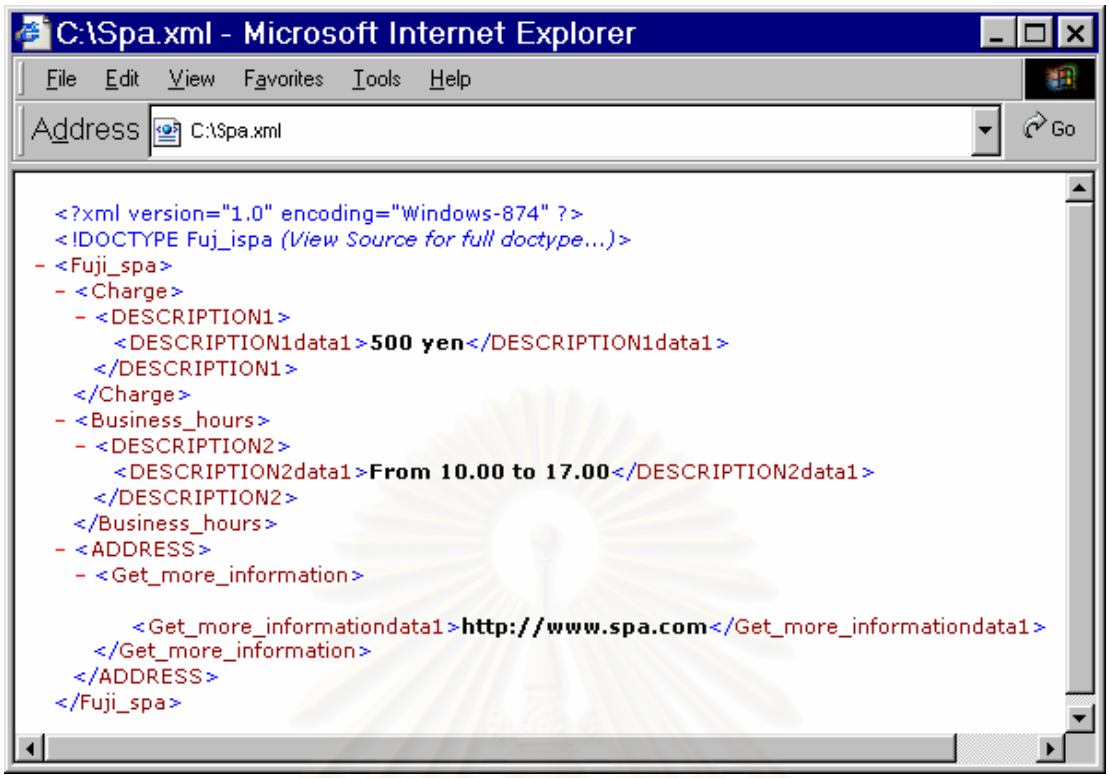
$\forall m : 1..n \bullet \exists c'_m \in seq\ char \bullet (c'_m, c_i) \in parent2 \land$

$\forall S'_m \in seq\ char \land S'_m = \emptyset \land$

S_i = "<" + c_i+ ">" + S'_1 + S'_2 +...+ S'_m + .. + S'_n + "</" + c_i+ ">" $\land$

$\forall m:1..n, xcrp' = xcrp \cup \{ (c'_m, S'_i) \}$

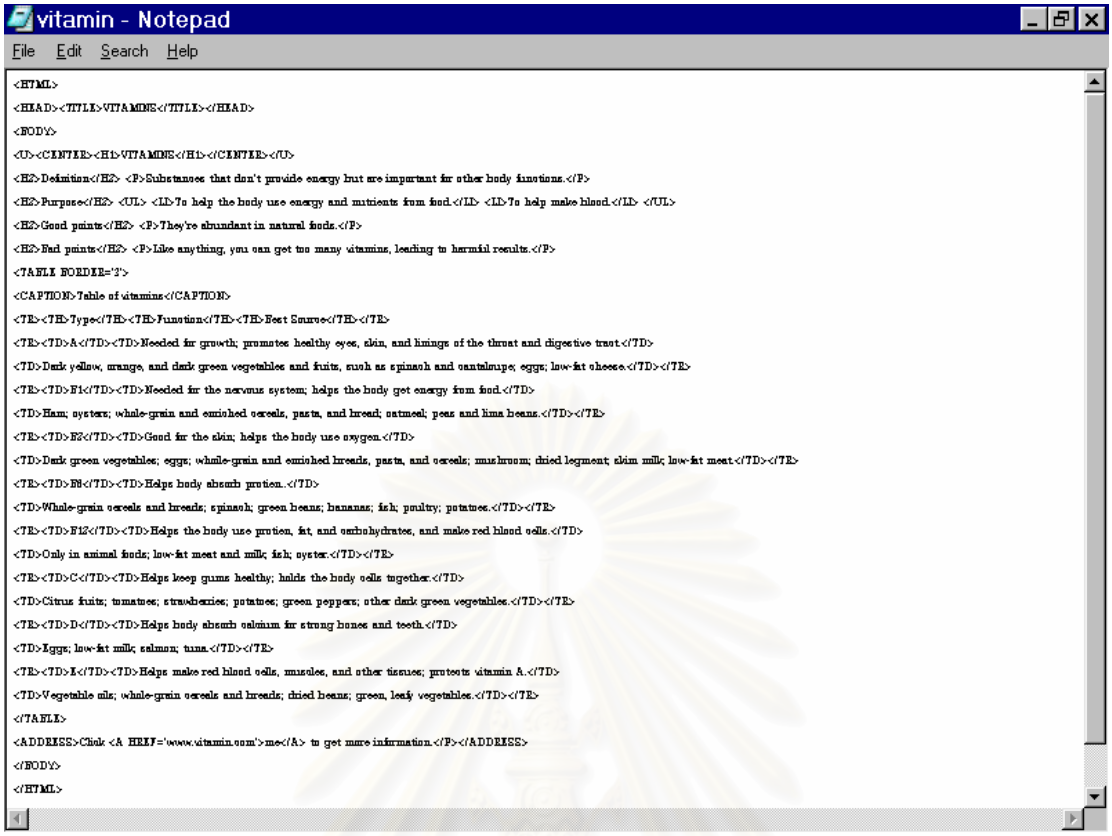
]



รูปที่ 5.6 เอกสาร XML ที่ถูกแปลงจาก DAH ที่ได้จากรูปที่ 5.5

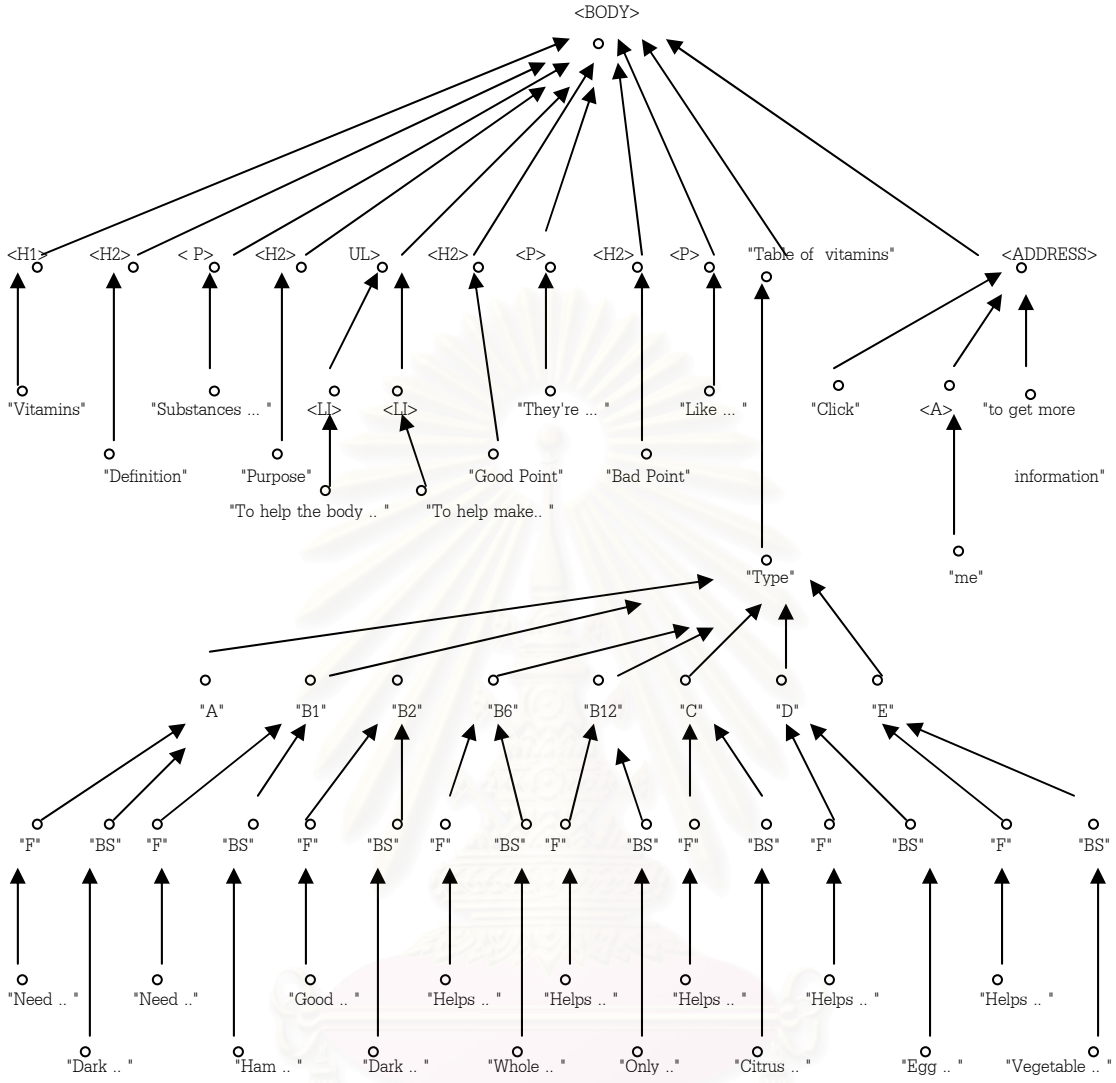
5.2 วิธีการทดลองการแปลงเอกสาร HTML ที่มีโครงสร้างซับซ้อนไปเป็นเอกสาร XML

ในหัวข้อนี้จะแสดงวิธีการแปลงเอกสาร HTML ไปเป็นเอกสาร XML แต่เอกสาร HTML ในหัวข้อนี้มีโครงสร้างซับซ้อนกว่าเอกสาร HTML ในหัวข้อ 5.1 มากเนื่องจากประกอบด้วย tag ที่มีความหลากหลายกว่าและมีจำนวนมากกว่า โดยเอกสาร HTML ที่จะนำมาเป็นตัวอย่างแสดงในรูปที่ 5.7



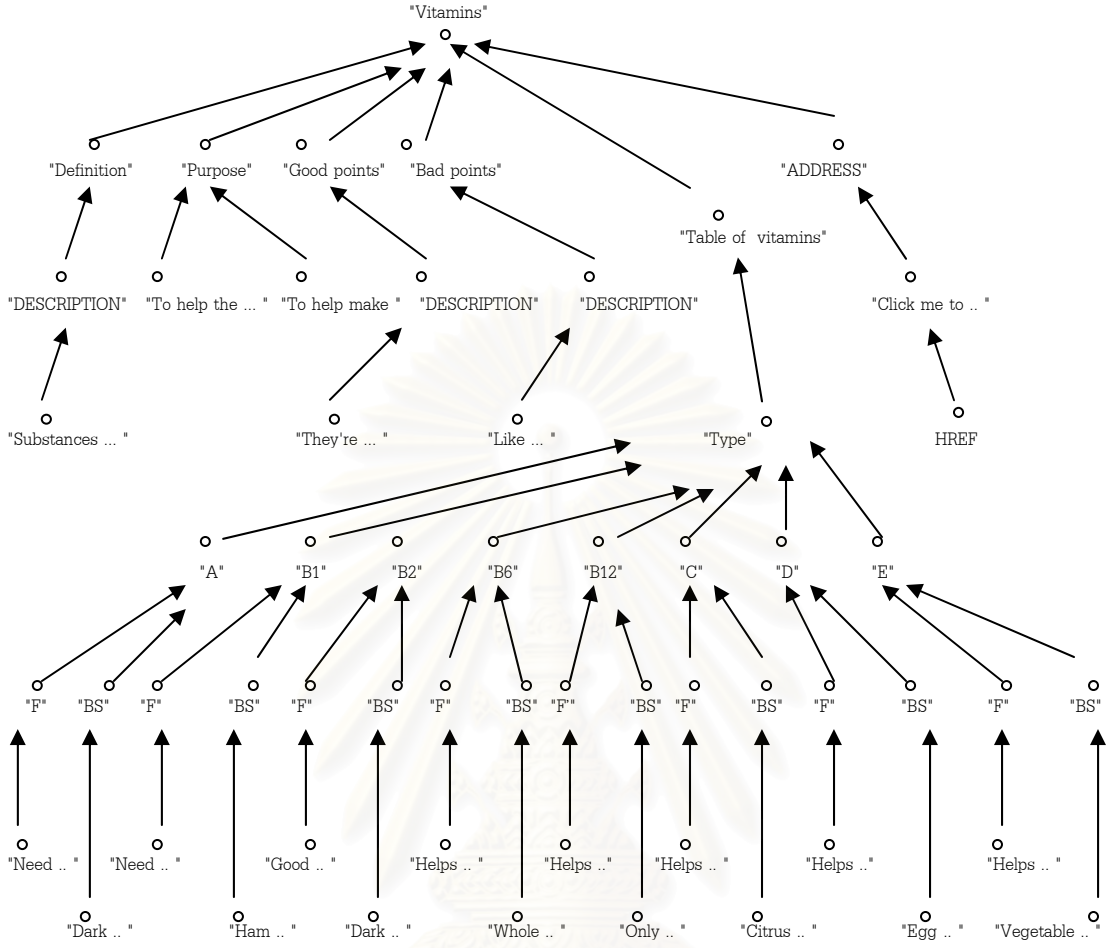
รูปที่ 5.7 แสดงเอกสาร HTML ที่มีโครงสร้างซับซ้อนในแฟ้มเอกสารชื่อ vitamin.html

เมื่อนำเอกสาร HTML จากรูปที่ 5.7 มาแปลงเป็น DOH โดยใช้เงื่อนไข container-content constraint จะได้ tree ดังรูปที่ 5.8



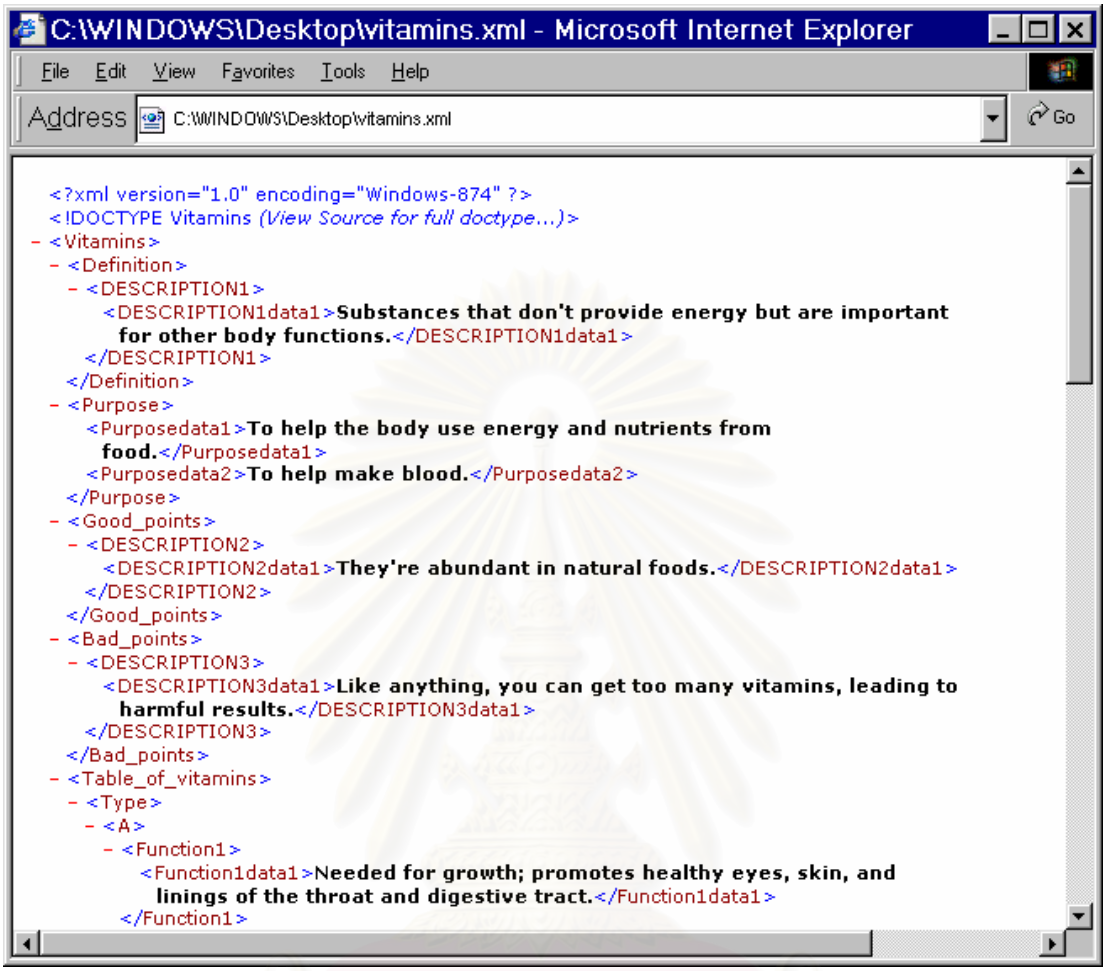
รูปที่ 5.8 โครงสร้าง DOH ที่ได้จาก vitamin.html (รูปที่ 5.7)

หลังจากที่นำ DOH ที่ได้จากรูปที่ 5.8 มาผ่านกระบวนการแปลงด้วย rules ต่างๆ ในขั้นตอนสุดท้าย ก็จะได้ DAH ดังรูปที่ 5.9



รูปที่ 5.9 DOH ที่ผ่านกระบวนการต่างๆจนได้เป็น DAH

และเมื่อนำ DAH tree ที่ได้ไปสร้างเป็นเอกสาร XML ก็จะได้ดังรูปที่ 5.10 ซึ่งเอกสารสมบูรณ์ได้แสดงในภาคผนวก ข



รูปที่ 5.10 เอกสาร XML บางส่วนที่ถูกแปลงจาก DAH ที่ได้จากรูปที่ 5.9

บทที่ 6

ผลการทดลอง

6.1 ผลการวิเคราะห์

ในการวิเคราะห์ผลการทดลองเราสนใจในเรื่องของความถูกต้องและความซับซ้อนของเอกสาร XML ที่ได้เพราะเอกสาร XML ที่มีความเหมาะสมที่จะนำไปใช้จะต้องเป็นเอกสารที่มีความถูกต้อง สามารถแสดงผลบน web browser ได้และเป็นเอกสาร XML ที่มีโครงสร้างไม่ซับซ้อน ทำให้สะดวกในการเข้าถึงข้อมูลและนำข้อมูลไปใช้ซึ่งมีรายละเอียดดังนี้

6.1.1 ความถูกต้องของเอกสาร XML ที่ได้

จากการทดลองผู้วิจัยไม่พบความผิดพลาดในการแสดงผลของเอกสาร XML ที่ได้บน web browser ซึ่งแสดงให้เห็นว่าโปรแกรมที่พัฒนาขึ้นมีความถูกต้องน่าเชื่อถือ และสิ่งนี้ชี้ให้เห็นว่าข้อกำหนดที่สร้างขึ้นจากภาษา Z มีความถูกต้องจึงทำให้โปรแกรมที่ได้มีความถูกต้อง เชื่อถือได้ และ ทำให้เราสามารถพัฒนาโปรแกรมขึ้นมาโดยไม่ต้องลองผิดลองถูก

6.1.2 ความซับซ้อนของเอกสาร XML ที่ได้

จากการทดลองผู้วิจัยพบว่ายิ่งเอกสาร HTML มีขนาดใหญ่และเป็นเอกสารที่ไม่มีโครงสร้างมากเท่าใด ก็ยิ่งทำให้เอกสาร XML ที่ได้มีขนาดใหญ่และเป็นเอกสารที่ขาดโครงสร้างที่ดีตามไปด้วย คือมีรูปแบบในการเก็บข้อมูลไม่ซ้ำกันเป็นชุดๆหรือ record ทำให้ไม่สะดวกต่อการนำไปใช้

6.2 ผลการวิเคราะห์ปัจจัย

ในหัวข้อนี้จะกล่าวถึงปัจจัยที่มีผลกระทบต่อความถูกต้องหรือความซับซ้อนของเอกสาร XML ที่ได้จากการทดลองซึ่งได้แก่ขนาดของเอกสาร HTML ที่เป็น input, tag ประเภทใหม่ๆที่ไม่ปรากฏในข้อกำหนดและการใช้ภาษา Z ในการสร้างข้อกำหนดของระบบงาน ซึ่งมีรายละเอียดดังนี้

6.2.1 ขนาดของเอกสาร HTML ที่เป็น input

จากการทดลองพบว่าขนาดของเอกสาร HTML ที่เป็น input ไม่มีผลต่อความถูกต้องของเอกสาร XML ที่ได้ แต่อาจทำให้ดูซับซ้อนขึ้นหากเอกสาร HTML ที่เป็น input มีขนาดใหญ่ขึ้น

6.2.2 การใช้ภาษา Z

เนื่องจากการเขียนข้อกำหนดของระบบงานนี้เป็นภาษา Z ซึ่งเป็นภาษาที่ใช้สัญลักษณ์ทางคณิตศาสตร์เช่น เซต, ฟังก์ชัน และ ตรรกะ ทำให้ข้อกำหนดที่ได้มีความถูกต้อง พิสูจน์ได้โดยวิธีทางคณิตศาสตร์ จึงทำให้โปรแกรมการแปลงเอกสาร HTML เป็นเอกสาร XML ที่พัฒนาขึ้นจากข้อกำหนดมีความถูกต้องน่าเชื่อถือและทำให้เอกสาร XML ที่ได้มีความถูกต้องตามข้อกำหนดด้วย

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการวิจัย

เราได้ทำการพัฒนาระเบียบวิธีสำหรับการแปลงเอกสาร HTML เป็นเอกสาร XML ภาษา XML เป็นภาษาที่เน้นเรื่องโครงสร้างข้อมูล สามารถสร้าง tag เพิ่มเติมและอธิบายความหมายได้ด้วยตัวเอง จำนวนองค์กรที่นำเอกสาร XML มาใช้มีปริมาณเพิ่มขึ้นอย่างรวดเร็ว ในขณะที่เอกสารบน web ยังคงใช้ภาษา HTML กันอยู่ การใช้ภาษาทั้ง 2 ระบบอาจทำให้ไม่สะดวกในการติดต่อสื่อสาร การกำจัดเอกสาร HTML ที่ใช้กันอยู่อาจดูเป็นการสิ้นเปลืองทรัพยากรที่ได้ลงทุนลงแรงไป ผู้วิจัยจึงมีแนวคิดที่จะสร้างเอกสาร XML จากเอกสาร HTML เพื่อตอบสนองต่อความต้องการที่อาจเกิดขึ้นในอนาคต

กระบวนการแปลงเริ่มจากการแปลงเอกสาร HTML ตามขั้นตอนวิธี [6] ให้อยู่ในรูปของต้นไม้ (tree) แล้วจึงแปลงต้นไม้ที่ได้ให้เป็นเอกสาร XML หากเป็นการพัฒนาโปรแกรมโดยทั่วไป เราจะไม่สามารถแน่ใจได้เลยว่าโปรแกรมนั้นๆมีความถูกต้องและเชื่อถือได้หรือไม่ เพราะว่าผู้เขียนโปรแกรมโดยทั่วไปมักเขียนโปรแกรมด้วยการลองผิดลองถูก แต่ในงานวิจัยนี้เราสร้างข้อกำหนดซึ่งอธิบายการทำงานของระบบขึ้นมาก่อนด้วยภาษา Z ภาษา Z คือ ภาษาที่ใช้สัญลักษณ์ทางคณิตศาสตร์เช่น เซต, ฟังก์ชัน และ ตรรกะ ซึ่งทำให้ข้อกำหนดของระบบการแปลงนี้มีความชัดเจน ไม่คลุมเครือ เป็นผลให้โปรแกรมที่เราพัฒนาขึ้นมามีความถูกต้อง เชื่อถือได้ และไม่ต้องลองผิดลองถูก

7.2 ข้อเสนอแนะ

1. เพิ่มความสามารถในการแปลงเอกสาร HTML ที่สร้างโดย version ที่ใหม่กว่า 2.0 ซึ่งหมายถึงการเพิ่มกฎ ต่างๆที่เหมาะสมเพื่อคงความถูกต้องของขั้นตอนการสร้าง DOH และ DAH trees
2. พัฒนาข้อกำหนดของการแปลงเอกสาร HTML เป็นเอกสาร XML ใหม่ให้แต่ละ schema มีขนาดเล็กลงและกะทัดรัดกว่านี้เพื่อที่จะได้ง่ายต่อการทำความเข้าใจและสะดวกต่อการพิสูจน์
3. แสดงความสัมพันธ์ระหว่างขนาดของเอกสาร HTML กับ time complexity ของการแปลงเอกสาร
4. พัฒนาโปรแกรมให้ครอบคลุม tag ใหม่ๆของ HTML ที่ไม่ได้คลอบคลุมในงานวิจัยนี้

รายการอ้างอิง

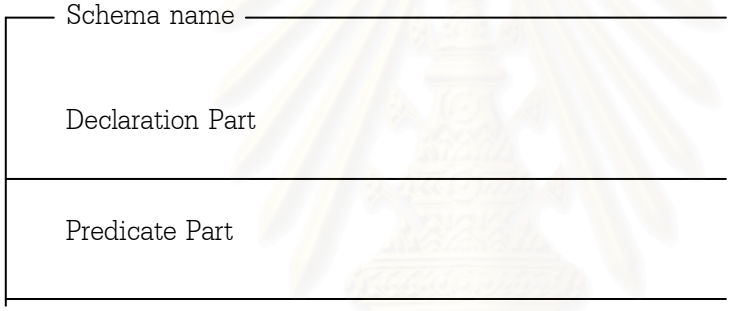
1. ขยัน จันทรสถาพร. เรียนลัด XML ฉบับรู้เต็มร้อย. กรุงเทพมหานคร : บริษัท เอ อาร์ อินฟอร์เมชัน แอนด์ พับลิเคชัน จำกัด, 2544.
2. ยัง, ไมเคิล เจ. XML Step by Step ฉบับภาษาไทย. แปลโดย ชวลิต จิรทีปติสุนทร. กรุงเทพมหานคร : สำนักพิมพ์สามย่าน.com, 2543.
3. ชัยดำรงค์ อุทิรัมย์. ปฏิบัติการเทคโนโลยีเว็บสุดร้อน. กรุงเทพมหานคร : สำนักพิมพ์สามย่าน.com, 2543.
4. สราวุธ อ้อยศรีสกุล. เริ่มคิด-เริ่มสร้าง-เริ่มใช้ XML. กรุงเทพมหานคร : บริษัท วิตตี้ กรุ๊ป จำกัด, 2544.
5. Berner-Lee, T., and Connolly, D. HTML 2.0 Materials [Online]. 1995. Available from : http://ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt [2002, June 5]
6. Lim, S.-J., and Ng, Y.-K. A Heuristic Approach for Converting HTML Documents to XML Documents. [Online]. 2000. Available from : http://lunar.cs.byu.edu/paper.html [2002, June 5]
7. Norcliffe, A., and Slater, G. MATHEMATICS OF SOFTWARE CONSTRUCTION. Chichester : Prentice Hall, 1991.
8. Ouahid, H., and Karmouch, A. Converting Web Pages into Well-formed Documents. [Online]. 2000. Available from : http://home.postech.ac.kr/~takyjin/xml/paper [2002, June 5]
9. Potter, B., Sinclair J., and Till D. An introduction to formal specification and Z. Hemel Hempstead : Prentice Hall. Hemel Hempstead, 1991.
10. Spivey, J.M. The Z Notation: A reference manual. Hemel Hempstead : Prentice Hall, 1989.

ภาคผนวก

ภาคผนวก ก
ความรู้เพิ่มเติมเกี่ยวกับภาษา Z

1. ภาษา Z

ภาษา Z (Z language) ถูกเสนอเป็นครั้งแรก โดย J.R. Abrial และต่อมาได้รับการพัฒนาจาก the University of Oxford ภาษา Z คือ ภาษาเชิงรูปนัย (formal language) ซึ่งใช้สัญลักษณ์ทางคณิตศาสตร์ เช่น เซต, ฟังก์ชัน และ ตรรกะ เป็นต้น โดยเราใช้ภาษา Z เพื่ออธิบายข้อกำหนดของโปรแกรมหรือระบบงาน ว่าทำอะไร แต่ไม่ได้บอกว่าทำอย่างไร ข้อกำหนดที่ถูกเขียนด้วย Z เรียกว่า สกีมา (schema) ซึ่งมีโครงสร้างคล้ายกล่อง (boxlike structure) ถ้าจะเปรียบเทียบก็จะคล้ายๆกับ subroutine หรือ procedure สกีมา (schema) ประกอบด้วยส่วนย่อยๆ 2 ส่วน ส่วนแรกคือส่วนประกาศ (Declaration part) ซึ่งจะประกาศตัวแปรที่ใช้ ส่วนที่ 2 คือ ส่วนที่บอกการทำงาน (Predicate part) จะแสดงถึงความสัมพันธ์ระหว่างตัวแปรที่กล่าวในส่วนประกาศดังรูป



2. ทำไมต้องใช้ Z SPECIFICATIONS

เหตุผลที่ควรจะใช้ภาษา Z ในการอธิบายข้อกำหนดของระบบงานมีดังนี้

2.1 บอกได้อย่างชัดเจนว่าแต่ละระบบงานที่กล่าวถึงทำอะไร เพราะรายละเอียดของงานกล่าวถึงโดยใช้สัญลักษณ์ทางคณิตศาสตร์

2.2 การตรวจสอบและแก้ไขรายละเอียดของระบบงานทำได้ง่าย

2.3 รายละเอียดของระบบงานเป็นที่เข้าใจตรงกันเนื่องจากข้อกำหนดที่เขียนด้วย Z เป็นมาตรฐานสากล

2.4 ทำให้รายละเอียดของระบบงานมีความกระชับ

3. การเขียน ภาษา Z โดยใช้ตัวอย่างประกอบ

เพื่อการทำความเข้าใจให้มากขึ้น จะยกตัวอย่างประกอบ 2 ตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 1 รายละเอียดของโปรแกรมเพื่อหาผลลัพธ์ จากการบวกเลขจำนวนเต็ม 2 จำนวน

AddNumber

op? : char

x?,y?,s! : $\mathbb{Z}$

result! : { operation_completed , operation_not_completed }

(x?$\in \mathbb{Z}$ $\wedge$ y?$\in \mathbb{Z}$ $\wedge$ op? = "+" $\wedge$ s! = x?+y? $\wedge$

result! = operation_completed) $\vee$

((x?$\notin \mathbb{Z}$ $\vee$ y?$\notin \mathbb{Z}$ $\vee$ op?$\neq$"+") $\wedge$

result! = operation_not_completed)

schema นี้ชื่อ Addnumber ในส่วน Declaration part มีตัวแปรอยู่ 5 ตัวคือ op มีประเภทของตัวแปรเป็น char ซึ่งในที่นี้หมายถึงตัวอักษรในรหัส ASCII, x,y,s เป็นตัวแปรจำนวนเต็ม ($\mathbb{Z}$) และ result ซึ่งมีค่าที่เป็นไปได้อยู่ 2 ค่าคือ operation_completed และ operation_not_completed เครื่องหมาย ? และ ! ที่ตามหลังตัวแปรเป็นการบอกว่าตัวแปรนั้นเป็น input และ output ตามลำดับ ในส่วนของ Predicate part จะมีการตรวจสอบว่าตัวแปร x และ y ที่รับเข้ามานั้นเป็นจำนวนเต็มหรือไม่รวมทั้งตรวจสอบว่า operation ที่กระทำระหว่าง x และ y เป็นเครื่องหมายบวกหรือไม่ ถ้าใช่ก็ให้เก็บผลรวมระหว่าง x และ y ไว้ในตัวแปร s รวมทั้งให้ส่งข้อความเพื่อบอกว่าระบบทำงานสมบูรณ์ในที่นี้คือแสดงด้วยตัวแปร result ซึ่งแสดงข้อความว่าการทำงานสมบูรณ์ (operation_completed) แต่ถ้าตัวแปร x หรือ y ที่รับเข้ามานั้นไม่เป็นจำนวนเต็มหรือ operation ที่กระทำระหว่าง x และ y ไม่ใช่เครื่องหมายบวกให้ส่งข้อความบอกผู้ใช้ว่าการทำงานไม่สมบูรณ์นั่นคือ result จะแสดงข้อความว่าการทำงานไม่สมบูรณ์ (operation_not_completed)

<u>ตัวอย่างที่ 2</u> รายละเอียดของระบบงานที่เกี่ยวกับวันเกิดของคน

BirthdayBook

known : P NAME

birthday : NAME$\rightarrow$DATE

---

known = dom birthday

schema BirthdayBook ทำหน้าที่เป็นแหล่งเก็บข้อมูล ในที่นี้ข้อมูลเป็นสมาชิกคู่อันดับที่ map จากเซตของชื่อ (NAME) ไปยังวันเกิด (DATE)

ในส่วนของ Declaration Part มีตัวแปร 2 ตัวคือมี known เป็นเซตของชื่อ (NAME) โดย P NAME คือ power set ของ NAME และ birthday เป็น function ที่ map จากชื่อ (NAME) ไปยังวันเกิด (DATE)

ในส่วนของ Predicate Part มีการประกาศว่า known เป็น domain ของ function birthday ในที่นี้ dom ก็คือ domain นั่นเอง

AddBirthday

ΔBirthdayBook

name? : NAME

date? : DATE

---

name? $\notin$ known

birthday' = birthday $\cup$ {name? $\rightarrow$ date?}

schema AddBirthday ทำงานด้วยการเพิ่มชื่อและวันเกิดของคนที่เป็น input เข้าไปในระบบในกรณีที่ยังไม่มีข้อมูลนั้นๆอยู่ในระบบ

ในส่วนของ Declaration Part มีการประกาศ ΔBirthdayBook ซึ่งหมายความว่าข้อมูลใน BirthdayBook มีการเปลี่ยนแปลง สัญลักษณ์ Δ ที่นำหน้าระบบใดๆ หมายความว่าระบบนั้นๆมีการเปลี่ยนแปลง name และ date เป็น input ซึ่งมีประเภทของตัวแปรเป็น NAME และ DATE ตามลำดับ

ในส่วนของ Predicate Part จะมีการตรวจสอบว่าชื่อที่เป็น input นั้นมีอยู่ในระบบ (BirthdayBook) ก่อนแล้วหรือไม่ ถ้าไม่มีจึงค่อยเพิ่มชื่อและวันเกิดของคนๆนั้นเข้าไปในระบบ ซึ่งมีผลให้ระบบเปลี่ยนแปลงคือมีจำนวนสมาชิกเพิ่มขึ้น ในที่นี้ birthday คือแสดงถึงระบบก่อนการเพิ่มข้อมูล และ birthday' แสดงถึงระบบหลังการเพิ่มข้อมูล

FindBirthday

ΞBirthdayBook

name? : NAME

date! : DATE

---

name? $\in$ known

date! = birthday(name?)

schema FindBirthday มีจุดประสงค์เพื่อการค้นหาว่าวันเกิดของคนๆหนึ่งซึ่งจะใส่ชื่อเป็น input นั้นว่าเกิดวันไหน

ในส่วนของ Declaration Part มีการประกาศ ΞBirthdayBook ซึ่งหมายความว่าข้อมูลใน BirthdayBook ไม่มีการเปลี่ยนแปลง สัญลักษณ์ Ξ ที่นำหน้าระบบใดๆ หมายความว่าระบบนั้นๆไม่มีการเปลี่ยนแปลง name และ date เป็น input ซึ่งมีประเภทของตัวแปรเป็น NAME และ DATE ตามลำดับ

ในส่วนของ Predicate Part จะมีการตรวจสอบว่าชื่อ (name) ที่เป็น input ที่ต้องการค้นหาวันเกิดนั้นมีอยู่ในระบบ (BirthdayBook) ก่อนแล้วหรือไม่เพราะถ้าไม่มีก็ไม่สามารถค้นหาวันเกิดของคนๆนั้นได้ แต่ถ้ามีก็ให้แสดงวันเกิดของคนๆนั้นในที่นี้คือตัวแปร date ซึ่งหามาจากการนำชื่อ (name) ซึ่งเป็น input ใส่เข้าไปใน function birthday

4. การพิสูจน์ความถูกต้องของข้อกำหนดของระบบงาน

หลังจากเราได้สร้างข้อกำหนดของระบบงานเรียบร้อยแล้ว เมื่อนำไปพัฒนาโปรแกรมทันทีอาจเกิดปัญหาอันเนื่องมาจากข้อกำหนดที่ไม่สมบูรณ์ เนื่องจากการสร้างข้อกำหนดโดยใช้ภาษาพูดอาจมีความกำกวมและไม่สามารถแสดงได้ว่าข้อกำหนดนั้นถูกหรือผิดอย่างไร แต่ภาษา Z สามารถแก้ปัญหาในส่วนนี้ได้เพราะนิยามของภาษา Z เองก็ใช้สัญลักษณ์ทางคณิตศาสตร์จึงสามารถแสดงได้ว่าข้อกำหนดที่สร้างขึ้นมานั้นถูกหรือผิดอย่างไร

จากการพิจารณาข้อกำหนดที่ผ่านมาเราคาดว่าในกรณีที่ name? $\notin$ known เราจะได้ว่า known' = known $\cup$ {name?} และในกรณีที่ name? $\in$known จะได้ว่า known' = known ซึ่งสามารถพิสูจน์ได้ดังนี้

4.1 ในกรณีที่ name? $\notin$ known เราจะพิสูจน์ว่า known' = known $\cup$ {name?}

เราสามารถพิสูจน์โดยอาศัยข้อกำหนดของ schema BirthdayBook, schema AddBirthday และกฎอีก 2 ข้อโดยกฎ 2 ข้อที่ใช้ในการพิสูจน์ คือ

1. dom (f $\cup$ g) = (dom f) $\cup$ (dom g)

2. dom {a$\rightarrow$b} = {a}

ซึ่งสามารถพิสูจน์ได้ดังนี้

known' = dom birthday'	จาก schema BirthdayBook
= dom(birthday $\cup$ {name? $\rightarrow$ date?})	จาก schema AddBirthday
= dom birthday $\cup$ dom {name? $\rightarrow$ date?}	จากกฎข้อที่ 1
= dom birthday $\cup$ {name?}	จากกฎข้อที่ 2
= known $\cup$ {name?}	จาก schema BirthdayBook

4.2 ในกรณีที่ name? $\in$known เราจะพิสูจน์ว่า known' = known

ซึ่งสามารถพิสูจน์ได้ดังนี้

known' = dom birthday'	จาก schema BirthdayBook
= dom birthday	เนื่องจากข้อมูลในระบบไม่มีการเปลี่ยนแปลง

= known จาก schema BirthdayBook

การพิสูจน์เช่นนี้ทำให้เรามั่นใจได้ว่าการให้ข้อกำหนดที่ผ่านมาจะไม่สร้างปัญหาในตอนของการ implementation

ภาคผนวก ข

เอกสาร HTML ที่เป็นข้อมูลเข้าและเอกสาร XML ที่ได้จากการแปลง

1. เอกสาร HTML แบบสมบูรณ์ที่เป็นข้อมูลเข้าของการทดลองในหัวข้อที่ 5.2

VITAMINS

Definition

Substances that don't provide energy but are important for other body functions.

Purpose

- To help the body use energy and nutrients from food.
- To help make blood.

Good points

They're abundant in natural foods.

Bad points

Like anything, you can get too many vitamins, leading to harmful results.

Table of vitamins

Type	Function	Best Source
A	Needed for growth; promotes healthy eyes, skin, and linings of the throat and digestive tract.	Dark yellow orange, and dark green vegetables and fruits, such as spinach and cantaloupe; eggs; low-fat cheese.
B1	Needed for the nervous system; helps the body get energy from food.	Ham; oysters; whole-grain and enriched cereals, pasta, and bread; oatmeal; peas and lima beans.
B2	Good for the skin; helps the body use oxygen.	Dark green vegetables; eggs; whole-grain and enriched breads, pasta, and cereals; mushroom; dried ligament; skim milk; low-fat meat.
B6	Helps body absorb protein.	Whole-grain cereals and breads; spinach; green beans; bananas; fish; poultry; potatoes.

B12	Helps the body use protein, fat, and carbohydrates, and make red blood cells.	Only in animal foods; low-fat meat and milk; fish; oyster.
C	Helps keep gums healthy; holds the body cells together.	Citrus fruits; tomatoes; strawberries; potatoes; green peppers; other dark green vegetables.
D	Helps body absorb calcium for strong bones and teeth.	Eggs; low-fat milk; salmon; tuna.
E	Helps make red blood cells, muscles, and other tissues; protects vitamin A.	Vegetable oils; whole-grain cereals and breads; dried beans; green, leafy vegetables.

Click me to get more information.

2. **เอกสาร XML แบบสมบูรณ์ที่ได้จากการแปลงเอกสาร HTML ในข้อ 1.**

```
<?xml version="1.0" encoding="Windows-874" ?>
 <!DOCTYPE Vitamins (View Source for full doctype...)>
- <Vitamins>
- <Definition>
- <DESCRIPTION1>
  <DESCRIPTION1data1>Substances that don't provide energy but are important for other
    body functions.</DESCRIPTION1data1>
    </DESCRIPTION1>
    </Definition>
- <Purpose>
  <Purposedata1>To help the body use energy and nutrients from food.</Purposedata1>
  <Purposedata2>To help make blood.</Purposedata2>
    </Purpose>
- <Good_points>
- <DESCRIPTION2>
  <DESCRIPTION2data1>They're abundant in natural foods.</DESCRIPTION2data1>
    </DESCRIPTION2>
    </Good_points>
- <Bad_points>
- <DESCRIPTION3>
```

<DESCRIPTION3data1>**Like anything, you can get too many vitamins, leading to harmful results.**</DESCRIPTION3data1>
</DESCRIPTION3>
</Bad_points>
- <Table_of_vitamins>
- <Type>
- <A>
- <Function1>
<Function1data1>**Needed for growth; promotes healthy eyes, skin, and linings of the throat and digestive tract.**</Function1data1>
</Function1>
- <Best_Source1>
<Best_Source1data1>**Dark yellow orange, and dark green vegetables and fruits, such as spinach and cantaloupe; eggs; low-fat cheese.**</Best_Source1data1>
</Best_Source1>
</A>
- <B1>
- <Function2>
<Function2data1>**Needed for the nervous system; helps the body get energy from food.**</Function2data1>
</Function2>
- <Best_Source2>
<Best_Source2data1>**Ham; oysters; whole-grain and enriched cereals, pasta, and bread; oatmeal; peas and lima beans.**</Best_Source2data1>
</Best_Source2>
</B1>
- <B2>
- <Function3>
<Function3data1>**Good for the skin; helps the body use oxygen.**</Function3data1>
</Function3>
- <Best_Source3>
<Best_Source3data1>**Dark green vegetables; eggs; whoile-grain and enriched breads, pasta, and cereals; mushroom; dried legment; skim milk; low-fat meat.**</Best_Source3data1>

</Best_Source3>

</B2>

- <B6>

- <Function4>

<Function4data1>**Helps body absorb protien.**</Function4data1>

</Function4>

- <Best_Source4>

<Best_Source4data1>**Whole-grain cereals and breads; spinach; green beans; bananas; fish; poultry; potatoes.**</Best_Source4data1>

</Best_Source4>

</B6>

- <B12>

- <Function5>

<Function5data1>**Helps the body use protien, fat, and carbohydrates, and make red blood cells.**</Function5data1>

</Function5>

- <Best_Source5>

<Best_Source5data1>**Only in animal foods; low-fat meat and milk; fish; oyster.**</Best_Source5data1>

</Best_Source5>

</B12>

- <C>

- <Function6>

<Function6data1>**Helps keep gums healthy; holds the body cells together.**</Function6data1>

</Function6>

- <Best_Source6>

<Best_Source6data1>**Citrus fruits; tomatoes; strawberries; potatoes; green peppers; other dark green vegetables.**</Best_Source6data1>

</Best_Source6>

</C>

- <D>

- <Function7>

<Function7data1>**Helps body absorb calcium for strong bones and teeth.**</Function7data1>

</Function7>
- <Best_Source7>
<Best_Source7data1>**Eggs; low-fat milk; salmon; tuna.**</Best_Source7data1>
</Best_Source7>
</D>
- <E>
- <Function8>
<Function8data1>**Helps make red blood cells, muscles, and other tissues; protects vitamin A.**</Function8data1>
</Function8>
- <Best_Source8>
<Best_Source8data1>**Vegetable oils; whole-grain cereals and breads; dried beans; green, leafy vegetables.**</Best_Source8data1>
</Best_Source8>
</E>
</Type>
</Table_of_vitamins>
- <ADDRESS>
- <Click_me_to_get_more_information>
<Click_me_to_get_more_informationdata1>**www.vitamin.com**</Click_me_to_get_more_informationdata1>
</Click_me_to_get_more_information>
</ADDRESS>
</Vitamins>

ภาคผนวก ค

source code ของโปรแกรมที่ทำการแยกโครงสร้างของเอกสาร HTML

```
#include <stdio.h>
#define Length    30
#define Num       10
int main(void)
{
   FILE *fp,*fi;
   char s[Length],c,tag=1;
   char *list[Length];
   int i,j=0,k,count=0;
   clrscr();
   for(i=0;i<Num;i++)
   {
      if ( (list[i]=malloc(Length*sizeof(char))) == NULL )
      {
            printf("Out of memory\n");
            return;
      }
   }
   i=0;
   fp=fopen("h.htm","r");
   fi=fopen("tree.dat","w");
   while ( (c=getc(fp)) !=EOF)
   {
      if (tag==0 && c!='<'&& c!='\n') s[i++]=c;
      if (c=='<')
      {
           if (i!=0)
           {
                    s[i]='\0';
                    fprintf(fi,"%30s%30s\n",list[j-1],s);
                    i=0;
           }
           tag=1; s[i++]=c;
      }
```

```
    if (tag==1 && c!='<' && c!='>') s[i++]=c;
    if (c=='>')
    {
        tag=0; s[i++]=c; s[i]='\0';
        if (s[1]!='/')
        {
                if(j!=0) { fprintf(fi,"%30s%30s\n",list[j-1],s); }
                strcpy(list[j++],s);
        }
        else
        {
                j--;
        }
        i=0;
    }
  } /* end while */
  fclose(fp);
  fclose(fi);
  getch();
}
```

ภาคผนวก ง

ตัวอย่างการพิสูจน์ความถูกต้องของข้อกำหนด

1. การพิสูจน์ความถูกต้องของ schema DefineRoot

จาก schema DefineRoot เมื่อเราสนใจในกรณีที่ $i_j \neq <>$ และ $e_j \neq <>$ เราจะได้ว่า domain ของฟังก์ชัน parent จะอยู่ในรูปของ $d_1' = d_1 \cup \{ i_j \}$, $d_1' = d_1 \cup \{ e_j \}$ และ เมื่อเราสนใจในกรณีที่มีการเพิ่ม attribute เราจะได้ว่า domain ของฟังก์ชัน attrib จะอยู่ในรูปของ $d_2' = d_2 \cup \{ g \}$ เมื่อ g คือ tag ที่เราต้องการเพิ่ม attribute ของมันเข้าไปในระบบของ ATB

1.1 จะพิสูจน์ว่า $d_1' = d_1 \cup \{ i_j \}$

$d_1' = \text{dom parent}'$	จาก schema DOH
$= \text{dom} (\text{parent} \cup \{ i_j \rightarrow t \})$	จาก schema DefineRoot
$= \text{dom parent} \cup \text{dom} \{ i_j \rightarrow t \}$	จากสมบัติของ dom
$= \text{dom parent} \cup \{ i_j \}$	จากสมบัติของ dom
$= d_1 \cup \{ i_j \}$	จาก schema DOH

1.2 จะพิสูจน์ว่า $d_1' = d_1 \cup \{e_j \}$

$d_1' = \text{dom parent}'$	จาก schema DOH
$= \text{dom} (\text{parent} \cup \{ e_j \rightarrow t \})$	จาก schema DefineRoot
$= \text{dom parent} \cup \text{dom} \{ e_j \rightarrow t \}$	จากสมบัติของ dom
$= \text{dom parent} \cup \{ e_j \}$	จากสมบัติของ dom
$= d_1 \cup \{ e_j \}$	จาก schema DOH

1.3 จะพิสูจน์ว่า $d_2' = d_2 \cup \{ g \}$

$d_1' = \text{dom attrib}'$	จาก schema ATB
$= \text{dom} (\text{attrib} \cup \{ g \rightarrow a \})$	จาก schema DefineRoot
$= \text{dom attrib} \cup \text{dom} \{ g \rightarrow a \}$	จากสมบัติของ dom
$= \text{dom attrib} \cup \{ g \}$	จากสมบัติของ dom
$= d_2 \cup \{ g \}$	จาก schema ATB

2. การพิสูจน์ความถูกต้องของ schema CompleteDOH

จาก schema CompleteDOH ในกรณีที่ element ที่ยังไม่ได้กระจายโครงสร้างซึ่งมี tag A เป็นตัวบรรจุ (container) เมื่อเราสนใจในกรณีที่ $i_j \neq <> $ และ $e_j \neq <>$ เราจะได้ว่า domain ของฟังก์ชัน parent จะอยู่ในรูปของ $d_1' = d_1 \cup \{ i_j \}$ และ $d_1' = d_1 \cup \{ e_j \}$ และจะไม่มี e เป็น domain ของฟังก์ชัน parent อีก แต่มี tag A ที่ต่อเชื่อมกับ attribute ซึ่งในที่นี้คือ front (tail o) เป็นสมาชิกใหม่แทน นั่นคือในฟังก์ชัน parent ไม่มี $\{ e \rightarrow p\}$ เป็นสมาชิกและจะได้ว่า $d_1' = d_1 \cup \{$ front (tail o) $\}$

ดังนั้นเราจะพิสูจน์ว่า $d_1' = d_1 \cup \{ i_j \}$, $d_1' = d_1 \cup \{ e_j \}$ และ $d_1' = d_1 \cup \{$ front (tail o) $\}$

2.1 จะพิสูจน์ว่า $d_1' = d_1 \cup \{ i_j \}$

d_1' = dom parent' จาก schema DOH

= dom (parent $\cup \{ i_j \rightarrow$ front (tail o) $\}$) จาก schema CompleteDOH

= dom parent $\cup$ dom $\{ i_j \rightarrow$ front (tail o) $\}$ จากสมบัติของ dom

= dom parent $\cup \{ i_j \}$ จากสมบัติของ dom

= $d_1 \cup \{ i_j \}$ จาก schema DOH

2.2 จะพิสูจน์ว่า $d_1' = d_1 \cup \{e_j \}$

d_1' = dom parent' จาก schema DOH

= dom (parent $\cup \{ e_j \rightarrow$ front (tail o) $\}$) จาก schema CompleteDOH

= dom parent $\cup$ dom $\{ e_j \rightarrow$ front (tail o) $\}$ จากสมบัติของ dom

= dom parent $\cup \{ e_j \}$ จากสมบัติของ dom

= $d_1 \cup \{ e_j \}$ จาก schema DOH

2.3 จะพิสูจน์ว่า $d_1' = d_1 \cup \{$ front (tail o) $\}$

d_1' = dom parent' จาก schema DOH

= dom (parent $\cup \{$ front (tail o) $\rightarrow$ t $\}$) จาก schema CompleteDOH

= dom parent $\cup$ dom $\{$ front (tail o) $\rightarrow$ t $\}$ จากสมบัติของ dom

= dom parent $\cup \{$ front (tail o) $\}$ จากสมบัติของ dom

= $d_1 \cup \{$ front (tail o) $\}$ จาก schema DOH

การพิสูจน์ในกรณีที่ element ที่ยังไม่ได้กระจายโครงสร้างซึ่งมีตัวบรรจุ (container) ไม่ใช่ tag A เมื่อเราสนใจในกรณีที่เราต้องการเพิ่ม attribute จะได้ว่า domain ของฟังก์ชัน attrib จะอยู่ในรูปของ d_2'=

$d_2 \cup \{ g \}$ เมื่อ g คือ tag ที่เราต้องการเพิ่ม attribute ของมันเข้าไปในระบบของ ATB และเมื่อเราสนใจในกรณีที่ $i_j \neq <>$ และ $e_j \neq <>$ เราจะได้ว่า domain ของฟังก์ชัน parent จะอยู่ในรูปของ $d_1' = d_1 \cup \{ i_j \}$, $d_1' = d_1 \cup \{e_j \}$ และ $d_1' = d_1 \cup \{ t \}$ เมื่อ t คือ tag ของ element ที่ถูกกระจายโครงสร้างซึ่งพิสูจน์ได้ดังนี้

2.4 จะพิสูจน์ว่า $d_2' = d_2 \cup \{ t \}$

$d_1' = \text{dom attrib}'$	จาก schema ATB
$= \text{dom} (\text{attrib} \cup \{ t \rightarrow a \})$	จาก schema CompleteDOH
$= \text{dom attrib} \cup \text{dom} \{ t \rightarrow a \}$	จากสมบัติของ dom
$= \text{dom attrib} \cup \{ t \}$	จากสมบัติของ dom
$= d_2 \cup \{ t \}$	จาก schema ATB

2.5 จะพิสูจน์ว่า $d_1' = d_1 \cup \{ i_j \}$

$d_1' = \text{dom parent}'$	จาก schema DOH
$= \text{dom} (\text{parent} \cup \{ i_j \rightarrow t \})$	จาก schema CompleteDOH
$= \text{dom parent} \cup \text{dom} \{ i_j \rightarrow t \}$	จากสมบัติของ dom
$= \text{dom parent} \cup \{ i_j \}$	จากสมบัติของ dom
$= d_1 \cup \{ i_j \}$	จาก schema DOH

2.6 จะพิสูจน์ว่า $d_1' = d_1 \cup \{e_j \}$

$d_1' = \text{dom parent}'$	จาก schema DOH
$= \text{dom} (\text{parent} \cup \{ e_j \rightarrow t \})$	จาก schema CompleteDOH
$= \text{dom parent} \cup \text{dom} \{ e_j \rightarrow t \}$	จากสมบัติของ dom
$= \text{dom parent} \cup \{ e_j \}$	จากสมบัติของ dom
$= d_1 \cup \{ e_j \}$	จาก schema DOH

2.7 จะพิสูจน์ว่า $d_1' = d_1 \cup \{ t \}$

$d_1' = \text{dom parent}'$	จาก schema DOH
$= \text{dom} (\text{parent} \cup \{ t \rightarrow p \})$	จาก schema CompleteDOH
$= \text{dom parent} \cup \text{dom} \{ t \rightarrow p \}$	จากสมบัติของ dom
$= \text{dom parent} \cup \{ t \}$	จากสมบัติของ dom
$= d_1 \cup \{ t \}$	จาก schema DOH

ประวัติผู้เขียนวิทยานิพนธ์

ชื่อ : ประยุทธ ลิมปกานนท์

ภูมิลำเนา : จังหวัดสุพรรณบุรี

การศึกษา : ปริญญาวิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์

ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ