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*

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

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 $AF_t(P)/\sim$ is constructed, one can generate the logic program $P_{AF_t(P)/\sim}$ 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 $AF_t(P)/\sim$, and, thus, whether an atom belongs to $\mathcal{M}_P^{GE}$.

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 $C' \in \mathit{Gclause}(P)$ such that C' dominates C and $\mathit{Body}(C') \subseteq I$.
   □

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 $\mathcal{M}_P^{\mathrm{Perf}}$. 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 $C \in \mathit{Gclause}(P)$, then
   (a) for each $b \in \mathit{Body}(C)$, $\mathit{head}(C) \leq_p b$,
   (b) for each $C' \in \mathit{Gclause}(P)$, if C' dominates C, then
      i. $\mathit{head}(C) \leq_p \mathit{head}(C')$,
      ii. for each $b' \in \mathit{Body}(C')$, $\mathit{head}(C) <_p b'$,
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 $\mathcal{M}_P^{\mathrm{Perf}}$, and for every other o-model N of P, $\mathcal{M}_P^{\mathrm{Perf}} \ll N$.* □

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 $Body(C) \subseteq \mathcal{M}_P^{GE}$.* $\Box$

Proof Assume the contrary, i.e., there exists a clause C such that C dominates $root(T)$ and $Body(C) \subseteq \mathcal{M}_P^{GE}$, and let $Body(C) = \{b_1, \ldots, b_n\}$, where $n \geq 0$. Thus 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 *$\mathcal{M}_P^{GE}$ is a perfect o-model of P.* $\Box$

Proof Note first that, by Lemma 3, if $T \in GE$, then $Body(root(T)) \subseteq \mathcal{M}_P^{GE}$. It will now be shown that $\mathcal{M}_P^{GE}$ is an o-model of P. Let $C \in P$ such that $Body(C) \subseteq \mathcal{M}_P^{GE}$ and let there exist no clause C' such that C' dominates C and $Body(C') \subseteq \mathcal{M}_P^{GE}$. 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 $head(C) \in \mathcal{M}_P^{GE}$.

Next, it will be shown that $\mathcal{M}_P^{GE}$ is a perfect o-model of P. Let N be an o-model of P such that $N = \mathcal{M}_P^{GE}$, 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.

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

<u>Induction</u> $height(T) > 0$. 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, $\mathcal{M}_P^{GE} \subseteq N$, whence $N \ll \mathcal{M}_P^{GE}$. ■

The main result of this section is:

Theorem 6 $\mathcal{M}_P^{\text{GE}} = \mathcal{M}_P^{\text{Perf}}$. □

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:

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

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 $AF_\iota(P_7)$ is delineated in Figure 7. Note that, in the figure, T_3 attacks itself. Let F be the characteristic function of $AF_\iota(P_7)$. Then, as $F(\emptyset) = \{T_1\} = F(F(\emptyset))$, $\mathcal{M}_{P_7}^{\text{GE}}$ 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

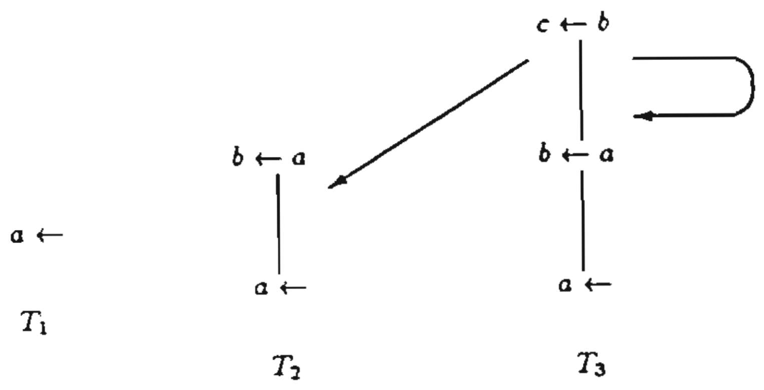


Figure 7: The argumentation framework for the program P_7.

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

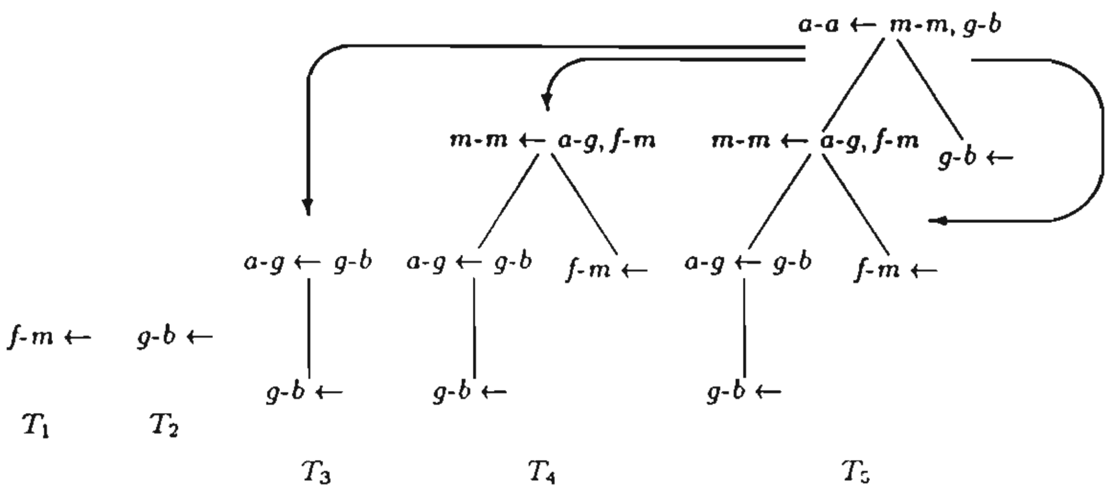
X: student[math-ability → good]	←	X[math-grade → b]
X: student[major → math]	←	X[math-ability → good],
		X[favourite-subject → math]
X: gr-student[math-ability → average]	←	X[major → math],
		X[math-grade → b]
tom[math-grade → b]	←	
tom[favourite-subject → math]	← .	

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



$a\text{-}a \leftarrow m\text{-}m, g\text{-}b$
$m\text{-}m \leftarrow a\text{-}g, f\text{-}m$
$m\text{-}m \leftarrow a\text{-}g, f\text{-}m$
$g\text{-}b \leftarrow$
$a\text{-}g \leftarrow g\text{-}b$
$a\text{-}g \leftarrow g\text{-}b$
$f\text{-}m \leftarrow$
$a\text{-}g \leftarrow g\text{-}b$
$f\text{-}m \leftarrow$
$f\text{-}m \leftarrow$
$g\text{-}b \leftarrow$
$g\text{-}b \leftarrow$
$g\text{-}b \leftarrow$
$g\text{-}b \leftarrow$
T_1
T_2
T_3
T_4
T_5

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 $AF_t(P_8)$ is depicted by Figure 8, where $a\text{-}g$, $a\text{-}a$, $m\text{-}m$, $g\text{-}b$ and $f\text{-}m$ 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 $\mathcal{M}^{GE}_{P_8}$ 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

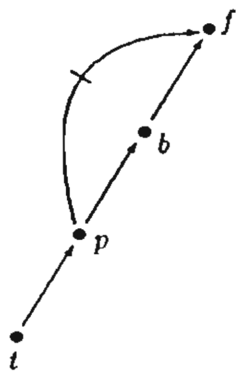


Figure 9: An example of inheritance networks.

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

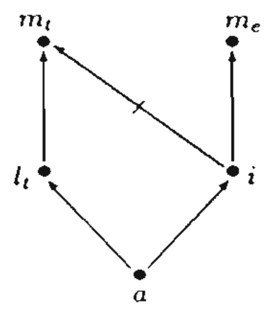


Figure 10: The definitions of medium-of-teaching represented by an inheritance network.

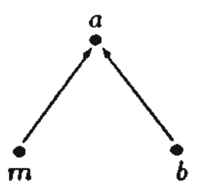


Figure 11: An unsatisfactory representation of the clause $a \leftarrow m, b$.

the individuals of type gr-student:

$$\begin{aligned} \mathsf{X}\colon \mathsf{gr\text{-}student}[\mathsf{math\text{-}ability} \rightarrow \mathsf{average}] \quad \leftarrow \quad & \mathsf{X}[\mathsf{major} \rightarrow \mathsf{math}], \\ & \mathsf{X}[\mathsf{math\text{-}grade} \rightarrow \mathsf{b}], \end{aligned}$$

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 Systems. 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].

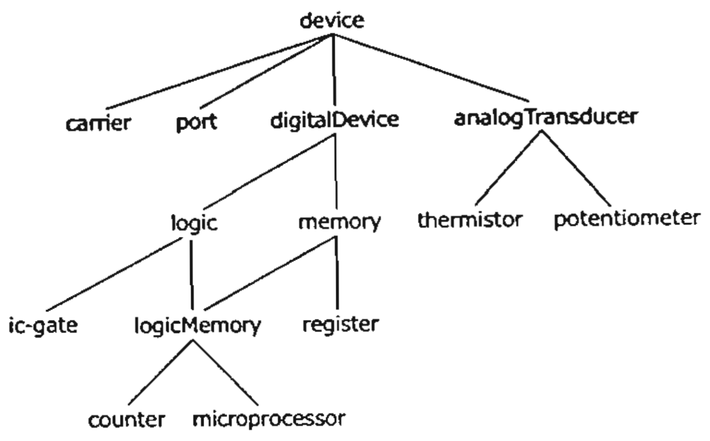


Figure 1: A partial hierarchy of concept types for digital systems

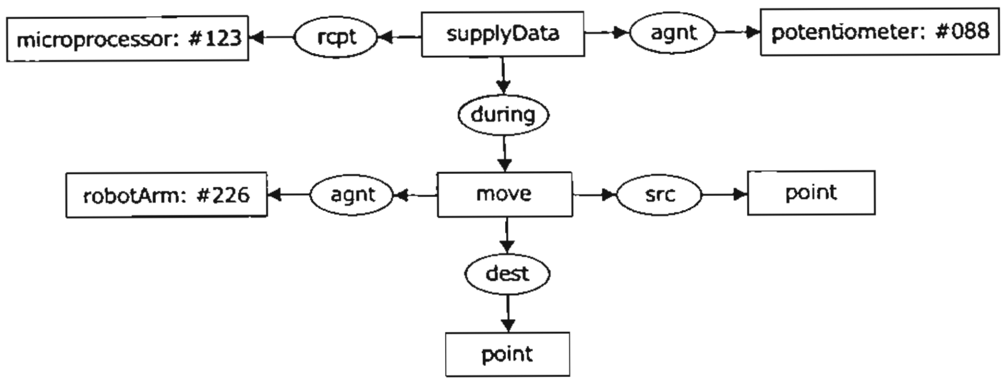


Figure 2: A conceptual graph $G1$

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

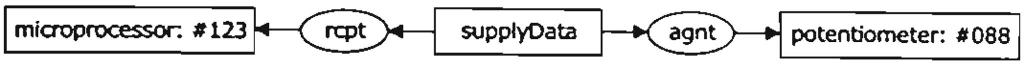


Figure 3: A conceptual graph $G2$

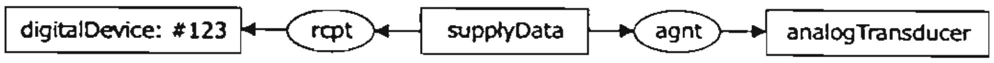


Figure 4: A conceptual graph $G3$

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.

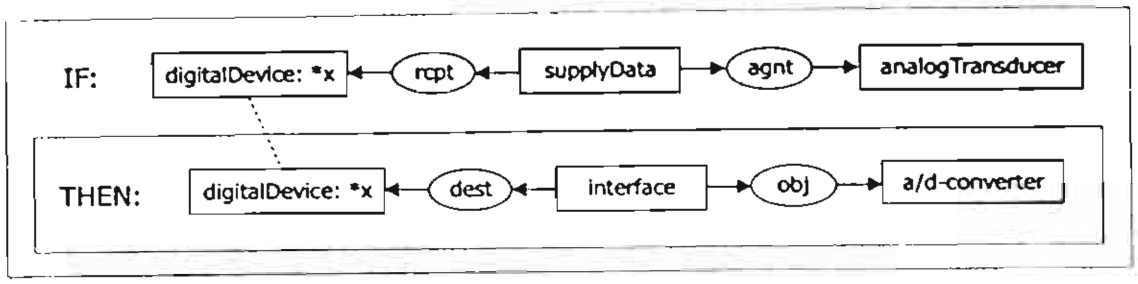


Figure 5: A conceptual graph C

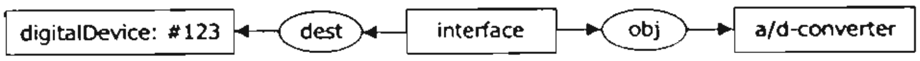


Figure 6: A conceptual graph $G4$

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.

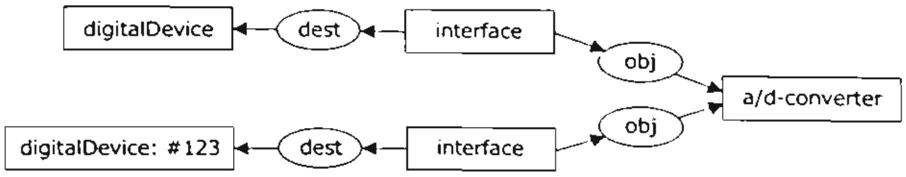


Figure 7: A redundant conceptual graph $G5$

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 suppresses 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:

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

where $n \geq 0$ and $a, b_1, \ldots, b_n$ are atoms in X. The atom a is denoted by $head(C)$ and the set $\{b_1, \ldots, b_n\}$ by $Body(C)$. A definite clause C such that $Body(C) = \emptyset$ is called *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 $X \subseteq \mathcal{G}$,

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

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

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

Some important properties of the mappings T_P and K_P follow:

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

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 $K_P(I) = I$.* $\Box$

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

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 $\perp$, 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 $\perp :: m$.

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. $\square$

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 $g' \in \bigcap_{j \in J} I_j$. ■

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

[microprocessor : #123] $\rightarrow$ (part) $\rightarrow$ [register : #001]

implicitly implies the atomic conceptual graph

[digitalDevice : #123] $\rightarrow$ (part) $\rightarrow$ [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 $\Rightarrow$ digitalValue]

implicitly implies the signature expression

counter[content $\Rightarrow$ 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

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

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

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 $K_P^{\sqsubseteq}$ 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 $K_P^{\sqsubseteq}$:

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 $\sqsubseteq$-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 $\mathcal{M}_P^{\sqsubseteq} = lfp(K_P^{\sqsubseteq})$. □

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:

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

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 $\{\mathcal{R}(X) \mid X \in 2^{\mathcal{G}}\}$. $\square$

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. $X \sqsubseteq Y \implies \mathcal{R}(X) \sqsubseteq \mathcal{R}(Y)$ & $\mathcal{E}(X) \subseteq \mathcal{E}(Y)$. $\square$

[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

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

■

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 $\sqsubseteq$-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 $(2^{\mathcal{G}}_{\mathcal{R}}/\sim, \sqsubseteq)$ according to Definition 11, *i.e.*, given $[X] \in 2^{\mathcal{G}}_{\mathcal{R}}/\sim$,

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

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

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

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 $2^{\mathcal{G}}_{\mathcal{R}}/\sim$,

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

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 $\Gamma_{\sqsubseteq}$. 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

$$\begin{aligned} X \sqsubseteq Y &\implies K_{\mathcal{P}}^{\xi}(X) \sqsubseteq K_{\mathcal{P}}^{\xi}(Y) \quad \text{(by Proposition 7)} \\ &\implies K_{\mathcal{P}}^{\sqsubseteq}(X) \sqsubseteq K_{\mathcal{P}}^{\sqsubseteq}(Y) \quad \text{(by Result 3 of Lemma 2).} \end{aligned}$$

■

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

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

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

Proposition 9 $(K_{\mathcal{P}}^{\xi})^{\sim} = (K_{\mathcal{P}}^{\sqsubseteq})^{\sim}$. □

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

As a result, given an equivalence class $[X]$ in $2^{\mathcal{G}}_{\mathcal{R}}/\sim$, $(K_{\mathcal{P}}^{\xi})^{\sim}([X])$ can be computed through the mapping $K_{\mathcal{P}}^{\sqsubseteq}$ by the equation:

$$(K_{\mathcal{P}}^{\xi})^{\sim}([X]) = (K_{\mathcal{P}}^{\sqsubseteq})^{\sim}([X]) = [\mathcal{R}(K_{\mathcal{P}}^{\sqsubseteq}(X))]. \quad (2)$$

Observe that, in the evaluation of $K_{\mathcal{P}}^{\sqsubseteq}(X)$, 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 $(K_{\mathcal{P}}^{\sqsubseteq})^{\sim}$ is equal to the minimal $\sqsubseteq$-closed model $\mathcal{M}_{\mathcal{P}}^{\sqsubseteq}$.

Theorem 10 $[\mathcal{R}(\mathcal{M}_{\mathcal{P}}^{\sqsubseteq})] = [\mathcal{R}(lfp(K_{\mathcal{P}}^{\xi}))] = lfp((K_{\mathcal{P}}^{\sqsubseteq})^{\sim})$. □

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 $K_P^{\sqsubseteq}$, the least fixpoint of which determines the proposed meaning $\mathcal{M}_P^{\sqsubseteq}$, as well as the operator $K_{\bar{P}}^{\sqsubseteq}$, 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 $\mathcal{M}_P^{\sqsubseteq}$ can be computed elegantly by means of the operator $(K_P^{\sqsubseteq})^{\sim}$, 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

```
? connect(X : device[family => ttl], Y : device[family => 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

```
connect(X : ic-gate, Y : ic-gate)   ←   send(X, Z : value, 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

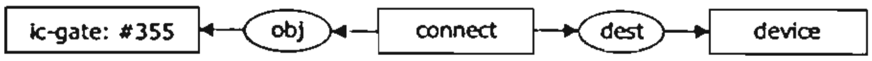


Figure 8: A conceptual graph G_6

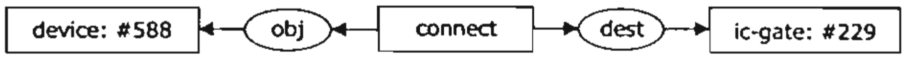


Figure 9: A conceptual graph G_7

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

send(X : ic-gate[family => ttl], Z : value, Y : ic-gate[family => 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,

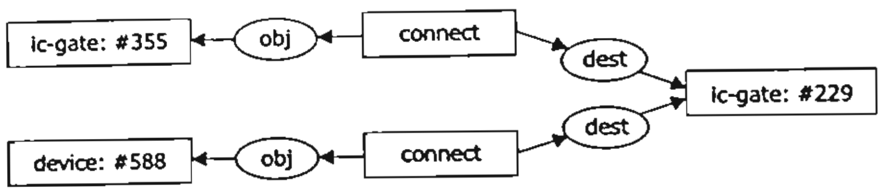


Figure 10: A conceptual graph G_8

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 $K_P^{\sqsubseteq}$ 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 $(K_{\overline{P}}^{\subseteq})^{\sim}$ 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 $\perp$. 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 $lub\{T^n(\perp) \mid n \geq 0\}$, 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:

$G1$: 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.

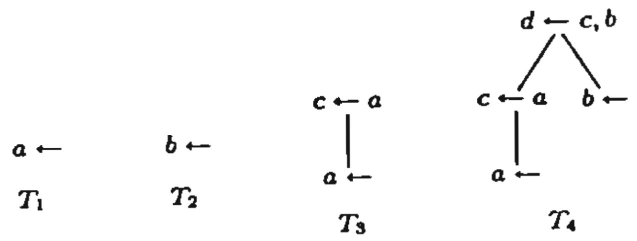


Fig. 1. The derivation trees of the program P_1.

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 $AF_t(P)$, 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 $AF_t(P) = (AR, attacks)$ 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 $AF_t(P_1) = (AR_{P_1}, attacks)$, 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.) □

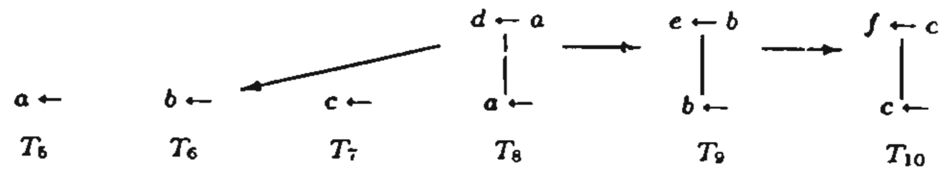


Fig. 2. The argumentation framework for the program P_2.

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

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

Four examples illustrating the proposed semantics are given below.

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

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 $AF_\iota(P_2) = (AR_{P_2}, attacks)$, 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 $AF_\iota(P_2)$. 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 $\mathcal{M}_{P_2}^{\text{GE}} = \{a, c, d, f\}$. 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 $\mathcal{M}_{P_3}^{\mathrm{GE}}$ 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], $\mathcal{M}_{P_3}^{\mathrm{GE}}$ 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 $\mathcal{M}_{P_4}^{\mathrm{GE}}$ 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 $C \in \mathit{Gclause}(P)$, if $\mathit{head}(C) \in G_\alpha$, then
   (a) for each $b \in \mathit{Body}(C)$, $b \in \bigcup_{\beta<\alpha} G_\beta$,
   (b) for each $C' \in \mathit{Gclause}(P)$ such that C' dominates C,
      i. $\mathit{head}(C') \in \bigcup_{\beta\leq\alpha} G_\beta$,
      ii. for each $b' \in \mathit{Body}(C')$, $b' \in \bigcup_{\beta<\alpha} G_\beta$.
2. There exists no infinite sequence $C_0, C_1, \ldots, C_n, \ldots$ of clauses in $\mathit{Gclause}(P)$ 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 $C \in \mathit{Gclause}(P)$, either I satisfies C or there exists $C' \in \mathit{Gclause}(P)$ such that C' dominates C and $\mathit{Body}(C') \subseteq I$. □

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 $C \in \mathit{Gclause}(P)$, then
   (a) for each $b \in \mathit{Body}(C)$, $\mathit{head}(C) \leq_p b$,
   (b) for each $C' \in \mathit{Gclause}(P)$, if C' dominates C, then
      i. $\mathit{head}(C) \leq_p \mathit{head}(C')$,
      ii. for each $b' \in \mathit{Body}(C')$, $\mathit{head}(C) <_p b'$,
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 $a <_r b$. 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 $\mathcal{M}^{\mathrm{GE}}_{P_5}$ 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 $\nu_X : \mathcal{C}_X \to partial\_map(\mathcal{A}_X)$ 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 $\nu : \mathcal{C} \to partial\_map(\mathcal{A})$ 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 $\mu : \mathcal{S} \to partial\_map(\mathcal{A})$ 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 $X \in 2^{\mathcal{G}_X}$,

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

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 $D_1 - D_7$:

```
D1:  <Person ssn="99999" gender="Male">
          <Name>John Smith</Name>
          <BirthYear>1975</BirthYear>
          <Parent mother="55555"/>
     </Person>         ←    .
D2:  <Person ssn="55555" gender="Female">
          <Name>Mary Smith</Name>
          <BirthYear>1950</BirthYear>
          <Parent father="11111"/>
     </Person>         ←    .
D3:  <Person ssn="11111" gender="Male">
          <Name>Tom Black</Name>
          <BirthYear>1920</BirthYear>
     </Person>         ←    .
D4:  <Ancestor ancestor=$S:Father descendent=$S:Person/>
          ←    <Person ssn=$S:Person $P:PersonAttr>
                    $E:Subexpression
                    <Parent father=$S:Father $P:ParentAttr/>
               </Person>.
D5:  <Ancestor ancestor=$S:Mother descendent=$S:Person/>
          ←    <Person ssn=$S:Person $P:PersonAttr>
                    $E:Subexpression
                    <Parent mother=$S:Mother $P:ParentAttr/>
               </Person>.
D6:  <Ancestor ancestor=$S:Father descendent=$S:Desc/>
          ←    <Ancestor ancestor=$S:Anc descendent=$S:Desc/>,
               <Person ssn=$S:Ancestor $P:PersonAttr>
                    $E:Subexpression
                    <Parent father=$S:Father $P:ParentAttr/>
               </Person>.
D7:  <Ancestor ancestor=$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:

```
V1:  <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>}).
V2:  <myDTD_Parent>
           <Parent father=$S:FatherSSN $P:MotherAttr/>
     </myDTD_Parent>
           ←     <myDTD_Parent>
                       <Parent $P:MotherAttr/>
                 </myDTD_Parent>,
V3:  <myDTD_Parent>
           <Parent mother=$S:MotherSSN/>
     </myDTD_Parent>         ←    .
V4:  <myDTD_Parent>
           <Parent/>
     </myDTD_Parent>         ←    .
```

```
V5:  <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