The Case for Explicit Exceptions.

L. Thorne McCarty
Rutgers, The State University of New Jersey
67 PUBLICATIONS **1,479** CITATIONS

The Case for Explicit Exceptions

L. Thorne McCarty
William W. Cohen[*]
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

November 1990
Revised: January 1992

This paper will appear in a special issue of *Methods of Logic in Computer Science*. It was first presented at a Workshop on *Logic Programming and NonMonotonic Reasoning* held in Austin, Texas, on November 1–2, 1990.

[*]Current address: AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

Abstract

Most of the work on inheritance hierarchies in recent years has had as its goal the design of general purpose algorithms that depend only on the topology of the inheritance network. This research has produced some important observations about the various strategies used in human common sense reasoning, but it has also produced a proliferation of incompatible systems. In this paper, we resurrect the alternative technique, originally proposed by Etherington and Reiter, of explicitly encoding exceptions to default rules. The main technical innovation is the use of a different logical framework: a logic programming language based on *intuitionistic logic.* Using a combination of full intuitionistic negation plus negation-as-failure to encode default rules, we obtain analogues of the normal, seminormal and nonnormal defaults of Reiter's default logic. The advantage of our approach is that, whereas there is no adequate proof theory in classical logic for seminormal defaults, the analogous queries to an intuitionistic default rulebase can be answered by a simple top-down goal-directed interpreter. The claim that a default rulebase with explicit exceptions is easy to write and debug has been substantiated by encoding more than 40 standard examples from the literature.

1 Introduction

Most of the work on inheritance hierarchies in recent years has tried to develop a *syntactic* characterization of default inheritance. The main goal has been to design general algorithms that depend only on the topology of the inheritance network, and yet correctly capture our intuitions about common sense reasoning. This research has produced some important observations about the various strategies used in human common sense reasoning [41, 22, 33, 43], but it has also produced a proliferation of incompatible systems. The design space is surveyed by Touretzky, et al., in [47]. Inheritance algorithms may be "skeptical" or "credulous", may use "upward" or "downward" reasoning, may adopt "on-path" or "off-path" preemption, and more. To sort through the possibilities, other researchers have proposed *semantic* characterizations of inheritance, based on conditional logics [9], for example, or probabilistic logics [18]. But these, too, have proliferated.

Faced with this diversity, and faced with the practical task of building a system, what should one do? There is an earlier approach to this problem, dating from the original paper by Etherington and Reiter [13], in which exceptions were explicitly encoded in the form of default rules and the inferences were those sanctioned by default logic [38]. However, in order to state correctly the relationship between interacting defaults, it seemed necessary, in general, to use *seminormal* in addition to purely *normal* default rules. The problem was that there existed no adequate proof theory for seminormal defaults. The only known algorithm, due to Etherington [11], and applicable only to *ordered* default theories, computed extensions bottom-up and could not be used, in practice, for top-down goal-directed reasoning. (See [24] for a recent analysis of the relative complexity of these two tasks.) Thus, when Etherington published a further paper on inheritance hierarchies [12], he abandoned the use of seminormal defaults altogether, writing out the inheritance rules now as a normal default theory. And since normal defaults admitted too many unintended extensions, Etherington proposed to use Touretzky's "inferential distance" algorithm [46] to further prune the possibilities. In other words, the exceptions were now encoded *implicitly* in the logic.

In this paper, we will resurrect the technique of explicit exceptions originally proposed in [13], but within a different logical framework. The main technical innovation in our work is a logic programming language based on *intuitionistic logic*. This language is described in detail in [28, 29], and briefly summarized in Section 2 below. Intuitionistic logic programming, unlike classical logic programming, allows negated expressions to appear in both the premise and the conclusion of a rule, and yet also accommodates negation-as-failure. We use a combination of these two kinds of negation, full intuitionistic negation plus negation-as-failure, to encode default rules. In this way we obtain analogues of the normal, seminormal and general nonnormal defaults of Reiter's default logic. We also obtain a very simple top-down goal-directed interpreter for default rules which, like other interpreters for logic programming languages, returns definite answer substitutions in response to existential queries. To show how simple this interpreter really is, we reproduce it in Figure 6 in Section 5.

Our proposed methodology for developing a rulebase is to start with a set of normal defaults, admitting numerous extensions; then run various anticipated queries through our interpreter; then progressively modify the rulebase in order to block the unintended inferences. We claim that a default rulebase is easy to write, easy to test, and easy to debug using this methodology. To substantiate this claim, we have encoded and tested more than 40 standard examples from the literature. Several of these are reproduced in Section 5.

Our approach is similar in some ways to Poole's approach [36], but the use of intuitionistic logic allows us to make subtle distinctions that are difficult to express within Poole's framework. For example, Poole has great difficulty blocking contrapositive inferences. In our framework, this is simple: If we want a contrapositive inference to go through, we use one syntactic form for the default rule, and if we want to block the contrapositive inference, we use another syntactic form. In general, Poole's encodings depend on a complex assemblage of “defaults”, “facts” and “constraints”, which are difficult to write and debug. In our encodings, we use a single type of rule, with a straightforward logical syntax, and we achieve different results by varying the components both within the rule and within other related rules. This increased modularity has the usual advantages, we claim, for software engineering.

There is much additional related work. In using a circumscriptive theorem-prover and an encoding of default rules by means of “abnormality” predicates, Ginsberg's approach [20] is similar to Poole's, and is therefore related to our approach as well. The use of intuitionistic logic for default reasoning has been advocated by Gabbay [15, 8], but Gabbay's definition of a valid nonmonotonic inference is very different from the definition adopted in our work, and it leads to peculiar results in the case of a normal default without prerequisites. In the database field, Borgida [6] has proposed the use of class hierarchies with explicit exceptions, for some of the same reasons suggested here, and has developed an interesting semantics for such hierarchies. The general literature on default reasoning is much too vast to be reviewed in this paper, but the reader will find useful surveys in [39, 19].

The present paper emphasizes (i) the intuitionistic proof theory for default rules, (ii) our PROLOG interpreter, and (iii) our methodology for developing a default rulebase. We also develop a semantic characterization of default inference using the entailment relation of intuitionistic logic, and we prove a soundness and completeness theorem. A more detailed discussion of the semantic theory is presented in a companion paper [31]. The reader should also consult [31] for a more thorough discussion of related work.

The paper is organized as follows: Section 2 is a brief summary of intuitionistic logic programming, as presented in [28, 29]. Section 3 introduces our notion of a default rule, and explains the intended semantics by means of a simple example. Section 4 presents the intuitionistic default proof theory, and Section 5 describes and illustrates our PROLOG interpreter. The soundness and completeness proof is relegated to Appendix A. Throughout the paper, we try to show, by example, how easy it is to write default rules with explicit exceptions in our language. We conclude by describing our experience encoding many of the benchmark problems in the literature.

2 Clausal Intuitionistic Logic

We begin by presenting a summary of the *monotonic* version of our logic programming language, which we call *clausal intuitionistic logic*. See [28, 29, 5] for a full discussion, and see [16, 17, 32, 21] for closely related work.

Clausal intuitionistic logic is a conservative extension of Horn-clause logic, in which implications may appear "embedded" on the right-hand side of a rule. For example, the following is a legal expression in our language:

$$\begin{aligned}\text{NearGraduate}(s, d) \;\Leftarrow\;& \text{Student}(s) \wedge \text{Department}(d) \wedge \text{Course}(c) \wedge \\ & [\text{Graduate}(s, d) \Leftarrow \text{Take}(s, c)].\end{aligned} \tag{1}$$

This rule should be read: "Student s is a near graduate of department d if there exists some course c such that, if s had taken c, then s would be a graduate of d." Rules of this sort, called *embedded implications*, have been shown to be useful for hypothetical reasoning [3], for legal reasoning [30], for modular logic programming [32], and for natural language processing [34].

In this paper, we are primarily interested in negation. To represent negation, we use a special nullary predicate '$\perp$' which denotes a contradiction, and we write '$\neg Q$' as an abbreviation of '$\perp \Leftarrow Q$'. We also write '$\neg P \Leftarrow Q$' as an abbreviation of '$\perp \Leftarrow P \wedge Q$'. For example, the following is a legal expression in our language:

$$\begin{aligned}\neg\text{Graduate}(s, d) \;\Leftarrow\;& \text{Student}(s) \wedge \text{Department}(d) \wedge \text{Required}(c, d) \wedge \\ & \neg\text{Take}(s, c).\end{aligned} \tag{2}$$

This expression should be read: "Student s is not a graduate of department d if there exists some required course c such that s has not taken c." Expanding the abbreviations in rule (2), we have:

$$\begin{aligned} \bot \;\Leftarrow\;& \text{Graduate}(s,d) \wedge \text{Student}(s) \wedge \text{Department}(d) \wedge \text{Required}(c,d) \wedge \\ & [\bot \Leftarrow \text{Take}(s,c)]. \end{aligned} \tag{3}$$

In other words, it is a contradiction if student s is a graduate of department d, course c is required in department d, and 'Take(s,c)' implies a contradiction.

We call rule (1) a *simple* embedded implication and rule (3) a *simple* embedded negation, since the expression on the right-hand side of each rule is just a Horn clause with no further embeddings. But multiple embeddings are permitted. In particular, we may write an expression '$\neg\neg Q(x)$' on the right-hand side of a rule, as in: $P(x) \Leftarrow \neg\neg Q(x)$. Expanding the double negation, this rule is equivalent to:

$$P(x) \;\Leftarrow\; [\bot \Leftarrow [\bot \Leftarrow Q(x)]].$$

Rules of this sort will play a prominent role in our encoding of defaults in Section 3. Note, however, that we can always eliminate multiple embeddings by defining new atomic predicates. For example, we can define the predicate *NotQ* as follows:

$$\begin{aligned} \bot \;&\Leftarrow\; Q(x) \wedge \mathit{NotQ}(x), \\ \mathit{NotQ}(x) \;&\Leftarrow\; [\bot \Leftarrow Q(x)], \end{aligned}$$

and then rewrite the previous rule as: $P(x) \Leftarrow [\bot \Leftarrow \mathit{NotQ}(x)]$.

Our intended semantics for embedded implications and embedded negations is *intuitionistic*, rather than classical. This means that '$Q \vee \neg Q$' is not a tautology, and '$\neg\neg Q$' does not imply 'Q'. We assume that the reader is generally familiar with the Kripke semantics for intuitionistic logic, as given in [25, 14, 48], and we will simply review our notation here. Let $\mathcal{L}$ be a first-order language, and let $\mathcal{L}(\mathbf{c})$ be the language $\mathcal{L}$ augmented by an arbitrary set of new constants $\mathbf{c}$. We write a Kripke structure for $\mathcal{L}$ as a quadruple $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$, where

- $\mathbf{K}$ is a nonempty set of *substates*,
- $\leq$ is a partial order on $\mathbf{K}$,
- $\mathbf{h}$ is a mapping from the substates of $\mathbf{K}$ to sets of ground atomic formulae in $\mathcal{L}(\mathbf{c})$, and
- $\mathbf{u}$ is a mapping from the substates of $\mathbf{K}$ to sets of individual constants in $\mathcal{L}(\mathbf{c})$.

In addition, for every $s \in \mathbf{K}$, we require that $\mathbf{u}(s)$ includes all the individual constants appearing in $\mathbf{h}(s)$, and for every s_1 and s_2 in $\mathbf{K}$, we require that $\mathbf{u}(s_1) \subseteq \mathbf{u}(s_2)$ and $\mathbf{h}(s_1) \subseteq \mathbf{h}(s_2)$ whenever $s_1 \leq s_2$. We then define a relation '$s, \mathbf{K} \models \mathcal{A}$' between substates s of $\mathbf{K}$ and formulae $\mathcal{A}$ of $\mathcal{L}(\mathbf{c})$, as follows:

Definition 2.1:

$s, \mathbf{K} \models A$	iff	$A \in \mathbf{h}(s)$, for A a ground atomic formula,
$s, \mathbf{K} \models \mathcal{A} \wedge \mathcal{B}$	iff	$s, \mathbf{K} \models \mathcal{A}$ and $s, \mathbf{K} \models \mathcal{B}$,
$s, \mathbf{K} \models \mathcal{A} \vee \mathcal{B}$	iff	$s, \mathbf{K} \models \mathcal{A}$ or $s, \mathbf{K} \models \mathcal{B}$, and all the individual constants in $\mathcal{A}$ and $\mathcal{B}$ are in $\mathbf{u}(s)$,
$s, \mathbf{K} \models \mathcal{B} \Leftarrow \mathcal{A}$	iff	$s', \mathbf{K} \models \mathcal{A}$ implies $s', \mathbf{K} \models \mathcal{B}$ for every $s' \geq s$ in $\mathbf{K}$, and all the individual constants in $\mathcal{A}$ and $\mathcal{B}$ are in $\mathbf{u}(s)$,
$s, \mathbf{K} \models (\exists x)\mathcal{A}(x)$	iff	$s, \mathbf{K} \models \mathcal{A}(c)$, for some individual constant c in $\mathbf{u}(s)$,
$s, \mathbf{K} \models (\forall x)\mathcal{A}(x)$	iff	$s', \mathbf{K} \models \mathcal{A}(c)$ for every $s' \geq s$ in $\mathbf{K}$, and for all individual constants c in $\mathbf{u}(s')$.

This relation is often read "s forces $\mathcal{A}$ in $\mathbf{K}$." If $s, \mathbf{K} \models \mathcal{A}$ for every $s \in \mathbf{K}$, we say that $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ *satisfies* $\mathcal{A}$. If $s, \mathbf{K} \models \mathcal{A}$ for every $s \in \mathbf{K}$ such that the individual constants in $\mathcal{A}$ are in $\mathbf{u}(s)$, then we say that $\mathcal{A}$ is *true* in $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$. Finally, if Φ is a set of sentences and ψ is a sentence, we write $\Phi \models \psi$ if and only if ψ is true in every Kripke structure that satisfies Φ.

Definition 2.1, as written, actually characterizes *minimal logic* [23, 14] rather than intuitionistic logic, since it does not include a rule for negation. To specify full first-order intuitionistic logic, we add the symbol '$\perp$' to our language, as suggested above, and we define '$\neg\mathcal{A}$' to be an abbreviation of '$\perp \Leftarrow \mathcal{A}$'. Semantically, we stipulate that $\perp \notin \mathbf{h}(s)$ for every $s \in \mathbf{K}$. Alternatively, we could define negation directly, as follows:

Definition 2.2:

$s, \mathbf{K} \models \neg\mathcal{A}$ iff $s', \mathbf{K} \not\models \mathcal{A}$ for every $s' \geq s$ in $\mathbf{K}$, and all the individual constants in $\mathcal{A}$ are in $\mathbf{u}(s)$.

It is straightforward to verify that these two formulations of negation are equivalent [14].

For logic programming purposes, of course, we are not interested in full first-order intuitionistic logic, but rather in a proper subset consisting of embedded implications. Formally, we characterize this class of rules as follows:

Definition 2.3:

- An atomic formula is an embedded implication.
- If A is an atomic formula and if $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are embedded implications, then $A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$ is an embedded implication.

Note that rule (1) is covered by Definition 2.3. If we treat '$\perp$' as an atomic formula, then embedded negations become a special case of embedded implications, and rule (3) is also covered by Definition 2.3. The atomic formulae in Definition 2.3 may contain variables as well as constants, in which case we simply add universal quantifiers at the top level to obtain the class of *closed* embedded implications. (In [28, 29], we consider a broader class of formulae containing *embedded universal quantifiers*, but in the present paper we will assume that universal quantification occurs only at the top level.)

In [28], we showed that embedded implications and embedded negations, interpreted intuitionistically, have several properties that make them suitable for logic programming. First, a set of simple embedded implications, $\mathcal{R}$, has a "largest" Kripke model $\langle \mathbf{K^*}, \leq, \mathbf{h^*}, \mathbf{u^*} \rangle$, which may be obtained by a fixed-point construction analogous to the van Emden-Kowalski construction for Horn-clause logic [49]. More precisely, $\langle \mathbf{K^*}, \leq, \mathbf{h^*}, \mathbf{u^*} \rangle$ is a *final Kripke model* for $\mathcal{R}$, in the following sense: Given any Kripke model $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ of $\mathcal{R}$, there exists a unique homomorphism τ from $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ into $\langle \mathbf{K^*}, \leq, \mathbf{h^*}, \mathbf{u^*} \rangle$. Second, $\mathbf{K^*}$ has a "smallest substate" denoted by $\sqcap \mathbf{K^*}$. This means that every query '$(\exists \mathbf{x}) P(\mathbf{x})$' entailed by $\mathcal{R}$ has a definite answer substitution for the variables '$\mathbf{x}$'. A full discussion of these semantic properties is beyond the scope of the present paper, but they will be referred to briefly in Section 3.

A third property of embedded implications is of central importance to the present paper: the existence of a PROLOG-style proof procedure. Figure 1 shows a simple example of a proof in clausal intuitionistic logic, based on the definition of 'Near-Graduate' in rule (1). Assume that 'alice' is a student, that 'physics' is a department, and that the university has the following (trivial) graduation requirement:

$$\begin{aligned} \text{Graduate}(s, \text{physics}) \quad \Leftarrow \quad & \text{Take}(s, \text{math100}) \ \wedge \\ & \text{Take}(s, \text{physics100}) \ \wedge \\ & \text{Take}(s, \text{physics200}). \end{aligned} \tag{4}$$

Assume also that 'alice' has taken 'math100' and 'physics100', and that these facts have been asserted in the database. (Note that we are using italic type for variables

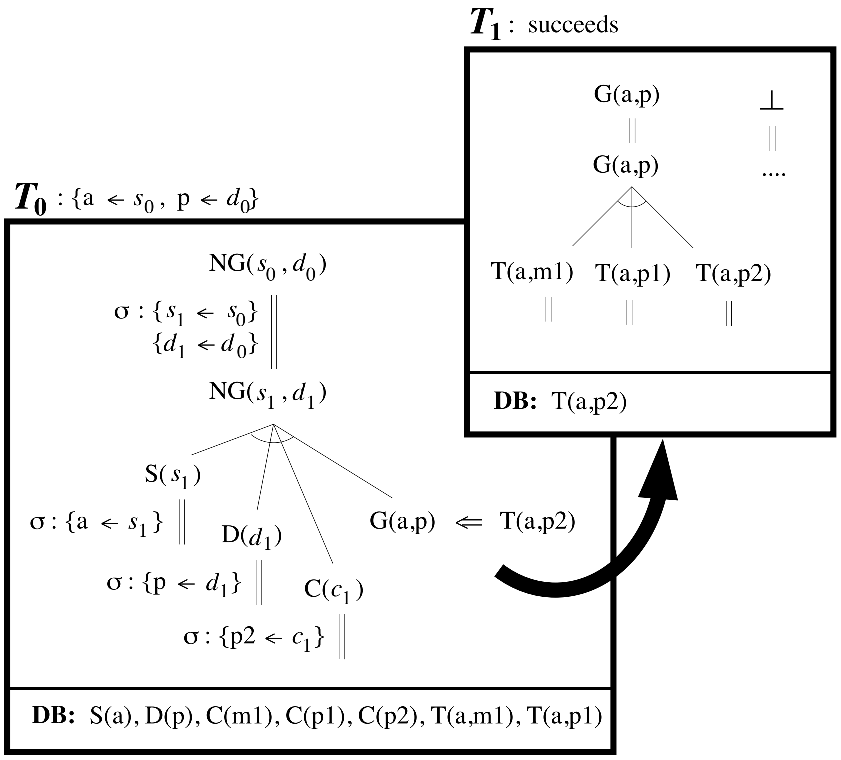

Figure 1: "Are there any near graduates?"

and roman type for constants.) We now pose the query:

$$(\exists s_0)(\exists d_0)\mathrm{NearGraduate}(s_0, d_0)?$$

Intuitively: "Are there any near graduates?" The proof procedure begins by constructing an SLD-refutation tree in an *initial tableau*, $\mathcal{T}_0$, with a *database* as shown, and with 'NG(s_0, d_0)' as the initial *goal*. Rule (1) is then applied to reduce this goal, in the usual way, to the goal:

$$\mathrm{Graduate(alice, physics)} \Leftarrow \mathrm{Take(alice, physics200)}.$$

Since this reduced goal is an implication, rather than an atomic formula, the proof procedure now constructs an *auxiliary tableau*, $\mathcal{T}_1$, with 'Take(alice,physics200)' in its database, and it tries to prove the goal 'Graduate(alice,physics)' in $\mathcal{T}_1$ using the databases of $\mathcal{T}_0$ and $\mathcal{T}_1$ combined. Because of rule (4), of course, the goal 'Graduate(alice,physics)' succeeds, which means that the tableau $\mathcal{T}_1$ succeeds. Thus the entire proof succeeds, with 's_0' bound to 'alice' and 'd_0' bound to 'physics'. We say that the proof in Figure 1 is a *closed tableau proof* with an *answer substitution* 'σ: $\{\mathrm{a} \leftarrow s_0,\ \mathrm{p} \leftarrow d_0\}$'. For additional examples of such proofs, see [29].

The proof in this example is particularly simple because the variables in the implicational goal can be fully instantiated before the auxiliary tableau is created. This will always be the case if each rule is *guarded* in the following sense: (1) every variable that appears in an implication on the right-hand side of the rule also appears in an atomic predicate on the right-hand side of the rule; and (2) there are no excess variables on the left-hand side of the rule. Note that rules (1) and (3) are both guarded. Given a set of guarded rules, we can always evaluate the atomic goals before the implicational goals and thus guarantee that the auxiliary tableaux will be ground. The initial query in $\mathcal{T}_0$ may contain free variables, of course, in which case the proof procedure will return a definite answer substitution. Let $\theta_1 \leq \theta_2$ mean that "the substitution θ_1 is less specific than the substitution θ_2," which is true if and only if there exists a substitution θ such that $\theta_1\theta = \theta_2$. Also, if σ is a substitution, let $\sigma(\mathbf{x})$ denote σ restricted to the variables '$\mathbf{x}$'. We then have the following result:

Theorem 2.4: Let $\mathcal{R}$ be a (possibly infinite) set of closed embedded implications, and let $\mathcal{A}(\mathbf{x})$ be an embedded implication with free variables '$\mathbf{x}$'. Let θ be a ground substitution for '$\mathbf{x}$'. Then $\mathcal{R} \models \mathcal{A}(\mathbf{x})\theta$ if and only if there exists a closed tableau proof of $\mathcal{A}(\mathbf{x})$ with an answer substitution σ such that $\sigma(\mathbf{x}) \leq \theta$.

Proof: See Theorem A.5 in Appendix A for a simple proof in the case of ground tableaux. The general case then follows from a lifting lemma [27, §8], in the usual way. For a direct proof of the soundness and completeness of closed tableaux with respect to the final Kripke model $\langle \mathbf{K^*}, \leq, \mathbf{h^*}, \mathbf{u^*} \rangle$, see [29].

The restriction of this result to guarded rules is not necessary: The proof procedure in [29] works for unguarded rules and embedded universal quantifiers combined, using two types of variables, $?\mathbf{x}$ and $!\mathbf{y}$. However, guarded rules are sufficient for the present

```
prove(tableau(G,DB)) :-
        db_member(G,DB).
prove(tableau((G,H),DB)) :-
        prove(tableau(G,DB)),
        prove(tableau(H,DB)).
prove(tableau(G,DB)) :-
        db_clause(G,H,DB),
        prove(tableau(H,DB)).
prove(tableau((G:-H),DB)) :-
        db_extend(H,DB,DB1),
        (prove(tableau(G,DB1))
        ;prove(tableau(f,DB1))).
```

Figure 2: An Interpreter for Clausal Intuitionistic Logic

paper, and we will simplify our discussion accordingly.

It is easy to write an interpreter for clausal intuitionistic logic that embodies the main features of the proof procedure in Figure 1. In fact, the `prove` predicate in Figure 2 is a simplified pedagogical version of our actual interpreter. The main data structure is `tableau(G,DB)` which asserts that the goal `G` is provable from the data base `DB` by means of a "tableau" proof. Most of the clauses of `prove` are taken from the standard meta-interpreter for PROLOG [44], except that `db_member` and `db_clause` retrieve items from the database `DB` as well as from the global rulebase. The new feature is the clause for embedded implications, `(G:-H)`. When this clause is encountered, the interpreter extends `DB` by adding `H` to it, and then tries to prove either `G` or `f` from the new database `DB1`. (The predicate '`f`' is our PROLOG encoding of the predicate '$\perp$'.) The attempt to prove `f` makes this an interpreter for intuitionistic logic; otherwise, it would be an interpreter for minimal logic. Of course, if `(G:-H)` is just `(f:-H)` then the interpreter tries (twice!) to prove `f`. In other words, to prove a negative goal '$\neg Q$' the interpreter simply adds 'Q' to the database and then tries to prove a contradiction.

Although the `prove` predicate in Figure 2 is inefficient and likely to loop, we have written more efficient versions that include loop checkers and other optimizations.[1] If our language is restricted to rules in the form (1)–(3), without function symbols and with universal quantifiers restricted to the top level only, then it is decidable. In fact, Bonner has shown in [2] that this version of the language is PSPACE complete. However, this worst-case complexity result does not appear to cause problems for the "common sense" rulebases we have investigated.

3 Default Rules

We now introduce a *nonmonotonic* version of our language. The basic idea is to combine a special failure operator, denoted by '$\sim$', with the operator for full intuitionistic negation, denoted by '$\neg$', to represent various kinds of default rules. We have to be careful about the intended interpretation of these rules, however. In [31], we develop a formal semantics for default rules using final Kripke models. In the present paper, we will proceed more informally, and try to convey an intuitive sense of the basic concept: the *justifiable presumption*. Monotonic rules generate logical consequences, we claim, but default rules only generate justifiable presumptions. What does this mean?

The general form of a default rule is as follows:

$$P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x}) \wedge \bigwedge_j \sim\neg D_j(\mathbf{x}) \tag{5}$$

where $\bigwedge_i G_i(\mathbf{x})$ is the monotonic component and each $\sim\neg D_j(\mathbf{x})$ is a default component of the rule. The intended interpretation is: "$P(\mathbf{x})$ is a justifiable presumption if each $G_i(\mathbf{x})$ is a justifiable presumption and each $D_j(\mathbf{x})$ is consistent." The monotonic component, $\bigwedge_i G_i(\mathbf{x})$, may be any expression that can appear on the right-hand side of a rule in clausal intuitionistic logic, but the default components must be *guarded*, i.e., $\bigwedge_i G_i(\mathbf{x})$ must include an atomic formula $A(x)$ for every variable x that appears

[1]We will send the reader, upon request, a copy of our full interpreter for clausal intuitionistic logic.

in $\bigwedge_j \sim\neg D_j(\mathbf{x})$. (This is to guarantee that all proofs by finite failure are ground.) Otherwise, the default component $\sim\neg D_j(\mathbf{x})$ may be any expression in the form:

$$\sim\neg \left[Q_{j,1}(\mathbf{x}) \wedge Q_{j,2}(\mathbf{x}) \wedge \ldots \bigwedge_k \neg [R_{j,k,1}(\mathbf{x}) \wedge R_{j,k,2}(\mathbf{x}) \wedge \ldots] \right], \tag{6}$$

where the Q_j's and the $R_{j,k}$'s are atomic formulae. Intuitively, the operator string '$\sim\neg$' means: "If you fail to show that it is not the case that ...," which also means: "If it is consistent that ...."

The flexibility provided by (6) is extremely useful. We will illustrate some of the possibilities with the "unemployed dropout" example from [40]. Consider the following set of rules:

$$\text{Adult}(x) \Leftarrow \text{Dropout}(x) \wedge \sim\neg\text{Adult}(x), \tag{7}$$

$$\neg\text{Employed}(x) \Leftarrow \text{Dropout}(x) \wedge \sim\neg\neg\text{Employed}(x), \tag{8}$$

$$\text{Employed}(x) \Leftarrow \text{Adult}(x) \wedge \sim\neg\neg\text{Dropout}(x). \tag{9}$$

Rule (7), in which the $R_{j,k}$'s in (6) are null, corresponds roughly to a normal default in Reiter's default logic. The literal reading of rule (7) is as follows: "If you show that x is a dropout, and if you fail to show that x is not an adult, then conclude that x is an adult." An alternative reading is: "If x is presumptively a dropout, and if it is consistent that x is an adult, then x is presumptively an adult." Rule (8) is similar, except that the negative literal '$\neg$Employed(x)' takes the place of the positive literal 'Adult(x)'. Note that rule (8) is actually an abbreviation for:

$$\perp \Leftarrow \text{Employed}(x) \wedge \text{Dropout}(x) \wedge \sim\neg\neg\text{Employed}(x),$$

which is an instance of the general form (5). Rule (9), in which the Q_j's in (6) are null, corresponds roughly to a nonnormal default in Reiter's default logic. It reads: "If x is presumptively an adult, and if it is consistent that x is not a dropout, then x is presumptively employed." However, another way to write rule (9) is to use both $Q_{1,1}$ and $R_{1,1,1}$ as follows:

$$\text{Employed}(x) \Leftarrow \text{Adult}(x) \wedge \sim\neg(\text{Employed}(x) \wedge \neg\text{Dropout}(x)). \tag{10}$$

Rule (10) corresponds roughly to a seminormal default in Reiter's default logic. It reads: "If x is presumptively an adult, and if it is consistent that x is both employed and not a dropout, then x is presumptively employed." As we will see, the difference between rules (9) and (10) is that rule (9) allows the contrapositive inference '$\neg\text{Adult}(x) \Leftarrow \neg\text{Employed}(x)$' to go through by default, while rule (10) automatically blocks it.

It is usually not very difficult to state the meaning of a default rule in isolation, whatever the system. The problems arise when we try to state the meaning of a set of interacting defaults. Let $\mathcal{R}$ be a set of embedded implications, and let $\mathcal{D}$ be a set of default rules. To arrive at our intended interpretation, let us first rewrite every rule in the form (5) as the following disjunction:

$$\left[P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})\right] \quad \vee \quad \bigvee_j \neg D_j(\mathbf{x}). \tag{11}$$

Note that (5) and (11) would be equivalent if '$\Leftarrow$' and '$\sim$' were interpreted as classical implication and classical negation, respectively, but they are not equivalent intuitionistically.[2] Nevertheless, we will take (11) as stating at least part of the meaning of (5). Either the implication '$P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})$' is true, it says, or else one of the expressions '$\neg D_j(\mathbf{x})$' is true, an indefinite state of affairs. We will now resolve this in-

[2]Each of the following steps would be an equivalence in classical logic:

$$P(\mathbf{x}) \;\Leftarrow\; \bigwedge_i G_i(\mathbf{x}) \;\wedge\; \bigwedge_j \sim\neg D_j(\mathbf{x}),$$

$$\left[P(\mathbf{x}) \;\Leftarrow\; \bigwedge_i G_i(\mathbf{x})\right] \;\Leftarrow\; \bigwedge_j \sim\neg D_j(\mathbf{x}),$$

$$\left[P(\mathbf{x}) \;\Leftarrow\; \bigwedge_i G_i(\mathbf{x})\right] \;\vee\; \sim\bigwedge_j \sim\neg D_j(\mathbf{x}),$$

$$\left[P(\mathbf{x}) \;\Leftarrow\; \bigwedge_i G_i(\mathbf{x})\right] \;\vee\; \bigvee_j \sim\sim\neg D_j(\mathbf{x}),$$

$$\left[P(\mathbf{x}) \;\Leftarrow\; \bigwedge_i G_i(\mathbf{x})\right] \;\vee\; \bigvee_j \neg D_j(\mathbf{x}),$$

but only the first of these steps would be an equivalence in intuitionistic logic.

definiteness by introducing an asymmetry into our interpretation of (11). Let us take '$P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})$' to be the *presumptive disjunct*, and let us derive definite conclusions from this disjunct together with all the other (definite!) embedded implications in $\mathcal{R}$. But what if '$\bigvee_j \neg D_j(\mathbf{x})$' is itself true? In this case, our presumption is *not justifiable*. In making this determination, however, we have to ask if '$\bigvee_j \neg D_j(\mathbf{x})$' is entailed by the monotonic rules $\mathcal{R}$ plus all the default rules $\mathcal{D}$ in the disjunctive form (11), since these are the statements known to be true in our domain. If so, we say that the presumption '$P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})$' is *blocked*. A presumption is justifiable, then, if and only if it is not blocked.

It is straightforward to extend this analysis from a single default rule to a set of default rules. We simply collect all of the disjuncts that are associated with the presumptions that we wish to use in our derivation, and we ask if this collective disjunction is entailed by $\mathcal{R} \cup \mathcal{D}$. For example, consider the rules (7)–(9) above. These rules would be rewritten as the following disjunctions:

$$[\text{Adult}(x) \Leftarrow \text{Dropout}(x)] \quad \vee \quad \neg\text{Adult}(x), \tag{12}$$

$$[\neg\text{Employed}(x) \Leftarrow \text{Dropout}(x)] \quad \vee \quad \neg\neg\text{Employed}(x), \tag{13}$$

$$[\text{Employed}(x) \Leftarrow \text{Adult}(x)] \quad \vee \quad \neg\neg\text{Dropout}(x). \tag{14}$$

Let us assume that $\mathcal{R}$ consists of the single assertion 'Dropout(andrew)', and let us adopt the obvious abbreviation: 'D(a)'. We can derive 'A(a) $\wedge$ $\neg$E(a)' using the implications in (12) and (13), as long as these presumptions are justifiable, that is, not blocked. We thus check to see if '$\neg$A(a) $\vee$ $\neg\neg$E(a)' is entailed by (12)–(14) in intuitionistic logic. It is not, and so we conclude that 'A(a) $\wedge$ $\neg$E(a)' is a justifiable presumption. Similarly, we could derive 'E(a)' using the implications in (12) and (14), if these presumptions were justifiable. However, 'D(a)' entails '$\neg\neg$D(a)' in intuitionistic logic, so the presumption in (14) is blocked. We thus conclude that 'Employed(andrew)' is not a justifiable presumption.

Using these concepts, it is easy to explain the difference between the default rules (9) and (10). If we retract 'D(a)' from the rulebase $\mathcal{R}$ and assert '$\neg$E(a)' instead, then '$\neg$A(a)' becomes a justifiable presumption by virtue of the implication in (14). In other words, rule (9) allows the contrapositive inference '$\neg$Adult(x) $\Leftarrow$ $\neg$Employed(x)' to go through by default. But suppose we replaced rule (9) by rule (10). The disjunctive form of rule (10) is:

$$[\text{Employed}(x) \Leftarrow \text{Adult}(x)] \quad \vee \quad \neg(\text{Employed}(x) \wedge \neg\text{Dropout}(x)), \tag{15}$$

and '$\neg$E(a)' entails '$\neg$(E(a) $\wedge$ $\neg$D(a))' in intuitionistic logic. Thus the presumption '$\neg$A(a)' would be blocked if we used this rule. In other words, a default rule in the form (10) does not allow contrapositive inferences.

We now present a precise characterization of these ideas using the intuitionistic entailment relation defined in Section 2. Let $\mathcal{R}$ be a set of embedded implications, and let $\mathcal{D}$ be a set of default rules written in the disjunctive form (11). We refer to $P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})$ as the *presumptive disjunct* of (11), and we refer to each expression $\neg D_j(\mathbf{x})$ in (11) as a *blocking disjunct*. Let s_0 be an *initial database* consisting of ground atomic formulae. Then:

Definition 3.1: $P(\mathbf{x})\theta$ is a *justifiable presumption* if and only if there exists some instantiated set of presumptive disjuncts $d = \{P_k(\mathbf{x})\theta_k \Leftarrow \bigwedge_i G_{k,i}(\mathbf{x})\theta_k\}$ such that

1. $\mathcal{R} \cup d \cup s_0 \models P(\mathbf{x})\theta$, but
2. $\mathcal{R} \cup \mathcal{D} \cup s_0 \not\models \bigvee_{k,j} \neg D_{k,j}(\mathbf{x})\theta_k$,

where $\{\neg D_{k,j}(\mathbf{x})\theta_k\}$ is the set of blocking disjuncts associated with d.

Note that the entailment relation in part (1) of this definition is just the entailment relation for clausal intuitionistic logic, since d is a set of embedded implications.

We develop an alternative formulation of Definition 3.1 in [31], using the concept of a final Kripke model. Although this discussion is beyond the scope of the present paper, we will outline the basic ideas here. First, we show that $\mathcal{R} \cup \mathcal{D}$ has a "largest" Kripke model $\langle \mathbf{E*}, \leq, \mathbf{h*}, \mathbf{u*} \rangle$, which may be obtained by a variant of the fixed-point construction in [28]. $\mathbf{E*}$ does not have a "smallest substate," of course, as $\mathbf{K*}$ does, but if $\mathbf{E*}$ is nonempty we can show that it has a nonempty set of *minimal* substates. We can thus use $\mathbf{E*}$ and its minimal substates to define the concept of a justifiable presumption. Specifically, we say that the presumptive disjunct $P(\mathbf{x})\theta \Leftarrow \bigwedge_i G_i(\mathbf{x})\theta$ is *blocked* at s if and only if $s, \mathbf{E*} \models \neg D_j(\mathbf{x})\theta$ for some associated disjunct $\neg D_j(\mathbf{x})\theta$. Now let $\Gamma(d)$ be the unique minimal substate generated by $\mathcal{R} \cup s_0$ plus some set of presumptive disjuncts $d = \{P_k(\mathbf{x})\theta_k \Leftarrow \bigwedge_i G_{k,i}(\mathbf{x})\theta_k\}$. This corresponds to part (1) of Definition 3.1. We say that $\Gamma(d)$ is a *uniformly justifiable substate* if there exists some minimal substate $e \geq \Gamma(d)$ in $\mathbf{E*}$ such that none of the presumptive disjuncts in d are blocked at e. This corresponds to part (2) of Definition 3.1. Finally, we say that $P(\mathbf{x})\theta$ is a *justifiable presumption* if it is contained in some uniformly justifiable substate. We provide several illustrations of these definitions in [31], and argue for their intuitive plausibility.

If $\mathcal{D}$ is a set of "normal defaults without prerequisites," i.e., if $D_j(\mathbf{x}) = P(\mathbf{x})$ and all the $G_i(\mathbf{x})$ are null, then these definitions give us a system similar to the system of Poole [36] or Ginsberg [20], except for the fact that we are using intuitionistic logic rather than classical logic. We have found, however, that the additional flexibility provided by the full syntax of (6) is invaluable if we are trying to write default rules in a natural way. Note that our definition of a justifiable presumption is a "brave" or "credulous" definition, since it refers to *some* uniformly justifiable substate. We could easily write a "cautious" or "skeptical" definition instead, by referring to *all maximal* uniformly justifiable substates. (We show in [31] that such maximal substates exist.) A similar alternative is provided in Poole's or Ginsberg's system.

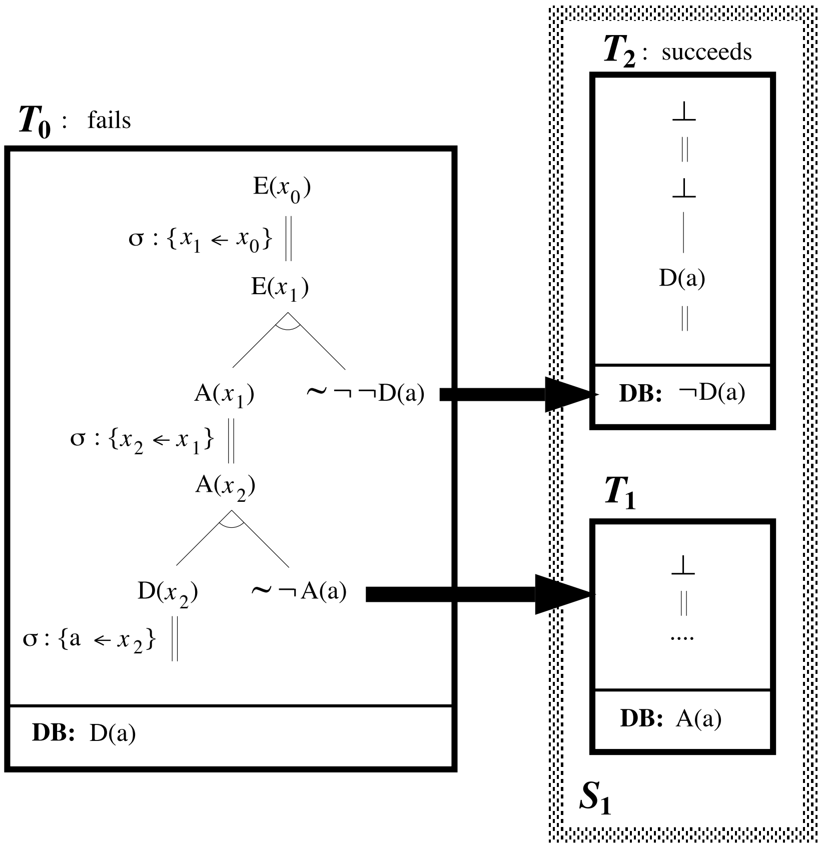

Figure 3: “Is anyone employed?”

4 Default Proofs

There are now countless proposals for systems of default reasoning in the literature [35]. Why propose another one? One of the main advantages of the default semantics outlined in Section 3 is that it possesses a very simple proof procedure. We describe our *default proof procedure* in this section, and we show in Section 5 how easy it is to implement in a PROLOG interpreter. The soundness and completeness of this proof procedure with respect to our definition of a justifiable presumption is proven in Appendix A.

An example of an intuitionistic default proof is shown in Figure 3. It has three parts. The first part, represented here by the tableau $\mathcal{T}_0$, is called a *monotonic tableau* *proof*. A monotonic tableau proof is exactly like the proofs for clausal intuitionistic logic illustrated in Figure 1, except for the possible presence of nodes containing the default operator '$\sim\neg$'. (In general, a monotonic tableau proof can sprout various auxiliary tableaux, as in Figure 1, although the proof in Figure 3 does not do so.) The second part, represented here by $\mathcal{T}_1$ and $\mathcal{T}_2$, consists of one or more *default tableaux* corresponding to the nodes in the monotonic tableau proof in which the operator '$\sim\neg$' appears. These default tableaux are grouped together as a *set*, which is labeled here as $\mathcal{S}_1$. In this example, the set $\mathcal{S}_1$ also comprises the third part of the default proof, which is called a *disjunctive proof*. (In general, a disjunctive proof would be more complex than this, as we will see in Figures 4 and 5.)

The basic idea of the proof procedure is simple: If the monotonic tableau proof succeeds, and the disjunctive proof fails, then the overall default proof succeeds. For example, Figure 3 shows an attempted proof of the query '$(\exists x_0)$Employed(x_0)?' given the default rules (7)–(9) and a database consisting of the assertion 'Dropout(andrew)'. Note that the initial tableau $\mathcal{T}_0$ would succeed with 'x_0' bound to 'andrew' if only the nodes containing '$\sim\neg$A(a)' and '$\sim\neg\neg$D(a)' could be presumed to succeed. To check this, we form the disjunction '$\neg$A(a) $\vee$ $\neg\neg$D(a)' and try to show that the proof of this disjunction fails. Thus, in $\mathcal{T}_1$, we add 'A(a)' to the database and try to prove '$\perp$', and in $\mathcal{T}_2$, we add '$\neg$D(a)' to the database and try to prove '$\perp$'. Of course, the proof in $\mathcal{T}_2$ succeeds immediately because of the presence of 'D(a)' in the database of $\mathcal{T}_0$. As a result, the disjunctive proof itself succeeds, which means that the overall default proof fails. In other words, 'Employed(andrew)' is not a justifiable presumption.

To analyze further the workings of this proof procedure, it is necessary to study an example that reveals more of the essence of intuitionistic reasoning. Consider the following rules:

$$\text{Scientist}(x) \;\Leftarrow\; \text{Professor}(x) \wedge \sim\neg(\text{Scientist}(x) \wedge \text{Conservative}(x)) \tag{16}$$

$$\text{Humanist}(x) \;\Leftarrow\; \text{Professor}(x) \wedge \sim\neg(\text{Humanist}(x) \wedge \text{Liberal}(x)) \tag{17}$$

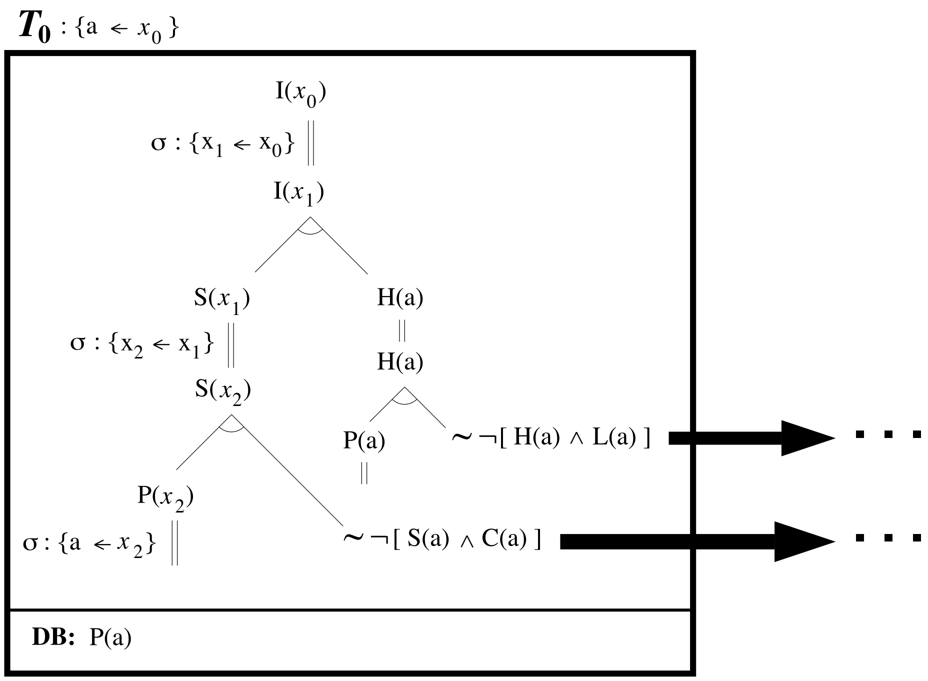

Figure 4: "Are there any intellectuals?"

$$\text{Intellectual}(x) \quad \Leftarrow \quad \text{Scientist}(x) \wedge \text{Humanist}(x) \tag{18}$$

$$\perp \quad \Leftarrow \quad \text{Conservative}(x) \wedge \text{Liberal}(x) \tag{19}$$

Intuitively: "If x is a professor, and if it is consistent that x is both a scientist and a conservative, then presume that x is a scientist. If x is a professor, and if it is consistent that x is both a humanist and a liberal, then presume that x is a humanist. An intellectual is someone who is both a scientist and a humanist, but nobody is both a conservative and a liberal." Let us now consider a database containing the assertion: 'Professor(aldous)', and let us pose the query:

$$(\exists x_0)\text{Intellectual}(x_0)?$$

In other words: "Are there any intellectuals?"

The monotonic tableau proof for this query is shown in Figure 4. As indicated, the initial tableau $\mathcal{T}_0$ would succeed with 'x_0' bound to 'aldous' as long as the nodes containing '$\sim\neg$[S(a)$\wedge$C(a)]' and '$\sim\neg$[H(a)$\wedge$L(a)]' could be presumed to succeed. To check this, we construct a default tableau $\mathcal{T}_1$ with 'S(a)' and 'C(a)' in its database and try to prove '$\bot$', and we construct a default tableau $\mathcal{T}_2$ with 'H(a)' and 'L(a)' in its database and try to prove '$\bot$'. Neither of these proofs would succeed using only the rules (16)–(19). Although '$\bot\Leftarrow$C(a)$\wedge$L(a)' implies '[$\bot\Leftarrow$C(a)] $\vee$ [$\bot\Leftarrow$L(a)]' in classical logic, this implication does not hold in intuitionistic logic. Thus, the disjunctive proof fails and the overall default proof succeeds. Intuitively, we are able to presume that 'aldous' is an intellectual because it is *consistent* that he is a conservative and it is *consistent* that he is a liberal, even though he cannot be both at the same time.

Figure 5 shows what happens to this example if we add the following rule to the set of defaults $\mathcal{D}$:

$$\text{Liberal}(x) \Leftarrow \text{Professor}(x) \wedge \sim\neg\text{Liberal}(x). \tag{20}$$

It is now possible to extend the proof inside the default tableau $\mathcal{T}_1$, as indicated. But how do we interpret the default operator '$\sim\neg$' when it occurs in this context? Recall that (20) is treated as a disjunctive assertion:

$$[\text{Liberal}(x) \Leftarrow \text{Professor}(x)] \quad \vee \quad \neg\text{Liberal}(x). \tag{21}$$

The tableau $\mathcal{T}_1$ succeeds using the presumptive disjunct 'L(a)$\Leftarrow$P(a)', but we must now check to see if the proof succeeds using the other disjunct '$\neg$L(a)'. To do this, we *split* the proof and copy the original default tableaux $\mathcal{T}_1$ and $\mathcal{T}_2$ into a set $\mathcal{S}_{11}$ with a *global database* that includes the assertion '$\neg$L(a)'. Since the tableau $\mathcal{T}_2$ now succeeds in $\mathcal{S}_{11}$, the disjunctive proof itself succeeds. Thus the overall default proof fails.

The example in Figure 5 is still relatively simple, since the disjunctive proof succeeds after a single split. In general, multiple disjunctive splits may be required. However, because of the simple structure of our default rules, a disjunctive proof can always be represented as a tree: $\mathcal{S}_1$, $\mathcal{S}_{11}$, $\mathcal{S}_{12}$, $\mathcal{S}_{121}$, $\mathcal{S}_{122}$, ... We define the concept

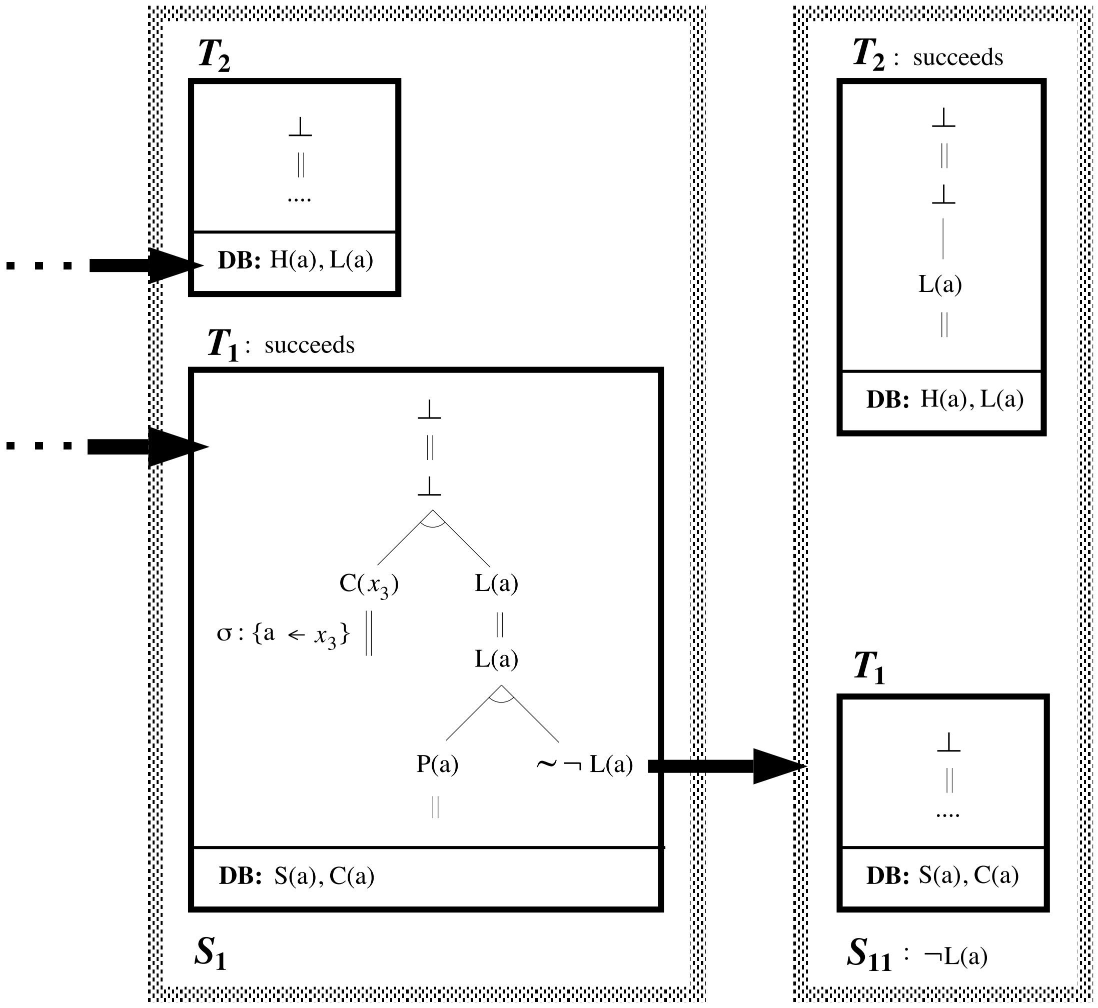

Figure 5: A successful disjunctive proof

of a *closed disjunctive proof* as follows: First, a set $\mathcal{S}_{i_1 i_2 \ldots i_n}$ is closed if one of the default tableaux in $\mathcal{S}_{i_1 i_2 \ldots i_n}$ closes without using any additional presumptive disjuncts. Second, $\mathcal{S}_{i_1 i_2 \ldots i_n}$ is closed if one of the default tableaux in $\mathcal{S}_{i_1 i_2 \ldots i_n}$ closes *with* the use of some set $\{P_k(\mathbf{x})\theta_k \Leftarrow \bigwedge_i G_{k,i}(\mathbf{x})\theta_k\}$ of presumptive disjuncts, as long as each set $\mathcal{S}_{i_1 i_2 \ldots i_n i_{n+1}}$ formed by adding one of the blocking disjuncts $\neg D_{k,j}(\mathbf{x})\theta_k$ to the global database of $\mathcal{S}_{i_1 i_2 \ldots i_n}$ also closes. In testing for the closure of these default tableaux, we use all the formulae in the global databases of $\mathcal{S}_{i_1 i_2 \ldots i_n}$ and its predecessors, and we apply the tableau proof procedure for clausal intuitionistic logic discussed in Section 2. Finally, we say that a disjunctive proof is closed if the initial set $\mathcal{S}_1$ is closed. In

Appendix A, we prove the following:

Theorem 4.1: Let $\mathcal{R}$ be a (possibly infinite) set of embedded implications, and let $\mathcal{D}$ be a (possibly infinite) set of default rules in the form (11). Let $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ be a set of default tableaux, and let s_0 be an initial database. Then $\mathcal{R} \cup \mathcal{D} \cup s_0 \models \bigvee_i \neg D_i(\mathbf{x})\theta_i$ if and only if there exists a closed disjunctive proof for $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ using $\mathcal{R}$ and $\mathcal{D}$ and the initial database s_0.

Note that Theorem 4.1 is a soundness and completeness result for the entailment relation in part (2) of Definition 3.1. Similarly, Theorem 2.4 is a soundness and completeness result for the entailment relation in part (1) of Definition 3.1. We have thus shown that our definition of a default proof procedure corresponds exactly to the definition of a justifiable presumption in Section 3.

There are several ways to organize the search for these proofs. In our examples, we searched first for a monotonic tableau proof, and then attempted to verify that the disjunctive proof failed. A similar search strategy seems reasonable for the disjunctive proof itself, i.e., in each set $\mathcal{S}$, we search first for a closed default tableau proof on some set of presumptive disjuncts, and we then attempt to verify that each disjunctive split $\mathcal{S}_j$ succeeds. This is the basis of the simple PROLOG interpreter described in the following section. Alternatively, if we use a mixed top-down/bottom-up search strategy, similar to the search strategy for clausal intuitionistic logic presented in [2], we can show that justifiable presumptions (in the function-free case) are decidable. In fact, since function-free clausal intuitionistic logic without embedded universal quantifiers is PSPACE complete [2], and since full propositional intuitionistic logic is also PSPACE complete [42], the same worst-case complexity bound would hold for our default proof procedure. Although there are special cases in which our proof procedure is in PTIME, e.g., if $\mathcal{R}$ consists of Horn clauses and $\mathcal{D}$ consists of normal defaults without prerequisites, the general question of complexity will not be investigated in the present paper.

5 A PROLOG Interpreter

Despite many years of work on many different versions of nonmonotonic logic, a practical system for default reasoning has remained an elusive goal [19]. Thus, one of the most important properties of the proof procedure described in Section 4 is that it can be implemented very easily as an interpreter for a logic programming language. Given such an interpreter, we can pose queries in the form '$(\exists \mathbf{x})P(\mathbf{x})$' to a rulebase consisting of a set of embedded implications, $\mathcal{R}$, and a set of default rules, $\mathcal{D}$. And, if it turns out that '$(\exists \mathbf{x})P(\mathbf{x})$' is a justifiable presumption from $\mathcal{R}$ and $\mathcal{D}$, then the interpreter will return a definite answer substitution for the variables '$\mathbf{x}$'. This makes it easy to write, debug and use a nonmonotonic rulebase.

In this section, we first describe our PROLOG interpreter, and we then discuss our experience encoding many of the benchmark problems in the literature.

5.1 The PROLOG Code

A simplified pedagogical version of our interpreter is reproduced in Figure 6. This version is inefficient and likely to loop, but we have written a more efficient version that includes a cache, a loop checker, and several simple search heuristics.[3] There are two stages to the interpreter, corresponding to the two stages of the proof procedure in Section 4. These two stages are encoded as the two conjuncts in the clause for `presume`. The first stage uses the `prove` predicate shown in Figure 2, but with one additional clause for `~G`. When the `prove` predicate encounters a goal in the form `~G`, it succeeds immediately, but adds `G` to a list of "finite failure" goals `FF`. The `presume` predicate then treats `FF` as a set of default tableaux with a global database `DB0`, and tries to show in the second stage of the proof that the `disjunctive_proof` predicate fails. However, the clause for `disjunctive_proof` succeeds (thus causing

[3]We will send the reader, upon request, a copy of our full interpreter for default rules. Our code also includes a preprocessor, so that rules written with the abbreviation '`not`' are translated into the internal form '`f:-`'.

```
presume(G,DB0) :-
        prove(tableau(G,DB0),FF),
        \+ disjunctive_proof(tableaux(FF,DB0)).

prove(tableau(G,DB),[]) :-
        db_member(G,DB).
prove(tableau((G,H),DB),FF) :-
        prove(tableau(G,DB),FF1),
        prove(tableau(H,DB),FF2),
        append(FF1,FF2,FF).
prove(tableau(G,DB),FF) :-
        db_clause(G,H,DB),
        prove(tableau(H,DB),FF).
prove(tableau((G:-H),DB),FF) :-
        db_extend(H,DB,DB1),
        (prove(tableau(G,DB1),FF)
        ;prove(tableau(f,DB1),FF)).
prove(tableau(~G,DB),[G]).

disjunctive_proof(tableaux(FF,DB)) :-
        member(F,FF),
        prove(tableau(F,DB),DBXs),
        split_proof_succeeds(DBXs,DB,tableaux(FF,DB)).

split_proof_succeeds([],_,_).
split_proof_succeeds([DBX|DBXs],DB,tableaux(FF,DB)) :-
        \+ db_member(DBX,DB),
        db_extend(DBX,DB,DB1),
        disjunctive_proof(tableaux(FF,DB1)),
        split_proof_succeeds(DBXs,DB,tableaux(FF,DB)).
```

Figure 6: An Interpreter for Default Rules

the overall proof to fail) if one of the tableaux in `FF` succeeds with a set of "finite failure" goals `DBXs`, and if all of the additional disjunctive proofs that arise when the global database is augmented by `DBXs` also succeed.

For example, suppose $\mathcal{D}$ consists of the rules (7)–(9) rewritten in the following notation:

```
adult(X) :- dropout(X), ~not adult(X).
```

```
not employed(X) :- dropout(X), ~not not employed(X).
employed(X) :- adult(X), ~not not dropout(X).
```

Suppose DB0 consists of the single assertion '`dropout(andrew)`'. Then the query '`employed(X)`' succeeds inside the `prove` predicate with

```
X = andrew,
FF = [(f:-adult(andrew)),(f:-(f:-(dropout(andrew))))].
```

However, the `disjunctive_proof` predicate also succeeds here, since the tableau proof for `(f:-(f:-(dropout(andrew))))` succeeds immediately with a null binding for `DBXs`. Thus the top-level `presume` predicate fails for the instantiation `X = andrew`. This shows that '`employed(andrew)`' is not a justifiable presumption.

For a more thorough understanding of the `disjunctive_proof` predicate, it is helpful to analyze the second example discussed in Section 4. Rules (16)–(19) would be rewritten in the following notation:

```
scientist(X) :- professor(X), ~not (scientist(X), conservative(X)).
humanist(X) :- professor(X), ~not (humanist(X), liberal(X)).
intellectual(X) :- scientist(X), humanist(X).
f :- conservative(X), liberal(X).
```

Suppose DB0 is '`[professor(aldous)]`', and suppose our goal is '`intellectual(X)`'. The initial `prove` predicate then succeeds, with `X = aldous`, and the interpreter tries to show that a disjunctive proof with

```
FF = [(f:-scientist(aldous), conservative(aldous)),
      (f:-humanist(aldous), liberal(aldous))]
```

fails. First, it sets up a tableau with `scientist(aldous)` and `conservative(aldous)` in the database and tries to prove `f`, but this fails. Second, it sets up a tableau with `humanist(aldous)` and `liberal(aldous)` in the database and tries to prove `f`, but this fails, too. Thus the `disjunctive_proof` predicate fails, and the top-level query succeeds with `X = aldous`.

However, let us see what happens to this example if we add the following version of rule (20) to the set of defaults $\mathcal{D}$:

```
liberal(X) :- professor(X), ~not liberal(X).
```

The attempt to prove `f` from `scientist(aldous)` and `conservative(aldous)` inside the `disjunctive_proof` predicate succeeds in this case, but with

```
DBXs = [(f:-liberal(aldous))].
```

The interpreter now invokes the `split_proof_succeeds` predicate. It is the job of `split_proof_succeeds` to loop through all the assertions in `DBXs`, add them to the global database `DB`, and then try to find a disjunctive proof from each extended database `DB1`. In our example, `DBXs` is a singleton, and the attempt to prove `f` from `humanist(aldous)` and `liberal(aldous)` succeeds immediately, once the assertion `(f:-liberal(aldous))` is added to the database. Thus the original disjunctive proof succeeds, and the top-level query fails for the instantiation `X = aldous`.

5.2 Examples

The existence of a PROLOG interpreter for our language makes it easy to write and debug a default rulebase with explicit exceptions. For example, suppose we wished to encode the rules: "Normally, elephants are gray" and "Normally, royal elephants are not gray" [41]. We suggest the following methodology: The first step is to write these rules down, in the simplest possible way, as normal defaults:

```
gray(X) :- elephant(X), ~not gray(X).
not gray(X) :- royal(X), elephant(X), ~not not gray(X).
```

However, it is well known that a system of normal defaults often generates too many answers [13, 12]. The next step, therefore, is to run various queries through our interpreter to determine whether any unintended presumptions are justifiable. For example, we might set up the following database:

```
DB0 = [elephant(clyde),african(clyde),royal(clyde),
        elephant(dumbo),african(dumbo)],
```

and discover that the query '`not gray(X)`' succeeds with `X = clyde`, as desired, but the query '`gray(X)`' succeeds with both `X = dumbo` and `X = clyde`. This tells us that we need to modify our rulebase, as follows:

```
gray(X) :- elephant(X), ~not (gray(X), not royal(X)).
not gray(X) :- royal(X), elephant(X), ~not not gray(X).
```

In the modified rulebase, we have replaced the normal default rule for '`gray(X)`' with a seminormal default rule that treats '`royal(X)`' as an explicit exception. We would now obtain the desired result: The query '`gray(X)`' succeeds with a single answer substitution, `X = dumbo`.

To test this methodology, we have tried out our interpreter on more than 40 standard examples from the literature, some of which are reproduced below. We found most of these examples easy, and many were quite trivial. We encountered no difficulties with examples *A1–A4, A6, B1–B4,* and *C1–C2* in Lifschitz's collection of benchmark problems for nonmonotonic reasoning [26]. Examples *A5* and *A7–A8* in [26] involve default quantifiers, which have not yet been incorporated into our interpreter, but which are discussed in [31]. We have also found satisfactory encodings for a set of 30 benchmark problems that were distributed at the *AAAI Workshop on Defeasible Reasoning with Specificity and Multiple Inheritance* in 1989 [10]. The following is a representative sample of our encodings:

Example 5.1: This example is from Horty and Thomason [22]. An English version might be written as follows:

- Normally, native speakers of Pennsylvania Dutch are born in America.

- Normally, native speakers of German are not born in America.
- All native speakers of Pennsylvania Dutch are native speakers of German.

This example presents some complications for inheritance networks, since it combines "strict" inheritance (the third rule) with "defeasible" inheritance (the first two rules). In our system, we simply put the third rule in $\mathcal{R}$ and the first two rules in $\mathcal{D}$, and we add an explicit exception to the second rule. Thus:

```
born(X,america) :-
        native_speaker(X,pa_dutch),
        ~not born(X,america).
not born(X,america) :-
        native_speaker(X,german),
        ~not (not born(X,america), not native_speaker(X,pa_dutch)).
native_speaker(X,german) :-
        native_speaker(X,pa_dutch).
```

Suppose we have the following initial database:

```
DB0 = [native_speaker(hermann,pa_dutch),
       native_speaker(fritz,german)].
```

Result: The query '`born(X,america)`' succeeds with `X = hermann`, and the query '`not born(X,america)`' succeeds with `X = fritz`. □

Example 5.2: This example is from Touretzky [46]. Most work on inheritance networks has been restricted to unary predicates, and this example was intended to illustrate a limited extension of the theory to binary predicates:

- Normally, citizens dislike crooks.
- Normally, gullible citizens like elected crooks.

In our system, of course, the arity of the predicates is irrelevant, and all rules are written in the same form:

```
dislike(X,Y) :-
        citizen(X), crook(Y),
        ~not not (gullible(X), elected(Y)).
like(X,Y) :-
        citizen(X), gullible(X), crook(Y), elected(Y),
        ~not like(X,Y).
f :- like(X,Y), dislike(X,Y).
```

Note that we have written the rule for '`dislike`' as a general nonnormal default, which allows contrapositive inferences. Alternatively, if we wished to block the contrapositive inferences, we could write the first rule as a seminormal default:

```
dislike(X,Y) :-
        citizen(X), crook(Y),
        ~not (dislike(X,Y), not (gullible(X), elected(Y))).
```

The forward inference works under either version. Suppose our initial database is:

```
DB0 = [citizen(fred),citizen(john),gullible(fred),
       crook(dick),elected(dick)].
```

Result: The query '`like(X,Y)`' succeeds with `X = fred, Y = dick`, and the query '`dislike(X,Y)`' succeeds with `X = john, Y = dick`. To see the contrapositive inference, assume now that we have written the first rule as a general nonnormal default. Suppose our initial database is:

```
DB0 = [like(mary,jane)].
```

Result: The query '`not (citizen(X),crook(Y))`' succeeds with `X = mary, Y = jane`. □

Example 5.3: This example is Problem *A6* in Lifschitz's collection of benchmark problems for nonmonotonic reasoning [26]. The English version is:

- Heavy blocks are normally located on the table.
- At least one of block 'a' or block 'b' is not on the table.
- Blocks 'a', 'b' and 'c' are heavy.

Lifschitz claims that the common sense conclusions should be:

- Block 'c' is on the table.
- Exactly one of block 'a' or block 'b' is not on the table.

We encode the rules in this problem as follows:

```
on(X,table) :- block(X), heavy(X), ~not on(X,table).
f :- on(a,table), on(b,table).
```

Note that the second rule here actually states that 'a' and 'b' are not *both* on the table, which is classically equivalent to the second sentence in the English version, but not equivalent intuitionistically. We encode the third sentence in the English version as the initial database:

```
DB0 = [block(a), block(b), block(c), heavy(a), heavy(b), heavy(c)].
```

Result: The query '`block(X), on(X,table)`' succeeds with `X = a`, `X = b` and `X = c`. In other words, the PROLOG interpreter returns each of these answer substitutions in succession. Similarly, the query '`block(X), not on(X,table)`' succeeds with `X = a` and `X = b`. This suggests that we try the query: '`not on(a,table), not on(b,table)`'. This query fails, as it should. Suppose now that the initial database is:

```
DB0 = [block(a), block(b), block(c), heavy(a), heavy(b), heavy(c),
        not on(b,table)]
```

Result: The query '`block(X), on(X,table)`' succeeds with `X = a` and `X = c`. The query '`block(X), not on(X,table)`' succeeds with `X = b`. □

In practice, for a small number of problems, we have found it useful to add two additional operators to our system. Brewka points out in [7] that certain kinds of rules involve exceptions to exceptions, as in the following example: "If you are an employee, you must go to the meeting, unless you have an excuse. If you are on vacation, that's an excuse. If you are sick, that's an excuse. But if you just have a cold, that's not an excuse." However, this example can be handled very nicely by adding stratified negation-as-failure [1, 37] to our language without invoking the full machinery of defaults. We have thus added the operator '`\+`' to our interpreter, and restricted it so that it uses only the monotonic rules $\mathcal{R}$ in searching for a proof. It is then natural to add the complementary operator '`\-`', with the same restriction that it uses only the monotonic rules $\mathcal{R}$. Thus '`\-`' means "provable without using any presumptions," and '`\+`' means "not provable without using any presumptions." Each of these operators requires only one additional line of code in our interpreter.

Example 5.4: This is an unusual example suggested by Ron Loui at the *AAAI Workshop on Defeasible Reasoning with Specificity and Multiple Inheritance* in St. Louis in 1989 [10]. The English version is:

- Dancers tend not to be ballerinas.
- Dancers tend to be graceful.
- Graceful dancers tend to be ballerinas.

Suppose we are told that "Naomi is a dancer." Do we conclude that Naomi is a ballerina? or not a ballerina? Do we arrive at a different conclusion if we are told directly that "Mikhail is a graceful dancer"?

Let us first encode these sentences as normal defaults:

```
not ballerina(X) :- dancer(X), ~not not ballerina(X).
graceful(X) :- dancer(X), ~not graceful(X).
ballerina(X) :- dancer(X), graceful(X), ~not ballerina(X).
```

Consider the initial database:

```
DB0 = [dancer(naomi)].
```

Result: The query '`ballerina(X)`' succeeds with `X = naomi`, and the query '`not ballerina(X)`' also succeeds with `X = naomi`. This ambiguous answer is certainly a plausible interpretation of the original sentences.

However, we may have in mind a different interpretation, in which the first sentence dominates unless we have been given conclusive (i.e., not presumptive) information that a particular dancer is graceful. We might then encode the ballerina example as follows:

```
not ballerina(X) :-
        dancer(X), \+ graceful(X), ~not not ballerina(X).
graceful(X) :- dancer(X), ~not graceful(X).
ballerina(X) :- dancer(X), \- graceful(X), ~not ballerina(X).
```

Now consider the initial database:

```
DB0 = [dancer(naomi),dancer(mikhail),graceful(mikhail)].
```

Result: The query '`ballerina(X)`' succeeds with `X = mikhail`, and the query '`not ballerina(X)`' succeeds with `X = naomi`. The query '`graceful(X)`' succeeds with both `X = naomi` and `X = mikhail`. □

We will not discuss the semantics of the operators '`\+`' and '`\-`' in the present paper. However, for a general discussion of stratified negation-as-failure added to (monotonic) clausal intuitionistic logic, see [4].

6 Conclusion

One alternative to developing a *syntactic* characterization of default inheritance is to explicitly encode exceptions to default rules. To date, the main obstacle to this approach has been the lack, in classical logic, of an adequate proof theory for seminormal or nonnormal defaults.

In this paper, we have proposed a system of default rules and explicit exceptions within the framework of an intuitionistic logic programming language. Intuitionistic logic programming, unlike classical logic programming, allows negated expressions to appear in both the premise and the conclusion of a rule. By combining full intuitionistic negation with a special kind of negation-as-failure, we obtain analogues of the normal, seminormal and general nonnormal defaults in Reiter's default logic. We also obtain a very simple top-down goal-directed interpreter for default rules which returns definite answer substitutions in response to existential queries. Our approach is similar in some ways to Poole's approach [36], but the use of intuitionistic logic allows us to make certain subtle distinctions (for instance, the blocking of contrapositive inferences) that are difficult to express within Poole's framework.

The use of default rules and explicit exceptions has often been criticized by the proponents of path-based inheritance systems [46]. The critique is based in part on the lack of a "modular" translation from inheritance networks to default logic [45]. However, this argument assumes one of the conclusions that needs to be established, namely, that inheritance networks provide the most perspicuous representation of default knowledge, and all other representations should be evaluated accordingly. Given the proliferation of alternative inference mechanisms that are now available for default inheritance networks (a recent count [10, p. 14] put the total at 72!), this proposition seems dubious at best. One could easily pose the translation question in reverse: For example, assuming that our analysis of rules (16)–(19) and rules (16)–(20) is intuitively correct ("Are there any intellectuals?"), does there exist a modular translation of this example into an inheritance network? If not, would this fact count against the theory of path-based inheritance systems?

A more important question, we believe, is the question of software engineering. Does our intuitionistic representation of default rules (or Poole's representation [36], or Reiter's representation [38], or Touretzky's representation [46], etc.) provide a useful tool for the construction of "common sense" knowledge bases? Is the language sufficiently expressive? Are the encoded rules comprehensible and modifiable? Are queries efficiently computable? Our experience so far has been promising, but we have only tested our system on small examples. We would like to apply our methodology to a large default rulebase, but we haven't been able to find one! If any of our readers have had some experience encoding complex systems of (real world!) defaults, we would be happy to make our PROLOG interpreter available for a series of experiments on a larger scale.

Acknowledgement

Thanks to David Poole and Randy Goebel for their helpful discussions, and to Alex Borgida, David Etherington, Stan Raatz and two anonymous referees for their comments on the original manuscript. William Cohen was supported by an AT&T Bell Laboratories Fellowship during the course of this research.

Appendix

A Soundness and Completeness

The goal in this appendix is to prove Theorem 4.1 stating the soundness and completeness of our disjunctive proof procedure. The proof is based on the soundness and completeness of clausal intuitionistic logic, which is also proven along the way as Theorem A.5. Lemma A.3 is central to both results.

We first present a modified sequent calculus for clausal intuitionistic logic. Let $\mathcal{R}$ be a set of closed embedded implications, as specified by Definition 2.3. We use the following inference system to determine whether $\mathcal{R} \vdash \psi$:

Definition A.1:

(1) $\mathcal{R} \vdash A$	if	there is a rule $P(\mathbf{x}) \Leftarrow \mathcal{A}_1(\mathbf{x}) \wedge \ldots \wedge \mathcal{A}_k(\mathbf{x})$ in $\mathcal{R}$ and a ground substitution θ such that $A = P(\mathbf{x})\theta$ and $\mathcal{R} \vdash \mathcal{A}_i(\mathbf{x})\theta$ for each i.
(2) $\mathcal{R} \vdash A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$	if	$\mathcal{R} \cup \{\mathcal{A}_1, \ldots, \mathcal{A}_k\} \vdash A$.
(3) $\mathcal{R} \vdash A$	if	$\mathcal{R} \vdash \bot$, for A a ground atomic formula.

We imagine that this inference system generates a *proof tree* with $\mathcal{R} \vdash \psi$ at the top (i.e., the proof tree is upside down for a sequent calculus, but rightside up for a tableau calculus). It is obvious that these proof trees correspond to ground instantiations of the tableau proofs discussed in Section 2.

We will use the inference system of Definition A.1 to construct a simple (almost trivial!) proof of the completeness of clausal intuitionistic logic. We first construct a *canonical Kripke structure* for $\mathcal{R}$. In our general definition of Kripke structures in Section 2, we worked with an extended language $\mathcal{L}(\mathbf{c})$, and we allowed $\mathbf{u}(s)$ to increase monotonically. However, for the simple class of rules given by Definition 2.3, it is sufficient to set $\mathbf{c} = \{\}$ and to define $\mathbf{u}(s)$ everywhere as the set of constants in $\mathcal{L}$. Thus:

Definition A.2: Let $\mathcal{R}$ be a (possibly infinite) set of closed embedded implications in $\mathcal{L}$, and let Φ be a *finite* set of ground embedded implications. We let $[\Phi]$ denote a *canonical substate* for $\mathcal{R}$, and we let $\mathbf{K}_{\mathcal{R}}$ be the set of all canonical substates for $\mathcal{R}$ as Φ ranges over all finite sets of ground embedded implications in $\mathcal{L}$. Define:

$[\Phi_1] \leq [\Phi_2]$ *iff* $\Phi_1 \subseteq \Phi_2$,

$$\mathbf{h}_{\mathcal{R}}([\Phi]) \;=\; \text{the set of ground atomic formulae, } A, \text{ in } \mathcal{L} \text{ such that } \mathcal{R} \cup \Phi \vdash A,$$

$$\mathbf{u}([\Phi]) \;=\; \text{the set of constants in } \mathcal{L}.$$

Now let $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}} = \{s \mid s \in \mathbf{K}_{\mathcal{R}}, \bot \notin s\}$. Then $\langle \mathbf{K}^{\mathbf{o}}_{\mathcal{R}}, \leq, \mathbf{h}_{\mathcal{R}}, \mathbf{u} \rangle$ is the *canonical Kripke structure* for $\mathcal{R}$. □

The following lemma states the most important property of canonical Kripke structures.

Lemma A.3: Let $[\Phi]$ be any substate in $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$, and let ϕ be any ground embedded implication in $\mathcal{L}$. Let $\mathcal{R}\Theta$ denote the set of all ground instantiations of the embedded implications in $\mathcal{R}$. Then:

1. $[\Phi], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \phi$ implies $\mathcal{R} \cup \Phi \vdash \phi$.
2. $\phi \in \mathcal{R}\Theta \cup \Phi$ implies $[\Phi], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \phi$.

Proof: The proof is by induction on the structure of ϕ. We will prove (1) and (2) simultaneously. For the base case, where ϕ is a ground atomic formula, (1) and (2) follow immediately from the definition of $\mathbf{h}_{\mathcal{R}}$ in the canonical Kripke structure. In particular, if $\phi = \bot$, then (1) and (2) are both true because their antecedents are false. For the inductive step:

1. Assume that $[\Phi], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \phi$, where ϕ has the form $A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$. Then:

   $$\begin{array}{rll} [\Phi \cup \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} & \models \mathcal{A}_i & \text{for } i = 1, \ldots, k, \text{ by part (2) of the induction hypothesis,} \\ [\Phi \cup \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} & \models A & \text{by Definition 2.1,} \\ \mathcal{R} \cup \Phi \cup \{\mathcal{A}_1, \ldots, \mathcal{A}_k\} & \vdash A & \text{by part (1) of the induction hypothesis in the base case.} \end{array}$$

   Thus $\mathcal{R} \cup \Phi \vdash A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$ by inference rule (2) in Definition A.1.

2. Assume that $\phi \in \mathcal{R}\Theta \cup \Phi$. Then ϕ has the form $A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$. Pick any substate $[\Phi'] \geq [\Phi]$ and assume that $[\Phi'], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \mathcal{A}_i$ for $i = 1, \ldots, k$.

Then:

$\mathcal{R} \cup \Phi'$	$\vdash \mathcal{A}_i$	for $i = 1, \ldots, k$, by part (1) of the induction hypothesis,
$\mathcal{R} \cup \Phi'$	$\vdash A$	by inference rule (1) in Definition A.1.

Now, if $A = \bot$, then $[\Phi'] \notin \mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$ by the definition of $\mathbf{h}_{\mathcal{R}}$, and we have a contradiction. Otherwise, $[\Phi'], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models A$ by the definition of $\mathbf{h}_{\mathcal{R}}$. In either case, it follows that $[\Phi], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models A \Leftarrow \mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_k$ by Definition 2.1.

$\Box$

Corollary A.4: If $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$ is nonempty, then $\langle \mathbf{K}^{\mathbf{o}}_{\mathcal{R}}, \leq, \mathbf{h}_{\mathcal{R}}, \mathbf{u} \rangle$ satisfies $\mathcal{R}$.

Proof: Pick any $\phi \in \mathcal{R}$ and any ground substitution θ for the variables in ϕ. Let $[\Phi]$ be any substate in $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$, and pick any $[\Phi'] \geq [\Phi]$. By part (2) of Lemma A.3, $[\Phi'], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \phi\theta$. Thus, by Definition 2.1, $[\Phi], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \phi$. Thus $\langle \mathbf{K}^{\mathbf{o}}_{\mathcal{R}}, \leq, \mathbf{h}_{\mathcal{R}}, \mathbf{u} \rangle$ satisfies ϕ. $\Box$

We can now prove the soundness and completeness of clausal intuitionistic logic for ground tableaux. Completeness for general tableaux then follows by a lifting lemma [27, §8], as stated in Theorem 2.4.

Theorem A.5: (Soundness and Completeness) Let ψ be a ground embedded implication in $\mathcal{L}$. Then $\mathcal{R} \models \psi$ if and only if $\mathcal{R} \vdash \psi$.

Proof: Soundness follows by a straightforward induction on the proof tree for $\mathcal{R} \vdash \psi$. For completeness, assume that $\mathcal{R} \models \psi$. We consider two cases. First, if $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$ is nonempty, then ψ is true in $\langle \mathbf{K}^{\mathbf{o}}_{\mathcal{R}}, \leq, \mathbf{h}_{\mathcal{R}}, \mathbf{u} \rangle$ by Corollary A.4. Let $\Phi_0 = \{\}$ and consider the substate $[\Phi_0]$ in $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$. Since $[\Phi_0], \mathbf{K}^{\mathbf{o}}_{\mathcal{R}} \models \psi$, it follows from part (1) of Lemma A.3 that $\mathcal{R} \cup \Phi_0 \vdash \psi$. In other words, $\mathcal{R} \vdash \psi$.

On the other hand, if $\mathbf{K}^{\mathbf{o}}_{\mathcal{R}}$ is empty, then every substate of $\mathbf{K}_{\mathcal{R}}$ must contain $\bot$. Letting $\Phi_0 = \{\}$ again, it follows from the definition of $\mathbf{h}_{\mathcal{R}}$ that $\mathcal{R} \vdash \bot$. Applying inference rules (2) and (3) from Definition A.1, we have $\mathcal{R} \vdash \psi$. $\Box$

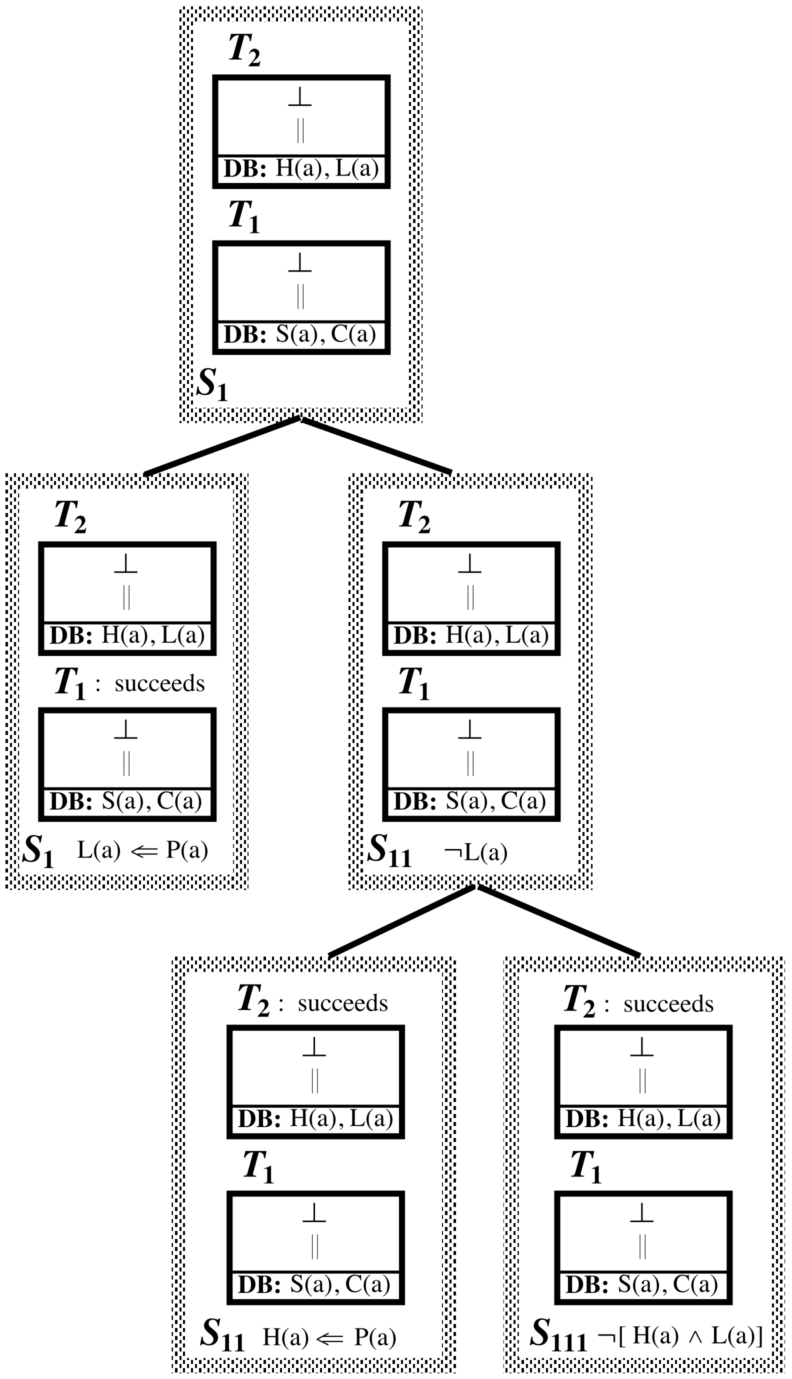

Figure 7: A disjunctive tree

This proof is a variant of the completeness proof presented in [5].

We will now use these results to prove Theorem 4.1 in Section 4. It is convenient to work directly with *disjunctive trees* instead of disjunctive proofs. Figure 7 shows a disjunctive tree that might correspond to the disjunctive proof in Figure 5. Each set $\mathcal{S}$ of default tableaux is the same, but the construction procedure is different. First, we explicitly write out the 'And/Or' refutation trees inside each tableau $\mathcal{T}_i$, so that we can record the progress of a breadth-first (or otherwise "fair") search procedure. Second, whenever a presumptive disjunct is used to extend one of the 'And/Or' trees, the overall proof tree branches downwards and the set of default tableaux in its current state of computation is copied exactly into the descendant nodes. The proof procedure then works only on the terminal nodes of the overall tree. We use the following convention: The *left-hand branch* is labeled with the presumptive disjunct $P(\mathbf{x})\theta \Leftarrow \bigwedge_i G_i(\mathbf{x})\theta$ that is used in the proof at that point, and the *right-hand branches* are labeled with the corresponding $\neg D_j(\mathbf{x})\theta$ instances. We think of each of these labels as additions to the *global rulebase* along the branch. The criteria for closure here is similar to the criteria for the closure of disjunctive proofs: A default tableau closes when its 'And/Or' tree closes; a set $\mathcal{S}$ of default tableaux closes when one member of the set closes; and a disjunctive tree closes when every branch terminates in a closed set of default tableaux. In Figure 7, we have assumed that the presumptive disjunct 'H(a)$\Leftarrow$P(a)' is used to extend the node $\mathcal{S}_{11}$ to the two descendant nodes: $\mathcal{S}_{11}$ and $\mathcal{S}_{111}$. However, each of these branches would close because of the presence of '$\neg$L(a)' in the global rule base, as in Figure 5.

For the completeness proof below, we need to work with a *fair* sequence of proof steps extending the 'And/Or' trees and the overall proof tree. This means: Whenever a proof step is applicable, it will be applied to extend the tree after some finite number of additional steps. It is easy to guarantee this: A queue of applicable proof steps will suffice, but many variations are possible. One useful restriction is to insist, whenever a presumptive disjunct is used to extend the node of an 'And/Or' tree, that this node is ineligible for subsequent extension in adjacent branches by the same presumptive disjunct. This forces the adjacent branches to close in some other way.

It is straightforward to convert a closed disjunctive tree into a closed disjunctive proof, and vice versa. Suppose we are given a closed disjunctive tree, as in Figure 7. First, identify the default tableau and the 'And/Or' subtree that caused the left-hand branch to close, and note the presumptive disjuncts, $P(\mathbf{x})\theta \Leftarrow \bigwedge_i G_i(\mathbf{x})\theta$, that were used in this closed subtree. Mark the corresponding right-hand branches, in which the formulae $\neg D_j(\mathbf{x})\theta$ have been added to the global rule base, and prune away the rest. Then repeat this procedure for all of the surviving (i.e., marked) right-hand branches. For example, in Figure 7, we would prune away the branch beginning with $\mathcal{S}_{111}$. Continue in this fashion until the entire disjunctive tree has been traversed. Then collapse the remaining left-hand branches into single sets of closed default tableaux (in this example: $\mathcal{S}_1$ and $\mathcal{S}_{11}$), and rearrange the resulting structure into the form of a disjunctive proof.

Conversely, suppose we are given a closed disjunctive proof, as in Figure 5. Start with the set of default tableaux in $\mathcal{S}_1$. One of these will close on some (possibly empty) set of presumptive disjuncts. Pick some ordering of these presumptive disjuncts consistent with the order imposed on the 'And/Or' trees, i.e., an order in which the guards close first. Then draw the left-hand branch of a disjunctive tree in this order, copying the default tableaux in $\mathcal{S}_1$ into successive nodes of the tree and copying the sets $\mathcal{S}_{11}, \mathcal{S}_{12}, \ldots$, into the appropriate right-hand branches. Repeat this procedure for $\mathcal{S}_{11}, \mathcal{S}_{12}, \ldots$, and so on. Continue in this fashion until the closed disjunctive proof has been completely copied into a closed disjunctive tree.

We will now prove the soundness and completeness of disjunctive trees, thus establishing the soundness and completeness of disjunctive proofs.

Theorem A.6: [Soundness] Let $\mathcal{R}$ be a (possibly infinite) set of embedded implications, and let $\mathcal{D}$ be a (possibly infinite) set of default rules in the form (11). Assume that there exists a closed disjunctive tree using these rules for some set of default tableaux $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ and some initial database s_0. Then $\mathcal{R} \cup \mathcal{D} \cup s_0 \models \bigvee_i \neg D_i(\mathbf{x})\theta_i$.

Proof: Assume, to the contrary, that there exists a Kripke model $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ which satisfies $\mathcal{R} \cup \mathcal{D} \cup s_0$, and a substate $s \in \mathbf{K}$ such that $s, \mathbf{K} \not\models \bigvee_i \neg D_i(\mathbf{x})\theta_i$. Since $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ satisfies $\mathcal{D}$, we can select a branch β in the closed disjunctive tree such that $s, \mathbf{K} \models \psi$ for every formula ψ in the global rulebase of β. Define:

$$\mathbf{K}(s) = \{s' \mid s' \geq s \text{ and } s' \in \mathbf{K}\}.$$

It is clear that $\langle \mathbf{K}(s), \leq, \mathbf{h}, \mathbf{u}, \rangle$ satisfies $\mathcal{R} \cup s_0$ plus the formulae in the global rulebase of β.

But this is a contradiction. Since the branch β is closed, the proof tree for one of the disjuncts $\neg D_i(\mathbf{x})\theta_i$ must be closed. Thus, by Theorem A.5 on the soundness of clausal intuitionistic logic, $s, \mathbf{K} \models \neg D_i(\mathbf{x})\theta_i$, which contradicts our original assumption that $s, \mathbf{K} \not\models \bigvee_i \neg D_i(\mathbf{x})\theta_i$. $\square$

Theorem A.7: [Completeness] Let $\mathcal{R}$ be a (possibly infinite) set of embedded implications, and let $\mathcal{D}$ be a (possibly infinite) set of default rules in the form (11). Let $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ be a set of default tableaux, and let s_0 be an initial database. Assume that $\mathcal{R} \cup \mathcal{D} \cup s_0 \models \bigvee_i \neg D_i(\mathbf{x})\theta_i$. Then there exists a closed disjunctive tree for $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ using $\mathcal{R}$ and $\mathcal{D}$ and the initial database s_0.

Proof: Assume, to the contrary, that there is no sequence of proof steps beginning with $\{\mathcal{T}_i : \neg D_i(\mathbf{x})\theta_i\}$ that results in a closed disjunctive tree. We will consider a *fair* sequence of proof steps, as described above, and use it to construct a Kripke model $\langle \mathbf{K}, \leq, \mathbf{h}, \mathbf{u} \rangle$ and a substate $s \in \mathbf{K}$ such that $s, \mathbf{K} \not\models \bigvee_i \neg D_i(\mathbf{x})\theta_i$. If the disjunctive tree generated by this fair sequence of proof steps is finite, we can select any finite failed branch β in the tree to use in our construction. If the disjunctive tree is infinite, we can select an infinite branch β by Koenig's lemma. In either case, to arrive at a contradiction, we will see that it is sufficient to construct a Kripke model using the formulae that were added to a single branch, β, during the course of the attempted proof.

Specifically, collect every disjunct $\neg D_j(\mathbf{x})\theta_j$ that is added to β in a (possibly infinite) rulebase $\mathcal{R}(\beta)$. In addition, if $P(\mathbf{x}) \Leftarrow \bigwedge_i G_i(\mathbf{x})$ is the presumptive disjunct of some default rule, and if θ is any ground substitution for the variables $\mathbf{x}$, and if there is no blocking disjunct $\neg D_j(\mathbf{x})\theta$ derived from this rule in β, then add the formula $P(\mathbf{x})\theta \Leftarrow \bigwedge_i G_i(\mathbf{x})\theta$ to $\mathcal{R}(\beta)$. By this construction, $\mathcal{R}(\beta)$ includes at least one disjunct from every instantiation of every default rule in $\mathcal{D}$.

Now let $\langle \mathbf{K^o}, \leq, \mathbf{h}, \mathbf{u} \rangle$ be the canonical Kripke structure for $\mathcal{R} \cup \mathcal{R}(\beta) \cup s_0$, as given in Definition A.2. Assume, first, that $\mathbf{K^o}$ is nonempty, and consider the substate $[\Phi_0]$ in which $\Phi_0 = \{\}$. $\langle \mathbf{K^o}, \leq, \mathbf{h}, \mathbf{u} \rangle$ satisfies $\mathcal{R} \cup \mathcal{R}(\beta) \cup s_0$ by Corollary A.4, and it is easy to see that $\langle \mathbf{K^o}, \leq, \mathbf{h}, \mathbf{u} \rangle$ also satisfies $\mathcal{R} \cup \mathcal{D} \cup s_0$. Suppose $[\Phi_0], \mathbf{K^o} \models \neg D_i(\mathbf{x})\theta_i$ for some disjunct $\neg D_i(\mathbf{x})\theta_i$ in the original set of default tableaux. Then, by part (1) of Lemma A.3, $\mathcal{R} \cup \mathcal{R}(\beta) \cup s_0 \vdash \neg D_i(\mathbf{x})\theta_i$. But this contradicts the fact that our proof procedure never closes on the branch β. We have thus shown that $[\Phi_0], \mathbf{K^o} \not\models \bigvee_i \neg D_i(\mathbf{x})\theta_i$.

On the other hand, if $\mathbf{K^o}$ is empty, then by the argument in the proof of Theorem A.5, $\mathcal{R} \cup \mathcal{R}(\beta) \cup s_0 \vdash \bot$. But this also contradicts the fact that our proof procedure never closes on the branch β. In either case, we have shown that the assumption that our proof procedure never closes leads to the conclusion that $\mathcal{R} \cup \mathcal{D} \cup s_0 \not\models \bigvee_i \neg D_i(\mathbf{x})\theta_i$. $\Box$

References

[1] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[2] A.J. Bonner. Hypothetical datalog: Complexity and expressibility. In *Proceedings of the Second International Conference on Database Theory*, pages 144–160. Springer-Verlag, 1988. Lecture Notes in Computer Science, volume 326.

[3] A.J. Bonner. A logic for hypothetical reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 480–484. AAAI, 1988.

[4] A.J. Bonner and L.T. McCarty. Adding negation-as-failure to intuitionistic logic programming. In *Logic Programming, Proceedings of the 1990 North American Conference*. MIT Press, 1990.

[5] A.J. Bonner, L.T. McCarty, and K. Vadaparty. Expressing database queries in intuitionistic logic. In *Logic Programming, Proceedings of the 1989 North American Conference*, pages 831–850. MIT Press, 1989.

[6] A. Borgida. Modelling class hierarchies with contradictions. In *Proceedings, ACM SIGMOD Conference*, pages 434–443, 1988.

[7] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1043–1048, 1989.

[8] M.R.B. Clarke and D.M. Gabbay. An intuitionistic basis for non-monotonic reasoning. In *Non-Standard Logics for Automated Reasoning*, pages 163–178. Academic Press, 1988.

[9] J.P. Delgrande. A first-order logic for prototypical properties. *Artificial Intelligence*, 33:105–130, 1987.

[10] J. Dorosh and R.P. Loui. Edited transcription, Workshop on defeasible reasoning with specificity and multiple inheritance. *ACM SIGART Bulletin*, 2(1):3–51, January 1991.

[11] D.W. Etherington. *Reasoning with Incomplete Information: Investigations of Nonmonotonic Reasoning.* PhD thesis, University of British Columbia, 1986.

[12] D.W. Etherington. More on inheritance hierarchies with exceptions: Default theories and inferential distance. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 352–357. AAAI, 1987.

[13] D.W. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 104–108. AAAI, 1983.

[14] M.C. Fitting. *Intuitionistic Logic, Model Theory and Forcing.* North-Holland, 1969.

[15] D.M. Gabbay. Intuitionistic basis for non-monotonic logic. In D.W. Loveland, editor, *Proceedings Sixth Conference on Automated Deduction*, pages 260–273. Springer-Verlag, 1982. Lecture Notes in Computer Science, volume 138.

[16] D.M. Gabbay and U. Reyle. N-PROLOG: An extension of PROLOG with hypothetical implications. I. *Journal of Logic Programming*, 1:319–355, 1984.

[17] D.M. Gabbay and M.J. Sergot. Negation as inconsistency. I. *Journal of Logic Programming*, 3:1–35, 1986.

[18] H. Geffner. On the logic of defaults. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 449–454. AAAI, 1988.

[19] M.L. Ginsberg, editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, 1987.

[20] M.L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[21] L. Hallnas and P. Schroeder-Heister. A proof-theoretic approach to logic programming. I. Generalized horn clauses. Technical Report R88005, Swedish Institute of Computer Science, 1988.

[22] J.F. Horty and R.H. Thomason. Mixing strict and defeasible inheritance. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 427–432. AAAI, 1988.

[23] I. Johansson. Der minimalkalkül, ein reduzierter intuitionistischer formalismus. *Compositio Mathematica*, 4:119–136, 1937.

[24] H.A. Kautz and B. Selman. Hard problems for simple default logics. In *Proceedings, First International Conference on Principles of Knowledge Representation and Reasoning*, pages 189–197, 1989.

[25] S.A. Kripke. Semantical analysis of intuitionistic logic. I. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130. North-Holland, 1965.

[26] V. Lifschitz. Benchmark problems for formal nonmonotonic reasoning. In M. Reinfrank et al., editors, *Nonmonotonic Reasoning: 2nd International Workshop*, pages 202–219. Springer-Verlag, 1988. Lecture Notes in Artificial Intelligence, vol. 346.

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

[28] L.T. McCarty. Clausal intuitionistic logic. I. Fixed-point semantics. *Journal of Logic Programming*, 5(1):1–31, 1988.

[29] L.T. McCarty. Clausal intuitionistic logic. II. Tableau proof procedures. *Journal of Logic Programming*, 5(2):93–132, 1988.

[30] L.T. McCarty. A language for legal discourse. I. Basic features. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 180–189. ACM Press, June 1989.

[31] L.T. McCarty. Programming directly in a nonmonotonic logic. Technical Report LRP-TR-21, Computer Science Department, Rutgers University, 1990.

[32] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79–108, 1989.

[33] L. Padgham. A model and representation for type information and its use in reasoning with defaults. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 409–414. AAAI, 1988.

[34] R. Pareschi. A definite clause version of categorial grammar. In *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, pages 270–277, Buffalo, NY, June 1988.

[35] D. Perlis. A bibliography of literature on nonmonotonic reasoning. In M.L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 466–477. Morgan Kaufmann, 1987.

[36] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[37] T. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.

[38] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[39] R. Reiter. Nonmonotonic reasoning. In *Annual Reviews of Computer Science*, volume 2, pages 147–187. 1987.

[40] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276, 1981.

[41] E. Sandewall. Nonmonotonic inference rules for multiple inheritance with exceptions. *Proceedings of the IEEE*, 74:1345–1353, 1986.

[42] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9(1):67–72, 1979.

[43] L.A. Stein. Skeptical inheritance: Computing the intersection of credulous extensions. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1153–1158, 1989.

[44] L. Sterling and E. Shapiro. *The Art of PROLOG: Advanced Programming Techniques.* MIT Press, 1986.

[45] R.H. Thomason and J.F. Horty. Logics for inheritance theory. In M. Reinfrank et al., editors, *Nonmonotonic Reasoning: 2nd International Workshop*, pages 220–237. Springer-Verlag, 1988. Lecture Notes in Artificial Intelligence, vol. 346.

[46] D.S. Touretzky. *The Mathematics of Inheritance Systems.* Pitman Research Notes in Artificial Intelligence. Pitman Publishing, 1986.

[47] D.S. Touretzky, J.F. Horty, and R.H. Thomason. A clash of intuitions: The current state of nonmonotonic multiple inheritance systems. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 476–482, 1987.

[48] A. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction.* North-Holland, 1988.

[49] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.