พื้นฐานของระบบฐานข้อมูลเชิงวัตถุแบบอนุมาน

Foundations of Deductive Object-Oriented Database Systems

โครงการวิจัยองค์ความรู้ใหม่ที่เป็นพื้นฐานต่อการพัฒนา
สนับสนุนโดย
สำนักงานกองทุนสนับสนุนการวิจัย
(เลขที่โครงการ BRG14/2540)

คณะผู้วิจัย: นายวิลาศ วูวงศ์ (หัวหน้าโครงการ)
นายเอกวิชญ์ นันทจีวรวัฒน์ (นักวิจัย)
นางสาวชุติพร อนุตริยะ (นักวิจัย)
Professor Kiyoshi Akama (ที่ปรึกษา)

BRG 104 0014

BRG/14/2540-อ. วิลาศ วูวงศ์

พื้นฐานของระบบฐานข้อมูลเชิงวัตถุแบบอนุมาน

Foundations of Deductive Object-Oriented Database Systems

โครงการวิจัยองค์ความรู้ใหม่ที่เป็นพื้นฐานต่อการพัฒนา

สนับสนุนโดย

สำนักงานกองทุนสนับสนุนการวิจัย

(เลขที่โครงการ BRG14/2540)

คณะผู้วิจัย: นายวิลาศ วูวงศ์ (หัวหน้าโครงการ)

นายเอกวิชญ์ นันทจีวรวัฒน์ (นักวิจัย)

นางสาวชุติพร อนุตริยะ (นักวิจัย)

Professor Kiyoshi Akama (ที่ปรึกษา)

บทคัดย่อ

งานวิจัยนี้ใช้ทฤษฎีโปรแกรมเชิงประกาศในการสร้างทฤษฎีพื้นฐานของระบบฐานข้อมูลเชิงวัตถุแบบอนุมานซึ่งครอบคลุมลักษณะหลักทั้งหมดของระบบอันได้แก่การนิรนัย การจัดลำดับชั้นของข้อมูล การถ่ายทอดคุณสมบัติ และการกำหนดสารสนเทศโดยนัย อีกทั้งยังประยุกต์สร้างทฤษฎีพื้นฐานสำหรับฐานข้อมูลเอกสารXML ซึ่งถือเป็นระบบฐานข้อมูลเชิงวัตถุแบบอนุมานประเภทหนึ่ง ทฤษฎีพื้นฐานดังกล่าวสามารถใช้สร้างแบบจำลองข้อมูลของเอกสารXML สร้างแบบจำลองของ XML DTD และสร้างกลไกสำหรับประมวลผลการสอบถามฐานข้อมูล XML

Abstract

This research project employs Declarative Program theory to develop a theoretical foundation for Deductive Object-Oriented Databases (DOODs) which covers deduction, hierarchical classification, inheritance and implicit information. In order to apply the concepts gained, this research project also develops a theoretical framework for XML document databases, which are a kind of DOOD. The framework permits representation of XML documents, modeling of XML DTDs and formulation as well as evaluation of XML queries.

สารบัญ

1. บทนำ

แนวความคิดในการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน[*] (Deductive Object-Oriented Representation) เกิดขึ้นเมื่อประมาณปี พ.ศ. 2530 จากการผสมผสานกันระหว่างแนวความคิดในการจัดเก็บข้อมูลและความรู้แบบนิรนัย (Deductive Representation) กับแนวความคิดในการจัดเก็บข้อมูลเชิงวัตถุ (Object-Oriented Representation) โดยมีจุดประสงค์ที่จะรวมข้อดีของระบบทั้งสองแบบและชดเชยข้อด้อยซึ่งกันและกัน [1] แนวความคิดดังกล่าวนี้ได้รับการคาดหวังว่าจะเป็นแนวทางที่เหมาะสมสำหรับการขยายศักยภาพของระบบฐานข้อมูลและความรู้ให้สามารถรองรับความต้องการใหม่ๆ ที่จะเกิดขึ้นในงานประยุกต์หลากหลายลักษณะในอนาคต ตัวอย่างเช่น งานทางด้านปัญญาประดิษฐ์ งานทางด้านมัลติมิเดีย งานทางด้านวิศวกรรมและอุตสาหกรรม และงานทางด้านสารสนเทศทางภูมิศาสตร์ เป็นต้น

ตั้งแต่ปี พ.ศ. 2532 เป็นต้นมา นักวิจัยหลายกลุ่มในหลายประเทศได้ตีพิมพ์บทความเพื่อเสนอต้นแบบของระบบฐานข้อมูลและความรู้เชิงวัตถุแบบอนุมาน บทความเหล่านี้มีรูปแบบคล้ายกันอย่างหนึ่งคือ จะเริ่มต้นด้วยการเสนอไวยากรณ์ของภาษาที่ใช้ในการบรรยายข้อมูลและความรู้ ตามด้วยการกำหนดความหมายของโปรแกรมในภาษานั้นๆ โดยความหมายของโปรแกรมนี้จะเป็นตัวกำหนดเนื้อหาของข้อมูลและความรู้ในระบบ รูปแบบดังกล่าวนี้นำไปสู่อุปสรรคที่สำคัญอย่างหนึ่งในการศึกษาและพัฒนาต้นแบบของระบบ กล่าวคือ ลักษณะอย่างหนึ่งของระบบมักจะถูกกำหนดโดยใช้ไวยากรณ์ที่แตกต่างกันและมีความคลาดเคลื่อนกันในด้านของความหมายในระบบที่แตกต่างกัน ผลที่ตามมาก็คือการขาดรากฐานทางทฤษฎีร่วมกันสำหรับการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน รากฐานทางทฤษฎีดังกล่าวนี้มีความสำคัญอย่างยิ่งในการกำหนดความหมายที่ชัดเจน ไม่คลุมเครือของฐานข้อมูลและความรู้ การกำหนดเนื้อหาของข้อมูลและความรู้ในระบบ การศึกษาและวิเคราะห์ฐานข้อมูลและความรู้อย่างละเอียด ตลอดจนการพัฒนาซอฟต์แวร์สำหรับปรับปรุงประสิทธิภาพการจัดเก็บและประมวลผลข้อมูลและความรู้

เพื่อแก้ไขปัญหาทางทฤษฎีดังกล่าว คณะผู้วิจัยพยายามที่จะพัฒนาทฤษฎีที่เป็นอิสระจากไวยากรณ์ของภาษาสำหรับระบบการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน ทฤษฎีนี้จะไม่ใช้ไวยากรณ์เฉพาะของภาษาใดภาษาหนึ่งในการบรรยายถึงส่วนประกอบต่างๆ ของระบบ แต่จะพิจารณาส่วนประกอบต่างๆ ของระบบว่าเป็นส่วนประกอบนามธรรม (Abstract Component) ซึ่งจะถูกกำหนดลักษณะโดยการบรรยายความเกี่ยวเนื่องสัมพันธ์และผลกระทบต่อกันกับส่วนประกอบอื่นๆ ของระบบโดยใช้แบบจำลองทางคณิตศาสตร์ (Mathematical Model) ทฤษฎีนี้ควรจะครอบคลุมถึงลักษณะหลักทั้งหมดของระบบ อันได้แก่ การนิรนัย (Deduction) การจัดลำดับชั้นของข้อมูล (Hierarchical Classification) การถ่ายทอดคุณสมบัติ (Inheritance) และการกำหนดสารสนเทศโดยนัย (Implicit Information) ผลลัพธ์ต่างๆในทฤษฎีนี้จะต้องได้รับการตรวจสอบอย่างชัดเจน โดยใช้วิธีการพิสูจน์ทางคณิตศาสตร์ (Mathematical Proof Methods)

---

[*] คำ deductive ควรจะใช้ภาษาไทยว่า นิรนัย มากกว่า อนุมาน

2. วิธีการดำเนินการวิจัย

1) สร้างทฤษฎีสำหรับระบบการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน ทฤษฎีนี้จะไม่ใช้ไวยากรณ์เฉพาะของภาษาใดภาษาหนึ่งเป็นพื้นฐานในการกำหนดส่วนประกอบต่างๆ ของระบบ แต่จะใช้แบบจำลองทางคณิตศาสตร์ในการกำหนดความสัมพันธ์และผลกระทบซึ่งกันและกันระหว่างส่วนประกอบต่างๆ ทั้งนี้ก็เพื่อให้ทฤษฎีนี้สามารถนำไปประยุกต์ใช้ได้กับระบบหลากหลายลักษณะ และเพื่อแก้ปัญหาการขาดรากฐานร่วมทางทฤษฎีสำหรับระบบฐานข้อมูลและความรู้เชิงวัตถุแบบอนุมาน

2) ศึกษาค้นคว้าถึงความสัมพันธ์และผลกระทบต่อกันระหว่างการนิรนัย (Deduction) กับการถ่ายทอดคุณสมบัติ (Inheritance) และระหว่างการนิรนัยกับการแจงเหตุสู่ผลโดยนัย (Implicit Implication) ที่ได้มาจากการวิเคราะห์ส่วนประกอบของประโยคโดยอาศัยข้อมูลเกี่ยวกับโครงสร้างลำดับชั้นของสารสนเทศ ซึ่งการนิรนัย การถ่ายทอดคุณสมบัติ และการแจงเหตุสู่ผลโดยนัยนี้เป็นกระบวนการหลักที่ใช้ในการอนุมาน (Inference) ในระบบฐานข้อมูลและความรู้เชิงวัตถุแบบอนุมาน

3) เสนอวิธีการสำหรับการกำหนดความหมายที่ถูกต้องชัดเจนของโปรแกรมที่ใช้การนิรนัยควบคู่ไปกับการถ่ายทอดคุณสมบัติและการแจงเหตุสู่ผลโดยนัย ซึ่งความหมายของโปรแกรมที่ว่านี้จะเป็นพื้นฐานสำคัญในการกำหนดเนื้อหาของข้อมูลและความรู้ทั้งหมดที่ถูกจัดเก็บอยู่ในระบบ และเป็นพื้นฐานของการศึกษาและวิเคราะห์ฐานข้อมูลและความรู้อย่างละเอียด วิธีการที่กำหนดขึ้นนี้ควรจะครอบคลุมถึงกรณีที่มีการขัดแย้งกันระหว่างการอนุมานและการถ่ายทอดคุณสมบัติด้วย

4) ประยุกต์ทฤษฎีที่พัฒนาขึ้นและความรู้ที่ได้มากับฐานข้อมูลเอกสาร XML (Extensible Markup Language)

3. ผลการวิจัย

โครงการนี้เป็นงานวิจัยทางทฤษฎีเป็นหลัก โดยเริ่มต้นจากการศึกษาและวิเคราะห์เปรียบเทียบต้นแบบของระบบการจัดเก็บความรู้เชิงวัตถุแบบอนุมานที่ถูกเสนอในบทความต่างๆ และสร้างแบบจำลองทางคณิตศาสตร์ (Mathematical Model) เพื่อใช้ในการกำหนดส่วนประกอบต่างๆ และกำหนดความสัมพันธ์ของส่วนประกอบต่างๆ ของระบบ หลังจากนั้นจึงเสนอทฤษฎีที่จะสามารถนำไปใช้เป็นพื้นฐานร่วมกันในการอธิบายคุณสมบัติของส่วนต่างๆ และคุณสมบัติโดยรวมของระบบต้นแบบต่างๆ และการกำหนดวิธีในการประมวลผล โดยใช้วิธีการพิสูจน์ทางคณิตศาสตร์ (Mathematical Proof Methods) ในการตรวจสอบความถูกต้องของทฤษฎีบทต่างๆ ที่เสนอขึ้น

ทฤษฎีที่เสนอขึ้นในโครงการวิจัย ครอบคลุมถึงลักษณะสำคัญต่างๆทั้งหมดของระบบจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน อันได้แก่ การบรรยายข้อมูลและความรู้โดยนัย การจัดลำดับชั้นของกลุ่มข้อมูล การอนุมานโดยใช้การนิรนัยร่วมกับการถ่ายทอดคุณสมบัติ และการแจงเหตุสู่ผลโดยนัย รวมไปถึงวิธีการในการ

กำหนดความหมายที่ชัดเจนของโปรแกรมที่ใช้ในการบรรยายข้อมูลและความรู้ในระบบ และวิธีการในการคำนวณหาความหมายของโปรแกรม

เนื้อหาหลักของโครงการวิจัยสามารถแบ่งออกได้เป็นสองส่วนใหญ่ๆ ดังนี้

1) ส่วนแรกเป็นการศึกษาค้นคว้าถึงความสัมพันธ์และผลกระทบต่อกันระหว่างการนิรนัยและการถ่ายทอดคุณสมบัติ คณะผู้วิจัยได้แจกแจงกลไกในกระบวนการถ่ายทอดคุณสมบัติออกเป็นสองลักษณะ ลักษณะแรกเป็นการถ่ายทอดคุณสมบัติที่ได้มาจากกระบวนการเพิ่มรายละเอียดเฉพาะให้แก่วัตถุ (Specialization Operation) ลักษณะที่สองเป็นการถ่ายทอดคุณสมบัติที่ได้มาจากการแจงเหตุสู่ผลโดยนัย (Implicit Implication) โดยอาศัยข้อมูลเกี่ยวกับโครงสร้างลำดับชั้น (Classification Hierarchy) ของสารสนเทศเป็นพื้นฐาน

   ในส่วนของการถ่ายทอดคุณสมบัติแบบแรก ผลการศึกษาและวิเคราะห์ปรากฏว่าการกำหนดความหมายของโปรแกรมในทฤษฎีโปรแกรมเชิงประกาศ (Declarative Programs Theory) [3] สามารถนำมาประยุกต์ใช้ได้โดยตรงในการกำหนดความหมายของโปรแกรมประเภทที่ไม่มีความขัดแย้งกันระหว่างการนิรนัยและการถ่ายทอดคุณสมบัติ แต่ไม่สามารถนำมาใช้ได้กับโปรแกรมที่มีการใช้การนิรนัยควบคู่กับการถ่ายทอดคุณสมบัติที่ขัดแย้งกัน ซึ่งการขัดแย้งกันระหว่างการนิรนัยและการถ่ายทอดคุณสมบัตินี้เป็นลักษณะที่พบได้บ่อยครั้งในโปรแกรมที่ใช้บรรยายข้อมูลและความรู้เชิงวัตถุแบบอนุมาน คณะผู้วิจัยได้เสนอวิธีการกำหนดความหมายของโปรแกรมขึ้นใหม่โดยใช้ทฤษฎีการโต้แย้ง (Argumentation Theoretic Framework) [7] เป็นพื้นฐาน วิธีการที่กำหนดขึ้นใหม่นี้สามารถนำไปใช้ได้กับโปรแกรมทั้งสองประเภท และสามารถนำไปประยุกต์ใช้ในการประมวลผลเพื่อคำนวณหาข้อมูลและความรู้ทั้งหมดในระบบได้โดยตรง เนื่องจากเป็นวิธีการที่ใช้การกระทำซ้ำ (Iteration) ในการคำนวณหาจุดตรึงที่น้อยที่สุด (Least Fixed Point) เพื่อใช้เป็นความหมายของโปรแกรม

   เพื่อที่จะประเมินผลวิธีการกำหนดความหมายของโปรแกรมที่เสนอขึ้น คณะผู้วิจัยได้ทำการศึกษาเปรียบเทียบความหมายของโปรแกรมที่กำหนดขึ้นนี้กับความหมายสมบูรณ์แบบ (Perfect Model Semantics) ที่ถูกเสนอโดย กิลเลียน ดอบบี และรอดนีย์ โทเปอร์ ในปี พ.ศ. 2536 [6] และได้ทำการพิสูจน์ว่าในกรณีของโปรแกรมที่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้ (Inheritance-Stratified Program) การกำหนดความหมายทั้งสองแบบนี้จะให้ผลลัพธ์เหมือนกัน ส่วนในกรณีของโปรแกรมที่ไม่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้นั้น วิธีการที่คณะผู้วิจัยเสนอขึ้นยังคงให้ผลลัพธ์ที่ถูกต้องสมเหตุสมผล ในขณะที่วิธีของ กิลเลียน ดอบบี และรอดนีย์ โทเปอร์ ไม่สามารถให้ความหมายที่เหมาะสมแก่โปรแกรมประเภทนี้ได้

   สำหรับการถ่ายทอดคุณสมบัติที่ได้มาจากกระบวนการแจงเหตุสู่ผลโดยนัยนั้น คณะผู้วิจัยได้เสนอวิธีการอีกแบบหนึ่งขึ้นมาใหม่สำหรับใช้ในการกำหนดความหมายของโปรแกรมที่มีการใช้การนิรนัยควบคู่ไปกับการถ่ายทอดคุณสมบัติประเภทนี้ โดยอาศัยสมมุติฐานที่ว่าการแจงเหตุสู่ผลโดยนัยมีลักษณะเป็นความสัมพันธ์แบบอันดับเบื้องต้น (Preorder) ซึ่งสามารถถูกกำหนดได้ล่วงหน้าจากการวิเคราะห์โครงสร้างลำดับชั้นของข้อมูล และได้ทำการพิสูจน์ว่าภายใต้การกำหนดความหมายที่เสนอขึ้นนี้โปรแกรมแต่ละโปรแกรมจะมีความหมายชัดเจนเป็นหนึ่งเดียว (Unique Meaning) ซึ่ง

ความหมายดังกล่าวนี้จะถูกคำนวณได้จากการประมวลผลหาจุดตรึงที่น้อยที่สุดโดยใช้ตัวดำเนินการ (Operator) ที่ถูกกำหนดได้จากประโยคต่างๆ ทั้งหมดในโปรแกรม

ต่อจากนั้น คณะผู้วิจัยได้นำทฤษฎีการหาค่าจุดตรึงโดยอาศัยลำดับความสำคัญ (Fixed Point Theory with Subsumption) ที่ถูกเสนอขึ้นในปี พ.ศ. 2538 โดย เจอร์ฮาร์ด โคสท์เลอร์ และคณะ [12] มาเป็นพื้นฐานในการแสดงว่าในกรณีที่การแจงเหตุสู่ผลโดยนัยมีลักษณะเป็นความสัมพันธ์แบบอันดับบางส่วน (Partial Order) ความหมายของโปรแกรมข้างต้นจะสามารถถูกคำนวณได้อย่างรวดเร็วและมีประสิทธิภาพมากยิ่งขึ้น โดยใช้การประมวลผลบนโดเมนที่มีขนาดเล็กลงอย่างมาก นอกจากนั้น คณะผู้วิจัยยังได้เสนอวิธีการในการกำหนดความหมายของโปรแกรมที่มีการใช้การนิรนัยควบคู่ไปกับการแจงเหตุสู่ผลโดยนัยประเภทที่ไม่สามารถกำหนดล่วงหน้าได้ และได้เสนอวิธีการในการประมวลผลเพื่อคำนวณหาความหมายของโปรแกรมในกรณีหลังนี้ด้วย

2) ส่วนที่สองเป็นการนำเอาผลการศึกษาและความเข้าใจในส่วนแรกมาประยุกต์ใช้ โดยพยายามที่จะพัฒนาทฤษฎีและแบบจำลองการคำนวณพื้นฐานสำหรับฐานข้อมูลเอกสาร XML เอกสาร XML มีลักษณะเป็นต้นไม้ (Tree) ที่ซ้อนลึก (Nested) ได้หลายชั้น จึงมีความซับซ้อน และถือได้ว่ามีลักษณะแบบกึ่งโครงสร้าง (Semi-structured) ทำให้ยากต่อการจัดเก็บ สืบค้น และเปลี่ยนแปลงข้อมูลมากกว่าข้อมูลที่มีโครงสร้างชัดเจนเรียบง่ายอย่างเช่นข้อมูลในฐานข้อมูลแบบสัมพันธ์ ทฤษฎีพื้นฐานที่พัฒนาขึ้นมาใช้ได้ทั้งการสร้างแบบจำลองข้อมูลของเอกสาร XML การสร้างแบบจำลองของ Document Type Definition (DTD) ของเอกสาร XML และการประมวลผลการสอบถาม (query processing) ดังนั้นจึงถือได้ว่าเป็นทฤษฎีพื้นฐานของเอกสาร XML ที่มีความยืดหยุ่นมาก

4. บทวิจารณ์

1) ในผลงานวิจัย Defeasible Inheritance Through Specialization [Output #1, #3] คณะผู้วิจัยได้ศึกษาเปรียบเทียบความหมายของโปรแกรมที่กำหนดขึ้นใหม่กับความหมายของโปรแกรมสมบูรณ์แบบ (Perfect Model Semantics) ของ กิลเลียน ดอบบี และรอดนีย์ โทเปอร์ [6] ผลของการเปรียบเทียบโดยกระบวนการทางคณิตศาสตร์ปรากฏว่า ในกรณีของโปรแกรมที่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้ (Inheritance-Stratified Program) วิธีการทั้งสองให้ผลลัพธ์เหมือนกัน ส่วนในกรณีของโปรแกรมที่ไม่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้นั้น วิธีการที่เสนอในผลงานวิจัยให้ผลลัพธ์ที่มีความถูกต้องสมเหตุสมผลมากกว่า

2) ในผลงานวิจัย Declarative Programs with Implicit Implication [Output #2] คณะผู้วิจัยเสนอมุมมองของการพิจารณาความสัมพันธ์ระหว่างประโยคพื้นฐานในโปรแกรมภาษา Conceptual Graph ที่เสนอโดย บีกาส ซี. โกส และวิลาศ วูวงศ์ [8-11, 14] และในภาษา F-logic ที่เสนอโดย ไมเคิล ไคเฟอร์ และคณะ [12] ว่าเป็นความสัมพันธ์แบบการแจงเหตุสู่ผลโดยนัย (Implicit Implication) วิธีการกำหนดความหมายของโปรแกรมที่คณะผู้วิจัยเสนอขึ้นใหม่ในผลงานวิจัยนี้ สามารถนำไปใช้กับโปรแกรมทั้งสองลักษณะนี้ได้

3) ในผลงานวิจัย A Foundation for XML Document Databases [Output #4, #5, #6] คณะผู้วิจัยได้พัฒนาวิธีการสำหรับการจำลองแบบเอกสาร XML และไวยากรณ์ DTD รวมทั้งการประมวลผลการสอบถาม บนกรอบทฤษฎีพื้นฐานเดียวกัน ที่ผ่านมายังไม่มีใครที่สามารถพัฒนาทฤษฎีที่มีขอบเขตกว้างขวางและยืดหยุ่น เช่นดังผลงานวิจัยนี้ได้ เป็นที่คาดว่ากรอบทฤษฎีพื้นฐานที่พัฒนาขึ้นมานี้จะนำไปสู่การวิจัยและพัฒนาทางด้านเทคนิคเพื่อให้เกิดระบบฐานข้อมูลสำหรับเอกสาร XML อันจะเป็นประโยชน์ในทางปฏิบัติมาก เนื่องจาก XML เป็นที่ยอมรับแล้วว่าเป็นมาตรฐานที่สำคัญในการแสดงและแลกเปลี่ยนข้อมูลบน Web

5. หนังสืออ้างอิง

[1] S. Abitelboul. Towards a Deductive Object-Oriented Database Language. *Data & Knowledge Engineering*, 5(4):263-287, 1990.

[2] P. Aczel. Replacement Systems and the Axiomatization of Situation Theory. In R. Cooper, K. Mukai, and J. Perry, editors, *Situation Theory and Its Applications*, volume 1, number 22 in CSLI Lecture Notes, pages 3-31. CSLI Publications, Stanford, 1990.

[3] K. Akama. Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advanced in Software Science and Technology*, 5:45-63, 1993.

[4] H. Ait-Kaci and R. Nasr. LOGIN: A Logic Programming Language with Built-in Inheritance. *The Journal of Logic Programming*, 3(3):185-215, 1986.

[6] G. Dobbie and R. Topor. On the Declarative and Procedural Semantics of Deductive Object-Oriented Systems. *Journal of Intelligent Information Systems*, 4(2):193-219, March 1995.

[7] P. M. Dung. On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and N-Person Games. *Artificial Intelligence*, 77(2):321-357, September 1995.

[8] B. C. Ghosh and V. Wuwongse. Declarative semantics of Conceptual Programs. In *Proceedings of the 1st International Conference on Conceptual structures (ICCS'93)*, Quebec City, Canada, August 1993.

[9] B. C. Ghosh and V. Wuwongse. Inference Systems for Conceptual Graph Programs. In W. M. Tepfenhart, J. P. Dick and J. F. Sowa (eds.), *Proceedings of the 2nd International Conference on Conceptual Strucrures (ICCS'94)*, Lecture Notes in Artificial Intelligence #835, 214-229, Springer-Verlag, College Park, Maryland, USA, August 1994.

[10] B. C. Ghosh and V. Wuwongse. A Direct Proof Procedure for Definite Conceptual Graph Programs. In G. Ellis, R. Levison, W. Rich and J. F. Sowa (eds.), *Proceedings of the 3rd International Conference on Conceptual Structures (ICCS'95)*, Lecture Notes in Artificial Intelligence #954, 158-172, Springer-Verlag, Santa Cruz, CA, USA, August 1995.

[11] B. C. Ghosh and V. Wuwongse. Conceptual Graph Programs and Their Declarative Semantics. *IEICE Transaction on Information and Systems*, E78-D(9):1208-1217, September 1995.

[12] G. Klostler, W. Kiebling, H. Thone and U. Guntzer. Fixpoint Iteration with Subsumption in Deductive Databases. *Journal of Intelligent Information Systems*, 4(2):123-148, March 1995.

[13] M. Kifer, G. Lausen and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the Association for Computing Machinery*, 42(4):741-843, July 1995.

[14] V. Wuwongse and B. C. Ghosh. Towards Deductive Object-Oriented Databases Based on Conceptual Graphs. In H. D. Pfeiffer and T. F. Nagle (eds.) *Proceedings of the 7th Annual Workshop on Conceptual Graphs*, Lecture Notes in Artificial Intelligence #754, 188-205, Springer-Verlag, Las Cruces, NM, USA, July 1992.

6. Output

[1] Ekawit Nantajeewarawat and Vilas Wuwongse, Defeasible Inheritance Through Specialization, *Computational Intelligence*, Vol. 17, No. 1, 2001 (to appear).

[2] Vilas Wuwongse and Ekawit Nantajeewarawat, Declarative Programs with Implicit Implication, *IEEE Transactions on Knowledge and Data Engineering* (1st revision).

[3] Ekawit Nantajeewarawat and Vilas Wuwongse, An Argumentation Approach to Semantics of Declarative Programs with Defeasible Inheritance, in P.S. Thiagarajan and R. Yap (eds.): *ASIAN'99*, Lecture Notes in Computer Science #1742, Springer-Verlag, pp.239-250, 1999.

[4] Chutiporn Anutariya, Vilas Wuwongse, Ekawit Nantajeewarawat and Kiyoshi Akama, Towards a Foundation for XML Document Databases, *Proceedings of the 1st Int. Conf. Electronic Commerce and Web Technologies*, Lecture Notes in Computer Science, Springer-Verlag, 2000 (to appear).

[5] Vilas Wuwongse, Kiyoshi Akama , Chutiporn Anutariya and Ekawit Nantajeewarawat, A Foundation for XML Databases: Data Model, *Int. J. Knowledge and Information Systems* (submitted).

[6] Chutiporn Anutariya, Vilas Wuwongse, Kiyoshi Akama and Ekawit Nantajeewarawat, A Foundation for XML Databases: DTD Modeling, *Int. J. Knowledge and Information Systems* (submitted).

7. การนำไปใช้ประโยชน์

เชิงวิชาการ

โครงการวิจัยนี้อธิบายอย่างละเอียดถึงลักษณะสำคัญต่างๆ ของระบบจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน โดยเฉพาะอย่างยิ่งในเรื่องของความสัมพันธ์และผลกระทบต่อกันระหว่างการนิรนัยและการถ่ายทอดคุณสมบัติ ซึ่งเป็นกระบวนการอนุมานหลักในระบบ ซึ่งเป็นพื้นฐานหลักในการกำหนดส่วนประกอบต่างๆ ของระบบ ในโครงการวิจัยนี้ได้มีการเสนอวิธีการกำหนดความหมายของโปรแกรมที่ใช้ในการบรรยาย

เนื้อหาของข้อมูลและความรู้ในระบบอย่างชัดเจน และได้เสนอวิธีการสำหรับประมวลผลเพื่อคำนวณหาความหมายของโปรแกรมที่กำหนดขึ้น ความหมายและวิธีการดังกล่าวเป็นพื้นฐานสำคัญในการศึกษาและวิเคราะห์เนื้อหาของข้อมูลและความรู้ในระบบอย่างละเอียด และเป็นพื้นฐานสำคัญในการออกแบบและการพัฒนาซอฟต์แวร์สำหรับปรับปรุงประสิทธิภาพการจัดเก็บและการประมวลผลข้อมูลและความรู้ในระบบ คณะผู้วิจัยคาดหวังว่าทฤษฎีที่เสนอขึ้นนี้จะเป็นรากฐานที่สำคัญสำหรับการพัฒนาระบบจัดการฐานข้อมูลและความรู้ในอนาคต

นอกจากนี้ คณะผู้วิจัยคาดหวังว่าโครงการวิจัยจะมีประโยชน์อย่างยิ่งสำหรับผู้ที่ต้องการศึกษาและค้นคว้าเกี่ยวกับระบบการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน แบบจำลองทางคณิตศาสตร์ที่เสนอในโครงการวิจัยสามารถนำไปใช้เป็นเครื่องมือช่วยในการทำความเข้าใจในระบบฐานข้อมูลและความรู้เชิงวัตถุแบบอนุมานหลากหลายลักษณะที่นักวิจัยหลายกลุ่มได้เสนอขึ้นมา การศึกษาระบบเหล่านี้โดยตรงทีละระบบโดยปราศจากทฤษฎีพื้นฐานร่วมอาจทำให้เกิดความสับสนขึ้นได้ง่าย เนื่องจากระบบเหล่านี้มีการกำหนดลักษณะสำคัญต่างๆ ของระบบโดยใช้ไวยากรณ์เฉพาะที่แตกต่างกันออกไป และมีความคลาดเคลื่อนกันในการกำหนดความหมายของลักษณะต่างๆ เหล่านี้ ซึ่งส่งผลให้เกิดความคลาดเคลื่อนกันในการกำหนดความหมายโดยรวมของโปรแกรมในระบบ

โครงการวิจัยนี้ได้มีส่วนช่วยสร้างนักวิจัยใหม่ 2 คน โดยคนแรกจบการศึกษาระดับปริญญาเอกแล้วและขณะนี้เป็นผู้ช่วยศาสตราจารย์ที่สถาบันเทคโนโลยีนานาชาติสิรินธร มหาวิทยาลัยธรรมศาสตร์ ส่วนอีกคนกำลังศึกษาในระดับปริญญาเอกที่ AIT

นอกจากนี้โครงการวิจัยนี้ยังมีส่วนกระชับความร่วมมือในการทำวิจัยของคณะผู้วิจัยกับ Prof. Kiyos Akama ที่มหาวิทยาลัย Hokkaido

เชิงพาณิชย์

เป็นที่ทราบกันดีว่าในสองทศวรรษที่ผ่านมา ผู้ผลิตซอฟต์แวร์จำนวนหนึ่งได้พัฒนาซอฟต์แวร์ระบบการจัดการฐานข้อมูลแบบความสัมพันธ์ (Relational Database Management System) ขึ้นมา และได้มีการซอฟต์แวร์ดังกล่าวนี้ไปใช้ในงานประยุกต์ทางด้านธุรกิจและด้านอื่นๆ อย่างประสบความสำเร็จอย่างกว้างข อย่างไรก็ดี นักวิทยาการคอมพิวเตอร์ส่วนใหญ่ยังมีความเห็นว่า ระบบฐานข้อมูลแบบสัมพันธ์ยังคงมีข้อจำกัดอีกหลายประการ ตัวอย่างเช่น ข้อจำกัดในเรื่องของ โครงสร้างข้อมูล (โครงสร้างข้อมูลพื้นฐาน ของระบบฐานข้อมูลแบบความสัมพันธ์เป็นโครงสร้างแบบ record ซึ่งเป็นอุปสรรคในการจัดเก็บข้อมูลที่มีโครงสร้างซ้อน) ข้อจำกัดในการค้นหาข้อมูลโดยใช้คำสั่งค้นหาที่มีการอ้างอิงถึงตัวเอง (Recursive Query) อุปสรร เรื่องของช่องว่างระหว่างภาษาที่ใช้เขียนโปรแกรมประยุกต์กับภาษาที่ใช้ในการหาข้อมูล (Impedance Mism Problem) ข้อจำกัดในการจัดเก็บข้อมูลแบบเป็นลำดับชั้น และการนำสารสนเทศโดยนัยที่มีอยู่ในข้อมูลไ ให้เป็นประโยชน์ เป็นต้น

Outp

To

นักวิทยาการคอมพิวเตอร์จำนวนมากเชื่อว่า ระบบฐานข้อมูลเชิงวัตถุแบบอนุมานมีศักยภาพที่จะสามารถแก้ไขข้อจำกัดต่างๆเหล่านี้ และสามารถที่จะรองรับงานประยุกต์ประเภทต่างๆได้หลากหลายมากขึ้น ทฤษฎีที่เสนอขึ้นในโครงการวิจัย จะเป็นพื้นฐานที่สำคัญสำหรับการพัฒนาซอฟต์แวร์ระบบการจัดการฐานข้อมูลเชิงวัตถุแบบอนุมาน (Deductive Object-Oriented Database Management System) ทฤษฎีบทต่าง ๆ ที่เชื่อมโยงความหมายของโปรแกรมกับจุดตรึงที่น้อยที่สุด (The Least Fixpoint) ของตัวดำเนินการ (operator) ที่ถูกกำหนดได้จากโปรแกรม สามารถนำไปประยุกต์ใช้ได้โดยตรง ในการประมวลผลหาเนื้อหาของข้อมูลในฐานข้อมูล และการค้นหาข้อมูลในระบบ โดยใช้อัลกอริธึม (Algorithm) มาตรฐานในการคำนวณหาค่าจุดตรึงที่น้อยที่สุดของตัวดำเนินการแบบโมโนโทนิก (Monototic Operator)

นอกจากนี้ในทฤษฎีที่เสนอขึ้น มีการวิเคราะห์และอธิบายถึงความหมายของโปรแกรมและเนื้อหาของฐานข้อมูลและความรู้อย่างละเอียดชัดเจน ซึ่งความชัดเจนไม่กำกวมของความหมายดังกล่าวนี้เป็นพื้นฐานที่จำเป็นอย่างยิ่งในการพัฒนาซอฟต์แวร์ประเภท Optimization Tools สำหรับใช้ในการปรับปรุงประสิทธิภาพของการจัดเก็บ ค้นหา และประมวลผลข้อมูลในฐานข้อมูลและความรู้ ตัวอย่างของซอฟต์แวร์ประเภทนี้ได้แก่ซอฟต์แวร์ที่ทำการเปลี่ยนคำสั่งค้นหาข้อมูลอย่างหนึ่งไปเป็นคำสั่งอีกอย่างหนึ่ง ที่ให้ผลลัพธ์เหมือนเดิมแต่ใช้เวลาและทรัพยากรของระบบในการค้นหาน้อยลงกว่าเดิม ซอฟต์แวร์ที่ใช้ลดขนาดของฐานข้อมูลลงโดยที่ยังคงมีเนื้อหาและความหมายคงเดิม เป็นต้น

ท้ายสุดนี้งานวิจัยประยุกต์ในส่วนของ XML คาดว่าจะมีประโยชน์อย่างยิ่งในทางปฏิบัติ เนื่องจาก XML ได้กลายเป็นมาตรฐานของการแสดงและแลกเปลี่ยนข้อมูลบน Internet ซึ่งจะครอบคลุมงานเกือบทุกประเภท ไม่ว่าจะเป็น พาณิชย์อีเล็กทรอนิกส์ ห้องสมุดดิจิตอล หรือ การเรียนรู้ทางไกล

Submitted to Int. J. Knowledge and Information Systems Output #5

A Foundation for XML Document Databases: Data Model*

Vilas Wuwongse[1], Kiyoshi Akama[3]
Chutiporn Anutariya[1] and Ekawit Nantajeewarawat[2]

[1] Computer Science & Information Management Program, Asian Institute of Technology
Pathumtani 12120, Thailand
{vw, ca}@cs.ait.ac.th

[2] Information Technology Program, Sirindhorn International Institute of Technology,
Thammasat University, Pathumtani 12120, Thailand
ekawit@siit.tu.ac.th

[3] Center for Information and Multimedia Studies,
Hokkaido University, Sapporo 060, Japan
akama@cims.hokudai.ac.jp

Abstract. The proposed data model for XML documents, based on *Declarative Description* (*DD*) theory, formally generalizes the definition of an *XML element* to an *XML expression* by incorporation of variables for representation of inherent implicit information and enhancement of its expressive power. An XML element is simply modeled as a *variable-free XML expression*, while an XML document – a set of XML elements – as an *XML declarative description* (*XML-DD*) which consists of *clauses* describing those elements in the document, their relationships as well as integrity constraints. Selective and complex queries, formulated as sets of clauses, about explicit information satisfying certain specified constraints as well as derivable information which is implicit in the documents, become then expressible and computable. Similarly, an XML DTD is modeled as a corresponding set of clauses which can be employed in order to validate an XML document with respect to that DTD. The proposed model thereby serves as an effective and well-founded XML document database management framework with succinct representational and operational uniformity, reasoning capabilities as well as complex and deductive query supports.

Key words. data model, document modeling, specialization system, XML declarative description, XML document, XML element, XML expression.

1 Introduction

Extensible Markup Language (*XML*) [9], a W3C recommendation which has recently emerged as a standard for data representation and interchange among various Web applications, is a simpler and convenient subset of *Standard Generalized Markup Language* (*SGML*). XML provides simple means for a more meaningful and understandable representation of Web content. In contrast to HTML, XML does not require a predefined fixed set of tags; it provides instead a facility to define new tag sets as well as structural relationships of tags via tag element nesting and referencing, whence it is *self-describing* and *extensible*.

An XML document is only required to be *well-formed*, i.e., its tags must be properly nested, but need not conform to a particular *Document Type Definition* (*DTD*) – a grammar defining restrictions on tags, attributes and content models. Hence, XML is considered as a variation of *semistructured data* – data that may be varied and are not restricted to any particular schema or structure; they are at times referred to as *schemaless* data [13]. Management of semistructured data by a highly-structured modeling technique, such as relational and object-oriented models, not only results in a very complicated logical schema, but also requires much efforts and frequent schema modifications. This difficulty has obstructed the use of such approaches to XML data modeling and management. Consequently, development of an appropriate and efficient data model for XML documents has become an active research area with major current models based on *trees* [7], *directed edge-labeled graphs* [8,10,11,13,19], *tree automata* theory [15,17,18] and *functional programming* [12].

* This paper is a substantially expanded version of [5].

A *declarative description data model for XML documents* [5] is developed by employment of *Declarative Description* (DD) theory [1,2,3], which has been developed with generality and applicability to data structures of a wide variety of domains, each characterized by a mathematical structure, called a *specialization system*. Based on the formulation of an appropriate *specialization system for XML expressions*, a framework for their representation, computation and reasoning is constructed. The definition of *XML expressions* introduced here is a formal extension of XML elements which allows representation of both explicit and implicit information by means of variables. In the proposed model, conventional XML elements are represented directly as ground (variable-free) XML expressions, without need for translation. An *XML declarative description* (*XML-DD*) comprises a set of XML expressions, formulated as *unit clauses*, and a (possibly empty) set of their relationships, formulated as *non-unit clauses*. The meaning of such an XML-DD will not only yield all the explicit information, represented in terms of unit clauses, but will also include all the implicit information derivable by application of non-unit clauses to the set of unit clauses, whence complex queries about this implicit information [4] can be formulated and executed. Non-unit clauses not only represent relationships among XML elements but can also be used to define *integrity constraints* which are important in a document, such as data integrity, *path and type constraints* [11]. Moreover, in order to check whether an element conforms to a given DTD or not, one can similarly apply the same convention, i.e., simply translate the DTD into a corresponding set of clauses, and then verify the validity of the element against the clauses. Such a validation process is usually applied when an element is inserted or updated.

Section 2 reviews major approaches to modeling semistructured/SGML/XML documents, Section 3 recalls fundamental definitions of DD theory, Section 4 develops a declarative description data model for XML documents, Section 5 presents an approach to modeling XML documents, Section 6 compares the proposed new approach with existing ones, and Section 7 draws conclusions and presents suggestions for future research.

2 Review of Data Models for Semistructured/SGML/XML Documents

Three important approaches to the modeling of semistructured/SGML data prior to 1995, i.e., *traditional information retrieval*, *relational model* and *object-oriented approaches*, have been reviewed in [20]. A review and evaluation of more recent work follows.

2.1 Tree-Based Approach

Based on the lexical structure of XML data, an XML document can be viewed as a *tree* corresponding to a document's text representation. An example of this approach is *Document Object Model* (*DOM*) for XML [7].

In XML, an element can have an attribute of type *ID* the value of which provides a unique identifier, referencable by other elements through attributes of type *IDREF* and *IDREFS*. However, by simply treating attributes of these types as nothing more than text strings, the tree-like representation of an XML document encounters a serious problem in capturing cross-link or referential relationships among XML elements. If this approach is employed, a query language itself must provide a means to associate these related elements. Otherwise, users cannot issue queries with referential relationships.

2.2 Graph-Based Approach

Since XML can be viewed as a variation of *semistructured data*, several models for semistructured data have been modified and extended to fully support such data [11,13]. Many of these semistructured data models, such as *Object Exchange Model* (*OEM*) or *Lore data model* [19] and *deterministic data model* [10], are intuitively based on *directed, edge-labeled graphs*.

In graph-based models, a collection of XML documents is represented as a directed, edge-labeled graph [8,10,11,13,19]. A non-leaf node in the graph, associated with a sequence of zero or more attribute-value pairs, represents an XML element, while a leaf node represents an XML element's textual content. An edge, pointing from a *parent* element to a *child* element and labeled with the child element's tag name, represents an element-subelement relationship. An IDREF(S) attribute is represented by an edge pointing from the referring element to the referred element and labeled with the attribute name.

The graph model can be viewed as an enhancement of the tree model with an attempt to represent and handle the referential relationships among arbitrary tree nodes, defined by means of attributes of the types ID and IDREF(S). Although a graph-based model provides an effective and straightforward way to handling XML data, it encounters difficulties in restricting XML data to a given DTD. For instance, the proposal [8] only provides a way to querying XML data but does not facilitate a means to represent the structure imposed by a DTD. The model requires substantial extension to overcome this difficulty. For example, by application of *first-order logic theory*, the proposal [11] has incorporated an ability to express *path and type constraints* for the specification of the structure of XML data; the integration of these *two different formalisms* also yields an ability to reason about path constraints. However, other forms of integrity constraints have not yet been included. In addition, the complex notions of *model* and *implication* in first-order logic tend to complicate the syntax and semantics of path constraints and make their understanding difficult.

Besides introducing *validating parsers* to restrict XML data to a particular DTD, Lore's XML model has proposed the use of *DataGuides* [19] – a graph describing the structure of documents stored in Lore's database – to capture the structure of documents imposed by a DTD. Apart from providing a computational mechanism, Lore's XML model does not possess a capability for reasoning about XML elements.

2.3 Hedge Automaton Approach

By means of *hedge automaton theory* [16] (aka. *tree automaton* and *forest automaton* theory [15]), developed by employment of the basic ideas of *string automaton* theory, the proposals [17,18] have constructed an approach to formalization of XML documents and their DTDs. A hedge is a sequence of trees or, in XML terminology, a sequence of XML elements. A document is therefore represented by a *hedge* and a set of documents conforming to a DTD by a *regular hedge language* (*RHL*), expressible by a *regular hedge expression* (*RHE*) or a *regular hedge grammar* (*RHG*). A RHG is a quintuple (Σ, X, N, P, r_f),

where – Σ : a set of symbols,
- X : a set of variables,
- N : a set of *non-terminals*,
- P : a set of *production rules*, and
- r_f : a regular expression over the non-terminals.

Each production rule is of the form $n \rightarrow x$ or $n \rightarrow$ <*a*> *r* </*a*>,

where – n : a non-terminal in N,
- x : a variable in X,
- a : a symbol in Σ, and
- r : a regular expression over the non-terminals

This formalism allows a DTD to be easily translated into a corresponding RHG, which describes a RHL, or in this context, a set of documents conforming to the DTD. A *hedge automaton* can be employed to determine whether a document conforms to a given grammar (representing some particular DTD) or not. This approach also provides a mechanism to transform XML documents and their DTDs [17,18]. However, it does not provide a means for incorporation of knowledge into those significant operations.

2.4 Functional Programming Approach

The proposal [12] has developed a *functional programming approach* to modeling XML documents and formalizing operations upon them by introduction of user-defined typed feature term, called *node*, as its underlying data structure. Nodes can be grouped into the three types *text*, *element* and *reference*, which contain a character string, a list of child nodes, and a referential relationship to another node, respectively. In the model, an XML element is represented as an element node, a sequence of its child elements as a list of child nodes and a textual content of an element as a text node. Attributes are also modeled as element nodes the names of which begin with '@'. An element node representing an attribute of type CDATA contains exactly one child text node, while a node representing an attribute of type IDREF or IDREFS comprises a number of child reference nodes referencing to the referred element nodes. Apparently, this data model is comparable to the graph-based data model.

Based on this data model, an algebra for XML queries, expressed in terms of *list comprehensions* in the functional programming paradigm, has also been constructed [12]. Using list comprehensions, various kinds of XML query operations, such as navigation, nesting, grouping and joins, can be expressed. However, this approach has considerable limitations, as it does not possess an ability to model an XML DTD, whence a mechanism to verify whether an XML document conforms to a given DTD or not is not readily devised.

3 Declarative Description Theory

This section recalls certain fundamental definitions [illegible]

3.1 Specialization Systems

A *specialization system* is an abstract structure derived from the generalization of *substitutions* in conventional logic program theory, and defined in terms of certain very simple axioms.

Definition 1 [Specialization System]

A specialization system is a quadruple $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ of three sets $\mathcal{A}$, $\mathcal{G}$, $\mathcal{S}$ and a mapping μ from $\mathcal{S}$ to the set of all partial mappings on $\mathcal{A}$ such that:

1. $\forall s_1, s_2 \in \mathcal{S}, \exists s \in \mathcal{S}: \mu(s) = \mu(s_1) \circ \mu(s_2)$,
2. $\exists s \in \mathcal{S}, \forall a \in \mathcal{A}: \mu(s)(a) = a$,
3. $\mathcal{G} \subset \mathcal{A}$

where $\mu(s_1) \circ \mu(s_2)$ is the composite mapping of the partial mappings $\mu(s_1)$ and $\mu(s_2)$. The set $\mathcal{G}$ is called the *interpretation domain*, and the elements of $\mathcal{A}$, $\mathcal{G}$ and $\mathcal{S}$ are called *objects*, *ground objects*, and *specializations*, respectively. □

When μ is clear from the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$. If there exists b such that $a\theta = b$, θ is said to be applicable to a, and a is specialized to b by θ. Given $a \in \mathcal{A}$, let $rep(a)$ denote the set of all ground objects which can be specialized from a, i.e., for $g \in \mathcal{G}$, $g \in rep(a)$ iff there exists a specialization θ in $\mathcal{S}$ such that $a\theta = g$.

3.2 Declarative Descriptions

Declarative Descriptions (DDs) and other related concepts can now be defined in terms of a specialization system $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$.

Let $\mathcal{K}$ be a set of constraint predicates. A *constraint* on Γ is a formula $q(a_1, \ldots, a_n)$, where q is a constraint predicate in $\mathcal{K}$ and a_i an object in $\mathcal{A}$. Given a ground constraint $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}$, its truth and falsity are assumed to be predetermined. Denote the set of all true ground constraints by *Tcon*. A specialization θ is applicable to a constraint $q(a_1, \ldots, a_n)$ if θ is applicable to $a_1, \ldots, a_n$. The result of the application $q(a_1, \ldots, a_n)\theta$ is the constraint $q(a_1\theta, \ldots, a_n\theta)$; and $q(a_1, \ldots, a_n)$ is said to be specialized to $q(a_1\theta, \ldots, a_n\theta)$ by θ.

The notion of constraints introduced above is useful for defining restrictions on objects in $\mathcal{A}$

Definition 2 [Declarative Description]

A *clause* on Γ is a formula of the form:

$$H \leftarrow B_1, B_2, \ldots, B_n \tag{1}$$

where $n \geq 0$, H is an object in $\mathcal{A}$ and B_i an object in $\mathcal{A}$ or a constraint on Γ. H is called the *head* and $(B_1, B_2, \ldots, B_n)$ the *body* of the clause. A *declarative description* (*DD*) on Γ is a (possibly infinite) set of clauses on Γ. □

Let C be a clause $(H \leftarrow B_1, B_2, \ldots, B_n)$. If $n = 0$, uch a clause C, is called a *unit clause*, and, if $n > 0$, a *non-unit clause*. The head of C will be denoted by *head*(C) and the set of all objects and constraints in the body of C by *object*(C) and *con*(C), respectively. Let *body*(C) = *object*(C) $\cup$ *con*(C). A clause C' is an *instance* of C iff there is a specialization $\theta \in \mathcal{S}$ such that θ is applicable to $H, B_1, B_2, \ldots, B_n$ and $C' = C\theta = (H\theta \leftarrow B_1\theta, B_2\theta, \ldots, B_n\theta)$. A clause C is a *ground clause* iff C comprises only ground objects and ground constraints. When it is clear from the context, a declarative description on Γ is simply called a *description*.

3.3 Semantics of Declarative Description

The mapping T_P: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$ defined for a declarative description P on Γ, will be used to define the declarative semantics of P in Definition 4

Definition 3 [Mapping T_P]

Let P be a declarative description on Γ. The definition of the mapping T_P: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$ follows:

For each $X \subset \mathcal{G}$ an XML element g is contained in $T_P(X)$ iff there exist a clause $C \in P$ and a specialization $\theta \in \mathcal{S}$ such that $C\theta$ is a ground clause the head of which is g and all the objects and constraints in the body of which belong to X and *Tcon*, respectively, i.e.:

$$T_P(X) = \{head(C\theta) \mid C \in P,\ \ \theta \in \mathcal{S},\ \ C\theta \text{ is a ground clause},\ object(C\theta) \subset X,\ \ con(C\theta) \subset Tcon\ \} \qquad (2)$$

□

Definition 4 [Semantics of Declarative Description]

Let P be a declarative description on Γ. The meaning of P, denoted by $\mathcal{M}(P)$, is defined by

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} [T_P]^n(\varnothing) \qquad (3)$$

where $\varnothing$ is the empty set and $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$. □

3.4 Equivalent Transformations

Equivalent Transformation (*ET*) [3] is a new computational model based on semantics preserving transformations (or *equivalent transformations*) of declarative descriptions. Basically, a declarative description P_1 is said to be transformed equivalently into a declarative description P_2 if they have exactly the same meaning, i.e., $\mathcal{M}(P_1) = \mathcal{M}(P_2)$. In the ET model, computation is defined by means of *equivalent transformation rules* (*ET rules*) to be applied to the components – objects and constraints – of a target clause.

4 Declarative Description Data Model for XML Documents

By means of DD theory, an XML data model is formulated. Subsection 4.1 specifies the format and structure of conventional XML elements, Subsection 4.2 gives formal definitions of *XML expressions*, Subsection 4.3 defines specialization operations for XML expressions, and Subsection 4.4 formulates a *specialization system for XML expressions*, denoted by $\Gamma_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{S}_X, \mu_X \rangle$, *XML Declarative Descriptions* and other related concepts.

4.1 Conventional XML Elements

By convention, XML elements are ground, i.e., they contain no variable, and assume formally the forms:

1. *empty element*: $<elem\_type\ \ attr_1=val_1\ \ldots\ attr_m=val_m/>$
2. *simple element*: $<elem\_type\ \ attr_1=val_1\ \ldots\ attr_m=val_m>\ val_{m+1}\ </elem\_type>$
3. *nested element*: $<elem\_type\ attr_1=val_1\ \ldots\ attr_m=val_m>\ e_1\ \ldots\ e_n\ </elem\_type>$

where – $n, m \geq 0$,
- *elem_type* : an element type (or tag name),
- $attr_i$: distinct attribute names,
- val_i : literals, and
- e_i : XML elements.

However, in order to express inherent implicit information and enhance its expressive power, the definition of an XML element will be formally extended by incorporation of variables, and then called an *XML expression*.

4.2 XML Expressions

Let Ω_X be an alphabet comprising the symbols in the following sets:

1. Σ : a set of *characters*
2. N : a set of *names* (which could be *element types* or *attribute names*)
3. $NVAR$: a set of *name-variables* (or *N-variables*)
4. $SVAR$: a set of *string-variables* (or *S-variables*)
5. $PVAR$: a set of *attribute-value-pair-variables* (or *P-variables*)
6. $EVAR$: a set of *XML-expression-variables* (or *E-variables*)

7. *IVAR* : a set of *intermediate-expression-variables* (or *I-variables*).

N-, *S*-, *P*-, *E*- and *I*-variables introduced here are useful for representation of implicit information contained in XML expressions. Intuitively, an *N*-variable will be instantiated to an element type or an attribute name in *N* and an *S*-variable to a string in Σ^*, while a *P*-variable will be specialized to a sequence of attribute-value pairs, an *E*-variable to a sequence of XML expressions and an *I*-variable to a part of an XML expression. A detailed explanation of the specializations of these variables is provided in the next subsection. In order to distinguish between elements of the above sets, let:

1. Every element in *NVAR* begin with "$N:", in *SVAR* with "$S:", in *PVAR* with "$P:", in *EVAR* with "$E:" and in *IVAR* with "$I:".
2. No element in *N* begin with "$N:" and '$' $\notin \Sigma$.

Based on the alphabet Ω_X, the formal definition of an XML expression is now given:

Definition 5 [XML expression]

An *XML expression* on Ω_X takes one of the following forms:

1. *evar*
2. <*elem_type* $attr_1=val_1$ ... $attr_k=val_k$ $pvar_1$ ... $pvar_m$ />
3. <*elem_type* $attr_1=val_1$ ... $attr_k=val_k$ $pvar_1$ ... $pvar_m$ > val_{k+1} </*elem_type*>
4. <*elem_type* $attr_1=val_1$ ... $attr_k=val_k$ $pvar_1$ ... $pvar_m$ > e_1 ... e_n </*elem_type*>
5. <*ivar*> e_1 ... e_n </*ivar*>

where
- *evar* $\in$ *EVAR*,
- $k, m, n \geq 0$,
- *elem_type*, $attr_i \in (N \cup NVAR)$,
- $pvar_i \in PVAR$,
- $val_i \in (\Sigma^* \cup SVAR)$,
- *ivar* $\in$ *IVAR*, and
- e_i are XML expressions on Ω_X.

The order of $pvar_i$ (*P*-variables) and that of pairs $attr_i= val_i$ (pairs of attribute name and value) are immaterial. □

By its definition, an XML expression is either an *E*-variable (XML-expression-variable) in *EVAR* or a tagged expression containing the following four components:

1. A tag name which could be a name in *N*, an *N*-variable (name-variable) in *NVAR* or an *I*-variable (intermediate-expression-variable) in *IVAR*;
2. A sequence of zero or more *P*-variables (attribute-value-pair-variable) in *PVAR*;
3. A sequence of zero or more attribute-value pairs, where an attribute could be either a name in *N* or an *N*-variable in *NVAR*, and a value be a string in Σ^* or an *S*-variable (string-variable) in *SVAR*;
4. An expression content which could be a string value (cf. Definition 5-3) or a sequence of zero or more subexpressions (cf. Definition 5-4, 5-5).

When an expression's tag name is represented by an *I*-variable, it must contain a sequence of zero or more subexpressions but neither a *P*-variable nor an attribute-value pair (cf. Definition 5-5). Intuitively, an *I*-variable is employed to represent an XML expression when its structure or nesting pattern is not fully known. For example, the expression <*ivar*> e_1 ... e_n </*ivar*>, where e_i are XML expressions, represents the XML expressions which contain the subexpression sequence e_1 ... e_n to an arbitrary depth.

Note:

1. The XML expressions without variable will be precisely called *ground XML expressions* or XML elements, while those with variables *non-ground XML expressions*.
2. An expression having the form

   <*elem_type* $attr_1=val_1$ ... $attr_k=val_k$ $pvar_1$ ... $pvar_m$> *val* </*elem_type*>

   or

   <*elem_type* $attr_1=val_1$ ... $attr_k=val_k$ $pvar_1$ ... $pvar_m$>e_1 ... e_n</*elem_type*>

   is often referred to as *elem_type* expression, while an expression

   <*ivar*>e_1 ... e_n</*ivar*>

   as *ivar* expression. For example, the expression

```
<Person SSN=$S:SSN>
      $E:PersonData
      <Mother>Mary Smith</Mother>
</Person>
```

is referred to as Person expression, while the subexpression <Mother>Mary Smith</Mother> nested inside that Person expression is referred to as Mother expression.

3. An expression

   <elem_type $attr_1$=val_1 ... $attr_k$=val_k $pvar_1$ ... $pvar_m$ > </elem_type>

   is considered to be identical to the empty-form expression

   <elem_type $attr_1$=val_1 ... $attr_k$=val_k $pvar_1$ ... $pvar_m$/>.

4. The parts

   <elem_type $attr_1$=val_1 ... $attr_k$=val_k $pvar_1$ ... $pvar_m$ >,

   </elem_type>

   and

   <elem_type $attr_1$=val_1 ... $attr_k$=val_k $pvar_1$ ... $pvar_m$/>

   will be simply called *tags* or, more specifically, *start-tag*, *end-tag* and *empty-element-tag*, respectively.

Definition 6 [$\mathcal{A}_X$, the set of XML expressions]

$\mathcal{A}_X$ is the set of all XML expressions on Ω_X. □

Example 1 As an example of non-ground XML expressions in $\mathcal{A}_X$, consider the following expression

```
<$I:Parent>
    <Father>Peter Smith</Father>
</$I:Parent>
```

Note that in this example:
- The given $I:Parent expression is intended to represent a class of XML expressions which encodes information about all the individuals having *Peter Smith* as their father. However, the exact structure, including tag names, list of attribute-value pairs and the nesting pattern of the expression containing the subexpression <Father>Peter Smith</Father> which encodes the individuals' father information, is unknown and represented by an *I*-variable $I:Parent.
- Father is a name in *N*.
- Peter Smith is a string in Σ^*. □

Definition 7 [$\mathcal{G}_X$, the set of ground XML expressions]

$\mathcal{G}_X$ is that subset of $\mathcal{A}_X$ which consists of all ground XML expressions in $\mathcal{A}_X$. □

Example 2 As an example of ground expressions in $\mathcal{G}_X$, consider the following XML expression encoding information about the individual with the name John Smith:

```
<Person ssn="99999">
    <Name>John Smith</Name>
    <Father>Peter Smith</Father>
    <Mother>Mary Smith</Mother>
</Person>
```

where – Person, ssn, Name, Father, and Mother are names in *N*.
– "99999", John Smith, Peter Smith and Mary Smith are strings in Ω^*. □

Example 2 shows that mappings between conventional XML elements and ground expressions in $\mathcal{G}_X$ are apparently straightforward, as no translation or modification is needed. Example 1 has demonstrated the the employment of various types of variables in XML expressions for the representation of a group or a class of XML elements with some common attributes or subelements.

4.3 Specializations

Definition 8 [ν_X, basic specialization mapping]

Let $\mathcal{C}_X$ be $(NVAR \times NVAR) \cup (SVAR \times SVAR) \cup (PVAR \times PVAR) \cup (EVAR \times EVAR) \cup (IVAR \times IVAR) \cup$
$(PVAR \times (NVAR \times SVAR \times PVAR)) \cup (EVAR \times (EVAR \times EVAR)) \cup ((PVAR \cup EVAR \cup IVAR) \times \{\varepsilon\}) \cup$
$(NVAR \times N) \cup (SVAR \times \Sigma^*) \cup (EVAR \times \mathcal{A}_X) \cup (IVAR \times (NVAR \times PVAR \times EVAR \times EVAR \times IVAR))$.
Elements of $\mathcal{C}_X$ are called *basic specializations*. Let $a \in \mathcal{A}_X$. The *basic specialization mapping* ν_X: $\mathcal{C}_X \rightarrow$ *partial_map*($\mathcal{A}_X$) is defined in Table 1. □

Table 1. The basic specialization mapping ν_X.

Type	Basic Specialization c in $\mathcal{C}_X$	Applicability Condition	$\nu_X(c)(a)$ is Obtained from a by
1. Variable Renaming	$c = (var_1, var_2) \in (NVAR \times NVAR) \cup (SVAR \times SVAR) \cup (PVAR \times PVAR) \cup (EVAR \times EVAR) \cup (IVAR \times IVAR)$	-	Replacement of all occurrences of var_1 in a by var_2.
2. Variable Expansion			
2.1 P-variable	$c = (pvar_1, (nvar, svar, pvar_2)) \in PVAR \times (NVAR \times SVAR \times PVAR)$	For every tag in a containing $pvar_1$, that tag does not contain $nvar$ as one of its attribute name	Replacement of all occurrences of $pvar_1$ in a by the sequence of the pair $nvar$=$svar$ and the P-variable $pvar_2$.
2.2 E-variable	$c = (evar, (evar_1, evar_2)) \in EVAR \times (EVAR \times EVAR)$	-	Replacement of all occurrences of $evar$ in a by the sequence $evar_1$ $evar_2$.
3. Variable Removal			
3.1. P- and E-variable	$c = (var, \varepsilon) \in (PVAR \cup EVAR) \times \{\varepsilon\}$, where ε denotes the null symbol	-	Removal of all occurrences of var in a.
3.2. I-variable	$c = (ivar, \varepsilon) \in IVAR \times \{\varepsilon\}$, where ε denotes the null symbol	-	Removal of all occurrences of <*ivar*> and </*ivar*> in a.
4. Variable Instantiation			
4.1. N-variable	$c = (nvar, n) \in NVAR \times N$	For every tag in a containing $nvar$ as one of its attribute name, that tag does not contain an attribute named n	Replacement of all occurrences of $nvar$ in a by n.
4.2. S- and E-variable	$c = (var, val) \in (SVAR \times \Sigma^*) \cup (EVAR \times \mathcal{A}_X)$	-	Replacement of all occurrences of var in a by val.
4.3. I-variable	$c = (ivar_1, (nvar, pvar, evar_1, evar_2, ivar_2)) \in IVAR \times (NVAR \times PVAR \times EVAR \times EVAR \times IVAR)$	-	Replacement of each occurrence of the $ivar_1$ expression nested in a by the expression of the form <*nvar pvar*> *evar*$_1$ <*ivar*$_2$> *content* </*ivar*$_2$> *evar*$_2$ </*nvar*> where *content* represents the content of that occurrence of *ivar* expression.

By the definitions of $\mathcal{C}_X$ and the basic specialization mapping ν_X, which is used to determine the application of each basic specialization $c \in \mathcal{C}_X$ to an expression $a \in \mathcal{A}_X$, there are four types of basic specializations:

1. Rename variables.
2. Expand a P- or an E-variable into a sequence of variables of their respective types.
3. Remove P- or E-variables.
4. Instantiate variables to some values which correspond to the types of the variables.

Definition 9 [$\mathcal{S}_X$, the set of specializations, and the specialization mapping μ_X]

Let $\mathcal{S}_X = \mathcal{C}_X^*$, i.e., the set of all sequences on $\mathcal{C}_X$. Based on ν_X, the specialization mapping μ_X: $\mathcal{S}_X \rightarrow$ *partial_map*($\mathcal{A}_X$) is defined by:

$\mu_X(\lambda)(a) = a$, where λ denotes the null sequence,

$\mu_X(c \cdot s)(a) = \mu_X(s)(\nu_X(c)(a))$, where $c \in \mathcal{C}_X$, $s \in \mathcal{S}_X$ and $a \in \mathcal{A}_X$.

Note that $\mu_X(s)(a)$ is defined only if all elements in s are successively applicable to a. □

Example 3 This example demonstrates that the expression shown in Example 1 can be specialized to the one given in Example 2 by means of the specialization operator μ_X. Let $\theta \in \mathcal{S}_X$ be the sequence ($c_1\ c_2\ c_3\ \ldots\ c_9$), defined in Table 2, and a_1 denote the expression in Example 1. Table 2 illustrates the derivation of the expression $a_1\theta$.

Table 2. Application of θ to a_1.

Definition of Basic Specialization $c_i \in \mathcal{C}_X$	**Application of**	**Resulting XML Expression**
c_1 = ($I:Parent, ($N:Person, $P:PersonAttr1, $E:PersonData1, $P:PersonData2, $I:Parent2)) $\in IVAR \times (NVAR \times PVAR \times EVAR \times EVAR \times IVAR)$	$\nu_X(c_1)$ to a_1	a_2 = <$N:Person $P:PersonAttr1> $E:PersonData1 <$I:Parent2> <Father>Peter Smith</Father> </$I:Parent2> $E:PersonData2 </Person>
c_2 = ($E:PersonData1, <Name>John Smith</Name>) $\in EVAR \times \mathcal{A}_X$	$\nu_X(c_2)$ to a_2	a_3 = <$N:Person $P:PersonAttr1> <Name>John Smith</Name> <$I:Parent2> <Father>Peter Smith</Father> </$I:Parent2> $E:PersonData2 </Person>
c_3 = ($E:PersonData2, <Mother>Mary Smith</Mother>) $\in EVAR \times \mathcal{A}_X$	$\nu_X(c_3)$ to a_3	a_4 = <$N:Person $P:PersonAttr1> <Name>John Smith</Name> <$I:Parent2> <Father>Peter Smith</Father> </$I:Parent2> <Mother>Mary Smith</Mother> </Person>
c_4 = ($I:Parent2, ε) $\in IVAR \times \{\varepsilon\}$	$\nu_X(c_4)$ to a_4	a_5 = <$N:Person $P:PersonAttr1> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>
c_5 = ($N:Person, Person) $\in NVAR \times N$	$\nu_X(c_5)$ to a_5	a_6 = <Person $P:PersonAttr1> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>
c_6 = ($P:PersonAttr1, ($N:SSN, $S:SSN, $P:PersonAttr2)) $\in PVAR \times (NVAR \times SVAR \times PVAR)$	$\nu_X(c_6)$ to a_6	a_7 = <Person $N:SSN=$S:SSN $P:PersonAttr2> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>
c_7 = ($N:SSN, ssn) $\in NVAR \times N$	$\nu_X(c_7)$ to a_7	a_8 = <Person ssn=$S:SSN $P:PersonAttr2> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>
c_8 = ($S:SSN, "99999") $\in SVAR \times C^*$	$\nu_X(c_8)$ to a_8	a_9 = <Person ssn="99999" $P:PersonAttr2> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>
c_9 = ($P:PersonAttr2, ε) $\in PVAR \times \{\varepsilon\}$	$\nu_X(c_9)$ to a_9	a_{10} = <Person ssn="99999"> <Name>John Smith</Name> <Father>Peter Smith</Father> <Mother>Mary Smith</Mother> </Person>

In other words, by successive applications of the members of θ to a_1, one obtains the element $a_1\theta = a_{10}$, which is the one shown in Example 2. Also, note that θ is not the only specialization that can specialize a_1 to a_{10}. For example, an alternative is $\theta' = (c_1\ c_3\ c_2\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9) \in \mathcal{S}_X$. □

4.4 Specialization System for XML Expressions and XML Declarative Description

In the sequel, let $\Gamma_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{S}_X, \mu_X \rangle$. The definitions of $\mathcal{A}_X$, $\mathcal{G}_X$, $\mathcal{S}_X$ and μ_X readily show that Γ_X is a specialization system since it satisfies the three requirements of specialization systems.

Definition 10 [Specialization system for XML expressions]

The *specialization system for XML expressions* is $\Gamma_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{S}_X, \mu_X \rangle$. □

Proposition 1 The specialization system for XML expressions in Definition 10 satisfies the three requirements of specialization systems, i.e.,

1. $\forall s_1, s_2 \in \mathcal{S}_X, \exists s \in \mathcal{S}_X : \mu_X(s) = \mu_X(s_1) \circ \mu_X(s_2)$,
2. $\exists s \in \mathcal{S}_X, \forall a \in \mathcal{A}_X : \mu_X(s)(a) = a$,
3. $\mathcal{G}_X \subset \mathcal{A}_X$. □

Proof

1. Let $s_1 = (d_1\ d_2 \ldots d_n)$ and $s_2 = d_1'\ d_2' \ldots d_m'$ be elements of $\mathcal{S}_X$. It follows immediately from the definitions of $\mathcal{S}_X$ and μ_X that there exists $s = (d_1'\ d_2' \ldots d_m'\ d_1\ d_2 \ldots d_n)$ such that $\mu_X(s) = \mu_X(s_1) \circ \mu_X(s_2)$.
2. Obviously, $\mu_X(\lambda)(a) = a$, for each $a \in \mathcal{A}_X$, where λ is the null sequence.
3. By Definition 7, $\mathcal{G}_X$ is a subset of $\mathcal{A}_X$. ■

After the specialization system for XML expressions is defined, the definitions of *XML clauses*, *XML declarative descriptions* (*XML-DD*) and the *declarative semantics* of an XML declarative description are obtained directly from the DD theory (cf. Section 3).

5 XML Document Modeling

XML-DD theory, formulated in Section 4, will be applied to the management of XML documents

A conventional XML element is represented directly as a ground XML expression in $\mathcal{G}_X$. A class of XML elements, which share certain similar components and structures, can also be represented as an XML expression with variables. These variables are used to represent unknown or similar components (which could be tag names, lists of attribute-value pairs, subelements or nesting structures) shared by the elements in the class. For instance, in order to represent a set of XML elements encoding information about individuals born in 1975, one can simply construct an XML expression with a Person tag name containing a BirthYear subexpression, the content of which is 1975. Other information that may vary, such as their Names, Parents and SSNs, is expressed implicitly through the use of variables. Thus, an XML document, comprising a set of n XML elements, is directly mapped into a set of n unit clauses, each of which describes its corresponding XML element in the document. Besides this simple and straightforward representation of XML elements, the proposed approach also permits to define in terms of non-unit clauses *integrity constraints*, e.g., data integrity, path and type constraints [11], as well as *knowledge* and *complex relationships* among XML elements, e.g., referential relationship.

A collection of XML documents, each of which contains a sequence of XML elements probably conforming to different DTDs, can be modeled by a description P consisting of *unit clauses* and *non-unit clauses*. Intuitively, for $\theta \in \mathcal{S}_X$ and a unit clause in P of the form $(H \leftarrow .)$, if $H\theta$ is a ground XML expression, then $H\theta$ will be included in the meaning of P, while a non-unit clause in P of the form $(H \leftarrow B_1, \ldots, B_n)$, $n \geq 1$, is interpreted as follows, for every $\theta \in \mathcal{S}_X$ that makes $B_1\theta, \ldots, B_n\theta$ true with respect to the meaning of P, the expression $H\theta$ will be derived and included in the meaning of P. In other words, for every binding of variables contained in such a clause that makes all the constraints and relationships specified in the body of that clause satisfied, an expression represented by the head of the clause will be included in the meaning of P. Therefore, the declarative meaning of P yields all the directly represented XML elements in the document collection, i.e., those expressed by unit clauses, together with all the derived ones, which may be restricted by constraints. Thus, in addition to the simple queries, which are only based on text pattern matching, one can also issue selective, complex queries about this derived information [4]. Moreover, by incorporation of set-of references, the proposed approach readily enables formulation and evaluation of group-by and aggregate queries. Detailed discussions on the formulation and the processing of XML queries under the proposed approach are presented in [4].

It is important to emphasize that the proposed approach also provides simple means for a restriction of XML data to those which satisfy a given DTD. They are materialized by directly translation of a DTD into a corresponding set of clauses for the checking of the validity of an XML document with respect to the DTD. The theoretical details of such formalization are available in [4].

```
<!ELEMENT   Person    (Name, BirthYear, Parent?)>
<!ATTLIST   Person    ssn  ID #REQUIRED
                      state  IDREF #REQUIRED
                      gender  (Male | Female) #REQUIRED>
<!ELEMENT   Name      (#PCDATA)>
<!ELEMENT   BirthYear (#PCDATA)>
<!ELEMENT   Parent    EMPTY>
<!ATTLIST   Parent    father  IDREF  #IMPLIED
                      mother  IDREF  #IMPLIED >
<!ELEMENT   State     (Name)>
<!ATTLIST   State     id  ID #REQUIRED>
```

Fig. 1. An XML DTD example

Example 4 Let P be a description which represents an XML document encoding demographic data and conforming to the DTD given in Fig. 1. Assume that such a document contains three Person elements and a State element and P comprises the following nine clauses, denoted by $C_1 - C_9$:

```
C1: <Person ssn="99999" state="NY" gender="Male">
        <Name>John Smith</Name>
        <BirthYear>1975</BirthYear>
        <Parent  mother="55555" />
    </Person>          ←       .

C2: <Person ssn="55555" state="NY"  gender="Female">
        <Name>Mary Smith</Name>
        <BirthYear>1950</BirthYear>
        <Parent  father="11111" />
    </Person>          ←       .

C3: <Person ssn="11111" state="NY"  gender="Male">
        <Name>Tom Black</Name>
        <BirthYear>1920</BirthYear>
    </Person>          ←       .

C4: <State  id="NY">
        <Name>New York</Name>
    </State>           ←       .

C5: <ValidPerson  ssn=$S:PersonSSN  state=$S:StateId  $P:PersonAttr>
        $E:PersonData
    </ValidPerson>
             ←   <Person ssn=$S:PersonSSN  state=$S:StateId  $P:PersonAttr>
                     $E:PersonData
                 </Person>,
                 <ValidState  id=$S:StateId>
                     $E:StateData
                 </ValidState>.

C6: <Ancestor  ancestor=$S:FatherSSN   descendent=$S:PersonSSN/>
             ←   <ValidPerson  ssn=$S:PersonSSN  $P:PersonAttr>
                     $E:PersonSubelement
                     <Parent  father=$S:FatherSSN  $P:ParentAttr/>
                 </ValidPerson>.
```

```
C7: <Ancestor ancestor=$S:MotherSSN descendent=$S:PersonSSN/>
        ←   <ValidPerson ssn=$S:PersonSSN $P:PersonAttr>
                $E:PersonSubelement
                <Parent mother=$S:MotherSSN $P:ParentAttr/>
            </ValidPerson>.

C8: <Ancestor ancestor=$S:FatherSSN descendent=$S:DescendentSSN/>
        ←   <Ancestor ancestor=$S:AncestorSSN
                      descendent=$S:DescendentSSN/>,

            <ValidPerson ssn=$S:AncestorSSN $P:PersonAttr>
                $E:PersonSubelement
                <Parent father=$S:FatherSSN $P:ParentAttr/>
            </ValidPerson>.

C9: <Ancestor ancestor=$S:MotherSSN descendent=$S:DescendentSSN/>
        ←   <Ancestor ancestor=$S:AncestorSSN
                      descendent=$S:DescendentSSN/>,

            <ValidPerson ssn=$S:AncestorSSN $P:PersonAttr>
                $E:PersonSubelement
                <Parent mother=$S:MotherSSN $P:ParentAttr/>
            </ValidPerson>.
```

Clauses $C_1 - C_3$ and C_4 represent Person and State elements in the document, respectively; clause C_5 defines an integrity constraint on the Person elements; and clauses $C_6 - C_9$ represent knowledge about ancestor relationship.

The given DTD shows that the state attribute belonging to the Person element is an attribute of type IDREF intended to refer to a State element. With respect to such a referential integrity constraint, clause C_5 specifies that a Person element is valid if the value of the state attribute matches the id of some particular valid State element. In addition to referential integrity constraints, other kinds of integrity constraints can be similarly defined; for instance, to restrict that

(i) the values of the father and mother attributes in a Parent element match the ssn of two particular Person elements,
(ii) a Person must be younger than his/her Parents, i.e., to assure that such Person's BirthYear must be greater than the Parents' BirthYears, and
(iii) the gender of a Person referred to as a father must be Male and a mother Female.

Clauses $C_6 - C_9$ derive ancestor relationships among the individuals in the collection. Clauses C_6 and C_7 specify that both father and mother of an individual are ancestors of such an individual. Clauses C_8 and C_9 recursively specify that the father and the mother of an individual's ancestors are also the individual's ancestors. This ancestor relationship represents an example of complex, recursive relationships which can be simply expressed in the proposed approach. Synonym relationships can be dealt with in a similar manner. □

6 Comparisons

Compared with other models, e.g., those based on graphs, hedge automaton and functional programming, the proposed data model for XML documents provides a more direct and succinct insight into computation of and reasoning with XML data.

From the reasoning point of view, employment of an existing deductive database approach [14, 21], such as Datalog, and some of its extensions, e.g., LDL and RelationLog, to XML document modeling may be proposed. However, since such an approach provides inexpressive flat structures and cannot directly support the complex, nesting structure common in XML syntax, it exhibits a significant problem in modeling and representing XML data. An XML element must be translated and expressed in terms of its permitted representations only, e.g., as a set of atomic formulas in Datalog. Identical XML elements may have several corresponding representations depending on the employed translational scheme. Moreover, the difficulties encountered during application of the relational approach to modeling XML data remain inherent in deductive database approaches. In addition, it is difficult to express a query when the document schema, element tag name or the nesting level at which the required element occurs is unknown. Such an approach therefore trades the structural information and the expressive power underlying in XML documents for an application of an existing theory.

Table 3 deliberately compares several important aspects of XML data management.

Table 3. Comparison of approaches to XML data management.

Characteristics/ Functionalities	*Approaches*				
	Graph-Based	**Hedge Automaton**	**Functional Programming**	**Datalog**	**Declarative Description**
XML data representation	Rooted, edge-labeled graphs	Hedges	Typed feature terms	Atomic formulas or relations	XML expressions
Integrity constraint support	Yes, by integration of first-order logic theory. However, the support is limited to only for path and type constraints.	No	No	Yes, by means of built-in predicates	Yes, by means of constraints in the description theory
Query processing support	Yes	Yes	Yes, by means of list comprehension evaluation	Yes, but rather difficult to deal with very complicated structured data	Yes, by means of equivalent transformation of XML-DDs
DTD validation and restriction support	No	Yes, by means of the hedge automaton	No	Yes, by means of Datalog programs	Yes, by means of descriptions
Possession of inference/ reasoning capability	Yes, but limited to only reasoning about path and type constraints	Yes, by means of the hedge automaton	Yes, by means of list comprehension evaluation	Yes	Yes
Regular path expression support	Yes	No	Yes	Indirect support by employment of variables and recursions in Datalog rules	Indirect support by employment of variables and recursions in clauses
Provision of succinct representation and operation of XML data	No	No	No	No	Yes

7 Conclusions

An expressive, declarative data model has been developed, founded on a theoretical basis upon which representation and computation of as well as reasoning with XML data can be carried out in a uniform and succinct manner. Integration of the proposed data model with an appropriate computational paradigm, e.g., *Equivalent Transformation* (*ET*) [3], allows efficient manipulation and transformation of XML documents, query evaluation and validation of XML data against some particular DTDs.

In order to help demonstrate and evaluate the effectiveness of the proposed approach, *XML-ETC Engine* – an easy-to-use, Web-based XML processor – has been implemented under *ETC* – a compiler for programming in ET paradigm. The system has been tested against a small XML database with good performance; a more thorough evaluation of the system with a large collection of XML documents is underway.

Acknowledgement

This work was supported in part by Thailand Research Fund.

References

1. Akama, K.: Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advances in Software Science and Technology*, Vol. 5 (1993) 45-63
2. Akama, K.: Declarative Description with References and Equivalent Transformation of Negative References. *Technical Report*, Department of Information Engineering, Hokkaido University, Japan (1998)

3. Akama, K., Shimitsu, T., Miyamoto, E. Solving Problems by Equivalent Transformation of Declarative Programs. *Journal of the Japanese Society of Artificial Intelligence*, Vol. 13, No. 6 (1998) 944-952 (in Japanese).
4. Akama, K., Anutariya, C., Wuwongse, V. and Nantajeewarawat, E.: A Foundation for XML Document Databases: Query Formulation and Evaluation. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)
5. Anutariya, C., Wuwongse, V., Nantajeewarawat, E. and Akama, K.: Towards a Foundation for XML Document Databases. *Proceedings of 1st International Conference on Electronic Commerce and Web Technologies (EC-Web 2000)*, London, UK. Lecture Notes in Computer Science, Springer Verlag (2000) (to appear)
6. Anutariya, C., Wuwongse, V., Akama, K. and Nantajeewarawat, E.: A Foundation for XML Document Databases: DTD Modeling. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)
7. Apparao, V., Et Al: Document Object Model (DOM) Level 1 Specification Version 1.0, October 1998. W3C Recommendation (1998) Available at http://www.w3.org/TR/REC-DOM-Level-1/
8. Beech, D., Malhotra, A., Rys, M.: A Formal Data Model and Algebra for XML. *W3C XML Query Working Group Note*, September 1999 (1999)
9. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, February 1998. *W3C Recommendation* (1998) Available at http://www.w3.org/TR/REC-xml
10. Buneman, P., Deutsch, A., Tan, W.C.: A Deterministic Model for Semi-Structured Data. *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* (1998) Available at http://db.cis.upenn.edu/DL/icdt.ps.gz
11. Buneman, P., Fan, W., Weinstein, S.: Interaction between Path and Type Constraints. *Proc. ACM Symposium on Principles of Database Systems, PODS* (1999) Available at ftp://ftp.cis.upenn.edu/pub/papers/db-research/pods99.ps.gz
12. Fernández, M., Siméon, J., Suciu, D. and Wadler, P.: A Data Model and Algebra for XML Query. *Draft Manuscript* (1999) Available at http://www.cs.bell-labs.com/~wadler/topics/xml.html#algebra
13. Goldman, R., McHugh, J., Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Proc. 2nd International Workshop on the Web and Databases, WebDB '99*, Philadelphia, Pennsylvania (1999)
14. Liu, M.: Deductive Database Languages: Problems and Solutions. *ACM Computing Surveys*, Vol. 31, No. 1 (1999)
15. Murata, M.: Forest-regular Languages and Tree-regular Languages. *Technical Report*, Fuji Xerox Information Systems (1995) Available at http://www.geocities.com/ResearchTriangle/Lab/6259/prelim1.pdf
16. Murata, M.: Hedge Automata: A Formal Model for XML Schemata. *Technical Report*, Fuji Xerox Information Systems, (1995) Available at http://www.geocities.com/ResearchTriangle/Lab/6259/hedge_nice.pdf
17. Murata, M.: Transformation of Documents and Schemas by Patterns and Contextual Conditions. *Principles of Document Processing. Proc. 3rd International Workshop* (1996) Available at http://www.geocities.com/ResearchTriangle/Lab/6259/podp96.pdf
18. Murata, M. DTD Transformation by Patterns and Contextual Conditions. *Proc. SGML/XML '97 Conference* (1997) Available at http://www.fxis.co.jp/DMS/sgml/xml/sgmlxml97.html
19. McHugh, J., Abiteboul, S., Goldman, R., Quass, D. and Widom, J.: Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, Vol. 26, No. 3 (1997) 54-66 Available at ftp://db.stanford.edu/pub/papers/lore97.ps
20. Sacks-Davis, R., Arnold-Moore, T., Zobel, J.: Database Systems for Structured Documents. *IEICE Transactions on Information and System*, Vol. E78-D, No. 11 (1995) 1335-1341
21. Ullman, J. D.: Principles of Database and Knowledge-Base Systems. Computer Science Press, Maryland (1988)

set of directed edges and vertices) needs to be defined. On the other hand, if a document's semantics is taken into account, the document is then represented as a set of related tuples contained in their corresponding relations; different documents are probably modeled differently, whence resulting in a very complex and huge database schemas. A DTD is then formalized as a set of Datalog rules, where predicates contained in each rule have structures corresponding to the selected document's relational representation.

A new approach is presented to the modeling of XML DTDs by employment of *XML Declarative Description* (*XML-DD*) *theory* [6,17] which serves as a foundation for the representation and computation of as well as reasoning with XML data. In this approach, an XML DTD is represented as an XML-DD which comprises a set of *clauses*, to be referred specifically as *DTD clauses*. Such an XML-DD is obtained directly by translation of each of the element type and attribute-list declarations contained in the DTD into a corresponding set of DTD clauses and consequence combination of these sets. This formalism also facilitates the development of a simple mechanism for convenient determination of whether a given XML element/document conforms to the grammar imposed by the DTD or not. Besides providing means for restriction of a document's syntactical constraints, this formalism can also be applied to enforce various kinds of integrity constraints which are not expressible in terms of DTDs but are extremely important in query evaluation [5] and optimization, e.g., atomic typing (char, integer, float, etc.) and restrictions on the type of IDREF(S).

Section 2 summarizes the XML-DD theory developed in [6,17] and presents its extension for dealing with *references*, Section 3 develops a formalism for modeling XML DTDs, Section 4 presents an approach to validation of an element/document against a particular DTD, and Section 5 concludes and outlines future research.

2 XML Declarative Description Theory

2.1 Declarative Description Data Model for XML Documents

In the declarative description data model for XML documents [17], developed by employment of *Declarative Description* (*DD*) theory [1,3,4], the definition of an XML element is formally extended by incorporation of variables in order to represent inherent implicit information and enhance its expressive power. Such extended XML elements, referred to as *XML expressions*, have a similar form to XML elements except that they can carry variables. XML expressions without variable will be called *ground XML expressions* or XML elements, those with variables *non-ground XML expressions*. There are several kinds of variables useful for the representation of implicit information contained in XML expressions: *name-variables* (*N-variables*), *string-variables* (*S-variables*), *attribute-value-pair-variables* (*P-variables*), *XML-expression-variables* (*E-variables*) and *intermediate-expression-variables* (*I-variables*). Every variable is preceded by '$' together with a character specifying its type, i.e., '$N', '$S', '$P', '$E' or '$I'.

An *XML expression alphabet* Ω_X comprises the symbols in the following sets: Σ (a set of *characters*), N (a set of *names*), $NVAR$ (a set of *N-variables*), $SVAR$ (a set of *S-variables*), $PVAR$ (a set of *P-variables*), $EVAR$ (a set of *E-variables*) and $IVAR$ (a set of *I-variables*).

Intuitively, an N-variable will be instantiated to an element type or an attribute name, an S-variable to a string on Σ^*, a P-variable to a sequence of attribute-value pairs, an E-variable to a sequence of XML expressions, an I-variable to a part of an XML expression. Such variable instantiations are defined by means of *basic specializations* each of which is a pair of the form (*var*, *val*), where *var* is the variable to be specialized and *val* a value or tuple of values describing the resulting structure. There are four types of basic specializations:

1. rename variables,
2. expand a P- or an E-variable into a sequence of variables of their respective types,
3. remove P-, E- or I-variables, and
4. instantiate variables to some values which correspond to the types of the variables.

Let $\mathcal{A}_X$ denote the set of all XML expressions on Ω_X, $\mathcal{G}_X$ the subset of $\mathcal{A}_X$ which comprises all ground XML expressions in $\mathcal{A}_X$, $\mathcal{C}_X$ the set of basic specializations and $\nu_X : \mathcal{C}_X \rightarrow partial\_map(\mathcal{A}_X)$ the mapping from $\mathcal{C}_X$ to the set of all partial mappings on $\mathcal{A}_X$ which determines for each basic specialization c in $\mathcal{C}_X$ the change of elements in $\mathcal{A}_X$ caused by c. Let $\Delta_X = \langle \mathcal{A}_X, \mathcal{G}_X, \mathcal{C}_X, \nu_X \rangle$ be a *specialization generation system*, which will be used to define a *specialization system* characterizing the data structure of XML expressions and sets of XML expressions.

Let V be a set of *set variables*.

$$\mathcal{A} = \mathcal{A}_X \cup 2^{(\mathcal{A}_X \cup V)},\ \mathcal{G} = \mathcal{G}_X \cup 2^{\mathcal{G}_X},\ \mathcal{C} = \mathcal{C}_X \cup (V \times 2^{(\mathcal{A}_X \cup V)}),$$

and

$$\nu: \mathcal{C} \rightarrow partial\_map(\mathcal{A})$$

the mapping from $\mathcal{C}$ to the set of all partial mappings on $\mathcal{A}$ which determines for each basic specialization c in $\mathcal{C}$ the change of objects in $\mathcal{A}$ caused by c such that

1. If $c \in \mathcal{C}_X$ and $a \in \mathcal{A}_X$,
   then $\nu(c)(a) = \nu_X(c)(a)$.
2. If $c \in (\mathcal{C} - \mathcal{C}_X)$ and $a \in \mathcal{A}_X$,
   then $\nu(c)(a) = a$.
3. If $c \in \mathcal{C}_X$, $S = \{a_1, \ldots, a_m, v_1, \ldots, v_n\} \in (\mathcal{A} - \mathcal{A}_X)$, $a_i \in \mathcal{A}_X$ and $v_j \in V$,
   then $\nu(c)(S) = \{\nu_X(c)(a_1), \ldots, \nu_X(c)(a_m), v_1, \ldots, v_n\}$
4. If $c = (v, R) \in (\mathcal{C} - \mathcal{C}_X)$ and $S = \{x_1, \ldots, x_n, v\} \in (\mathcal{A} - \mathcal{A}_X)$,
   then $\nu(c)(S) = \{x_1, \ldots, x_n\} \cup R$.

In order to distinguish a set variable from other types of variables, every set variable in V will be preceded by '\$V'. In the sequel, let

$$\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle \tag{1}$$

be a *specialization system for XML expressions with flat sets*, where $\mathcal{S} = \mathcal{C}^*$, and $\mu : \mathcal{S} \rightarrow partial\_map(\mathcal{A})$ such that, for $a \in \mathcal{A}$

$\mu(\lambda)(a) = a$, where λ denotes the null sequence,
$\mu(c \cdot s)(a) = \mu(s)(\nu(c)(a))$, where $c \in \mathcal{C}$ and $s \in \mathcal{S}$

Elements of $\mathcal{A}$, $\mathcal{G}$ and $\mathcal{S}$ are called *objects*, *ground objects* and *specializations*, respectively. The mapping μ is called the *specialization mapping*. Note that when μ is clear from the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$, and, for $X \in V$, a singleton $\{X\}$ will be written as X.

The definition of *XML declarative description with references* together with its related concepts can be given in terms of $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$.

2.2 XML Declarative Description with References

An *XML declarative description* on Γ, simply called an *XML-DD* or a *description*, is a (possibly infinite) set of *clauses* on Γ, each in the form

$$H \leftarrow B_1, B_2, \ldots, B_n. \tag{2}$$

where $n \geq 0$, H is an XML expression in $\mathcal{A}_X$, and B_i an XML expression in $\mathcal{A}_X$, a *constraint* or a *reference* on Γ. H is called the *head* and $(B_1, B_2, \ldots, B_n)$ the *body* of the clause. Such a clause, if $n = 0$, is specifically called a *unit clause*, and, if $n > 0$, a *non-unit clause*.

Let K be a set of *constraint predicates*. A *constraint* on Γ is a formula $q(a_1, \ldots, a_n)$, where $n > 0$, q is a constraint predicate in K and a_i an object in $\mathcal{A}$. Given a *ground constraint* $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}$, its truth or falsity is assumed to be predetermined. Denote the set of all true ground constraints by *Tcon*. The notion of constraints introduced here is useful for defining restrictions on objects in $\mathcal{A}$, i.e., both on XML expressions in $\mathcal{A}_X$ and on sets of XML expressions in $2^{(\mathcal{A}_X \cup V)}$.

Let F be the set of all mappings: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$, the elements of which are called *reference functions*. A *reference* on Γ is a triple $r = \langle a, f, P \rangle$ of an object a in $\mathcal{A}$, a reference function f in F and a description P, which will be called the *referred description* of r. A reference $\langle g, f, P \rangle$ is a *ground reference* iff $g \in \mathcal{G}$. Such a notion of references introduced here together with appropriate definitions of *id-*, *idref-* and *idrefs-reference functions* in F (cf. Definition 9, Subsection 3.2) will be employed to restrict *uniqueness* and *referential* constraints imposed by attributes of types ID and IDREF(S), respectively (cf. Definition 11, Subsection 3.2). For instance, given an XML element identified by x, in order to ensure the uniqueness of such an identifier x with respect to a particular XML document represented by a description P, an *id reference* $\langle$<id value=x/>, $id_{\$v,J}$, $P\rangle$ is formulated.

Given a specialization $\theta \in \mathcal{S}$, application of θ to a constraint $q(a_1, \ldots, a_n)$ is the constraint $q(a_1\theta, \ldots, a_n\theta)$, to a reference $\langle a, f, P \rangle$ the reference $\langle a, f, P \rangle\theta = \langle a\theta, f, P \rangle$ and to a clause $(H \leftarrow B_1, B_2, \ldots, B_n)$ the clause $(H\theta \leftarrow B_1\theta, B_2\theta, \ldots, B_n\theta)$. The head of a clause C will be denoted by $head(C)$ and the set of all objects (XML expressions), constraints and references in the body of C by $object(C)$, $con(C)$ and $ref(C)$, respectively. Let $body(C) = object(C) \cup con(C) \cup ref(C)$. A clause C is a *ground clause* iff C comprises only ground objects, ground constraints and ground references.

Let C be a clause (either unit or non-unit clause) and P a description on Γ. The *height* of C and P, denoted by $hgt(C)$ and $hgt(P)$, respectively, are defined as follows:

1. The height of the clause C is zero if C contains no reference, i.e., if $ref(C) = \emptyset$.
2. If the clause C contains references, its height is equal to the maximum height of the all referred descriptions contained in its body plus one.
3. The height of the description P is the maximum height of all the clauses in P.

Let P be a description on Γ. The meaning of P, denoted by $\mathcal{M}(P)$, is defined inductively as follows:

1. Given the meaning, $\mathcal{M}(Q)$, of a description Q with the height m, a reference $r = \langle g, f, Q\rangle$ is a true reference iff $g \in f(\mathcal{M}(Q))$. For any $m \geq 0$, define $Tref(m)$ as the set of all true references the heights of the referred description of which are smaller than or equal to m, i.e.:

$$Tref(m) = \{\langle g, f, R\rangle \mid g \in \mathcal{G},\ f \in F,\ hgt(R) \leq m,\ g \in f(\mathcal{M}(R))\} \qquad (3)$$

2. The meaning, $\mathcal{M}(P)$, of the description P with the height $m + 1$ is a set of ground XML expressions defined by

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty}[T_P]^n(\emptyset) \qquad (4)$$

where $\emptyset$ is the empty set, $[T_P]^n(\emptyset) = T_P([T_P]^{n-1}(\emptyset))$ and the mapping T_P: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$ is defined as follows: For each $X \subset \mathcal{G}$, $g \in T_P(X)$ iff there exist a clause $C \in P$ and a specialization $\theta \in \mathcal{S}$ such that $C\theta$ is a ground clause the head of which is g and all the objects, constraints and references in the body of which belong to X, $Tcon$ and $Tref(n)$, for some $n \leq m$, respectively, i.e.:

$$\begin{aligned} T_P(X) = \{head(C\theta) \mid C \in P,\ \ \theta \in \mathcal{S},\ \ & C\theta \text{ is a ground clause,} \\ & object(C\theta) \subset X,\ \ con(C\theta) \subset Tcon, \\ & ref(C\theta) \subset Tref(n),\ \ n \leq m\ \} \end{aligned} \qquad (5)$$

Intuitively, given a description P, its meaning, $\mathcal{M}(P)$, is a set of all the ground XML expressions which can be derived from the clauses in P. In other words, given a clause $C = (H \leftarrow B_1, B_2, \ldots, B_n)$, $n \geq 0$, in P, for every $\theta \in \mathcal{S}$ that makes $B_1\theta$, $B_2\theta$, ..., $B_n\theta$ true with respect to the meaning of P, the expression $H\theta$ will be derived and included in the meaning of P.

3 XML DTD Modeling

This section employs the XML-DD theory, formulated in Section 2, to model XML DTDs.

3.1 Element Type and Attribute-List Declarations

Element type and *attribute-list declarations* are two essential declarations contained in an XML DTD, used to define the ordering and structuring of elements in a document. An element type declaration typically specifies the element's content model. In other words, it provides a grammar regulating the structure of the element's content which could be empty, character data or a valid sequence of the allowed types of child elements. An attribute-list declaration specifies the names, data types as well as default values (if any) of attributes associated with a given element type.

XML elements' content models can be categorized into three classes: *empty*, *simple* and *complex* (or *nested*) *content models*. An element type has empty content if elements of that type are empty, i.e., they are encoded by empty-element tags only, it has simple content if elements of that type contain merely character data, and it has complex content if elements of that type contain a sequence of one or more child elements. For these three classes of content models, there are also three corresponding forms of element type declarations: empty, simple and complex forms. Each form is used to declare element types with the respective content model, i.e., an empty-content element type is declared by an empty-formed declaration, a simple-content element type by a simple-formed declaration and a complex-content element type by a complex-formed declaration which employs *content particles* (simply referred to as *particles*) to constrain the element's content.

Given below is the formal definition of content particles which will be used in the definition of complex-formed element type declarations (cf. Definition 2-3).

Definition 1 [Content particles]

A content particle on a set of names N takes one of the forms:

1. *Unqualified content particle*
   1.1. *atomic form*: *elem-type*
   1.2. *choice-list form*: $(cp_1 \mid \dots \mid cp_n)$
   1.3. *sequence-list form*: $(cp_1 , \dots , cp_n)$
2. *Qualified content particle*:
   2.1. *?-form*: $cp\,?$
   2.2. *+-form*: $cp\,+$
   2.3. **-form*: $cp\,*$

where – *elem-type* is an element type in N,
– $n > 1$,
– cp_i is a content particle,
– cp is an unqualified content particle.

Let CP be the set of all content particles on N. □

Apparently, a content particle is simply a *regular expression* over element types in N.

Definition 2 [Element type declarations]

An element type declaration on N assumes one of the forms:

1. *empty form*: <!ELEMENT *elem-type* EMPTY >
2. *simple form*: <!ELEMENT *elem-type* (#PCDATA) >
3. *complex form*: <!ELEMENT *elem-type* *content-particle* >

where – *elem-type* $\in N$ specifies the element type being declared
– *content-particle* $\in CP$ describes the element's content model.

Let ETD be the set of all element type declarations on N. □

From the definition of element type declarations, an element type having a very complex content model can be simply described by a content particle which is formed by combinations of nested content particles and occurrence qualifiers '?', '+' or '*'.

Definition 3 [Attribute-list declarations]

An attribute-list declaration has the form:

<!ATTLIST *elem-type* $attr\text{-}name_1$ $attr\text{-}type_1$ $attr\text{-}default_1$
...
$attr\text{-}name_n$ $attr\text{-}type_n$ $attr\text{-}default_n$ >

where – $n \geq 1$,
– *elem-type* $\in N$ specifies the type of element that will be associated by the specified set of attributes,
– the $attr\text{-}name_i \in N$ are distinct attribute names,
– $attr\text{-}type_i \in$ {CDATA, ID, IDREF, IDREFS} $\cup \{(value_1 \mid \dots \mid value_m) \mid value_j \in \Sigma^*$ are distinct enumerated values$\}$,
– $attr\text{-}default_i \in$ {#REQUIRED, #IMPLIED} $\cup$ {#FIXED *fixed-value* | *fixed-value* $\in \Sigma^*$} $\cup\ \Sigma^*$.

Let ALD be the set of all attribute-list declarations. □

Definition 4 [Document type declarations]

A document type declaration is a sequence $d_1\ d_2\ \ldots\ d_m$ where $d_i \in (ETD \cup ALD)$. Let $DTD = (ETD \cup ALD)^*$, i.e., the set of all sequences on $(ETD \cup ALD)$, be the set of all document type declarations. □

3.2 XML DTD Translation

In the proposed approach, an XML DTD is modeled as a description comprising a set of clauses. Such clauses, precisely referred to as *DTD clauses*, are obtained directly by translation of each of the element type and attribute-list declarations contained in the DTD into a corresponding set of clauses and then combination of these sets. The numbers of clauses formulated for an element type declaration and for an attribute-list declaration depend solely on the complexity of the element type's content model and on the number of the declared attributes, their specified types and default values, respectively. The more complex is an element's structure, the greater a number of DTD clauses is obtained.

There are two classes of DTD clauses, namely, those that restrict element types' content model and those that constrain associated lists of attributes. The tag name of the head expression of each DTD clause starts simply with the name of the translated DTD, concatenated with the name of the element type being restricted. Such a head expression only describes certain particular restrictions on the element type's content model and merely specifies a general pattern of associated attribute list. Additional restrictions on the element's content model (e.g., descriptions of valid sequences of child elements) and on its associated attribute list (e.g., attribute type and default value constraints) are defined by appropriate specifications of XML expressions, constraints and references in a clause's body. An XML expression contained in a clause's body will be further restricted by the other DTD clauses the head of which can be matched with that XML expression. Constraints and references in a clause's body are used to impose conditions on attribute types and default values.

An XML element is valid with respect to a given DTD, if such an element can successfully match with the head of some clause translated from the DTD and all the restrictions specified in the body of such a clause are satisfied.

Let *XClause* denote the set of all clauses on Γ. Given next is the formal definition of the mapping τ_{CP} to be used for the definition of the element-type-declaration translator, τ_E (cf. Definition 6). Intuitively, τ_{CP} recursively translates a given pair (*cp*, *cp-specification*) into a corresponding set of clauses, where *cp* is a content particle and *cp-specification* an underscored separated element type in *N* having the form *dtd_elem-type_position*, where *dtd* specifies the translated DTD, *elem-type* the declared element type and *position* the location that *cp* occurs in the declaration of *elem-type*. In the sequel, assume that the DTD being translated is denoted by "*dtd*".

Definition 5 [τ_{CP}, the *content-particle translator*]

Let $cp \in CP$ and $cp\text{-}spec \in N$. The *content-particle translator* τ_{CP}: $(CP \times N) \rightarrow 2^{XClause}$ is defined by Table 1. □

Table 1. τ_{CP}, the *content-particle translator*.

Types of Content Particles	Content Particle $cp \in CP$	$\tau_{CP}(cp, cp\text{-}spec)$
1. Unqualified Content Particle		
1.1. Atomic Form	$cp = (elem\text{-}type)$, where $elem\text{-}type \in N$	$\tau_{CP}(cp, cp\text{-}spec) = \{C\}$, where *C*: <*cp-spec*> $E:subexp </*cp-spec*> ← <*dtd_elem-type*> $E:subexp </*dtd_elem-type*>.
1.2 Choice-List Form	$cp = (\ cp_1 \mid \ldots \mid cp_n\ )$, where $n \geq 1$, $cp_i \in CP$	$\tau_{CP}(cp, cp\text{-}spec) = \bigcup_{i=1}^{n} \tau_{CP}\ (cp_i, cp\text{-}spec\_i) \cup \{C_1, \ldots, C_n\}$, where, for each $i \in \{1, \ldots, n\}$, C_i: <*cp-spec*> $E:subexp </*cp-spec*> ← <*cp-spec_i*> $E:subexp </*cp-spec_i*>.

Types of Content Particles	**Content Particle $cp \in CP$**	**$\tau_{CP}(cp, cp\text{-}spec)$**
1.3. Sequence-List Form	$cp = (\ cp_1, \ldots, cp_n\ )$, where $n \geq 1$, $cp_i \in CP$	$\tau_{CP}(cp, cp\text{-}spec) = \bigcup_{i=1}^{n} \tau_{CP}\ (cp_i, cp\text{-}spec\_i) \cup \{C\}$, where C: *<cp-spec>* `$E:subexp_1` … `$E:subexp_n` *</cp-spec>* ← *<cp-spec_1>* `$E:subexp_1` *</cp-spec_1>*, … , *<cp-spec_i>* `$E:subexp_n` *</cp-spec_i>*.
2. Qualified Content Particles		
2.1. ?-Form	$cp = (\ cp_1\ ?\ )$, where $cp_1 \in CP$	$\tau_{CP}(cp, cp\text{-}spec) = \tau_{CP}(cp_1, cp\text{-}spec\_1) \cup \{C_1, C_2\}$, where C_1: *<cp-spec>* `$E:subexp` *</cp-spec>* ← *<cp-spec_1>* `$E:subexp` *<cp-spec_1>*. C_2: *<cp-spec>* *</cp-spec>* ← .
2.2. +-Form	$cp = (\ cp_1 +\ )$, where $cp_1 \in CP$	$\tau_{CP}(cp, cp\text{-}spec) = \tau_{CP}(cp_1, cp\text{-}spec\_1) \cup \{C_1, C_2\}$, where C_1: *<cp-spec>* `$E:subexp` *</cp-spec>* ← *<cp-spec_1>* `$E:subexp` *</cp-spec_1>*. C_2: *<cp-spec>* `$E:subexp_1` `$E:subexp_2` *</cp-spec>* ← *<cp-spec_1>* `$E:subexp_1` *</cp-spec_1>*, *<cp-spec>* `$E:subexp_2` *</cp-spec>*.
*2.3. *-Form*	$cp = (\ cp_1\ ^*\ )$, where $cp_1 \in CP$	$\tau_{CP}(cp, cp\text{-}spec) = \tau_{CP}(cp_1, cp\text{-}spec\_1) \cup \{C_1, C_2\}$, where C_1: *<cp-spec>* `$E:subexp_1` `$E:subexp_2` *</cp-spec>* ← *<cp-spec_1>* `$E:subexp_1` *<cp-spec_1>*, *<cp-spec>* `$E:subexp_2` *</cp-spec>*. C_2: *<cp-spec>* *</cp-spec>* ← .

Example 1 Given a content particle cp = (Organizer+ | Sponsor*) together with its specification myDTD_Conference_1_2 which describes that cp occurs in the declaration of Conference element type of myDTD, by means of the translator τ_{CP}, the pair (cp, myDTD_Conference_1_2) can be translated into a corresponding set of clauses:

1. τ_{CP}((Organizer+ | Sponsor*), myDTD_Conference_1_2)
   = τ_{CP}(Organizer+, myDTD_Conference_1_2_1)
   $\cup$ τ_{CP}(Sponsor*, myDTD_Conference_1_2_2)

$\cup \{C_1, C_2\}$

where

C_1:
```
<myDTD_Conference_1_2>
      $E:subexp
</myDTD_Conference_1_2>          ←    <myDTD_Conference_1_2_1>
                                            $E:subexp
                                      </myDTD_Conference_1_2_1>.
```

C_2:
```
<myDTD_Conference_1_2>
      $E:subexp
</myDTD_Conference_1_2>          ←    <myDTD_Conference_1_2_2>
                                            $E:subexp
                                      </myDTD_Conference_1_2_2>.
```

2. τ_{CP}(Organizer+, myDTD_Conference_1_2_1)

   = τ_{CP}(Organizer, myDTD_Conference_1_2_1_1)

   $\cup \{C_3, C_4\}$

where

C_3:
```
<myDTD_Conference_1_2_1>
      $E:subexp
</myDTD_Conference_1_2_1>        ←   <myDTD_Conference_1_2_1_1>
                                           $E:subexp
                                     <myDTD_Conference_1_2_1_1>.
```

C_4:
```
<myDTD_Conference_1_2_1>
      $E:subexp_1
      $E:subexp_2
</myDTD_Conference_1_2_1>        ←   <myDTD_Conference_1_2_1_1>
                                           $E:subexp_1
                                     <myDTD_Conference_1_2_1_1>,

                                     <myDTD_Conference_1_2_1>
                                           $E:subexp_2
                                     </myDTD_Conference_1_2_1>.
```

3. τ_{CP}(Organizer, myDTD_Conference_1_2_1_1)

   = $\{C_5\}$

where

C_5:
```
<myDTD_Conference_1_2_1_1>
      $E:subexp
</myDTD_Conference_1_2_1_1>      ←    <myDTD_Organizer>
                                            $E:subexp
                                      </myDTD_Organizer>.
```

4. τ_{CP}(Sponsor*, myDTD_Conference_1_2_2)

   = τ_{CP}(Sponsor, myDTD_Conference_1_2_2_1)

   $\cup \{C_6, C_7\}$

where

C_6:
```
<myDTD_Conference_1_2_2>
      $E:subexp_1
      $E:subexp_2
</myDTD_Conference_1_2_2>        ← <myDTD_Conference_1_2_2_1>
                                           $E:subexp_1
                                   </myDTD_Conference_1_2_2_1>,

                                   <myDTD_Conference_1_2_2>
                                           $E:subexp_2
                                   </myDTD_Conference_1_2_2>.
```

C_7:
```
<myDTD_Conference_1_2_2>
</myDTD_Conference_1_2_2>        ←        .
```

5. τ_{CP}(Sponsor, myDTD_Conference_1_2_1_1)
   = $\{C_8\}$

   where

   C_8:
```
<myDTD_Conference_1_2_2_1>
    $E:subexp
</myDTD_Conference_1_2_2_1>    ←    <myDTD_Sponsor>
                                         $E:subexp
                                     </myDTD_Sponsor>.
```

Then, let $P_1 = \tau_{CP}(cp,$ myDTD_Conference_1_2$) = \{C_1, \ldots, C_8\}$. □

Based on the definition of the content particle translator τ_{CP}, the definition of *element-type-declaration translator* τ_E is now given.

Definition 6 [τ_E, the *element-type-declaration translator*]

Let $d \in ETD$ be an element type declaration. The *element-type-declaration translator* τ_E: $ETD \rightarrow 2^{XClause}$ is defined by Table 2. □

Table 2. τ_E, the *element-type-declaration translator.*

Types of Element Type Declarations	Element Type Declaration $d \in ETD$	$\tau_E(d)$

1. Empty Form	<!ELEMENT *elem-type* EMPTY>,
	where *elem-type* $\in N$
		$\tau_E(d) = \{C\}$, where
		C: <*dtd_elem-type*>
		    <*elem-type* \$P:attrList/>
		  </*dtd_elem-type*>
		      ← <*dtd_elem-type*_attrList_1 \$P:attrList/>.

2. Simple Form	<!ELEMENT *elem-type* (#PCDATA)>,
	where *elem-type* $\in N$
		$\tau_E(d) = \{C\}$, where
		C: <*dtd_elem-type*>
		    <*elem-type* \$P:attrList>
		      \$S:pcdata
		    </*elem-type*>
		  </*dtd_elem-type*>
		      ← <*dtd_elem-type*_attrList_1 \$P:attrList/>.

3. Complex Form	<!ELEMENT *elem-type cp*>,
	where *cp* $\in CP$
		$\tau_E(d) = \tau_{CP}(cp, dtd\_elem\text{-}type\_1) \cup \{C\}$, where
		C: <*dtd_elem-type*>
		    <*elem-type* \$P:attrList>
		      \$E:subexp
		    </*elem-type*>
		  </*dtd_elem-type*>
		      ← <*dtd_elem-type*_attrList_1 \$P:attrList/>,
		        <*dtd_elem-type_1*>
		          \$E:subexp
		        </*dtd_elem-type_1*>.

Example 2 Denote the DTD of Fig. 1 by myDTD. This example demonstrates a translation of Conference element type declaration d_1 into a corresponding set of clauses.

```
d1:     <!ELEMENT Conference (Name, (Organizer+ | Sponsor*))>
d2:     <!ATTLIST Conference url ID #REQUIRED
                             type (International | Local) #REQUIRED
                             chair IDREF #IMPLIED>
d3:     <!ELEMENT Name (#PCDATA)>
d4:     <!ELEMENT Organizer (#PCDATA)>
d5:     <!ELEMENT Sponsor (#PCDATA)>
d6:     <!ELEMENT Person (#PCDATA)>
d7:     <!ATTLIST Person ssn ID #REQUIRED>
```

Fig. 1. An XML DTD Example.

1. $\tau_E(d_1)$ = τ_{CP}((Name, (Organizer+ | Sponsor*)), myDTD_Conference_1)
   $\cup \{C_9\}$

   where

   C_9:
```
<myDTD_Conference>
        <Conference $P:attrList>
            $E:subexp
        </Conference>
</myDTD_Conference>            ←        <myDTD_Conference_attrList_1  $P:attrList />,

                                        <myDTD_Conference_1>
                                            $E:subexp
                                        </myDTD_Conference_1>.
```

2. τ_{CP}((Name, (Organizer+ | Sponsor*)), myDTD_Conference_1)
   = τ_{CP}(Name, myDTD_Conference_1_1)
   $\cup$ τ_{CP}((Organizer+ | Sponsor*), myDTD_Conference_1_2)
   $\cup \{C_{10}\}$

   where

   C_{10}:
```
<myDTD_Conference_1>
        $E:subexp_1
        $E:subexp_2
</myDTD_Conference_1>          ←        <myDTD_Conference_1_1>
                                            $E:subexp_1
                                        </myDTD_Conference_1_1>,

                                        <myDTD_Conference_1_2>
                                            $E:subexp_2
                                        </myDTD_Conference_1_2>.
```

3. τ_{CP}(Name, myDTD_Conference_1_1)
   = $\{C_{11}\}$

   where

   C_{11}:
```
<myDTD_Conference_1_1>
        $E:subexp
</myDTD_Conference_1_1>        ←        <myDTD_Name>
                                            $E:subexp
                                        </myDTD_Name>.
```

4. τ_{CP}((Organizer+ | Sponsor*), myDTD_Conference_1_2)
   = P_1
   = $\{C_1, \ldots, C_8\}$

These four steps yield $P_2 = \tau_E(d_1) = \{C_9, C_{10}, C_{11}\} \cup P_1$.

Clause C_9 imposes some restrictions on the Conference element. Its head specifies that every conforming Conference element must contain a list of associated attribute-value pairs as well as a sequence of subelements, represented by the P-variable $P:attrList and the E-variable $E:subexp, respectively. Its first and second body elements indicate that the validity of the attribute list and the subelement sequence will be determined by clauses with the heads: myDTD_Conference_attrList_1 and myDTD_Conference_1 elements, i.e., by those clauses obtained by translation of the declaration of Conference's attributes (cf. Example 3) and by clause C_{10}, respectively.

Clause C_{10} divides the subelement sequence of a Conference element into arbitrary two subelement sequences and then specifies that restrictions on the first sequence are defined by means of the myDTD_Conference_1_1 expression (i.e., by clause C_{11} obtained from τ_{CP}(Name,myDTD_Conference_1_1))) while restrictions on the second sequence by myDTD_Conference_1_2 expression (i.e., by clauses C_1 and C_2 in description P_1 obtained from τ_{CP}((Organizer+ | Sponsor*), myDTD_Conference_1_2)). Clause C_{11} simply constrains that such a first sequence must contain exactly one element conforming to the grammar defined for the Name element type, i.e., it must satisfy the clauses the head of which are myDTD_Name expressions.

Clauses C_1 and C_2 demand that the second sequence must conform to the restriction defined by clauses C_3–C_4 or by clauses C_6–C_7, respectively. Clauses C_3 and C_4 together specify that such a sequence may consist of *one or more* sub-sequences each sub-sequence of which is restricted by clause C_5, i.e., it must contain a valid Organizer element. Alternatively, clauses C_6 and C_7 indicate that such a second sequence of a Conference element may comprise *zero or more* sub-sequences each of which is constrained by clause C_8, i.e., each sub-sequence must contain a valid Sponsor element. ☐

In the sequel, let \$S:id be an S-variable in $SVAR$.

Definition 7 [Mapping *EID*]

A mapping $ElementID : DTD \rightarrow 2^{\mathcal{A}_X}$ is

$EID(dtd) = Y \subset \mathcal{A}_X$ such that

the XML expressions

```
<$I:anExpression>
    <elem-type_i  attr-name_i=$S:id  $P:attrList>
        $S:content
    </elem-type_i>
</$I:anExpression>
```

and

```
<$I:anExpression>
    <elem-type_i  attr-name_i=$S:id  $P:attrList>
        $E:subexp
    </elem-type_i>
</$I:anExpression>
```

where \$I:anExpression $\in IVAR$, \$S:content $\in SVAR$, \$P:attrList $\in PVAR$, \$E:subexp $\in EVAR$, will be contained in Y iff

```
<!ATTLIST  elem-type name_1  type_1  default_1
                     ...
                     name_i  ID  default_i
                     ...
                     name_n  type_n  default_n>
```

is an attribute-list declaration in *dtd*. ☐

Given $dtd \in DTD$, $EID(dtd)$ returns a set of non-ground XML expressions in $\mathcal{A}_X$ which represent classes of XML elements having associated attributes of type ID, defined by the given *dtd*.

Definition 8 [Mapping *GetID*]

Based on the mapping EID, let $GetID$: $(2^{\mathcal{G}_X} \times 2^{DTD}) \rightarrow 2^{\mathcal{G}_X}$ be

$$\text{Given } X \subset \mathcal{G}_X,\ dtd \in DTD, \tag{6}$$
$$GetID(X,dtd) = \{\text{<id value=\$S:id/>}\,\theta \mid a \in EID(dtd),\ \theta \in \mathcal{S}_X,\ a\theta \in X\}$$

☐

Intuitively, given a subset X of $\mathcal{G}_X$, and *dtd* in DTD, $GetID(X, dtd)$ is a set containing XML elements, each of the form <id value=*elem-id*/>, where $elem\text{-}id \in \Sigma^*$ is an identity of an XML element in X.

Definition 9 [*id-, idref-, idrefs-reference functions*]

Given $dtd \in DTD$, let id_{dtd}: $2^{\mathcal{G}_X} \rightarrow 2^{\mathcal{G}_X}$, $idref_{dtd} : 2^{\mathcal{G}_X} \rightarrow 2^{\mathcal{G}_X}$ and $idrefs_{dtd} : 2^{\mathcal{G}_X} \rightarrow 2^{\mathcal{G}}$ be reference functions in F defined in terms of the mapping *GetID* as follows:
For each $X \subset \mathcal{G}_X$,

$$id_{dtd}(X) = \mathcal{G}_X - GetID(X, dtd) \tag{7}$$

$$idref_{dtd}(X) = GetID(X, dtd) \tag{8}$$

$$idrefs_{dtd}(X) = 2^{GetID(X,\ dtd)} \tag{9}$$

id_{dtd}, $idref_{dtd}$ and $idrefs_{dtd}$ will be referred to as *id-*, *idref-* and *idrefs-reference* functions. ☐

Note that references $\langle a,\ id_{dtd},\ R\rangle$, $\langle a,\ idref_{dtd},\ R\rangle$ and $\langle S,\ idrefs_{dtd},\ R\rangle$ will be called *id*, *idref* and *idrefs references*, respectively, iff

- a = <id value=*elem-id*/> $\in \mathcal{A}_X$ of the form <id value=*elem-id*/>, where $elem\text{-}id \in (\Sigma^* \cup SVAR)$,

- $S \in V$ or $S = \{a_1, \ldots, a_n\} \subset \mathcal{A}_X$, where a_i has the form <id value=*elem-id*$_i$/> and *elem-id*$_i \in (\Sigma^* \cup SVAR)$,
- $dtd \in DTD$,
- R is a description on Γ specifying an XML document upon which a given XML element will be validated against.

Such concepts of id and idref(s) references defined here are useful for specification of *uniqueness* and *referential* constraints defined by attributes of types ID and IDREF(S), respectively.

The definition of *true references* in Section 2.2 shows that the conditions specified in Table 3 must hold for a particular id and idref(s) references to be true references.

Table 3. Satisfiability conditions for true id and idref(s) references

Reference	Satisfiability Conditions
1. id reference $\langle g, id_{dtd}, R\rangle$, where g = <id value=*elem-id*/> $\in \mathcal{G}_X$	The value specified by *elem-id* does not occur as an ID of any XML elements in $\mathcal{M}(R)$
2. idref reference $\langle g, id_{dtd}, R\rangle$, where g = <id value=*elem-id*/> $\in \mathcal{G}_X$	There exists an XML element in $\mathcal{M}(R)$ the ID of which is *elem-id*.
3. idrefs reference $\langle X, id_{dtd}, R\rangle$, where $X = \{g_1, \ldots, g_n\}$ and g_i = <id value=*elem-id*$_i$/> $\in \mathcal{G}_X$	For each $i \in \{1, \ldots, n\}$, there exists an XML element in $\mathcal{M}(R)$ which is uniquely identified by *elem-id*$_i$.

In the sequel, let R be a description on Γ, which specifies an XML document against which a given XML element will be validated.

Definition 10 [Equal, IsMemberOf and IdrefsSplitUp constraints]

Let Equal, IsMemberOf and IdrefsSplitUp be constraint predicates in K. The constraints Equal, IsMemberOf and IdrefsSplitUp on Γ are:

1. Equal(a_1, a_2), where $a_1, a_2 \in \mathcal{A}$,
2. IsMemberOf(a, X), where $a \in \mathcal{A}_X$, $X \in 2^{(\mathcal{A} \cup V)}$,
3. IdrefsSplitUp(<idrefs value=*string*/>, X), where $string \in SVAR \cup \Sigma^*$ and $X \in 2^{(\mathcal{A} \cup V)}$.

Such constraints are true constraints in *Tcon* iff they assume the forms:

1. Equal(g, g), where $g \in \mathcal{G}$,
2. IsMemberOf(g, X), where $g \in \mathcal{G}_X$, $X \in 2^{\mathcal{G}_X}$, and $g \in X$,
3. IdrefsSplitUp(<idrefs value="*string*"/>, X), where $string \in SVAR$ and X = {<id value="*string*$_1$"/>, ..., <id value="*string*$_n$"/>} $\in 2^{\mathcal{G}_X}$ such that *string* is the white-spaced separated sequence of *string*$_1$, *string*$_2$, ..., *string*$_n$.

□

Intuitively,

1. For $a_1, a_2 \in \mathcal{A}$, a constraint Equal(a_1, a_2) is used to ensure that the objects a_1 and a_2 are identical.
2. For $a \in \mathcal{A}_X$ and $X \in 2^{(\mathcal{A} \cup V)}$, a constraint IsMemberOf(a, X) ensure that the XML element represented by a is a member of the set X.
3. For $string \in SVAR \cup \Sigma^*$ and $X \in 2^{(\mathcal{A} \cup V)}$, a constraint IdrefsSplitUp(<idrefs value=*string*/>, X) ensure that X is specialized to a set {<id value="*string*$_1$"/>, ..., <id value="*string*$_n$"/>} $\in 2^{\mathcal{G}_X}$ such that *string* is the white-spaced separated sequence of *string*$_1$, *string*$_2$, ..., *string*$_n$.

Definition 11 [τ_A, the *attribute-list-declaration translator*]

Let τ_A: $ALD \rightarrow 2^{XClause}$ denote *attribute-list-declaration translator*. For an attribute-list declaration $d \in ALD$ defined in the DTD *dtd* and having the form

```
<!ATTLIST  elem-type  name_1  type_1  default_1
                      ...
                      name_n  type_n  default_n>, where n ≥ 1,
```

$\tau_A(d)$ is a set comprising m+1 clauses, where $n \leq m \leq 2n$.

An algorithm describing the formulation of such m+1 clauses follows:

Step 1: [Formulation of the first *m* clauses]

```
1:  For ($i$=1; $i \leq n$; $i$=$i$+1)
2:      Let $j = i + 1$.
3:      If ($default_i$ is #REQUIRED)
        Then
4:          Let $m = 1$.
5:          Formulate clause $C_{i1}$, where
                $C_{i1}$:  <dtd_elem-type_attrList_i  name_i=$S:value  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />.
6:      Else-If ($default_i$ is #IMPLIED)
        Then
7:          Let $m = 2$.
8:          Formulate clauses $C_{i1}$ and $C_{i2}$, where
                $C_{i1}$:  <dtd_elem-type_attrList_i  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />.
                $C_{i2}$:  <dtd_elem-type_attrList_i  name_i=$S:value  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />.
9:      Else-If ($default_i$ is #FIXED fixed-value)
        Then
10:         Let $m = 1$.
11:         Formulate clause $C_{i1}$, where
                $C_{i1}$:  <dtd_elem-type_attrList_i  name_i=$S:value  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />,
                             Equal(<Value>$S:value</Value>,
                                   <Value>fixed_value</Value>).
12:     Else-If ($default_i$ is fixed-value)
        Then
13:         Let $m = 2$.
14:         Formulate clauses $C_{i1}$ and $C_{i2}$, where
                $C_{i1}$:  <dtd_elem-type_attrList_i  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />.
                $C_{i2}$:  <dtd_elem-type_attrList_i  name_i=$S:value  $P:attrList />
                        ←    <dtd_elem-type_attrList_j  $P:attrList />.
        End-If.
15:     If ($type_i$ is ID)
        Then
16:         For ($k$=1; $k \leq m$; $k$=$k$+1)
17:             Add the reference ⟨<id value= $S:value />, $f_{id,dtd}$, $R$⟩ to the body of clause $C_{ik}$
            End-For.
18:     Else-If ($type_i$ is IDREF)
        Then
19:         For ($k$=1; $k \leq m$; $k$=$k$+1)
20:             Add the reference ⟨<id value= $S:value />, $f_{idref,dtd}$, $R$⟩ to the body of clause $C_{ik}$
            End-For.
21:     Else-If ($type_i$ is IDREFS)
        Then
22:         For ($k$=1; $k \leq m$; $k$=$k$+1)
23:             Add the constraint IdrefsSplitUp(<idrefs value= $S:value />, $V:SetOfIds) and
                the reference ⟨$V:SetOfIds, $f_{idrefs,dtd}$, $R$⟩ to the body of clause $C_{ik}$
            End-For.
24:     Else-If ($type_i$ is an enumeration ($value_1$ | … | $value_k$))
        Then
25:         For ($k$=1; $k \leq m$; $k$=$k$+1)
26:             Add the constraint
                    IsMemberOf(<Value>$S:value</Value>,
                        {<Value>$value_1$</Value>, …,<Value>$value_k$</Value>})
                to the body of clause $C_{ik}$
            End-For.
        End-If.
    End-For.
```

```
Step 2: [Formulation of the $(m+1)^{th}$ clauses]
27:     Let $j = n + 1$.
28:     Formulate clause $C_{n+1}$, where
            $C_{n+1}$: <dtd_elem-type_attrList_j />   ←   .
□
```

In order to determine the validity of a list of attribute-value pairs associated with an element of *elem-type*, these $m+1$ clauses, $n \le m \le 2n$, work in $n+1$ steps:

- In the i^{th} step, $1 \le i \le n$, the validity of the specification of the attribute $name_i$ is verified by means of clause C_{ij}. If such specification is valid, the pair of that attribute $name_i$ and its value is removed from the list and the next step, i.e., the $(i+1)^{th}$ step, is taken. Otherwise, the verification fails.
- In the last step, i.e., the $(n+1)^{th}$ step, clause C_{n+1} verifies that no undeclared attribute can appear in the list, i.e., the list of attribute-value pairs must now be empty.

Note that when there is no attribute-list declaration provided for *elem-type*, the following clause must be formulated instead:

<*dtd_elem-type*_attrList_1 />   ←   .

Such clause merely restricts that elements of *elem-type* cannot have an associated list of attribute-value pairs.

Example 3 As an example of the translation of an attribute-list declaration, let P_3 be a description obtained by translation of d_2, the declaration of attributes associated with Conference element (Fig. 1). In other words, $P_3 = \tau_A(d_2)$ comprises the following five clauses, denoted by $C_{12} - C_{16}$:

```
C12: <myDTD_Conference_attrList_1 url=$S:value $P:attrList />
        ←     <myDTD_Conference_attrList_2 $P:attrList />,
              (<id value= $S:value />, f_{id, myDTD}, R).

C13: <myDTD_Conference_attrList_2 type=$S:value $P:attrList />
        ←     <myDTD_Conference_attrList_3 $P:attrList />,
              IsMemberOf(<Value>$S:value</Value>,
                      {<Value>International</Value>, <Value>Local</Value>}).

C14: <myDTD_Conference_attrList_3 chair=$S:value $P:attrList />
        ←     <myDTD_Conference_attrList_4 $P:attrList />,
              (<id value= $S:value />, f_{idref, myDTD}, R).

C15: <myDTD_Conference_attrList_3 $P:attrList />
        ←     <myDTD_Conference_attrList_4 $P:attrList />.

C16: <myDTD_Conference_attrList_4 />    ←    .
```

Clause C_{12} specifies constraints imposed on the list of attribute-value pairs associated with a Conference element. It ensures that the list contains a specification of url attribute, while the other attributes, represented by $P:attrList, will be additionally constrained by a clause the head of which is myDTD_Conference_attrList_2 expression, i.e., clause C_{13}. Moreover, the id reference contained in the body of C_{12} specifies that the value of url attribute, represented by $S:url, must be unique with respect to description *R*, i.e., $S:url does not occur as an ID for any element defined in description *R*.

Clause C_{13} imposes that a Conference element must contain also a type attribute the value of which must be either International or Local. Clauses C_{14} and C_{15} then enforce that the element may optionally contain a chair attribute. The idref reference contained in the body of C_{14} specifies that the value of chair attribute, represented by $S:value, is a reference to another element defined in description *R* and having the same value as its ID. Clause C_{16} specifies that the Conference element cannot contain attributes other than the url, type and chair attributes. □

Definition 12 [τ_{DTD}, the *document-type-declaration translator*]

The *element-type-and-attribute-list-declaration translator* $\tau_{E\&A}$: $(ETD \cup ALD) \rightarrow 2^{XClause}$ is:

$\tau_{E\&A}(d) = \tau_E(d)$, if $d \in ETD$,
$\tau_{E\&A}(d) = \tau_A(d)$, if $d \in ALD$.

Let $dtd = (d_1 \ldots d_n) \in DTD$. The *document-type-declaration translator* τ_{DTD}: $DTD \rightarrow 2^{XClause}$ is:

$$\tau_{DTD}(dtd) = \bigcup_{i=1}^{n} \tau_{E\&A}(d_i)$$

$\cup$ {<*dtd_elem-type*_attrList_1/> ← . | <!ELEMENT *elem-type content-model*> $\in$ *dtd*,

<!ATTLIST *elem-type name*$_1$ *type*$_1$ *default*$_1$ ... *name*$_n$ *type*$_n$ *default*$_n$> $\notin$ *dtd* }.

□

Example 4 This example demonstrates a translation of myDTD (Fig. 1). into a corresponding set of clauses. Let Q be a description obtained from translating myDTD. Then,

$Q = \tau_{DTD}(\text{myDTD}) = P_1 \cup P_2 \cup P_3 \cup P_4,$

where P_4 comprises the six clauses $C_{17} - C_{25}$:

```
C17: <myDTD_Name>
         <Name  $P:attrList>
             $S:pcdata
         </Name>
     </myDTD_Name>                       ←        <myDTD_Name_attrList_1  $P:attrList />.
C18: <myDTD_Name_attrList_1 />           ←        .
C19: <myDTD_Organizer>
         <Organizer  $P:attrList>
             $S:pcdata
         </Organizer>
     </myDTD_Organizer>                  ←        <myDTD_Organizer_attrList_1  $P:attrList />.
C20: <myDTD_Organizer_attrList_1 />      ←        .
C21: <myDTD_Sponsor>
         <Sponsor  $P:attrList>
             $S:pcdata
         </Sponsor>
     </myDTD_Sponsor>                    ←        <myDTD_Sponsor_attrList_1  $P:attrList />.
C22: <myDTD_Sponsor_attrList_1 />        ←        .
C23: <myDTD_Person  $P:attrList>
         <Person  $P:attrList>
             $S:pcdata
         </Person>
     </myDTD_Person>                     ←        <myDTD_Person_attrList_1  $P:attrList />.
C24: <myDTD_Person_attrList_1  ssn=$S:value  $P:attrList/>
                                         ←        <myDTD_Person_attrList_2  $P:attrList />,
                                                  ⟨<id  value=$S:value />, f_id.myDTD, R⟩.
C25: <myDTD_Person_attrList_2 />         ←        .
```

□

3.3 DTD Translation Optimization

Since this is an attempt to outline a general translation scheme for all possible XML DTDs, it may be pointed out that the number of DTD clauses obtained from modeling some particular DTD is rather large and could lead to an inefficient approach. This limitation can be alleviated by application of the *optimization algorithm* (cf. Appendix) which rewrites and removes redundant DTD clauses. For example, clauses C_{24} and C_{25} of Example 4 can be replaced by the clause:

<myDTD_Person_attrList_1 ssn=$S:value/> ← ⟨<id value=$S:value />, $f_{id.\text{myDTD}}$, R⟩.

Appendix also gives a description Q' obtained by application of the developed optimization algorithm to description Q of Example 4.

4 XML Element Validity Checking

Given an XML DTD represented by a description P, in order to determine the validity of an XML element, say x, with respect to such DTD, a single clause D is formulated:

$$D\text{: } a \quad \leftarrow \quad \textit{<dtd\_elem-type>}\ x\ \textit{</dtd\_elem-type>}. \tag{10}$$

The head of D, represented by a, is an XML expression in $\mathcal{A}_X$ which will be derived if the given element x is valid. The body of D contains a single XML expression with a tag name of the form *dtd_elem-type*, where *dtd* is the name of the DTD to be checked and *elem-type* the type of the validated element. Such a body expression contains the validated element x as its only child element. If x is valid, the element represented by a will be derived from or contained in the meaning of the description $(P \cup \{D\})$. More precisely, to say that x is valid, such a description $(P \cup \{D\})$ must be able to be *transformed equivalently* and *successively* into the description $(P \cup \{D'\})$, where D' is an *ground unit clause* of the form

$$V'\text{:} \quad a\theta \quad \leftarrow \quad . \tag{11}$$

Example 5 Referring to description Q of Example 4 which represents myDTD, in order to determine whether the Conference element:

```
<Conference url="http://www.cs.ait.ac.th/smartnet99/" type="International" chair="12345">
    <Name>SmartNet'99</Name>
    <Organizer>Asian Institute of Technology</Organizer>
    <Organizer>International Federation Information Processing</Organizer>
    <Organizer>Telecommunication of Thailand</Organizer>
</Conference>
```

conforms to myDTD or not, the following clause is formulated:

```
D:  <Valid_XML url="http://www.cs.ait.ac.th/smartnet99/" />
        ←   <myDTD_Conference>
                <Conference url="http://www.cs.ait.ac.th/smartnet99/"
                  type="International" chair="12345">
                    <Name>SmartNet'99</Name>
                    <Organizer>Asian Institute of Technology</Organizer>
                    <Organizer>International Federation Information Processing</Organizer>
                    <Organizer>Telecommunication of Thailand</Organizer>
                </Conference>
            </myDTD_Conference>
```

Suppose that the referred description R, which represents an XML document to be validated against, comprises the two clauses E_1 and E_2:

```
E1: <Conference url="http://www.cs.ait.ac.th/ijwdl98/" type="International">
        <Name>International Joint Workshop on Digital Libraries</Name>
        <Organizer>Asian Institute of Technology</Organizer>
    </Conference>           ←       .

E2: <Person ssn="12345">Vilas Wuwongse</Person>          ←       .
```

Since the description $(Q \cup \{D\})$ can be successively transformed into the description $(Q \cup \{D'\})$, where

```
D': <Valid_XML url="http://www.cs.ait.ac.th/smartnet99/" />        ←       .,
```

the given Conference element is valid with respect to myDTD. Validating other Conference elements is similar. □

5 Conclusions

An approach to the determination of the grammatical correctness of a given XML element/document with respect to a particular DTD has been developed, by incorporation of the expressiveness and efficient computational mechanism facilitated by Declarative Description theory and Equivalent Transformation (ET) paradigm, respectively. It represents an XML DTD as a corresponding set of DTD clauses, which describe valid elements' content models as well as restrictions on associated lists of attributes, e.g., uniqueness, referential and type constraints. Thus, the developed approach is *complete* with respect to XML DTD modeling and document validating.

Research on an extension of *XML-ETC Engine*, a Web-based XML processor developed under *Equivalent Transformation Compiler* (*ETC*) environment, by integration of supports for DTD modeling and validation is continuing. Moreover, formalisms for *DTD transformation and combination*, e.g., union, concatenation, intersection and complement, are envisaged, in order to provide a complete support for DTD and document processing.

Acknowledgement

This work was supported in part by Thailand Research Fund.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, CA (2000)
2. Akama, K.: Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advances in Software Science and Technology*, Vol. 5 (1993) 45-63
3. Akama, K.: Declarative Description with References and Equivalent Transformation of Negative References. *Technical Report*, Department of Information Engineering, Hokkaido University, Japan (1998)
4. Akama, K., Shimitsu, T., Miyamoto, E.: Solving Problems by Equivalent Transformation of Declarative Programs. *Journal of the Japanese Society of Artificial Intelligence*, Vol. 13 No. 6 (1998) 944-952 (in Japanese)
5. Akama, K., Anutariya, C., Wuwongse, V. and Nantajeewarawat, E.: A Foundation for XML Document Databases: Query Formulation and Evaluation. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)
6. Anutariya, C., Wuwongse, V., Nantajeewarawat, E. and Akama, K.: Towards a Foundation for XML Document Databases. *Proceedings of 1st International Conference on Electronic Commerce and Web Technologies (EC-Web 2000)*, London, UK. Lecture Notes in Computer Science, Springer Verlag (2000) (to appear)
7. Apparao, V., Et Al.: Document Object Model (DOM) Level 1 Specification Version 1.0, October 1998. W3C Recommendation (1998) Available at http://www.w3.org/TR/REC-DOM-Level-1/
8. Beech, C., Malhotra, A., Rys, M.: A Formal Data Model and Algebra for XML. *W3C XML Query Working Group Note*, September 1999 (1999)
9. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, February 1998. *W3C Recommendation*. (1998) Available at http://www.w3.org/TR/REC-xml
10. Buneman, P., Deutsch, A., Tan, W.C.: A Deterministic Model for Semi-Structured Data. *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats* (1998) Available at http://db.cis.upenn.edu/DL/icdt.ps.gz
11. Buneman, P., Fan, W., Weinstein, S.: Interaction between Path and Type Constraints. *Technical Report*, Department of Computer and Information Science, University of Pennsylvania (1998) Available at ftp://ftp.cis.upenn.edu/pub/papers/db-research/tr9816.ps.gz
12. Fernández, M., Siméon, J., Suciu, C. and Wadler, P.: A Data Model and Algebra for XML Query. *Draft Manuscript* (1999) Available at http://www.cs.bell-labs.com/~wadler/topics/xml.html#algebra
13. Goldman, R., McHugh, J., Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania (1999) Available at http://www-db.stanford.edu/lore/pubs/xml.pdf
14. Makoto, M.: Forest-regular Languages and Tree-regular Languages. *Technical Report*, Fuji Xerox Information Systems, (1995) Available at http://wwtataw.geocities.com/ResearchTriangle/Lab/6259/prelim1.pdf
15. Makoto, M.: Transformation of Documents and Schemas by Patterns and Contextual Conditions. *Principles of Document Processing, Proceedings of the Third International Workshop*, Vol. 1293 (1997)
16. Makoto, M.: DTD Transformation by Patterns and Contextual Conditions. *SGML/XML '97 Conference Proceedings* (1997)
17. Wuwongse, V., Akama, K., Anutariya, C. and Nantajeewarawat, E.: A Foundation for XML Document Databases: Data Model. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)

Appendix

An optimization algorithm for the developed DTD translation scheme is sketched.

Let aDTD be an XML DTD and $P = \{C_1, \ldots, C_m\}$ a description obtained by translation of aDTD into a corresponding set of DTD clauses, i.e., $P = \tau_{DTD}$(aDTD). An algorithm which can reduce the complexity of such a description P by removal and rewriting of some redundant DTD clauses contained in P follows:

```
1:   Let $Q = \{C_1\theta_1, \ldots, C_m\theta_m\}$, where $\theta_1, \ldots, \theta_m$ are specializations in $\mathcal{S}$ which rename variables in $C_1, \ldots, C_m$,
          respectively, such that $C_1\theta_1, \ldots, C_m\theta_m$ do not have any variable name in common.
2:   Repeat
3:        Find a clause $C = (H \leftarrow B_1, B_2, \ldots, B_n) \in Q$ that satisfies the following two conditions:
            – head(C)'s tag name has the form
                  aDTD_elem-type_level
              or
                  aDTD_elem-type_attrList_level
              where elem-type is an element type declared in aDTD and level is a sequence of number
              separated by underscores, e.g., 1_2_1.
            – There is no clause $D \in Q$ such that head(D)'s tag name is the same as head(C)'s tag name.
4:        If (such a clause C (in Step 3) is found)
          Then
5:             Let $Q = Q - \{C\}$, i.e., remove clause C from description Q.
6:             For each (clause $D = (H' \leftarrow B'_1, B'_2, \ldots, B'_{i-1}, B'_i, B'_{i+1}, \ldots, B'_u) \in Q$, where $u \geq 0$)
7:                  If (there exists $\theta \in \mathcal{S}$ such that $B'_i\theta = H$)
                    Then
8:                       Let $D' = (H'\theta \leftarrow B'_1\theta, B'_2\theta, \ldots, B'_{i-1}\theta, B_1, B_2, \ldots, B_m, B'_{i+1}\theta, \ldots, B'_u\theta)$.
9:                       Let $Q = Q - \{D\} \cup \{D'\}$, i.e., replace clause D in Q by D'.
                    End-If.
               End-For-each.
          End-If.
10:  Until (such a clause C is not found in Q).
```

Based on the above algorithm, let Q' be a description obtained by optimization of description Q of Example 4 and containing the following 13 DTD clauses, denoted by $C'_1 - C'_{13}$:

```
C'_1: <myDTD_Conference>
          <Conference url=$S:value  type=$S:value1  $P:attrList>
              $E:subexp_1
              $E:subexp_2
          </Conference>
      </myDTD_Conference>                 ←   <myDTD_Conference_attrList_3  $P:attrList />,
                                              (<id  value= $S:value />, f_{id, myDTD}, R),
                                              IsMemberOf(<Value>$S:value1</Value>,
                                                         {<Value>International</Value>,
                                                          <Value>Local</Value>})
                                              <myDTD_Name>
                                                         $E:subexp
                                              </myDTD_Name>,
                                              <myDTD_Conference_1_2>
                                                         $E:subexp_2
                                              </myDTD_Conference_1_2>.

C'_2: <myDTD_Conference_1_2>
          $E:subexp
      </myDTD_Conference_1_2>             ←   <myDTD_Conference_1_2_1>
                                                  $E:subexp
                                              </myDTD_Conference_1_2_1>.
```

```
C'3:  <myDTD_Conference_1_2>
          $E:subexp
      </myDTD_Conference_1_2>          ←   <myDTD_Conference_1_2_2>
                                                $E:subexp
                                            </myDTD_Conference_1_2_2>.

C'4:  <myDTD_Conference_1_2_1>
          $E:subexp
      </myDTD_Conference_1_2_1>        ←   <myDTD_Organizer>
                                                $E:subexp
                                            </myDTD_Organizer>.

C'5:  <myDTD_Conference_1_2_1>
          $E:subexp_1
          $E:subexp_2
      </myDTD_Conference_1_2_1>        ←   <myDTD_Organizer>
                                                $E:subexp_1
                                            </myDTD_Organizer>,

                                            <myDTD_Conference_1_2_1>
                                                $E:subexp_2
                                            </myDTD_Conference_1_2_1>.

C'6:  <myDTD_Conference_1_2_2>
          $E:subexp_1
          $E:subexp_2
      </myDTD_Conference_1_2_2>        ←   <myDTD_Sponsor>
                                                $E:subexp_1
                                            </myDTD_Sponsor>,
                                            <myDTD_Conference_1_2_2>
                                                $E:subexp_2
                                            </myDTD_Conference_1_2_2>.

C'7:  <myDTD_Conference_1_2_2>
      </myDTD_Conference_1_2_2>        ←        .

C'8:  <myDTD_Conference_attrList_3  chair=$S:value/>
                                       ←        ⟨<id  value= $S:value />, f_idref,myDTD, R⟩.

C'9:  <myDTD_Conference_attrList_3/>   ←        .

C'10: <myDTD_Name>
          <Name>
              $S:pcdata
          </Name>
      </myDTD_Name>                    ←        .

C'11: <myDTD_Organizer >
          <Organizer>
              $S:pcdata
          </Organizer>
      </myDTD_Organizer>               ←        .

C'12: <myDTD_Sponsor>
          <Sponsor>
              $S:pcdata
          </Sponsor>
      </myDTD_Sponsor>                 ←        .

C'13: <myDTD_Person  ssn=$S:value>
          <Person>
              $S:pcdata
          </Person>
      </myDTD_Person>                  ←        ⟨<id  value=$S:value />, f_id,myDTD, R⟩.
```

[11] B. C. Ghosh and V. Wuwongse. Conceptual Graph Programs and Their Declarative Semantics. *IEICE Transaction on Information and Systems*, E78-D(9):1208-1217, September 1995.

[12] G. Klostler, W. Kiebling, H. Thone and U. Guntzer. Fixpoint Iteration with Subsumption in Deductive Databases. *Journal of Intelligent Information Systems*, 4(2):123-148, March 1995.

[13] M. Kifer, G. Lausen and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the Association for Computing Machinery*, 42(4):741-843, July 1995.

[14] V. Wuwongse and B. C. Ghosh. Towards Deductive Object-Oriented Databases Based on Conceptual Graphs. In H. D. Pfeiffer and T. F. Nagle (eds.) *Proceedings of the 7th Annual Workshop on Conceptual Graphs*, Lecture Notes in Artificial Intelligence #754, 188-205, Springer-Verlag, Las Cruces, NM, USA, July 1992.

6. Output

[1] Ekawit Nantajeewarawat and Vilas Wuwongse, Defeasible Inheritance Through Specialization, *Computational Intelligence*, Vol. 17, No. 1, 2001 (to appear).

[2] Vilas Wuwongse and Ekawit Nantajeewarawat, Declarative Programs with Implicit Implication, *IEEE Transactions on Knowledge and Data Engineering* (1st revision).

[3] Ekawit Nantajeewarawat and Vilas Wuwongse, An Argumentation Approach to Semantics of Declarative Programs with Defeasible Inheritance, in P.S. Thiagarajan and R. Yap (eds.): *ASIAN'99*, Lecture Notes in Computer Science #1742, Springer-Verlag, pp.239-250, 1999.

[4] Chutiporn Anutariya, Vilas Wuwongse, Ekawit Nantajeewarawat and Kiyoshi Akama, Towards a Foundation for XML Document Databases, *Proceedings of the 1st Int. Conf. Electronic Commerce and Web Technologies*, Lecture Notes in Computer Science, Springer-Verlag, 2000 (to appear).

[5] Vilas Wuwongse, Kiyoshi Akama , Chutiporn Anutariya and Ekawit Nantajeewarawat, A Foundation for XML Databases: Data Model, *Int. J. Knowledge and Information Systems* (submitted).

[6] Chutiporn Anutariya, Vilas Wuwongse, Kiyoshi Akama and Ekawit Nantajeewarawat, A Foundation for XML Databases: DTD Modeling, *Int. J. Knowledge and Information Systems* (submitted).

7. การนำไปใช้ประโยชน์

เชิงวิชาการ

โครงการวิจัยนี้อธิบายอย่างละเอียดถึงลักษณะสำคัญต่างๆ ของระบบจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน โดยเฉพาะอย่างยิ่งในเรื่องของความสัมพันธ์และผลกระทบต่อกันระหว่างการนิรนัยและการถ่ายทอดคุณสมบัติ ซึ่งเป็นกระบวนการอนุมานหลักในระบบ ซึ่งเป็นพื้นฐานหลักในการกำหนดส่วนประกอบต่างๆ ของระบบ ในโครงการวิจัยนี้ได้มีการเสนอวิธีการกำหนดความหมายของโปรแกรมที่ใช้ในการบรรยาย

เนื้อหาของข้อมูลและความรู้ในระบบอย่างชัดเจน และได้เสนอวิธีการสำหรับประมวลผลเพื่อคำนวณหาความหมายของโปรแกรมที่กำหนดขึ้น ความหมายและวิธีการดังกล่าวเป็นพื้นฐานสำคัญในการศึกษาและวิเคราะห์เนื้อหาของข้อมูลและความรู้ในระบบอย่างละเอียด และเป็นพื้นฐานสำคัญในการออกแบบและการพัฒนาซอฟต์แวร์สำหรับปรับปรุงประสิทธิภาพการจัดเก็บและการประมวลผลข้อมูลและความรู้ในระบบ คณะผู้วิจัยคาดหวังว่าทฤษฎีที่เสนอขึ้นนี้จะเป็นรากฐานที่สำคัญสำหรับการพัฒนาระบบจัดการฐานข้อมูลและความรู้ในอนาคต

นอกจากนี้ คณะผู้วิจัยคาดหวังว่าโครงการวิจัยจะมีประโยชน์อย่างยิ่งสำหรับผู้ที่ต้องการศึกษาและค้นคว้าเกี่ยวกับระบบการจัดเก็บข้อมูลและความรู้เชิงวัตถุแบบอนุมาน แบบจำลองทางคณิตศาสตร์ที่เสนอในโครงการวิจัยสามารถนำไปใช้เป็นเครื่องมือช่วยในการทำความเข้าใจในระบบฐานข้อมูลและความรู้เชิงวัตถุแบบอนุมานหลากหลายลักษณะที่นักวิจัยหลายกลุ่มได้เสนอขึ้นมา การศึกษาระบบเหล่านี้โดยตรงทีละระบบโดยปราศจากทฤษฎีพื้นฐานร่วมอาจทำให้เกิดความสับสนขึ้นได้ง่าย เนื่องจากระบบเหล่านี้มีการกำหนดลักษณะสำคัญต่างๆ ของระบบโดยใช้ไวยากรณ์เฉพาะที่แตกต่างกันออกไป และมีความคลาดเคลื่อนกันในการกำหนดความหมายของลักษณะต่างๆ เหล่านี้ ซึ่งส่งผลให้เกิดความคลาดเคลื่อนกันในการกำหนดความหมายโดยรวมของโปรแกรมในระบบ

โครงการวิจัยนี้ได้มีส่วนช่วยสร้างนักวิจัยใหม่ 2 คน โดยคนแรกจบการศึกษาระดับปริญญาเอกแล้วและขณะนี้เป็นผู้ช่วยศาสตราจารย์ที่สถาบันเทคโนโลยีนานาชาติสิรินธร มหาวิทยาลัยธรรมศาสตร์ ส่วนอีกคนกำลังศึกษาในระดับปริญญาเอกที่ AIT

นอกจากนี้โครงการวิจัยนี้ยังมีส่วนกระชับความร่วมมือในการทำวิจัยของคณะผู้วิจัยกับ Prof. Kiyoshi Akama ที่มหาวิทยาลัย Hokkaido

เชิงพาณิชย์

เป็นที่ทราบกันดีว่าในสองทศวรรษที่ผ่านมา ผู้ผลิตซอฟต์แวร์จำนวนหนึ่งได้พัฒนาซอฟต์แวร์ระบบการจัดการฐานข้อมูลแบบความสัมพันธ์ (Relational Database Management System) ขึ้นมา และได้มีการนำซอฟต์แวร์ดังกล่าวนี้ไปใช้ในงานประยุกต์ทางด้านธุรกิจและด้านอื่นๆ อย่างประสบความสำเร็จอย่างกว้างขวาง อย่างไรก็ดี นักวิทยาการคอมพิวเตอร์ส่วนใหญ่ยังมีความเห็นว่า ระบบฐานข้อมูลแบบสัมพันธ์ยังคงมีข้อจำกัดอยู่อีกหลายประการ ตัวอย่างเช่น ข้อจำกัดในเรื่องของ โครงสร้างข้อมูล (โครงสร้างข้อมูลพื้นฐาน ของระบบฐานข้อมูลแบบความสัมพันธ์เป็นโครงสร้างแบบ record ซึ่งเป็นอุปสรรคในการจัดเก็บข้อมูลที่มีโครงสร้างซับซ้อน) ข้อจำกัดในการค้นหาข้อมูลโดยใช้คำสั่งค้นหาที่มีการอ้างอิงถึงตัวเอง (Recursive Query) อุปสรรคในเรื่องของช่องว่างระหว่างภาษาที่ใช้เขียนโปรแกรมประยุกต์กับภาษาที่ใช้ในการหาข้อมูล (Impedance Mismatch Problem) ข้อจำกัดในการจัดเก็บข้อมูลแบบเป็นลำดับชั้น และการนำสารสนเทศโดยนัยที่มีอยู่ในข้อมูลไปใช้ให้เป็นประโยชน์ เป็นต้น

นักวิทยาการคอมพิวเตอร์จำนวนมากเชื่อว่า ระบบฐานข้อมูลเชิงวัตถุแบบอนุมานมีศักยภาพที่จะสามารถแก้ไขข้อจำกัดต่างๆเหล่านี้ และสามารถที่จะรองรับงานประยุกต์ประเภทต่างๆได้หลากหลายมากขึ้น ทฤษฎีที่เสนอขึ้นในโครงการวิจัย จะเป็นพื้นฐานที่สำคัญสำหรับการพัฒนาซอฟต์แวร์ระบบการจัดการฐานข้อมูลเชิงวัตถุแบบอนุมาน (Deductive Object-Oriented Database Management System) ทฤษฎีบทต่าง ๆ ที่เชื่อมโยงความหมายของโปรแกรมกับจุดตรึงที่น้อยที่สุด (The Least Fixpoint) ของตัวดำเนินการ (operator) ที่ถูกกำหนดได้จากโปรแกรม สามารถนำไปประยุกต์ใช้ได้โดยตรง ในการประมวลผลหาเนื้อหาของข้อมูลในฐานข้อมูล และการค้นหาข้อมูลในระบบ โดยใช้อัลกอริธึม (Algorithm) มาตรฐานในการคำนวณหาค่าจุดตรึงที่น้อยที่สุดของตัวดำเนินการแบบโมโนโทนิก (Monototic Operator)

นอกจากนี้ในทฤษฎีที่เสนอขึ้น มีการวิเคราะห์และอธิบายถึงความหมายของโปรแกรมและเนื้อหาของฐานข้อมูลและความรู้อย่างละเอียดชัดเจน ซึ่งความชัดเจนไม่กำกวมของความหมายดังกล่าวนี้เป็นพื้นฐานที่จำเป็นอย่างยิ่งในการพัฒนาซอฟต์แวร์ประเภท Optimization Tools สำหรับใช้ในการปรับปรุงประสิทธิภาพของการจัดเก็บ ค้นหา และประมวลผลข้อมูลในฐานข้อมูลและความรู้ ตัวอย่างของซอฟต์แวร์ประเภทนี้ได้แก่ ซอฟต์แวร์ที่ทำการเปลี่ยนคำสั่งค้นหาข้อมูลอย่างหนึ่งไปเป็นคำสั่งอีกอย่างหนึ่ง ที่ให้ผลลัพธ์เหมือนเดิมแต่ใช้เวลาและทรัพยากรของระบบในการค้นหาน้อยลงกว่าเดิม ซอฟต์แวร์ที่ใช้ลดขนาดของฐานข้อมูลลงโดยที่ยังคงมีเนื้อหาและความหมายคงเดิม เป็นต้น

ท้ายสุดนี้งานวิจัยประยุกต์ในส่วนของ XML คาดว่าจะมีประโยชน์อย่างยิ่งในทางปฏิบัติ เนื่องจาก XML ได้กลายเป็นมาตรฐานของการแสดงและแลกเปลี่ยนข้อมูลบน Internet ซึ่งจะครอบคลุมงานเกือบทุกประเภท ไม่ว่าจะเป็น พาณิชย์อิเล็กทรอนิกส์ ห้องสมุดดิจิตอล หรือ การเรียนรู้ทางไกล

Output #1

To appear in Computational Intelligence, Vol. 17, No. 1, 2001

Defeasible Inheritance Through Specialization

Subject Categories:

Formal Underpinnings, Knowledge Representation

EKAWIT NANTAJEEWARAWAT

ekawit@siit.tu.ac.th

Department of Information Technology

Sirindhorn International Institute of Technology, Thammasat University

P.O. Box 22, Thammasat-Rangsit Post Office, Pathumthani 12121

Thailand

VILAS WUWONGSE

vw@cs.ait.ac.th

Computer Science and Information Management Program

School of Advanced Technologies, Asian Institute of Technology

P.O. Box 4, Klongluang, Pathumthani 12120

Thailand

June 13, 2000

Abstract

Typed substitution provides a means of capturing inheritance in logic deduction systems. However, in the presence of method overriding and multiple inheritance, inheritance is known to be nonmonotonic and the semantics of programs becomes a problematic issue. This paper attempts to provide a general framework, based on Dung's argumentation theoretic framework, for developing a natural semantics for programs with dynamic nonmonotonic inheritance. The relationship between the presented semantics and perfect model (with overriding) semantics, proposed by Dobbie and Topor (1995), is investigated. It is shown that for inheritance-stratified programs, the two semantics coincide. However, the proposed semantics also provides correct skeptical meanings for the programs which are not inheritance-stratified.

Key words: nonmonotonic inheritance, argumentation, skeptical semantics, inheritance stratification, deductive object-oriented systems, dynamic method resolution

1 Introduction

Deduction and inheritance are two important reasoning mechanisms in deductive object-oriented database (DOOD) systems. Most DOOD languages provide these two mechanisms in certain ways. However, there are subtle differences in their interpretation and realization, *e.g.*, datalog^{meth} (Abiteboul, Lausen, Uphoff and Waller 1993) captures inheritance by transforming subclass relationships into rules of the form $class(X) \leftarrow subclass(X)$, LOGIN (Aït-Kaci and Nasr 1986) and LIFE (Aït-Kaci and Podelski 1993) incorporate inheritance into unification algorithms, F-logic (Kifer, Lausen and Wu 1995) considers inheritance as implicit implication on an interpretation domain, and Gulog (Dobbie and Topor 1995) models inheritance by means of typed substitutions. This paper focuses on the interaction between deduction and the inheritance that is realized through typed substitutions, and its effects on program semantics.

1.1 Motivation

Most DOOD languages support inclusion polymorphism (or subtyping), *i.e.*, the extension of one type (the set of all individuals belonging to the type) can be defined to be a subset of the extension of another type. With inclusion polymorphism, inheritance can be captured in an intuitive way by means of typed substitutions. To illustrate, suppose that ait is an object of type int(ernational)-school and int-school is a subtype of school. Then, given a program clause:

$$C1: \quad \mathsf{X{:}\,school[medium\text{-}of\text{-}teaching} \rightarrow \mathsf{thai]} \quad \text{if} \quad \mathsf{X[located\text{-}in} \rightarrow \mathsf{thailand]},$$

which is intended to state that for any object X of type school, the medium of teaching at X is thai, if X is located in thailand; one can obtain by the application of the typed substitution {X: school/ait} to $C1$ the ground clause:

$$G1: \quad \mathsf{ait[medium\text{-}of\text{-}teaching} \rightarrow \mathsf{thai]} \quad \text{if} \quad \mathsf{ait[located\text{-}in} \rightarrow \mathsf{thailand]}.$$

Naturally, the clause $C1$ can be viewed as a conditional definition of the method medium-of-teaching attached to the type (class[1]) school and the clause $G1$ as a definition of the same method inherited from the type school for the object ait.

However, under the usual way of defining program semantics, *e.g.*, the minimal model semantics, inasmuch as every single ground instance of a program is required

[1] In this paper, the terms "type" and "class" are used interchangeably.

to be satisfied by the meaning of that program, the inheritance thus obtained is inherently *indefeasible*, *i.e.*, every inherited definition will be compulsorily used.[2] Undeniably, indefeasible inheritance is not always most appropriate. Inheritance of a property can reasonably be expected to be blocked owing to property overriding. For example, let a unit clause:

$C2:$ X: int-school[medium-of-teaching $\rightarrow$ english],

defining the method medium-of-teaching for the objects of type int-school, be also given. Then, as int-school is a subtype of school, it is reasonable to suppose that the definition of medium-of-teaching for ait obtained from $C2$, *i.e.*,

$G2:$ ait[medium-of-teaching $\rightarrow$ english],

is more specific than the previous inherited definition $G1$ and is likely to supersede $G1$.

In the case of multiple inheritance, where there are several possible inherited definitions none of which is more specific than the others, with indefeasible inheritance all the definitions will be employed. This is again not always natural. Another reasonable option is either to selectively use only some of those inheritable definitions based on some resolution criteria and some additional information, leaving the others ineffective, or even to skeptically discard all of them.

When some inherited information does not apply in the presence of more specific information or some other eligible heritage, inheritance is said to be *defeasible*. In general, the choice of the most suitable inheritance strategy (indefeasible or defeasible) may seem to be a matter of opinion. However, indefeasible inheritance tends to cause unintended inconsistency when methods are required to behave as (partial) functions, *i.e.*, when the invocation of a method on a particular object is required to yield a unique value whenever that invocation is defined. Referring to the clauses $G1$ and $G2$ above, for example, if the method medium-of-teaching is supposed to return a single value for the object ait, then $G1$ conflicts with $G2$ when they are both active. With such a functionality requirement, defeasible inheritance is therefore particularly preferable. Furthermore, from the modelling viewpoint, it has been widely recognized that defeasible inheritance appears to be more suitable for reasoning about the behavioral aspect of objects.

[2]Indefeasible inheritance is also called *strict* inheritance.

Indefeasible inheritance is known as monotonic inheritance, because it always increases derived information monotonically as the number of program clauses increases. By contrast, defeasible inheritance typically causes *nonmonotonic* behavior, *i.e.*, addition of a new clause to a program may result in the withdrawal of some conclusions which were previously derivable by inheritance from that program. Coping with defeasible (nonmonotonic) inheritance is not simple. One primary problem is how to deal with situations wherein some inherited information conflicts with some other information (possibly also inherited). This subject has been intensively studied in artificial intelligence (Touretzky 1986; Horty, Thomason and Touretzky 1990; Stein 1992; Thirunarayan and Kifer 1993; Dung and Son 1995); however, most of these studies discussed nonmonotonic inheritance not in the context of rule-based deductive systems.

1.2 The Proposed Work

This paper applies Dung's theory of argumentation (Dung 1995) to the development of an appropriate declarative semantics for programs with defeasible inheritance. In order to resolve inheritance conflicts, a binary relation on program ground clauses, called the *domination relation*, which determines among possibly conflicting (inherited) definitions whether one is intended to be preferable to another, is required. The domination relation, for example, may provide the information that between the clauses $G1$ and $G2$ given in Subsection 1.1, $G2$ is more suitable. A program will be transformed into an argumentation framework, which captures the logical interaction between the intended deduction and domination; and, then, the meaning of the program will be defined based on the grounded extension of this argumentation framework.

Using this approach, conflict resolution is performed *dynamically* with respect to the applicability of method definitions. That is, the domination of one method definition over another is effective only if the antecedent of the dominating definition succeeds. The appropriateness of dynamic method resolution in the context of deductive rule-based systems, where the definitions of methods in a class are often conditional and may be inapplicable to certain objects of the class, is advocated by Abiteboul, Lausen, Uphoff and Waller (1993). In particular, with the possibility of overriding, when the definitions in the most specific class are not applicable, it is reasonable to endeavour to apply those in a more general class.

In order to argue for the correctness and the generality of the presented semantics in the presence of method overriding, its relationship to the perfect model (with overriding) semantics proposed by Dobbie and Topor (Dobbie and Topor 1993; Dobbie and Topor 1995) is investigated. The investigation reveals that these two semantics coincide for inheritance-stratified programs. Furthermore, while the perfect model semantics fails to provide natural meanings for programs which are not inheritance-stratified, the presented semantics still yields their correct skeptical meanings.

For the sake of simplicity and generality, this paper uses Akama's axiomatic theory of logic programs (Akama 1993), called DP theory (the theory of declarative programs), as its primary logical basis. The rest of this paper is organized as follows. Section 2 recalls some basic definitions and results from Akama's DP theory and Dung's argumentation-theoretic foundation. Section 3 describes the proposed semantics. Section 4 formulates the notions of inheritance-stratified program and perfect model with overriding based on DP theory. Section 5 establishes the relationship between the proposed semantics and the perfect model (with overriding) semantics. Section 6 compares the presented approach with works on inheritance networks.

2 Preliminaries

2.1 DP Theory

DP theory (Akama 1993) is an axiomatic theory which purports to generalize the concept of conventional logic programs to cover a wider variety of data domains. As an introduction to DP theory, the notion of a specialization system is reviewed first. It is followed by the concepts of declarative programs and their minimal model semantics on a specialization system.

Definition 1 (Specialization System) A *specialization system* is a 4-tuple $(\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ of three sets $\mathcal{A}, \mathcal{G}$ and $\mathcal{S}$, and a mapping μ from $\mathcal{S}$ to $partial\_map(\mathcal{A})$ (*i.e.*, the set of all partial mappings on $\mathcal{A}$), that satisfies the conditions:

1. $(\forall s, s' \in \mathcal{S})(\exists s'' \in \mathcal{S}) : \mu s'' = (\mu s') \circ (\mu s)$,

2. $(\exists s \in \mathcal{S})(\forall a \in \mathcal{A}) : (\mu s)a = a$,

3. $\mathcal{G} \subseteq \mathcal{A}$.

The elements of $\mathcal{A}$ are called *atoms*; the set $\mathcal{G}$ is called the *interpretation domain*; the elements of $\mathcal{S}$ are called *specialization parameters* or simply *specializations*; and the mapping μ is called the *specialization operator*. A specialization $s \in \mathcal{S}$ is said to be *applicable* to $a \in \mathcal{A}$, iff $a \in dom(\mu s)$. □

By formulating a suitable specialization operator together with a suitable set of specialization parameters, the typed-substitution operation can be regarded as a special form of specialization operation. Throughout this subsection, let $\Gamma = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ be a specialization system. A specialization in $\mathcal{S}$ will often be denoted by a Greek letter such as θ. When there is no danger of confusion, a specialization $\theta \in \mathcal{S}$ will be identified with the partial mapping $\mu\theta$ and used as a postfix unary (partial) operator on $\mathcal{A}$, *e.g.*, $(\mu\theta)a$ will be written as $a\theta$.

A declarative program on Γ is defined as a set of definite clauses constructed out of atoms in $\mathcal{A}$. Every logic program in the conventional theory can be regarded as a declarative program on some specialization system.

Definition 2 (Definite Clause and Declarative Program) Let X be a subset of $\mathcal{A}$. A *definite clause C on X* is a formula of the form:

$$a \quad \leftarrow \quad b_1, \ldots, b_n$$

where $n \geq 0$ and $a, b_1, \ldots, b_n$ are atoms in X. The atom a is denoted by $head(C)$ and the set $\{b_1, \ldots, b_n\}$ by $Body(C)$. A definite clause C such that $Body(C) = \emptyset$ is called a *unit clause*. The set of all definite clauses on X is denoted by $Dclause(X)$. An element of $Dclause(\mathcal{G})$ is called a *ground* clause. A *declarative program* on Γ is a (possibly infinite) subset of $Dclause(\mathcal{A})$. □

A declarative program will also be simply called a *program* in this paper.

Let C be a definite clause $(a \leftarrow b_1, \ldots, b_n)$ on $\mathcal{A}$. A definite clause C' is an *instance* of C, iff there exists $\theta \in \mathcal{S}$ such that θ is applicable to $a, b_1, \ldots, b_n$ and $C' = (a\theta \leftarrow b_1\theta, \ldots, b_n\theta)$. Such an instance C' of C is denoted by $C\theta$, and the set of all instances of C by $Instance(C)$. Given a declarative program P on Γ, $Gclause(P)$ denotes the set

$$\bigcup_{C \in P} (Instance(C) \cap Dclause(\mathcal{G})).$$

i.e., the set of all instances of clauses in P which are constructed solely out of atoms in $\mathcal{G}$.

An interpretation assigning truth values to the atoms in the interpretation domain $\mathcal{G}$ and the truth value of a definite clause on $\mathcal{A}$ with respect to a particular interpretation are next defined:

Definition 3 (Interpretation) An *interpretation* is a subset of $\mathcal{G}$. Given a definite clause C on $\mathcal{G}$, an interpretation I is said to *satisfy* C, iff

$$(head(C) \in I) \text{ or } (Body(C) \not\subseteq I).$$

A definite clause C on $\mathcal{A}$ is said to be *true* with respect to an interpretation I, iff

$$(\forall C' \in (Instance(C) \cap Dclause(\mathcal{G}))) : \ I \text{ satisfies } C'. \quad \square$$

As in the conventional theory, an interpretation I is a *model* of a declarative program P on Γ, iff all definite clauses in P are true with respect to I; and, the meaning of P is defined as the *minimal model* of P, which is the intersection of all models of P. A fixpoint characterization of this minimal model semantics is also discussed in (Akama 1993).

Examples 1 and 2 below demonstrate how to regard conventional logic programs and (simplified) typed logic programs with subtyping, respectively, as special forms of declarative programs.

Example 1 Let an alphabet $\Delta = (V, K, F, R)$ be given, where V, K, F and R are mutually disjoint sets of variables, constants, function symbols and predicate symbols, respectively. Let a specialization system $\Gamma_1 = (\mathcal{A}_1, \mathcal{G}_1, \mathcal{S}_1, \mu_1)$ be defined as follows: $\mathcal{A}_1$ is the set of all first-order atoms over Δ; $\mathcal{G}_1$ is the subset of $\mathcal{A}_1$ that consists of all variable-free atoms in $\mathcal{A}_1$; $\mathcal{S}_1$ is the set of all usual substitutions over Δ; and, for each $s \in \mathcal{S}_1$ and $a \in \mathcal{A}_1$, $(\mu_1 s)a$ is the result obtained by applying the substitution s to a in the usual way. From the basic concepts and results[3] for logic programming, it can be seen that Γ_1 satisfies all the three conditions of Definition 1. The declarative programs on Γ_1 are conventional logic programs, and their meanings according to DP theory are exactly their conventional meanings. $\square$

[3] See, for example, the first chapter of (Lloyd 1987).

Example 2 This example illustrates one among several possible ways of formulating specialization systems for typed logic programs. To simplify the presentation, only typed logic programs without function symbols will be considered. Let T be set of types partially ordered by $\leq$. Let V, K and R be mutually disjoint sets of variables, constants and predicate symbols, respectively. Assume that each variable in V as well as each constant in K has exactly one type in T, and each m-ary predicate symbol in R has a unique type of the form $\tau_1 \times \cdots \times \tau_m$, where the τ_i belong to T. Let a specialization system $\Gamma_2 = (\mathcal{A}_2, \mathcal{G}_2, \mathcal{S}_2, \mu_2)$ be defined as follows:

1. $\mathcal{A}_2$ is the set of all typed atoms of the form $p(t_1, \ldots, t_m)$, where $m \geq 1$, p is an m-ary predicate symbol in R having type $\tau_1 \times \cdots \times \tau_m$ and for each $i \in \{1, \ldots, m\}$, t_i belongs to either V or K and if the type of t_i is τ_i', then $\tau_i' \leq \tau_i$.

2. $\mathcal{G}_2$ is the subset of $\mathcal{A}_2$ that consists of all variable-free typed atoms in $\mathcal{A}_2$.

3. $\mathcal{S}_2$ is the set of all typed substitutions of the form $\{v_1/t_1, \ldots, v_n/t_n\}$, where the v_i are distinct variables in V, each of the t_i belongs to either V or K, and for each $j \in \{1, \ldots, n\}$, $v_j \neq t_j$ and if v_j has type τ_j and t_j has type τ_j', then $\tau_j' \leq \tau_j$.

4. For each $s \in \mathcal{S}_2$, and $a \in \mathcal{A}_2$, $(\mu_2 s)a$ is the result obtained by applying the typed substitution s to a in the usual way.

Given any $s, s' \in \mathcal{S}_2$, it is clear that if s'' denotes the composition, defined in the usual way, of s and s', then $(\mu_2 s'')a = ((\mu_2 s) \circ (\mu_2 s'))a$, for each $a \in \mathcal{A}_2$. So Γ_2 satisfies Condition 1 of Definition 1. Obviously, Γ_2 also satisfies the other two conditions of Definition 1. The declarative programs on Γ_2 are typed logic programs, and the declarative semantics defined in DP theory yields their expected meanings. □

2.2 Argumentation Framework

Based on the basic idea that a statement is believable if some argument supporting the statement can be successfully defended against its counterarguments, Dung has developed an abstract theory of argumentation and demonstrated that many of the major approaches to nonmonotonic reasoning in artificial intelligence can be viewed as special forms of argumentation (Dung 1995). In this subsection, the basic concepts and results from this theory are summarized.

Definition 4 (Argumentation Framework) An *argumentation framework* is a pair $(AR, attacks)$, where AR is a set and *attacks* is a binary relation on AR. The elements of AR are called *arguments*. □

In the sequel, let $AF = (AR, attacks)$ be an argumentation framework. An argument $a \in AR$ is said to *attack* an argument $b \in AR$, iff $(a, b) \in attacks$. A set $A \subseteq AR$ is said to *attack* an argument $b \in AR$, iff some argument in A attacks b. An argument $a \in AR$ is said to *attack* a set $B \subseteq AR$, iff a attacks some argument in B.

The concept of acceptability of argument and the notions of conflict-free and admissible sets of arguments are now recalled.

Definition 5 (Acceptable Argument) An argument $a \in AR$ is said to be *acceptable* with respect to a set $A \subseteq AR$, iff, for each $b \in AR$, if b attacks a, then A attacks b. □

Definition 6 (Conflict-Free Set and Admissible Set)

1. A set $A \subseteq AR$ is said to be *conflict-free*, iff there do not exist arguments $a, b \in A$ such that a attacks b.

2. A set $A \subseteq AR$ is said to be *admissible*, iff A is conflict-free and every argument in A is acceptable with respect to A. □

The credulous semantics and the stable semantics of AF are defined by the notions of preferred extension and stable extension, respectively:

Definition 7 (Preferred Extension) A *preferred extension* of AF is a maximal (with respect to set inclusion) admissible subset of AR. □

Definition 8 (Stable Extension) A set $A \subseteq AR$ is called a *stable extension* of AF, iff A is conflict-free and A attacks every argument in $AR - A$. □

Lemma 1 *A set $A \subseteq AR$ is a stable extension of AF, iff*

$$A = \{a \in AR \mid A \text{ does not attack } a\}. \quad \square$$

The grounded (skeptical) semantics of AF is defined (Definition 10) as the least fixpoint of the characteristic function of AF, which is given below.

Definition 9 (Characteristic Function) The *characteristic function* of AF, $F_{AF}: 2^{AR} \rightarrow 2^{AR}$, is defined by

$$F_{AF}(X) \quad = \quad \{a \mid a \text{ is acceptable with respect to } X\},$$

for each $X \subseteq AR$. □

Proposition 1 *F_{AF} is monotonic with respect to set inclusion, but, in general, is not continuous. However, if for each argument $a \in AR$, there exist only finitely many arguments in AR which attack a, then F_{AF} is continuous.* □

Definition 10 (Grounded Extension) The *grounded extension* of an argumentation framework AF is the least fixpoint of F_{AF}. □

Extensions of the three kinds are illustrated by the next two examples.

Example 3 Let $AF_1 = (AR_1, attacks)$, where $AR_1 = \{a, b, c\}$ and $attacks = \{(a,b),(b,c)\}$. AF_1 has only one preferred extension, *i.e.*, $\{a, c\}$, which is also its only stable extension. Since $F_{AF_1}(\emptyset) = \{a\}$ and $F^2_{AF_1}(\emptyset) = \{a, c\} = F^3_{AF_1}(\emptyset)$, the grounded extension of AF_1 is also the set $\{a, c\}$. □

Example 4 Let $AF_2 = (AR_2, attacks)$, where $AR_2 = \{a, b, c, d\}$ and $attacks = \{(b,c),(c,b),(b,d)\}$. Then, AF_2 has two preferred extensions, *i.e.*, $\{a, b\}$ and $\{a, c, d\}$, which are also stable extensions. As $F_{AF_2}(\emptyset) = \{a\} = F^2_{AF_2}(\emptyset)$, the grounded extension of AF_2 is $\{a\}$. □

Well-foundedness of an argumentation framework, recalled next, is a sufficient condition for the coincidence between the grounded semantics, preferred extension semantics and stable semantics (Theorem 1).

Definition 11 (Well-Founded Argumentation Framework) An argumentation framework is *well-founded*, iff there exists no infinite sequence of arguments $a_0, a_1, \ldots, a_n, \ldots$ such that for each $i \geq 0$, a_{i+1} attacks a_i. □

Theorem 1 *Every well-founded argumentation framework has exactly one preferred extension and one stable extension. Moreover, its grounded extension, preferred extension and stable extension are equal to each other.* □

Example 5 The argumentation framework AF_2 of Example 4 is not well-founded, since the arguments b and c attack each other. □

3 The Proposed Semantics

In the sequel, let $\Gamma = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ be a specialization system and P a declarative program on Γ. Let *dominates* be a binary relation on $Gclause(P)$. A ground clause C of P is said to *dominate* another ground clause C' of P, iff $(C, C') \in$ *dominates*. It will be assumed henceforth that the relation *dominates* prioritizes the ground clauses of P; more precisely, for any ground clauses C, C' of P, C dominates C', iff C is preferable to C' and whenever $Body(C)$ is satisfied, C' will be inactive.[4] It should be emphasized that the domination of a ground clause C over another ground clause C' is intended to be *dynamically* operative with respect to the applicability of C, *i.e.*, the domination is effective only if the condition part of C is satisfied. The relation *dominates* will also be referred to as the *domination relation* of P.

A program is said to be *domination-free*, iff there do not exist any ground clauses C, C' of the program such that C dominates C'.

3.1 Derivation Trees

The notion of a derivation tree of a program will be introduced first. A derivation tree of P represents a derivation of one conclusion from P. It will be considered as an argument that supports its derived conclusion. Every conclusion in the minimal model of P is supported by at least one derivation tree of P.

Definition 12 (Derivation Tree) A *derivation tree* of P is defined inductively as follows:

1. If C is a unit clause in $Gclause(P)$, then the tree of which the root is C and the height is 0 is a *derivation tree* of P.

2. If $C = (a \leftarrow b_1, \ldots, b_n)$ is a clause in $Gclause(P)$ such that $n > 0$ and $T_1, \ldots, T_n$ are derivation trees of P with roots $C_1, \ldots, C_n$, respectively, such that

$$head(C_i) = b_i,$$

for each $i \in \{1, \ldots, n\}$, then the tree of which the root is C and the immediate subtrees are exactly $T_1, \ldots, T_n$ is a *derivation tree* of P.

[4] The relation *dominates* is inspired partly by the relationship "possibly overrides" in (Dobbie and Topor 1993; Dobbie and Topor 1995).

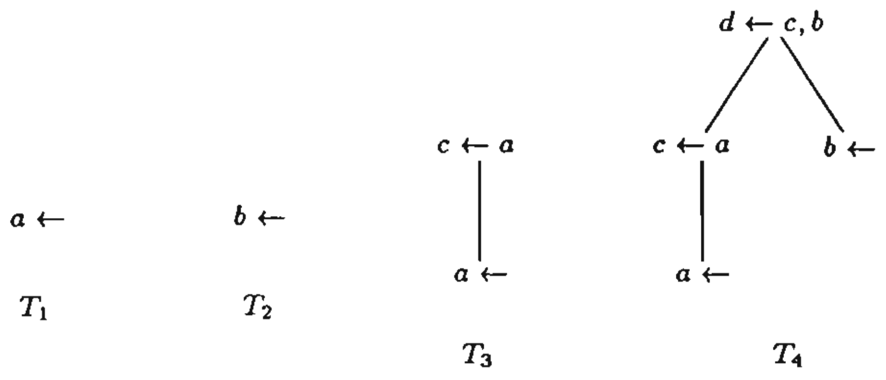

Figure 1: The derivation trees of the program P_1.

3. Nothing else is a derivation tree of P.

The set of all derivation trees of P is denoted by $Tree(P)$. □

Note that as the derivation trees are inductively generated, only well-founded derivation trees are obtained. In the sequel, for any derivation tree T of P, let $root(T)$ and $height(T)$ denote the root and the height, respectively, of T.

Example 6 Let P_1 be a declarative program which consists of the following five ground clauses:

$$\begin{array}{lll} a & \leftarrow & \\ b & \leftarrow & \\ c & \leftarrow & a \\ d & \leftarrow & c, b \\ f & \leftarrow & e. \end{array}$$

Then, P_1 has exactly four derivation trees, which are shown by Figure 1. Note that the derivation trees T_1, T_2, T_3 and T_4 in the figure depict the derivation of the conclusions a, b, c and d, respectively. □

In the ensuing discussion, a derivation tree T will be regarded as an argument that supports the activation of the ground clause $root(T)$ (and, therefore, supports the conclusion $head(root(T))$).

3.2 Transformation into Argumentation Framework

In order to define the meaning of P with respect to the domination relation, the program P will be transformed into an argumentation framework $AF_t(P)$, which

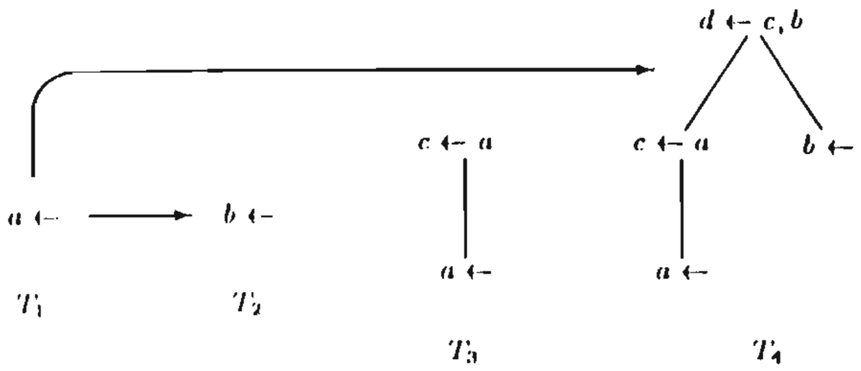

Figure 2. The argumentation framework for the program P_1.

provides an appropriate structure for understanding the dynamic interaction of the deduction process of P and the specified domination relation. Intuitively, one argument (derivation tree) attacks another argument (derivation tree), when the ground clause supported by the former dominates some ground clause used in the construction of the latter.

Definition 13 The argumentation framework $AF_t(P) = (AR, \mathit{attacks})$ is defined as follows:

1. $AR = Tree(P)$

2. For any $T, T' \in AR$, T attacks T', iff $root(T)$ dominates some node of T'.

   □

Example 7 Referring to the program P_1 of Example 6, suppose that the ground clause $a \leftarrow$ dominates the ground clause $b \leftarrow$, and for any other two ground clauses in P_1, one does not dominate the other. Then $AF_t(P_1) = (Tree(P_1), \mathit{attacks})$, where $Tree(P_1)$ consists of the four derivation trees depicted by Figure 1 and $\mathit{attacks} = \{(T_1, T_2), (T_1, T_4)\}$ as represented by the arrows in Figure 2. (Note that T_1 attacks T_4 as the root of T_1 dominates the right leaf of T_4.) □

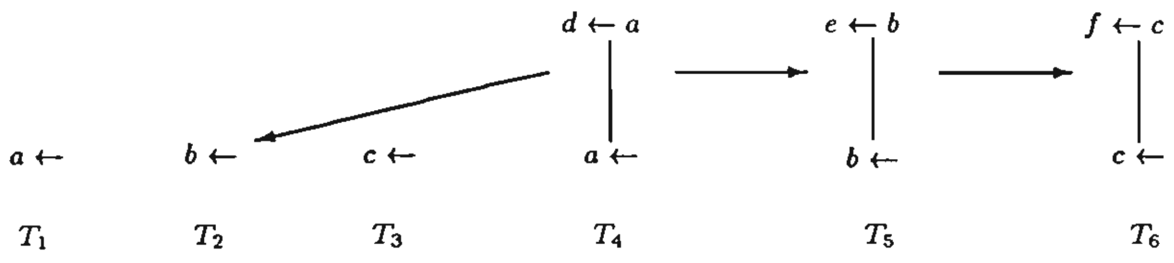

Figure 3: The argumentation framework for the program P_2.

Example 8 Let a declarative program P_2 comprise the following ground clauses:

$$\begin{aligned} a &\leftarrow \\ b &\leftarrow \\ c &\leftarrow \\ d &\leftarrow a \\ e &\leftarrow b \\ f &\leftarrow c. \end{aligned}$$

Let $d \leftarrow a$ dominate $b \leftarrow$ and $e \leftarrow b$ dominate $f \leftarrow c$, and assume that for any other two ground clauses in P_2, one does not dominate the other. Then $AF_\iota(P_2) = (Tree(P_2), attacks)$, where $Tree(P_2)$ consists of the six derivation trees given in Figure 3 and $attacks = \{(T_4, T_2,), (T_4, T_5), (T_5, T_6)\}$ as shown by the arrows between the derivation trees in the figure. □

3.3 Grounded-Extension-Based Semantics

The meaning of a program is now defined as the set of all conclusions which are supported by some arguments in the grounded extension of the argumentation framework for the program.

Definition 14 (Grounded-Extension-Based Meaning) Let GE be the grounded extension of $AF_\iota(P)$. Then the *grounded-extension-based meaning* of P, denoted by $\mathcal{M}_P^{GE}$, is defined by

$$\mathcal{M}_P^{GE} = \{head(root(T)) \mid T \in GE\}. \quad \square$$

Example 9 Consider $AF_\iota(P_1)$ of Example 7 (Figure 2). Let F be its characteristic function (see Definition 9). Clearly, $F(\emptyset) = \{T_1, T_3\} = F(F(\emptyset))$. Thus $F(\emptyset)$ is the grounded extension of $AF_\iota(P_1)$, and, then, $\mathcal{M}_{P_1}^{GE} = \{a, c\}$. □

Example 10 Refer to Example 8 (Figure 3). Let F be the characteristic function of $AF_\iota(P_2)$, then $F(\emptyset) = \{T_1, T_3, T_4\}$, and $F^2(\emptyset) = \{T_1, T_3, T_4, T_6\} = F^3(\emptyset)$. Thus $F^2(\emptyset)$ is the grounded extension of $AF_\iota(P_2)$, whence $\mathcal{M}^{\text{GE}}_{P_2} = \{a, c, d, f\}$. □

Example 10 above also illustrates the dynamic conflict resolution in the proposed approach, *i.e.*, the domination of the ground clause $e \leftarrow b$ over the ground clause $f \leftarrow c$ does not always prevent the activation of the latter.

The next example shows how to deal with the problem raised at the beginning of the paper.

Example 11 Let ait be an instance of type int-school and int-school a subtype of school. Let P_3 be a declarative program which consists of the following three clauses:

X: school[medium-of-teaching → thai]	←	X[located-in → thailand]
X: int-school[medium-of-teaching → english]	←	
ait[located-in → thailand]	← .	

For the sake of simplicity, assume that P_3 has only three ground clauses:

$G1$:	ait[medium-of-teaching → thai]	←	ait[located-in → thailand]
$G2$:	ait[medium-of-teaching → english]	←	
$G3$:	ait[located-in → thailand]	← .	

As explained at the beginning of Section 1, $G2$ is supposed to override $G1$; therefore, let $G2$ dominate $G1$. Figure 4 depicts the resulting argumentation framework $AF_\iota(P_3)$, where $m\text{-}t$, $m\text{-}e$ and $l\text{-}t$ denote the ground atoms ait[medium-of-teaching → thai], ait[medium-of-teaching → english] and ait[located-in → thailand], respectively. Now let F be the characteristic function of $AF_\iota(P_3)$. As $F(\emptyset) = \{T_1, T_3, \} = F^2(\emptyset)$, $F(\emptyset)$ is the grounded extension of $AF_\iota(P_3)$. Thus $\mathcal{M}^{\text{GE}}_{P_3}$ is the set

{ait[located-in → thailand], ait[medium-of-teaching → english]},

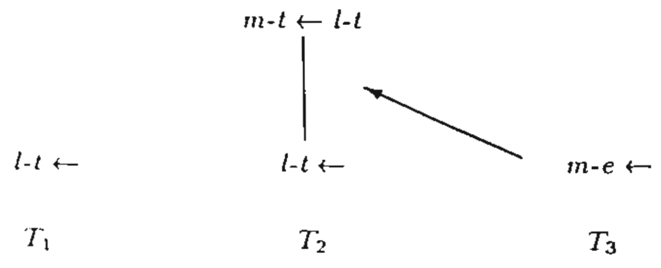

Figure 4 The argumentation framework for the program P_3.

which is the expected meaning of P_3. □

The next two examples illustrate method resolution in case of multiple inheritance. The first one shows how the proposed approach deals with the well-known Nixon's Diamond. The second discusses the case when method definitions are conditional.

Example 12 Let nixon be an individual both of type quaker and of type republican, and P_4 a declarative program consisting of the following two clauses:

$$\begin{array}{lll} C1: & \mathsf{X{:}quaker[policy \to pacifist]} & \leftarrow \\ C2: & \mathsf{X{:}republican[policy \to hawk]} & \leftarrow . \end{array}$$

For simplicity, assume that $C1$ and $C2$ have as their ground instances only the clauses $G1$ and $G2$, given below, respectively:

$$\begin{array}{lll} G1: & \mathsf{nixon[policy \to pacifist]} & \leftarrow \\ G2: & \mathsf{nixon[policy \to hawk]} & \leftarrow . \end{array}$$

Suppose that being a quaker and being a republican are believed to neutralize each other, and, consequently, that the domination relation is defined in such a way that $G1$ and $G2$ dominate each other. Then, as the derivation trees of P_4 attack each other, it is clear that $\mathcal{M}^{\mathsf{GE}}_{P_4}$ is the empty set, which is the expected meaning of P_4. On the other hand, suppose that being a republican is believed to have more influence than being a quaker, and, then, that $G2$ is considered to dominate $G1$ but not vice versa. It is readily seen that $\mathcal{M}^{\mathsf{GE}}_{P_4}$ now becomes the set $\{\mathsf{nixon[policy \to hawk]}\}$, which is, in this case, the desired meaning of P_4. □

Example 13 Let tom belong both to type student and to type employee. Consider a program P_5 comprising the following five clauses:

$$\begin{array}{lll} C1: & \mathsf{X{:}student[residence \to north\text{-}dorm]} & \leftarrow \\ & & \mathsf{X[lives\text{-}in \to rangsit\text{-}campus]}, \\ & & \mathsf{X[sex \to male]} \\ C2: & \mathsf{X{:}employee[residence \to west\text{-}apartments]} & \leftarrow \\ & & \mathsf{X[lives\text{-}in \to rangsit\text{-}campus]}, \\ & & \mathsf{X[marital\text{-}status \to married]} \\ C3: & \mathsf{tom[lives\text{-}in \to rangsit\text{-}campus]} & \leftarrow \\ C4: & \mathsf{tom[sex \to male]} & \leftarrow \\ C5: & \mathsf{tom[marital\text{-}status \to married]} & \leftarrow . \end{array}$$

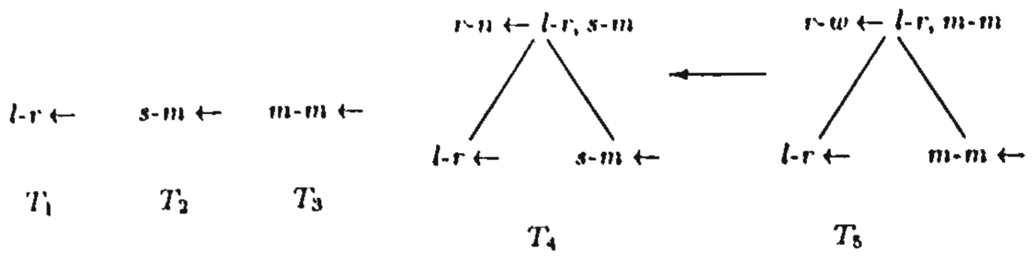

Figure 5: The argumentation framework for the program P_5.

Assume, for simplicity, that $C1$ and $C2$ have the clauses, given below, $G1$ and $G2$, respectively, as their only ground instances:

$G1$: tom[residence → north-dorm] ←
tom[lives-in → rangsit-campus],
tom[sex → male]

$G2$: tom[residence → west-apartments] ←
tom[lives-in → rangsit-campus],
tom[marital-status → married].

Suppose that students who are also employees usually prefer the accommodation provided for employees, and, therefore, that $G2$ dominates $G1$. Figure 5 shows the argumentation framework $AF_1(P_5)$, where *l-r*, *s-m*, *m-m*, *r-n* and *r-w* denote tom[lives-in → rangsit-campus], tom[sex → male], tom[marital-status → married], tom[residence → north-dorm] and tom[residence → west-apartments], respectively. Obviously, $\mathcal{M}_{P_5}^{GE}$ contains tom[residence → west-apartments] but does not contain tom[residence → north-dorm], and provides the desired meaning of P_5.

Next, suppose that the clause $C5$ is removed from P_5. Then, as T_5 in Figure 5 is now not a derivation tree of P_5, the domination of $G2$ over $G1$ is not effective. As a result, instead of containing tom[residence → west-apartments], $\mathcal{M}_{P_5}^{GE}$ contains tom[residence → north-dorm]; and, it still yields the correct meaning of P_5 in this case. □

In general, $\mathcal{M}_{P}^{GE}$ is a subset of the minimal model of P. However, it is readily seen that

Proposition 2 *If P is domination-free, then $\mathcal{M}_{P}^{GE}$ is the minimal model of P.* □

Proof If P is domination-free, then GE is the set of all derivation trees of P. Thus $\mathcal{M}_{P}^{GE}$ is exactly the minimal model of P. ■

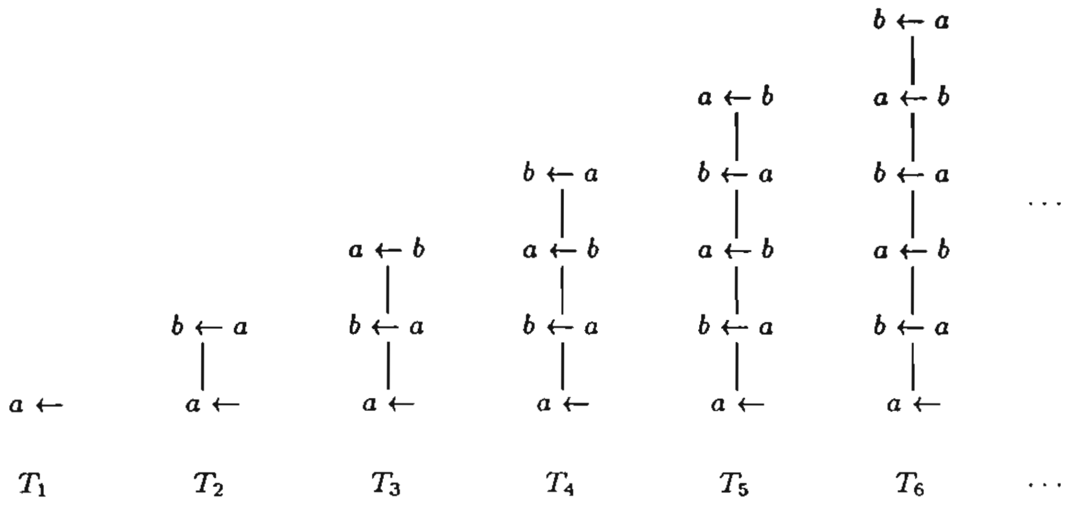

Figure 6: Infinitely many derivation trees of P_6.

3.4 Computing on Equivalence Classes of Derivation Trees

An atom a is said to be *self-dependent with respect to P*, iff there exists a sequence of (not necessarily distinct) clauses $C_1, \ldots, C_n$, where $n > 1$, in $Gclause(P)$ such that $head(C_1) = a = head(C_n)$ and for each $i \in \{1, \ldots, n-1\}$, $head(C_i) \in Body(C_{i+1})$. The next example demonstrates that if there exists a self-dependent atom with respect to P, then the set of all derivation trees of P may be infinite (even when $Gclause(P)$ is a finite set).

Example 14 Let P_6 be a program consisting of the following three ground clauses:

$$\begin{array}{lll} a & \leftarrow & \\ b & \leftarrow & a \\ a & \leftarrow & b. \end{array}$$

The atom a is self-dependent with respect to P_6. (So is the atom b.) P_6 has infinitely many derivation trees, as depicted by Figure 6. $\square$

As a consequence, it is, in general, not possible to construct the set of all derivation trees of a given program entirely in finitely many discrete computation steps. Moreover, since the number of nodes in a derivation tree is, in the worst case, exponential in the height of the tree, determining whether one derivation tree attacks another by examining directly whether the root of the former dominates some node of the latter is, in general, inefficient.

However, a closer examination of the interrelation among the derivation trees in $AF_\iota(P)$ reveals that it is not always necessary to generate all the derivation trees of P so as to determine $\mathcal{M}_P^{GE}$. Different derivation trees with the same root and the same set of nodes, *e.g.*, T_3 and T_5 in Figure 6, always support the same conclusion, attack the same derivation trees, and, furthermore, are attacked by the same derivation trees. Such derivation trees can therefore be considered to be equivalent to each other in this sense, and it is therefore sufficient to use only one of them in the computation of $\mathcal{M}_P^{GE}$. It will be seen in this subsection that when $Gclause(P)$ is a finite set, $\mathcal{M}_P^{GE}$ can always be determined by considering finitely many equivalence classes of derivation trees.

The above idea will now be presented in a precise way. Let the set of all nodes of a derivation tree T be denoted by $Node(T)$. Let an equivalence relation $\sim$ on $Tree(P)$ be defined as follows: $T_1 \sim T_2$ iff $root(T_1) = root(T_2)$ and $Node(T_1) = Node(T_2)$. As usual, let the equivalence class modulo $\sim$ containing a derivation tree T be denoted by $[T]$, and the quotient set of $Tree(P)$ modulo $\sim$ (*i.e.*, the set of all equivalence classes of $Tree(P)$ modulo $\sim$) be denoted by $Tree(P)/\sim$.

Example 15 Refer to the program P_6 of Example 14 and the derivation trees of P_6 shown in Figure 6. It is readily seen that $[T_1]$ and $[T_2]$ are singletons; $[T_3]$ and $[T_4]$ are infinite sets, which include $\{T_3, T_5\}$ and $\{T_4, T_6\}$, respectively; and, $Tree(P)/\sim$ is the set $\{[T_1], [T_2], [T_3], [T_4]\}$. □

Now, let $AF_\iota(P)/\sim$ denote the argumentation framework $(Tree(P)/\sim, attacks)$, where for any $[T_1], [T_2] \in Tree(P)/\sim$, $[T_1]$ attacks $[T_2]$ iff $root(T_1)$ dominates some clause in $Node(T_2)$. In the sequel, let F and $F_\sim$ denote the characteristic functions of $AF_\iota(P)$ and $AF_\iota(P)/\sim$, respectively. The next proposition follows immediately from the definitions of $AF_\iota(P)$ and $AF_\iota(P)/\sim$.

Proposition 3 *Let* $T \in Tree(P)$, $B \subseteq Tree(P)$ *and* $C = \{[T'] \mid T' \in B\}$. *Then,* $T \in F(B)$, *iff* $[T] \in F_\sim(C)$. □

Corollary 1

1. *If* A *is a fixpoint of* $F_\sim$, *then* $\bigcup_{[T]\in A}[T]$ *is a fixpoint of* F.

2. *If* B *is a fixpoint of* F, *then* $\{[T] \mid T \in B\}$ *is a fixpoint of* $F_\sim$. □

Proof The results follow directly from Proposition 3. ■

Theorem 2 *If A is the least fixpoint of $F_\sim$, then $\bigcup_{[T]\in A}[T]$ is the least fixpoint of F.* $\square$

Proof Let A be the least fixpoint of $F_\sim$ and B the set $\bigcup_{[T]\in A}[T]$. By Result 1 of Corollary 1, B is a fixpoint of F. Suppose that B is not the least fixpoint of F. Then, there exists a fixpoint B' of F such that $B \not\subseteq B'$. Let A' be the set $\{[T'] \mid T' \in B'\}$. By Result 2 of Corollary 1, A' is a fixpoint of $F_\sim$. As $B \not\subseteq B'$, there exists $U \in B$ such that $U \notin B'$. As $U \in B$, $[U]$ belongs to A. As $U \notin F(B')$, it follows from Proposition 3 that $[U] \notin F_\sim(A')$, and, thus, $[U] \notin A'$. So $A \not\subseteq A'$, which is a contradiction. $\blacksquare$

Theorem 2 implies that $\mathcal{M}_P^{\text{GE}}$ can be obtained through the grounded extension of $AF_\iota(P)/\sim$, *i.e.*,

$$\mathcal{M}_P^{\text{GE}} = \{head(root(T)) \mid [T] \in GE_\sim\},$$

where $GE_\sim$ denotes the grounded extension of $AF_\iota(P)/\sim$.

Computing $\mathcal{M}_P^{\text{GE}}$ on $AF_\iota(P)/\sim$ has two important advantages. Firstly, since for each $T \in Tree(P)$, $Node(T) \subseteq Gclause(P)$, checking whether one equivalence class modulo $\sim$ attacks another is linear in the number of clauses in $Gclause(P)$. Secondly, when $Gclause(P)$ is a finite set, the quotient set $Tree(P)/\sim$ is always finite, and can be constructed incrementally as follows. For each unit clause $C \in Gclause(P)$, construct the pair $(C, \{C\})$; and, then, perform the following repetition:

```
Repeat
    For each clause $(a \leftarrow b_1, \ldots, b_m)$, where $m \geq 1$, in $Gclause(P)$
        If pairs $(C_1, S_1), \ldots, (C_m, S_m)$ such that
        $head(C_i) = b_i$, for each $i \in \{1, \ldots, m\}$, have been constructed
        before the current execution of the repeat-loop
        then construct the pair
            $((a \leftarrow b_1, \ldots, b_m), S_1 \cup \cdots \cup S_m \cup \{(a \leftarrow b_1, \ldots, b_m)\})$
until no new pair is constructed
```

It is simple to show that if T is a derivation tree of P, then the pair $(root(T), Node(T))$ is constructed, by induction on the height of T, and, conversely, that if a pair (C, S) is constructed, then there exists a derivation tree T' of P such that $root(T') = C$ and $Node(T') = S$, by induction on the number of times the repeat-loop has been executed when the pair (C, S) is constructed for the first time. Now, for any $[T] \in Tree(P)/\!\sim$, let $[T]$ be represented by the pair $(root(T), Node(T))$. It follows that when $Gclause(P)$ is a finite set, the above procedure always terminates and generates exactly all the pairs representing the equivalence classes of $Tree(P)$ modulo $\sim$.[5]

Once the argumentation framework $AF_t(P)/\!\sim$ is constructed, its grounded extension can be computed using a usual iterative procedure for computing the least fixpoint of a monotonic function.[6] The next subsection discusses an alternative way of reasoning about $\mathcal{M}_P^{ge}$ by considering $AF_t(P)/\!\sim$ as a logic program with negation.

3.5 Meta-Interpreters for Argumentation Systems

Dung demonstrated in his paper (Dung 1995) that argumentation can be viewed as logic programming, and introduced a general method for generating meta-interpreters for argumentation systems. This subsection first summarizes this method and then explains its application to the presented work.

Given an argumentation framework $AF = (AR, attacks)$, let the logic program P_{AF} be defined as the union of two logic programs, AGU_{AF} (argument generation unit) and APU_{AF} (argument processing unit), where

$$AGU_{AF} \quad = \quad \{\mathsf{attacks}(X, Y) \leftarrow \ \mid (X, Y) \in attacks\},$$

and APU_{AF} consists of the following two clauses:

$$\begin{array}{llll} C1: & \mathsf{acceptable}(X) & \leftarrow & \neg\mathsf{defeated}(X) \\ C2: & \mathsf{defeated}(X) & \leftarrow & \mathsf{attacks}(Y, X), \mathsf{acceptable}(Y). \end{array}$$

The clause C_1 means that an argument is acceptable if it is not defeated; and, the clause C_2 means that an argument is defeated if it is attacked by some acceptable argument. P_{AF} is regarded as a meta-interpreter in the sense that it is independent of any particular argument framework; the arguments in AF are considered as

[5] In particular, when no atom is self-dependent with respect to P, the number of times the repeat-loop is executed is bounded by the number of clauses in $Gclause(P)$ (since the height of a derivation tree of P is bounded by this number).

[6] See (for example) the procedure LFP given in (Ceri, Gottlob and Tanca 1990).