ELSEVIER

Artificial Intelligence 113 (1999) 247–268

Artificial Intelligence

www.elsevier.com/locate/artint

Research Note

Compiling defeasible inheritance networks to general logic programs

Jia-Huai You *, Xianchang Wang, Li Yan Yuan

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

Received 28 September 1998; received in revised form 30 July 1999

Abstract

We present a method of compiling arbitrary defeasible (inheritance) networks to general logic programs. We show a one-to-one correspondence between the credulous extensions of a defeasible network and the stable models of the translated logic program. This result leads to the discovery that an elegant query answering procedure for Horty's credulous extensions had long existed: the abductive proof procedure formulated by Eshghi and Kowalski for general logic programs is sound and complete for acyclic defeasible networks under the proposed translation. Since the translation is faithful to the commonly accepted notion of specificity, it leads to a novel transformational approach: any semantics defined for general logic programs yields an extension semantics for networks, and any query answering procedure developed for general logic programs can be used to answer queries for networks under the same semantics. This approach also yields new insights into the difficulties confronting path-based formalisms. Essentially, reasoning with logic programs overcomes the difficulty with path-based formalisms in dealing with the interactions of cascaded effects of link sequences, possibly compounded by preemption of paths. This difficulty has been particularly evident in the past in trying to understand the meaning of the networks that involve cycles, both semantically and proof-theoretically, and in the "directly skeptical" approach to defeasible inheritance.

Keywords: Defeasible inheritance; Logic programming; Nonmonotonic reasoning

1. Introduction

Inheritance networks were formalized to deal with reasoning in problems where taxonomic information is naturally and readily available. When placed in the context of

* Corresponding author. Email: you@cs.ualberta.ca.

0004-3702/99/$ – see front matter
PII: S0004-3702(99)00055-7

common sense knowledge, such networks are called defeasible networks, since their links may be defeated when contrary yet more specific information is present.

An early approach, which has been called *indirect* or *translational*, is based on some intuitive representations of a network in a nonmonotonic logic. Thus, the meaning of a network can be understood, indirectly, by the meaning associated with the representing theory in the logic (see, e.g., [5,7–9,12,18]).

On the other hand, Touretzky [21] shows that a semantic theory of defeasible networks can be defined entirely in the network language itself in terms of nodes, links, and paths. These paths are like arguments conducted over a given net. This approach has been called *path-based*, and the semantic theories developed this way have been called *direct theories*. The approach is further pursued by a number of researchers, and the paper by Horty [10] provides a detailed review of the major developments. The recent investigations by Morgenstern and her colleagues [15,16] have shown that direct theories developed this way are highly relevant in some industrial applications.

By a *semantics* or a *semantic theory*, we mean a machine independent account of the meaning of the networks. In logic programming, semantics have been defined and characterized in terms of model theories, fixpoints, argumentation, and abduction.

Despite some successes of direct theories, important questions remain to be answered. The most pressing one is the relationships between defeasible networks and general nonmonotonic formalisms. Although a significant body of knowledge has been accumulated providing us with a good understanding of general nonmonotonic formalisms, little is known about how path-based reasoning is related to other forms of nonmonotonic reasoning. For this, Horty raises the question of whether it is possible, and if yes, how to specify the consequences of a network by interpreting it in some more standard nonmonotonic formalism [10].

Another question is about query answering for defeasible networks. Although Horty's notion of credulous extension has been considered some kind of standard in credulous reasoning with defeasible inheritance, to our knowledge, there has been no goal-oriented proof procedure that soundly and completely answers a query of whether a property holds for an object in a credulous extension (though there are procedures formulated under a different notion of extension [20], and practical procedures designed for industrial applications [14,15]). The lack of such a procedure is another indication that our understanding of defeasible inheritance has not been satisfactory. In contrast, at least in logic programming proof theories and procedures have been investigated substantially, in many cases with systems built.

In this paper we show that general logic programming is an adequate representation language for defeasible networks under Horty's notion of credulous extension. This is achieved by a faithful yet tractable translation from networks to general logic programs. Furthermore, we show that the Eshghi–Kowalski abductive procedure is sound and complete for the class of general logic programs translated from acyclic networks. This result shows that a query answering procedure for Horty's credulous extensions had long existed in the form of reasoning with general logic programs.

The compilation method yields yet another indirect approach to the semantics of defeasible networks. That is, the meaning of a network may be understood by a commonly accepted semantics of its translated logic program. This differs from the previous indirect

approach in that the translation here is faithful to the standard notion of specificity for *any* network. In this way we obtain new semantics for defeasible networks that are based on the same notion of specificity and defined for all networks. These semantics include the well-founded model semantics [23] and the regular model semantics [26]. These two semantics possess sound and complete backward chaining proof procedures which can be adopted directly for query answering for networks. In addition, it provides us a way to understand some of the problems that arise in path-based formalisms; e.g., the difficulty to deal with cyclic nets both semantically and proof-theoretically, and the troublesome *zombie path problem* (that a skeptical conclusion is only a credulous conclusion because it is not in all reasonable credulous extensions [13]) in Horty et al.'s skeptical extension. These difficulties are essentially due to the lack of a simple and eloquent way to articulate how reasoning with some paths may be affected by reasoning with others. In the translational approach, since these effects are embedded in the translated logic programs, their interactions are dealt with automatically by reasoning with logic programs.

The next section reviews the background for defeasible networks and the semantics for general logic programs. Section 3 shows a faithful translation from networks to programs. In Section 4 we compare the translational approach with Horty et al.'s direct approach, and discuss how the semantics of networks, including those that do not possess any credulous extensions, may be understood in terms of the semantics of the translated logic programs. We conclude the paper in Section 5.

The work presented here includes a nontrivial improvement over our earlier work in [24] where a tractable transformation from defeasible nets to default theories is presented. The work also differs substantially from the one by Dung and Son [3] where an exponential algorithm is given to transform acyclic defeasible nets to extended logic programs (in the case of cyclic nets, their transformation could yield an infinite program). The compilation method presented in this paper shows that the language of general logic programming constitutes an adequate representation language for reasoning with defeasible inheritance. It is essentially this result that allows us to adopt proof procedures developed for general logic programs for query answering for networks.

2. Background

In this section we recall the definitions of defeasible networks and review the semantics of general logic programs.

2.1. Defeasible networks

A defeasible (inheritance) network Γ is a tuple $\langle N, E\rangle$ where N is a finite set of nodes and E a finite collection of positive and negative links between nodes. If x and y are nodes then $x.y$ (respectively, $x.\neg y$) denotes a positive (respectively, negative) *link* from x to y where x is called the *root* and y is called the *head*. The dot between two nodes may be omitted when no confusion arises. Links are usually denoted by $l, l', l_1, \ldots$. Nodes are divided into two disjoint classes: *object nodes* which are denoted by $a, b, \ldots$, and *property nodes* (also called *class nodes* or *category nodes*) which are denoted by $p, q, \ldots$, etc. We

assume that an object node can only be used as a root node. A link is called an *object link* if its root is an object node.

A *positive path* of Γ is a path $l_1.l_2.\cdots.l_n$ where all the links are positive. A *negative path* is a path $l_1.l_2.\cdots.l_n$ where except the last link all the other links are positive. A path $l_1.\cdots.l_n$ where l_i's root node is x_i and l_i's head node is x_{i+1} may also be denoted by $x_1.\cdots.x_{n+1}$ when it is a positive path, and by $x_1.\cdots.x_n.\neg x_{n+1}$ when it is a negative path. The dots between links and nodes may be omitted if no confusion arises. For convenience, we may denote a middle segment of a path by δ, σ, τ, etc. Thus we may write $x_1\delta x_{n+1}$, $x_1\sigma\neg x_{n+1}$, etc. If a middle segment τ is empty, $x\tau y$ simply denotes the link $x.y$.

A *general path* is a link sequence like a path, except that it may contain (possibly more than one) negative links anywhere. A net is said to be *acyclic* just in case it contains no general path whose initial node is identical with its end node.

We now give the off-path, forward chaining credulous extension of Horty [10].

Definition 2.1 (*Path constructibility and conflict in a path set*). Suppose Γ is a defeasible network, Φ is a path set of Γ. A path σ of Γ is *constructible* in Φ iff σ is an object link or $\sigma = x_1\sigma_1 x_n x_{n+1}$ (respectively, $\sigma = x_1\sigma_1 x_n\neg x_{n+1}$), $x_n x_{n+1}$ is a link in Γ, and $x_1\sigma_1 x_n \in \Phi$. A path $\sigma = x\tau y$ (respectively, $\sigma = x\tau\neg y$) of Γ is *conflicting* in Φ iff $\sigma \in \Phi$ and there is a path $\sigma' = x\tau'\neg y \in \Phi$ (respectively, $\sigma' = x\tau' y \in \Phi$).

Definition 2.2 (*Off-path preemption*). A positive path $r\sigma u.s$ (respectively, a negative path $r\sigma u.\neg s$) is *preempted* in Φ (see Fig. 1) iff $r\sigma u \in \Phi$ and there is a node v such that

(i) $v.\neg s \in \Gamma$ (respectively, $v.s \in \Gamma$) and
(ii) either $r = v$ or there is a path $r\tau_1 v\tau_2 u \in \Phi$.

It is "off-path" because there is no requirement that the preempting node v should lie on the initial segment of the path it preempts. Thus, under the specified conditions, not only a path from r to s via v and u is preempted, but a path from r to s via u but not v is also preempted.

Definition 2.3 (*Defeasible inheritability*). A path σ (either positive or negative) is *defeasibly inheritable* in Φ, written as $\Phi \vdash_d \sigma$, iff all of the following conditions are satisfied:

(i) σ is constructible in Φ;
(ii) σ is not conflicting in Φ;
(iii) σ is not preempted in Φ.

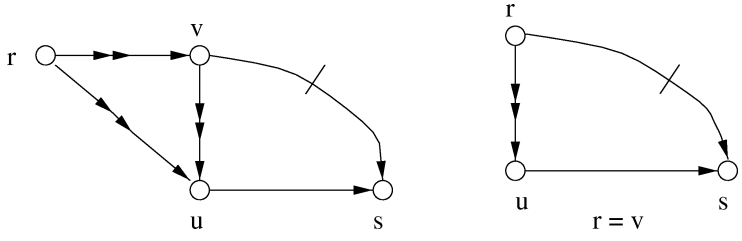

Fig. 1. Preemption is about specificity (multiple arrows denote a positive path).

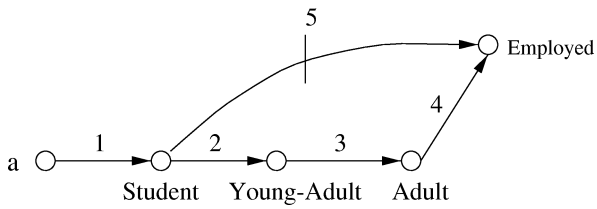

Fig. 2. Students are typically not employed.

Definition 2.4 (*Credulous extension*). A set Φ of paths is a *credulous extension* of a net Γ iff $\Phi = \{\sigma\colon\ \Phi \vdash_d \sigma\}$.

For example, the network in Fig. 2 possesses a unique credulous extension which is $\{1, 1.2, 1.2.3, 1.5\}$.

An alternative notion of preemption is that of *on-path* preemption. Suppose, for example, the net in Fig. 2 had an additional link from a to *YoungAdult*. On-path preemption would allow the conclusion that a is employed in one extension, as it bypasses the node *Student* which leads to *not Employed*. This seems to violate the principle of specificity in defeasible inheritance: that a is a young adult is by virtue of a being a student and students being young adults. For this reason, almost all subsequent work on defeasible inheritance adopt off-path preemption (see [15]).

There are two technical differences between the definitions given here and those of Horty.

First, any link in Horty's definition is defeasibly inheritable automatically. This allows a pair of contradictory links $a.p$ and $a.\neg p$ to be included in any credulous extension. These networks are said to be *inconsistent*. Our treatment follows that of Touretzky [21] which splits contradictory links into different extensions. A technical merit of this treatment is that contradictory links have the same effect as contradictory paths (such as $a.p.w$ and $a.q.\neg w$), and there is no need to identify inconsistent networks since all networks are consistent under this definition.

The second difference lies in the definition of constructibility that only permits paths from an object node. In Horty's any connected links could form a constructible path; e.g., in Fig. 2 the path 2.3 is a constructible path. This allows subtype relations (such as students are adults) to be included in an extension without relying on the presence of any particular object. Our choice aims at simplifying the presentation of the translation from networks to logic programs. Generalizations to accommodate Horty's notion of constructibility are possible. We will discuss such a generalization in the next section after the translation is introduced.

2.2. Semantics of general logic programs

In the past few years three semantics have become some kind of standard ones for general logic programs, the stable model semantics [6], the well-founded model semantics [23], and the regular model semantics [26]. The stable models are 2-valued while the well-founded and regular models are 3-valued. There are a variety of ways to define

these semantics [27]. The primary interest of this paper is in the stable model semantics. However, the other two semantics will also be mentioned for illustrating how the meaning of a network may be understood via the translational approach.

Let L be a first order language.

A *general logic program* is a collection of clauses of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, \mathbf{not}\ \gamma_1, \ldots, \mathbf{not}\ \gamma_n \tag{1}$$

where α, β_i's, and γ_i's are atoms of L. **not** γ_i's are called *default negations*.

For the objective of this paper it is sufficient to consider ground programs, which are instantiated from an underlying domain of elements. In the sequel, when there is no confusion the underlying language L is assumed to be ground.

A model M of a program P, be it either 2-valued or 3-valued, can be described by a set of default negations S; the atomic counterpart of a default negation in S is a false atom and any atom that is derivable from $P \cup S$ is a true atom. In the case of 3-valued models, the rest of the atoms are said to be *undefined*. In the sequel, we may represent a model by a set of default negations.

First let us define a function $\mathcal{F}_P$ over sets S of default negations as

$$\mathcal{F}_P(S) = \{\mathbf{not}\ \phi\colon\ \phi \text{ is an atom in } L \text{ such that } P \cup S \nvdash \phi\}$$

where $\vdash$ is the standard propositional derivation relation with each default negation **not** ϕ being treated as a named atom **not** $\_\phi$.

Given a set of atoms S, we use the notation $S_{\mathbf{not}} = \{\mathbf{not}\ \phi\colon\ \phi \notin S\}$.

Definition 2.5. A stable model M of a program P is a set of atoms such that $M_{\mathbf{not}}$ is a fixpoint of the function $\mathcal{F}_P$; i.e., $\mathcal{F}_P(M_{\mathbf{not}}) = M_{\mathbf{not}}$.

The well-founded model and regular models are certain fixpoints of the function that applies $\mathcal{F}_P$ twice, denoted $\mathcal{F}_P^2$.

It can be verified easily that $\mathcal{F}_P^2$ is a monotonic function, i.e., $S_1 \subseteq S_2$ implies $\mathcal{F}_P^2(S_1) \subseteq \mathcal{F}_P^2(S_2)$. Hence according to the fixpoint theory, commonly referred to as the Tarski–Knaster fixpoint theorem, $\mathcal{F}_P^2$ possesses a least fixpoint, a maximum fixpoint, and possibly some others over the domain of a complete lattice (in this case the set of all subsets of default negations). These fixpoints have been called *alternating fixpoints* [22].

Definition 2.6. Let P be a general logic program. The *well-founded model* of P is defined by the least fixpoint of $\mathcal{F}_P^2$. The *regular models* of P are maximal fixpoints M of $\mathcal{F}_P^2$ satisfying $M \subseteq \mathcal{F}_P(M)$.

Since a fixpoint E of $\mathcal{F}_P$ is a maximal fixpoint of $\mathcal{F}_P^2$ satisfying $E \subseteq \mathcal{F}_P(E)$, a stable model is a regular model. But the reverse is generally not true.

The well-founded semantics can be computed tractably. Though the regular semantics is not tractable, it possesses an elegant query answering procedure, the so called Eshghi–Kowalski abductive proof procedure [2,4]. The procedure is sound and complete (with positive loop checking) for finite, ground programs: A ground atom ϕ is true in a regular model iff there is a refutation for the goal $\leftarrow \phi$ by the Eshghi–Kowalski procedure.

3. From networks to general logic programs

We are interested in a semantics preserving translation from defeasible networks to logic programs: a path from an object node a to a property node p is in a credulous extension of a network if and only if $p(a)$ is true in a stable model of its translation.

We first explain informally the key technicalities that are adopted in the translation. One of our goals is to represent the paths in a credulous extension. Since the number of paths in an extension may be exponential to the number of nodes and links of the network, one should not attempt to represent paths by enumerating them. Paths can be represented by links. Since a link in a network may be used by more than one object, the validity of a link must be associated with an object. Hence we need to represent in a credulous extension the fact whether a link is valid and used in forming a path from an object node.

A link being invalidated may be due to

(i) a path being preempted (thus the last link on it for the concerned object should be removed), and

(ii) the necessity of removing it in order to preserve consistency for the underlying extension.

Hence we use a predicate $in_{p.q}(x)$ to represent that the link $p.q$ is valid and used in a path for object x in the underlying extension; i.e., it is *in* a path from object x. For example, the paths in the credulous extension $\{1, 1.2, 1.2.3, 1.5\}$ of the net in Fig. 2 are represented by the atoms $in_1(a)$, $in_2(a)$, $in_3(a)$, and $in_5(a)$ in the intended model of the translated logic program.

A translation to general logic programs is possible only if the negation of an atom is representable syntactically by an atom. For this purpose, we use $\widehat{p}(x)$ to denote the "negation" of atom $p(x)$. Specifically, given a link l, $\widehat{in_l}(x)$ means that l is removed in the paths from object x. This will be used to implement preemption of paths. For the example at hand, the atom $\widehat{in_4}(a)$ (meaning that the link 4 is removed for object a, effectively preempting the path 1.2.3.4) should also be included in the intended model.

Continuing with this example, suppose there is a link from an object node b to the class node *Adult*. Then the following additional links would be included: $in_{b.Ad}(b)$ and $in_4(b)$. Note that in this case link 4 (*Adult.Employed*) is removed in the path from a but not in the path from b.

For the presentation purposes, we give the translation in two steps: in one individual links are mapped to program clauses, and in the other preemption is realized.

Definition 3.1 (*Translation of links*). Let Γ be an arbitrary defeasible net. Each positive (nonobject) link $p.q$ of Γ is translated to two program clauses with the same body:

$$q(x) \leftarrow p(x), \textbf{not } \widehat{q}(x), \textbf{not } \widehat{in_{p.q}}(x)$$

$$in_{p.q}(x) \leftarrow p(x), \textbf{not } \widehat{q}(x), \textbf{not } \widehat{in_{p.q}}(x)$$

Each negative (nonobject) link $p.\neg q$ of Γ is translated to two program clauses with the same body:

$$\widehat{q}(x) \leftarrow p(x), \textbf{not } q(x), \textbf{not } \widehat{in_{p.\neg q}}(x)$$

$$in_{p.\neg q}(x) \leftarrow p(x), \textbf{not } q(x), \textbf{not } \widehat{in_{p.\neg q}}(x)$$

A positive object link $a.q$ is translated to

$$q(a) \leftarrow \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{a.q}}(a)$$

$$in_{a.q}(a) \leftarrow \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{a.q}}(a)$$

and a negative object link $a.\neg q$ is translated to

$$\widehat{q}(a) \leftarrow \textbf{not } q(a), \textbf{not } \widehat{in_{a.\neg q}}(a)$$

$$in_{a.\neg q}(a) \leftarrow \textbf{not } q(a), \textbf{not } \widehat{in_{a.\neg q}}(a)$$

In this translation, each nonobject link $p.q$ (similarly for $p.\neg q$) is mapped to two clauses with the same body which requires three conditions to be satisfied:

(1) $p(x)$ is already satisfied;
(2) it is consistent to conclude $q(x)$; and
(3) the link $p.q$ is not removed for x due to preemption.

The importance of (1) is obvious. Without (2), consistency cannot be guaranteed; e.g., in the case where a net consists of $a.p$ and $a.\neg p$. A derivation of $\widehat{in_{u.v}}(x)$ means that the link in a path from object x, which is represented by $in_{u.v}(x)$, is removed due to preemption. This part will be given shortly in the second part of our translation.

For object links, apparently (1) is not needed.

Example 3.2. For the net in Fig. 2 we have the following translation of links (with the property names abbreviated):

- *a.Student*:

$$St(a) \leftarrow \textbf{not } \widehat{St}(a), \textbf{not } \widehat{in_1}(a)$$

$$in_1(a) \leftarrow \textbf{not } \widehat{St}(a), \textbf{not } \widehat{in_1}(a)$$

- *Student.YoungAdult*:

$$YA(x) \leftarrow St(x), \textbf{not } \widehat{YA}(x), \textbf{not } \widehat{in_2}(x)$$

$$in_2(x) \leftarrow St(x), \textbf{not } \widehat{YA}(x), \textbf{not } \widehat{in_2}(x)$$

- *YoungAdult.Adult*:

$$Ad(x) \leftarrow YA(x), \textbf{not } \widehat{Ad}(x), \textbf{not } \widehat{in_3}(x)$$

$$in_3(x) \leftarrow YA(x), \textbf{not } \widehat{Ad}(x), \textbf{not } \widehat{in_3}(x)$$

- *Adult.Employed*:

$$Emp(x) \leftarrow Ad(x), \textbf{not } \widehat{Emp}(x), \textbf{not } \widehat{in_4}(x)$$

$$in_4(x) \leftarrow Ad(x), \textbf{not } \widehat{Emp}(x), \textbf{not } \widehat{in_4}(x)$$

- *Student*.¬*Employed*:

$$\widehat{Emp}(x) \leftarrow St(x), \textbf{not } Emp(x), \textbf{not } \widehat{in_5}(x)$$
$$in_5(x) \leftarrow St(x), \textbf{not } Emp(x), \textbf{not } \widehat{in_5}(x)$$

The second part of the translation directly implements the mechanism of preemption in forming a credulous extension. We use a predicate $C_{p,q}(x)$ to code the positive connectivity from node p to node q for object x (with respect to a path set). These predicates are defined as follows:

$$C_{p,q}(x) \leftarrow in_{p.q}(x)$$
$$C_{p,q}(x) \leftarrow C_{p,u}(x), in_{u.q}(x)$$

where $p.q$ and $u.q$ are positive links.

Definition 3.3 (*Translation for preemption*). Let Γ be an arbitrary defeasible net. By abuse of notation, let $u.s$ denote a link, either positive or negative, from node u to node s. Then, the translation of Γ includes a clause

$$\widehat{in_{u.s}}(x) \leftarrow C_{r,u}(x)$$

if Γ has another link from r to s that conflicts with $u.s$.

The reader may want to compare this definition with that of off-path preemption depicted in Fig. 1. The translation here corresponds to the graph at the right hand side of that figure.

The reason that we do not need to implement the case depicted at the left hand side of Fig. 1 is as follows. Recall that preempting a path as given in Definition 2.2 can be achieved by invalidating the last link on it for the concerned object. Yet invalidating this link has the effect of preempting all the paths whose construction relies on it. Thus explicit preemption of any path $r\sigma u.s$ that is not via v (see the left graph in Fig. 1) in order to capture off-path preemption is no longer needed, since removing the link $u.s$ for the concerned object removes all these paths automatically.

From now on, we will denote by $\Pi(\Gamma)$ the translated logic program from the network Γ, which includes the clauses generated by Definitions 3.1 and 3.3, and those that define the connectivity for any two nodes p and q. The language of such a program consists of object nodes as domain elements, property nodes as predicates, and additional utility predicates such as the in_l predicates and the connectivity predicates.

Example 3.4. Continuing with the translation given in Example 3.2, the second part of the translation will yield one clause for implementing preemption

$$\widehat{in_{Ad.Emp}}(x) \leftarrow C_{St,Ad}(x)$$

along with the clauses for connectivity. The reader can verify that the only stable model of the translated logic program consists of the atoms $St(a)$, $YA(a)$, $Ad(a)$, $\widehat{Emp}(a)$, along with $in_1(a)$, $in_2(a)$, $in_3(a)$, $\widehat{in_4}(a)$, $in_5(a)$, and the true connectivity predicates $C_{a,St}(a)$, $C_{a,YA}(a)$, $C_{a,Ad}(a)$, $C_{St,YA}(a)$, $C_{St,Ad}(a)$, and so on.

We now discuss how the proposed translation may be used to accommodate subtype relations. Recall that a constructible path, as defined earlier in this paper, is one that must originate from an object node. As a result, an extension contains no information about subtype relations. Here we describe a generalization.

First observe that a path in the translation is indexed by an object x through the use of the $in_l(x)$ predicates. Then, to accommodate arbitrary paths, what we need is a mechanism to index those paths that start from a nonobject node. Technically, this can be achieved by using an appropriate naming mechanism to identify a class node also as an "object" node in disguise. That is, each class node p plays two roles: one as a normal class node, and the other as a disguised object node which serves as the head of the paths that are constructed from it. [1] This second role may be realized by giving the node p a distinct name, say o_p. Then, for any class node q, the conclusion that o_p is a q will be interpreted as p's are q's. A path that does not start from a real object node is then indexed by such a disguised object node.

For example, for the network at the left hand side of Fig. 1, suppose r is an object node. Besides the paths from r, the following paths are also constructible (in some appropriate context Φ): $o_v.u$, $o_v.\neg s$, $o_v.u.s$, and $o_u.s$. Hence the translated logic program will also contain the translation of the "object" links from o_v and o_u. Among these paths, only $o_v.u.s$ is preempted. Hence the unique credulous extension now contains the other three paths. We therefore conclude that v's are u's, u's are s's, but v's are not s's.

Clearly, this extension only involves adding the translation of the additional "object" links, and using the intended interpretation for the paths that are constructed from the corresponding "object" nodes.

3.1. Complexity of the translation

Let n be the number of nodes and e the number of links in a given net Γ. Clearly, the complexity of the translation of individual links is linear to e. The encoding of connectivity is bounded by $O(n^2)$, and in Definition 3.3 each link is examined against the connectivity of any pair of nodes. Thus, the whole translation is bounded by $O(e \times n^2)$.

Theorem 3.5. *Given a defeasible net Γ, let n be the number of nodes in Γ and e be the number of links in Γ. Then, the time complexity of obtaining $\Pi(\Gamma)$ is bounded by* $O(e \times n^2)$.

3.2. Correctness of the translation

We now show that the translation presented earlier in this section is semantics preserving. We first need a notation to relate models that consist of atoms and extensions that consist of paths. We define $\mathcal{P}(M)$ as

$$\{l_1 \ldots l_n\colon\ l_1 \ldots l_n \text{ is a path of } \Gamma \text{ from an object node } a \text{ and } \forall l_i, in_{l_i}(a) \in M\}.$$

[1] This is different from adding a new object node to each class node. That method works for skeptical inheritance but not for credulous inheritance, due to the problem of *decoupling* (cf. [10, p. 153]).

Theorem 3.6. *For any defeasible net Γ, Φ is a credulous extension of Γ iff $\Pi(\Gamma)$ has a stable model M such that $\mathcal{P}(M) = \Phi$.*

Proof. In the following we often use a generic notation for a path $a \ldots y$ which could be either positive or negative. If $w.y$ is a negative link then $w.\neg y$ denotes the corresponding positive link. Similarly, $p(x)$ could denote an atom ϕ or a negated atom $\widehat{\phi}$, and if $p(x)$ is $\widehat{\phi}$ then $\widehat{p}(x)$ denotes ϕ. These notations make it convenient to present a generic argument for the symmetric cases of dealing with either a positive path or a negative path.

($\Rightarrow$) Let Φ be a credulous extension of Γ. Let $M = I \cup \widehat{I} \cup P \cup C$ where

$$I = \{in_l(a)\text{: there is a path } a \ldots y \text{ via link } l \text{ in } \Phi\},$$

$$\widehat{I} = \{\widehat{in_{w.y}}(a)\text{: there is a path } a \ldots wy \text{ that is constructible but preempted in } \Phi\},$$

$$P = \{p(a)\text{: there is a positive path } a \ldots p \text{ in } \Phi\} \cup \{\widehat{p}(a)\text{: there is a negative path } a \ldots \neg p \text{ in } \Phi\},$$

$$C = \{C_{u,v}(a)\text{: there is a positive path } u \ldots v \text{ such that } in_l(a) \in I \text{ for each link } l \text{ in it}\}.$$

We show that M is a stable model of $\Pi(\Gamma)$, that is $\mathcal{F}_{\Pi(\Gamma)}(M_{\mathbf{not}}) = M_{\mathbf{not}}$.

Let $\sigma = p_1 \ldots p_n$ be a path in Φ. (By the notations above, the last link $p_{n-1}.p_n$ on σ could be either positive or negative; so the argument below applies to both cases.) By definition, σ is constructible from an object node, say a, i.e., $p_1 = a$. From the definition of M, and by the fact that σ is not preempted in Φ, we have $\widehat{in_{p_i.p_{i+1}}}(a) \notin M$, for any $1 \leqslant i \leqslant n-1$ (note that in off-path preemption, if a path $a \ldots p_i\, p_{i+1}$ is preempted, any path that originates from a with $p_i\, p_{i+1}$ as its last link will also be preempted). By the definition of P above, we also have $p_i(a) \in M$ for $2 \leqslant i \leqslant n$. In addition, from the fact that Φ is conflict-free we know $\widehat{p_i}(a) \notin M$ for $2 \leqslant i \leqslant n$. The translation of any nonobject link (for object a) consists of

$$p_{i+1}(a) \leftarrow p_i(a), \mathbf{not}\ \widehat{p_{i+1}}(a), \mathbf{not}\ \widehat{in_{p_i.p_{i+1}}}(a)$$
$$in_{p_i.p_{i+1}}(a) \leftarrow p_i(a), \mathbf{not}\ \widehat{p_{i+1}}(a), \mathbf{not}\ \widehat{in_{p_i.p_{i+1}}}(a)$$

Clearly, $\Pi(\Gamma) \cup M_{\mathbf{not}} \vdash p_{i+1}(a), in_{p_i.p_{i+1}}(a)$. We thus conclude that for any $\phi \in P$ and any $\varphi \in I$, $\Pi(\Gamma) \cup M_{\mathbf{not}} \vdash \phi, \varphi$. On the other hand, for any property atom $p_n(a)$, $\Pi(\Gamma) \cup M_{\mathbf{not}} \vdash p_n(a)$ only if there exist $\widehat{in_{p_{n-1}.p_n}}(a)$, $\widehat{p_n}(a) \notin M$ and $p_{n-1}(a) \in M$. Continuing this argument, it is clear that there must be a path $a \ldots p_n \in \Phi$, hence $p_n(a) \in P$. That is, for any property atom $\phi \notin P$, $\Pi(\Gamma) \cup M_{\mathbf{not}} \nvdash \phi$. A similar conclusion can be reached for all in_l atoms not in I. We therefore conclude that P and I of M remain the same by derivations from $\Pi(\Gamma) \cup M_{\mathbf{not}}$. Since both $\widehat{I}$ and C only depend on I and the given net Γ, they remain the same too. We thus have $\mathcal{F}_{\Pi(\Gamma)}(M_{\mathbf{not}}) = M_{\mathbf{not}}$.

($\Leftarrow$) Assume M is a stable model of $\Pi(\Gamma)$. We show that $\mathcal{P}(M)$ is a credulous extension of Γ. We need to show that every path in $\mathcal{P}(M)$ is defeasibly inheritable in $\mathcal{P}(M)$. That is, for any path $\tau \in \mathcal{P}(M)$,

(i) τ is constructible in $\mathcal{P}(M)$,
(ii) τ is conflict-free in $\mathcal{P}(M)$, and
(iii) τ is not preempted in $\mathcal{P}(M)$.

We will complete the proof by showing the fixpoint equation $\mathcal{P}(M) = \{\sigma\colon\ \mathcal{P}(M) \vdash_d \sigma\}$.

(i) holds trivially. To show (ii) assume that τ is not conflict-free. Then there exist a property p and an object a such that $p(a)$ and $\widehat{p}(a)$ are both in M. By virtue of the clauses in $\Pi(\Gamma)$, it is clear that neither of them can be derived from $\Pi(\Gamma) \cup M_{\mathbf{not}}$. This implies $\mathcal{F}_{\Pi(\Gamma)}(M_{\mathbf{not}}) \neq M_{\mathbf{not}}$, contradicting the assumption that M is a stable model. To prove (iii) let $\tau = r \ldots s$ be a path in $\mathcal{P}(M)$. Clearly, if τ is only a link it cannot be preempted. Assume τ is a path $r \ldots u.s$ consisting of at least two links. Note that by constructibility of τ, r must be an objective node. That $\tau \in \mathcal{P}(M)$ implies that $in_l(r) \in M$ for every link l on τ. Since M is a stable model of the translated logic program $\Pi(\Gamma)$, for $in_l(r)$ to be derivable from $\Pi(\Gamma) \cup M_{\mathbf{not}}$, $\widehat{in_l}(r)$ must not be in M. Now assume that the path τ is preempted in $\mathcal{P}(M)$. Then, there exist a path $\theta = r \ldots v \ldots u \in \mathcal{P}(M)$, where either $r = v$ (in which case θ is $v \ldots u$) or $r \neq v$, and a link $v.\neg s$ in Γ (now referring to Fig. 1). In this case, we would have either $\Pi(\Gamma) \cup M_{\mathbf{not}} \vdash \widehat{in_{u.s}}(r)$, or for some link l on either τ or θ, $\Pi(\Gamma) \cup M_{\mathbf{not}} \nvdash in_l(r)$. In either case, M cannot be a stable model of $\Pi(\Gamma)$ since $\mathcal{F}_{\Pi(\Gamma)}(M_{\mathbf{not}}) \neq M_{\mathbf{not}}$. Therefore, τ is not preempted in $\mathcal{P}(M)$.

The above proof in fact has shown $\mathcal{P}(M) \subseteq \{\sigma\colon\ \mathcal{P}(M) \vdash_d \sigma\}$. To show the other direction, suppose that a path $\sigma = p_1 \ldots p_{i+1}$ is defeasibly inheritable in $\mathcal{P}(M)$. If σ is an object link, apparently it is in $\mathcal{P}(M)$. Assume $i \geqslant 2$. By definition, we know $p_1 \ldots p_i \in \mathcal{P}(M)$. Since σ is defeasibly inheritable in $\mathcal{P}(M)$, we have $\Pi(\Gamma) \cup M_{\mathbf{not}} \vdash in_{p_i.p_{i+1}}(a)$ (assuming $p_1 = a$), due to the following clause in $\Pi(\Gamma)$ for the link $p_i.p_{i+1}$

$$in_{p_i.p_{i+1}}(a) \leftarrow p_i(a), \mathbf{not}\ \widehat{p_{i+1}}(a), \mathbf{not}\ \widehat{in_{p_i.p_{i+1}}}(a).$$

Hence we know $in_{p_i.p_{i+1}}(a) \in M$ (since M is a stable model of $\Pi(\Gamma)$). It then follows that $p_1 \ldots p_{i+1} \in \mathcal{P}(M)$. We therefore conclude $\{\sigma\colon\ \mathcal{P}(M) \vdash_d \sigma\} \subseteq \mathcal{P}(M)$. This completes the proof. □

3.3. A query answering procedure for acyclic nets

From the studies of logic programming semantics, it is known that the abductive procedure proposed by Eshghi and Kowalski [4] is a proof procedure for the regular model semantics. It is also known that if a general logic program has no *odd-dependency loops*, its stable models and regular models coincide. In this subsection we show that the general logic programs translated from acyclic nets satisfy this condition, and therefore the Eshghi–Kowalski procedure can be used directly to answer queries for networks under Horty's credulous semantics.

Definition 3.7. Let P be a general logic program. Define a directed graph G_P, called the graph of P, over the set of atoms as follows: for each clause

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, \mathbf{not}\ \gamma_1, \ldots, \mathbf{not}\ \gamma_n$$

in P, place a *positive* arc from β_i to α for each $1 \leqslant i \leqslant m$, and place a *negative* arc from γ_j to α for each $1 \leqslant j \leqslant n$.

A path on G_P is said to be *cyclic* if it begins and ends with the same atom. When such a path consists of only positive arcs, it is called *a positive* (*dependency*) *loop*. When it contains at least one negative arc, it is called *a negative* (*dependency*) *loop*.

Theorem 3.8 (You and Yuan [26]). *Let P be a general logic program and G_P be its graph. The stable models and the regular models of P coincide if there is no negative loop with an* odd *number of negative arcs.*

Theorem 3.9. *Let Γ be an acyclic net. The graph of $\Pi(\Gamma)$ has no positive loops, neither does it have any negative loop with an odd number of negative arcs.*

Proof. From the syntax of $\Pi(\Gamma)$, it is clear that there is no positive loop in the graph of $\Pi(\Gamma)$ for any atom, and there is no negative loop for any in_l atom. If a property atom has a negative loop then the number of negative arcs is a multiple of two.

The special syntactic form of the translated logic programs from acyclic nets allows the Eshghi–Kowalski procedure to be used without the need of positive loop checking. This procedure is given in Appendix A along with an example to illustrate how it may be used to answer queries for networks.

4. Network semantics by transformation

The logic programming interpretation of defeasible networks as presented in the preceding section provides a way to understand, indirectly, the possible semantics of defeasible nets. Namely, any semantics for general logic programs yields a semantics for defeasible nets. Since the translation is defined for all defeasible nets, such a semantics is defined for the class of all defeasible nets, including cyclic nets, a subject recently investigated in [1,24]. By our compilation method, at least two semantics defined this way are interesting, the well-founded semantics and the regular semantics. These two semantics are not only defined for all networks, they also possess sound and complete backward chaining proof procedures. In this section we compare this approach with Horty et al.'s "directly skeptical" approach. We also show how the translational approach may help resolve the problem of nonexistence of extensions in the direct approach to defeasible inheritance.

4.1. The well-founded semantics of defeasible networks

We begin with Horty et al.'s definition of skeptical extension [11].

Definition 4.1 (*Horty et al.'s skeptical grounded extension* [2]). Let Γ be an acyclic net. Φ is a *skeptical grounded extension* of Γ iff $\Phi = \bigcup_{i=0}^{i=\infty} \phi_i$ where ϕ_i is defined as follows:

[2] The definition is modified slightly for notational consistency.

(a) $\phi_0 = \{l\colon\ l$ is an object link of $\Gamma\}$;
(b) ϕ_{i+1} contains ϕ_i and each path σ of degree [3] $i+1$ with the following properties:
  (1) σ is inheritable in ϕ_i;
  (2) there is no path τ such that τ is inheritable in ϕ_i but τ conflicts with σ.

Note that by this construction conclusions are drawn essentially link by link. This appears to be a major drawback of the direct approach to skeptical reasoning, as there appears to be difficulties articulating the cascaded effects generated by sequences of links, possibly compounded by preemption. This is also the reason why in this approach only acyclic nets could be accommodated. (Note that the type of construction given here relies on the assumption that the given net is acyclic.)

We say that a conclusion is a *floating conclusion* if under any reasonable account of (credulous) extensions, every extension contains some path supporting this conclusion, but there is no skeptically acceptable path supporting this conclusion. Stein [19] as well as Makinson and Schlechta [13] argue that floating conclusions should be included in a skeptical semantics of defeasible nets.

The *zombie path problem* refers to the problem where a conclusion supported by the skeptical extension fails in some (reasonable) credulous extension and therefore is not in the intersection of conclusions supported by all credulous extensions [13,19].

If the problem of floating conclusion is a completeness problem, then the zombie path problem is a soundness problem. The latter is generally considered more serious as it could give us the conclusions that are not sanctioned by the intended reasoning mode.

We use the following example to compare the well-founded semantics with Horty et al.'s skeptical extension.

Example 4.2. Consider the net in Fig. 3. There are three credulous extensions:

$\{af, am, afg, afn, afny, afn\neg x, afnyp\}$,

$\{af, am, afg, afn, afny, afgx, afnyp, afgxp\}$,

$\{af, am, afg, am\neg n, afgx, afgxp\}$.

Since there is no possibility of preemption, these three credulous extensions represent three different ways to avoid contradictory conclusions.

The inference that $\underline{a \text{ is a } p}$ is a floating conclusion because every credulous extension contains some positive path from a to p, but there is no skeptical path from a to p.

By Horty et al.'s construction, the conclusion $\underline{a \text{ is a } p}$ is a skeptical one, as shown below:

$$\Phi = \bigcup_{i=0}^{i=\infty} \phi_i$$

[3] A path $x_1 \ldots x_n$'s degree is the length of the longest path of the net from node x_1 to node x_n. This term is only meaningful when the net is acyclic.

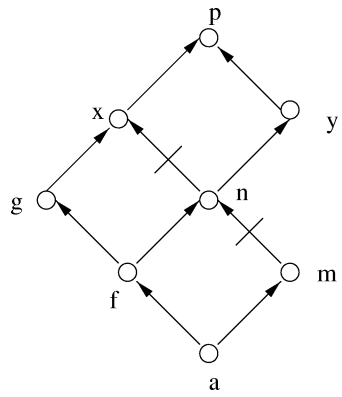

Fig. 3. The floating conclusion and zombie path problems.

where

$$\phi_0 = \{af, am\},$$

$$\phi_1 = \{af, am, afg\},$$

$$\phi_2 = \{af, am, afg, afgx\},$$

$$\phi_3 = \{af, am, afg, afgx, afgxp\},$$

$$\phi_{i+1} = \phi_i \quad \text{for every } i \geqslant 3.$$

Though the floating conclusion is not missed, the zombie path problem arises. For example, for the path *afgx* concluded in ϕ_2 above, there is a counter path $afn\neg x$; so *afgx* should not be a skeptical conclusion.

It can be verified that the well-founded model of the translated program corresponds to the path set $\{af, am, afg\}$. The reason for the exclusion of the path *afgx* is that *the counter path, $afn\neg x$, is computed easily in the construction of the well-founded model*. Clearly, articulating these types of computations and their effects is difficult in terms of links and paths, and generally considered foreign to the path-based approach.

That the zombie path problem does not arise in the well-founded semantics is guaranteed by the fact that the well-founded model of a general logic program is the intersection of all 3-valued stable models [17]. Since regular models and stable models are 3-valued stable models, it is not possible for a conclusion to be true in the well-founded model but not true in any regular or stable model.

4.2. The problem of nonexistence of extensions

We illustrate how our understanding of logic programming semantics may help us deal with the so called no extension problem that arises in the direct approach.

From the discussion in the preceding subsection, we see that the direct approach to skeptical inheritance is not defined for networks that involve cycles. For credulous

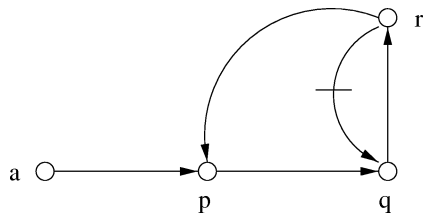

Fig. 4. To preempt or not to preempt: the cause of no credulous extensions.

inheritance, Horty shows that any acyclic network possesses at least one credulous extension, but a network involving cycles is not guaranteed to have any extension [10]. Consider, for example, the network in Fig. 4. The network has no credulous extensions. This can be seen as follows. First we notice that

(1) the node q is the only one that has both positive and negative links towards it, and
(2) the only node that could be a preempting node is the node r.

Then the question is whether the path $a.p.q$ is preempted or not. Now let's suppose the network had a credulous extension, say Φ. It would then contain the path $a.p.q$, or otherwise. In the first case it would also contain the path $a.p.q.r.p$; so the path $a.p.q$ would be preempted. On the other hand, if Φ did not contain $a.p.q$, it could not possibly contain $a.p.q.r.p$ either; so the path $a.p.q$ would not be preempted.

We now show that the translation of this network accurately captures this contradiction. In the following discussion, we will omit any default negation **not** ϕ in the body of a clause if ϕ does not appear in the head of any clause (in this case ϕ is false in the well-founded model, any stable model, and any regular model of the program).

Given the above network, according to Definition 3.3, we have only one clause with a negated in_l predicate in the head:

$$\widehat{in_{p.q}}(a) \leftarrow C_{r,p}(a)$$

which, according to the definition of the connectivity predicates, reduces to

$$\widehat{in_{p.q}}(a) \leftarrow in_{r.p}(a)$$

We now proceed to expand this clause. The following (reduced) clauses in the translated logic program are relevant.

$$in_{r.p}(a) \leftarrow r(a)$$
$$r(a) \leftarrow q(a)$$
$$q(a) \leftarrow p(a), \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{p.q}}(a)$$
$$p(a) \leftarrow$$
$$\widehat{q}(a) \leftarrow r(a), \textbf{not } q(a)$$

Reducing the clause that defines $\widehat{in_{p.q}}(a)$ above, and those that involve $q(a)$ and $\widehat{q}(a)$, gives us

$$\widehat{in_{p.q}}(a) \leftarrow \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{p.q}}(a)$$
$$\widehat{q}(a) \leftarrow \textbf{not } q(a), \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{p.q}}(a)$$
$$q(a) \leftarrow \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_{p.q}}(a)$$

The first clause says that a path that originates from a and contains the link $p.q$ is preempted only if it is not preempted. This captures precisely the reason why there exists no credulous extension. The involvement of $q(a)$ and $\widehat{q}(a)$ is because our encoding of the network also guarantees that all extensions (if there is any) are consistent. With this in mind it would be interesting to see that the second clause above describes the impossible situation where the derivation that $\underline{a \text{ is not a } q}$ depends upon

- not concluding $q(a)$, for the reason of consistency (of course), and
- a *negative* path from a to q (except for **not** $\widehat{in_{p.q}}(a)$, all the other default negations are omitted in the clause), in which one must not conclude $\widehat{q}(a)$ (since one gets $q(a)$ by following the path).

It is easy to see why the translated logic program has no stable models. $\widehat{in_{p.q}}(a)$ cannot be true in any stable model since its justification depends on the condition that $\widehat{in_{p.q}}(a)$ is false in the same model. On the other hand, $\widehat{in_{p.q}}(a)$ cannot be false either since it requires $\widehat{in_{p.q}}(a)$ to be true (for the first clause to be satisfied and for the reason that $\widehat{q}(a)$ cannot be true in any stable model).

Since credulous extensions are precisely stable models, we know that the notion of Horty's credulous extension is based on a two-valued interpretation of preemption: a path (satisfying the conditions of constructibility and nonconflict) is either *preempted* or *not preempted*. This notion collapses when neither can happen. The logic programming semantics that are based on three-valued logic shed light on how such situations might be interpreted. An extension now can be three-valued: a (constructible and nonconflicting) path is *preempted*, or *not preempted*, or *neither can be confirmed*. The last corresponds to the notion of *undefined* in three-valued logic.

Under the translational approach, various semantics for logic programs give us possible ways to understand the meaning of the networks. In particular, the well-founded and regular semantics localize the effect of a contradiction and allow noncontradictory conclusions to be drawn. For example, for the network in Fig. 4, it is clear that the fact $\underline{a \text{ is a } p}$ is undisputed. We get into trouble only when we insist on deciding whether $\underline{a \text{ is a } q}$, and the others that depend on it. As expected, its translated logic program has the well-founded model (which is also its unique regular model in this case) that concludes $in_{a.p}(a)$ and $p(a)$, and treats $in_{p.q}(a)$, $\widehat{in_{p.q}}(a)$, and the others as undefined.

5. Final remarks

We have presented a framework of reasoning with defeasible inheritance: a defeasible inheritance network is compiled to a general logic program and a query on the network is answered by a logic programming proof procedure with respect to the compiled logic

program. The compilation method yields additional insights into how the semantics of defeasible networks might be characterized. This is demonstrated by a comparison between the well-founded semantics of translated logic programs and the skeptical extension of Horty et al., and by a discussion of the problem of nonexistence of extensions.

A three-valued notion of preemption was also investigated by Antonelli in a path-based formalism [1]. The key technical idea is a fixpoint definition that prevents the contradictory situation where the construction of a path leads to its preemption. A credulous semantics is defined which guarantees at least one extension for any defeasible net. The similarities between this idea and those of three-valued semantics for general logic programs point to the possibility of a close relation between the two. The precise relationship deserves further investigation. However, it is far from clear how skeptical inheritance might be treated in a path-based formalism. Any iterative construction based on a link-by-link extension will inevitably give rise to the zombie path problem, because shorter paths tend to get concluded before contradiction arises. The main difficulty, as pointed out in this paper, is the lack of a simple way to describe the interactions of the reasoning generated by different (and often interleaving) link sequences, possibly compounded by preemption of paths. In contrast, these types of reasoning in the translational approach are captured naturally by reasoning with the translated logic program.

Recently there is a renewed interest in defeasible inheritance in object-oriented languages and systems. Inheritance plays a central role in code reuse in these systems. Researchers in the object-oriented community have been investigating possible mechanisms to restructure inheritance hierarchies. This is because programming nowadays is no longer only about procedures and input and output relations. Rather, it has become a modeling process, where incomplete and changing information must be dealt with constantly. The framework of reasoning with defeasible inheritance as presented in this paper is relevant to the problem of how inheritance hierarchies might be maintained dynamically. Some initial discussions can be found in [25].

Acknowledgement

The authors thank the two referees for their careful reading of the manuscript and constructive comments.

Appendix A

The Eshghi–Kowalski (EK) procedure can be used, without positive loop checking, to answer queries for acyclic networks via their translated logic programs.

A.1. Key notations in the EK procedure

There are two types of derivations in the EK procedure: *abductive derivations* for proving negation by failure, and *consistency derivations* for consistency checks.

We associate a goal G_i with a set of default negations H_i, and write (G_i, H_i) to keep track the default negations that have been proved or are currently being proved.

Assume subgoal selection is from left to right. So a goal is written as $\leftarrow l, N$ where l is the selected literal and N is the set of the rest subgoals.

A.2. The EK procedure for acyclic defeasible nets

An *abductive derivation* from (G_1, H_1) to (G_n, H_n) is a sequence $(G_1, H_1), \ldots, (G_n, H_n)$ where each G_i has the form $\leftarrow l, N$ and (G_{i+1}, H_{i+1}) is obtained from (G_i, H_i) as

(AD1) if l is an atom then

$$G_{i+1} = C \quad \text{and} \quad H_{i+1} = H_i,$$

where C is the resolvent of G_i with some program clause on the selected literal l;

(AD2) if l is a default negation **not** ϕ, and **not** $\phi \in H_i$, then

$$G_{i+1} = \leftarrow N \quad \text{and} \quad H_{i+1} = H_i;$$

(AD3) if l is a default negation **not** ϕ, and **not** $\phi \notin H_i$, and there is a consistency derivation from $(\{\leftarrow \phi\}, H_i \cup \{\textbf{not } \phi\})$ to $(\emptyset, H')$, then

$$G_{i+1} = \leftarrow N \quad \text{and} \quad H_{i+1} = H'.$$

A *consistency derivation* from (F_1, H_1) to (F_n, H_n) is a sequence where F_i has the form $\{\leftarrow l, N\} \cup F_i'$ and (F_{i+1}, H_{i+1}) is obtained from (F_i, H_i) as:

(CO1) if l is an atom then

$$F_{i+1} = G \cup F_i' \quad \text{and} \quad H_{i+1} = H_i,$$

where G is the set of all resolvent goals from $\leftarrow l, N$ on the selected literal l and $\square \notin G$ ($\square$ denotes the empty goal);

(CO2) if l is a default negation **not** $\phi \notin H_i$, then if there is an abductive derivation from $(\leftarrow \phi, H_i)$ to $(\square, H')$, then

$$F_{i+1} = F_i' \quad \text{and} \quad H_{i+1} = H'$$

else if N is nonempty then

$$F_{i+1} = \{\leftarrow N\} \cup F_i' \quad \text{and} \quad H_{i+1} = H_i;$$

(CO3) if l is a default negation **not** $\phi \in H_i$ and N is nonempty, then

$$F_{i+1} = \{\leftarrow N\} \cup F_i' \quad \text{and} \quad H_{i+1} = H_i.$$

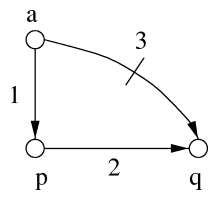

Fig. A.1. A simple net for an illustration of the EK procedure.

```
-------------------------------------------------------------------
|  <- -q(a)                                                    {}
|  <- not q(a), not -in_3(a)
|     (AD3) ---------------------------------------------------
|     ! { <- q(a)}                                    {not q(a)}
|     ! { <- p(a), not -q(a), not -in_2(a)}
|     ! { <- not -p(a), not -in_1(a), not -q(a), not -in_2(a)}
|     ! { <- not -in_1(a), not -q(a), not in_2(a)}
|     !                                % by CO2's else-part
|     ! { <- not -q(a), not -in_2(a)}   % by CO2's else-part
|     !    (CO2)-------------------------------------------------
|     !    | <- -q(a)                                  {not q(a)}
|     !    | <- not q(a), not -in_3(a)
|     !    | <- not -in_3(a)             % by AD2
|     !    |    (AD3)-----------------------------------------
|     !    |    !  { <- -in_3(a)}           {not -in_3(a),not q(a)}
|     !    |    !  \O
|     !    |    !-----------------------------------------------
|     !    | []
|     !    |-----------------------------------------------------
|     ! \O
|     !-------------------------------------------------------------
|  <- not -in_3(a)                          {not -in_3(a),not q(a)}
| []
|-------------------------------------------------------------------
```

Fig. A.2. A proof of $\widehat{q}(a)$.

Given an acyclic net Γ, an object a has the property p iff there is an abductive derivation from $(\leftarrow p(a), \emptyset)$ to $(\square, H_n)$ (called a *refutation*) for the program $\Pi(\Gamma)$.

We use a simple example to illustrate how this procedure works. Consider the net Γ in Fig. A.1. Note that it is sufficient to consider the ground program instantiated from $\Pi(\Gamma)$ over object a. Thus $\Pi(\Gamma)$ consists of the following clauses:

$$p(a) \leftarrow \textbf{not } \widehat{p}(a), \textbf{not } \widehat{in_1}(a)$$

$$in_1(a) \leftarrow \textbf{not } \widehat{p}(a), \textbf{not } \widehat{in_1}(a)$$

$$q(a) \leftarrow p(a), \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_2}(a)$$

$$in_2(a) \leftarrow p(a), \textbf{not } \widehat{q}(a), \textbf{not } \widehat{in_2}(a)$$

$$\widehat{q}(a) \leftarrow \textbf{not } q(a), \textbf{not } \widehat{in_3}(a)$$

$$in_3(a) \leftarrow \textbf{not } q(a), \textbf{not } \widehat{in_3}(a)$$

```
-------------------------------------------------------------------
|  <- q(a)                                                    {}
|  <- p(a), not -q(a), not -in_2(a)
|  <- not -p(a), not -in_1(a), not -q(a), not -in_2(a)
|     ..... % steps for proving the first two subgoals successfully
|  <-  not -q(a), not -in_2(a)            {not -p(a), not -in_1(a)}
|     (AD3) ----------------------------------------------------
|     ! { <- -q(a)}            {not -q(a), not -p(a), not -in_1(a)}
|     ! { <- not q(a), not -in_3(a)}
|     !    (CO2)---------------------------------------------------
|     !    | <- q(a)           {not -q(a), not -p(a), not -in_1(a)}
|     !    | <- p(a), not -q(a), not -in_2(a)
|     !    | <- not -p(a), not -in_1(a), not -q(a), not -in_2(a)
|     !    | <- not -in_2(a)      % by applying AD2 three times
|     !    |    (AD3)-----------------------------------------------
|     !    |    ! { <- -in_2(a)}          {not -in_2(a), not -q(a),
|     !    |    !                          not -p(a), not -in_1(a)}
|     !    |    ! { <- C_{a,p}}
|     !    |    ! { <- in_1(a)}
|     !    |    ! { <- not -p(a), not -in_1(a)}
|     !    |    ! cannot reduce to \0; backtrack but eventually fail
```

Fig. A.3. A failed attempt to prove $q(a)$.

$$\widehat{in_2}(a) \leftarrow C_{a,p}(a)$$

$$C_{a,p}(a) \leftarrow in_1(a)$$

We show a proof of the goal $\leftarrow \widehat{q}(a)$ in Fig. A.2 and a failed attempt to prove $\leftarrow q(a)$ in Fig. A.3. Semantically, one can see that the former must rely on the fact that the path $a.\neg q$ is not preempted and $q(a)$ is not derivable. Thus one would expect that the derivation of $\widehat{q}(a)$ should be supported by the default negations **not** $\widehat{in_3}(a)$ and **not** $q(a)$. The failure to prove the second query is due to the fact that the path $a.p.q$ is preempted.

In drawing these proofs, we use `\0` for empty set $\emptyset$, `[]` for empty clause $\square$, and `-u` for negated atom $\widehat{u}$.

References

[1] G. Antonelli, Defeasible inheritance on cyclic networks, Artificial Intelligence 92 (1997) 1–23.

[2] P. Dung, An argumentation theoretic foundation for logic programming, J. Logic Programming 22 (1995) 151–177.

[3] P. Dung, T. Son, Nonmonotonic inheritance, argumentation and logic programming, in: Proc. LPNMR-95, 1995, pp. 317–329.

[4] K. Eshghi, R.A. Kowalski, Abduction compared with negation by failure, in: Proc. 6th ICLP, MIT Press, Cambridge, MA, 1989, pp. 234–254.

[5] D. Etherington, R. Reiter, On inheritance hierarchies with exceptions, in: Proc. AAAI-83, Washington, DC, Morgan Kaufmann, San Mateo, CA, 1983, pp. 104–108.
[6] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. 5th ICLP, MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[7] M. Gelfond, H. Przymusinska, Formalization of inheritance reasoning in autoepistemic logic, Fund. Inform. XIII (1990) 403–443.
[8] E. Gregorie, Skeptical theories of inheritance and nonmonotonic logics, Methodologies for Intelligence System 4 (1989) 430–438.
[9] B. Haugh, Tractable theories of multiple defeasible inheritance in ordinary nonmonotonic logics, in: Proc. AAAI-88, St. Paul, MN, AAAI Press, 1988, pp. 421–426.
[10] J.F. Horty, Some direct theories of nonmonotonic inheritance, in: M. Gabbay, C. Hogger, J. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, Vol 3: Nonmonotonic Reasoning and Uncertain Reasoning, Oxford University, Oxford, 1994, pp. 111–187.
[11] J. Horty, R. Thomason, D. Touretzky, A skeptical theory of inheritance in nonmonotonic semantic networks, Artificial Intelligence 42 (1990) 311–348.
[12] F. Lin, A study of nonmonotonic reasoning, PhD Thesis, Stanford University, 1991.
[13] D. Makinson, K. Schlechta, Floating conclusions and zombie paths: Two deep difficulties in the "directly skeptical" approach to defeasible inheritance nets, Artificial Intelligence 48 (1991) 109–209.
[14] L. Morgenstern, Inheriting well-formed formulae in formula-augmented semantic network, in: Proc. Internat. Conference on the Principles of Knowledge Representation and Reasoning (KR-96), Cambridge, MA, 1996.
[15] L. Morgenstern, Inheritance comes of age, in: Proc. IJCAI-97, Nagoya, Japan, 1997.
[16] L. Morgenstern, M. Singh, An expert system using nonmonotonic techniques for benefits inquiry in the insurance industry, in: Proc. IJCAI-97, Nagoya, Japan, 1997.
[17] T.C. Przymusinski, Well-founded semantics coincides with three-valued stable semantics, Fund. Inform. 13 (1990) 445–463.
[18] R. Reiter, G. Criscuolo, On interacting defaults, in: Readings in Nonmonotonic Reasoning, Morgan Kaufmann, San Mateo, CA, 1987, pp. 94–100.
[19] L.A. Stein, Skeptical inheritance: Computing the intersection of credulous extensions, in: Proc. IJCAI-89, Detroit, MI, 1989.
[20] L.A. Stein, Resolving ambiguity in nonmonotonic inheritance hierarchies, Artificial Intelligence 55 (1992) 259–310.
[21] D. Touretzky, The Mathematics of Inheritance Systems, Morgan Kaufmann, San Mateo, CA, 1986.
[22] A. Van Gelder, The alternating fixpoint of logic programs with negation, J. Comput. System Sci. 47 (1) (1993) 185–221.
[23] A. Van Gelder, K. Ross, J. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650.
[24] X. Wang, J. You, L. Yuan, A default interpretation of defeasible networks, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 156–162.
[25] J. You, Inheritance with exceptions and logic programming, in: Tutorial Proc. Joint Internat. Conference and Symposium on Logic Programming, 1998.
[26] J. You, L. Yuan, A three-valued semantics for deductive databases and logic programs, J. Comput. System Sci. 49 (1994) 334–361.
[27] J. You, L. Yuan, On the equivalence of semantics for normal logic programs, J. Logic Programming 22 (1995) 212–221.