Nonmonotonic Inheritance in Object-Oriented Deductive Database Languages

Wolfgang May Paul-Th. Kandzia
{may|kandzia}@informatik.uni-freiburg.de

TECHNICAL REPORT No. 114

Institut für Informatik
Albert-Ludwigs-Universität
Am Flughafen 17
79110 Freiburg, Germany

January 1999

Keywords: Nonmonotonic reasoning, Inheritance, Default Logic, Deductive object-oriented databases, F-Logic.

Abstract

Deductive object-oriented frameworks integrate logic rules and inheritance. There, specific problems arise: Due to the combination of deduction and inheritance, (a) deduction can take place depending on inherited facts, thus raising indirect conflicts, and (b) also the class hierarchy and -membership is subject to deduction. From this point of view, we investigate the application of the extension semantics of Default Logic to deductive object-oriented database languages. By restricting the problem to Horn programs and a special type of defaults tailored to the semantics of inheritance, a forward-chaining construction of a Herbrand-style representation of extensions is possible. This construction is compared with a solution as implemented in the F-Logic system FLORID which is based on a combination of classical deductive fixpoints and an inheritance-trigger mechanism.

From the F-Logic point of view, the main contribution of the report is to investigate the relationship between *inheritance-canonic* models as defined in [KLW95] and classical AI frameworks: we show that the semantics which is defined and implemented for F-Logic coincides with the standard semantics of Default Logic and Inheritance Networks. In this report, we restrict ourselves to scalar methods.

A preliminary version of this technical report has been published at *13. Workshop logische Programmierung - WLP'98*, Vienna, October 1998 [MK98].

Contents

1 Introduction

In deductive object-oriented database languages, a class hierarchy and non-monotonic inheritance is used for modeling an application domain. Facts can be derived either by classical deduction, or by *inheritance*: Assume that an object o is an instance of a class c, and it is known that a "typical" instance of c has a property p. Then, if it can consistently be assumed that p holds for o, it is added to the model. Exact notions are defined in course of the paper.

The combination of deductive rules with inheritance is significantly more complex than pure deduction or pure inheritance concepts (e.g., Description Logics), where efficient implementations exist. In this work, we study the restricted case where defeasible reasoning is only concerned with inheritance. This combination is particularly of interest in deductive object-oriented databases.

The AI Viewpoint. In the AI community, several frameworks for non-monotonic reasoning have been presented which implement a notion of *defaults* (for an overview, see [GHR94, Bre91]).

Nonmonotonic reasoning is integrated into logic programming with negation; such programs are evaluated wrt. well-founded semantics [VGRS88] or stable semantics [GL88, BF91]; for an overview, see [Dix95]. Moreover, *extended logic programs* allow negation in rule heads and provide two types of negation, i.e., negation by failure and classical, strong negation, evaluated by extensions of stable and well-founded semantics [Prz91, GL90]. *Circumscription* [Lif94] uses the same syntax as first-order logic (and classical logic programming), augmented with a special predicate abnormal. The intended models are those which minimize the abnormal predicate. In *Default Logic* [Rei80, Poo94, MT93], defeasible reasoning is expressed by *defaults*: $a{:}b/c$ denotes that, given a, if b can be assumed consistently, we can conclude c (*precondition:justification/consequence*). Default Logic is presented in more detail in Section 4.1. *Inheritance Networks* [Tou86, Hor94] provide a comprehensive framework for specifying *typical* or *atypical* properties. An inheritance network is given as a graph, consisting of *defeasible* links and *strict* links, the former represent defeasible knowledge whereas the latter represent conditionals. An approach to inheritance in frame systems based on Circumscription is presented in [Bre87]. As a *semantic* approach, *preferential models* [Sho88, KLM90, Mak94] provide a very general formalization of nonmonotonic reasoning. Except inheritance networks, the above approaches are based on first-order syntax. There, deductive rules can be incorporated into the consequence relation (e.g., defaults without justifications, rules without the $\neg$ abnormal-literal). A derived class-membership is supported (since classes are represented by predicates). In extended logic programs, Default Logic, and Circumscription, nonmonotonic reasoning is not restricted to inheritance, but includes general conclusions.

For the above frameworks, due to possible conflicts, *credulous* and *skeptical* inheritance semantics can be defined. According to [Hor94], for purely defeasible networks, the complexity depends on the exact definition of the semantics, ranging from polynomial to NP-complete; for mixed networks, complexity issues are not yet solved. Default Logic, in general, is not even semi-decidable; although in the *Theorist* system [PGA87], default reasoning has been implemented for empirical studies. Hence, the above approaches in their full extent are not very useful for practical systems. In [Mor98], *formula-augmented semantic networks* (there, formulas can be inherited) have been successfully used in a commercial application.

The DOOD Situation. On the other hand, in the deductive database community, nonmonotonic features (except strict negation) are still very rare. The paradigm of deductive object-oriented database languages conceptually includes nonmonotonic inheritance, but this is not actually integrated into existing languages and implementations. Here, *structural* inheritance denotes a refining, but not fully overriding inheritance on the *signature* level: if a is a b, and the signature of class b provides a method m which results in type t, then a also provides m, resulting in a subtype of t. In contrast, *value inheritance* denotes the concept of nonmonotonic inheritance known from AI.

The early object-oriented logics focussed on complex objects, but still lacked a class-hierarchy or inheritance. A class hierarchy with only structural inheritance has been introduced in LOGRES [CCCR$^+$90], IQL [AK92], and ROL [Liu96].

Nonmonotonic value inheritance can be found in Gulog [DT95]. There, the class hierarchy and class membership are static, thus, inheritance conflicts can be detected a priori. Additionally, consistency wrt. scalar methods is enforced by the condition that for every ground method definition, there is at most one rule instance which possibly defines it. Programs satisfying these conditions are called "well-defined", resulting in a very restricted language.

F-Logic [KLW95] supports nonmonotonic value inheritance with overriding together with a class hierarchy which can be defined by rules. It has been successfully applied for AI techniques in [KS97]. In the F-Logic system FLORID[1] [FHK$^+$97], a T_P-like operator evaluating the classical logic part of a program, and a trigger mechanism handling nonmonotonic inheritance are implemented. This semantics is investigated in Section 8.

The paper is structured as follows: Section 2 introduces the concept of inheritance from the AI point of view and relates it with the requirements on inheritance in the deductive object-oriented database area. Section 3 presents the syntax and semantics of F-Logic used throughout the paper and illustrates the problem arising from the combination of inheritance and

[1] available from `http://www.informatik.uni-freiburg.de/~dbis/florid.html`.

deduction. In Section 4, Default Logic and its semantics is introduced, and a characterization of inheritance by defaults is given. In Section 5, the global semantics of default theories via *extensions* as given in [Rei80, Poo94, MT93] is investigated. In Section 6, we adapt the results to default theories consisting of a Horn program and the special "Horn-like" defaults which characterize inheritance, resulting in a Herbrand-style representation of extensions. In Section 7, the consequences of the previous sections for the special type of defaults needed for inheritance are investigated. In Section 8, we present the semi-declarative semantics which is defined and implemented for F-Logic. This semantics is based on logical deduction, *inheritance triggers* and *inheritance-canonic models*. Section 9 shows the relation between the presented concepts wrt. the problem of inheritance and shows the equivalence and correctness of the F-Logic solution. Section 10 contains some remarks on set-oriented vs. element-oriented strategies, classifying properties of extensions, and implementation and complexity issues. In Section 11, we give some classic examples illustrating our approach and showing how different modeling concepts can be used to obtain the intended behavior. In another example, the concept is applied to implement dynamic behaviour, where the frame problem is solved via inheritance, also leading to an elegant solution of the ramification problem.

2 Inheritance

The idea of inheritance in an object-oriented setting can informally be described as follows: Assume that an object o is an instance of a class c, and it is known that a "typical" instance of c has a property p. Then, if it can consistently be assumed that p holds for o, it is added to the model.

2.1 Inheritance Networks

For a formal *direct* characterization of inheritance and for reasoning about inheritance strategies, Inheritance Networks [Tou86, Hor94] provide an intuitive and expressive graph-based framework. Here, *direct* means that reasoning in Inheritance Networks is done in terms of the network itself as a "semantical", *path-based* approach. This stands in contrast to *translational theories* where the consequences of a network are interpreted in some more standard *syntactical* nonmonotonic formalism (e.g., Default Logic, Circumscription, or Autoepistemic Logic, or the approach discussed in the present paper). Nevertheless, the *concepts* are best motivated in this semantical formalism and have then to be implemented in a target formalism.

Inheritance networks are more general than inheritance in the object-oriented model: *Nodes* correspond to *individuals* (nodes with only outgoing links) or to *properties* an individual can have, and links describe the connections between nodes. Here, *defeasible links* play the main role: a defeasible link from an property p to a property q means that a typical object which satisfies p also satisfies q, e.g., "*being a bird* $\to$ *flies*". A negative link from p to q denotes that a typical object which satisfies p does not satisfy q; e.g., "*being a bird* $\not\to$ *swims*". Nevertheless, there are atypical birds, e.g., penguins which do not fly but swim. Links from individuals to properties are e.g. of the form "*tweety* $\to$ *being a bird*".

Strict links are an extension of basic inheritance networks, leading to *mixed inheritance networks* (cf. [Hor94, Ch. 3]). Strict links denote that *every/no* object which satisfies p satisfies q, e.g., "*being a penguin* $\Rightarrow$ *being a bird*", or "*being a penguin* $\not\Rightarrow$ *being a mammal*".

Mapping an object-oriented model to an inheritance network results in three types of nodes: objects, classes, and properties. The subclass-relation is encoded into strict links between classes, the is-a-relation is encoded into strict links from objects to classes, and inheritable properties are encoded into defeasible links from classes to properties. Additionally, facts and rules concerning only a single class or object can be encoded into strict links between different types of nodes. Thus, the combination of deductive rules and inheritance results in mixed networks.

The following classics "Nixon Diamond" and "Tweety Triangle" illustrate the central concepts in inheritance. The corresponding inheritance networks are given in Figure 1.

Example 1 (Nixon Diamond) We know that Nixon is a republican and a quaker. A typical republican's policy is being a hawk, the typical policy of a quaker is being a pacifist. Now, there is a direct conflict with Nixons policy. □

Example 2 (Tweety Triangle) Typical birds fly. Tweety is a bird. Penguins are birds. Tweety is a penguin. Typical penguins do not fly. Here, since "*Tweety is a penguin*" is more specific than "*Tweety is a bird*", one should conclude that Tweety does not fly, i.e., it is a typical penguin, but not a typical bird. □

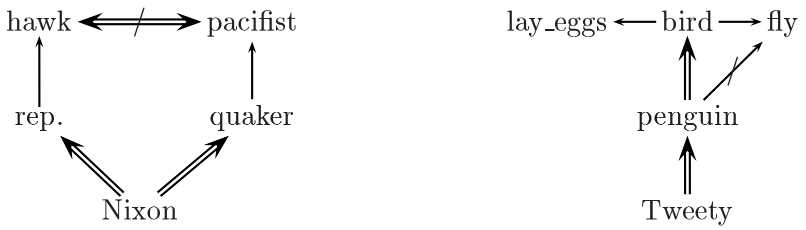


Figure 1: Inheritance Networks of Nixon and Tweety

Reasoning in inheritance networks is based on *paths* in the network. A path is a sequence of links where only the last link can be a negative link; every path can be seen as an argument, e.g., in Figure 1, the path *Nixon*

– *republican* – *hawk* is an argument that Nixon is believed to be a political hawk. On the other hand, the path *Nixon* – *quaker* – *pacifist* is an argument that he is a pacifist, and not a hawk. The paths *Nixon* – *republican* – *hawk* and *Nixon* – *quaker* – *pacifist* are said to be *in conflict* (here, we follow the notion of a *mixed conflict*, cf. [Hor94, Ch. 3]). Both are equally reasonable, leading to two different solutions.

In the Tweety example, the path *Tweety* – *penguin* – *bird* – *flies* is an argument that Tweety is believed to be able to fly, whereas the path *Tweety* – *penguin* $\not-$ *flies* is an argument that it is not. Again, these paths conflict each other, nevertheless, here, the conclusion that Tweety does not fly should be preferred since it is grounded on the more specific fact that Tweety is a penguin. The path *Tweety* – *penguin* – *bird* – *flies* is said to be *preempted* by the path *Tweety* – *penguin* $\not-$ *flies*, thus it cannot be used for inference. On the other hand, the path *Tweety* – *penguin* – *bird* – *lay_eggs* is not preempted.

The basic concepts of *conflict* and *preemption* are conceptionally clear and well-defined, even in the case of mixed networks.

The semantics of inheritance networks is defined in terms of *extensions*: intuitively, an extension is a set Φ of paths (arguments) that an ideal reasoner might accept – i.e., Φ must not contain conflicting or preempted paths (for formal definitions see [Tou86, Hor94]).

Nevertheless, there are several more involved notions, dealing with the intuitive "adequacy" of extensions:

Decoupling. Consider the inheritance networks given in Fig. 2 (from [Tou86]); note that the networks show some similarities with the Nixon Diamond. Obviously, there is a conflict between the paths $n - r \not- p$ and $n - q - p$, both arguments are equally reasonable.

The left network allows an extension Φ containing the paths $n - r \not- p$, $a - n - q - p$, and $b - n - r \not- p$: a typical n satisfies r and does not satisfy p, whereas a is a typical n and satisfies p. The path $a - n - q - p$ is called *decoupled* in Φ, the conclusions about a are not properly coupled with those about typical n's. b is another typical n which satisfies r and does not satisfy p, although the net gives identical information about a and b. Here, prohibiting decoupling leads to the intended result that all typical n's behave like a typical n should behave.

On the other hand, in the net on the right in Fig. 2, an extension containing the paths $n - r \not- p$ and $a - n - q - p$ is reasonable: although the conflict between $n - r \not- p$ and $n - q - p$ is solved in favour of r, a which is a typical n, but does not satisfy r, is believed to satisfy q and p. Here, the knowledge about typical n's is in fact disjunctive information which does not actually need a decision how a *typical* n should behave, but which would better be solved for each individual n. In Section 11.1, we show how disjunctive information is handled in our approach.

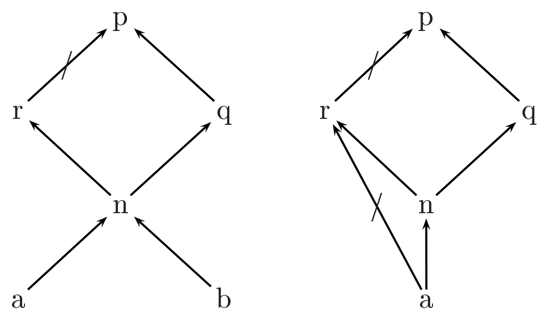


Figure 2: Decoupling

On-path preemption. In [Hor94], *on-path* preemption is investigated as an additional mode of preemption: it occurs if a link in the net is a shortcut for a path, thereby overlooking more specific information which would preempt the link (example: a direct link *Tweety* – *bird* in Figure 1 which would advocate concluding that tweety flies).

When giving a formal characterization, it becomes obvious that several details of this idea can be understood differently in different application areas. Using *formula-augmented inheritance networks* (there, formulas can be inherited) in a commercial application, Morgenstern [Mor98] advocates different strategies from the application's point of view. In fact, it is unlikely that a uniform inheritance strategy can be given which provides the intended semantics in all cases.

2.2 Inheritance in the Database Context

By *path-based* reasoning in inheritance nets, a large part of the conclusions is of the form "a typical x satisfies p", or – even more general "typically, an individual which satisfies p also satisfies q". Only some conclusions are concerned with individuals, saying "it is reasonable to believe that a satisfies p".

When applying inheritance concepts in the object-oriented deductive database area, the focus is not on general nonmonotonic reasoning, but on its consequences *in a given database instance* – i.e, on properties of individual objects. The more general conclusions of the form "a typical x satisfies p", or "typically, an individual which satisfies p also satisfies q" are then subject to the application of data mining algorithms on the database instance.

By focussing on objects in a given database instance and their properties instead of reasoning about abstract typical individuals, some problems coming with the path-oriented approach of inheritance networks can be circumvented: conflicts naturally occur when reasoning about properties of individuals, not when reasoning about typical properties of classes.

Obviously, the problem of decoupling is strongly related with the path-based approach. It can nearly be ignored in the database setting – moreover, it even allows for specifying disjunctive information (cf. Section 11.1): For a class, properties can be defined as inheritable to member objects although they are conflicting if both of them *would actually be* inherited – the conflict (and the decision which alternative should be preferred) can be limited to the *instance level* since there is no statement which properties a typical x should actually have. In Section 11.1, it is shown how the situation given in Fig. 2 is solved in this way. Similarly, on-path preemption is naturally solved by the approach which systematically propagates knowledge downwards the class hierarchy.

For object-oriented databases, the concepts of a class hierarchy and inheritable properties induce a special structure of the "network":

- there are three disjoint types of nodes: objects, classes, and properties,
- the class hierarchy is represented by *strict links* from (sub)classes and objects to (super)classes,
- facts are represented by *strict links* from classes and objects to properties, and by *strict symmetric negative links* between conflicting properties,
- inheritable properties are represented by *defeasible links* from classes to properties (inheritable to subclasses and objects),

⇒ these are the only defeasible links, thus there are no chains of defeasible links.

In this model, there is a concise distinction between classes and objects since a class does *not* represent simultaneously its typical member (although a class can be regarded as an object of itself – which is a completely different notion); thus, conflicts can be handled separately on class level or on object level.

Remark 1 Consider an inheritance network defining a class hierarchy with inheritable properties of classes and properties of objects. Let Φ be a decoupling-free extension of the network, containing a path o–c_1–$\ldots$–c_n–c–p (i.e., o is a c via several intermediate classes $c_1, \ldots, c_n$, and c provides the inheritable property p, and o is believed to inherit this property from c). Then, Φ contains also the path c_i–$\ldots$–c_n–c–p for every i (which results in believing that c_i provides the inheritable property p). □

On the other hand, deductive rules have in general no representation in inheritance networks[2]. Thus, inheritance nets are not suitable for formalizing inheritance in deductive databases. Instead, some syntax-based nonmonotonic formalism has to be employed to meet the requirements of logic programming. For this work, we decided to use Default Logic. The following effects of deductive rules on inheritance have to be considered:

- conflicts due to deductive closure: before inheriting, the deductive closure after inheritance has to be checked for conflicts,

[2]Rules concerning only a single object or class can be encoded by introducing conjunctive properties corresponding to rule bodies.

- derived class hierarchy and class membership: deriving membership in intermediate classes can lead to preemption.

2.3 The Inheritance Strategy

In inheritance networks, the conclusions are based on reasoning about paths which cannot be encoded into translational approaches – such as Default Logic. Thus, a formalization in Default Logic requires a different strategy which can be expressed by formulas or rules. In this work, we adopt the inheritance mechanism from object-oriented programming languages: inheritance to a class or object takes place from a *direct* (OO programming: the minimal) superclass (which in course can inherit from its direct superclass(es)). With this, the problem of *decoupling* is solved in a natural way: a subclass or an object can only inherit properties which are known to be inheritable in a direct superclass.

Note that, in contrast to class hierarchies in object-oriented programming languages, we do not require unique minimal superclasses.

Remark 2 As long as coupling is required, this *localized* strategy is equivalent with the path-based concept of inheritance nets:

- every path is equivalent to a sequence of inheritance steps downwards through the class hierarchy and deductive steps.
- Let a–c_1– … –c_i–c_{i+1}– … –c_n–p be a path where a is an individual, c_i are classes, and p is a property, such that the path is preempted by another path a–c_1– … –$c_i \not- p$. Then, p is inherited to all classes from c_n to c_{i+1}, but inheritance stops with the step from c_{i+1} to c_i.
- conflicting paths result in a conflict in the uppermost node they have in common. By changing the modeling, the conflict can be decided either on the class level in some class they have in common, or on the object level at the receiving object (cf. Section 11.1). □

3 F-Logic: Language and Basic Concepts

This report has been motivated by the problem of integrating non-monotonic *value inheritance* into the deductive object-oriented database language F-Logic (cf. [KLW95]) and the Florid prototype ([FLO98]). The characterization given in Section 8 is implemented in Florid.

Definition 1 (Syntax of F-Logic) The syntax of F-Logic (without multivalued methods and schema reasoning) is defined as follows:

- The alphabet consists of a set $\mathcal{F}$ of *object constructors*, playing the role of function symbols, a set $\mathcal{V}$ of variables, and several auxiliary symbols. Object constructors are denoted by lowercase letters and variables by uppercase ones.
- *id-terms* are composed from object constructors and variables. They are interpreted by elements of the universe.

In the sequel, let O, $O_1, \ldots, O_n$, C, D, M, and V denote id-terms.

- An *is-a atom* is an expression of the form O isa C (object O is a member of class C), or $C :: D$ (class C is a subclass of class D).[3]
- The following are *object atoms*:
  - $O[M \rightarrow V]$: applying the *scalar* method M to O – as an object – results in V,
  - $O[M \bullet\!\!\rightarrow V]$: O – as a class – provides the *inheritable scalar* method M. For a member o isa O, inheritance results in $o[M \rightarrow V]$; for a subclass $c :: O$, inheritance results in $c[M \bullet\!\!\rightarrow V]$.
  - Analogously $O[M@(O_1, \ldots, O_n) \rightarrow V]$ and $O[M@(O_1, \ldots, O_n) \bullet\!\!\rightarrow V]$ with $n \in \mathbb{N}$ for parameterized methods.
- *Formulas* are built from atoms using first-order logic connectives.
- An F-Logic *rule* is a logic rule $h \leftarrow b$ over atoms.
- An F-Logic *program* is a set of rules. □

Note that F-Logic does not distinguish between classes, methods, and objects which uniformly are denoted by id-terms; also variables can occur at arbitrary positions of an atom.

The semantics of F-Logic extends the semantics of first-order predicate logic. Formulas are interpreted over a semantic structure. We restrict our discussion to Herbrand-interpretations where the universe consists of ground id-terms. An *H-structure* is a set of ground F-Logic atoms describing an object world, thus it has to satisfy several *closure axioms* related to general object-oriented properties:

Definition 2 (Closure Axioms) A (possibly infinite) set $\mathcal{H}$ of ground atoms is an *H-structure* if the following conditions hold for arbitrary ground id-terms u, $u_0, \ldots, u_n$, and u_m occurring in $\mathcal{H}$:

- $u :: u \in \mathcal{H}$ (subclass reflexivity),
- if $u_1 :: u_2 \in \mathcal{H}$ and $u_2 :: u_3 \in \mathcal{H}$ then $u_1 :: u_3 \in \mathcal{H}$ (subclass transitivity),
- if $u_1 :: u_2 \in \mathcal{H}$ and $u_2 :: u_1 \in \mathcal{H}$ then $u_1 = u_2 \in \mathcal{H}$ (subclass acyclicity),
- if u_1 isa $u_2 \in \mathcal{H}$ and $u_2 :: u_3 \in \mathcal{H}$ then u_1 isa $u_3 \in \mathcal{H}$ (instance-subclass dependency),

$*$ there are no ground id-terms u and u' such that $u_0[u_m \rightsquigarrow u] \in \mathcal{H}$ and $u_0[u_m \rightsquigarrow u'] \in \mathcal{H}$, where $\rightsquigarrow$ stands for $\rightarrow$ or $\bullet\!\!\rightarrow$ (uniqueness of scalar methods).

For a set M of ground atoms, $\mathcal{C}\ell(M)$ denotes the closure of M wrt. the above axioms, $\mathcal{C}\ell(M) = \perp$ if the constraint $(*)$ is violated in M.

By $\mathrm{Th}_{\mathrm{FL}}(F)$, we denote the F-Logic theory of a set F of formulas which means the closure of F wrt. a complete set of axioms of first-order logic and

[3] we use isa instead of the original ":" since ":" is already used for defaults.

the axioms

$$\frac{}{X :: X} \qquad \frac{X_1 :: X_2\ ,\ X_2 :: X_3}{X_1 :: X_3} \qquad \frac{X_1 :: X_2\ ,\ X_2 :: X_1}{X_1 = X_2}$$

$$\frac{X_1 \text{ isa } X_2\ ,\ X_2 :: X_3}{X_1 \text{ isa } X_3} \qquad \frac{O[M{\leadsto}V]\ ,\ O[M{\leadsto}V']\ ,\ V \neq V'}{\text{false}}$$

(again, $\leadsto$ stands for $\rightarrow$ or $\bullet\!\!\rightarrow$.) □

For an H-structure, the truth of atoms and formulas is given in the usual way [KLW95]. Positive F-Logic programs are evaluated bottom-up by a T_P-like operator including $\mathcal{C}\ell$, providing a minimal model semantics:

Definition 3 (Deductive Fixpoint)
For an F-Logic program P and an H-structure $\mathcal{H}$,

$$\begin{aligned}
T_P(\mathcal{H}) &:= \mathcal{H} \cup \{h \mid (h \leftarrow b_1, \ldots, b_n) \text{ is a ground instance of some rule of } P \\
&\qquad\qquad \text{and } b_i \in \mathcal{H} \text{ for all } i = 1, \ldots, n\}\ , \\
T_P^0(\mathcal{H}) &:= \mathcal{C}\ell(\mathcal{H})\ , \\
T_P^{i+1}(\mathcal{H}) &:= \mathcal{C}\ell(T_P(T_P^i(\mathcal{H})))\ , \\
T_P^{\omega}(\mathcal{H}) &:= \begin{cases} \lim_{i\to\infty} T_P^i(\mathcal{H}) & \text{if the sequence } T_P^0(\mathcal{H}), T_P^1(\mathcal{H}), \ldots \text{ converges,} \\ \bot & \text{otherwise.} \end{cases}
\end{aligned}$$

Note that $\mathcal{C}\ell(\mathcal{H}) = \bot$ can also lead to the result $\bot$. □

The above T_P-operator does not deal with inheritance. In [KLW95], *inheritance-canonic* models are defined, based on *inheritance triggers* which extend the above fixpoint semantics with some procedural flavor. This definition is further investigated in Section 8.
First, we give some introductory examples which show that logical deduction in combination with inheritance can lead to semantical difficulties and even contradictions already in simple settings:

Example 3 (Nixon Diamond) Consider the program

$P = \{$quaker[policy•→pacifist], republican[policy•→hawk],
r_nixon isa quaker, r_nixon isa republican$\}$.

which is the F-Logic representation of the Nixon Diamond given in Example 1.
Here, nothing can be derived by classical deduction. Both policies can be argued to be inherited. Each of them can be inherited without any problem, making r_nixon[policy→_] defined, "blocking" the other. □

Example 4 (Nixon Family)
Consider again the Nixon-Diamond, augmented by

$\{$W[policy→P] $\leftarrow$ W[husband→O],O[policy→P], mrs_nixon[husband→r_nixon] ,
mrs_nixon isa quaker$\}$

Now, there are the following possibilities:

- r_nixon inherits r_nixon[policy→hawk] and from this, classical deduction derives mrs_nixon[policy→hawk]. In this case, mrs_nixon[policy→pacifist] must not be inherited – thus, she is an atypical quaker.
- r_nixon inherits r_nixon[policy→pacifist] – in which case classical deduction derives mrs_nixon[policy→pacifist] which is the same value as she would (have) inherit(ed) from being a quaker. □

The next example is a slight variation, showing this conflict even stronger:

Example 5 (Extended Nixon Family) Consider the following version of the Nixon family:

$$P = \{\mathsf{r\_nixon\ isa\ republican},\ \mathsf{republican[policy{\bullet\!\!\to}hawk]},\\ \mathsf{mrs\_nixon[policy{\to}pacifist]},\ \mathsf{mrs\_nixon[husband{\to}r\_nixon]},\\ \mathsf{W[policy{\to}P]} \leftarrow \mathsf{W[husband{\to}O]}{\wedge}\mathsf{O[policy{\to}P]}\}\ .$$

Here, although there is no *direct* conflict when inheriting r_nixon[policy→hawk], the logical consequences require mrs_nixon[policy→hawk], leading to an inconsistency. Thus, a "responsible" semantics must not inherit in this situation, though leaving the policy of r_nixon undefined. □

Among the related frameworks, inheritance nets [Hor94] take care about such indirect conflicts with their notion of *mixed conflicts*. Default Logic [Poo94, MT93] also incorporates this notion within the justifications – a default only applies if its justification is consistent with the resulting structure. Circumscription [Lif94] would result in the fact that Nixon is abnormal. In Gulog [DT95], this program does not satisfy the restrictions for a well-defined program.

Example 6 Consider the Tweety example in F-Logic:

$$P = \{\mathsf{bird[fly{\bullet\!\!\to}true;\ laying\_eggs{\bullet\!\!\to}true]},\ \mathsf{penguin[fly{\bullet\!\!\to}false]},\\ \mathsf{penguin :: bird},\ \mathsf{tweety\ isa\ penguin}\}\ .$$

With the above definition, $\mathcal{C}\ell(P) = P \cup \{\mathsf{tweety\ isa\ bird}\}$. Here, tweety should inherit tweety[fly→false] from penguin, not tweety[fly→true] from bird since the potential inheritance of tweety[fly→true] from bird is *preempted* by the intermediate class penguin[fly•→false]. On the other hand, the property [laying_eggs•→true] should be inherited from bird to penguin[laying_eggs•→true] and to tweety[laying_eggs→true] (cf. Example 11). □

This example motivates one of the strategies which are applied in the sequel: properties are inherited *stepwise* downwards the class hierarchy.

The result of this section is that application of inheritance has to deal with two kinds of facts:

1. explicit: checking the superclass condition, that inheritance is not preempted, and the requirement that the method to be inherited is not yet defined,

2. implicit: there can be facts which would be inconsistent with the inherited property, although they are not rejected by (1) (cf. Ex. 5).

Here, (2) cannot be provided by a non-defeasible encoding into logic rules. Instead, in the next sections it is shown that the framework of *defaults* covers the meaning of defeasible inheritance and how this can be integrated with the classical logic programming idea underlying F-Logic and similar deductive database languages.

4 Default Logic and Inheritance

4.1 Default Logic: The Framework

In *Default Logic* [Rei80, Poo94, MT93], defeasible reasoning is expressed by *defaults*: a default

$$d \;=\; \frac{\alpha : \beta_1, \ldots, \beta_n}{w}$$

consists of a *precondition* $p(d) = \alpha$, a *justification* $J(d) = \beta = \{\beta_1, \ldots, \beta_n\}$ and a *consequence* $c(d) = w$; for a set D of defaults, $J(D) := \bigcup_{d \in D} J(d)$, analogous $c(D)$. Given α, if β can be assumed consistently, one can conclude w. If β is true, the default $\alpha{:}\beta/w$ is equivalent to the logic rule $w \leftarrow \alpha$ as long as only consistent interpretations are considered. A *default theory* is a pair $\Delta = (D, F)$ where D is a set of defaults and F is a set of formulas.

By this, application of defaults deals in a general way with the above-mentioned two kinds of facts:

1. the precondition represents the explicitly required knowledge,
2. the justification lists facts which must be consistent with the knowledge, but not necessarily must belong to the knowledge.

4.2 Inheritance in Default Logic

In an inheritance framework, the superclass condition belongs to (1); whereas the checks that inheritance is not preempted and that the inherited value must be consistent with the knowledge (wrt. the logical rules of the program) fall under (2).

For characterizing inheritance, only a specialized form of defaults is needed, called *semi-normal defaults*. Semi-normal defaults are of the form $\alpha(\bar{x}){:}\beta(\bar{x})/w(\bar{x})$ where the precondition $\alpha(\bar{x})$ is a conjunction of atoms, the consequence $w(\bar{x})$ is also an atomic formula, and $\forall \bar{x} : \beta(\bar{x}) \rightarrow w(\bar{x})$ holds. Translating the path-based concept of inheritance networks, inheritance in F-Logic syntax can be specified by defaults of the form

$$D'_{inh} \;:=\; \frac{\begin{array}{c} O \;\mathsf{isa}\; C \;,\; C[M {\bullet\!\!\rightarrow} V] \;, \phi_{\text{path}}(O\text{–}C_1\text{–}\ldots\text{–}C_n\text{–}C) \;: \\ \phi_{\text{not\_preempted}}(O\text{–}C_1\text{–}\ldots\text{–}C_n\text{–}C, M {\bullet\!\!\rightarrow} V) \;,\; O[M {\rightarrow} V] \end{array}}{O[M {\rightarrow} V]}$$

(analogous for $C' :: C$)

where ϕ_{path} is a meta-predicate which states that $O\text{–}C_1\text{–}\ldots\text{–}C_n\text{–}C$ is a path in the class-hierarchy, and $\phi_{\text{not\_preempted}}(O\text{–}C_1\text{–}\ldots\text{–}C_n\text{–}C, M \bullet\!\!\to V)$ is a meta-predicate which states that inheritance of $M \bullet\!\!\to V$ along the path $O\text{–}C_1\text{–}\ldots\text{–}C_n\text{–}C$ is not preempted; i.e., that $c'[M \bullet\!\!\to V]$ is consistent for all intermediate classes c' on this path.

Due to the fact that variables are also allowed at class and method positions, every instance of inheritance of an inheritable non-parameterized scalar method (which are denoted by $\bullet\!\!\to$) is an instance of the above default schema.

Note, that for an H-structure $\mathcal{H}$, $o[m \to v]$ can only be assumed consistently if there is no $v' \neq v$ such that $o[m \to v] \in \mathcal{H}$.
By Remark 1, inheritance along a path can be split into a sequence of smaller steps (until in every step, inheritance takes place from an immediate superclass) which do not require path-based reasoning:

$$D_{inh} = \frac{\begin{array}{l} O \ \mathsf{isa}\ C\ ,\ C[M \bullet\!\!\to V]\ : \\ \quad \forall C'((O \ \mathsf{isa}\ C' \wedge C' :: C) \to C'[M \bullet\!\!\to V])\ ,\ O[M \to V] \end{array}}{O[M \to V]}\ .$$

(analogous for $C' :: C$.)

Definition 4 For a given F-Logic program P, by Δ_P we denote the Horn default theory (D_P, P) where D_P contains the above default schema D_{inh} for every arity of methods and for inheritance to subclasses. Let $n_{\max}$ be the maximal arity of a method occurring in P. Then, D_P contains for every $n \leq n_{\max}$ the following default schemata:

$$\frac{\begin{array}{l} O \ \mathsf{isa}\ C\ ,\ C[M@(O_1,\ldots,O_n) \bullet\!\!\to V]\ : \\ \quad \forall C'((O \ \mathsf{isa}\ C' \wedge C' :: C) \to C'[M@(O_1,\ldots,O_n) \bullet\!\!\to V])\ ,\ O[M@(O_1,\ldots,O_n) \to V] \end{array}}{O[M@(O_1,\ldots,O_n) \to V]}$$

$$\frac{\begin{array}{l} SC :: C\ ,\ C[M@(O_1,\ldots,O_n) \bullet\!\!\to V]\ : \\ \quad \forall C'((SC \ \mathsf{isa}\ C' \wedge C' :: C) \to C'[M@(O_1,\ldots,O_n) \bullet\!\!\to V])\ ,\ SC[M@(O_1,\ldots,O_n) \bullet\!\!\to V] \end{array}}{SC[M@(O_1,\ldots,O_n) \bullet\!\!\to V]}$$

□

5 Global Semantics of Default Theories

5.1 Extensions

The semantics of a default theory is defined in terms of *extensions*. In the following, for a set S of formulas, let $\mathrm{Th}(S)$ denote the theory of S.[4]

Definition 5 (Extension; based on [Poo94])
Let $\Delta = (D, F)$ be a default theory. For sets S, T of formulas, let

$$\begin{array}{rl} GD(S,T,D) := \{d \mid & d \text{ is an instance of a default in } D,\ \mathrm{Th}(T) \models p(d)\ \text{, and} \\ & \mathrm{Th}(S \cup \{\beta\}) \text{ is consistent for every } \beta \in J(d)\} \end{array}$$

[4] wrt. the respective framework, e.g., propositional, first-order, or F-Logic.

(*generating defaults*). Then, for all sequences $S_0 = F, S_1, S_2, \ldots$ of sets of formulas s.t. $S = (\bigcup_{i=0}^{\infty} S_i)$ and

$$S_{i+1} = S_i \cup C_i \quad \text{where } C_i = c(GD(S, S_i, D)) \ ,$$

$\mathrm{Th}(S)$ is an *extension* of Δ. Since S is needed later on, we call it an *extension base* of Δ. □

Definition 6 Let D be a set D of defaults and S a set of formulas. Then,

- $GD(S, D) := GD(S, S, D)$ is the set of applicable defaults in S,
- $GD^+(S, D) := \{d \in GD(S, D) \mid c(d) \notin \mathrm{Th}(S)\}$ is the set of applicable defaults which add knowledge not (yet) contained in S. □

Remark 3 Note that $S_i = F \cup \bigcup_{j=0,\ldots,i-1} C_j$ and $S = F \cup \bigcup_{i=0}^{\infty} C_i = F \cup c(GD(S, D))$ and $GD^+(S, D) = \emptyset$, i.e., for all defaults d which are applicable in S, the consequence of d is in S. □

In [Mak94], this is termed a *quasi-inductive* definition: in the step $i \to i{+}1$, all β_j are required to be consistent with $\mathrm{Th}(S) = \mathrm{Th}(\bigcup_{i=0}^{\infty} S_i)$, thus, assumptions about *future* stages are made (note that in contrast, the evaluation of α does not use S). Note that, depending on which assumptions are made, there can be several *different* extensions (cf. Ex. 3).
In [MT93], an equivalent definition is given in terms of a *belief set* S and *S-proofs*:

Definition 7 (cf. [MT93]) Let (D, F) be a default theory and S a theory.

- An *S-proof* of a formula φ wrt. (D, F) is a finite sequence $\varphi_1, \ldots, \varphi_n = \varphi$ such that for every $1 \leq i \leq n$, one of the following conditions hold:
  - $\varphi_i \in F$,
  - φ_i is provable with first-order logic from $\varphi_1, \ldots, \varphi_{i-1}$,
  - there is an instance $\alpha(\bar{c}){:}\beta_1(\bar{c}), \ldots, \beta_m(\bar{c})/w(\bar{c})$ of a default in D such that $\alpha(\bar{c}) = \varphi_j$ for some $j < i$ and $\beta_k(\bar{c})$ is consistent with S for all $1 \leq k \leq m$.
- $\mathrm{conseq}^{D,S}(F)$ denotes the set of all formulas having an S-proof wrt. (D, F).
- Then, a theory S is an extension for (D, F) if $S = \mathrm{conseq}^{D,S}(F)$. □

In both definitions, S must be guessed to prove that it is an extension, then it can be checked if S is the result of the fixpoint process (first definition) or satisfies the given equation (second definition), respectively.

5.2 Forward Chaining Evaluation

Motivated by the fixpoint semantics for positive logic programs, the evaluation of logic programs with inheritance should also be based on a forward-chaining approach, i.e. without having to guess S first. From Definition 5, a forward-chaining, *inflationary* strategy can be defined by replacing "Th($S \cup \{\beta\}$) is consistent" with "Th($S_i \cup \{\beta\}$) is consistent", i.e. evaluating defaults against the *current* belief set. In contrast to Def. 5, in every step, we allow the application of exactly one default (in Section 10.1.1 it will be proven that this makes no difference as long as only positive programs and defaults with positive preconditions are considered).

Definition 8 (Inflationary extension)
Let $\Delta = (D, F)$ be a default theory. Let $AD_0 = \emptyset$ and $S_0 = F, S_1, S_2, \ldots, S_\eta$ be a sequence of sets of formulas such that

$$S_{i+1} = S_i \cup \{c(d_i)\}\ ,\ AD_{i+1} = AD_i \cup \{d_i\} \text{ where } d_i \in GD^+(S_i, D)\ ,$$

and $GD^+(S_\eta, D) = \emptyset$ (for the definition of GD^+ see Def. 6). Then, with $S = (\bigcup_{i=0,\ldots,\eta} S_i)$, Th($S$) is called an *inflationary extension* of Δ; we call S an *inflationary extension base* of Δ. □

Remark 4 Note that again, $S_i = F \cup \bigcup_{j=0,\ldots,i-1}\{c(d_j)\}$ and $S = F \cup \bigcup_{i=0,\ldots,\eta}\{c(d_i)\}$ and $GD^+(S, D) = \emptyset$. □

This approach is, e.g., investigated in [MT93, Section 3.7, Def. 3.61]. As shown there, the above method is complete, but not sound: it generates theories which are no extensions.

Proposition 1 (Extensions vs. Inflationary Extensions)
Let $\Delta = (D, F)$ be a Default theory.

1. *Every extension S of Δ is also an inflationary extension of Δ, and*
2. *Let S be an inflationary extension computed by the algorithm given in Definition 8. If for every $\beta \in J(AD_\eta)$, β is consistent with S, then S is an extension of Δ.* □

Proof 1. cf. [MT93, Cor. 3.68 and 3.71, Th. 3.73].
2. cf. [MT93, Th. 3.65]. ■

The strategy is inflationary in the sense that a default which has been once applied is not undone (which would require to undo also all its logical consequences) if in a later step one of its *justifications* turns out to be wrong which is exactly the tested criterion in (2) of the above proposition.

Remark 5 There are two alternatives how to deal with this problem: (i) forbid the application of defaults whose justifications will be falsified later, or (ii) forbid the application of a default whose logical consequences would falsify the justifications of another default which has been applied earlier.

As we see, the notion of extensions includes (i) whereas (ii) is much easier to implement. (i) leads to theories where no further default is applicable whereas (ii) can lead to structures where some defaults are still applicable. On the other hand (i) does not guarantee that such a structure exists, whereas a structure satisfying (ii) always exists.

We will see that (ii) is weaker than (i), but the difference can be controlled in case of inheritance. □

Example 7
Consider a default theory $(\{d_1, d_2\}, F)$ such that $GD(F, \{d_1, d_2\}) = \{d_1, d_2\}$, $GD(F \cup c(d_1)) = d_2$, $GD(F \cup c(d_2)) = \emptyset$, and $c(d_2) \to \neg J(d_1)$.
Here, both $T_1 = \mathrm{Th}(F \cup c(d_2))$ and $T_2 = \mathrm{Th}(F \cup c(d_1) \cup c(d_2))$ are inflationary extensions. T_1 is the only extension. T_2 is not an extension since $T_2 \models \neg J(d_1)$, thus, the justification of d_1 is falsified by application of d_2.
$T_3 = \mathrm{Th}(F \cup c(d_1))$ is *not* an inflationary extension (and also not an extension) since $GD^+(T_3, D) = d_2$.
The strategy (ii) above would result in T_1 and T_3 as acceptable structures. □

Cautious inflationary extensions are defined similar to Definition 8, following strategy (ii), i.e., avoiding the falsification of previous justifications:

Definition 9 (Cautious inflationary extension)
Let $\Delta = (D, F)$ be a default theory. For a set S of formulas and a set AD of ground instances of defaults, let

$$\begin{aligned} GD^+_{caut}(S, D, AD) := \{d \mid\ & d \text{ is an instance of a default in } D,\ \mathrm{Th}(S) \models p(d)\ , \\ & \text{and } \mathrm{Th}(S \cup c(d) \cup \beta) \text{ is consistent} \\ & \text{for every } \beta \in J(AD \cup \{d\}) \text{ and } c(d) \notin \mathrm{Th}(S)\}\ . \end{aligned}$$

Let $AD_0 = \emptyset$ and $S_0 = F, S_1, S_2, \ldots, S_\eta$ be a sequence of sets of formulas such that

$$S_{i+1} = S_i \cup \{c(d_i)\}\ ,\ AD_{i+1} = AD_i \cup \{d_i\} \quad \text{where } d_i \in GD^+_{caut}(S_i, D, AD_i)\ ,$$

and $GD^+_{caut}(S_\eta, D, AD_\eta) = \emptyset$. Then, with $S = (\bigcup_{i=0,\ldots,\eta} S_i)$, $\mathrm{Th}(S)$ is called a *cautious inflationary extension* of Δ; we call S a *cautious inflationary extension base* of Δ. □

Remark 6 Note that again, $S_i = F \cup \bigcup_{j=0,\ldots,i-1}\{c(d_j)\}$ and $S = F \cup \bigcup_{i=0,\ldots,\eta}\{c(d_i)\}$, but now $GD^+(S, D) \neq \emptyset$ is possible, i.e, there can be applicable defaults d in S such that $c(d) \notin S$ (then, $c(d)$ would lead to falsification of a justification of a previously applied default, thus, $d \notin GD^+_{caut}(S, D, AD_\eta)$). □

Example 8 The above notions define strictly different notions of extensions. Consider the following default theory:

$$(D, \{p\}) \quad \text{where} \quad D = \left\{ \frac{p : \neg q}{r, s}\ ,\ \frac{p}{r, q} \right\}\ .$$

Here, $S = \{p, r, q\}$ is the only extension, generated by $GD(\{p, r, q\}, \{p\}, D) = \{p/r, q\}$. S is also an inflationary extension and a cautious inflationary extension.
But, $GD(\{p\}, D)$ does not only contain $p/r, q$ since $p : \neg q/r, s \in GD(\{p\}, D)$. Applying $p : \neg q/r, s$ in $\{p\}$ leads to $S' = \{p, r, s\}$ which is not an extension since $GD^+(\{p, r, s\}, D) = \{p/r, q\}$. Subsequent application of $\{p/r, q\}$ results in $\{p, r, s, q\}$ which is an inflationary extension, but the justification of the previously applied default $p : \neg q/r, s$ is invalidated. Thus, S' is a cautious inflationary extension – with $GD^+(S', D) \neq \emptyset$.

There is no extension where $\neg q$ is consistent, and the default $p : \neg q/r, s$ is not applied in the construction of any extension. Thus, when the inflationary strategy chooses to apply the default $p : \neg q/r, s \in GD^+(\{p\}, D)$ it runs into a *garden path* – it is not possible then to reach a valid extension. □

As in the above example, the cautious strategy can run into garden paths, i.e., applying defaults such that it is not possible to reach an extension. Garden paths can only be cured by backtracking.

Proposition 2 (Cautious Inflationary vs. Inflationary Extensions)
Let $\Delta = (D, F)$ be a default theory. Then,

- *The computations of cautious inflationary extensions are the maximal prefixes of computations of inflationary extensions such that no justification of a previously applied default is falsified.*
- *A cautious inflationary extension S of Δ is an inflationary extension if $GD^+(S, D) = \emptyset$.*
- *If an inflationary extension S satisfies the criterion given in Proposition 1(2), then S is also a cautious inflationary extension.* □

Note that an inflationary extension not necessarily contains a cautious inflationary extension:

Example 9 (Cautious Inflationary vs. Inflationary Extensions)
Consider a default theory (D, F) with $D = \{d_1, d_2, d_3\}$ such that $GD(F, D) = \{d_1\}$, $GD(F \cup \{c(d_1)\}) = \{d_2, d_3\}$, $GD(F \cup \{c(d_1), c(d_2)\}) = GD(F \cup \{c(d_1), c(d_3)\}) = \emptyset$, and $F \cup \{c(d_1), c(d_2)\}$ is consistent with $\beta(d_1)$, whereas $F \cup \{c(d_1), c(d_3)\}$ is inconsistent with $\beta(d_1)$.

Then, $\mathrm{Th}(F \cup \{c(d_1), c(d_2)\})$ is an extension (and also a cautious inflationary extension), and $\mathrm{Th}(F \cup \{c(d_1), c(d_3)\})$ is an inflationary extension which does not satisfy Proposition 1(2) and which does not contain a cautious inflationary extension. □

Proposition 3 (Extensions vs. Cautious Inflationary Extensions)
Given a default theory $\Delta = (D, F)$, a cautious inflationary extension S of Δ is an extension of Δ if $GD^+(S, D) = \emptyset$. □

Proof By Proposition 2, every cautious inflationary extension S such that $GD^+(S, D) = \emptyset$ is an inflationary extension. Since every cautious inflationary extension satisfies the additional criterion stated in Proposition 1, it is then an extension of Δ. ■

6 The Horn Case

Given a default theory (D, P) which consists of a set P of Horn formulas and a set D of semi-normal defaults[5], both in Definitions 5 and 8, every S_i and the resulting base $S = P \cup \bigcup_{i=0}^{\infty} C_i$ is Horn. Thus, the semantics can equivalently be given in a Herbrand style similar to minimal models in logic programming. In the following, we consider the case of F-Logic programs and restrict ourselves to finite extensions.

Definition 10 Given an F-Logic program P and an extension base S of Δ_P (cf. Def. 4), $\mathcal{H} := T_S^\omega(\emptyset)$ is called the *H-extension* of Δ_P to S (*inflationary H-extensions*, or *cautious inflationary H-extensions*).[6] □

The forward-chaining approach of Definition 8 can also be used for computing the inflationary H-extensions of a program without manipulating sets of formulas:

Proposition 4 *Let P be an F-Logic program with a semi-normal default theory $\Delta = (D, P)$. For an H-structure $\mathcal{H}$, let*

$$\begin{aligned} GD^+(\mathcal{H}, D) := \{d \mid\ & d \text{ is a ground instance of a default in } D,\ p(d) \subseteq \mathcal{H}, \\ & Th_{\mathrm{FL}}(P \cup \mathcal{H} \cup \{\beta\}) \text{ is consistent for every } \beta \in J(d), \\ & \text{and } c(d) \notin \mathcal{H}\}\ . \end{aligned}$$

Let $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_\eta$ be a sequence of H-structures such that $\mathcal{H}_0 = T_P^\omega(\emptyset)$ and

$$\mathcal{H}_{i+1} = T_P^\omega(\mathcal{H}_i \cup \{c(d_i)\})\ ,\ AD_{i+1} = AD_i \cup \{d_i\} \quad \text{where } d_i \in GD^+(\mathcal{H}_i, D)\ ,$$

and $GD^+(\mathcal{H}_\eta, D) = \emptyset$. If $\mathcal{H} := \bigcup_{i=0}^{\eta} \mathcal{H}_i \neq \bot$, then $\mathcal{H}$ is an inflationary H-extension of Δ. Moreover, every inflationary H-extension can be computed by such a sequence. □

Proof The proof uses the fact that $\mathcal{H}_i = T_P^\omega(\bigcup_{j \in 0,\ldots,i-1}\{c(d_j)\})$.
There is a bijection between computations as in Definition 8 and Proposition 4 such that for every i, $\mathcal{H}_i = T_{S_i}^\omega(\emptyset)$ and the d_i coincide:
$i = 0$: $\mathcal{H}_0 = T_P^\omega(\emptyset) = T_{S_0}^\omega(\emptyset)$.
$i \to i+1$: For every instance d of a default in Δ, $p(d) \subseteq \mathcal{H}_i$ if and only if $T_{S_i}^\omega(\emptyset) \models p(d)$. Since S_i is Horn, this is the case if and only if $\mathrm{Th}_{\mathrm{FL}}(S_i) \models p(d)$.

[5] recall that there, for every default d, $p(d)$ is a conjunction of atoms and $w(d)$ is an atomic formula.

[6] note that by Def. 3, T_P^ω includes the closure $\mathcal{C}\ell$.

By induction hypothesis, $\mathrm{Th}_{\mathrm{FL}}(P \cup \mathcal{H}_i \cup \{\beta\}) = \mathrm{Th}_{\mathrm{FL}}(P \cup T^{\omega}_{S_i}(\emptyset) \cup \{\beta\})$. Since $S_i = P \cup \bigcup_{j=0\ldots i-1}\{c(d_i)\}$, $\mathrm{Th}_{\mathrm{FL}}(P \cup T^{\omega}_{S_i}(\emptyset) \cup \{\beta\}) = \mathrm{Th}_{\mathrm{FL}}(S_i \cup \{\beta\})$, thus, the consistency requirement is identical in both cases. Hence, $GD^+(\mathcal{H}_i, D) = GD^+(S_i, D)$, thus, the same d_i can be chosen.

Then, $\mathcal{H}_{i+1} = T^{\omega}_P(\mathcal{H}_i \cup \{c(d_i)\}) = T^{\omega}_P(T^{\omega}_P(\bigcup_{j\in 0,\ldots,i-1}\{c(d_j)\}) \cup \{c(d_i)\}) = T^{\omega}_P(\bigcup_{j\in 0,\ldots,i}\{c(d_j)\}) = T^{\omega}_{P\cup\bigcup_{j\in 0,\ldots,i}\{c(d_j)\}}(\emptyset) = T^{\omega}_{S_{i+1}}(\emptyset)$. $\mathcal{H}_{i+1} \neq \bot$ if and only if $\mathrm{Th}_{\mathrm{FL}}(S_{i+1})$ is consistent.

Since the sequences S_i and $\mathcal{H}_i$ are monotonic, $\mathcal{H} = T^{\omega}_S(\emptyset)$. ■

The criterion given in Prop. 1(2) carries over to H-extensions:

Proposition 5 *Let $\mathcal{H}$ be an inflationary H-extension of Δ computed by the above algorithm. If for every $\beta \in J(AD_\eta)$, β is consistent with $\mathcal{H}$, then $\mathcal{H}$ is an H-extension of Δ.* □

7 Application to Inheritance

For inheritance, only defaults of the form given in D_{inh} are used. D_{inh} incorporates a design decision which gives a higher priority to preemption than to refinement. This decision can be argued for or against – for a discussion which defends our decision, see Section 11.4.

For the forward-chaining strategy, the class hierarchy in S is not completely known when computing S_i. Instead, the fragment already known in S_{i-1} must be used for checking the consistency of the justifications. In D_{inh}, a justification can be annulled in later steps only when some path is chosen which is not preempted in S_i, but it turns out to be preempted in later steps. This can be due to one of the following effects:

(P1): for some class c', which is already known in S_i to be an intermediate class on the path, c'[m•→v] turns out to be inconsistent. This can be solved by fixing c'[m•→v] for all intermediate classes c' (see the remainder of this Section).

(P2): in a later step, a new intermediate class-membership o isa c' :: c on this path is derived for which c'[m•→v] is inconsistent.

By avoiding (P1) and (P2), *cautious* computations are obtained. As stated in Proposition 3, a cautious inflationary extension S is an extension of Δ_P only if it is not a *garden path*, i.e., $GD^+(S, D_P) = \emptyset$ (cf. Examples 8 and 9).

(P1) can be solved easily, simultaneously solving the problem of decoupling by using the observation stated in Remark 1: In every step, inheritance can only take place from an *immediate* superclass, fixing c'[m•→v] for all visited classes c'. We restate Remark 1 for defaults in F-Logic:

Proposition 6 *Let P be an F-Logic program, S an extension of Δ_P. Let $d =$ o isa c, c[m•→v] : .../o[m→v] be an instance of an inheritance default of* *the form given in D_P which is applied in the construction of S. Then, there is a sequence of immediate superclasses* $\mathsf{o} - \mathsf{c}_1 - \ldots - \mathsf{c}_n - \mathsf{c}$ *such that for every i,* $\mathsf{c}_i[\mathsf{m} \bullet\!\!\to \mathsf{v}] \in S$*. (analogous for* $\mathsf{c}' :: \mathsf{c}$*.)* □

Proof The justification of d requires that it is consistent with S to assume that for all intermediate classes c', $\mathsf{c}'[\mathsf{m} \bullet\!\!\to \mathsf{v}]$ is consistent. Thus, for every c', the default $\mathsf{c}' :: \mathsf{c},\ \mathsf{c}[\mathsf{m} \bullet\!\!\to \mathsf{v}] : \ldots / \mathsf{c}'[\mathsf{m} \bullet\!\!\to \mathsf{v}]$ is applicable in S. Since S is an extension, its consequence must be contained in S, i.e., $\mathsf{c}'[\mathsf{m} \bullet\!\!\to \mathsf{v}] \in S$. ■

Thus, a step-by-step inheritance strategy downwards the class hierarchy can be formulated, thereby avoiding (P1):

Definition 11 Let

$$D^+_{inh} := \frac{O \text{ isa } C\ ,\ C[M \bullet\!\!\to V]\ :\ \neg\exists C'(O \text{ isa } C' \wedge C' :: C)\ ,\ O[M \to V]}{O[M \to V]}\ ,$$

for an F-Logic program P, let D^+_P be defined like D_P with the schema given in D^+_{inh} instead of D_{inh}. □

Proposition 7 *For every F-Logic program P, the following sets coincide:*

- *the set of inflationary extensions of Δ^+_P, and*
- *the set of theories which can be computed by prefixes of computations of inflationary extensions of Δ_P such that (P1) does not occur.* □

Proof Completeness: Every application of a default over n intermediate classes to a receiving subclass or object can be split into n elementary steps, of inheritance to an immediate subclass (which is already known at this stage) or to the target object. In every step, the condition $\mathsf{c}'[\mathsf{m} \bullet\!\!\to \mathsf{v}]$ is fixed for an intermediate class c'. This also excludes (P1).
Correctness: By Proposition 6, $\mathsf{c}'[\mathsf{m} \bullet\!\!\to \mathsf{v}] \in S$ for every intermediate class c'. ■

Corollary 1 *For every F-Logic program P, the following sets coincide:*

- *the set of inflationary extensions of Δ^+_P which can be computed such that (P2) does not occur, and*
- *the set of extensions of Δ_P.* □

Thus, by avoiding (P1), D^+_{inh} is a valid step to a correct inflationary strategy for inheritance. Avoiding (P1) does not introduce garden paths.

The inflationary strategy using D^+_{inh} (which is implemented in FLORID) deviates from the "real" semantics only when *after* inheritance, a new intermediate class-membership is derived which preempts the path (P2).

Proposition 8 (Static Class Hierarchy) *For an F-Logic program P with a static class hierarchy, the set of extensions of Δ_P and the set of inflationary extensions of Δ^+_P coincide.* □

A non-static class hierarchy, (P2), and *garden paths* will be considered in Section 9.2.

Note that D^+_{inh} is designed only for use with the *inflationary* strategy: the part of the justification concerning potential intermediate classes has been dropped (the justification is only checked against the current fragment of an extension; intermediate classes are precluded by the precondition – thus, it would be trivial).

Regarding the sets of *extensions* of Δ_P and Δ^+_P, Δ^+_P is too strong. Since in D^+_{inh}, the existence of an intermediate subclass is excluded in the justification, with the *non-inflationary* strategy, this condition is stated against the *final* theory S.

The following example illustrates the relationship between Δ_P and Δ^+_P, wrt. the inflationary and the non-inflationary strategy:

Example 10

Consider the following program, where an intermediate subclass membership is derived *after* inheritance has taken place:

$$P := \{\mathsf{c[m{\bullet}\!\!\rightarrow v]}\ ,\ \ \mathsf{o\ isa\ c}\ ,$$
$$\mathsf{o\ isa\ c'} \leftarrow \mathsf{o[m\rightarrow v]}\ ,\ \ \mathsf{c' :: c} \leftarrow \mathsf{o\ isa\ c'}\}\ .$$

The only extension of Δ_P is

$$\Phi = \mathrm{Th}(P \cup \{\mathsf{o[m\rightarrow v]}\ ,\ \mathsf{o\ isa\ c'}\ ,\ \mathsf{c' :: c}\ ,\ \ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}\})\ ,$$

with the extension base $S = P \cup \{\mathsf{o[m\rightarrow v]}\ ,\ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}\}$, the set of ground facts in Φ is

$$\mathsf{facts}(\Phi) = \{\mathsf{c[m{\bullet}\!\!\rightarrow v]}\ ,\ \ \mathsf{o\ isa\ c}\ ,\ \ \mathsf{o[m\rightarrow v]}\ ,\ \mathsf{o\ isa\ c'}\ ,\ \mathsf{c' :: c}\ ,\ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}\}\ .$$

Φ is computed by

$$
\begin{aligned}
&S_0 = P\ ,\\
&\mathsf{facts}(\mathrm{Th}(S_0)) = \{\mathsf{c[m{\bullet}\!\!\rightarrow v]}\ ,\ \mathsf{o\ isa\ c}\}\ ,\\
&GD(S, S_0, D_P) = \{\mathsf{o\ isa\ c,c[m{\bullet}\!\!\rightarrow v]}\!:\ \ldots /\ \mathsf{o[m\rightarrow v]}\}\ ,\\
&S_1 = P \cup \{\mathsf{o[m\rightarrow v]}\}\ ,\\
&\mathsf{facts}(\mathrm{Th}(S_1)) = \{\mathsf{c[m{\bullet}\!\!\rightarrow v]}\ ,\ \mathsf{o\ isa\ c}\ ,\ \mathsf{o[m\rightarrow v]}\ ,\ \mathsf{o\ isa\ c'}\ ,\ \mathsf{c' :: c}\}\ ,\\
&GD(S, S_1, D_P) = GD(S, S_0, D_P) \cup \{\mathsf{c' :: c,c[m{\bullet}\!\!\rightarrow v]}\!:\ \ldots /\ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}\}\ ,\\
&S_2 = P \cup \{\mathsf{o[m\rightarrow v]}\ ,\ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}\}\ ,\\
&GD^+(S_2, D_P) = \emptyset\ ,\\
&\Phi = \mathrm{Th}_{\mathrm{FL}}(S_2)\ .
\end{aligned}
$$

On the other hand, using D^+_P, $GD(S, S_0, D^+_P) = \emptyset$: The default $\mathsf{o\ isa\ c,c[m{\bullet}\!\!\rightarrow v]}$: $\ldots /\ \mathsf{o[m\rightarrow v]}$ is *not* applicable since in $\mathrm{Th}(S)$ there is the intermediate class c' between c and o. The default $\mathsf{c' :: c,c[m{\bullet}\!\!\rightarrow v]}$: $\ldots /\ \mathsf{c'[m{\bullet}\!\!\rightarrow v]}$ is also not applicable since in $\mathrm{Th}(S_0)$, this class does not yet exist.

For the *inflationary* strategy with D^+, the computation sequence is the same as above: $GD^+(S_0, D^+_P) = \{\mathsf{o\ isa\ c,c[m{\bullet}\!\!\rightarrow v]}\!:\ \ldots /\ \mathsf{o[m\rightarrow v]}\}$, since the intermediate class is not yet known and the justifications are checked against the *current* theory. □

For the same reason, a cautious strategy with D^+_{inh} makes no sense: it would prohibit the derivation of (non-preempting) intermediate class-memberships (cf. (P2)) on a path where inheritance has already taken place.

In the next section, we describe the trigger-based inheritance mechanism defined for F-Logic [KLW95] which is implemented in the FLORID system. The mechanism implements exactly the non-cautious inflationary strategy for D^+_{inh} (this will be shown in Theorem 1).

8 Inheritance via Inheritance Triggers

The deductive part of F-Logic programs is evaluated wrt. an inflationary fixpoint semantics (cf. Def. 3), additionally, user-defined stratification is supported. Non-monotonic inheritance is implemented via a trigger mechanism in a *deduction precedes inheritance* manner: The evaluation of a program is defined by alternatingly computing a classical deductive fixpoint and carrying out a specified amount of inheritance. The strategy is formally characterized as follows, based on *inheritance triggers*:

Definition 12 (Inheritance Triggers) Let $\mathcal{H}$ be an H-structure.

- An *inheritance trigger* in $\mathcal{H}$ is a pair $(o\sharp c, m{\bullet\!\!\to} v)$ such that $(o\sharp c) \in \mathcal{H}$ and $c[m{\bullet\!\!\to} v] \in \mathcal{H}$, and there is no $o \neq c' \neq c$ such that $\{o\sharp c', c' :: c\} \subseteq \mathcal{H}$ (where $\sharp$ stands for isa or $::$).
- An inheritance trigger $(o \;\mathsf{isa}\; c, m{\bullet\!\!\to} v)$ or $(c' :: c, m{\bullet\!\!\to} v)$ is *active* in $\mathcal{H}$ if there is no v' such that $o[m{\to} v'] \in \mathcal{H}$ or $c'[m{\bullet\!\!\to} v'] \in \mathcal{H}$, respectively.
- $\mathbf{T}(\mathcal{H})$ denotes the set of active inheritance triggers in $\mathcal{H}$.
- An inheritance trigger $(o \;\mathsf{isa}\; c, m{\bullet\!\!\to} v)$ or $(c' :: c, m{\bullet\!\!\to} v)$ is *blocked* in $\mathcal{H}$ if $o[m{\to} v'] \in \mathcal{H}$ or $c'[m{\bullet\!\!\to} v'] \in \mathcal{H}$, respectively, for some $v' \neq v$.

Note that this definition depends only on $\mathcal{H}$, not on a program. □

The value of a method is inherited from a class to an object or a subclass only if no other value for this method can be derived for the object or the subclass, respectively. Hence, inheritance is done *after* classical deduction, leading to an alternating sequence of (deductive) fixpoint computations and inheritance steps.

Definition 13 (Firing a Trigger) For an H-structure $\mathcal{H}$ and an active trigger $t = (o \;\mathsf{isa}\; c, m{\bullet\!\!\to} v)$ or $t = (c' :: c, m{\bullet\!\!\to} v)$, the H-structure after firing t, $t(\mathcal{H})$, is defined as $\mathcal{H} \cup \{o[m{\to} v]\}$ or $\mathcal{H} \cup \{c'[m{\bullet\!\!\to} v]\}$, respectively.
In accordance to [KLW95], for an H-structure $\mathcal{H}$ and an active trigger t, $\mathcal{I}^t_P(\mathcal{H}) := T^\omega_P(t(\mathcal{H}))$ denotes the *one step inheritance transformation.*[7] □

Proposition 9 (Correctness of one-step-inheritance) *Let P be a program and $\mathcal{H}$ an H-structure which is a model of P (i.e., $\mathcal{H} \models h \leftarrow b$ for every rule in P). For every $t \in \mathbf{T}(\mathcal{H})$, if $\mathcal{I}^t_P(\mathcal{H}) = T^\omega_P(t(\mathcal{H}))$ is consistent, then it is also a model of P.* □

[7] note that by Def. 3, T^ω_P includes the closure $\mathcal{C}\ell$.

Note that the notion of a model of an F-Logic program does not require closure wrt. inheritance (e.g., in Ex. 5 there exists no model which is closed wrt. inheritance).

In [KLW95], *inheritance-canonic* models of F-Logic programs are defined. The original definition is given for transfinite sequences:

Definition 14 (Inheritance-Canonic Model [KLW95]) For an F-Logic program P, an H-structure $\mathcal{M} \neq \bot$ is an *inheritance-canonic* model if there is a (possibly transfinite) sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_\alpha$ such that

(i) $\mathcal{M}_0 = T_P^\omega(\emptyset)$ and $\mathcal{M}_\alpha = \mathcal{M}$,

(ii) $\mathcal{M}_\alpha$ has no active triggers,

(iii) if γ is a non-limit ordinal, then $\mathcal{M}_\gamma = \mathcal{I}_P^t(\mathcal{M}_{\gamma-1})$ for some active trigger t in $\mathcal{M}_{\gamma-1}$, and

(iv) if γ is a limit ordinal, then $M_\gamma = \bigcup_{\beta<\gamma} \mathcal{M}_\beta$. □

Obviously, an F-Logic program P can have several inheritance-canonic models. For relating the concept of inheritance-canonic models, we reformulate the definition for finite computations.

Definition 15 (Inheritance-Canonic Model) (Finite variant)
For an F-Logic program P, a sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ of H-structures is an *$\mathcal{I}_P$-sequence* if $\mathcal{M}_0 = T_P^\omega(\emptyset)$ and for all i, there is a $t_i \in \mathbf{T}(\mathcal{M}_i)$ such that $\mathcal{M}_{i+1} = \mathcal{I}_P^{t_i}(\mathcal{M}_i)$.

An H-structure $\mathcal{M}$ is an *inheritance-canonic* model of P if there is an $\mathcal{I}_P$-sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M} \neq \bot$ such that $\mathcal{M}$ has no active triggers. □

Proposition 10 *Let P be an F-Logic program. Then,*

- *For finite computations, Definitions 14 and 15 coincide.*
- *If $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ is a $\mathcal{I}_P$-sequence, then every $\mathcal{M}_i$ is a model of P.* □

For computing inheritance-canonic models, the process is stopped if there are no more active triggers. With this, the computation does not stop “in time”, but often gets trapped in an inconsistency (which corresponds to a default whose precondition is satisfied and the method to be inherited is yet undefined, but the result of inheritance is inconsistent). In such cases, the final consistent H-structure is of interest. So, we suggest to extend the F-Logic terminology [KLW95] by the following definition:

Definition 16 (Final Consistent Inheritance-Canonic Models)
For an F-Logic program P, an H-structure $\mathcal{H}$ is a *final consistent inheritance-canonic model* of P if there exists an $\mathcal{I}_P$-sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{H}$ such that $\mathcal{I}_P^t(\mathcal{H}) = \bot$ for every $t \in \mathbf{T}(\mathcal{H})$.

Let $\mathcal{S}_\mathcal{I}(P)$ be the set of final consistent inheritance-canonic models of P. □

Corollary 2 *For an F-Logic program P, every inheritance-canonic model is a final consistent inheritance-canonic model of P.* □

In general, for a final consistent inheritance-canonic model $\mathcal{H}$, $\mathbf{T}(\mathcal{H})$ is not empty, but firing any of these triggers leads to an inconsistency, i.e., $\mathcal{I}_P^t(\mathcal{H}) = \bot$ for every $t \in \mathbf{T}(\mathcal{H})$.
The following example illustrates the trigger mechanism and the propagation of properties through a class hierarchy, considering possible preemption:

Example 11 (Inheritance in a Class Hierarchy) Consider the program from Example 6:

$P = \{\mathsf{bird[laying\_eggs{\bullet\!\!\to}true;fly{\bullet\!\!\to}true]},\ \mathsf{penguin[fly{\bullet\!\!\to}false]},$
$\quad \mathsf{penguin :: bird},\ \mathsf{tweety\ isa\ penguin}\},$

Starting with $S_0 = P$

$$GD(S_0, \Delta_P) = \left\{ \frac{\mathsf{tweety\ isa\ penguin, penguin[fly{\bullet\!\!\to}false]}\ :\ \neg\exists\, \mathsf{C'(tweety\ isa\ C'} \wedge \mathsf{C' :: penguin)}\ ,\ \mathsf{tweety[fly \to false]}}{\mathsf{tweety[fly \to false]}}\ , \right.$$
$$\left. \frac{\mathsf{penguin :: bird, bird[laying\_eggs{\bullet\!\!\to}true]}\ :\ \neg\exists\, \mathsf{C'(penguin :: C'} \wedge \mathsf{C' :: bird)}\ ,\ \mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}}{\mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}} \right\} .$$

Note that there is no applicable default inheriting $\mathsf{bird[laying\_eggs{\bullet\!\!\to}true]}$ directly to tweety since there is the intermediate class $\mathsf{penguin}$. Applying the first default, $S_1 = P \cup \{\mathsf{tweety[fly \to false]}\}$ and

$$GD(S_1, \Delta_P) = \left\{ \frac{\mathsf{penguin :: bird, bird[laying\_eggs{\bullet\!\!\to}true]}\ :\ \neg\exists\, \mathsf{C'(penguin :: C'} \wedge \mathsf{C' :: bird)}\ ,\ \mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}}{\mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}} \right\} .$$

Applying this default,
$S_2 = P \cup \{\mathsf{tweety[fly \to false]}\} \cup \{\mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}\}$ and

$$GD(S_2, \Delta_P) = \left\{ \frac{\mathsf{tweety\ isa\ penguin, penguin[laying\_eggs{\bullet\!\!\to}true]}\ :\ \neg\exists\, \mathsf{C'(tweety\ isa\ C'} \wedge \mathsf{C' :: penguin)}\ ,\ \mathsf{tweety[laying\_eggs \to true]}}{\mathsf{tweety[laying\_eggs \to true]}} \right\} .$$

Finally,

$$S_3 = P \cup \{\mathsf{tweety[fly \to false;laying\_eggs \to true]}\} \cup \{\mathsf{penguin[laying\_eggs{\bullet\!\!\to}true]}\} .$$

is an extension base.
Analogously, for H-structures, starting with $\mathcal{H}_0 = T_P^\omega(P) = P$,

$$\mathbf{T}(\mathcal{H}_0) = \{(\mathsf{tweety\ isa\ penguin, fly{\bullet\!\!\to}false}), (\mathsf{penguin :: bird, laying\_eggs{\bullet\!\!\to}true})\}$$

By firing the first one, we get $\mathcal{H}_1 = T_P^\omega(S_1)$ and

$$\mathbf{T}(\mathcal{H}_1) = \{(\mathsf{penguin :: bird, laying\_eggs{\bullet\!\!\to}true})\} ,$$

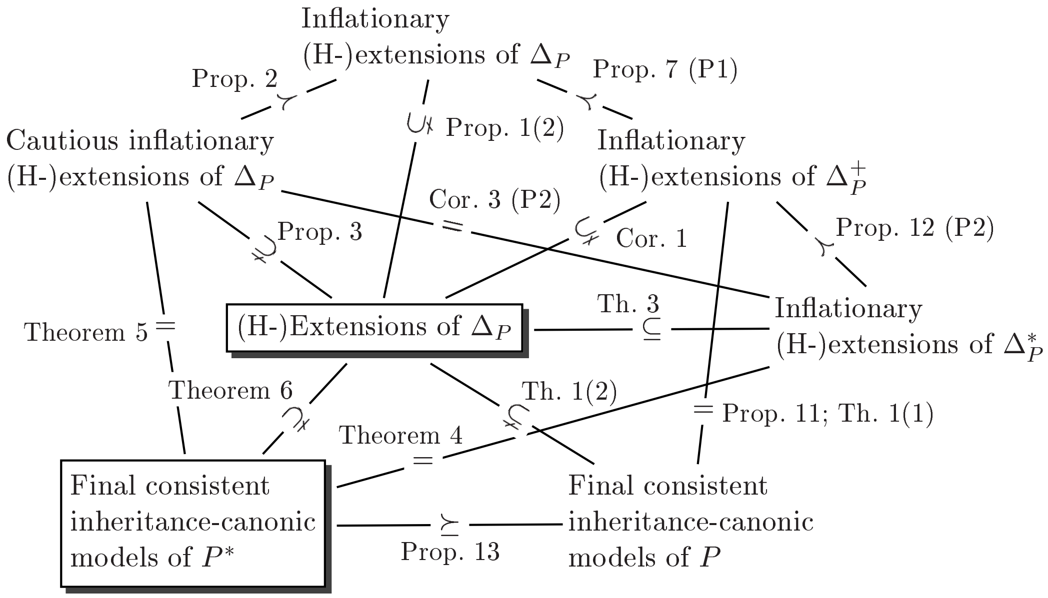
resulting in $\mathcal{H}_2 = T_P^\omega(S_2)$ with

$$\mathbf{T}(\mathcal{H}_2) = \{(\mathsf{tweety\ isa\ penguin, laying\_eggs{\bullet\!\!\to}true})\}$$

and $\mathcal{H}_3 = T_P^\omega(S_3)$. □

9 Comparison

In this section, the relationships between the concepts of extensions, inflationary (H-)extensions, and inheritance-canonic H-structures are investigated, and criteria for isolating one class from the other are given.
In anticipation of the results of this section, these concepts compare as shown in Figure 3.



($M_1 \prec M_2$ denotes that every structure/theory in M_2 can be extended to one in M_1.)

Figure 3: Comparison of the concepts

9.1 Inheritance-Canonic Models and Inflationary H-Extensions

The computation of inheritance-canonic models implements the process described in Proposition 4 for computing inflationary H-extensions of the default theory Δ_P^+:

Proposition 11
Let P be an F-Logic program. Then, the following sets coincide:

- *the set of $\mathcal{I}_P$-sequences (cf. Definition 15) $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ such that $\mathcal{M}_n \neq \bot$, and*
- *the set of prefixes $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n$ of sequences of H-structures as described in Proposition 4 for Δ_P^+ (computation of inflationary H-extensions).*

□

Proof by induction on the length of the sequence:

$i=1$: There is exactly one sequence of length 1: $\mathcal{H}_0 = T_P^\omega(\emptyset) = \mathcal{M}_0$.
$i \to i{+}1$: Consider a sequence $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{i\text{-}1}$ of length i.
$\supseteq$: Let

$$d := \frac{\begin{array}{l}(o \text{ isa } c, c[m \bullet\!\!\to v]) \ : \\ \quad (\neg\exists C'(o \text{ isa } C' \wedge C' :: c)\ ,\ o[m \to v])\end{array}}{o[m \to v]} \in GD^+(\mathcal{H}_{i\text{-}1}, \Delta_P^+) \ .$$

Then,

- $\{o \text{ isa } c, c[m \bullet\!\!\to v]\} \subseteq \mathcal{H}_{i\text{-}1}$,
- $\mathrm{Th}_{\mathrm{FL}}(P \cup \mathcal{H}_{i\text{-}1} \cup \{\neg\exists C'(o \text{ isa } C' \wedge C' :: c)\})$ is consistent,
- $\mathrm{Th}_{\mathrm{FL}}(P \cup \mathcal{H}_{i\text{-}1} \cup \{o[m \to v]\})$ is consistent, and
- $o[m \to v] \notin \mathcal{H}_{i\text{-}1}$,

thus, $t(d) := (o \text{ isa } c, m \bullet\!\!\to v) \in \mathbf{T}(\mathcal{H}_{i\text{-}1})$ and firing it results in the consistent H-structure $T_P^\omega(\mathcal{H}_{i\text{-}1} \cup \{o[m \to v]\})$. Let $d \in GD^+(\mathcal{H}_{i\text{-}1}, \Delta_P^+)$ be the ground default which is applied in the step from $\mathcal{H}_{i\text{-}1}$ to $\mathcal{H}_i = T_P^\omega(\mathcal{H}_{i\text{-}1} \cup \{o[m \to v]\})$. Then $\mathcal{H}_i = \mathcal{I}_P^{t(d)}(\mathcal{H}_{i\text{-}1})$ is a valid inheritance step.
Thus, the continuation $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{i\text{-}1}, \mathcal{H}_i$ wrt. Proposition 4 is an $\mathcal{I}_P$-sequence.

$\subseteq$: Let $t = (o \text{ isa } c, m \bullet\!\!\to v) \in \mathbf{T}(\mathcal{H}_{i\text{-}1})$ s.t. $\mathcal{M}_i := \mathcal{I}_P^t(\mathcal{H}_{i\text{-}1}) = T_P^\omega(t(\mathcal{H}_{i\text{-}1})) \neq \perp$. Then,

- $\{o \text{ isa } c, c[m \bullet\!\!\to v]\} \subseteq \mathcal{H}_{i\text{-}1}$,
- there is no intermediate class $c' \neq c$ such that $\{o \text{ isa } c', c' :: c\} \subseteq \mathcal{H}_{i\text{-}1}$, thus, $\mathrm{Th}_{\mathrm{FL}}(P \cup \mathcal{H}_{i\text{-}1} \cup \{\neg\exists C'(o \text{ isa } C' \wedge C' :: c)\})$ is consistent,
- there is no v' such that $o[m \to v'] \in \mathcal{H}_{i\text{-}1}$, and
- since $T_P^\omega(t(\mathcal{H}_{i\text{-}1})) \neq \perp$, $\mathrm{Th}_{\mathrm{FL}}(P \cup \mathcal{H}_{i\text{-}1} \cup \{o[m \to v]\})$ is also consistent.

Thus, the corresponding ground instance $[o/O, c/c, m/M, v/V]$ of the default schema D_{inh}^+ is in $GD^+(\mathcal{H}_{i\text{-}1}, \Delta_P^+)$, and we have $\mathcal{H}_{i\text{-}1} \cup \{c(d)\} = t(\mathcal{H}_{i\text{-}1})$ and $\mathcal{M}_i = T_P^\omega(\mathcal{H}_{i\text{-}1} \cup \{c(d)\})$. Hence, the continued $\mathcal{I}_P$-sequence $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_{i\text{-}1}, \mathcal{M}_i$ is also a prefix of a computation as given in Prop. 4.

(both directions analogous for inheritance to subclasses.) ■

Theorem 1 ($\mathcal{I}_P$-sequences and inflationary H-Extensions)
Let P be an F-Logic program. Then,

1. *the set $\mathcal{S}_\mathcal{I}(P)$ of final consistent inheritance-canonic models of P is the set of inflationary H-extensions of Δ_P^+.*
2. *A final consistent inheritance-canonic model $\mathcal{H} \in \mathcal{S}_\mathcal{I}(P)$ is an H-extension of Δ_P if and only if there is an $\mathcal{I}_P$-sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{H}$ such that (P2) does not occur.* □

Proof 1. The inclusion is shown in both directions:
$\subseteq$: Let $\mathcal{H} \in \mathcal{S}_\mathcal{I}(P)$. By Definition 16, there are no triggers in $\mathbf{T}(\mathcal{H})$ whose firing leads to a consistent H-structure. This condition is equivalent with

$GD^+(\mathcal{H}, D_P) = \emptyset$. Thus, by the equivalence given in Proposition 11, $\mathcal{H}$ is an inflationary H-extension of Δ_P.

$\supseteq$: follows immediately from Proposition 11 and the stopping criteria of both characterizations.

2. Follows from (1), Proposition 11, and Corollary 1. ■

With this, Proposition 8 can be extended to final consistent inheritance-canonic models:

Theorem 2 (Static Class Hierarchy) *For an F-Logic program P with a static class hierarchy, i.e. no* isa *-atom or* :: *-atom occurs in any non-fact rule head, the set of final consistent inheritance-canonic models of P and the set of H-extensions of Δ_P coincide.* □

9.2 Cautious Inflationary Extensions for Inheritance

The relationship between extensions and inflationary extensions has been clarified by Proposition 1(2), giving a criterion for identifying inflationary extensions which are no extensions: an inflationary extension S is an extension if every justification of every default which is applied in the computation of S is consistent with $\mathrm{Th}(S)$. By the concept of cautious inflationary extensions this property has been enforced allowing a forward-chaining construction.

For defaults of the form D_{inh} occuring in the default theory of an F-Logic program, the only justification which can be invalidated by later steps is that the path which has been used for inheritance turns out to be preempted. In Section 7, we have split this requirement into two parts: (P1) for the subclasses known at the time where the default has been applied, and (P2) for intermediate class-memberships which are derived in subsequent steps.

(P1) has been solved by using the revised default schema D^+_{inh} with the inflationary strategy (cf. Proposition 7).

In presence of a non-static class hierarchy, (P2) can be termed as *postemption*[8]:

Example 12 Consider the following program which inserts a postempting intermediate class-membership *after* inheritance has taken place:

$P = \{\mathsf{cl1[m{\bullet}\!\!\to v1]},\ \mathsf{x\ isa\ cl1},\ \mathsf{cl2 :: cl1},\ \mathsf{cl2[m{\bullet}\!\!\to v2]},\ \ \mathsf{x\ isa\ cl2} \leftarrow \mathsf{x[m\to v1]}\}$.

The only computation sequence is

T_P^ω: $\{\mathsf{x\ isa\ cl1},\ \mathsf{cl2 :: cl1},\ \mathsf{cl1[m{\bullet}\!\!\to v1]},\ \mathsf{cl2[m{\bullet}\!\!\to v2]}\}$

Inh.:$\{\mathsf{x\ isa\ cl1},\ \mathsf{cl2 :: cl1},\ \mathsf{cl1[m{\bullet}\!\!\to v1]},\ \mathsf{cl2[m{\bullet}\!\!\to v2]},\ \mathsf{x[m\to v1]}\}$

T_P^ω: $\{\mathsf{x\ isa\ cl1},\ \mathsf{cl2 :: cl1},\ \mathsf{x\ isa\ cl2},\ \mathsf{cl1[m{\bullet}\!\!\to v1]},\ \ \mathsf{cl2[m{\bullet}\!\!\to v2]},\ \mathsf{x[m\to v1]}\}$,

[8]in contrast to preemption, where inheritance is not applied due to an already known intermediate class.

which yields an inflationary H-extension where postemption occurs: inheritance from $\mathsf{cl1}$ to x is postempted by the intermediate class $\mathsf{cl2}$ although it has been justified (i.e., the trigger has been active). There is *no* "justified" model since inheritance is postempted exactly if it takes place. Note that this is not a *logical* inconsistency as in Ex. 5 which would prohibit inheritance. Here, P has no extension; a similar *cyclic* inheritance network is given in [Hor94, Sec. 2.3.1] as an example for a network which does not have a (credulous) extension. $\square$

Cautious inflationary computations can be enforced by augmenting the consequence of D^+_{inh} such that it forces all newly derived intermediate class-memberships to provide the appropriate inheritable method:

Definition 17 Let

$$D^*_{inh} := \frac{O \text{ isa } C, C[M \bullet\!\!\to V] \ : \ \neg\exists C'(O \text{ isa } C' \wedge C' :: C) \ , \ O[M \to V]}{\forall C'((O \text{ isa } C' \wedge C' :: C) \to C'[M \bullet\!\!\to V]) \ , \ O[M \to V]}$$

For an F-Logic program P, let D^*_P be defined like D_P with D^*_{inh} instead of D_{inh}. $\square$

Note that the consequence in D^*_{inh} is no longer a set of atoms, thus, the generated extension bases also contain universal Horn formulas of the form $\forall C'((O \text{ isa } C' \wedge C' :: C) \to C'[M \bullet\!\!\to V])$ which represent explicit knowledge about (potential) intermediate classes where inheritance has taken place. For an extension base S of Δ^*_P, let $\mathsf{strip}(S)$ denote S without these formulas. These formulas are only needed in course of the computation for avoiding preemption – facts induced by S and $\mathsf{strip}(S)$ are the same:

Extending Proposition 7, Δ^*_P also avoids the invalidation of a justification by (P2):

Proposition 12
Let P be an F-Logic program. Then, the following sets coincide:

- *the set of $Th(\mathsf{strip}(S))$ such that S is an inflationary extension base of Δ^*_P, and*
- *the set of theories which can be computed by prefixes of computations of inflationary extensions of Δ^+_P such that (P2) does not occur.* $\square$

Corollary 3 *Let P be an F-Logic program. Then, the following sets coincide also with the sets given in Proposition 12:*

- *the set of theories which can be computed by prefixes of computations of inflationary extensions of Δ_P such that neither (P1) nor (P2) occur, and*
- *the set of cautious inflationary extensions of Δ_P,* $\square$

Proof follows immediately from Prop. 7 and Prop. 2(1). $\blacksquare$

Note that the above theories can contain garden paths by avoiding (P2). Here, the criterion of Proposition 3 can be applied. D^*_{inh} turns out to be a good approximation for computing extensions:

Theorem 3
Let P be an F-Logic program. Then, the following sets coincide:
- *the set of $Th(\mathsf{strip}(S))$ such that S is an inflationary extension base of* Δ^*_P and $GD^+(\mathsf{strip}(S), D_P) = \emptyset$, *and*
- *the set of extensions of Δ_P.* □

Proof follows from Proposition 12/Corollary 3 and Proposition 3. ■

The proof can alternatively be given directly, using the concepts instead of propositions and corollaries:

Proof The proof is divided into two parts, each for inclusion in one direction:
1. For every extension base S of Δ_P, there is an (inflationary) extension base S' of Δ^*_P such that $S = \mathsf{strip}(S')$:
   By [MT93, Cor. 3.71], every extension base S of Δ_P is generated by a sequence of applications of defaults which satisfies criterion Prop. 1(2), i.e., which is a cautious sequence. Let S' be the extension base which is generated by the corresponding sequence using Δ^*_P. Then, $\mathsf{strip}(S') = S$.
2. Let S' be an extension of Δ^*_P such that $GD^+(\mathsf{strip}(S'), D_P) = \emptyset$. Then, $\mathsf{strip}(S')$ is an extension base of Δ_P:
   S' is generated by a sequence of applications of defaults using Δ^*_P. Let S be the set of formulas which is generated by the corresponding sequence using Δ_P. Then, by construction of Δ^*_P, $S = \mathsf{strip}(S')$ is a cautious inflationary extension base of Δ_P. By assumption, $GD^+(S, D_P) = \emptyset$, thus, by Proposition 3, $S = \mathsf{strip}(S')$ is an extension base of Δ_P. ■

Remark 7 Consider again part (2) of the above proof. In contrast, assume $GD^+(\mathsf{strip}(S'), D_P) \neq \emptyset$. By assumption, $GD^+(S', D_P) = \emptyset$, thus, for every $d \in GD^+(\mathsf{strip}(S'), D_P)$, $\mathrm{Th}(\mathsf{strip}(S') \cup c(d))$ is consistent, but $\mathrm{Th}(S' \cup c(d))$ is inconsistent. This inconsistency must be due to the additional knowledge which is added for the justifications of previously applied defaults, i.e., an intermediate class would be inserted where inheritance has already taken place. □

Since the consequences in D^*_{inh} are no longer sets of atoms, D^*_{inh} cannot be directly translated to H-extensions and inheritance-canonic models. In the following section, these notions are integrated by extending the program P appropriately.

9.3 Inheritance-Canonic Models and Cautious Extensions

The intended semantics of an F-Logic program P are the H-extensions of Δ_P. Thus, the set of $\mathcal{I}_P$-sequences has to be restricted to sequences which are (P2)-free, i.e., where no postemption occurs, resulting in H-extensions or at least in cautious inflationary H-extensions.

The universal Horn formulas added to the consequence in the step from D^+_{inh} to D^*_{inh} can be implemented by adding a rule

$$r(t) \;:=\; \mathsf{C[m}{\bullet\!\!\to}\mathsf{v]} \leftarrow \mathsf{o} \sharp\ \mathsf{C},\ \mathsf{C} :: \mathsf{c}$$

to the program whenever an inheritance trigger $t = (o\sharp c, m{\bullet\!\!\to}v)$ is fired. This requires only a slight modification in the concept of $\mathcal{I}_P$-sequences:

Definition 18
For an F-Logic program P, a sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ of H-structures is an $\mathcal{I}^*_P$*-sequence* if $\mathcal{M}_0 = T^\omega_P(\emptyset)$ and for all i, there is a $t_i \in \mathbf{T}(\mathcal{M}_i)$ such that $\mathcal{M}_{i+1} = \mathcal{I}^{t_i}_{P_{i+1}}(\mathcal{M}_i) \neq \bot$ where $P_0 = P$ and $P_{i+1} = P_i \cup r(t_i)$.
Let $\mathcal{S}^*_\mathcal{I}(P)$ be the set of H-structures $\mathcal{H}$ such that there exists an $\mathcal{I}^*_P$-sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n = \mathcal{H}$, and $\mathcal{I}^t_{P_n \cup r(t)}(\mathcal{H}) = \bot$ for every $t \in \mathbf{T}(\mathcal{H})$. □

Proposition 13 ($\mathcal{I}_P$- and $\mathcal{I}^*_P$-sequences)
Let P be an F-Logic program. Then,
1. *for every $\mathcal{I}^*_P$-sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$, every $\mathcal{M}_i$ is a model of P,*
2. *every $\mathcal{I}^*_P$-sequence is (P2)-free, and*
3. *every $\mathcal{I}^*_P$-sequence is a prefix of an $\mathcal{I}_P$-sequence.* □

Proof
1. $\mathcal{M}_i = \mathcal{I}^t_{P_i}(\mathcal{M}_{i-1}) = T^\omega_{P_i}(t_i(\mathcal{M}_{i-1}))$ where $P_i = P \cup r(\{t_j \mid 0 \le j \le i\})$.
2. The construction of the P_i guarantees that (P2) cannot occur.
3. By induction: In every step, if $T^\omega_{P_i}(t_i(\mathcal{M}_{i-1}))$ is consistent, then $\mathcal{I}^{t_i}_{P_i}(\mathcal{M}_{i-1}) = T^\omega_{P_i}(t_i(\mathcal{M}_{i-1})) = T^\omega_P(t_i(\mathcal{M}_{i-1})) = \mathcal{I}^{t_i}_P(\mathcal{M}_{i-1})$, thus, the step $\mathcal{M}_{i-1} \to \mathcal{M}_i$ is also an $\mathcal{I}_P$-step. ■

Theorem 4 *Let P be an F-Logic program. Then,*

$$\mathcal{S}^*_\mathcal{I}(P) = \{T^\omega_{\mathsf{strip}(S)} \mid S \textit{ is an inflationary extension base of } \Delta^*_P\} \quad . \qquad \square$$

Proof Analogous to Theorem 1(1). ■

Theorem 5 ($\mathcal{I}^*_P$-sequences and cautious H-Extensions)
Let P be an F-Logic program. Then, $\mathcal{S}^*_\mathcal{I}(P)$ is the set of cautious inflationary H-extensions of Δ_P. □

Proof Follows from Proposition 12/Corollary 3. Alternatively, one could use the same inductive argumentation as in the proofs of Propositions 11 and 4 by investigating the T_P-rounds. ■

There is still the discrepancy between $\mathcal{S}^*_{\mathcal{I}}(P)$ (corresponding to cautious inflationary extensions) and (H-)extensions due to *garden paths* (cf. Proposition 3):

Theorem 6 ($\mathcal{I}^*_P$-sequences and H-Extensions)
Let P be an F-Logic program. Then, the following statements are equivalent:
- $\mathcal{H} \in \mathcal{S}^*_{\mathcal{I}}(P)$ *and* $\mathcal{I}^t_P(\mathcal{H}) = \bot$ *for every* $t \in \mathbf{T}(\mathcal{H})$,
- $\mathcal{H}$ *is an H-extension of* Δ_P. □

Proof Here, we use the fact that every $\mathcal{H} \in \mathcal{S}^*_{\mathcal{I}}(P)$ is computed by augmenting the program by $r(t)$ for every fired trigger t. The cautionsness of these rules avoids that garden paths are postempted by introducing intermediate classes which have not been known when the respective default d had been applied: the firing of any trigger which would cause the garden path to be postempted leads to an inconsistency due to $r(d)$.

Thus, if there is a trigger t' in $\mathcal{H}$ which would be applicable consistently wrt. the original program P, then t' would postempt some garden path, i.e., $\mathcal{H}$ is not a valid H-extension. If no such trigger exists, there are no garden paths, and $\mathcal{H}$ is an H-extension. Formally, if $\mathcal{I}^t_P(\mathcal{H}) = \bot$ for every $t \in \mathbf{T}(\mathcal{H})$, then $GD^+(S, D_P) = \emptyset$ and the claim follows from Proposition 3. ■

This is all we can do. When applying a default, it is in general not decidable if it is a garden path. Thus, the only way to compute the extensions is to compute $\mathcal{S}^*_{\mathcal{I}}(P)$ and to discard those H-structures which contain garden paths by the above criterion.

A First Conclusion With the strategies of $\mathcal{I}_P$- and $\mathcal{I}^*_P$-sequences, the semantics of cautions and non-cautious *inflationary* extensions can be implemented in F-Logic in a forward-chaining way. *Inflationary* means that no application of a default is undone.

10 Miscellaneous

10.1 Element-oriented vs. Set-oriented Application

Comparing Definitions 5 and 8, one sees that in the first case, all applicable defaults are applied in a step where in the latter case, an individual default is chosen. Sections 6 and 8 follow the latter strategy. In this section, we investigate the situation where an arbitrary subset of applicable defaults is applied or an arbitrary set of active triggers is fired, respectively.

10.1.1 Default Theories

Proposition 14 *Let $\Delta = (D, F)$ be a Default theory. Let $F = S'_0, S'_1, \ldots,$ η', and S' as in Definition 8 (inflationary extensions), except that*

$$S'_{i+1} = S'_i \cup c(D_i)\ ,\ AD'_{i+1} = AD'_i \cup D_i \quad \textit{where } D_i \subseteq GD^+(S'_i, D)\ .$$

Then, the set of $Th(S')$ which are computed by the above sequences is a superset of the set of inflationary extensions of Δ. □

Proof For a given sequence $S_0, S_1, \ldots$ and $d_0, d_1, \ldots$ resulting in S, choose $D_i = \{d_i\}$. ■

Example 13 There are $\mathrm{Th}(S')$ as above which are no inflationary extensions: Let $GD^+(P, D) = \{d_1, d_2\}$ such that $\mathrm{Th}(F \cup c(d_1) \cup c(d_2))$ is consistent, but $\mathrm{Th}(F \cup c(d_1)) \cup \{J(d_2)\}$ and $\mathrm{Th}(F \cup c(d_2)) \cup \{J(d_1)\}$ both are inconsistent. Then, $\mathrm{Th}(F \cup c(d_1))$ and $\mathrm{Th}(F \cup c(d_2))$ are the only inflationary extensions. On the other hand, with $D_0 = \{d_1, d_2\}$, $\mathrm{Th}(S') = \mathrm{Th}(F \cup c(d_1) \cup c(d_2))$ which is not an inflationary extension. $\mathrm{Th}(S')$ does not satisfy the criterion given in Prop. 1(2), thus it is not an extension of Δ. □

There are computation sequences (yielding S' as above) which can be obtained by set-oriented application, but not by element-wise application, but all these do not satisfy the criterion given in Prop. 1(2):

Proposition 15
Let $\Delta = (D, F)$, $F = S'_0, S'_1, \ldots, S'_{\eta'}$ and S' as in Proposition 14.
Then, the set of Th(S') which are computed by the above sequences and satisfy the criterion given in Proposition 1(2) coincides with the set of inflationary extensions of Δ which satisfy this criterion. □

Proof The inclusion from unary sets to arbitrary sets is trivial. For the other direction, let $S'_0 = F, S'_1, \ldots$ be such a sequence with $D_0, D_1, \ldots$ and $d_{i,0}, \ldots, d_{i,k_i}$ an enumeration of D_i.
Then, the sequence $S_{0,0} = F, \ldots, S_{0,k_0}, S_{1,0}, \ldots, S_{1,k_1}, \ldots$ obtained by application of $\{d_{0,0}\}, \ldots, \{d_{0,k_0}\}, \{d_{1,0}\}, \ldots, \{d_{1,k_1}\}, \ldots$ is a possible computation sequence which satisfies the given criterion:

- obviously, $S_{i,j} = F \cup \{c(d_{i',j'}) \mid (i', j') \leq (i, j)\}$; thus, $S_{i,0} = S'_i$, $S_{i,j} = S_{i,0} \cup \{c(d_{i,j'}) \mid j' \leq j\}$, and $S = S'$.
- $d_{i,j}$ is applicable in $S_{i,0}$ for all j: Since $d_{i,j} \in D_i$, $\mathrm{Th}(S'_i) \models p(d_{i,j})$ and $\mathrm{Th}(S'_i) \cup \{\beta\}$ is consistent for every $\beta \in J(d_{i,j})$. Since $S_{i,j} \supseteq S_{i,0} \subseteq S'$, $\mathrm{Th}(S_{i,j}) \supseteq \mathrm{Th}(S_{i,0})$ and $\mathrm{Th}(S_{i,j}) \models p(d_{i,j})$. Moreover, since the computation of S' satisfies the criterion given in Prop. 1(2), $\mathrm{Th}(S' \cup \{\beta\})$ is consistent for every $\beta \in J(d_{i,j})$, this also holds for $\mathrm{Th}(S_{i,j} \cup \{\beta\})$ since $S_{i,j} \subseteq S'$.
- Since $AD_\eta = AD'_{\eta'}$ and $S = S'$, $\mathrm{Th}(S')$ satisfies the criterion given in Prop. 1(2). ■

Thus, if the criterion given in Prop. 1(2) is tested after the computation, applying one default at a time or applying arbitrary defaults at a time yields the same set of acceptable structures. Moreover, application of only one default leads to less results which have to be rejected by this criterion.

10.1.2 The Horn Case: Positive Programs and Positive Preconditions

In the Horn case, Herbrand structures are considered instead of general theories, hence, there is no explicit knowledge about negative literals. For an applicable rule or default whose applicability depends on a negative atom, it is possible that it is not applicable in a later stage of the computation, thus, the results of the previous section in general do not hold. In this section, we consider *positive* F-Logic programs P and the corresponding default theory Δ_P.

Proposition 16 *Let P be an F-Logic program with a semi-normal default theory Δ. Let $\mathcal{H}'_0, \mathcal{H}'_1, \ldots, \mathcal{H}'_{\eta'}$ be defined as in Proposition 4, except that*

$$\mathcal{H}'_{i+1} = T_P^{\omega}(\mathcal{H}'_i \cup \{c(D_i)\}) \ , \ AD'_{i+1} = AD'_i \cup D_i \quad \text{where } D_i \subseteq GD^+(\mathcal{H}'_i, D) \ .$$

Then, the following set coincide:

- *the set of $\mathcal{H}'$ which are computed by the above sequences and satisfy the criterion given in Prop. 5, and*
- *the set of inflationary H-extensions of Δ which satisfy this criterion.* □

Proof The proof is analogous to Proposition 15. ■

Corollary 4 *For an F-Logic program P with the default theory Δ_P^+, the set of $\mathcal{H}'$ which are computed by the above sequences and are (P2)-free coincides with the set of inflationary H-extensions of Δ_P^+ which are (P2)-free.* □

Proof The corollary is the specialization of Proposition 16 for Δ_P^+. The claim follows then from Proposition 5 and Corollary 1. ■

Thus, also in this case, it is sufficient to apply one ground instance of an applicable default at a time.

10.1.3 Inheritance via Inheritance Triggers

The same coincidence still holds when application of defaults in H-structures is formalized by triggers:

Proposition 17 *For an F-Logic program P, let $\mathcal{J}_P$-sequences be defined like $\mathcal{I}_P$-sequences (cf. Definition 15), except that for all i, there is a $\mathcal{T}_i \subseteq \mathbf{T}(\mathcal{M}_i)$ such that $\mathcal{M}_{i+1} = \mathcal{I}_P^{\mathcal{T}_i}(\mathcal{M}_i)$.*
Then, the set of $\mathcal{I}_P$-sequences which are (P2)-free coincides with the set of $\mathcal{J}_P$-sequences which are (P2)-free. □

Proof Again, the proof is analogous to Proposition 15. ■

Since $\mathcal{I}_P^*$-sequences enforce (P2)-freeness (by augmenting the logic program accordingly), the above proposition yields the following (cf. Theorem 5):

Corollary 5 *For an F-Logic program P, let $\mathcal{J}_P^*$-sequences be defined like $\mathcal{I}_P^*$-sequences (cf. Definition 18), except that for all i, there is a $\mathcal{T}_i \subseteq \mathbf{T}(\mathcal{M}_i)$ such that $\mathcal{M}_{i+1} = \mathcal{I}_{P_{i+1}}^{\mathcal{T}_i}(\mathcal{M}_i)$.*
Then, $\mathcal{S}_{\mathcal{J}}^*(P)$ is the set of cautious inflationary H-extensions of Δ_P and for every $\mathcal{H} \in \mathcal{S}_{\mathcal{J}}^*(P)$, if $\mathcal{I}_P^t(\mathcal{H}) = \perp$ for every $t \in \mathbf{T}(\mathcal{H})$, then $\mathcal{H}$ is an H-extension of Δ_P. □

Thus, for positive F-Logic programs, it is sufficient to consider only strategies which apply one trigger at a time.

For regarding negation in logic programs in combination with default reasoning, one has to define a semantics combining stratified, well-founded, or stable semantics with defaults. In the current system, a user-stratified semantics is implemented. In [MLL97], a well-founded evaluation of F-Logic programs has been presented.

10.2 Classification of H-extensions.

Often the semantics of nonmonotonic frameworks is classified by regarding *several* structures which are regarded as models of a given input F: the *sceptical (conservative)* perspective accepts only facts which are true in *all* models, the *credulous (liberal, brave)* perspective accepts all facts which are true in some model, and the *choice* perspective simply yields (nondeterministically) one of the models (cf. [McD82]).

It is well-known that under most approaches (including circumscription and normal Default Logic), the union of two models is inconsistent, i.e. the credulous semantics yields inconsistent structures. This also holds for logic programming with inheritance (cf. Example 1). In case of logic programming without negation and with inheritance, the sceptical perspective always yields a model (this is not the case when negation is allowed). The choice perspective always yields a model, but is nondeterministic.

In the above approach, $\mathcal{I}_P$- and $\mathcal{I}_P^*$-sequences are representatives of the choice strategy. By exploring the state space of $\mathcal{I}_P$-sequences systematically, i.e. computing $\mathcal{S}_{\mathcal{I}}(P)$, sceptical and credulous semantics can also be implemented.

Instead of comparing all elements of $\mathcal{S}_{\mathcal{I}}(P)$, a *localized* sceptical or credulous strategy can be implemented by comparing possible inheritance steps:

Definition 19 (Credulous and Skeptical Firing)

- A trigger $t \in \mathbf{T}(\mathcal{H})$ is *credulously* applicable in $\mathcal{H}$ if the subsequent deductive fixpoint $T_P^\omega(t(\mathcal{H}))$ is consistent (i.e., no scalar method is assigned two different values).
- A trigger $t \in \mathbf{T}(\mathcal{H})$ is *skeptically* applicable in $\mathcal{H}$ if the subsequent deductive fixpoint $T_P^\omega(t(\mathcal{H}))$ is consistent and there is no trigger which is active in $\mathcal{H}$ and blocked in $T_P^\omega(t(\mathcal{H}))$.

For comparing transitions and computations, the degree of credulity, skepticism, abnormality, and undefinedness can be measured as the number of blocked triggers or non-fired active triggers: for an H-structure $\mathcal{H}$ and an $\mathcal{I}_P$-sequence $\mathcal{H}_0, \mathcal{H}_1, \ldots$,

$$
\begin{aligned}
cr(\mathcal{H}_{n+1}) &:= cr(\mathcal{H}_n) + |\{t \mid t \in \mathbf{T}(\mathcal{H}_n),\ t \text{ is blocked in } \mathcal{H}_{n+1}\}|\ ,\\
sc(\mathcal{H}_{n+1}) &:= sc(\mathcal{H}_n) + |\{t \mid t \in \mathbf{T}(\mathcal{H}_n),\ t \text{ is still active in } \mathcal{H}_{n+1}\}|\ ,
\end{aligned}
$$

Additionally, the number of "abnormal" objects wrt. inheritance, i.e. objects whose properties differ from those of a typical object of their class can be used for comparison:

$$
\begin{aligned}
ab(\mathcal{H}) &:= |\{(o, m, v') \mid o[m \rightarrow v'] \in \mathcal{H} \text{ and there is a trigger}\\
&\qquad\qquad (o \text{ isa } c, m \bullet\!\!\rightarrow v) \text{ in } \mathcal{H} \text{ s.t. } v' \neq v\}|\ ,\\
un(\mathcal{H}) &:= |\{(o, m, v) \mid \text{there is a } c \text{ s.t. } (o \text{ isa } c, m \bullet\!\!\rightarrow v) \in \mathbf{T}(\mathcal{H})\}|\ . \qquad \square
\end{aligned}
$$

Intuitively, this means that for everything that is not explicitly known to be abnormal, the default value holds. Note that there actually are $\mathcal{H} \in \mathcal{S}_{\mathcal{I}}$ with $un(\mathcal{H}) > 0$ (cf. Example 5).

If a trigger is postempted by introducing an intermediate class with a different value of an inheritable method (cf. Example 12), the intermediate class comes up with a blocked trigger for every member object. Thus, postemption is in general expensive wrt. credulity and abnormality.

10.3 Implementation

In the current FLORID implementation, indirect conflicts due to multiple assignment of scalar methods are handled differently: By equating two objects if they are results of the same scalar method application to an object, there is no notion of inconsistency. The semantics is defined to be the set of states which are reachable this way where no more inheritance triggers are active. Additionally, a strategy is implemented in an internal version of FLORID where *all* active triggers are fired.

As long as no object creation takes place, every T_P^ω computation is polynomial. Since the number of potential triggers is also polynomial, an H-extension $\mathcal{H} \in \mathcal{S}_{\mathcal{I}}(P)$ (or $\mathcal{S}_{\mathcal{I}^*}(P)$) can be computed in polynomial time. With object creation, the computations can become infinite.

By implementing defaults $a{:}b/c$ as general triggers which are active in an H-structure $\mathcal{H}$ if $\mathcal{H} \models a$ and $T_P^\omega(\mathcal{H} \cup b) \neq \perp$, the approach can be extended to defaults where the justifications are conjunctions of atoms.

11 Examples

The examples in this section show that the class hierarchy and the handling of class information must be carefully designed to cover the intended meaning. With an appropriate design, various application semantics can be encoded into the behavior of inheritance. The last example shows how the inheritance mechanism can be “misused” to implement a state sequence with a built-in frame semantics.

11.1 Disjunctive Information

Consider again the Nixon Diamond (cf. Examples 1 and 3) and the Nixon Family (cf. Ex. 5). Here, the fact that Nixon is a quaker and a republican represents disjunctive information. In the diamond, both policies can be inherited. In the Nixon Family, inheriting policy $\rightarrow$ hawk turns out to be inconsistent, thus, policy $\rightarrow$ pacifist is inherited.

As an abstraction of this case, a class *Republican Quaker* can be introduced.

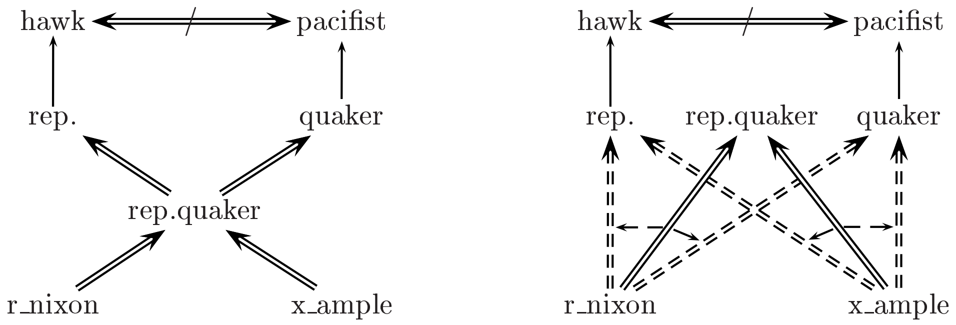


Figure 4: Inheritance Networks for Republican Quakers

- In the first version, *Republican Quaker* is introduced as a subclass of *Republican* and *Quaker* (cf. on the left of Figure 4, similar to the network given in Fig. 2):

  $P = \{$quaker[policy•→pacifist], republican[policy•→hawk],
  rep_quaker :: republican, rep_quaker :: quaker,
  r_nixon isa rep_quaker, x_ample isa rep_quaker$\}$.

  Here, $T_P^\omega(\emptyset) = P$, the method rep_quaker[policy•→_] is still undefined. In the first inheritance step, either the value rep_quaker[policy•→hawk] or rep_quaker[policy•→pacifist] is inherited – defining the *inheritable* policy for *all* republican quakers, solving this conflict on class level – which is clearly not the intended semantics: Then, both r_nixon and x_ample must inherit the same policy.

  Moreover, if rep_quaker[policy•→hawk] is inherited and r_nixon[policy→hawk] would be inconsistent as in the Nixon Family, r_nixon cannot inherit policy→pacifist since inheritance from quaker to r_nixon is preempted by rep_quaker.
- The second version introduces *Republican Quaker* as the conjunction of being a *Republican* and a *Quaker*, but *not* as a subclass of them (cf. on the right of Figure 4; this way of modeling disjunctive information has first been published in [Kan97]):

$P = \{\mathsf{quaker[policy} \bullet\!\!\to \mathsf{pacifist]},\ \mathsf{republican[policy} \bullet\!\!\to \mathsf{hawk]},$
$\quad \mathsf{O\ isa\ quaker} \leftarrow \mathsf{O\ isa\ rep\_quaker},$
$\quad \mathsf{O\ isa\ republican} \leftarrow \mathsf{O\ isa\ rep\_quaker},$
$\quad \mathsf{r\_nixon\ isa\ rep\_quaker},\ \mathsf{x\_ample\ isa\ rep\_quaker}\}.$

Here, $T_P^\omega(\emptyset) = P \cup \{\mathsf{r\_nixon\ isa\ quaker}, \mathsf{r\_nixon\ isa\ republican}, \mathsf{x\_ample\ isa\ quaker}, \mathsf{x\_ample\ isa\ republican}\}$.

Thus, in the inheritance step, both policies can be inherited individually by r_nixon and x_ample.

11.2 Conflict Detection on the Class Level

Consider the following example, taken from [Hor94] for illustrating *mixed preemption*; the inheritance net is depicted in Fig. 5.

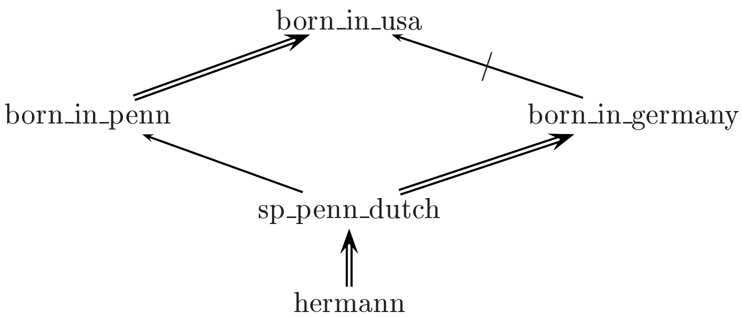


Figure 5: Mixed Preemption

Here, *Hermann* is a native speaker of *Pennsylvania Dutch*, thus, also a native speaker of *German* (g). Typically, native speakers of *Pennsylvania Dutch* (pd) are born in *Pennsylvania* (p), thus, born in the *USA* (u). On the other hand, native speakers of *German* are typically *not* born in the *USA*. The concept of *mixed preemption* interprets the combination of the defeasible link pd–p and the strict link p–u as a unit, thus, being more specific than pd–g and $g \not\to u$. Consequently, *Hermann* is believed to be born in the *USA*.

In the F-Logic equivalent the class hierarchy must be accordingly designed:

$P = \{\mathsf{hermann\ isa\ native\_sp\_penn\_dutch},$
$\quad \mathsf{native\_sp\_penn\_dutch :: native\_sp\_german},$
$\quad \mathsf{native\_sp\_penn\_dutch[born\_state} \bullet\!\!\to \mathsf{pennsylvania]},$
$\quad \mathsf{native\_sp\_german[born\_country} \bullet\!\!\to \mathsf{germany]},$
$\quad \mathsf{O[born\_country} \to \mathsf{usa]} \leftarrow \mathsf{O[born\_state} \to \mathsf{pennsylvania]}\}.$

Here, $T_P^\omega(\emptyset) = P \cup \{\mathsf{hermann\ isa\ native\_sp\_german}\}$. Active triggers are
$(\mathsf{hermann\ isa\ native\_sp\_penn\_dutch},\ \mathsf{born\_state} \bullet\!\!\to \mathsf{pennsylvania})$ and
$(\mathsf{native\_sp\_penn\_dutch :: native\_sp\_german},\ \mathsf{born\_country} \bullet\!\!\to \mathsf{german})$.

Thus, inheritance could result in native_sp_penn_dutch[born_country•→germany], which is obviously wrong – here, the conflict (which preempts the path *Hermann – Pennsylvania Dutch – German – Germany*) is semantically located on the *class level*.

In P it has been forgotten to define native_sp_penn_dutch[born_country•→_], either by a fact, or by a rule. The clean alternative is, to *lift* the above rule to the class level:

$$P' := P \cup \{\mathsf{C[born\_country}{\bullet\!\!\to}\mathsf{usa]} \leftarrow \mathsf{C[born\_state}{\bullet\!\!\to}\mathsf{pennsylvania]}\}.$$

Now, $T^{\omega}_{P'}(\emptyset) = P' \cup \{\mathsf{native\_sp\_penn\_dutch[born\_country}{\bullet\!\!\to}\mathsf{usa]}\}$ and the active triggers are

(hermann isa native_sp_penn_dutch, born_state•→pennsylvania) and
(hermann isa native_sp_penn_dutch, born_country•→usa) .

After inheritance (one or two steps, depending on the chosen trigger and strategy), the intended model containing hermann[born_country•→usa] is obtained.

11.3 Generalizations as Classes

Another problem occurs, when clichés (and chains of clichés) are used: in this case, there is no strict inclusion between the classes of objects *actually* satisfying these properties, but only a strict inclusion of objects belonging to classes which *are believed* to satisfy these properties. Here, also a classic example is given in [Hor94][9]. The network is given in Figure 6.

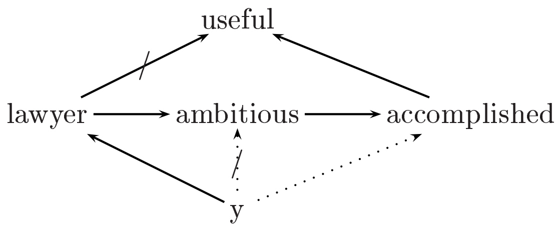


Figure 6: Chains of clichés

Here, y is a *lawyer*. *Lawyers* are supposed to be both *ambitious* and *socially useless*. People who are *ambitious* tend to be *accomplished*. Most *accomplished* people are *socially useful*. Here, obviously, this is not a class hierarchy (then, it would simply be inconsistent): e.g., *ambitious* people are not necessary *accomplished*.

Note that this network mainly consists of a chain of defeasible links, representing clichés. As stated in Section 2.2, this chain cannot be mapped immediately to a network as required in the database setting.

[9] there, with a footnote that "this node labelling is adapted from an earlier example of Ginsberg's (personal communication), which displayed a similar attitude toward lawyers".

On the other hand, the clichés apply stepwise on an abstract level: the class of lawyers is a subclass of the people who are supposed to be ambitious (due to their profession) – most of them actually are ambitious. People who are supposed to be ambitious are a subclass of people who are potentially accomplished – again, most of these actually are accomplished. Most accomplished people are socially useful; as a hypothesis this can be assumed also for people who are assumed to be accomplished.

Only the dotted lines in the net, i.e, that y is not ambitious but accomplished represent hard facts about y.

In the net, *without* the dashed lines, the path y – *lawyer* – ... – *useful* is preempted by the path *lawyer* $\not\to$ *useful*. With the dashed lines, the path y – *accomplished* – *useful* is not preempted, thus it is simply in conflict with y – *lawyer* $\not\to$ *useful*.

In the F-Logic representation, the focus is on the classes of people who are supposed to satisfy some property:

$P = \{$y isa lawyer, lawyer[useful•→no]
lawyer :: supp_amb , supp_amb[ambitious•→yes] ,
supp_amb :: supp_acc , supp_acc[accomplished•→yes] ,
supp_acc[useful•→yes]$\}$.

With this class hierarchy, the following facts are inherited:

supp_amb[accomplished•→yes; useful•→yes] ,
lawyer[accomplished•→yes; ambitious•→yes] ,
y[useful→no; accomplished→yes; ambitious→yes] .

For $P' = P \cup \{$y[accomplished→yes; ambitious→no]$\}$, the following facts are inherited:

supp_amb[accomplished•→yes; useful•→yes],
lawyer[accomplished•→yes; ambitious•→yes],
y[useful→no] ,

which is still not the intended result!

As in the previous example, the modeling is insufficient: there is only a class of people who are *supposed* to be accomplished. Since y is a member of *this* class only in virtue of being a lawyer, inheritance is still preempted. The modeling simply neglects the fact that there is another class of people who *actually* are accomplished, and that most of those people actually are socially useful:

For

$P'' = P' \cup \{$O isa actually_acc ← O[accomplished→true] ,
actually_acc[useful•→yes]$\}$,

$T^{\omega}_{P''}(\emptyset) = P \cup \{$y isa actually_acc$\}$ and the triggers

(y isa lawyer, useful•→no) and
(y isa actually_acc, useful•→yes) .

are both active. Thus, both results are possible.

11.4 Playing with Preemption and Unsupported Conclusions

D_{inh} gives a higher priority to preemption than to refinement. This conflict can theoretically occur when an inheritance default D_{inh} is applied on a path which later – enabled only after application of this default – is refined by introducing alternative intermediate subclasses such that a preempted and a non-preempted refined path emerge. We decided not to accept the refinement in this case.

We present two slightly different versions of such (theoretical) scenarios for arguing that our "restrictive" strategy is reasonable. Both are versions of a diamond:

$$\begin{tikzcd}
& \mathrm{c}[\mathrm{m}{\bullet\!\!\rightarrow}\mathrm{v}] & \\
\mathrm{c'}[\mathrm{m}{\bullet\!\!\rightarrow}\mathrm{w}] \arrow[ur] & & \mathrm{c''}[\mathrm{m}{\bullet\!\!\rightarrow}\mathrm{v}] \arrow[ul] \\
& \mathrm{o} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Example 14 (Preemption or no Preemption) Consider the following program P where a conclusion by inheritance (a) preempts itself and (b) bypasses this preemption by itself:

$$\begin{aligned} P = \{\ & \mathsf{c[m{\bullet\!\!\rightarrow}v]}\ ,\ \mathsf{c' :: c}\ ,\ \mathsf{c'[m{\bullet\!\!\rightarrow}w]}\ ,\ \mathsf{c'' :: c}\ ,\ \mathsf{o\ isa\ c}\ , \\ & \mathsf{o\ isa\ c''} \leftarrow \mathsf{o[m{\rightarrow}v]}\ ,\ \ \mathsf{o\ isa\ c'} \leftarrow \mathsf{o[m{\rightarrow}v]}\ \}\ . \end{aligned}$$

Here, $$\begin{aligned} S_0 &= P\ ,\ \text{and} \\ \mathsf{facts}(\mathrm{Th}(S_0)) &= \{\mathsf{c[m{\bullet\!\!\rightarrow}v]}\ ,\ \mathsf{c' :: c}\ ,\ \ \mathsf{c'[m{\bullet\!\!\rightarrow}w]}\ ,\ \mathsf{c'' :: c}\ ,\ \mathsf{o\ isa\ c}\}\ , \end{aligned}$$

thus, inheritance of $\mathsf{c[m{\bullet\!\!\rightarrow}v]}$ along $\mathsf{o - c}$ to o and along $\mathsf{c'' - c}$ to $\mathsf{c''}$ seems reasonable. But, then,

$$\begin{aligned} S_1 &= P \cup \{\mathsf{c''[m{\bullet\!\!\rightarrow}v]}\ ,\ \mathsf{o[m{\rightarrow}v]}\}\ , \\ \mathsf{facts}(\mathrm{Th}(S_1)) &= \mathsf{facts}(\mathrm{Th}(S_0)) \cup \{\mathsf{c''[m{\bullet\!\!\rightarrow}v]}\ ,\ \mathsf{o[m{\rightarrow}v]}\} \cup \{\mathsf{o\ isa\ c'}\ ,\ \mathsf{o\ isa\ c''}\}\ . \end{aligned}$$

There, (a) the path $\mathsf{o - c}$ is preempted by $\mathsf{c'[m{\bullet\!\!\rightarrow}w]}$, which is an argument against inheritance. On the other hand, this preemption is bypassed by the path $\mathsf{o - c''[m{\bullet\!\!\rightarrow}v] - v}$, thus, in S_1 there *is* a path from c to o which is *not* preempted. But, this path exists only *when* inheritance is already *assumed* to be possible.

An argument *for* accepting $\mathrm{Th}(S_1)$ as an extension would be to say that $\mathsf{o - c''[m{\bullet\!\!\rightarrow}v] - v}$ is a *refinement* of the preempted path o - c. But, *we* argue, that $\mathrm{Th}(S_1)$ is not a valid extension, this decision is additionally motivated by the next example which is a slight variation. □

Example 15 (Unsupported Conclusions) Consider the following program P where a conclusion by inheritance exists only due to itself:

$$P = \{\ \mathsf{c[m{\bullet\!\!\rightarrow}v]}\ ,\ \mathsf{c' :: c}\ ,\ \mathsf{c'[m{\bullet\!\!\rightarrow}w]}\ ,\ \mathsf{c'' :: c}\ ,\ \mathsf{o\ isa\ c'}\ ,\ \mathsf{o\ isa\ c''} \leftarrow \mathsf{o[m{\rightarrow}v]}\}\ .$$

Here, one could argue that

$$\Phi = \mathrm{Th}(P \cup \{\mathsf{c"[m{\bullet\!\!\to}v]}\ ,\ \mathsf{o\ isa\ c}\ ,\ \mathsf{o\ isa\ c'}\ ,\ \mathsf{o\ isa\ c"}\ ,\ \mathsf{o[m{\to}v]}\})$$

is an extension of Δ_P:

$$\begin{aligned} S_0 &= P\ , \\ \mathsf{facts}(\mathrm{Th}(S_0)) &= \{\mathsf{c[m{\bullet\!\!\to}v]}\ ,\ \mathsf{c'::c}\ ,\ \mathsf{c'[m{\bullet\!\!\to}w]}\ ,\ \mathsf{c"::c}\ ,\ \mathsf{o\ isa\ c}\ ,\ \mathsf{o\ isa\ c'}\}\ . \end{aligned}$$

With the same argumentation as in the previous example, accepting a path which only exists in the resulting theory, inheritance of $\mathsf{o[m{\to}v]}$ along o − $\mathsf{c"}$ − c is not preempted, (obviously, $\mathsf{c[m{\bullet\!\!\to}v]}$ can be inherited along $\mathsf{c"}$ − c to $\mathsf{c"}$), thus,

$$\begin{aligned} S_1 &= S_0 \cup \{\mathsf{c"[m{\bullet\!\!\to}v]}\ ,\ \mathsf{o[m{\to}v]}\}\ , \\ \mathsf{facts}(\mathrm{Th}(S_1)) &= \mathsf{facts}(\mathrm{Th}(S_0)) \cup \{\mathsf{c"[m{\bullet\!\!\to}v]}\ ,\ \mathsf{o[m{\to}v]}\} \cup \{\mathsf{o\ isa\ c"}\}\ . \end{aligned}$$

Here, accepting $\Phi = \mathrm{Th}(S)$ as an extension is very unintuitive. □

11.5 Application: Modeling Dynamics with Inheritance Triggers

The trigger mechanism can be used to insert atoms into the database after a deductive fixpoint has been reached by specifying a suitable class hierarchy. With this, a state-by-state evaluation of a logic program can be enforced, defining the state sequence via a sequence of deductive fixpoint computations.

Inheritance can be used to implement a solution of the frame problem in an elegant way: Every state is made a class, and each of its immediate successors (for linear time: its unique successor) is a subclass of it. By controlling the evolution of the class hierarchy, it is possible to compute the changes performed in a transition, then making the successor state a subclass of the current state (as a class) and inheriting the frame knowledge in a single step without explicit deduction.

To provide the frame semantics, every method application which is defined in some state has to be determined in the subsequent state either by the action performed in a transition and possible ramification effects, or it is inherited in the subsequent inheritance step.

Example 16 This example makes use of path expressions and object creation in F-Logic: if m and o are id-terms, the *path expression* $o.m$, denotes the object resulting from applying m to o (if this object does not already exist, it is created when some object atom $(o.m)[\ldots]$ is defined).

Consider the following classical example for ramification: When pulling the plug from a filled tub, not only the plug is pulled in the next state, but additionally, the tub runs empty.

We start with a set of facts describing state 1 with a filled tub:

$$\mathcal{H}_0 = \{\mathsf{x\ isa\ tub},\ \mathsf{(x.1)[filled{\bullet\!\!\to}true]},\ \mathsf{(x.1)[plug{\bullet\!\!\to}in]},\ \mathsf{(state.1)[active{\bullet\!\!\to}true]}\}\ .$$

The state sequence is formalized by

$(\mathsf{O.S{+}1})\ ::\ \mathsf{O.S} \leftarrow (\mathsf{state.S})[\mathsf{active} \bullet\!\!\to \mathsf{true}]$.

The immediate effect of pulling the plug is axiomatized by

$(\mathsf{O.S{+}1})[\mathsf{plug} \bullet\!\!\to \mathsf{out}] \leftarrow (\mathsf{O.S})[\mathsf{pull\_plug} \to \mathsf{true}],\ \mathsf{O\ isa\ tub}$.

The effect of a pulled plug on a tub is axiomatized by a local rule (ramification):

$(\mathsf{O.S})[\mathsf{filled} \bullet\!\!\to \mathsf{false}] \leftarrow (\mathsf{O.S})[\mathsf{plug} \bullet\!\!\to \mathsf{out}]$.

Additionally we assume that in state 2, the plug gets pulled:

$(\mathsf{x.2})[\mathsf{pull\_plug} \to \mathsf{true}] \leftarrow (\mathsf{state.2})[\mathsf{active} \bullet\!\!\to \mathsf{true}]$.

The first application of T_P^ω computes the first state

$$\mathcal{H}_1 = \{\mathsf{x\ isa\ tub},\ (\mathsf{state.1})[\mathsf{active} \bullet\!\!\to \mathsf{true}],\ (\mathsf{x.1})[\mathsf{filled} \bullet\!\!\to \mathsf{true}],\ (\mathsf{x.1})[\mathsf{plug} \bullet\!\!\to \mathsf{in}],\\ (\mathsf{state.2}) :: (\mathsf{state.1}),\ (\mathsf{x.2})\ ::\ (\mathsf{x.1})\}\ .$$

the other methods on x are undefined. The subsequent inheritance step fires all active triggers, leading to

$$\mathcal{H}_2' = \mathcal{H}_1 \cup \{(\mathsf{state.2})[\mathsf{active} \bullet\!\!\to \mathsf{true}],\ (\mathsf{x.2})[\mathsf{filled} \bullet\!\!\to \mathsf{true}],\ (\mathsf{x.2})[\mathsf{plug} \bullet\!\!\to \mathsf{in}]\}\ .$$

The next application of T_P^ω derives

$$\mathcal{H}_2 = \mathcal{H}_2' \cup \{(\mathsf{x.2})[\mathsf{pull\_plug} \to \mathsf{true}],\ (\mathsf{x.3})[\mathsf{plug} \bullet\!\!\to \mathsf{out}],\\ (\mathsf{state.3}) :: (\mathsf{state.2}),\ (\mathsf{x.3}) :: (\mathsf{x.2})\}\ .$$

Here, $(\mathsf{x.3})[\mathsf{plug} \bullet\!\!\to \mathsf{out}]$ blocks inheritance of $(\mathsf{x.3})[\mathsf{plug} \bullet\!\!\to \mathsf{in}]$ from x.2. By the ramification rule, it is clear that in the next T_P step, $(\mathsf{x.3})[\mathsf{filled} \bullet\!\!\to \mathsf{false}]$ is derived. Thus, for obtaining a consistent state, $(\mathsf{x.2})[\mathsf{filled} \bullet\!\!\to \mathsf{true}]$ must not be inherited to x.3. With this, the only credulous inheritance step leads to

$$\mathcal{H}_3' = \mathcal{H}_2 \cup \{(\mathsf{state.3})[\mathsf{active} \bullet\!\!\to \mathsf{true}]\}\ ,$$

and with the subsequent T_P^ω step:

$$\mathcal{H}_3 = \mathcal{H}_3' \cup \{(\mathsf{x.3})[\mathsf{filled} \bullet\!\!\to \mathsf{false}],\ (\mathsf{state.3}) :: (\mathsf{state.2}),\ (\mathsf{x.3}) :: (\mathsf{x.2})\}\ . \ \Box$$

In [MSL97], it has been shown how the *one-trigger-at-a-time* strategy can be used to provide an effective stratification in an operational way. There, the method has been used for modeling and implementing dynamics in deductive object-oriented databases.

12 Conclusion

We have shown how inheritance can be integrated into a deductive object-oriented database language. By considering the Horn fragment (i.e., logic programming rules) and restricting the use of defaults to the object-oriented notion of inheritance, we could tailor the semantics to the requirements in this area. Given a program P, the presented algorithm computes those Herbrand-like structures which represent the extensions of the default theories corresponding to P in a forward-chaining way.

Regarding the F-Logic project, we have shown that the semi-declarative semantics which is defined for F-Logic and implemented in the FLORID system [FHK+97] coincides with the standard semantics of Default Logic and Inheritance Networks.

Acknowledgements. We want to thank the current and former F-Logic team members at *Freiburg University*, Georg Lausen, Jürgen Frohn, Rainer Himmeröder, Bertram Ludäscher, Christian Schlepphorst, and Heinz Uphoff for fruitful discussions.

References

[AK92] S. Abiteboul and P. C. Kanellakis. Object Identity as a Query Language Primitive. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building an Object-Oriented Database System – The Story of O_2*, chapter 5, pp. 98–127. Morgan Kaufmann, 1992.

[BF91] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78:85–112, 1991.

[Bre87] G. Brewka. The Logic of Inheritance in Frame Systems. In *Intl. Joint Conference on Artificial Intelligence*, pp. 483–488, 1987.

[Bre91] G. Brewka. *Nonmonotonic Reasoning: Logical Foundations of Commonsense.* Cambridge University Press, 1991.

[CCCR+90] F. Cacace, S. Ceri, S. Crespi-Reghizzi, L. Tanca, and R. Zicari. Integrating Object-Oriented Data Modeling with a Rule-Based Programming Paradigm. In H. Garcia-Molina and H. V. Jagadish, editors, *ACM Intl. Conference on Management of Data (SIGMOD)*, pp. 225–236, 1990.

[Dix95] J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information.* de Gruyter, 1995.

[DT95] G. Dobbie and R. Topor. On the Declarative and Procedural Semantics of Deductive Object-Oriented Systems. *Journal of Intelligent Information Systems*, 4(2):193–219, 1995.

[FHK+97] J. Frohn, R. Himmeröder, P.-T. Kandzia, G. Lausen, and C. Schlepphorst. FLORID: A Prototype for F–Logic. In *Intl. Conference on Data Engineering (ICDE)*, 1997.

[FLO98] Florid Homepage. `http://www.informatik.uni-freiburg.de/~dbis/florid/`, 1998.

[GHR94] D. M. Gabbay, C. J. Hogger, and J. A. Robinson. *Handbook of Logic in Artificial Intelligence and Logic Programming.* Oxford Science Publications, 1994.

[GL88] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *Intl. Conference on Logic Programming (ICLP)*, pp. 1070–1080, 1988.

[GL90] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. In *Intl. Conference on Logic Programming (ICLP)*, pp. 579–597. MIT Press, 1990.

[Hor94] J. F. Horty. Some direct Theories of Nonmonotonic Inheritance. In *Handbook of Logic in Artificial Intelligence and Logic Programming* [GHR94].

[Kan97] P.-T. Kandzia. Nonmonotonic Reasoning in FLORID. In *Intl. Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, number 1265 in LNAI. Springer, 1997.

[KLM90] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44:167–207, 1990.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[KS97] P.-T. Kandzia and C. Schlepphorst. DOOD and DL – Do we Need an Integration? In *Workshop Knowledge Representation Meets Databases (KRDB)*, 1997.

[Lif94] V. Lifschitz. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming* [GHR94].

[Liu96] M. Liu. ROL: A Deductive Object Base Language. *Information Systems*, 21(5):431–457, 1996.

[Mak94] D. Makinson. General Patterns in Nonmonotonic Reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming* [GHR94].

[McD82] D. McDermott. Non-Monotonic Logic II. *Journal of the ACM*, 29:33–57, 1982.

[MK98] W. May and P.-T. Kandzia. Nonmonotonic Inheritance in Object-Oriented Deductive Database Languages. In U. Egly and H. Tompits, editors, *Workshop logische Programmierung - WLP'98*, pp. 173–186. Technical Report 1843-1998-10, Institut für Informationssysteme, Technical University Vienna, 1998.

[MLL97] W. May, B. Ludäscher, and G. Lausen. Well-Founded Semantics for Deductive Object-Oriented Database Languages. In F. Bry, K. Ramamohanarao, and R. Ramakrishnan, editors, *Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, number 1341 in LNCS, pp. 320–336, Montreux, Switzerland, 1997. Springer.

[Mor98] L. Morgenstern. Inheritance Comes of Age: Applying Nonmonotonic Techniques to Problems in Industry. *Artificial Intelligence*, 103:1–34, 1998.

[MSL97] W. May, C. Schlepphorst, and G. Lausen. Integrating Dynamic Aspects into Deductive Object-Oriented Databases. In A. Geppert and M. Berndtsson, editors, *3rd Intl. Workshop on Rules in Database Systems (RIDS)*, number 1312 in LNCS, Skövde, Sweden, 1997.

[MT93] V. W. Marek and M. Truszczyński. *Nonmonotonic Logic*. Springer, 1993.

[PGA87] D. Poole, R. Goebel, and R. Aleliunas. Theorist: A Logical Reasoning System for Defaults and Diagnosis. In *The Knowledge Frontier: Essays in the Representation of Knowledge*, pp. 331–352. Springer, 1987.

[Poo94] D. Poole. Default Logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming* [GHR94].

[Prz91] T. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, (9):401–424, 1991.

[Rei80] R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 12(1,2):81–132, 1980.

[Sho88] Y. Shoham. *Reasoning about Change*. MIT Press, 1988.

[Tou86] D. Touretzky. *The Mathematics of Inheritance*. Morgan-Kaufmann, Los Altos, CA, 1986.

[VGRS88] A. Van Gelder, K. Ross, and J. Schlipf. Unfounded Sets and Well-Founded Semantics for General Logic Programs. In *ACM Symposium on Principles of Database Systems (PODS)*, pp. 221–230, 1988.