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)

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 คอบบี และรอดนีย์ โทเปอร์ ในปี พ.ศ. 2536 [6] และได้ทำการพิสูจน์ว่าในกรณีของโปรแกรมที่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้ (Inheritance-Stratified Program) การกำหนดความหมายทั้งสองแบบนี้จะให้ผลลัพธ์เหมือนกัน ส่วนในกรณีของโปรแกรมที่ไม่สามารถจัดการถ่ายทอดคุณสมบัติเป็นชั้นได้นั้น วิธีการที่คณะผู้วิจัยเสนอขึ้นยังคงให้ผลลัพธ์ที่ถูกต้องสมเหตุสมผล ในขณะที่วิธีของ กิลเลียน คอบบี 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 อย่างประสบความสำเร็จอย่างกว้างขวาง อย่างไรก็ดี นักวิทยาการคอมพิวเตอร์ส่วนใหญ่ยังมีความเห็นว่า ระบบฐานข้อมูลแบบสัมพันธ์ยังคงมีข้อจำกัดอีกหลายประการ ตัวอย่างเช่น ข้อจำกัดในเรื่องของ โครงสร้างข้อมูล (โครงสร้างข้อมูลพื้นฐาน ของระบบฐานข้อมูลแบบความสัมพันธ์เป็นโครงสร้างแบบ record ซึ่งเป็นอุปสรรคในการจัดเก็บข้อมูลที่มีโครงสร้างซับซ้อน) ข้อจำกัดในการค้นหาข้อมูลโดยใช้คำสั่งค้นหาที่มีการอ้างอิงถึงตัวเอง (Recursive Query) อุปสรรคเรื่องของช่องว่างระหว่างภาษาที่ใช้เขียนโปรแกรมประยุกต์กับภาษาที่ใช้ในการหาข้อมูล (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 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 $n \rightarrow \langle a \rangle\ r\ \langle /a \rangle$,

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 of [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: $2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ 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: $2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ 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.:

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

□

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

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

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 ... $attr_m$=val_m/>
2. *simple element*: <*elem_type* $attr_1$=val_1 ... $attr_m$=val_m> val_{m+1} </*elem_type*>
3. *nested element*: <*elem_type* $attr_1$=val_1 ... $attr_m$=val_m> e_1 ... 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

   $\langle elem\_type\ \ attr_1{=}val_1\ \ldots\ attr_k{=}val_k\ \ pvar_1\ \ldots\ pvar_m \rangle\ \langle /elem\_type \rangle$

   is considered to be identical to the empty-form expression

   $\langle elem\_type\ \ attr_1{=}val_1\ \ldots\ attr_k{=}val_k\ \ pvar_1\ \ldots\ pvar_m /\rangle$.

4. The parts

   $\langle elem\_type\ \ attr_1{=}val_1\ \ldots\ attr_k{=}val_k\ \ pvar_1\ \ldots\ pvar_m \rangle$,

   $\langle /elem\_type \rangle$

   and

   $\langle elem\_type\ \ attr_1{=}val_1\ \ldots\ attr_k{=}val_k\ \ pvar_1\ \ldots\ pvar_m /\rangle$

   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 $\theta' = (c_1\ c_3\ c_2\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9) \in \mathcal{S}_\Lambda$. □

4.4 Specialization System for XML Expressions and XML Declarative Description

In the sequel, let $\Gamma_X = (\mathcal{L}_\Lambda, \mathcal{C}_\Lambda, \mathcal{S}_\Lambda, \mu_\Lambda)$. The definitions of $\mathcal{L}_\Lambda$, $\mathcal{C}_\Lambda$, $\mathcal{S}_\Lambda$ and μ_Λ 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 $\Gamma_X = (\mathcal{L}_\Lambda, \mathcal{C}_\Lambda, \mathcal{S}_\Lambda, \mu_\Lambda)$. □

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

1. $\forall s_1, s_2 \in \mathcal{S}_\Lambda, \exists s \in \mathcal{S}_\Lambda : \mu_\Lambda(s) = \mu_\Lambda(s_1) \circ \mu_\Lambda(s_2)$,
2. $\exists s \in \mathcal{S}_\Lambda, \forall a \in \mathcal{L}_\Lambda : \mu_\Lambda(s)(a) = a$,
3. $\mathcal{C}_\Lambda \subset \mathcal{L}_\Lambda$. □

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 $\mathcal{S}_\Lambda$. It follows immediately from the definitions of $\mathcal{S}_\Lambda$ and μ_Λ that there exists $s = (d_1'\ d_2'\ \ldots\ d_m'\ d_1\ d_2\ \ldots\ d_n)$ such that $\mu_\Lambda(s) = \mu_\Lambda(s_1) \circ \mu_\Lambda(s_2)$.
2. Obviously, $\mu_\Lambda(\lambda)(a) = a$, for each $a \in \mathcal{L}_\Lambda$, where λ is the null sequence.
3. By Definition 7, $\mathcal{C}_\Lambda$ is a subset of $\mathcal{L}_\Lambda$. ■

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 $\mathcal{C}_\Lambda$. 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 $\theta \in \mathcal{S}_\Lambda$ 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 $\theta \in \mathcal{S}_\Lambda$ 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 the underlying 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 $C_1 - C_9$:

C_1: <Person ssn="99999" state="NY" gender="Male">
<Name>John Smith</Name>
<BirthYear>1975</BirthYear>
<Parent mother="55555" />
</Person> ← .

C_2: <Person ssn="55555" state="NY" gender="Female">
<Name>Mary Smith</Name>
<BirthYear>1950</BirthYear>
<Parent father="11111" />
</Person> ← .

C_3: <Person ssn="11111" state="NY" gender="Male">
<Name>Tom Black</Name>
<BirthYear>1920</BirthYear>
</Person> ← .

C_4: <State id="NY">
<Name>New York</Name>
</State> ← .

C_5: <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>.

C_6: <Ancestor ancestor=$S:FatherSSN descendent=$S:PersonSSN/>
← <ValidPerson ssn=$S:PersonSSN $P:PersonAttr>
$E:PersonSubelement
<Parent father=$S:FatherSSN $P:ParentAttr/>
</ValidPerson>.

C_7: <Ancestor ancestor=$S:MotherSSN descendent=$S:PersonSSN/>
← <ValidPerson ssn=$S:PersonSSN $P:PersonAttr>
$E:PersonSubelement
<Parent mother=$S:MotherSSN $P:ParentAttr/>
</ValidPerson>.

C_8: <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>.

C_9: <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 $\Delta_X = (\mathcal{A}_X, \mathcal{G}_X, \mathcal{C}_X, \nu_X)$ 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 $\mathcal{A}$ and on sets of XML expressions in $2^{(\mathcal{A} \cup V)}$.

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 *idfefs-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/$\rangle$, id_{xy}, $P\rangle$ 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 $M(P)$, is defined inductively as follows:

1. Given the meaning, $M(Q)$, of a description Q with the height m, a reference $r = \langle g, f, Q \rangle$ is a true reference iff $g \in f(M(Q))$. 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.:

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

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

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

where $\emptyset$ is the empty set, $[T_P]^n(\emptyset) = T_P([T_P]^{n-1}(\emptyset))$ and the mapping T_P: $2^{\mathcal{X}} \to 2^{\mathcal{G}}$ 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.:

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

Intuitively, given a description P, its meaning, $M(P)$, 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 $B_1\theta, B_2\theta, \ldots, B_n\theta$ true with respect to the meaning of P, the expression $H\theta$ will be derived and included in the meaning of P.

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*: $elem\text{-}type$
   - 1.2. *choice-list form*: $(cp_1 \mid \ldots \mid cp_n)$
   - 1.3. *sequence-list form*: $(cp_1 , \ldots , cp_n)$
2. *Qualified content particle*:
   - 2.1. ?-*form*: cp ?
   - 2.2. +-*form*: cp +
   - 2.3. *-*form*: cp *

where – $elem\text{-}type$ 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 $elem\text{-}type$ EMPTY >
2. *simple form*: <!ELEMENT $elem\text{-}type$ (#PCDATA) >
3. *complex form*: <!ELEMENT $elem\text{-}type$ $content\text{-}particle$ >

where – $elem\text{-}type \in N$ specifies the element type being declared
– $content\text{-}particle \in CP$ 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 $elem\text{-}type$ $attr\text{-}name_1$ $attr\text{-}type_1$ $attr\text{-}default_1$
...
$attr\text{-}name_n$ $attr\text{-}type_n$ $attr\text{-}default_n$ >

where – $n \geq 1$,
– $elem\text{-}type \in N$ 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}
$\cup$ {($value_1 \mid \ldots \mid value_m$) | $value_j \in \Sigma^*$ are distinct enumerated values},
– $attr\text{-}default_i \in$ {#REQUIRED, #IMPLIED}
$\cup$ {#FIXED $fixed\text{-}value$ | $fixed\text{-}value \in \Sigma^*$}
$\cup \Sigma^*$.

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)
   ∪ 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 $P_1 = \tau_{CP}(cp, \text{myDTD\_Conference\_1\_2}) = \{C_1, \ldots, C_8\}$. □

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 ∈ $IVAR$, $S:content ∈ $SVAR$, $P:attrList ∈ $PVAR$, $E:subexp ∈ $EVAR$,

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

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

□

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-id* ∈ Σ* 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 $X \subset \mathcal{G}_X$,

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

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

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

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

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

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 $X \in 2^{(\mathcal{A}_X \cup \mathcal{V})}$, 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 $X \in 2^{(\mathcal{A}_X \cup \mathcal{V})}$, 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 $n \geq 1$,

$\tau_A(d)$ 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≤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≤m; k=k+1)
17: Add the reference (<id value= $S:value />, $f_{id,did}$, R) to the body of clause C_{ik}
End-For.
18: Else-If ($type_i$ is IDREF)
Then
19: For (k=1; k≤m; k=k+1)
20: Add the reference (<id value= $S:value />, $f_{idref,did}$, R) to the body of clause C_{ik}
End-For.
21: Else-If ($type_i$ is IDREFS)
Then
22: For (k=1; k≤m; k=k+1)
23: Add the constraint IdrefsSplitUp(<idrefs value= $S:value />, $V:SetOfIds) and the reference ($V:SetOfIds, $f_{idrefs,did}$, 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≤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 $C_{12} - C_{16}$:

C_{12}: <myDTD_Conference_attrList_1 url=$S:value $P:attrList />
← <myDTD_Conference_attrList_2 $P:attrList />,
(<id value= $S:value />, $f_{id, myDTD}$, R).

C_{13}: <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>}).

C_{14}: <myDTD_Conference_attrList_3 chair=$S:value $P:attrList />
← <myDTD_Conference_attrList_4 $P:attrList />,
(<id value= $S:value />, $f_{idref,myDTD}$, R).

C_{15}: <myDTD_Conference_attrList_3 $P:attrList />
← <myDTD_Conference_attrList_4 $P:attrList />.

C_{16}: <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:

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

$$\cup \{\text{<}dtd\_elem\text{-}type\text{\_attrList\_1/>} \leftarrow . \mid \text{<!ELEMENT } elem\text{-}type\ content\text{-}model\text{>} \in dtd,$$

$$\text{<!ATTLIST } elem\text{-}type\ name_1\ type_1\ default_1 \ldots name_n\ type_n\ default_n\text{>} \notin dtd \}.$$

□

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 $C_{17} - C_{25}$:

C_{17}: <myDTD_Name>
<Name $P:attrList>
$S:pcdata
</Name>
</myDTD_Name> ← <myDTD_Name_attrList_1 $P:attrList />.

C_{18}: <myDTD_Name_attrList_1 /> ← .

C_{19}: <myDTD_Organizer>
<Organizer $P:attrList>
$S:pcdata
</Organizer>
</myDTD_Organizer> ← <myDTD_Organizer_attrList_1 $P:attrList />.

C_{20}: <myDTD_Organizer_attrList_1 /> ← .

C_{21}: <myDTD_Sponsor>
<Sponsor $P:attrList>
$S:pcdata
</Sponsor>
</myDTD_Sponsor> ← <myDTD_Sponsor_attrList_1 $P:attrList />.

C_{22}: <myDTD_Sponsor_attrList_1 /> ← .

C_{23}: <myDTD_Person $P:attrList>
<Person $P:attrList>
$S:pcdata
</Person>
</myDTD_Person> ← <myDTD_Person_attrList_1 $P:attrList />.

C_{24}: <myDTD_Person_attrList_1 ssn=$S:value $P:attrList/>
← <myDTD_Person_attrList_2 $P:attrList />,
(<id value=$S:value />, $f_{id,\,\text{myDTD}}$, R).

C_{25}: <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:

$$D: a \leftarrow \quad \langle dtd\_elem\text{-}type \rangle\, x\, \langle /dtd\_elem\text{-}type \rangle. \tag{10}$$

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

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

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 พาณิชย์อีเล็กทรอนิกส์ 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:

$C1$: X:school[medium-of-teaching $\rightarrow$ thai] if X[located-in $\rightarrow$ 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:

$G1$: ait[medium-of-teaching $\rightarrow$ thai] if ait[located-in $\rightarrow$ 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:

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

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

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

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 1995; Dobbie and Topor 1995).

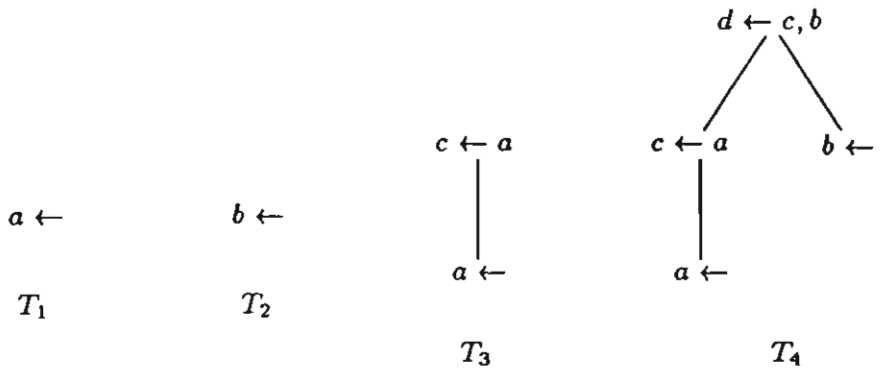

Figure 1: The derivation trees of the program P_1.

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:

$a \leftarrow$

$b \leftarrow$

$c \leftarrow a$

$d \leftarrow c, b$

$f \leftarrow e.$

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 $AF_1(P)$, which

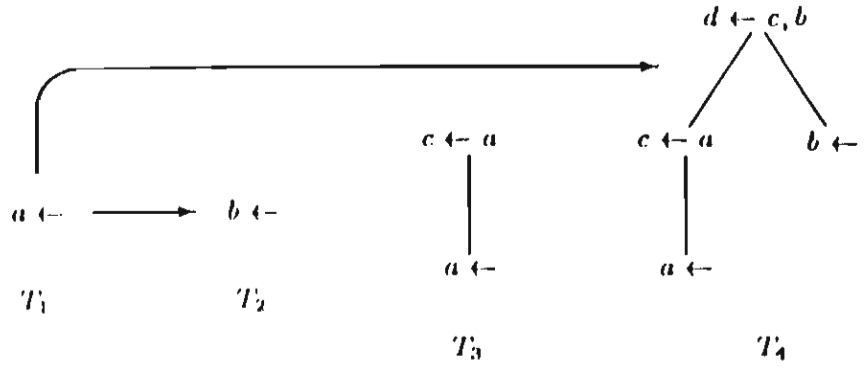

Figure 2. The argumentation framework for the program P_1.

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 $AF_1(P) = (AR, attacks)$ 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 $AF_1(P_1) = (Tree(P_1), attacks)$, where $Tree(P_1)$ consists of the four derivation trees depicted by Figure 1 and $attacks = \{(T_1, T_2), (T_1, T_4)\}$ 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.) □

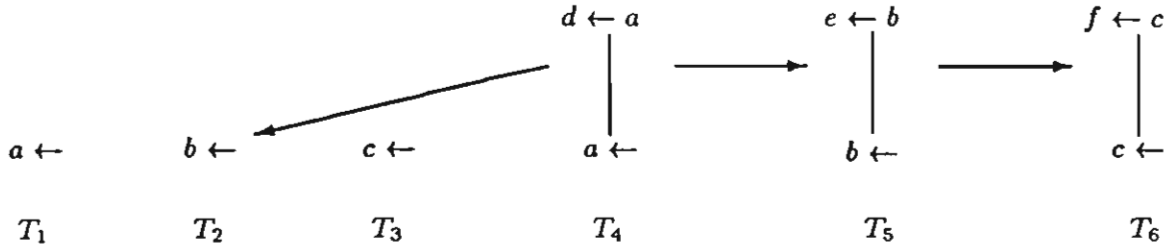

Figure 3: The argumentation framework for the program P_2.

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 $AF_t(P_2) = (Tree(P_2), attacks)$, 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 $AF_t(P)$. Then the *grounded-extension-based meaning* of P, denoted by $\mathcal{M}_P^{\mathsf{GE}}$, is defined by

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

Example 9 Consider $AF_t(P_1)$ 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 $AF_t(P_1)$, and, then, $\mathcal{M}_{P_1}^{\mathsf{GE}} = \{a, c\}$. □

Example 10 Refer to Example 8 (Figure 3). Let F be the characteristic function of $AF_t(P_2)$, 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 $AF_t(P_2)$, whence $\mathcal{M}^{\mathbf{GE}}_{P_2} = \{a, c, d, f\}$. □

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 $AF_t(P_3)$, where *m-t*, *m-e* and *l-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 $AF_t(P_3)$. As $F(\emptyset) = \{T_1, T_3, \} = F^2(\emptyset)$, $F(\emptyset)$ is the grounded extension of $AF_t(P_3)$. Thus $\mathcal{M}^{\mathbf{GE}}_{P_3}$ is the set

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

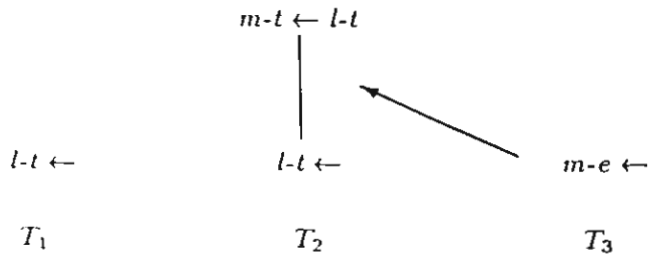

Figure 4 The argumentation framework for the program P_3.

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:

$C1$: X:quaker[policy → pacifist] ←

$C2$: X:republican[policy → hawk] ← .

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

$G1$: nixon[policy → pacifist] ←

$G2$: nixon[policy → hawk] ← .

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 $\mathcal{M}^{GE}_{P_4}$ 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 $\mathcal{M}^{GE}_{P_4}$ now becomes the set {nixon[policy → 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:

$C1$: X:student[residence → north-dorm] ←
X[lives-in → rangsit-campus],
X[sex → male]

$C2$: X:employee[residence → west-apartments] ←
X[lives-in → rangsit-campus],
X[marital-status → married]

$C3$: tom[lives-in → rangsit-campus] ←

$C4$: tom[sex → male] ←

$C5$: tom[marital-status → married] ← .

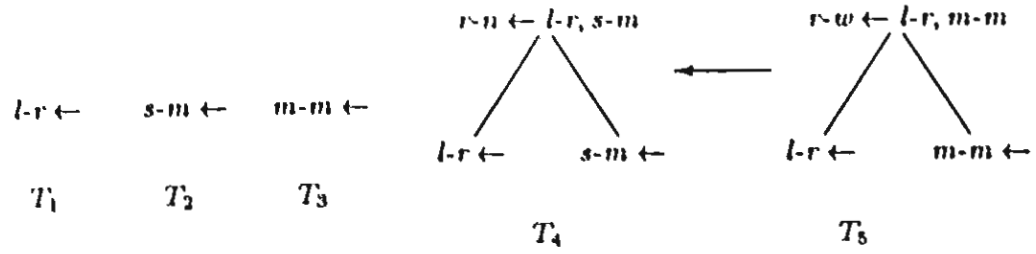

Figure 5: The argumentation framework for the program P_5.

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, $M_{P_5}^{GE}$ 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], $M_{P_5}^{GE}$ contains tom[residence → north-dorm]; and, it still yields the correct meaning of P_5 in this case. □

In general, 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 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 M_P^{GE} is exactly the minimal model of P. ■

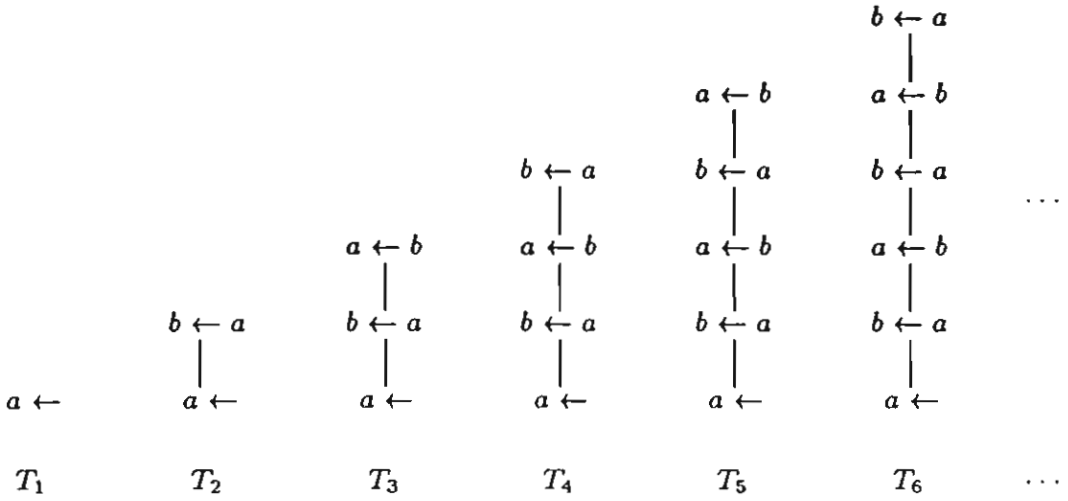

Figure 6: Infinitely many derivation trees of P_6.

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:

$a \leftarrow$

$b \leftarrow a$

$a \leftarrow b.$

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. □

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 $AF_t(P)$ reveals that it is not always necessary to generate all the derivation trees of P so as to determine $\mathcal{M}_P^{\mathbf{GE}}$. 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 $\mathcal{M}_P^{\mathbf{GE}}$. It will be seen in this subsection that when $Gclause(P)$ is a finite set, $\mathcal{M}_P^{\mathbf{GE}}$ 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 $AF_t(P)/\sim$ 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 $AF_t(P)$ and $AF_t(P)/\sim$, respectively. The next proposition follows immediately from the definitions of $AF_t(P)$ and $AF_t(P)/\sim$.

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.* □

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. ■

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

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

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

Computing $\mathcal{M}_P^{\mathrm{GE}}$ on $AF_t(P)/\sim$ 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

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

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

$$C1: \ \mathsf{acceptable}(X) \leftarrow \neg\mathsf{defeated}(X)$$
$$C2: \ \mathsf{defeated}(X) \leftarrow \mathsf{attacks}(Y, X), \mathsf{acceptable}(Y).$$

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 (Gen, Gottlob and Tanca 1990).

distinct elements in the Herbrand universe of P_{AF}. It is shown in (Dung 1995) that:

Theorem 3 *Let AF be an argumentation framework. Then, E is the grounded extension of AF, iff*

$$AGU_{AF} \cup \{\mathsf{acceptable}(X) \mid X \in E\}$$
$$\cup \{\mathsf{defeated}(Y) \mid Y \text{ is attacked by some element of } E\}$$
$$\cup \{\neg\mathsf{defeated}(Z) \mid Z \in E\}$$

is the well-founded model (Van Gelder, Ross and Schlipf 1988; Van Gelder, Ross and Schlipf 1991) of P_{AF}. □

It follows directly from this theorem that if $WFM_{P_{AF}}$ denotes the well-founded model of P_{AF}, then the set

$$\{X \mid \mathsf{acceptable}(X) \in WFM_{P_{AF}}\}$$

is the grounded extension of AF. As a result, given a declarative program P, after the argumentation framework $AF_\iota(P)/\sim$ is constructed, one can generate the logic program $P_{AF_\iota(P)/\sim}$ and then use an evaluation procedure based on the well-founded semantics of logic programs to determine whether an equivalence class of $Tree(P)$ modulo $\sim$ belongs to the grounded extension of $AF_\iota(P)/\sim$, and, thus, whether an atom belongs to $\mathcal{M}_P^{\mathrm{GE}}$.

4 Perfect Model (with Overriding) Semantics

Dobbie and Topor defined a deductive object-oriented language called Gulog (Dobbie and Topor 1993; Dobbie and Topor 1995), in which inheritance is realized through typed substitutions, and studied the interaction of deduction, inheritance and overriding in the context of this language. The declarative semantics for Gulog programs is based on Przymusinski's perfect model semantics for logic programs (Przymusinski 1988), but using the possibility of overriding instead of negation in defining a priority relationship between ground atoms. This perfect model (with overriding) semantics provides the correct meanings for the programs which are inheritance-stratified.

In order to investigate the relationship between the grounded-extension-based semantics and the perfect model (with overriding) semantics, the notions of inheritance stratification and perfect model are reformulated in the framework of DP

theory in this section. The relationship between the two kinds of semantics will be discussed in Section 5.

4.1 Inheritance-Stratified Programs

According to (Dobbie and Topor 1995), a program is inheritance-stratified if there is no cycle in any definition of a method, *i.e.*, a definition of a method does not depend on an inherited definition of the same method. The notion of inheritance stratification is reformulated based on DP theory as follows:

Definition 15 (Inheritance Stratification) A declarative program P on Γ is said to be *inheritance-stratified*, iff it is possible to decompose the interpretation domain $\mathcal{G}$ into disjoint sets, called *strata*, $G_0, G_1, \ldots, G_\gamma, \ldots$, where $\gamma < \delta$ and δ is a countable ordinal, such that the following conditions are all satisfied.

1. For each $C \in Gclause(P)$, if $head(C) \in G_\alpha$, then
   (a) for each $b \in Body(C)$, $b \in \bigcup_{\beta \leq \alpha} G_\beta$,
   (b) for each $C' \in Gclause(P)$ such that C' dominates C,
      i. $head(C') \in \bigcup_{\beta \leq \alpha} G_\beta$,
      ii. for each $b' \in Body(C')$, $b' \in \bigcup_{\beta < \alpha} G_\beta$.
2. There exists no infinite sequence $C_0, C_1, \ldots, C_n, \ldots$ of clauses in $Gclause(P)$ such that for each $i \geq 0$, C_{i+1} dominates C_i.[7]

Any decomposition $\{G_0, G_1, \ldots, G_\gamma, \ldots\}$ of $\mathcal{G}$ satisfying the above conditions is called an *inheritance stratification* of P. □

Two examples of non-inheritance-stratified programs are given in Subsection 5.2.

The next proposition is an important result. It illuminates the coincidence between the grounded extension, preferred extension and stable extension of the argumentation framework for an inheritance-stratified program (see Theorem 1).

Proposition 4 *If P is inheritance-stratified, then $AF_i(P)$ is well-founded.* □

[7] By Condition 1 solely, there may exist an infinite sequence $C_0, C_1, \ldots, C_n, \ldots$ of clauses in $Gclause(P)$ such that for each $i \geq 0$, $head(C_i)$ and $head(C_{i+1})$ belong to the same stratum and C_{i+1} dominates C_i. The nonexistence of such a sequence is required by Proposition 4.

Proof Let P be inheritance-stratified. First observe that, by Conditions 1a and 1(b)i of Definition 15, for any derivation trees T, T' of P, if T attacks T', then the stratum containing $head(root(T))$ can not be higher than the stratum containing $head(root(T'))$. Consequently, since the ordinals are well-founded, it suffices to show that there exists no infinite sequence $T_0, T_1, \ldots, T_n, \ldots$ of derivation trees of P such that for each $i \geq 0$, T_{i+1} attacks T_i and $head(root(T_i))$ and $head(root(T_{i+1}))$ belong to the same stratum.

Suppose that such an infinite sequence exists. It will now be shown that for each $i > 0$, $root(T_{i+1})$ necessarily dominates $root(T_i)$. Assume the contrary, *i.e.*, there exists $j > 0$ such that $root(T_{j+1})$ does not dominate $root(T_j)$. Then $root(T_{j+1})$ dominates some node C of an immediate subtree of T_j. As $root(T_j)$ dominates some node of T_{j-1}, it follows from Conditions 1a and 1(b)ii of Definition 15 that the stratum containing $head(C)$ is strictly lower than the stratum containing $head(root(T_{j-1}))$. Then, by Condition 1(b)i of Definition 15, it is impossible that $head(root(T_{j-1}))$ and $head(root(T_{j+1}))$ belong to the same stratum. This is a contradiction.

As a result, the existence of such an infinite sequence implies that there exists an infinite sequence of clauses $root(T_1), root(T_2), \ldots, root(T_n), \ldots$ such that for each $i \geq 1$, $root(T_{i+1})$ dominates $root(T_i)$, which violates Condition 2 of Definition 15. ■

It follows immediately from Proposition 4 and Theorem 1 that:

Corollary 2 *The grounded extension of $AF_i(P)$ is stable.* □

4.2 Perfect Model (with Overriding) Semantics

With overriding, not every ground clause of a program is expected to be satisfied by a *reasonable* model of that program. More precisely, a ground clause need not be satisfied if it is overridden by some ground clause whose premise is satisfied. This leads to the following notion of a model with overriding:

Definition 16 (Model with Overriding) An interpretation I is a *model with overriding* (for short, *o-model*) of P, iff for each $C \in Gclause(P)$, *at least one* of the following conditions is satisfied:

1. I satisfies C.

2. There exists $C' \in Gclause(P)$ such that C' dominates C and $Body(C') \subseteq I$.

□

Notice that every model of P is also an o-model of P, but not vice versa. However, if P is domination-free, then an o-model of P is also a model of P.

A program may have more than one o-model. Following (Dobbie and Topor 1995), a priority relationship between ground atoms is defined based on the possibility of overriding (Definition 17). This priority relationship will be used to determine a preference relationship between o-models (Definition 18). The meaning of an inheritance-stratified program P is then defined as the o-model of P to which none of other o-models of P is preferable, called its *perfect o-model* and denoted by $\mathcal{M}_P^{\text{Perf}}$. This meaning is uniquely determined for every inheritance-stratified program (Theorem 4).

Definition 17 (Priority Relations $<_p$ and $\leq_p$) Priority relations $<_p$ and $\leq_p$ on $\mathcal{G}$ are defined by the following rules:

1. If $C \in Gclause(P)$, then
   (a) for each $b \in Body(C)$, $head(C) \leq_p b$,
   (b) for each $C' \in Gclause(P)$, if C' dominates C, then
      i. $head(C) \leq_p head(C')$,
      ii. for each $b' \in Body(C')$, $head(C) <_p b'$,
2. If $a \leq_p b$ and $b \leq_p c$, then $a \leq_p c$,
3. If $a \leq_p b$ and $b <_p c$ (respectively, $d <_p a$), then $a <_p c$ (respectively, $d <_p b$),
4. If $a <_p b$, then $a \leq_p b$,
5. Nothing else satisfies $<_p$ or $\leq_p$. □

Definition 18 (Preference Relation $\ll$ and Perfect O-Model) Let M and N be o-models of P. M is said to be *preferable* to N, in symbols, $M \ll N$, iff $M \neq N$ and for each $a \in M - N$, there exists $b \in N - M$ such that $a <_p b$. M is said to be a *perfect o-model* of P, iff there exists no o-model of P preferable to M. □

The following results are analogous to and inspired by the corresponding results for inheritance-stratified Gulog programs presented in (Dobbie and Topor 1993;

Dobbie 1994; Dobbie and Topor 1995). Their proofs, which are given completely in (Nantajeewarawat 1997), are guided partly by (Przymusinski 1988) and (Dobbie 1994).

Lemma 2 *If P is a domination-free program, then the minimal model of P is the unique perfect o-model of P.* □

Theorem 4 *Every inheritance-stratified program P has exactly one perfect o-model, which will be denoted by $\mathcal{M}_P^{\text{Perf}}$, and for every other o-model N of P, $\mathcal{M}_P^{\text{Perf}} \ll N$.* □

5 Relationship between the Proposed Semantics and Perfect Model Semantics

This section first shows that for inheritance-stratified programs, the grounded-extension-based semantics and the perfect model (with overriding) semantics coincide with each other (Subsection 5.1). Then, it uses two simple examples to show that the grounded-extension-based semantics also provides non-inheritance-stratified programs with their correct skeptical meanings, whereas the perfect model semantics fails to provide sensible meanings for them (Subsection 5.2).

5.1 Coincidence between the Two Kinds of Semantics

Throughout this subsection, let $\{G_0, \ldots, G_\gamma, \ldots\}$, where $\gamma < \delta$, be an inheritance stratification of P, and GE be the grounded extension of $AF_\iota(P)$, and assume that the domination relation associated with P is transitive. It is important to note that this transitivity requirement does not weaken the results of this subsection, because the domination due to overriding is typically transitive. Also note that, though the domination relation is transitive, the attack relation of $AF_\iota(P)$ is not necessarily transitive.

Lemma 3 *Let $T \in GE$. Then every subtree of T belongs to GE.* □

Proof Assume the contrary, *i.e.*, there exists a subtree T' of T such that $T' \notin GE$. As GE is stable (by Corollary 2), GE attacks T'. Then, obviously, GE also attacks T, which is a contradiction. ■

Lemma 4 *Let $T \in GE$. Then there exists no clause C such that C dominates $root(T)$ and $Body(C) \subseteq \mathcal{M}_P^{GS}$.* □

Proof Assume the contrary, i.e., there exists a clause C such that C dominates $root(T)$ and $Body(C) \subseteq \mathcal{M}_P^{GS}$, and let $Body(C) = \{b_1, \ldots, b_n\}$, where $n \geq 0$. Then there exist $T_1, \ldots, T_n \in GE$ such that $head(root(T_i)) = b_i$ for each $i \in \{1, \ldots, n\}$, and, then, there also exists a derivation tree T' of P such that $root(T') = C$ and the immediate subtrees of T' are exactly the T_i. Since T' attacks T, GE attacks T'. As GE is conflict-free, GE attacks none of the T_i. Therefore there exists $T'' \in GE$ such that $root(T'')$ dominates C. Since *dominates* is assumed to be transitive, $root(T'')$ dominates $root(T)$. So T'' attacks T, contradicting the fact that GE is conflict-free. ■

Theorem 5 *$\mathcal{M}_P^{GS}$ is a perfect o-model of P.* □

Proof Note first that, by Lemma 3, if $T \in GE$, then $Body(root(T)) \subseteq \mathcal{M}_P^{GS}$. It will now be shown that $\mathcal{M}_P^{GS}$ is an o-model of P. Let $C \in P$ such that $Body(C) \subseteq \mathcal{M}_P^{GS}$ and let there exist no clause C' such that C' dominates C and $Body(C') \subseteq \mathcal{M}_P^{GS}$. It follows that, for any $T \in GE$, $root(T)$ does not dominate C. Now let $Body(C) = \{b_1, \ldots, b_m\}$, where $m \geq 0$. Then there exist $T_1, \ldots, T_m \in GE$ such that $head(root(T_i)) = b_i$ for each $i \in \{1, \ldots, m\}$. Thus there exists a derivation tree T of P of which the root is C and the immediate subtrees are exactly the T_i. As GE is conflict-free, GE attacks none of the T_i. Consequently, GE does not attack T. Then, since GE is stable, by Corollary 2, T belongs to GE. Hence $head(C) \in \mathcal{M}_P^{GS}$.

Next, it will be shown that $\mathcal{M}_P^{GS}$ is a perfect o-model of P. Let N be an o-model of P such that $N = \mathcal{M}_P^{GS}$, and let $T \in GE$. By Lemma 4, N necessarily satisfies $root(T)$. It can be shown by induction on $height(T)$ that $head(root(T))$ belongs to N.

Base $height(T) = 0$. Then $root(T)$ is a unit clause, and, thus, $head(root(T)) \in N$.

Induction $height(T) > 0$. Since all immediate subtrees of T belong to GE by Lemma 3, it follows from the induction hypothesis that $Body(root(T)) \subseteq N$. So $head(root(T)) \in N$.

As a result, $\mathcal{M}_P^{GS} \subseteq N$, whence $N \not\ll \mathcal{M}_P^{GS}$. ■

The main result of this section is:

Theorem 6 $\mathcal{M}_P^{\mathrm{OE}} = \mathcal{M}_P^{\mathrm{Perf}}$. □

Proof The result follows immediately from Theorems 4 and 5. ■

5.2 Generality of the Proposed Semantics

One approach to dealing with conflicts caused by inheritance is to discard all conflicting definitions. This approach is called the *skeptical* approach. The next two examples show that for programs which are not inheritance-stratified, the proposed semantics still provides their correct skeptical meanings.

Example 16 Let P_7 be a declarative program which consists of three ground clauses:

$$a \leftarrow$$
$$b \leftarrow a$$
$$c \leftarrow b.$$

Assume that $c \leftarrow b$ dominates $b \leftarrow a$ and for any other two clauses in P_7, one does not dominate the other. According to Definition 15, any inheritance stratification of P_7 requires b to belong to a stratum which is lower than the stratum containing b, which is impossible. So P_7 is not inheritance-stratified. Note that here the dominating clause $c \leftarrow b$ depends solely on the dominated clause $b \leftarrow a$; thus, it is unsound to use any of them. As a result, neither b nor c should be derived. However, according to Definition 16, it is not difficult to see that every o-model of P_7 must contain a, b and c; and, therefore, no o-model of P_7 provides its sensible meaning.

Now consider the grounded-extension-based meaning of P_7. The argumentation framework $AF_i(P_7)$ is delineated in Figure 7. Note that, in the figure, T_3 attacks itself. Let F be the characteristic function of $AF_i(P_7)$. Then, as $F(\emptyset) = \{T_1\} = F(F(\emptyset))$, $\mathcal{M}_{P_7}^{\mathrm{OE}}$ is the set $\{a\}$, which is a correct skeptical meaning of P_7. □

Example 17 Let tom be an instance of type gr(aduate)-student and gr-student is

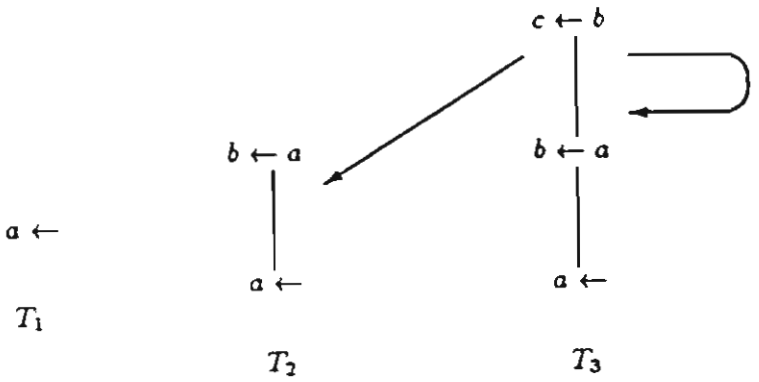

Figure 7: The argumentation framework for the program P_7.

a subtype of student. Consider the following declarative program P_8:

X: student[math-ability → good] ← X[math-grade → b]

X: student[major → math] ← X[math-ability → good],
X[favourite-subject → math]

X: gr-student[math-ability → average] ← X[major → math],
X[math-grade → b]

tom[math-grade → b] ←

tom[favourite-subject → math] ← .

For the sake of simplicity, suppose that P_8 has only five ground clauses:

$G1$: tom[math-ability → good] ← tom[math-grade → b]

$G2$: tom[major → math] ← tom[math-ability → good],
tom[favourite-subject → math]

$G3$: tom[math-ability → average] ← tom[major → math],
tom[math-grade → b]

$G4$: tom[math-grade → b] ←

$G5$: tom[favourite-subject → math] ← .

The ground clauses $G1$ and $G3$ are considered as definitions of the method math-ability taken from the types student and gr-student, respectively. As gr-student is more specific than student, $G3$ is expected to dominate $G1$. Then, every inheritance stratification of P_8 requires that the ground atom tom[major → math] must be in a stratum which is lower than the stratum containing it, which is a contradiction. Hence P_8 is not inheritance-stratified.

Observe that $G3$ dominates $G1$, but $G3$ also depends on $G1$; more precisely, here, the activation of $G1$ results in the activation of $G3$, which is supposed to override

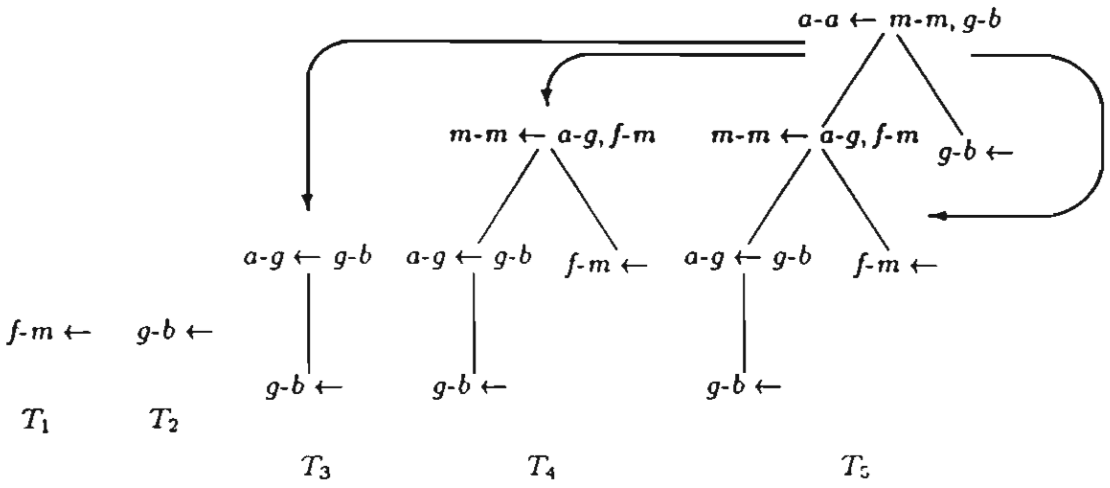

Figure 8: The argumentation framework for the program P_8.

$G1$. Therefore, it is not reasonable to use any of them. As a consequence, none of the ground atoms tom[math-ability → good], tom[major → math] and tom[math-ability → average] should be derived. However, it can be shown that each o-model of P_8 contains both tom[major → math] and tom[math-ability → average]. So every o-model of P_8 does not serve as its reasonable meaning.

Now consider the proposed semantics. The argumentation framework $AF_i(P_8)$ is depicted by Figure 8, where a-g, a-a, m-m, g-b and f-m denote the ground atoms tom[math-ability → good], tom[math-ability → average], tom[major → math], tom[math-grade → b] and tom[favourite-subject → math], respectively. Note that, in Figure 8, T_5 attacks itself. It is simple to see that $\mathcal{M}^{o\varepsilon}_{P_8}$ is the set

{tom[math-grade → b], tom[favourite-subject → math]},

which is the correct skeptical meaning of P_8 (*i.e.*, the meaning obtained in the usual way after discarding the conflicting clauses $G1$ and $G3$). □

6 Comparisons with Works on Inheritance Networks

Nonmonotonic inheritance has been studied intensively in the context of inheritance networks (Touretzky 1986; Horty, Thomason and Touretzky 1990; Stein 1992). An inheritance network is a directed acyclic graph with positive and negative edges. A

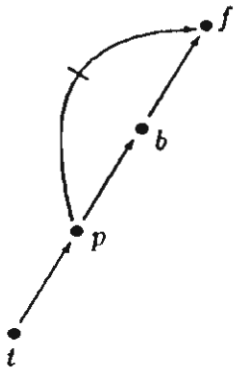

Figure 9: An example of inheritance networks.

vertex in the network represents an object (individual) or a kind of object. Positive and negative edges are intended to denote "is-a" and "is-not-a", respectively. A positive path from a vertex a to a vertex x, *i.e.*, a sequence of positive edges $(a, s_1), (s_1, s_2), \ldots, (s_{n-1}, s_n), (s_n, x)$, where $n \geq 0$, supports the inference "a is an x". On the other hand, a negative path from a to x, *i.e.*, a sequence of positive edges $(a, s_1), (s_1, s_2), \ldots, (s_{m-1}, s_m)$ followed by a single negative edge from s_m to x, where $m \geq 0$, supports the inference "a is not an x". When the network contains paths that support conflicting conclusions, the topological properties of the network will be employed to resolve the conflicts based on the principle that more specific information is more directly relevant. For example, consider the inheritance network in Figure 9. Let the vertices t, p, b and f denote "Tweety", "penguin", "bird" and "flying thing", respectively. This network then contains the information that Tweety is a penguin, that penguins are birds, that birds fly, and that penguins do not fly. The positive path from t (through p and b) to f enables the conclusion that Tweety flies, while the negative path from t (through p) to f supports the opposite conclusion. In terms of the topology of this network, since there is a path from t through p to b, it is natural to suppose that p provides more specific information about t than b does. The positive path from t to f is therefore considered to be preempted, and an inheritance reasoner infers that Tweety does not fly.

In contrast with the works on inheritance networks, this paper assumes that a hierarchy of types, partially ordered by the subtype relation (in other words, partially ordered by the inclusion relation on the extensions of the types), is given. The hierarchy itself does not contain any conflicting information, *i.e.*, a type either is or is not a (direct or indirect) subtype of another type, but not both. Method

definitions (possibly conditional), expressed as definite clauses, are associated with a type, and are inherited by an individual of the type. From a consistent type hierarchy, conflicts between method definitions inherited form different types may arise and can be resolved based on a specified domination relation on ground clauses.

In some cases, by using an appropriate kind of inheritance reasoner, method definitions can be encoded in an inheritance network. For example, the two definitions of the method medium-of-teaching in Example 11 can be represented by the network in Figure 10, where a, i, l_t, m_t and m_e denote "AIT", "international school", "school that is located in Thailand", "school at which the medium of teaching is Thai" and "school at which the medium of teaching is English", respectively. As there exists an uncontested path from a to m_e, a skeptical inheritance reasoner will infer from this network that the medium of teaching at AIT is English.

However, it is pointed out by Horty, Thomason and Touretzky (1990) that:

> Of course, the process of drawing conclusions from a set of defeasible hypotheses through inheritance reasoning is quite different from the process of drawing conclusions from (the set) through deduction. Inheritance reasoning doesn't depend on the interplay of connectives, for example, since there aren't really any connectives, to speak out, in our semantic nets ... (Horty, Thomason and Touretzky 1990)

The edges in an inheritance network are not connectives, since they apply to individuals and kinds rather than sentences. It is therefore not always possible to represent a method definition expressed by a definite clause by an inheritance network. For example, consider the definite clause defining the method math-ability for

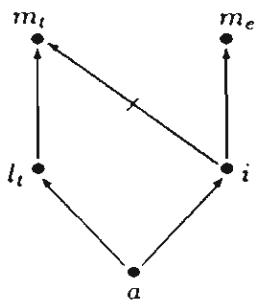

Figure 10: The definitions of medium-of-teaching represented by an inheritance network.

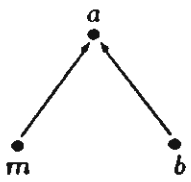

Figure 11: An unsatisfactory representation of the clause $a \leftarrow m, b$.

the individuals of type gr-student:

$$\mathsf{X{:}\,gr\text{-}student[math\text{-}ability \rightarrow average]} \leftarrow \mathsf{X[major \rightarrow math]},\ \mathsf{X[math\text{-}grade \rightarrow b]},$$

given in Example 17. The antecedent of this clause is a conjunction of two atoms and the clause cannot be represented satisfactorily by the part of an inheritance network shown in Figure 11, where a, b and m denote "graduate student whose mathematical ability is average", "graduate student whose mathematics grade is B" and "graduate student whose major is mathematics", respectively. (This part of the network merely states that a graduate student whose mathematics grade is B has average mathematical ability, and that a graduate student whose major is mathematics has average mathematical ability.) Furthermore, as conditional method definitions cannot, in general, be represented, dynamic method resolution is not discussed in the context of inheritance networks.

Notwithstanding, the works on inheritance networks provide the presented approach with a foundation for determining the domination relation among ground method definitions. The hierarchy of types together with the membership relation associating individuals with their types can be represented as a network, and the domination relation can then be determined based on the topological information of the network. For example, if there exists a path from an individual a through a type t to a type t' in the network, then the method definitions for the individual a inherited from the type t can reasonably be considered to dominate those inherited from the type t'.

7 Conclusions

A framework for discussing a declarative semantics for declarative programs with defeasible inheritance, based on Dung's argumentation framework (Dung 1995),

is proposed. The framework requires a domination relation on program ground clauses, specifying their priority, to be explicitly given as additional information. In practice, when a hierarchy of types is given, a suitable domination relation with respect to method overriding can be determined by syntactic examination of a program. With a specified domination relation, a program is transformed into an argumentation framework which provides an appropriate structure for analyzing the dynamic interaction of the intended deduction and domination. The meaning of the program is defined based on the grounded extension of this argumentation framework. This paper not only shows that the proposed semantics and Dobbie and Topor's perfect model (with overriding) semantics coincide for inheritance-stratified programs (Theorem 6), but also claims that the proposed semantics provides correct skeptical meanings for non-inheritance-stratified programs.

Acknowledgement

The proposed semantics is inspired by Phan Minh Dung. This work is supported in part by the Thailand Research Fund.

References

ABITEBOUL, S., G. LAUSEN, H. UPHOFF, and E. WALLER. 1993. Methods and Rules. Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data, pages 32–41, Washington, DC, May. ACM Press.

AÏT-KACI, H., and R. NASR. 1986. LOGIN: A Logic Programming Language with Built-in Inheritance. The Journal of Logic Programming, 3:185–215.

AÏT-KACI, H., and A. PODELSKI. 1993. Towards a Meaning of Life. The Journal of Logic Programming, 16:195–234.

AKAMA, K. 1993. Declarative Semantics of Logic Programs on Parameterized Representation Systems. Advances in Software Science and Technology, 5:45–63.

CERI, S., G. GOTTLOB, and L. TANCA. 1990. Logic Programming and Databases. Springer-Verlag.

DOBBIE, G. 1994. Foundations of Deductive Object-Oriented Database Sys-

tems. PhD thesis, Department of Computer Science, The University of Melbourne, Parkville 3052, Australia.

DOBBIE, G., and R. TOPOR. 1993. A Model for Sets and Multiple Inheritance in Deductive Object-Oriented Systems. In S. Ceri, K. Tanaka, and S. Tsur, editors, Proceedings of the Third International Conference on Deductive and Object-Oriented Databases (DOOD'93), pages 473–488, Phoenix, Arizona, December. Volume 760 of Lecture Notes in Computer Science. Springer-Verlag.

DOBBIE, G., and R. TOPOR. 1995. On the Declarative and Procedural Semantics of Deductive Object-Oriented Systems. Journal of Intelligent Information Systems, 4:193–219.

DUNG, P. M. 1995. On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and N-Person Games. Artificial Intelligence, 77:321–357.

DUNG, P. M., and T. C. SON. 1995. Nonmonotonic Inheritance, Argumentation and Logic Programming. In V. W. Marek and A. Nerode, editors, Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'95), pages 316–329, Lexington, KY, June. Volume 928 of Lecture Notes in Computer Science. Springer-Verlag.

HORTY, J. F., R. H. THOMASON, and D. S. TOURETZKY. 1990. A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. Artificial Intelligence, 42:311–348.

KIFER, M., G. LAUSEN, and J. WU. 1995. Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the Association for Computing Machinery, 42:741–843.

LLOYD, J. W. 1987. Foundations of Logic Programming, Second, Extended Edition. Springer-Verlag.

NANTAJEEWARAWAT, E. 1997. An Axiomatic Framework for Deductive Object-Oriented Representation Systems Based-on Declarative Program Theory. PhD thesis, CS-97-7, Computer Science and Information Management Program, School of Advanced Technologies, Asian Institute of Technology, Bangkok, Thailand.

PRZYMUSINSKI, T. C. 1988. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker, editor, Foundations of Deductive Databases and Logic Programming, pages 193–216. Morgan Kaufmann, Los Altos, CA.

STEIN, L. A. 1992. Resolving Ambiguity in Nonmonotonic Inheritance Hierarchies. Artificial Intelligence, 55:259–310.

THIRUNARAYAN, K., and M. KIFER. 1993. A Theory of Nonmonotonic Inheritance Based on Annotated Logic. Artificial Intelligence, 60:23–50.

TOURETZKY, D. S. 1986. The Mathematics of Inheritance. Morgan Kaufmann, Los Altos, CA.

VAN GELDER, A., K. ROSS, and J. SCHLIPF. 1988. Unfounded Set and Well-Founded Semantics for General Logic Programs. Proceedings of the Seventh ACM Symposium on Principles of Database Systems, pages 221-230. ACM Press.

VAN GELDER, A., K. ROSS, and J. SCHLIPF. 1991. Well-Founded Semantics for General Logic Programs. Journal of the Association for Computing Machinery, 38:620–650.

Output #2
Submitted to IEEE Trans. Knowledge and Data Engineering
(1st revision)

Regular Paper

Declarative Programs with Implicit Implication*

VILAS WUWONGSE
E-mail: *vw@cs.ait.ac.th*
Computer Science & Information Management Program
School of Advanced Technologies
Asian Institute of Technology
Pathumthani 12120, Thailand

EKAWIT NANTAJEEWARAWAT
E-mail: *ekawit@siit.tu.ac.th*
Department of Information Technology
Sirindhorn International Institute of Technology
Thammasat University
Pathumthani 12121, Thailand

Index Terms— Declarative program, implicit implication, subsumption, taxonomy, deductive object-oriented database, model-theoretic semantics, fixpoint semantics

*This paper is a substantially revised and extended version of [35].

Abstract— In the presence of taxonomic information, there often exists implicit implication among atoms in an interpretation domain. A general framework is proposed for the discussion of an appropriate semantics for declarative programs with respect to such implicit implication. It is first assumed that the implicit implication can be predetermined and represented by a preorder on the interpretation domain. Under the consequent constraint that every interpretation must conform to the implicit implication, an appropriate model-theoretic semantics as well as its corresponding fixpoint semantics for declarative programs is described. Based on Köstler *et. al.*'s foundation of fixpoint with subsumption, it is shown that, if the implicit-implication relation is, in addition, assumed to be a partial order, then the meaning of a program can be determined more efficiently by application of an immediate-consequence operator which involves only reduced representations, basically consisting only of their maximal elements, of subsets of the interpretation domain.

Index terms— Declarative program, implicit implication, subsumption, taxonomy, deductive object-oriented database, model-theoretic semantics, fixpoint semantics

1 Introduction

Ontological categories of entities constitute an important part of knowledge. In order to organize and simplify a knowledge base, the categories of things that exist in the domain of interest are commonly arranged into taxonomic hierarchies according to levels of generality. The idea of such arrangement dates from ancient times [26, 28]; and generalization taxonomies of categories have been constructed as parts of several modern knowledge-based systems, *e.g.*, [16, 20, 30]. A general category covers a number of specialized categories sharing some similarities. The intensional description of a general category captures the commonalities, but suppresses the differences in the intensional descriptions of more specific categories [5, 29]. Categories are also called classes, collections, concepts, kinds, types,

sorts, and concept types. The selection of categories determines the vocabulary used for representing knowledge. In addition to containing facts about individual objects and their interactions, a knowledge base usually contains general statements concerning categories; and much of reasoning takes place at the level of categories [26, 31].

Logic as well as ontology is an essential foundation of virtually every knowledge representation scheme. Logic provides forms of sentences, interpretation structures for specifying the meanings of sentences, and rules of inferences. In logic-based deductive systems with taxonomic information, such as deductive object-oriented systems, atomic formulae (atoms) in an interpretation domain, which serve as basic sentences for describing objects, are usually interrelated semantically, *i.e.*, some atom may implicitly imply others, based on their structures, their intended meanings and class/subclass information. In particular, in a system which separates taxonomic schema declarations from data definitions[1], such an interrelation can, in general, be predetermined. For example, in a conceptual graph language [13, 15, 34], if generalization lattices of concept types, relation types and markers are provided, irredundant atomic conceptual graphs[2] can be partially ordered into a generalization hierarchy in which a graph logically implies each of its more general graphs [25, 27]. Similarly, in description logics [7, 8, 6, 22, 32], which are descendants of the KL-ONE language [9], subsumption relationships among structured descriptions, where a subsumee entails its subsumers, can be derived automatically from their structures with respect to a given generalization taxonomy of primitive descriptions. This kind of implication is implicit in the sense that it does not need to be declared in the assertional parts of knowledge bases, but is embodied in the systems' reasoning apparatuses.

[1] In such a system, taxonomic information (*e.g.*, class/subclass relation and class population) is treated as part of the system schema, and is not defined by program clauses.

[2] A conceptual graph is *redundant* if it is logically equivalent to some of its proper subgraphs. It is *atomic* if it does not contain any context as its concept node [15].

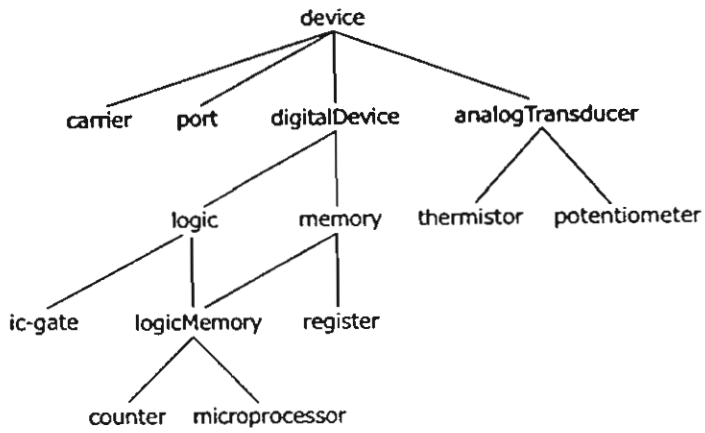

Figure 1: A partial hierarchy of concept types for digital systems

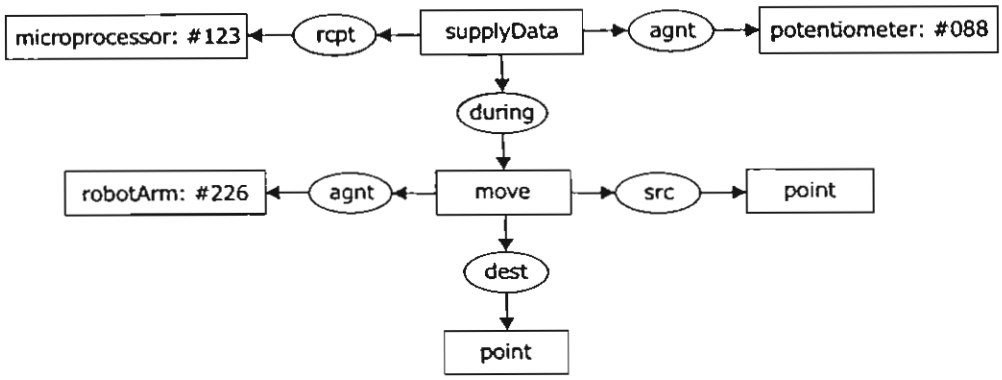

Figure 2: A conceptual graph $G1$

1.1 A Motivating Example

Figure 1 illustrates a partial generalization hierarchy of concept types in the domain of digital system specifications and requirements, which is inspired by [10, 11]. The type device encompasses all hardwar elements. The types logic and memory embrace devices that contain logic for data manipulation and devices that contain memory for storage of values, respectively; and their common subtype, logicMemory, represents devices that are both of type logic and of type memory. The type analogTransducer covers devices that convert

Figure 3: A conceptual graph $G2$

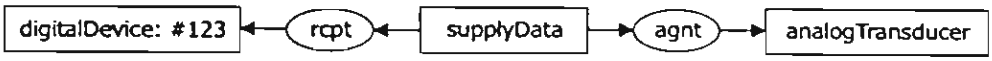

Figure 4: A conceptual graph $G3$

physical quantities into electrical analog quantities. The universal type, $\top$, which is the most general concept type, and the absurd type, $\bot$, which is a subtype of every concept type, are not shown in the figure.

Now consider the atomic conceptual graph $G1$ in Figure 2, which is intended to mean "the potentiometer #088 supplies data to the microprocessor #123 when the robot arm #226 moves from one point to another". This graph is more specific than and hence implicitly implies, for instance, the graph $G2$ in Figure 3, which states "the potentiometer #088 supplies data to the microprocessor #123". According to the hierarchy in Figure 1, since microprocessor and potentiometer are subtypes of digitalDevice and analogTransducer, respectively, the graph $G2$ in turn implicitly implies the graph $G3$ in Figure 4, which is intended to mean "an analog transducer supplies data to the digital device #123". This kind of implicit implication between conceptual graphs can be determined by examining their syntactic structures and components.

Then, consider a conceptual graph program which contains as its program clauses the graph C in Figure 5 and the graph $G1$. The intended meaning of the conceptual graph C is "if an analog transducer supplies data to a digital device, then an A/D converter is interfaced to the digital device". Since $G1$ implicitly implies $G3$, which satisfies the antecedent of C, the clause C fires and thus yields the graph $G4$ in Figure 6, the intended meaning of which is "an A/D converter is interfaced to the digital device #123", as derived information.

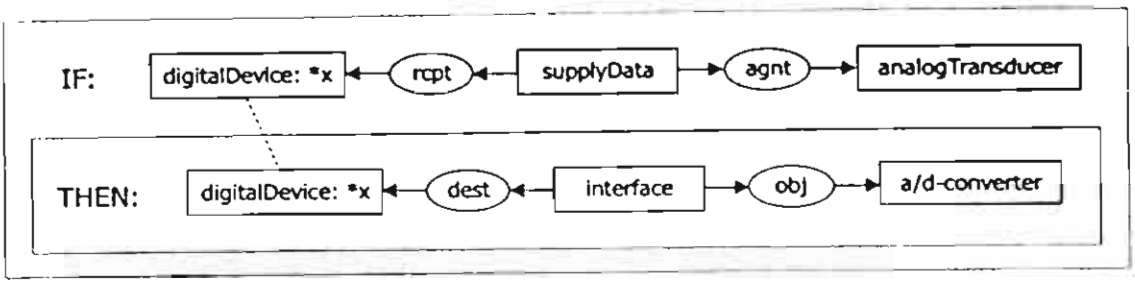

Figure 5: A conceptual graph C

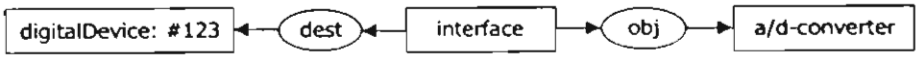

Figure 6: A conceptual graph $G4$

1.2 The Presented Work

The effect of such implicit implication on the declarative meanings of assertional knowledge bases expressed as logic-programming-style definite programs is studied in this paper. It is first assumed that there exists predetermined implicit implication among atoms in an interpretation domain and this implicit implication can be described by a binary relation on the domain. By the characteristics of implication, such a relation is typically a preorder (quasi-order), *i.e.*, it is reflexive and transitive. Under this assumption, an interpretation must be closed with respect to the preorder. An appropriate model-theoretic semantics for declarative programs together with its corresponding fixpoint semantics is developed accordingly.

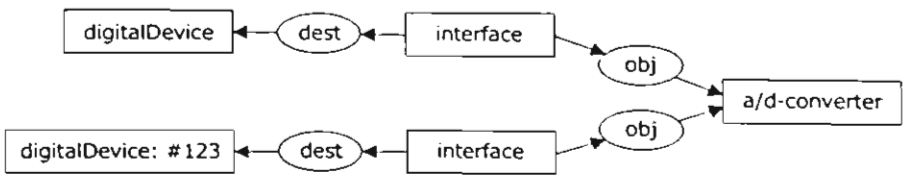

Figure 7: A redundant conceptual graph $G5$

Next, a stronger assumption is considered, namely that the implicit-implication relation is a partial order (antisymmetric preorder). To illustrate the practicality of this assumption, consider the implicit implication on atomic conceptual graphs. This implicit implication is, in general, a preorder but not a partial order [12, 25]; *e.g.*, the conceptual graph $G4$ in Figure 6 and the redundant conceptual graph $G5$ in Figure 7 implicitly imply each other, and, thus, the implicit implication is not antisymmetric. However, when consideration is only given to irredundant conceptual graphs, which are actually used in practical applications, the implicit implication is a partial order [25]. Under this stronger assumption, a legitimate interpretation can be represented equivalently by its reduced version which, intuitively, consists only of its maximal elements with respect to the order. In addition, based on the foundation of fixpoint iteration with subsumption [18, 19], the reduced representation of the meaning of a program can be directly computed by an immediate-consequence operator on a quotient set of the reduced interpretations.

For the sake of simplicity and generality, this paper uses as its primary logical basis Akama's axiomatic theory of logic programs [3], *i.e.*, DP (declarative programs) theory. Section 2 recalls some basic definitions and results of DP theory. Section 3 discusses the model-theoretic semantics together with the fixpoint semantics of a declarative program under the preorder assumption. Section 4 recalls certain definitions and results related to subsumption ordering [18, 19] and describes a more elegant fixpoint semantics under the stronger assumption of partial order. Results concerning the continuous operators on complete lattices, used in this paper, are given in the appendix.

2 DP Theory

Akama's DP theory [3] is an axiomatic theory which purports to generalize the concept of conventional logic programs to cover a wider variety of data domains. The theory sup-

presses the differences in the forms of (extended) atoms in various logic-programming-style knowledge representation languages, and captures the common interrelations between atoms and substitutions by a mathematical abstraction, called a specialization system. Despite its simplicity, the specialization system provides a sufficient structure for defining declarative programs together with their declarative meanings.

DP theory provides a template for developing a declarative semantics for declarative programs constructed out of atoms in any specific data domain. For example, in [4, 33], after a concrete specialization system for RDF/XML elements is formulated, all the results of DP theory can be employed to determine the meanings of RDF/XML declarative programs. Likewise, as will be seen in Subsection 2.4, by formulating an appropriate specialization system for atomic conceptual graphs, DP theory provides a framework for discussing the meanings of definite conceptual graph programs. In addition to program semantics, declarative programs on such specific domains can inherit properties or findings related to DP theory including the ones presented in this paper. Therefore, it is often more advantageous to work on DP theory than to work on some specific declarative program framework.

2.1 Specialization Systems and Declarative Programs

The concepts of specialization system and declarative program on a specialization system are reviewed first.

Definition 1 [3] **(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 $\mathcal{G}$ *interpretation domain*, the elements of $\mathcal{S}$ *specialization parameters* or simply *specializations*, and the mapping μ *specialization operator*. A specialization $s \in \mathcal{S}$ is said to be *applicable* to $a \in \mathcal{A}$, iff $a \in dom(\mu s)$. □

Throughout this section, 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. In the absence 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.*, $(\mu\theta)a = a\theta$.

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 [3] **(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:

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

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 *unit clause*. The set of all definite clauses on X is denoted by $Dclause(X)$. A *declarative program* on Γ is a (possibly infinite) subset of $Dclause(\mathcal{A})$. □

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 $C' = (a\theta \leftarrow b_1\theta, \ldots, b_n\theta)$. Denote by $C\theta$ such an instance C' of C and by $Instance(C)$ the set of all instances of C. Given a declarative program P on Γ, denote by $Gclause(P)$ 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 $\mathcal{G}$.

2.2 Model-Theoretic Semantics

For a discussion of the model-theoretic semantics of a declarative program on Γ, an interpretation assigning truth values to atoms in the interpretation domain $\mathcal{G}$, the truth value of a definite clause on $\mathcal{A}$ with respect to a particular interpretation, and a model of a declarative program on Γ are given by:

Definition 3 [3] **(Interpretation)** An *interpretation* is a subset of $\mathcal{G}$. A definite clause C on $\mathcal{A}$ is true with respect to an interpretation I, iff

$$(\forall C' \in Instance(C) \cap Dclause(\mathcal{G})) : \ ((head(C') \in I) \text{ or } (Body(C') \not\subseteq I)). \quad \square$$

Definition 4 [3] **(Model)** 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. $\square$

As in the conventional theory, the model intersection property also holds for declarative programs on Γ, and the semantics of a declarative program P on Γ is defined as the intersection of all models of P, called the minimal model of P and denoted by $\mathcal{M}_P$.

Proposition 1 [3] **(Model Intersection Property)** *The intersection of more than one model of a declarative program P on Γ is also a model of P.* $\square$

Theorem 1 [3] *Every declarative program P on Γ has the minimal model $\mathcal{M}_P$, which is the intersection of all models of P.* $\square$

2.3 Fixpoint Semantics

Throughout this subsection, let P be a declarative program on Γ. Associated with P are the mappings T_P and K_P on the complete lattice $(2^{\mathcal{G}}, \subseteq)$, the least fixpoints of which are equal to the minimal model $\mathcal{M}_P$. T_P and K_P are given by:

Definition 5 [3] For each $X \subseteq \mathcal{G}$,

$$T_P(X) = \{head(C) \mid C \in Gclause(P) \ \& \ Body(C) \subseteq X\}. \quad \square$$

Definition 6 [3] For each $X \subseteq \mathcal{G}$,

$$K_P(X) \ = \ T_P(X) \cup X. \quad \square$$

Some important properties of the mappings T_P and K_P follow:

Proposition 2 [3] *T_P and K_P are $\subseteq$-continuous.* $\square$

Theorem 2 [3] *Let I be an interpretation. Then*

1. *I is a model of P, iff $T_P(I) \subseteq I$,*
2. *I is a model of P, iff $K_P(I) = I$.* $\square$

Theorem 3 [3] $\mathcal{M}_P = lfp(T_P) = lfp(K_P)$. $\square$

2.4 Examples

Examples 1 and 2 below demonstrate how to regard conventional logic programs and definite conceptual graph programs [13, 15, 34], 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

[3] See, for example, the first chapter of [21].

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. □

Example 2 Let a concept universe $\mathcal{U} = ((T_c, \leq_c), T_r, M, V, ::)$ be given, where $(T_c, \leq_c)$ is a lattice of concept types with the maximum element $\top$ and the minimum element $\bot$, T_r is a set of relation types, M is a set of individual markers, V is a set of variables[4], and $::$ is a binary relation from T_c to M, called the *conformity relation*, satisfying the conditions:

- For any $s, t \in T_c$, $m \in M$,
  - if $s :: m$ and $s \leq_c t$, then $t :: m$,
  - if $s :: m$, $t :: m$ and u is the greatest lower bound of s and t, then $u :: m$.
- For any $m \in M$, $\top :: m$, but not $\bot :: m$.

An individual marker $m \in M$ is said to *conform* to a conceptual type $t \in T_c$, iff $t :: m$. In general, the sets T_r and M may be partially ordered. To simplify the presentation, the partial orders on these two sets are not considered in this example.

An *atomic conceptual graph on* $\mathcal{U}$ is a bipartite, connected, finite, directed graph $G = (C, R, E, lab)$, where C and R are two classes of vertices, the elements of which are called *concepts* and *conceptual relations*, respectively, E is a set of edges, and lab is a mapping that associates with each vertex a label satisfying the conditions:

- For each $c \in C$, either $lab(c)$ is a concept type in T_c, or $lab(c)$ is of the form $t : r$, where $t \in T_c$ and r is either an individual marker in M that conforms to t or a variable in V.
- For each $r \in R$, $lab(r)$ is a relation type in T_r.

Let a specialization system $\Gamma_2 = (\mathcal{A}_2, \mathcal{G}_2, \mathcal{S}_2, \mu_2)$ be defined as follows:

[4] In conceptual graphs, a variable is usually represented by the generic marker *, followed by an identifier for indicating cross references.

1. $\mathcal{A}_2$ is the set of all atomic conceptual graphs on $\mathcal{U}$.

2. $\mathcal{G}_2$ is the subset of $\mathcal{A}_2$ that consists of all variable-free atomic conceptual graphs.

3. $\mathcal{S}_2$ is the set of all substitutions of the form $\{v_1/r_1, \ldots, v_n/r_n\}$, where the v_i are distinct variables in V, and for each $j \in \{1, \ldots, n\}$, $r_j \in M \cup V$ and $v_j \neq r_j$.

4. Given $s = \{v_1/r_1, \ldots, v_n/r_n\} \in \mathcal{S}_2$ and $a \in \mathcal{A}_2$, if the result obtained from a by simultaneously replacing each occurrence of v_i in a by r_i, for each $i \in \{1, \ldots, n\}$, is a conceptual graph on $\mathcal{U}$,[5] then $(\mu_2 s)a$ is defined to be that result, otherwise $\mu_2 s$ is not applicable to a.

It is not difficult to see that Γ_2 satisfies the three conditions of Definition 1. A declarative program P on Γ_2 such that for each clause $C \in P$, every variable occurring in $head(C)$ also occurs in $Body(C)$ is a conceptual graph program; and, the semantics of declarative programs with implicit implication, which will be developed in Sections 3 and 4, yields its expected meaning. □

3 Implicit Implication as a Preorder

When a generalization taxonomy of types or classes is provided, an implicit-implication relation among the atoms in an interpretation domain can often be determined by examination of their structures and intended meanings. For example, in a conceptual graph language [13, 15, 34], since atomic conceptual graphs, in the linear notation, $[t : o] \rightarrow (r) \rightarrow [t' : o']$ and $[t : o] \rightarrow (r) \rightarrow [t']$ are intended to mean "there exist objects o of type t and o' of type t' such that o has relation r to o'" and "there exists an object o of type t such that o has

[5] This condition is satisfied iff for each concept c in a and each binding $v_j/r_j \in s$, if $lab(c) = t : v_j$ and $r_j \in M$, then $t :: r_j$.

relation r to some object of type t'", respectively, the atomic conceptual graph

[microprocessor : #123] → (part) → [register : #001]

implicitly implies the atomic conceptual graph

[digitalDevice : #123] → (part) → [memory],

provided that the types microprocessor and register are more specific than the types digitalDevice and memory, respectively. In F-logic [17], as a signature expression $c[m \Rightarrow c']$ is intended to mean "if a method m for an object of class c is defined or derived, then it must return an object of class c'", the signature expression

memory[content ⇒ digitalValue]

implicitly implies the signature expression

counter[content ⇒ value],

provided that the id-terms counter and digitalValue are subclasses of the id-terms memory and value, respectively. Likewise, in KL-ONE-like languages [7, 8, 9, 22], the conceptual description

[device that receives data from at least one analogTransducer]

subsumes the conceptual description

[microprocessor that receives data from at least three thermistors],

provided that the primitive concepts [device] and [analogTransducer] subsume the primitive concepts [microprocessor] and [thermistor], respectively; and, therefore, if an individual object satisfies the latter description, the object will also satisfy the former description. This kind of system-defined implicit implication should be separated from application-dependent implication explicitly defined by definite clauses in application programs, and, as will be described in Section 4, can be employed to enhance systems' computation mechanisms.

This section assumes that the implicit implication among the atoms in an interpretation domain can be predetermined and explicitly represented by a preorder on the domain. More precisely, in the sequel, it will be assumed that $\Gamma_{\sqsubseteq} = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ is a specialization system and $\sqsubseteq$ is a preorder on $\mathcal{G}$ such that for any $g, g' \in \mathcal{G}$, $g \sqsubseteq g'$, iff g is implicitly implied by g'. Under the constraint that every interpretation must conform to the implicit implication, an appropriate model-theoretic semantics for declarative programs with respect to $\sqsubseteq$ along with its corresponding fixpoint semantics will now be developed.

3.1 Model-Theoretic Semantics

In the original DP theory, an interpretation arbitrarily assigns truth values to the atoms in an interpretation domain, whence every subset of the domain can serve as one possible interpretation. Under the established assumption, in contrast, the truth values of the atoms must be consistent with the implicit implication described by the preorder $\sqsubseteq$, and thus cannot be randomly assigned. Accordingly, not all of the original interpretations, but only those which are closed with respect to $\sqsubseteq$ will be used henceforth to discuss the model-theoretic semantics for declarative programs on $\Gamma_{\sqsubseteq}$.

Definition 7 ($\sqsubseteq$-**Closed Interpretation**) An interpretation I is said to be $\sqsubseteq$-*closed*, iff

$$(\forall g \in I)(\forall g' \in \mathcal{G}): \quad (g' \sqsubseteq g \implies g' \in I). \quad \square$$

Lemma 1 *The intersection of more than one $\sqsubseteq$-closed interpretation is also a $\sqsubseteq$-closed interpretation.* $\square$

Proof For some index set J, let $\{I_j \mid j \in J\}$ be a non-empty set of $\sqsubseteq$-closed interpretations. Let $g \in \bigcap_{j \in J} I_j$ and let $g' \in \mathcal{G}$ such that $g' \sqsubseteq g$. For each $j \in J$, since $g \in I_j$ and I_j is $\sqsubseteq$-closed, it follows that $g' \in I_j$, whence $g' \in \bigcap_{j \in J} I_j$. $\blacksquare$

relation r to some object of type t'", respectively, the atomic conceptual graph

[microprocessor : #123] → (part) → [register : #001]

implicitly implies the atomic conceptual graph

[digitalDevice : #123] → (part) → [memory],

provided that the types microprocessor and register are more specific than the types digitalDevice and memory, respectively. In F-logic [17], as a signature expression $c[m \Rightarrow c']$ is intended to mean "if a method m for an object of class c is defined or derived, then it must return an object of class c'", the signature expression

memory[content ⇒ digitalValue]

implicitly implies the signature expression

counter[content ⇒ value],

provided that the id-terms counter and digitalValue are subclasses of the id-terms memory and value, respectively. Likewise, in KL-ONE-like languages [7, 8, 9, 22], the conceptual description

[device that receives data from at least one analogTransducer]

subsumes the conceptual description

[microprocessor that receives data from at least three thermistors],

provided that the primitive concepts [device] and [analogTransducer] subsume the primitive concepts [microprocessor] and [thermistor], respectively; and, therefore, if an individual object satisfies the latter description, the object will also satisfy the former description. This kind of system-defined implicit implication should be separated from application-dependent implication explicitly defined by definite clauses in application programs, and as will be described in Section 4, can be employed to enhance systems' computation mechanisms.

This section assumes that the implicit implication among the atoms in an interpretation domain can be predetermined and explicitly represented by a preorder on the domain. More precisely, in the sequel, it will be assumed that $\Gamma_{\sqsubseteq} = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ is a specialization system and $\sqsubseteq$ is a preorder on $\mathcal{G}$ such that for any $g, g' \in \mathcal{G}$, $g \sqsubseteq g'$, iff g is implicitly implied by g'. Under the constraint that every interpretation must conform to the implicit implication, an appropriate model-theoretic semantics for declarative programs with respect to $\sqsubseteq$ along with its corresponding fixpoint semantics will now be developed.

3.1 Model-Theoretic Semantics

In the original DP theory, an interpretation arbitrarily assigns truth values to the atoms in an interpretation domain, whence every subset of the domain can serve as one possible interpretation. Under the established assumption, in contrast, the truth values of the atoms must be consistent with the implicit implication described by the preorder $\sqsubseteq$, and thus cannot be randomly assigned. Accordingly, not all of the original interpretations, but only those which are closed with respect to $\sqsubseteq$ will be used henceforth to discuss the model-theoretic semantics for declarative programs on $\Gamma_{\sqsubseteq}$.

Definition 7 ($\sqsubseteq$-**Closed Interpretation**) An interpretation I is said to be $\sqsubseteq$-*closed*, iff

$$(\forall g \in I)(\forall g' \in \mathcal{G}) : \quad (g' \sqsubseteq g \implies g' \in I). \quad \square$$

Lemma 1 *The intersection of more than one* $\sqsubseteq$*-closed interpretation is also a* $\sqsubseteq$*-closed interpretation.* $\square$

Proof For some index set J, let $\{I_j \mid j \in J\}$ be a non-empty set of $\sqsubseteq$-closed interpretations. Let $g \in \bigcap_{j \in J} I_j$ and let $g' \in \mathcal{G}$ such that $g' \sqsubseteq g$. For each $j \in J$, since $g \in I_j$ and I_j is $\sqsubseteq$-closed, it follows that $g' \in I_j$, whence $g' \in \bigcap_{j \in J} I_j$. ■

The truth value of a definite clause on $\mathcal{A}$ with respect to a $\sqsubseteq$-closed interpretation is still defined as in the original DP theory (see Definition 3). A *$\sqsubseteq$-closed model* of a declarative program P on $\Gamma_{\sqsubseteq}$ is defined in a straightforward manner as a $\sqsubseteq$-closed interpretation which is also a model (according to Definition 4) of P. The meaning with respect to $\sqsubseteq$ of a declarative program P on $\Gamma_{\sqsubseteq}$ is then defined as the intersection of all $\sqsubseteq$-closed models of P, which is the *minimal $\sqsubseteq$-closed model* of P (see Proposition 3 and Theorem 4 below) and is denoted by $\mathcal{M}_P^{\sqsubseteq}$.

Proposition 3 ($\sqsubseteq$-Closed-Model Intersection Property) *The intersection of more than one $\sqsubseteq$-closed model of a declarative program P on $\Gamma_{\sqsubseteq}$ is also a $\sqsubseteq$-closed model of P.* □

Proof The proof follows immediately from Proposition 1 and Lemma 1. ■

Theorem 4 *Every declarative program P on $\Gamma_{\sqsubseteq}$ has the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$, which is the intersection of all $\sqsubseteq$-closed models of P.* □

Proof Since $\mathcal{G}$ is a model of every declarative program on $\Gamma_{\sqsubseteq}$ and is also $\sqsubseteq$-closed, P has at least one $\sqsubseteq$-closed model. Then, by Proposition 3, $\mathcal{M}_P^{\sqsubseteq}$ is a $\sqsubseteq$-closed model of P, which is obviously minimal with respect to set inclusion. ■

3.2 Fixpoint Semantics

In order to provide fixpoint characterization of $\sqsubseteq$-closed model semantics, the $\subseteq$-continuous mapping $K_P^{\mathcal{E}}$ on the complete lattice $(2^{\mathcal{G}}, \subseteq)$ is associated with a declarative program P on $\Gamma_{\sqsubseteq}$. An important virtue of this mapping is that each of its fixpoints determines a $\sqsubseteq$-closed model of P (Theorem 6). Therefore, the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$ can be obtained by computing its least fixpoint (Theorem 7). Introduce now the notion of expanded version of a subset of $\mathcal{G}$ with respect to the preorder $\sqsubseteq$, which will be used in the definition of the mapping $K_P^{\mathcal{E}}$:

Definition 8 (Expanded Set)[6] Let $X \subseteq \mathcal{G}$. The *expanded version* of X, denoted by $\mathcal{E}(X)$, is defined as:

$$\mathcal{E}(X) \quad = \quad \{g \in \mathcal{G} \mid (\exists x \in X) : g \sqsubseteq x\}. \quad \square$$

The next proposition links the notion of $\sqsubseteq$-closed interpretation to that of expanded version of an interpretation.

Proposition 4 *An interpretation I is $\sqsubseteq$-closed, iff $\mathcal{E}(I) = I$.* $\square$

Proof Note first that, by the reflexivity of $\sqsubseteq$, $\mathcal{E}(I) \supseteq I$. Thus

$$\begin{aligned} (\exists g \in I)(\exists g' \in \mathcal{G}) : g' \sqsubseteq g \ \ \& \ \ g' \notin I \quad &\Longleftrightarrow \quad (\exists g' \in \mathcal{G}) : g' \in \mathcal{E}(I) \ \ \& \ \ g' \notin I \\ &\Longleftrightarrow \quad \mathcal{E}(I) \supset I \\ &\Longleftrightarrow \quad \mathcal{E}(I) \neq I, \end{aligned}$$

i.e., I is not $\sqsubseteq$-closed, iff $\mathcal{E}(I) \neq I$. ■

Now, let P be a declarative program on $\Gamma_{\sqsubseteq}$. Note that $\mathcal{E}(\mathcal{M}_P)$, the expanded version of the minimal model $\mathcal{M}_P$, is possibly not a model of P (since there may exist some clause $C \in$ *Gclause*(P) such that *Body*$(C) \not\subseteq \mathcal{M}_P$ and *head*$(C) \notin \mathcal{M}_P$, whereas *Body*$(C) \subseteq \mathcal{E}(\mathcal{M}_P)$ and *head*$(C) \notin \mathcal{E}(\mathcal{M}_P)$). This clarifies that the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$ is, in general, not equal to $\mathcal{E}(\mathcal{M}_P)$; and, accordingly, $\mathcal{M}_P^{\sqsubseteq}$ cannot be obtained simply by expanding the least fixpoint of K_P. Next, consider the mapping $K_P^{\mathcal{E}}$, the least fixpoint of which equals $\mathcal{M}_P^{\sqsubseteq}$:

Definition 9 The mapping $K_P^{\mathcal{E}}: 2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$ is given by

$$K_P^{\mathcal{E}}(X) \quad = \quad K_P(\mathcal{E}(X)),$$

for each $X \subseteq \mathcal{G}$. $\square$

[6] This definition is an adaptation of that of an expanded set with respect to a partial order on a basic set, given in [19].

Proposition 5 and Theorems 5 and 6 below describe some properties of the mapping $K_P^{\mathcal{E}}$.

Proposition 5 *$K_P^{\mathcal{E}}$ is $\subseteq$-continuous.* □

Proof Considering $\mathcal{E}$ as a mapping from $2^{\mathcal{G}}$ to $2^{\mathcal{G}}$, it will first be shown that $\mathcal{E}$ is $\subseteq$-continuous. Let X be a directed subset of $2^{\mathcal{G}}$, $\bigcup X$ and $\bigcup \mathcal{E}(X)$ denote $\bigcup_{x \in X} x$ and $\bigcup_{x \in X} \mathcal{E}(x)$, respectively. Then

$$\begin{aligned} g \in \mathcal{E}(\bigcup X) &\iff (\exists g' \in \bigcup X) : g \sqsubseteq g' \\ &\iff (\exists x \in X)(\exists g' \in x) : g \sqsubseteq g' \\ &\iff (\exists x \in X) : g \in \mathcal{E}(x) \\ &\iff g \in \bigcup \mathcal{E}(X). \end{aligned}$$

Thus $\mathcal{E}(\bigcup X) = \bigcup \mathcal{E}(X)$, *i.e.*, $\mathcal{E}(lub(X)) = lub(\mathcal{E}(X))$, whence $\mathcal{E}$ is $\subseteq$-continuous. Then, it follows from Proposition 2 and Result 1 of Lemma 3 in the appendix that $K_P^{\mathcal{E}} = K_P \circ \mathcal{E}$ is $\subseteq$-continuous. ■

Theorem 5 *$lfp(K_P^{\mathcal{E}}) = K_P^{\mathcal{E}} \uparrow \omega$.* □

Proof The proof follows directly from Proposition 5 and Proposition 10 in the appendix. ■

Theorem 6 *Let I be an interpretation, then $K_P^{\mathcal{E}}(I) = I$, iff I is a $\subseteq$-closed model of P.* □

Proof It is clear that, for each $X \subseteq \mathcal{G}$, $K_P(X) \supseteq X$ and $\mathcal{E}(X) \supseteq X$. Thus, for each interpretation I,

$$K_P^{\mathcal{E}}(I) = K_P(\mathcal{E}(I)) = I \iff K_P(I) = I \ \&\ \mathcal{E}(I) = I.$$

The result then follows from Theorem 2 and Proposition 4. ■

The main result of this subsection is:

Theorem 7 $\mathcal{M}_{\overline{P}}^{\Xi} = lfp(K_{\overline{P}}^{\xi})$. □

Proof The result follows from Theorems 4 and 6. ■

4 Implicit Implication as a Partial Order

As has been illustrated at the beginning of the last section, the implicit implication among atoms in a particular system, where their intended meanings are clearly known, can, in general, be decided upon by examination of their forms and the generalization relationship between the types or classes occurring at the corresponding positions in them. In a conceptual graph language, for example, given two canonical conceptual graphs G and H, G is more specific than, and, thus, implicitly implies H, iff there exists a projection[7] from H to G [23, 25]. The existence of a projection, which is a kind of graph morphism, from one conceptual graph to another depends solely on the syntactic structures of the two graphs and the generalization hierarchies of concept types, relation types and markers. An algorithm for computing a projection from one conceptual graph to another was developed in [24, 25]. In particular, it is shown in [24, 25] that if G is a conceptual tree, *i.e.*, a conceptual graph without cycles, except for cycles created by multi-edges between a relation vertex and one of its neighbours, then a projection from G to any conceptual graph can be computed in polynomial time.[8]

A generalization hierarchy of types or classes commonly has a partial-order structure,

[7] A projection [27] is a graph morphism, preserving the order on edges and complying with some additional rules on vertex labels.

[8] Trees seem to be very frequent in conceptual graph applications [25]. The irredundant atomic conceptual graphs shown by the figures in this paper are all conceptual trees. A general algorithm for computing a projection from one conceptual graph to another conceptual graph by converting the former into a tree and then using the polynomial-time algorithm for computing a projection from a tree to a graph as a preprocessing part is also described in [25].

and consequently, the implicit-implication relation is often also a partial order. For instance, the implicit implication on irredundant atomic conceptual graphs is a partial order. It will now be assumed, in addition, that the implicit-implication relation on the interpretation domain is a partial order, *i.e.*, the preorder $\sqsubseteq$ in the last section is, in addition, assumed to be antisymmetric. Under this stronger assumption, it applies the results on fixpoint iteration with subsumption provided by [18, 19] (Subsection 4.1) to describe more elegant fixpoint semantics for declarative programs with respect to the implicit implication (Subsection 4.2).

Formally, throughout this section, let $\Gamma_{\sqsubseteq} = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ be a specialization system and $\sqsubseteq$ a *partial order* on $\mathcal{G}$ such that for any $g, g' \in \mathcal{G}$, $g \sqsubseteq g'$, iff g is assumed to be implicitly implied by g'. All the definitions and results in the previous section apply in this section.

4.1 Basic Definitions and Results

Recall now some definitions and results from [18, 19], which will be used in Subsection 4.2.[9]

Based on the partial order $\sqsubseteq$ on $\mathcal{G}$, the binary relation $\sqsubseteq$ on $2^{\mathcal{G}}$ is defined by

$$X \sqsubseteq Y \iff (\forall x \in X)(\exists y \in Y) : x \sqsubseteq y,$$

for any $X, Y \subseteq \mathcal{G}$. This relation is a preorder on $2^{\mathcal{G}}$, but not necessarily a partial order.[10] Based on it, the equivalence relation $\sim$ on $2^{\mathcal{G}}$ is defined by

$$X \sim Y \iff X \sqsubseteq Y \ \& \ Y \sqsubseteq X,$$

for any $X, Y \subseteq \mathcal{G}$. The preorder $\sqsubseteq$ on $2^{\mathcal{G}}$ is extended to the quotient set of $2^{\mathcal{G}}$ modulo $\sim$ (*i.e.*, $2^{\mathcal{G}}/\sim$) by

$$[X] \sqsubseteq [Y] \iff X \sqsubseteq Y,$$

[9] It should be noted that it is only assumed in [18, 19] that $\sqsubseteq$ is a partial order on a basic set (in this paper, the interpretation domain $\mathcal{G}$), i.e., all the results presented in this subsection still hold without the condition that the partial order $\sqsubseteq$ represents an implicit-implication relation on $\mathcal{G}$.

[10] This preorder on $2^{\mathcal{G}}$ is usually called Hoare's ordering.

for any $[X], [Y] \in 2^{\mathcal{G}}/\sim$. This extended relation is a partial order on $2^{\mathcal{G}}/\sim$.

Next, consider the notion of a reduced version of a subset of $\mathcal{G}$:

Definition 10 [19] **(Reduced Set)** Let $X \subseteq \mathcal{G}$, $\mathcal{C}$ be the set of maximal (with respect to set inclusion) chains[11] of X, and, for each $C \in \mathcal{C}$, $max_{\sqsubseteq}(C)$ denote the maximum (with respect to $\sqsubseteq$) element, if it exists, of C. The *reduced version* of X, denoted by $\mathcal{R}(X)$, is defined by:

$$\mathcal{R}(X) \;=\; \bigcup_{C \in \mathcal{C}} R_C,$$

where

$$R_C = \begin{cases} \{max_{\sqsubseteq}(C)\}, & \text{if } max_{\sqsubseteq}(C) \text{ exists.} \\ C, & \text{otherwise.} \end{cases}$$

Denote by $2^{\mathcal{G}}_{\mathcal{R}}$ the set of all reduced subsets of $\mathcal{G}$, *i.e.*, the set $\{\mathcal{R}(X) \mid X \in 2^{\mathcal{G}}\}$. □

For each $X \subseteq \mathcal{G}$, the maximal chains in X without maximum elements[12] are left unchanged in $\mathcal{R}(X)$, while those with maximum elements are reduced to their maximum elements in $\mathcal{R}(X)$. Thus, if X is a finite set, then $\mathcal{R}(X)$ consists only of the maximal elements of X. The next proposition interrelates reduced sets, expanded sets, set inclusion, and the relations $\sqsubseteq$ and $\sim$ on $2^{\mathcal{G}}$.

Proposition 6 [19] *If* $X, Y \subseteq \mathcal{G}$, *then*

1. $X \sim \mathcal{R}(X) \sim \mathcal{E}(X)$,
2. $X \sqsubseteq Y \implies \mathcal{R}(X) \sqsubseteq \mathcal{R}(Y)$ & $\mathcal{E}(X) \subseteq \mathcal{E}(Y)$. □

[11] The maximal chains of a partially-ordered set X are the totally-ordered subsets of X that are maximal with respect to set inclusion. For example, let $X = \{a, b, c, d\}$ be partially ordered by $a \sqsubseteq b \sqsubseteq c$ and $a \sqsubseteq d$. Then the maximal chains of X are $\{a, b, c\}$ and $\{a, d\}$.

[12] Only infinite chains may have no maximum elements.

Using Result 2 of Proposition 6, it can be shown that:

Proposition 7 *$K_P^{\mathcal{E}}$ is $\sqsubseteq$-monotonic.* □

Proof Let $X, Y \subseteq \mathcal{G}$. Then

$$\begin{aligned} X \sqsubseteq Y &\implies \mathcal{E}(X) \subseteq \mathcal{E}(Y) && \text{(by Result 2 of Proposition 6)} \\ &\implies K_P(\mathcal{E}(X)) \subseteq K_P(\mathcal{E}(Y)) && \text{(as } K_P \text{ is } \subseteq\text{-monotonic}^{13}\text{)} \\ &\implies K_P(\mathcal{E}(X)) \sqsubseteq K_P(\mathcal{E}(Y)) && \text{(by the reflexivity of } \sqsubseteq\text{)} \\ &\implies K_P^{\mathcal{E}}(X) \sqsubseteq K_P^{\mathcal{E}}(Y). \end{aligned}$$

■

It is shown in [19] that the partially-ordered set $(2^{\mathcal{G}}_{\mathcal{R}}/\sim, \sqsubseteq)$ is a complete lattice, where the top element is $[\mathcal{R}(\mathcal{G})]$, the bottom element is $[\emptyset]$ and $lub\{[X_j] \mid j \in J\} = [\mathcal{R}(\bigcup_{j \in J} X_j)]$. Next, recall the main theoretical results on fixpoint iteration on this complete lattice, provided by [18, 19].

In the sequel, let $F: 2^{\mathcal{G}} \to 2^{\mathcal{G}}$.

Definition 11 [19] If F is $\sqsubseteq$-monotonic, then the mappings $F_{\mathcal{R}}$ and $F^{\sim}$ are defined by:

1. $F_{\mathcal{R}}: 2^{\mathcal{G}} \to 2^{\mathcal{G}}$ such that $F_{\mathcal{R}}(X) = \mathcal{R}(F(X))$, for each $X \subseteq \mathcal{G}$,
2. $F^{\sim}: 2^{\mathcal{G}}_{\mathcal{R}}/\sim \to 2^{\mathcal{G}}_{\mathcal{R}}/\sim$ such that $F^{\sim}([X]) = [F_{\mathcal{R}}(X)]$, for each $[X] \in 2^{\mathcal{G}}_{\mathcal{R}}/\sim$.[14] □

Theorem 8 [19] *If F is $\sqsubseteq$-monotonic, then $F^{\sim}$ has a least fixpoint.* □

Theorem 9 [19] *If F is $\sqsubseteq$-monotonic and $\subseteq$-continuous, then*

1. $lfp(F^{\sim}) = F^{\sim} \uparrow \omega$,
2. $[\mathcal{R}(F \uparrow n)] = F^{\sim} \uparrow n = [F_{\mathcal{R}} \uparrow n]$, *for any* $n < \omega$.
3. $[\mathcal{R}(lfp(F))] = lfp(F^{\sim}) = lub\{[F_{\mathcal{R}} \uparrow n] \mid n < \omega\}$. □

[13] K_P is $\subseteq$-continuous, by Proposition 2, and, hence, $\subseteq$-monotonic.

[14] It is shown in [19] that, for any $X, Y \subseteq \mathcal{G}$, $X \sim Y$ implies $\mathcal{R}(F(X)) \sim \mathcal{R}(F(Y))$, and, thus, $F^{\sim}$ is well defined. Moreover, $F_{\mathcal{R}}$ and $F^{\sim}$ are both $\subseteq$-monotonic.

4.2 More Elegant Fixpoint Semantics

Under the assumption of this section, it follows from Propositions 4 and 6 that, for every $\sqsubseteq$-closed interpretation I, $\mathcal{E}(\mathcal{R}(I)) = \mathcal{E}(I) = I$, *i.e.*, every $\sqsubseteq$-closed interpretation can be recaptured from its reduced version by expansion. This suggests that $\sqsubseteq$-closed interpretations can be equivalently represented by their reduced versions. Moreover, for any declarative program P on $\Gamma_{\sqsubseteq}$, as $K_P^{\mathcal{E}}$ is both $\subseteq$-continuous and $\sqsubseteq$-monotonic (Propositions 5 and 7), $K_P^{\mathcal{E}}$ determines the mapping $(K_P^{\mathcal{E}})^{\sim}$ on the complete lattice $(2_{\mathcal{R}}^{\mathcal{G}}/\sim, \sqsubseteq)$ according to Definition 11, *i.e.*, given $[X] \in 2_{\mathcal{R}}^{\mathcal{G}}/\sim$,

$$(K_P^{\mathcal{E}})^{\sim}([X]) \quad = \quad [\mathcal{R}(K_P^{\mathcal{E}}(X))],$$

and it follows from Theorem 7 and Result 3 of Theorem 9 that

$$[\mathcal{R}(\mathcal{M}_P^{\sqsubseteq})] \quad = \quad [\mathcal{R}(lfp(K_P^{\mathcal{E}}))] \quad = \quad lfp((K_P^{\mathcal{E}})^{\sim}).$$

Hence, if $lfp((K_P^{\mathcal{E}})^{\sim}) = [A]$, then, by Result 2 of Proposition 6, $\mathcal{E}(A) = \mathcal{E}(\mathcal{R}(\mathcal{M}_P^{\sqsubseteq}))$, and, as $\mathcal{M}_P^{\sqsubseteq}$ is $\sqsubseteq$-closed, then, $\mathcal{E}(A) = \mathcal{M}_P^{\sqsubseteq}$. Therefore, the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$ can be obtained by expansion of any representative of the least fixpoint of $(K_P^{\mathcal{E}})^{\sim}$.

At first glance, computing $\mathcal{M}_P^{\sqsubseteq}$ by application of $(K_P^{\mathcal{E}})^{\sim}$, described above, seems to be efficient, in that $(K_P^{\mathcal{E}})^{\sim}$ is a mapping on a quotient set of the reduced subsets of $\mathcal{G}$. However, on closer examination of Definitions 9 and 11, one finds that for any equivalence class $[X]$ in the quotient set $2_{\mathcal{R}}^{\mathcal{G}}/\sim$,

$$(K_P^{\mathcal{E}})^{\sim}([X]) \quad = \quad [\mathcal{R}(K_P^{\mathcal{E}}(X))] \quad = \quad [\mathcal{R}(K_P(\mathcal{E}(X)))]. \tag{1}$$

Therefore, if $(K_P^{\mathcal{E}})^{\sim}([X])$ is evaluated directly according to Equation (1), *i.e.*, by means of the mapping K_P, then the reduced set X must be expanded and the merit of computation on reduced sets will be lost. It will next be shown that, instead of using Equation (1), $(K_P^{\mathcal{E}})^{\sim}([X])$ can be computed by using another mapping $K_P^{\sqsubseteq}$, which does not involve the expanded version of X.

In the sequel, let P be a declarative program on $\Gamma_{\subseteq}$. The next definition associates with P a mapping $T_P^{\sqsubseteq}$ on $2^{\mathcal{G}}$, based on which the mapping $K_P^{\sqsubseteq}$ is defined.

Definition 12 For each $X \subseteq \mathcal{G}$,

$$T_P^{\sqsubseteq}(X) = \{head(C) \mid C \in Gclause(P) \ \& \ Body(C) \sqsubseteq X\}. \quad \square$$

Definition 13 The mapping $K_P^{\sqsubseteq}: 2^{\mathcal{G}} \to 2^{\mathcal{G}}$ is defined by

$$K_P^{\sqsubseteq}(X) = T_P^{\sqsubseteq}(X) \cup X,$$

for each $X \subseteq \mathcal{G}$. $\square$

The next lemma and proposition assert some characteristics of the mappings $T_P^{\sqsubseteq}$ and $K_P^{\sqsubseteq}$.

Lemma 2 *Let $X \subseteq \mathcal{G}$, then*

1. $T_P(\mathcal{E}(X)) = T_P^{\sqsubseteq}(X)$,
2. $K_P^{\mathcal{E}}(X) \supseteq K_P^{\sqsubseteq}(X)$,
3. $K_P^{\mathcal{E}}(X) \sim K_P^{\sqsubseteq}(X)$. $\square$

Proof

1. By Definition 8 and the definition of $\sqsubseteq$ on $2^{\mathcal{G}}$, for any $Y, Z \subseteq \mathcal{G}$,

$$\begin{aligned} Y \subseteq \mathcal{E}(Z) &\iff (\forall y \in Y) : y \in \mathcal{E}(Z) \\ &\iff (\forall y \in Y)(\exists z \in Z) : y \sqsubseteq z \\ &\iff Y \sqsubseteq Z. \end{aligned}$$

Then, it follows directly from the definitions of T_P and $T_P^{\sqsubseteq}$ (Definitions 5 and 12) that $T_P(\mathcal{E}(X)) = T_P^{\sqsubseteq}(X)$, for each $X \subseteq \mathcal{G}$.

3. Let $X, Y \subseteq \mathcal{G}$. Then

$$X \sqsubseteq Y \implies K_P^{\mathcal{E}}(X) \sqsubseteq K_P^{\mathcal{E}}(Y) \quad \text{(by Proposition 7)}$$
$$\implies K_P^{\sqsubseteq}(X) \sqsubseteq K_P^{\sqsubseteq}(Y) \quad \text{(by Result 3 of Lemma 2).}$$

■

As $K_P^{\sqsubseteq}$ is always $\sqsubseteq$-monotonic as well as $\subseteq$-continuous (Proposition 8), the mapping $(K_P^{\sqsubseteq})^{\sim}$ on the complete lattice $(2_{\mathcal{K}}^{\mathcal{V}}/\sim, \sqsubseteq)$ is well-defined by Definition 11, *i.e.*, for each $[X] \in 2_{\mathcal{K}}^{\mathcal{V}}/\sim$,

$$(K_P^{\sqsubseteq})^{\sim}([X]) \quad = \quad [\mathcal{R}(K_P^{\sqsubseteq}(X))],$$

and $(K_P^{\sqsubseteq})^{\sim}$ has all the properties listed in Theorem 9. Proposition 9 below establishes the equality between the mappings $(K_P^{\mathcal{E}})^{\sim}$ and $(K_P^{\sqsubseteq})^{\sim}$.

Proposition 9 $(K_P^{\mathcal{E}})^{\sim} = (K_P^{\sqsubseteq})^{\sim}$. □

Proof By Result 3 of Lemma 2 and Result 1 of Proposition 6, $\mathcal{R}(K_P^{\mathcal{E}}(X)) \sim \mathcal{R}(K_P^{\sqsubseteq}(X))$ for each $X \subseteq \mathcal{G}$. Hence, the mappings $(K_P^{\mathcal{E}})^{\sim}$ and $(K_P^{\sqsubseteq})^{\sim}$ are equal.

As a result, given an equivalence class $[X]$ in $2_{\mathcal{K}}^{\mathcal{V}}/\sim$, $(K_P^{\mathcal{E}})^{\sim}([X])$ can be computed through the mapping $K_P^{\sqsubseteq}$ by the equation:

$$(K_P^{\mathcal{E}})^{\sim}([X]) \quad = \quad (K_P^{\sqsubseteq})^{\sim}([X]) \quad = \quad [\mathcal{R}(K_P^{\sqsubseteq}(X))]. \tag{2}$$

Observe that, in the evaluation of $K_P^{\sqsubseteq}(X)$, X is not expanded, but directly compared with $Body(C)$, based on the preorder $\sqsubseteq$ on $2^{\mathcal{V}}$, for each $C \in Gclause(P)$.

The next theorem is the main result of this section. It intimates that the expanded version of any arbitrary representative of the least fixpoint of $(K_P^{\sqsubseteq})^{\sim}$ is equal to the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$.

Theorem 10 $[\mathcal{R}(\mathcal{M}_P^{\sqsubseteq})] = [\mathcal{R}(lfp(K_P^{\mathcal{E}}))] = lfp((K_P^{\sqsubseteq})^{\sim})$. □

Proof The result follows from Theorem 7, Result 3 of Theorem 9, and Proposition 9. ■

4.3 Comparisons with Related Works

In [18, 19], Köstler *et. al.* augmented logic programming by incorporating semantic control knowledge in the form of user-supplied subsumption information, which may be used to expedite query evaluation process, and extended the classical theorems for least models and least fixpoints accordingly. In their proposals, a user can provide a subsumption ordering on the Herbrand base, by means of meta-rules, in order to specify that some ground atom is semantically preferable to, or more intended than, or more useful than another ground atom. For example, in the problem of computing shortest paths, as illustrated in [19], the atom $\mathsf{path}(a, b, c_1)$, asserting that there exists a path the cost of which is c_1 from a vertex a to a vertex b, can be considered to subsume the atom $\mathsf{path}(a, b, c_2)$ if c_1 is less than c_2.

On condition that the conventional immediate consequence operator T_P is monotonic with respect to the subsumption ordering, Köstler *et. al.* succeeded in applying their elegant theorem of fixpoint iteration with subsumption (Theorem 9) to the development of efficient iteration schemes for bottom-up query evaluation with respect to the conventional semantics of logic programs and the supplied subsumption information. As pointed out in [19], however, the operator T_P is, in general, not monotonic with respect to the subsumption ordering, and, consequently, Theorem 9 does not always apply

This paper, in contrast, focuses on the implicit-implication relation due to taxonomic information, and develops a natural semantics for declarative programs, which accounts for the impact of the implicit implication. Under the practical assumption that the implicit-implication relation can be determined in advance and represented by a partial order $\sqsubseteq$ on the interpretation domain $\mathcal{G}$, the operator $K_P^{\mathcal{E}}$, the least fixpoint of which determines the proposed meaning $\mathcal{M}_P^{\sqsubseteq}$, as well as the operator $K_P^{\sqsubseteq}$, which is specifically devised for efficient computation of $\mathcal{M}_P^{\sqsubseteq}$ on reduced subsets of $\mathcal{G}$, is always monotonic with respect to $\sqsubseteq$ (Propositions 7 and 8), and Theorem 9 always applies. Based on these results, the

reduced representation of $\mathcal{M}_{\mathcal{P}}^{\sqsubseteq}$ can be computed elegantly by means of the operator $(K_{\mathcal{P}}^{\sqsubseteq})^{\sim}$, an application of which compares a reduced subset of $\mathcal{G}$ directly, with respect to $\sqsubseteq$, with the bodies of ground program clauses. Such a comparison is especially suitable for dealing with the implicit implication; and, as illustrated in the beginning of this section, it mainly involves examination of the internal structures of atoms, and is often inexpensive in practice.

In their remarkable work [1], Aït-Kaci and Nasr introduced an extended form of first-order terms, called *ψ-terms*, and incorporated the employment of taxonomic information into ψ-term unification process. A ψ-term is a record-like type structure, which denotes a set of objects. For example, the set of all IC gates in the TTL family may be denoted by the ψ-term ic-gate[family => ttl], provided that the type ic-gate embraces all IC gates. One ψ-term is considered to subsume another ψ-term if the set of objects denoted by the former is a superset of that denoted by the latter. This subsumption relation is a partial order on the set of ψ-terms. The unification of two given ψ-terms is the operation that computes their greatest lower bound, which denotes the intersection of the sets of objects denoted by the two ψ-terms. For instance, assuming that ic-gate is a subtype of device, the unification of the ψ-terms device[family => ttl] and ic-gate yields the ψ-term ic-gate[family => ttl].

The unification of ψ-terms is used in [1], instead of the usual unification of first-order terms, in the goal-directed SLD-resolution mechanism of PROLOG. Suppose, for example, that one has the query

$$? \ \mathsf{connect}(\mathsf{X} : \mathsf{device}[\mathsf{family} => \mathsf{ttl}], \mathsf{Y} : \mathsf{device}[\mathsf{family} => \mathsf{cmos}]),$$

asking for all TTL devices that are connected to some CMOS devices. Through the unification of ψ-terms, the query can unify with the head of the program clause

$$\mathsf{connect}(\mathsf{X} : \mathsf{ic\text{-}gate}, \mathsf{Y} : \mathsf{ic\text{-}gate}) \quad \leftarrow \quad \mathsf{send}(\mathsf{X}, \mathsf{Z} : \mathsf{value}, \mathsf{Y}),$$

stating that an IC gate X is connected to an IC gate Y if X sends some value Z to Y, and can thus be resolved with this program clause. The unification coerces the ψ-terms

Figure 8: A conceptual graph G_6

Figure 9: A conceptual graph G_7

X : ic-gate and Y : ic-gate in the program clause to the ψ-terms X : ic-gate[family => ttl] and Y : ic-gate[family => cmos], respectively, and the resolvent thus obtained is

$$\text{send}(X : \text{ic-gate}[\text{family} => \text{ttl}], Z : \text{value}, Y : \text{ic-gate}[\text{family} => \text{cmos}]),$$

which becomes the new goal to be proven.

In comparison, the partial order in this paper represents the implicit-implication relation on atoms in the interpretation domain, each of which does not denote a set of objects, but a statement about objects. The greatest lower bound of two given atoms, if exists, is an atom which implicitly implies each of the two atoms; and, its existence does not, in general, signify that the two atoms are relevant to and can unify with each other. Referring to the hierarchy of concept types in Figure 1, for example, the conceptual graphs G_6 in Figure 8 and G_7 in Figure 9 have the conceptual graph G_8 in Figure 10 as their greatest lower bound. Notwithstanding, the graph G_6 is hardly relevant to the graph G_7, and if one has G_6 as a goal conceptual graph, it is hardly useful to try to prove G_6 by resolving it with a program clause the head of which is G_7.

By considering atoms in the interpretation domain as abstract entities, which are characterized by their implicit-implication relationship with others, this paper provides a general foundation for efficient bottom-up, forward-chaining evaluation of declarative programs. A top-down, goal-driven proof procedure, on the other hand, usually depends on the syntax and the internal structures of atoms, which vary with knowledge-representation languages,

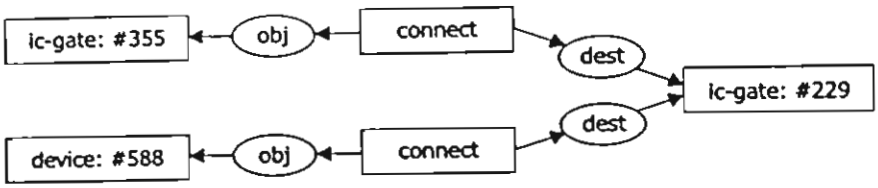

Figure 10: A conceptual graph G_8

and is normally tailored for each individual language. For example, the unification algorithms used in [1, 2] are designed specifically for atoms involving ψ-terms. For definite conceptual-graph programs, a goal-directed proof procedure was developed in [14].

5 Summary

Atoms in an interpretation domain normally serve as basic statements which describe various kinds of relationships among objects. When class/subclass information is provided, there usually exists an implicit-implication relation among the atoms, which can be determined by considering their forms and their intended meanings. In a system which regards taxonomic information as schema information, atoms are not used to describe subclass relationships, and, as a result, the implicit-implication relation does not depend on any particular interpretation and can usually be determined in advance.

This paper first assumes that the implicit-implication relation among the atoms in an interpretation domain is predetermined and described by a preorder $\sqsubseteq$ on the domain. The meaning of a declarative program P with respect to the implicit implication is then defined as the minimal $\sqsubseteq$-closed model $\mathcal{M}_P^{\sqsubseteq}$, which can also be characterized as the least fixpoint of the immediate-consequence operator $K_P^{\mathcal{E}}$ on the power set of the interpretation domain (Theorem 7). Afterwards, it is furthermore assumed that the implicit-implication relation $\sqsubseteq$ is a partial order and it is shown that, under this stronger assumption, the meaning $\mathcal{M}_P^{\sqsubseteq}$ of any program P can be computed more elegantly and more efficiently by expanding

any representative of the least fixpoint of the immediate-consequence operator $(K_P^{\sqsubseteq})^{\sim}$ on a quotient set of reduced subsets of the interpretation domain (Theorem 10).

Acknowledgement

The authors would like to thank the anonymous reviewers for their comments and suggestions which helped to improve the paper. This work was supported by the Thailand Research Fund.

Appendix: Continuous Operators on Complete Lattices

Throughout this appendix, let $(L, \leq)$ be a complete lattice, the minimal element of which is $\bot$. Given $T: L \to L$, let $T(X)$ denote $\{T(x) \mid x \in X\}$ for each $X \subseteq L$, and $T \uparrow \omega$ denote $lub\{T^n(\bot) \mid n \geq 0\}$, where $T^0(x) = x$ and $T^n(x) = T(T^{n-1}(x))$ for $n \geq 1$ [3]. A set $X \subseteq L$ is said to be *directed*, iff every finite subset of X has an upper bound in X [21]. A mapping $T: L \to L$ is said to be $\leq$-*continuous*, iff $T(lub(X)) = lub(T(X))$, for each directed subset X of L, and is said to be $\leq$-*monotonic*, iff $T(x) \leq T(y)$, for any $x, y \in L$ such that $x \leq y$ [21]. For any mappings $T, T': L \to L$, let the mapping $T' + T: L \to L$ be defined by

$$(T' + T)(x) \quad = \quad lub\{T'(x), T(x)\},$$

for each $x \in L$ [3].

Proposition 10 [21] *If $T: L \to L$ is a $\leq$-continuous mapping, then $lfp(T) = T \uparrow \omega$.* □

Lemma 3 *If $T, T': L \to L$ are both $\leq$-continuous mappings, then*

1. *$T' \circ T$ is $\leq$-continuous,*
2. *$T' + T$ is $\leq$-continuous.* □

Proof The first result of this lemma is well-known. Only the second result will be proven here. Let X be a directed subset of L. Note first that, if $a \in L$, then

$$\begin{aligned} \forall x \in X : a \geq (T' + T)(x) &\iff \forall x \in X : a \geq lub\{T'(x), T(x)\} \\ &\iff \forall x \in X : a \geq T'(x) \ \& \ a \geq T(x) \\ &\iff a \geq lub(T'(X)) \ \& \ a \geq lub(T(X)), \end{aligned}$$

i.e., a is an upper bound of $(T' + T)(X)$, iff a is an upper bound of $\{lub(T'(X)), lub(T(X))\}$. Thus

$$lub((T' + T)(X)) = lub\{lub(T'(X)), lub(T(X))\}.$$

It follows that

$$\begin{aligned} &(T' + T)(lub(X)) \\ &\quad = lub\{T'(lub(X)), T(lub(X))\} \\ &\quad = lub\{lub(T'(X)), lub(T(X))\} \quad \text{(as } T' \text{ and } T \text{ are } \leq\text{-continuous)} \\ &\quad = lub((T' + T)(X)). \end{aligned}$$

■

References

[1] H. Aït-Kaci and R. Nasr, "LOGIN: A Logic Programming Language with Built-in Inheritance". *J. Logic Programming*, vol. 3, no. 3, pp. 185–215, 1986.

[2] H. Aït-Kaci and A. Podelski, "Towards a Meaning of Life", *J. Logic Programming*, vol. 16, no. 3/4, pp. 195–234, 1993.

[3] K. Akama, "Declarative Semantics of Logic Programs on Parameterized Representation Systems", *Advances in Software Science and Technology*, vol. 5, pp. 45–63, 1993.

[4] C. Anutariya, V. Wuwongse, E. Nantajeewarawat and K. Akama, "Towards Computation with RDF Elements", *Proc. 1999 Int'l Symposium on Digital Libraries (ISDL)*, Tsukuba, Japan, pp. 112–119, 1999.

[5] A. Borgida, J. Mylopoulos and H. K. T. Wong, "Generalization/Specialization as a Basis for Software Specification", M. L. Brodie, J. Mylopoulos and J. W. Schmidt, eds., *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pp. 87–117, Springer-Verlag, 1984.

[6] A. Borgida, "Description Logics in Data Management", *IEEE Trans. on Knowledge and Data Eng.*, vol. 7, no. 5, pp. 671–682, 1995.

[7] R. J. Brachman, R. E. Fikes, and H. J. Levesque, "KRYPTON: A Functional Approach to Knowledge Representation", *IEEE Computer*, vol. 16, no. 10, pp. 67–73, 1983.

[8] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick, "Living with CLASSIC: When and How to Use a KL-ONE-Like Language", J. F. Sowa, ed., *Principles of Semantic Networks*, pp. 401–456, Morgan Kaufmann, 1991.

[9] R. J. Brachman and J. G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science*, vol. 9, no. 2, pp. 171–216, 1985.

[10] W. Cyre, "A Requirements Sublanguage for Automated Analysis", *Int'l J. of Intelligent Systems*, vol. 10, no. 7, pp. 665–689, 1995.

[11] W. Cyre, "Capture, Integration, and Analysis of Digital System Requirements with Conceptual Graphs", *IEEE Trans. on Knowledge and Data Eng.*, vol. 9, no. 1, pp. 8–23, 1997.

[12] G. Ellis, "Compiling Conceptual Graphs", *IEEE Trans. on Knowledge and Data Eng.*, vol. 7, no. 1, pp. 68–81, 1995.

[13] B. C. Ghosh and V. Wuwongse, "Inference Systems for Conceptual Graph Programs", *Proc. 2nd Int'l Conf. Conceptual Structures (ICCS)*, College Park, Maryland, Lecture Notes in Artificial Intelligence, vol. 835, pp. 214–229, Springer-Verlag, 1994.

[14] B. C. Ghosh and V. Wuwongse, "A Direct Proof Procedure for Definite Conceptual Graph Programs", *Proc. 3rd Int'l Conf. Conceptual Structures (ICCS)*, Santa Cruz, California, Lecture Notes in Artificial Intelligence, vol. 954, pp. 158–172, Springer-Verlag, 1995.

[15] B. C. Ghosh and V. Wuwongse, "Conceptual Graph Programs and Their Declarative Semantics", *IEICE Trans. on Information and Systems*, vol. E78-D, no. 9, pp. 1208–1217, 1995.

[16] J. R. Hobbs, "Overview of the TACITUS Project", *Computational Linguistics*, vol. 12, no. 3, pp. 220–222, 1986.

[17] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages", *J. ACM*, vol. 42, pp. 741–843, 1995.

[18] G. Köstler, W. Kießling, H. Thöne, and U. Güntzer. "The Differential Fixpoint Operator with Subsumption", *Proc. 3rd Int'l Conf. Deductive and Object-Oriented Databases (DOOD)*, Phoenix, Arizona, Lecture Notes in Computer Science, vol. 760, pp. 35–48, Springer-Verlag. 1993.

[19] G. Köstler, W. Kießling, H. Thöne, and U. Güntzer, "Fixpoint Iteration with Subsumption in Deductive Databases", *J. Intelligent Information Systems*, vol. 4, no. 2, pp. 123–148, 1995.

[20] D. B. Lenat and R. V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*, Addison Wesley. 1990.

[21] J. W. Lloyd, *Foundations of Logic Programming*, second, extended edition, Springer-Verlag, 1987.

[22] R. Mac Gregor, "A Deductive Pattern Matcher", *Proc. 1988 National Conference on Artificial Intelligence (AAAI)*, pp. 403–408, Menlo Park, California, 1988.

[23] M. L. Mugnier and M. Chein, "Characterization and Algorithmic Recognition of Canonical Conceptual Graphs", *Proc. 5th Int'l Conf. Conceptual Structures (ICCS)*, Quebec City, Canada, Lecture Notes in Artificial Intelligence, vol. 699, pp. 294–311, Springer-Verlag, 1993.

[24] M. L. Mugnier and M. Chein, "Polynomial Algorithms for Projection and Matching", T. E. Nagle and H. D. Pfeiffer, eds., *Conceptual Structures: Theory and Implementation*, Lecture Notes in Computer Science, vol. 754, pp. 239–251, Springer-Verlag, 1993.

[25] M. L. Mugnier, "On Generalization/Specialization for Conceptual Graphs", *J. Experimental and Theoretical Artificial Intelligence*, vol. 7, pp. 325–344, 1995.

[26] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, ch. 8. Prentice Hall, 1995.

[27] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison Wesley, 1984.

[28] J. F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks/Cole Publishing, 2000.

[29] A. Taivalsaari, "On the Notion of Inheritance", *ACM Computing Surveys*, vol. 28, no. 3, pp. 438–479, 1996.

[30] D. H. D. Warren and F. C. N. Pereira, "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", *Computational Linguistics*, vol. 8, no. 3/4, pp. 110–122, 1982.

[31] W. A. Woods, "Understanding Subsumption and Taxonomy: A Framework for Progress", J. F. Sowa, ed., *Principles of Semantic Networks*, pp. 45–94, Morgan Kaufmann, 1991.

[32] W. A. Woods and J. G. Schmolze, "The KL-ONE Family", *Computers & Mathematics with Applications*, vol. 23, pp. 133–177, 1992.

[33] V. Wuwongse, C. Anutariya and E. Nantajeewarawat, "Reasoning about RDF Elements", *Proc. Int'l Joint Workshop on Digital Libraries (IJWDL)*, Bangkok, Thailand, Digital Libraries, No. 12, pp. 83–94, 1998.

[34] V. Wuwongse and B. C. Ghosh, "Towards Deductive Object-Oriented Databases Based on Conceptual Graphs", T. E. Nagle and H. D. Pfeiffer, eds., *Conceptual Structures: Theory and Implementation*, Lecture Notes in Computer Science, vol. 754, pp. 188–205, Springer-Verlag, 1993.

[35] V. Wuwongse and E. Nantajeewarawat, "Declarative Program Theory with Implicit Implication", *Proc. 4th Pacific Rim Int'l Conf. Artificial Intelligence (PRICAI)*, Cairns, Australia, Lecture Notes in Artificial Intelligence, vol. 1114, pp. 97–108, Springer-Verlag, 1996.

Output #3

An Argumentation Approach to Semantics of Declarative Programs with Defeasible Inheritance

Ekawit Nantajeewarawat[1] and Vilas Wuwongse[2]

[1] Information Technology Program,
Sirindhorn International Institute of Technology, Thammasat University,
P.O. Box 22, Thammasat-Rangsit Post Office, Pathumthani 12121, Thailand
ekawit@siit.tu.ac.th
[2] Computer Science and Information Management Program,
School of Advanced Technologies, Asian Institute of Technology,
P.O. Box 4, Klongluang, Pathumthani 12120, Thailand
vw@cs.ait.ac.th

Abstract. Inheritance is a characteristic reasoning mechanism in systems with taxonomic information. In rule-based deductive systems with inclusion polymorphism, inheritance can be captured in a natural way by means of typed substitution. However, with method overriding and multiple inheritance, it is well-known that inheritance is nonmonotonic and the semantics of inheritance becomes problematical. We present a general framework, based on Dung's abstract theory of argumentation, for developing a natural semantics for declarative programs with dynamic defeasible inheritance. We investigate the relationship between the presented semantics and Dobbie and Topor's perfect model (with overriding) semantics, and show that for inheritance-stratified programs, the two semantics coincide. The proposed semantics, nevertheless, still provides the correct skeptical meanings for non-inheritance-stratified programs, while the perfect model semantics fails to yield sensible meanings for them.

1 Introduction

One of the most salient features associated with generalization taxonomy is inheritance. In logic-based deduction systems which support inclusion polymorphism (or subtyping), inheritance can be captured in an intuitive way by means of typed substitutions. To illustrate this, suppose that tom is an individual of type student. Then, given a program clause:

$$C1: \ \mathsf{X{:}\,student[residence \rightarrow east\text{-}dorm]} \leftarrow \mathsf{X[lives\text{-}in \rightarrow rangsit\text{-}campus]}, \ \mathsf{X[sex \rightarrow male]},$$

which is intended to state that for any student X, if X lives in rangsit-campus and X is male, then X's residence place is east-dorm; one can obtain by the application of the typed substitution {X: student/tom} to $C1$ the ground clause:

P.S. Thiagarajan, R. Yap (Eds.): ASIAN'99, LNCS 1742, pp. 239-250, 1999.

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

The clause $C1$ can naturally be considered as a conditional definition of the method residence associated with the type (class[1]) student and the clause $G1$ as a definition of the same method for tom inherited from the type student.

However, when a method is supposed to return a unique value for an object, definitions of a method inherited from different types, tend to conflict. For example, suppose that tom is also an individual of type employee and a clause:

$C2$: X: employee[residence → west-flats] ← X[lives-in → rangsit-campus],
X[marital-status → married],

defining the method residence for an employee is also given. Then, the definition of residence for tom obtained from $C2$, *i.e.*,

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

conflicts with the previously inherited definition $G1$ when they are both applicable. In the presence of such conflicting definitions, the usual semantics of definite programs, *e.g.*, the minimal model semantics, does not provide satisfactory meanings for programs; for example, if a program has both $G1$ and $G2$ above as its ground instances, then, whenever its minimal model entails each atom in the antecedents of $G1$ and $G2$, it will entail the conflicting information that tom's residence place is east-dorm and is west-flats.

In order to provide appropriate meanings for programs with such conflicting inherited definitions, a different semantics that allows some ground clauses whose antecedents are satisfied to be inactive is needed. This paper applies Dung's theory of argumentation [6] to the development of such a semantics. To resolve inheritance conflicts, the proposed approach requires a binary relation on program ground clauses, called the *domination relation*, which determines among possibly conflicting definitions whether one is intended to defeat another. For example, with additional information that students who are also employees usually prefer the accommodation provided for employees, $G2$ is supposed to defeat $G1$. With such a domination relation, 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 method definitions are often conditional and may be inapplicable to certain objects, is advocated by [1]. In particular, with the possibility of overriding, when the definitions in the most

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

specific type are inapplicable, it is reasonable to try to apply those in a more general type.

In order to argue for the correctness and the generality of the proposed semantics in the presence of method overriding, its relationship to the perfect model (with overriding) semantics proposed by Dobbie and Topor [5] is investigated. The investigation reveals that these two semantics coincide for inheritance-stratified programs. Moreover, while the perfect model semantics fails to provide sensible 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 [4], 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 Dung's argumentation-theoretic foundation and DP theory. Section 3 describes the proposed semantics. Section 4 establishes the relationship between the proposed semantics and the perfect model (with overriding) semantics. Section 5 discusses other related works and summarizes the paper.

2 Preliminaries

2.1 Argumentation Framework

Based on the basic idea that a statement is believable if some argument supporting it can be defended successfully against attacking arguments, Dung has developed an abstract theory of argumentation [6] and demonstrated that many approaches to nonmonotonic reasoning in AI are special forms of argumentation. In this subsection, the basic concepts and results from this theory are recalled.

Definition 1. An *argumentation framework* is a pair $(AR, attacks)$, where AR is a set and $attacks$ is a binary relation on AR. □

In the sequel, let $AF = (AR, attacks)$ be an argumentation framework. The elements of AR are called *arguments*. An argument $a \in AR$ is said to *attack* an argument $b \in AR$, iff $(a, b) \in attacks$. Let $B \subseteq AR$. B is said to *attack* an argument $b \in AR$, iff some argument in B attacks b. An argument $a \in AR$ is said to be *acceptable* with respect to B, iff, for each $b \in AR$, if b attacks a, then B attacks b. B is said to be *conflict-free*, iff there do not exist arguments $a, b \in B$ such that a attacks b. B is said to be *admissible*, iff B is conflict-free and every argument in B is acceptable with respect to B.

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

Definition 2. A *preferred extension* of AF is a maximal (with respect to set inclusion) admissible subset of AR. A set $A \subseteq AR$ is called a *stable extension* of AF, iff A is conflict-free and A attacks every argument in $AR - A$. □

To define the grounded (skeptical) semantics of AF (Definition 3), the function F_{AF} on 2^{AR}, called the *characteristic function* of AF, is defined by:

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

Clearly, F_{AF} is monotonic (with respect to $\subseteq$), and, thus, has the least fixpoint.

Definition 3. The *grounded extension* of AF is the least fixpoint of F_{AF}. □

The next example illustrates the three kinds of extensions.

Example 1. Let $AF = (AR, attacks)$, where $AR = \{a, b, c, d, e\}$ and $attacks = \{(a, b), (b, c), (d, e), (e, d)\}$. Then, AF has two preferred extensions, i.e., $\{a, c, d\}$ and $\{a, c, e\}$, which are also stable extensions. As $F_{AF}(\emptyset) = \{a\}$ and $F^2_{AF}(\emptyset) = \{a, c\} = F^3_{AF}(\emptyset)$, the grounded extension of AF is $\{a, c\}$. □

Well-foundedness of an argumentation framework, recalled next, is a sufficient condition for the coincidence between the three kinds of extensions.

Definition 4. AF 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. *If AF is well-founded, then it has exactly one preferred extension and one stable extension, each of which is equal to its grounded extension.* □

2.2 DP Theory

DP theory [4] 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 5. 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. $\mathcal{G} \subseteq \mathcal{A}$. □

In the rest of this subsection, let $\Gamma = (\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu)$ be a specialization system. 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.

Let X be a subset of $\mathcal{A}$. A *definite clause* C *on* X is a formula of the form $(a \leftarrow b_1, \ldots, b_n)$, 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 $Body(C)$. When $n = 0$, C is called a *unit clause*. A definite clause C' is an *instance* of C, iff there exists $s \in \mathcal{S}$ such that s is applicable to $a, b_1, \ldots, b_n$ and $C' = ((\mu s)a \leftarrow (\mu s)b_1, \ldots, (\mu s)b_n)$. A definite clause on $\mathcal{G}$ is called a *ground clause*. A *declarative program* on Γ is a set of definite clauses on $\mathcal{A}$. Given a declarative program P on Γ, let $Gclause(P)$ denote the set of all ground instances of clauses in P. Conventional (definite) logic programs as well as typed logic programs can be viewed as declarative programs on some specialization systems.

An *interpretation* is defined as a subset of $\mathcal{G}$. Let I be an interpretation. If C is a definite clause on $\mathcal{G}$, then I is said to *satisfy* C iff $(head(C) \in I)$ or $(Body(C) \not\subseteq I)$. If C is a definite clause on $\mathcal{A}$, then I is said to *satisfy* C iff for every ground instance C' of C, I satisfies C'. I is a *model* of a declarative program P on Γ, iff I satisfies every definite clause in P. The meaning of P is defined as the *minimum model* of P, which is the intersection of all models of P.

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. 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.

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 minimum model of P is supported by at least one derivation tree of P.

Definition 6. 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.

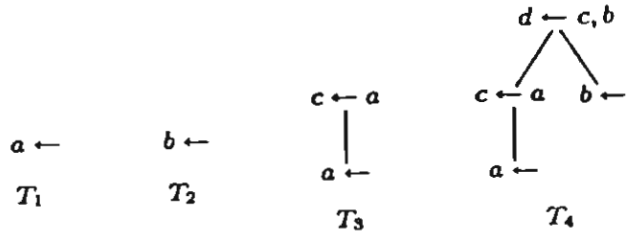

Fig. 1. The derivation trees of the program P_1.

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.
3. Nothing else is a derivation tree of P. □

Example 2. Let P_1 be a declarative program comprising the five ground clauses:

$$a \leftarrow \qquad b \leftarrow \qquad c \leftarrow a \qquad d \leftarrow c, b \qquad f \leftarrow e$$

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 sequel, the root of a derivation tree T will be denoted by $root(T)$. A derivation tree T will be regarded as an argument that supports the activation of the ground clause $root(T)$ (and, thus, supports the conclusion $head(root(T))$).

3.2 Grounded-Extension-Based Semantics

In order to define the meaning of P with respect to the domination relation, the program P will be transformed into an argumentation framework $AF_i(P)$, which 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 7. The argumentation framework $AF_i(P) = (AR, attacks)$ is defined as follows: AR is the set of all derivation trees of P, and for any $T, T' \in AR$, T attacks T', iff $root(T)$ dominates some node of T'. □

Example 3. Referring to the program P_1 of Example 2, 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 $AF_i(P_1) = (AR_{P_1}, attacks)$, where AR_{P_1} consists of the four derivation trees in Figure 1 and $attacks = \{(T_1, T_2), (T_1, T_4)\}$. (Note that T_1 attacks T_4 as the root of T_1 dominates the right leaf of T_4.) □

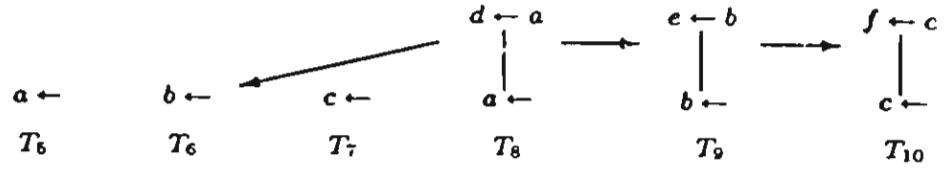

Fig. 2. The argumentation framework for the program P_2.

The meaning of P is now defined as the set of all conclusions which are supported by some arguments in the grounded extension of $AF_t(P)$.

Definition 8. The *grounded-extension-based meaning* of P, denoted by $\mathcal{M}_P^{GE}$, is defined as the set $\{head(root(T)) \mid T \in GE\}$, where GE is the grounded extension of $AF_t(P)$. □

Four examples illustrating the proposed semantics are given below.

Example 4. Consider $AF_t(P_1)$ of Example 3. Let F be the characteristic function of $AF_t(P_1)$. Clearly, $F(\emptyset) = \{T_1, T_3\} = F(F(\emptyset))$. Thus $F(\emptyset)$ is the grounded extension of $AF_t(P_1)$, and, then, $\mathcal{M}_{P_1}^{GE} = \{a, c\}$. □

Example 5. Let a declarative program P_2 comprise the six ground clauses:

$$a \leftarrow \qquad b \leftarrow \qquad c \leftarrow \qquad d \leftarrow a \qquad e \leftarrow b \qquad f \leftarrow c$$

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 $AF_t(P_2) = (AR_{P_2}, attacks)$, where AR_{P_2} consists of the six derivation trees shown in Figure 2 and $attacks = \{(T_8, T_6,), (T_8, T_9), (T_9, T_{10})\}$ as depicted by the darker arrows between the derivation trees in the figure. Let F be the characteristic function of $AF_t(P_2)$. Then $F(\emptyset) = \{T_5, T_7, T_8\}$, and $F^2(\emptyset) = \{T_5, T_7, T_8, T_{10}\} = F^3(\emptyset)$. So $\mathcal{M}_{P_2}^{GE} = \{a, c, d, f\}$. This example also illustrates dynamic conflict resolution, *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. □

Example 6. Refer to the clauses $C1, C2, G1$ and $G2$ given at the beginning of Section 1. Let tom belong both to type student and to type employee. Consider a program P_3 comprising $C1, C2$ and the following three clauses:

$C3$: tom[lives-in → rangsit-campus] ←
$C4$: tom[sex → male] ←
$C5$: tom[marital-status → married] ←

Assume, for simplicity, that $C1$ and $C2$ have $G1$ and $G2$, respectively, as their only ground instances. Suppose that students who are also employees prefer the accommodation provided for employees, and, then, that $G2$ dominates $G1$. Then,

it is simple to see that $\mathcal{M}_{P_3}^{CE}$ contains tom[residence → west-flats] but does not contain tom[residence → east-dorm], and yields the desired meaning of P_3.

To demonstrate dynamic conflict resolution, suppose next that the clause $C5$ is removed from P_3. Then, instead of containing tom[residence → west-flats], $\mathcal{M}_{P_3}^{CE}$ contains tom[residence → east-dorm]; and, it still provides the correct meaning of P_3 in this case. □

Example 7. This example illustrates method overriding. Let ait be an instance of type int(ernational)-school and int-school be a subtype of school. Let a program P_4 comprise 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_4 has only three ground clauses:

$G3$: ait[medium-of-teaching → thai] ← ait[located-in → thailand]
$G4$: ait[medium-of-teaching → english] ←
$G5$: ait[located-in → thailand] ←

Since int-school is more specific than school, $G4$ is supposed to override $G3$; therefore, let $G4$ dominate $G3$. It is readily seen that $\mathcal{M}_{P_4}^{CE}$ is the set consisting of the two atoms ait[located-in → thailand] and ait[medium-of-teaching → english], which is the expected meaning of P_4. □

4 Perfect Model (with Overriding) Semantics

Dobbie and Topor defined a deductive object-oriented language called Gulog [5], in which inheritance is realized through typed substitutions, and studied the interaction of deduction, inheritance and *overriding* in the context of this language. The declarative semantics for Gulog programs is based on Przymusinski's perfect model semantics for logic programs [11], but using the possibility of overriding instead of negation in defining a priority relationship between ground atoms. The perfect model (with overriding) semantics provides the correct meanings for inheritance-stratified programs. In order to investigate the relationship between this semantics and the grounded-extension-based semantics, the notions of inheritance stratification and perfect model will be reformulated in the framework of DP theory in Subsection 4.1. The relationship between the two kinds of semantics will then be discussed in Subsection 4.2.

4.1 Inheritance-Stratified Programs and Perfect Models

According to [5], a program is inheritance-stratified if there is no cycle in any definition of a method, i.e., a definition of a method does not depend on an inherited definition of the same method. More precisely:

Definition 9. A declarative program P on Γ is said to be *inheritance-stratified*, iff it is possible to decompose the interpretation domain $\mathcal{G}$ into disjoint sets, called *strata*, $G_0, G_1, \ldots, G_\gamma, \ldots$, where $\gamma < \delta$ and δ is a countable ordinal, such that the following conditions are all satisfied.

1. For each $C \in Gclause(P)$, if $head(C) \in G_\alpha$, then
   (a) for each $b \in Body(C)$, $b \in \bigcup_{\beta<\alpha} G_\beta$,
   (b) for each $C' \in Gclause(P)$ such that C' dominates C,
      i. $head(C') \in \bigcup_{\beta\leq\alpha} G_\beta$,
      ii. for each $b' \in Body(C')$, $b' \in \bigcup_{\beta<\alpha} G_\beta$.
2. There exists no infinite sequence $C_0, C_1, \ldots, C_n, \ldots$ of clauses in $Gclause(P)$ such that for each $i \geq 0$, C_{i+1} dominates C_i.

Any decomposition $\{G_0, G_1, \ldots, G_\gamma, \ldots\}$ of $\mathcal{G}$ satisfying the above conditions is called an *inheritance stratification* of P. □

An example of non-inheritance-stratified programs will be given in Subsection 4.2 (Example 8). The next theorem illuminates the coincidence between the grounded extension, preferred extension and stable extension of the argumentation framework for an inheritance-stratified program (see Theorem 1 in Subsection 2.1). Its proof can be found in the full version of this paper [10].

Theorem 2. *If P is inheritance-stratified, then $AF_\iota(P)$ is well-founded.* □

With overriding, not every ground clause of a program is expected to be satisfied by a *reasonable* model of that program. More precisely, a ground clause need not be satisfied if it is overridden by some ground clause whose premise is satisfied. This leads to the following notion of a model with overriding:

Definition 10. An interpretation I is a *model with overriding* (for short, *o-model*) of P, iff for each $C \in Gclause(P)$, either I satisfies C or there exists $C' \in Gclause(P)$ such that C' dominates C and $Body(C') \subseteq I$. □

A program may have more than one o-model. Following [5], priority relations between ground atoms are defined based on the possibility of overriding.

Definition 11. Priority relations $<_p$ and $\leq_p$ on $\mathcal{G}$ are defined as follows:

1. If $C \in Gclause(P)$, then
   (a) for each $b \in Body(C)$, $head(C) \leq_p b$,
   (b) for each $C' \in Gclause(P)$, if C' dominates C, then
      i. $head(C) \leq_p head(C')$,
      ii. for each $b' \in Body(C')$, $head(C) <_p b'$,
2. If $a \leq_p b$ and $b \leq_p c$, then $a \leq_p c$,
3. If $a \leq_p b$ and $b <_p c$ (respectively, $d <_p a$), then $a <_p c$ (respectively, $d <_p b$),
4. If $a <_p b$, then $a \leq_p b$,
5. Nothing else satisfies $<_p$ or $\leq_p$. □

A preference relationship among o-models will then be defined based on the priority relation $<_p$.

Definition 12. Let M and N be o-models of P. M is said to be *preferable* to N, in symbols, $M \ll N$, iff $M \neq N$ and for each $a \in M - N$, there exists $b \in N - M$ such that $a <_P b$. M is said to be a *perfect o-model* of P, iff there exists no o-model of P preferable to M. □

Every inheritance-stratified program P has exactly one perfect o-model,[2] denoted by $\mathcal{M}_P^{Perf}$, which provides the correct meaning of P with respect to method overriding.

4.2 Relationship between the Proposed Semantics and Perfect Model (with Overriding) Semantics

It is shown in the full version of this paper [10] that:

Theorem 3. *If P is inheritance-stratified and the domination relation is transitive, then $\mathcal{M}_P^{GS} = \mathcal{M}_P^{Perf}$.* □

It is important to note that since the domination due to method overriding is typically transitive, the transitivity requirement does not weaken Theorem 3.

For programs that are not inheritance-stratified, the perfect model semantics fails to provide their sensible meanings, while the proposed semantics still yields their correct skeptical meanings. (The skeptical approach to method resolution discards all conflicting definitions.) This is demonstrated by the next example.

Example 8. Let tom be an instance of type gr(aduate)-student and gr-student is a subtype of student. Consider the declarative program P_5 comprising the following five clauses:

$C6$: X:student[math-ability → good] ← X[math-grade → b]
$C7$: X:student[major → math] ← X[math-ability → good], X[favourite-subject → math]
$C8$: X:gr-student[math-ability → average] ← X[major → math], X[math-grade → b]
$C9$: tom[math-grade → b] ←
$C10$: tom[favourite-subject → math] ←

Without loss of generality, suppose for simplicity that $C6, C7$ and $C8$ have as their ground instances only the clauses $G6, G7$ and $G8$, given below, respectively:

$G6$: tom[math-ability → good] ← tom[math-grade → b]
$G7$: tom[major → math] ← tom[math-ability → good], tom[favourite-subject → math]
$G8$: tom[math-ability → average] ← tom[major → math], tom[math-grade → b]

[2] This result is analogous to and inspired by the corresponding result for inheritance-stratified Gulog programs [5]. Its proof is given completely in [9].

The ground clauses $G6$ and $G8$ are considered as definitions of the method math-ability inherited from the types student and gr-student, respectively. As gr-student is more specific than student, $G8$ is supposed to dominate $G6$. Then, every inheritance stratification of P_5 requires that the ground atom tom[major $\rightarrow$ math] must be in a stratum which is lower than the stratum containing it, which is a contradiction. Hence P_5 is not inheritance-stratified.

Observe that $G8$ dominates $G6$, but $G8$ also depends on $G6$; that is, the activation of $G6$ results in the activation of $G8$, which is supposed to override $G6$. Therefore, it is not reasonable to use any of them. As a consequence, none of the conclusions of $G6$, $G7$ and $G8$ should be derived. However, it can be shown that each o-model of P_5 contains both tom[major $\rightarrow$ math] and tom[math-ability $\rightarrow$ average]. So every o-model of P_5 does not serve as its reasonable meaning.

Now consider the proposed semantics. It is simple to see that $\mathcal{M}_{P_5}^{\mathrm{GE}}$ is the set {tom[math-grade $\rightarrow$ b], tom[favourite-subject $\rightarrow$ math]}, which is the correct skeptical meaning of P_5 (*i.e.*, the meaning obtained in the usual way after discarding the conflicting clauses $G6$ and $G8$). □

5 Related Works and Conclusions

Defeasible inheritance has been intensively studied in the context of inheritance networks [7,12,13]. Although the process of drawing conclusions from a set of defeasible hypotheses in inheritance networks is quite different from the process of deduction (as pointed out in [7]) and these works do not discuss dynamic method resolution, they do provide the presented approach with a foundation for determining the domination relation among ground clauses. A type hierarchy and a membership relation can be represented as a network, and the domination relation can then be determined based on the topological structure of the network. For example, if there exists a path from an object o through a type t to a type t' in the network, then it is natural to suppose that the ground method definitions for o inherited from t dominate those inherited from t'.

Besides [5], distinguished proposals that incorporate inheritance in the context of logic-based deduction systems include [1,2,3,8]. However, in [1] and [8], inheritance is realized by other means than typed substitution; *i.e.*, [1] captures inheritance by transforming subclass relationships into rules of the form $class(X) \leftarrow subclass(X)$, and [8] models inheritance as implicit implication on interpretation domains (called H-structures). [2] and [3] incorporate inheritance into unification algorithms but do not discuss nonmonotonic inheritance.

This paper studies the interaction of inheritance, realized by means of typed substitution, and deduction, and proposes a framework for discussing a declarative semantics for definite declarative programs with nonmonotonic inheritance. The framework uses a domination relation on program ground clauses, specifying their priority, as additional information for resolving conflicting method definitions. With a specified domination relation, a program is transformed into an argumentation framework which provides an appropriate structure for analyzing the interrelation between the intended deduction and domination. The meaning

of the program is defined based on the grounded extension of this argumentation framework. Method resolution in the framework is dynamic with respect to the applicability of methods. The paper not only shows that the proposed semantics and Dobbie and Topor's perfect model (with overriding) semantics [5] coincide for inheritance-stratified programs (Theorem 3), but also claims that the proposed semantics provides correct skeptical meanings for non-inheritance-stratified programs.

Acknowledgement

The proposed semantics is inspired by Phan Minh Dung. This work is supported in part by the Thailand Research Fund.

References

1. Abiteboul, S., Lausen, G., Uphoff, H., Waller, E.: Methods and Rules. In: Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data. ACM Press (1993) 32–41
2. Aït-Kaci, H., Nasr, R.: LOGIN: A Logic Programming Language with Built-in Inheritance. The Journal of Logic Programming **3** (1986) 185–215
3. Aït-Kaci, H., Podelski, A.: Towards a Meaning of Life. The Journal of Logic Programming **16** (1993) 195–234
4. Akama, K.: Declarative Semantics of Logic Programs on Parameterized Representation Systems. Advances in Software Science and Technology **5** (1993) 45–63
5. Dobbie, G., Topor, R.: On the Declarative and Procedural Semantics of Deductive Object-Oriented Systems. Journal of Intelligent Information Systems **4** (1995) 193–219
6. Dung, P.M.: On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and *N*-Person Games. Artificial Intelligence **77** (1995) 321–357
7. Horty, J.F., Thomason, R.H., Touretzky, D.S.: A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. Artificial Intelligence **42** (1990) 311–348
8. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the Association for Computing Machinery **42** (1995) 741–843
9. Nantajeewarawat, E.: An Axiomatic Framework for Deductive Object-Oriented Representation Systems Based-on Declarative Program Theory. PhD thesis, CS-97-7, Asian Institute of Technology, Bangkok, Thailand (1997)
10. Nantajeewarawat, E., Wuwongse, V.: Defeasible Inheritance Through Specialization. Technical Report, Computer Science and Information Management Program, Asian Institute of Technology, Bangkok, Thailand (1999)
11. Przymusinski, T.C.: On the Declarative Semantics of Deductive Databases and Logic Programs. In: Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1988) 193–216
12. Stein, L. A.: Resolving Ambiguity in Nonmonotonic Inheritance Hierarchies. Artificial Intelligence **55** (1992) 259–310
13. Touretzky, D.S.: The Mathematics of Inheritance. Morgan Kaufmann (1986)

Output #4

To appear in Proc. 1st Int. Conf. Electronic Commerce and Web Technologies (Lecture Notes in Computer Science, Springer-Verlag), 2000

Towards a Foundation for XML Document Databases

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

[1] Computer Science & Information Management Program,
Asian Institute of Technology, Pathumtani 12120, Thailand
{ca, vw}@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 Unversity,
Sapporo 060, Japan
akama@cims.hokudai.ac.jp

Abstract. This paper develops a theoretical framework for modeling and managing XML documents by employment of *Declarative Description* (*DD*) theory. In the framework, 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. An XML document - a set of XML elements - is simply modeled as an *XML declarative description* which consists of *object descriptions*, representing XML elements in the document, and *relationship descriptions*, specifying relationships among the elements as well as integrity constraints. DTDs and complex queries can also be expressed and evaluated.

1 Introduction

Modeling and managing XML [8] data have several challenges. An obvious difficulty is that XML is considered as a variation of *semistructured data* - data that may be varied, irregular and unrestricted to any particular schema. An XML document must only be *well-formed* but need not conform to a particular *Document Type Definition* (*DTD*). Mapping of semistructured data into well-defined and highly-structured schemas, such as those in the relational and object-oriented models, often requires a lot of efforts and frequent schema modifications. This difficulty has obstructed the use of relational and object-oriented approaches to XML data modeling. Therefore, development of an appropriate and efficient data model for XML documents has become an active research area. Major current models are based on *directed edge-labeled graphs* [7, 9, 10, 13, 16], *hedge automaton theory* [14, 15] and *functional programming* [12].

A *declarative description data model* for XML documents is developed by employment of *Declarative Description* (*DD*) theory [1–3], 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*. An appropriate *specialization system for XML elements* is formulated and a framework for their representation, computation and reasoning is constructed. XML elements defined in this paper can represent both explicit and implicit information through the employment of variables. Conventional XML elements are directly represented in the proposed model as ground (variable-free) XML elements, with no translation needed. An *XML declarative description* (*XML-DD*) comprises a set of *XML elements*, called *object descriptions* (*ODs*), and a (possibly empty) set of their relationships, called *relationship descriptions* (*RDs*). The meaning of such an XML-DD will not only yield all the explicit information, represented in terms of ODs, but will also include all the implicit information derivable by application of the RDs to the set of ODs, whence complex queries about both kind of information can be formulated and executed [6].

RDs not only represent relationships among XML elements, but can also be used to define *integrity constraints* that are important in a document, such as data integrity, *path and type constraints* [10]. Moreover, in order to restrict XML elements to only those that satisfy a given DTD, a simple and effective mechanism is to directly map the DTD into a corresponding set of RDs for checking the validity of an element with respect to the DTD [5].

Sect. 2 reviews major approaches to modeling semistructured/SGML/XML documents, Sect. 3 develops a declarative description data model for XML documents, Sect. 4 presents approaches to modeling XML documents and their DTDs, Sect. 5 outlines how to formulate and evaluate queries, and Sect. 6 draws conclusions and presents future research directions.

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

Three important approaches to modeling semistructured/SGML data before 1995, i.e., *traditional information retrieval*, *relational model* and *object-oriented approaches*, have been reviewed in [19]. This section reviews the more recent ones which are based on graphs, hedge automaton theory and functional programming.

In *graph-based models*, an XML document is mapped into a directed, edge-labeled graph [7, 9, 10, 13, 16] consisting of nodes and directed edges, which, respectively, represent XML elements in the document and relationships among the elements, e.g., element-subelement and referential relationships. Although a graph-based model provides an effective and straightforward way to handle XML documents, it exhibits a difficulty in restricting a document to a given DTD. The proposal [7], for instance, only provides a way to query XML documents but does not facilitate a means of representing the structure imposed by a DTD. A substantial extension to the model is required to overcome this difficulty. For example, by application of *first-order logic theory*, the proposal [10] has incorporated the ability to express *path and type constraints* for specification

of the document structure; the integration of these *two different formalisms* also results in an ability to reason about path constraints.

Employing *hedge automaton* theory [14] (aka. *tree automaton* and *forest automaton* theory), developed by using the basic ideas of *string automaton* theory, the proposals [15] have constructed an approach to formalizing XML documents and their DTDs. A *hedge* is a sequence of trees or, in XML terminology, a sequence of XML elements. An XML document is represented by a hedge and a set of documents conforming to a DTD by a *regular hedge language* (*RHL*), which can be described by a *regular hedge expression* (*RHE*) or a *regular hedge grammar* (*RHG*). By means of a *hedge automaton*, one can validate whether a document conforms to a given RHG (representing some particular DTD) or not.

A *functional programming approach* to modeling XML documents and formalizing operations upon them has been developed in the proposal [12] by introduction of the notion of *node* as its underlying data structure. An algebra for XML queries, expressed in terms of *list comprehensions* in the functional programming paradigm, has also been constructed. Using list comprehensions, various kinds of query operations, such as navigation, grouping and joins, can be expressed. However, this approach has considerable limitations as it does not possess an ability to model a DTD, whence a mechanism for verifying whether a document conforms to a given DTD or not is not readily devised.

3 Declarative Description Data Model for XML Documents

XML declarative description (*XML-DD*) theory, which has been developed by employment of *Declarative Description* (*DD*) theory [1–3] and serves as a data model for XML documents [4], is summarized.

In XML-DD theory, 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 similar form as XML elements except that they can carry variables. The XML expressions without variable will be precisely called *ground XML expressions* or XML elements, while those with variables *non-ground XML expressions*.

There are several kinds of variables useful for the expression of implicit information contained in XML expressions: *name-variables* (*N*-variables), *string-variables* (*S*-variables), *attribute-value-pair-variables* (*P*-variables), *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'`.

Intuitively, an *N*-variable will be instantiated to an element type or an attribute name, an *S*-variable to a string, a *P*-variable 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. 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:

i) rename variables,
ii) expand *P*- or *E*-variables into sequences of variables of their respective types,
iii) remove *P*-, *E*- or *I*-variables, and
iv) instantiate variables to some values corresponding to the variables' types.

Let $\mathcal{A}_X$ denote the set of all XML expressions, $\mathcal{G}_X$ the subset of $\mathcal{A}_X$ comprising all ground XML expressions in $\mathcal{A}_X$, $\mathcal{C}_X$ the set of basic specializations and $\nu_X : \mathcal{C}_X \rightarrow \mathit{partial\_map}(\mathcal{A}_X)$ the mapping from $\mathcal{C}_X$ to the set of all partial mappings on $\mathcal{A}_X$ which determines for each 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 $\nu : \mathcal{C} \rightarrow \mathit{partial\_map}(\mathcal{A})$ the mapping which determines for each basic specialization c in $\mathcal{C}$ the change of elements in $\mathcal{A}$ caused by c.

In the sequel, let $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$ be a *specialization system for XML expressions with flat sets*, where $\mathcal{S} = \mathcal{C}^*$ and $\mu : \mathcal{S} \rightarrow \mathit{partial\_map}(\mathcal{A})$ such that

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

Elements of $\mathcal{S}$ are called *specializations*. Note that when μ is clear in the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$.

The definition of *XML declarative description* together with its related concepts can be given in terms of $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$. An XML declarative description (simply referred to as an *XML-DD*) on Γ is a set of *descriptions*, each having the form

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

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 *set-of reference* on Γ. Such a description, if $n = 0$, is called an *object description* or an *OD*, and, if $n > 0$, a *relationship description* or an *RD*.

A *constraint* on Γ is a formula $q(a_1, \ldots, a_n)$, where q is a constraint predicate and a_i an element in $\mathcal{A}$. Given a ground constraint $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}$, its truth and falsity is assumed to be predetermined.

A *set-of reference* on Γ is a triple $r = \langle S, f_{x,a}, P \rangle$ of a set $S \in 2^{(\mathcal{A}_X \cup V)}$, a set-of function $f_{x,a}$, and an XML declarative description P, which will be called the referred description of r. Given $x, a \in \mathcal{A}_X$, a set-of function $f_{x,a}$ can be defined as follows: For each $X \in 2^{\mathcal{G}_X}$,

$$f_{x,a}(X) = \{x\theta \in \mathcal{G}_X \mid a\theta \in X, \theta \in \mathcal{C}_X^*\}. \tag{2}$$

In other words, for each $X \in 2^{\mathcal{G}_X}$, $x\theta \in f_{x,a}(X)$ iff there exists $\theta \in \mathcal{C}_X^*$ such that $a\theta$ and $x\theta$ are ground XML expressions in X and $\mathcal{G}_X$, respectively. Intuitively, a and x are used, respectively, to define the condition for the construction of a set and to determine the elements comprising that set, i.e., $x\theta \in f_{x,a}(X)$ iff

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

Fig. 1. An XML DTD example

$a\theta \in X$. The objects a and x will be referred to as *filter* and *constructor* objects, respectively.

Given a specialization $\theta \in \mathcal{S}$, application of θ to a constraint $q(a_1, \ldots, a_n)$ yields the constraint $q(a_1\theta, \ldots, a_n\theta)$, to a reference $\langle S, f_{x,a}, P\rangle$ the reference $\langle S, f_{x,a}, P\rangle\theta = \langle S\theta, f_{x,a}, P\rangle$ and to a description $(H \leftarrow B_1, B_2, \ldots, B_n)$ the description $(H\theta \leftarrow B_1\theta, B_2\theta, \ldots, B_n\theta)$. The head of a description D will be denoted by $head(D)$ and the set of all objects (XML expressions), constraints and references in the body of D by $object(D)$, $con(D)$ and $ref(D)$, respectively. Let $body(D) = object(D) \cup con(D) \cup ref(D)$.

Given an XML-DD P, its meaning, $\mathcal{M}(P)$ is the set of all the ground XML expressions that can be derived from the descriptions in P. Intuitively, given a description $D = (H \leftarrow B_1, B_2, \ldots, B_n)$ in P, for every $\theta \in \mathcal{S}$ that makes $B_1\theta, B_2\theta, \ldots, B_n\theta$ true with respect to the meaning of P, the expression $H\theta$ will be derived and included in the meaning of P.

4 Modeling XML Documents and DTDs

4.1 XML Document Modeling

A conventional XML element is represented directly as a ground XML expression in $\mathcal{G}_X$. A class of XML elements sharing 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, attribute-value pairs, subexpressions or nesting structures) shared by the elements in the class.

A collection of XML documents can be modeled by an XML-DD consisting of *ODs* and *RDs*. The meaning of such an XML-DD yields all the directly represented XML elements in the document collection, i.e., those expressed by ODs, together with all the derived ones, which may be restricted by constraints.

Example 1. Let P be an XML-DD 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 P comprises the following seven descriptions, denoted by $D_1 - D_7$:

D_1:
```
<Person ssn="99999" gender="Male">
      <Name>John Smith</Name>
      <BirthYear>1975</BirthYear>
      <Parent mother="55555"/>
</Person>        ←    .
```
D_2:
```
<Person ssn="55555" gender="Female">
      <Name>Mary Smith</Name>
      <BirthYear>1950</BirthYear>
      <Parent father="11111"/>
</Person>        ←    .
```
D_3:
```
<Person ssn="11111" gender="Male">
      <Name>Tom Black</Name>
      <BirthYear>1920</BirthYear>
</Person>        ←    .
```
D_4:
```
<Ancestor ancestor=$S:Father descendent=$S:Person/>
      ←    <Person ssn=$S:Person $P:PersonAttr>
                $E:Subexpression
                <Parent father=$S:Father $P:ParentAttr/>
           </Person>.
```
D_5:
```
<Ancestor ancestor=$S:Mother descendent=$S:Person/>
      ←    <Person ssn=$S:Person $P:PersonAttr>
                $E:Subexpression
                <Parent mother=$S:Mother $P:ParentAttr/>
           </Person>.
```
D_6:
```
<Ancestor ancestor=$S:Father descendent=$S:Desc/>
      ←    <Ancestor ancestor=$S:Anc descendent=$S:Desc/>,
           <Person ssn=$S:Ancestor $P:PersonAttr>
                $E:Subexpression
                <Parent father=$S:Father $P:ParentAttr/>
           </Person>.
```
D_7:
```
<Ancestor ancestor=$S:Mother descendent=$S:Desc/>
      ←    <Ancestor ancestor=$S:Anc descendent=$S:Desc/>,
           <Person ssn=$S:Anc $P:PersonAttr>
                $E:Subexpression
                <Parent mother=$S:Mother $P:ParentAttr/>
           </Person>.
```

Descriptions $D_1 - D_3$ represent Person elements in the document; Descriptions $D_4 - D_7$ derive ancestor relationships among the individuals in the collection. Descriptions D_4 and D_5 specify that both father and mother of an individual are ancestors of such individual. Descriptions D_6 and D_7 recursively specify that the father and the mother of an individual's ancestor 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. □

4.2 XML DTD Modeling

An XML DTD is represented, in the proposed approach, as an XML-DD comprising a set of RDs [5]. Such RDs, referred to as *DTD-RDs*, are obtained directly from translating each of the element type and attribute-list declarations contained in the DTD into a corresponding set of DTD-RDs and then combining these sets together.

The head expression of such a DTD-RD only imposes the general structure of its corresponding element type and merely specifies the valid pattern of the associated attribute list. 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 constraints and XML expressions in the DTD-RD's body. An XML expression contained in a DTD-RD's body will be further restricted by the DTD-RDs the head of which can be matched with that XML expression.

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

Example 2. This example demonstrates a translation of the DTD given in Fig. 1, which will be referred to as `myDTD`, into a corresponding set of DTD-RDs:

V_1:
```
<myDTD_Person>
      <Person ssn=$S:SSN gender=$S:Gender>
            <Name>$S:Name</Name>
            <BirthYear>$S:BirthYear</BirthYear>
            $E:Parent
      </Person>
</myDTD_Person>
      ←     <myDTD_Parent>
                  $E:Parent
            </myDTD_Parent>,
            IsMemberOf(<Value>$S:Gender</Value>,
                  {<Value>"Male"</Value>,
                   <Value>"Female"</Value>}).
```

V_2:
```
<myDTD_Parent>
      <Parent father=$S:FatherSSN $P:MotherAttr/>
</myDTD_Parent>
      ←     <myDTD_Parent>
                  <Parent $P:MotherAttr/>
            </myDTD_Parent>,
```

V_3:
```
<myDTD_Parent>
      <Parent mother=$S:MotherSSN/>
</myDTD_Parent>        ←     .
```

V_4:
```
<myDTD_Parent>
      <Parent/>
</myDTD_Parent>        ←     .
```

V_5: <myDTD_Parent>
</myDTD_Parent> ← .

Description V_1 imposes restrictions on the **Person** element. The head expression of V_1 specifies that every conforming **Person** element must contain **ssn** and **gender** attributes as well as **Name** and **BirthYear** elements as its first and second subelements, respectively. The only restriction on **Name** and **BirthYear** elements stating that their contents must be textual data is simply represented by the *S*-variables **$S:Name** and **$S:BirthYear**, respectively, and is defined within the restrictions on the **Person** element, i.e., within the head of V_1. The *E*-variable **$E:Parent** is defined such that, following the **Name** and **BirthYear** subelements, a **Person** element can optionally contain a **Parent** element. The **myDTD_Parent** element contained in the body of V_1 specifies that such **Parent** subelement will be further restricted by the descriptions the heads of which are **myDTD_Parent** expressions, i.e., descriptions $V_2 - V_5$.

The constraint **IsMemberOf** enforces that the value of the **gender** attribute, represented by **$S:Gender**, must be either **"Male"** or **"Female"**.

Moreover, it should be noted that since validation of *uniqueness* and *referential integrity constraints* defined by means of attributes of types ID and IDREF/IDREFS, respectively, requires additional concepts of *id* and *idref/idrefs references* [5] which are beyond the scope of this paper, this example omits validation of such constraints.

Descriptions $V_2 - V_5$ can be interpreted in a similar way as description V_1. □

5 Query Processing

As details of the query formulation and evaluation based on the proposed data model are available in [6], this section merely sketches the basic ideas.

A query is formalized as an XML-DD, comprising one or more RDs, called *query RDs*. Each query RD is written as a description D, where $head(D)$ describes the structure of the resulting XML elements, $object(D)$ represents some particular XML documents or XML elements to be selected, $con(D)$ describes selection criteria and $ref(D)$ constructs sets or groups of related XML elements to be used for computing summary information. This syntax intuitively separates a query into three parts: a *pattern*, a *filter* and a *constructor*, where the pattern is described by $object(D)$, the filter by $con(D)$ and $ref(D)$, and the constructor by $head(D)$. The five basic query operations [11, 17, 18]: *extraction*, *selection*, *combination*, *transformation* and *aggregation*, can be formulated [6].

Given an XML-DD P specifying a collection of XML documents together with their relationships, a query represented by an XML-DD Q is evaluated by transforming the XML-DD $(P \cup Q)$ successively until it becomes the XML-DD $(P \cup Q')$, where Q' consists of only *ground object descriptions*. In order to guarantee that the answers to a given query are always preserved, only *semantics-preserving transformations* or *equivalent transformations* [1–3] will be applied in every transformation step. The equivalent transformation is a new computational model which is considered to be more efficient than the inference in

the logic paradigm and the function evaluation in the functional programming paradigm. The *unfolding transformation*, a widely used program transformation in the conventional logic and functional programming, is a kind of equivalent transformation.

Example 3. Referring to XML-DD P of Example 1, a query which lists the names of all the John Smith's ancestors can be formulated as:

```
D:   <JohnAncestor>$S:Name<JohnAncestor/>
        ←   <Person ssn=$S:JohnSSN $P:JohnAttr>
                <Name>John Smith</Name>
                $E:JohnSubExp
            </Person>,
            <Ancestor ancestor=$S:Anc descendent=$S:JohnSSN/>,
            <Person ssn=$S:Anc $P:AncAttr>
                <Name>$S:Name</Name>
                $E:AncestorSubExp
            </Person>.
```

By means of unfolding transformation, XML-DD $(P \cup \{D\})$ can be successively transformed into XML-DD $(P \cup \{D', D''\})$, where

```
D':   <JohnAncestor>Mary Smith<JohnAncestor/>     ←   .
D'':  <JohnAncestor>Tom Black<JohnAncestor/>      ←   .
```

Since $\mathcal{M}(P \cup \{D\}) = \mathcal{M}(P \cup \{D', D''\})$ and the heads of D' and D'' are the only `JohnAncestor` elements in $\mathcal{M}(P \cup \{D', D''\})$, such elements are the only answers to the query. □

6 Conclusions

This paper has proposed and developed an expressive, declarative framework which can succinctly and uniformly model XML elements/documents, integrity constraints, element relationships, DTDs as well as formulate queries. By integrating the framework with an appropriate computational model, e.g., the Equivalent Transformation (ET), one will be able to efficiently manipulate and transform XML documents, evaluate queries, and validate XML data against some particular DTDs. The framework, therefore, provides a foundation for representation and computation of as well as reasoning with XML data.

A Web-based XML processor which can help demonstrate and evaluate the effectiveness of the proposed framework has been implemented using *ETC* - a compiler for programming in ET paradigm. The system has been tested against a small XML database and preliminary good performance is obtained; and a more thorough evaluation with a large collection of XML documents is underway. Other interesting future plans include development of indexing and query optimization techniques for XML document databases.

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. Tech. Report, 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. Anutariya, C., Wuwongse, V., Nantajeewarawat, E., Akama, K.: A Foundation for XML Document Databases: Data Model. Tech. Report, Computer Science and Information Management, Asian Institute of Technology, Thailand (1999)
5. Anutariya, C., Wuwongse, V., Nantajeewarawat, E., Akama, K.: A Foundation for XML Document Databases: DTD Modeling. Techn. Report, Computer Science and Information Management, Asian Institute of Technology, Thailand (1999)
6. Anutariya, C., Wuwongse, V., Nantajeewarawat, E., Akama, K.: A Foundation for XML Document Databases: Query Processing. Tech. Report, Computer Science and Information Management, Asian Institute of Technology, Thailand (1999)
7. Beech, D., Malhotra, A., Rys, M.: A Formal Data Model and Algebra for XML. W3C XML Query Working Group Note (1999)
8. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0. W3C Recommendation. (1998)
9. 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)
10. Buneman, P., Fan, W., Weinstein, S.: Interaction between Path and Type Constraints. Proc. ACM Symposium on Principles of Database Systems (1999)
11. Fankhauser, P., Marchiori, M., Robie, J.: XML Query Requirements, January 2000. W3C Working Draft, (2000)
12. Fernández, M., Siméon, J., Suciu, D., Wadler, P.: A Data Model and Algebra for XML Query. Draft Manuscript (1999)
13. Goldman, R., McHugh, J., Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Proc. 2nd Int. Workshop on the Web and Databases (WebDB'99), Pennsylvania (1999)
14. Murata, M.: Hedge Automata: A Formal Model for XML Schemata. Technical Report, Fuji Xerox Information Systems (1999)
15. Murata, M.: Transformation of Documents and Schemas by Patterns and Contextual Conditions. Principles of Document Processing '96. Lecture Notes in Computer Science, Vol. 1293 (1997)
16. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record, Vol. 26, No. 3 (1997) 54–66
17. Quass, D.: Ten Features Necessary for an XML Query Langauge. Proc. Query Languages Workshop (QI '98), Boston, MA, (1998)
18. Robie, J., Lapp, J., Schach, D.: XML Query Language (XQL). Proc. Query Languages Workshop (QL'98), Boston, MA, (1998)
19. 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