

Reasoning with infinite stable models

Piero A. Bonatti

Dip. di Tecnologie dell'Informazione - Università di Milano
I-26013 Crema, Italy
bonatti@dti.unimi.it

Abstract

The existing proof-theoretic and software tools for nonmonotonic reasoning can only handle finite domains. In this paper we introduce a class of normal logic programs, called *finitary programs*, whose domain may be infinite, and such that credulous and skeptical entailment under the stable model semantics are computable. Finitary programs—that are characterized by two conditions on their dependency graph—are computationally complete (they can simulate arbitrary Turing machines). Further results include a compactness theorem and the proof that the two conditions defining finitary programs are, in some sense, “minimal”. The existing methods for automated nonmonotonic reasoning are either complete for finitary programs, or can be easily extended to cover them.

1 Introduction

There is a complexity gap between propositional and first-order non-monotonic theories. Finite propositional theories are decidable (as in first-order logic), while the consequences of first-order theories are not recursively enumerable, in general. A computationally complete (Turing-equivalent) layer is missing between the two classes of theories. For this reason, most of the research on automated reasoning and proof-theory for nonmonotonic logics has been focussed on finite propositional theories or equivalent formalisms, such as function-free logic programs. In this way, the ability of reasoning about recursive data structures and infinite domains (such as lists, trees, XML/HTML documents, time, and so on) is completely lost. This is a strong limitation, both for standard tasks—such as reasoning about action and change when time is explicitly represented—and for emerging applications—such as using XML document bases as knowledge bases.

Of course, there may exist interesting, semi-decidable fragments of first-order nonmonotonic logics. For example, in some cases, the second-order circumscription formula can be expressed as a finite first-order formula. Similar examples are missing so far for Default Logic and Autoepistemic Logic.

Normal logic programs under the stable model semantics can be regarded as a fragment of both logics. In this paper

we introduce a Turing-equivalent class of programs, called *finitary programs*, that opens the way to effective default and autoepistemic reasoning about infinite domains. The main theoretical features of finitary programs are the following:

- Their domain may be infinite.
- Nonetheless both credulous and skeptical reasoning are semi-decidable. Ground queries are decidable.
- A form of compactness holds.
- Finitary programs are computationally complete, i.e. each Turing machine can be simulated by some finitary program.
- Finitary programs are defined by two conditions on their dependency graph, that are minimal, in the sense that if any condition were dropped, then semi-decidability and compactness would not be guaranteed anymore.

The paper is organized as follows: after a few technical preliminaries (Section 2), we characterize the subprogram needed to reason about a given ground formula F (Section 3). Then we introduce finitary programs (Section 4) and study their theoretical properties and expressiveness. In Section 5, the completeness proof for the skeptical resolution calculus introduced in [Bonatti, 1997] (originally formulated for function-free programs) is extended to all finitary programs and a generalization thereof (*almost finitary programs*). After a brief sketch of how existing credulous reasoners can be extended to finitary programs (Section 6), we conclude with a discussion of the results and related work. Some of the proofs are omitted due to space limitations.

2 Preliminaries

We assume the reader to be familiar with the classical theory of logic programming [Lloyd, 1984]. *Normal logic programs* (hereafter called simply “programs”) are sets of rules of the form $A \leftarrow L_1, \dots, L_n$ ($n \geq 0$) such that A is a logical atom and each L_i ($i = 1, \dots, n$) is a literal. As usual by “head” and “body” of such a rule we mean A and $L_1, \dots, L_n$, respectively. A program is positive if it contains no occurrences of $\neg$. The ground instantiation of a program P is denoted by $Ground(P)$. The *Gelfond-Lifschitz transformation* P^I of program P w.r.t. an Herbrand interpretation I (represented, as usual as a set of ground atoms) is obtained by removing from $Ground(P)$ all the rules containing a negative literal

$\neg B$ such that $B \in I$, and by removing from the remaining rules all negative literals. An interpretation M is a *stable model* of P if M is the least Herbrand model of P^M . A formula F is *credulously* (resp. *skeptically*) *entailed* by P iff F is satisfied by some (resp. each) stable model of P .

The *dependency graph* of a program P is a labelled directed graph whose vertices are the ground atoms of P 's language. Moreover, i) there exists an edge from B to A iff there is a rule $r \in \text{Ground}(P)$ with A in the head and an occurrence of B in the body; ii) such edge is labelled “negative” if B occurs in the scope of $\neg$, and “positive” otherwise. An atom A depends positively (resp. negatively) on B if there is a directed path from B to A in the dependency graph with an even (resp. odd) number of negative edges. A program is *call-consistent* if no atom depends negatively on itself. By *odd-cycle* we mean a cycle in the dependency graph with an odd number of negative edges. Call-consistency coincides with the absence of odd-cycles. Every call-consistent program has at least one stable model [Dung, 1992].

In the context of normal logic programs, a *splitting set* for a program P [Lifschitz and Turner, 1994] is a set of atoms U containing all the atoms occurring in the body of any rule $r \in \text{Ground}(P)$ whose head is in U . The set of rules $r \in \text{Ground}(P)$ whose head is in U —called the “bottom” of P w.r.t. U —will be denoted by $b_U(P)$. By $e_U(P, I)$ we denote the following partial evaluation of P w.r.t. $I \cap U$: remove from $\text{Ground}(P)$ each rule $A \leftarrow L_1, \dots, L_n$ such that some L_i containing an atom of U is false in I , and remove from the remaining rules all the L_i containing a member of U . The following is a specialization to normal programs of a result in [Lifschitz and Turner, 1994].

Theorem 2.1 (Splitting theorem) *Let U be a splitting set for a normal logic program P . An interpretation M is a stable model of P iff $M = J \cup I$, where*

1. I is a stable model of $b_U(P)$, and
2. J is a stable model of $e_U(P \setminus b_U(P), I)$.

3 Relevant subprograms

This section contains the basic technical results needed to investigate decidability and undecidability issues. These results prove that a certain strict subset of $\text{Ground}(P)$ suffices to decide credulous and skeptical entailment. Intuitively, all is needed for inference are the rules affecting the behavior of odd-cycles (which may generate inconsistencies in the form of instability) and the definitions of the predicates on which the given goal F depends.

Definition 3.1 [Relevant universe and subprogram] The *relevant universe* for a ground formula F (w.r.t. program P), denoted by $U(P, F)$, is the set of all ground atoms A such that the dependency graph of P contains a path from A to an atom occurring either in F or in some odd-cycle of the graph.

The *relevant subprogram* for a ground formula F (w.r.t. program P), denoted by $R(P, F)$, is the set of all rules in $\text{Ground}(P)$ whose head belongs to $U(P, F)$. ■

Example 3.2 Consider the program P consisting of the rules:

$$\begin{aligned} p(f(X)) &\leftarrow p(X), q(X) \\ q(X) &\leftarrow s(X) \\ u(a) &\leftarrow \neg u(a) \\ z(X) &\leftarrow p(X) \end{aligned}$$

and let $F = p(f(a))$. Here

$$\begin{aligned} U(P, F) &= \{ p(f(a)), p(a), q(a), s(a), u(a) \}, \\ R(P, F) &= \{ (p(f(a)) \leftarrow p(a), q(a)), (q(a) \leftarrow s(a)), \\ &\quad (u(a) \leftarrow \neg u(a)) \}. \end{aligned}$$

Note that for all rules $r \in \text{Ground}(P)$, if the head of r belongs to $U(P, F)$ then also all the other atoms in r belong to $U(P, F)$. The next proposition follows easily.

Proposition 3.3 *$U(P, F)$ is a splitting set for $\text{Ground}(P)$, and $R(P, F)$ coincides with $b_{U(P, F)}(P)$.*

Now the basic technical results can be proved.

Lemma 3.4 *For all ground formulae F , $R(P, F)$ has a stable model M_F iff P has a stable model M such that $M \cap U(P, F) = M_F$.*

Proof. (“If” part) Suppose M is a stable model of P . By Proposition 3.3, $U(P, F)$ is a splitting set for P and $R(P, F) = b_{U(P, F)}(P)$. Then, by the splitting theorem [Lifschitz and Turner, 1994], there exist a stable model I of $R(P, F)$, and a stable model J of $e_{U(P, F)}(P \setminus R(P, F), I)$, such that $M = I \cup J$. By definition, no atom in $U(P, F)$ occurs in $e_{U(P, F)}(P \setminus R(P, F), I)$, therefore $J \cap U(P, F) = \emptyset$. It follows that $M \cap U(P, F) = I$, and hence $M \cap U(P, F)$ is a stable model of $R(P, F)$.

(“Only if” part) Suppose $R(P, F)$ has a stable model M_F . By definition, all the atoms occurring in an odd-cycle belong to $U(P, F)$. Consequently, the dependency graph of $e_{U(P, F)}(P \setminus R(P, F), I)$ contains no odd-cycles, i.e. $e_{U(P, F)}(P \setminus R(P, F), I)$ is call-consistent. Then, by a well-known result in [Dung, 1992], $e_{U(P, F)}(P \setminus R(P, F), I)$ has a stable model J . Let $M = J \cup M_F$. By the splitting theorem, the interpretation M is a stable model of P . Moreover, since $J \cap U(P, F) = \emptyset$ (cf. point 1), $M \cap U(P, F) = M_F$. ■

Theorem 3.5 *For all ground formulae F ,*

1. P credulously entails F iff $R(P, F)$ does.
2. P skeptically entails F iff $R(P, F)$ does.

Proof. if P credulously entails F , then there exists a stable model M of P such that $M \models F$. By Lemma 3.4, $M \cap U(P, F)$ is a stable model of $R(P, F)$. Moreover, since by definition $U(P, F)$ contains all the atoms occurring in F , F must have the same truth value in M and $M \cap U(P, F)$, and hence $M \cap U(P, F) \models F$. As a consequence $R(P, F)$ credulously entails F .

Conversely, suppose that $R(P, F)$ credulously entails F . Then there exists a stable model M_F of $R(P, F)$ such that $M_F \models F$. By Lemma 3.4, P has a stable model M such that $M \cap U(P, F) = M_F$. Then the models M and M_F must agree on the valuation of F (cf. the “only if” part of the proof) and hence $M \models F$, which means that P credulously entails F . This completes the proof of 1).

To prove 2), we demonstrate the equivalent statement:

P does not skeptically entail F iff $R(P, F)$ does not skeptically entail F.

This statement is equivalent to: P credulously entails $\neg F$ iff $R(P, F)$ credulously entails $\neg F$, that follows immediately from 1). ■

4 Finitary programs

Now we introduce a class of programs whose consequences are recursively enumerable, although their domain may be infinite.

Definition 4.1 [Finitary programs] We say a program P is *finitary* if the following conditions hold:

1. For each node A of the dependency graph of P , the set of all nodes B such that A depends (either positively or negatively) on B is finite.
2. Only a finite number of nodes of the dependency graph of P occurs in an odd-cycle. ■

For example, most classical programs on recursive data structures such as lists and trees (e.g. predicates `member`, `append`, `reverse`) satisfy the first condition. In these programs, the terms occurring in the body of a rule occur also in the head, typically as strict subterms of the head's arguments. This property clearly entails Condition 1.

The second condition is satisfied by most of the programs used for embedding NP-hard problems into logic programs [Cholewiński *et al.*, 1995; Eiter and Gottlob, 1993; Gottlob, 1992]. Such programs can be (re)formulated by using a single odd cycle involving one atom p and defined by simple rules such as $p \leftarrow \neg p$ and $p \leftarrow f, \neg p$ (if p does not occur elsewhere, then f can be used as the logical constant *false* in the rest of the program).

An example of finitary program without odd-cycles is illustrated in Figure 1. It credulously entails a ground goal $s(t)$ iff t encodes a satisfiable formula. By adding rule $\perp \leftarrow \neg s(f), \neg \perp$ we obtain another finitary program with one odd-cycle, such that $s(t)$ is skeptically entailed iff the formula encoded by t is a logical consequence of the one encoded by f .

The following proposition follows straightforwardly from the definitions of $U(P, G)$, $R(P, G)$ and finitary programs. It will be needed to prove computability results.

Proposition 4.2 *If P is finitary then, for all ground goals G , $U(P, G)$ and $R(P, G)$ are finite.*

4.1 Compactness

In classical first-order logic, an infinite set of formulae is inconsistent iff it contains an inconsistent finite subset. A similar property—that in general does not apply to nonmonotonic logics—is enjoyed by finitary programs, too.

Definition 4.3 An *unstable kernel* for a program P is a subset K of $\text{Ground}(P)$ with the following properties:

1. K is *downward closed*, that is, for each atom A occurring in K , K contains all the rules $r \in \text{Ground}(P)$ whose head is A .
2. K has no stable models. ■

Theorem 4.4 (Compactness) *A finitary program P has no stable models iff it has a finite unstable kernel.*

Proof. Let G be any ground atom in the language of P . From Lemma 3.4, it follows that P has no stable models iff $R(P, G)$ has no stable models. Clearly, $R(P, G)$ is downward closed by definition. Moreover, by Proposition 4.2, $R(P, G)$ is finite. Therefore P has no stable models iff $R(P, G)$ is a finite unstable kernel of P . ■

4.2 Decidability and semi-decidability of inference

Hereafter we focus on the complexity of inference within the class of finitary programs. This subsection deals with upper bounds. We start with the proof that, for all ground goals, both credulous and skeptical inference are decidable.

Theorem 4.5 *For all finitary programs P and ground goals G , both the problem of deciding whether G is a credulous consequence of P and the problem of deciding whether G is a skeptical consequence of P are decidable.*

Proof. By Theorem 3.5, G is a credulous (resp. skeptical) consequence of P iff G is a credulous (resp. skeptical) consequence of $R(P, G)$. Moreover, by Proposition 4.2, $R(P, G)$ is finite, so the set of its stable models can be computed in finite time. It follows that the inference problems for P and G are both decidable. ■

It follows easily that existentially quantified goals are semi-decidable.

Theorem 4.6 *For all finitary programs P and all goals G , both the problem of deciding whether $\exists G$ is a credulous consequence of P and the problem of deciding whether $\exists G$ is a skeptical consequence of P are semi-decidable.*

Proof. The formula $\exists G$ is credulously (res. skeptically) entailed by P iff there exists a grounding substitution θ such that $G\theta$ is credulously (res. skeptically) entailed by P . The latter problem is decidable (by Theorem 4.5), and all grounding θ for G can be recursively enumerated, so existential entailment can be reduced to a potentially infinite recursive sequence of decidable tests, that terminates if and only if some $G\theta$ is entailed. ■

We shall see in the following section that this is a strict bound, i.e. existential entailment can be undecidable (cf. Corollary 4.11).

4.3 Minimality and expressiveness

Next we focus on lower bounds to the complexity of inference for the class of finitary programs and relaxations thereof. The next two results show that both of the conditions in Definition 4.1 are necessary for semi-decidability, i.e. Definition 4.1 is in some sense minimal.

Proposition 4.7 *Credulous and skeptical inference are not semi-decidable for the class of all programs satisfying Condition 2 of Definition 4.1.*

Proof. Note that locally stratified programs trivially satisfy Condition 2 of Definition 4.1 while some of them may violate Condition 1. For locally stratified programs, credulous and skeptical inference coincide with the well-founded semantics [Gelfond and Lifschitz, 1988] and are not semi-decidable [Schlipf, 1990], so the proposition immediately follows. ■

$s(\text{and}(X, Y)) \leftarrow s(X), s(Y)$	$s(A) \leftarrow \text{member}(A, [p, q, r, s]), \neg ns(A)$
$s(\text{or}(X, Y)) \leftarrow s(X)$	$ns(A) \leftarrow \text{member}(A, [p, q, r, s]), \neg s(A)$
$s(\text{or}(X, Y)) \leftarrow s(Y)$	$\text{member}(A, [A L])$
$s(\text{not}(X)) \leftarrow \neg s(X)$	$\text{member}(A, [B L]) \leftarrow \text{member}(A, L)$

Figure 1: A finitary program for SAT

The other lower-bound results are based on positive programs that decide whether a given Turing machine terminates using a fixed portion of its tape.

Let $\mathcal{M}$ be an arbitrary Turing machine; let S and V be $\mathcal{M}$'s set of states and vocabulary, respectively. Recall that the actions of $\mathcal{M}$ are defined by 5-tuples $\langle s, v, v', s', m \rangle$, where s and v are the current state and symbol, respectively, v' is the symbol to be overwritten on v , s' is the next state, and m specifies $\mathcal{M}$'s head movement.

Consider the positive program $P_{\mathcal{M}}$ in Figure 2. The reader may easily verify that $t(s, L, V, R)$ is entailed by $P_{\mathcal{M}}$ iff there exists a finite computation of $\mathcal{M}$, starting from a configuration with state s and tape described by L, V, R , and using only the finite portion of tape corresponding to L, V, R . Here L is a list representing a finite portion of the tape on the left of $\mathcal{M}$'s head, in reverse order; V is the current symbol; R is a (non reversed) finite portion of the tape on the right of $\mathcal{M}$'s head.

Program $P_{\mathcal{M}}$ satisfies Condition 1 of Definition 4.1. To prove this, note that the recursive calls to predicate t preserve the length k of the portion of tape encoded in the head. Therefore, for each ground atom $t(s, l, v, r)$, the number of atoms $t(s', l', v', r')$ connected to $t(s, l, v, r)$ by a directed path in the dependency graph is bounded by $|S| \cdot |V|^k$.

The next two results are based on $P_{\mathcal{M}}$ and prove that if Condition 2 in Definition 4.1 were dropped, then inference would not be semi-decidable anymore. More precisely, the theorems say that some inferences would be as complex as deciding the termination of an arbitrary Turing machine.

Theorem 4.8 *For each Turing machine $\mathcal{M}$ with initial state s and tape τ with non-blank portion $\langle v_0, v_1, \dots, v_n \rangle$, a program $P_{\mathcal{M}}^1$ and a goal G can be recursively constructed, such that $P_{\mathcal{M}}^1$ satisfies only Condition 1 in Definition 4.1, and $P_{\mathcal{M}}^1$ skeptically entails G iff $\mathcal{M}$ terminates.*

Proof. (Sketch) Let $P_{\mathcal{M}}^1$ consist of the program defined in Figure 2 plus the rules

$$\begin{aligned} u(L, R) &\leftarrow \text{blank_list}(L), \text{blank_list}(R), \\ &\quad t(s, L, v_0, [v_1, \dots, v_n | R]), \neg u(L, R). \\ \text{blank_list}([]). \\ \text{blank_list}([b | L]) &\leftarrow \text{blank_list}(L). \end{aligned}$$

where b represents the blank symbol. $P_{\mathcal{M}}^1$ satisfies Condition 1 of Definition 4.1 (cf. the argument for $P_{\mathcal{M}}$). Condition 2 of Definition 4.1 is violated, as there exist infinitely many odd-cycles, one for each ground atom $u(x, y)$. Clearly, for any grounding substitution θ , the goal

$$(\text{blank_list}(L), \text{blank_list}(R), t(s, L, v_0, [v_1, \dots, v_n | R]))\theta$$

can be derived from the subprogram $P_{\mathcal{M}}$ and the clauses for `blank_list` iff $\mathcal{M}$ terminates using the portion of τ represented by $L\theta, v_0, [v_1, \dots, v_n | R\theta]$. At the same time, $P_{\mathcal{M}}^1$ has a stable model iff the above goal cannot be derived from $P_{\mathcal{M}}$, because of the odd-cycle containing $u(R, L)\theta$. It follows that for all G consisting of a propositional symbol not occurring in $P_{\mathcal{M}}^1$, $P_{\mathcal{M}}^1$ skeptically entails G iff $\mathcal{M}$ terminates. ■

Theorem 4.9 *For each Turing machine $\mathcal{M}$ with initial state s and tape τ with non-blank portion $\langle v_0, v_1, \dots, v_n \rangle$, a program $P_{\mathcal{M}}^2$ and a goal G can be recursively constructed, such that $P_{\mathcal{M}}^2$ satisfies only Condition 1 in Definition 4.1, and $P_{\mathcal{M}}^2$ credulously entails G iff $\mathcal{M}$ terminates.*

Proof. Let $P_{\mathcal{M}}^2 = P_{\mathcal{M}}^1 \cup \{p\}$, where p is a new propositional symbol not occurring in $P_{\mathcal{M}}^1$, and let $G = p$. Clearly $P_{\mathcal{M}}^2$ credulously entails G iff $P_{\mathcal{M}}^2$ has a stable model. Since $P_{\mathcal{M}}^2$ has a stable model iff $\mathcal{M}$ terminates (cf. the previous proof), the theorem immediately follows. ■

The next result is based on a modification of $P_{\mathcal{M}}$ that keeps track of the result of the computation. The modified program $P_{\mathcal{M}}^3$ has one extra argument to return the final state of the tape. The theorem proves that the class of finitary programs is computationally complete by showing how any Turing machine can be simulated by a suitable finitary program.

Theorem 4.10 *For each Turing machine $\mathcal{M}$ with initial state s and tape τ with non-blank portion $\langle v_0, v_1, \dots, v_n \rangle$, a (positive) finitary program $P_{\mathcal{M}}^3$ and a goal $p(L, R, X)$ can be recursively constructed, in such a way that for all grounding substitutions θ , $P_{\mathcal{M}}^3 \models p(L, R, X)\theta$ iff $\mathcal{M}$ terminates and $X\theta$ encodes the final tape of the computation.*

Corollary 4.11 *The problems of deciding whether a finitary program P credulously/skeptically entails an existentially quantified goal $\exists G$ are not decidable.*

5 Resolution calculus

Here the results of the previous sections are applied to the resolution calculus for skeptical stable model semantics introduced in [Bonatti, 1997]. In the original paper, the resolution calculus was proved complete w.r.t. function-free programs (soundness holds for all programs). In this section, we extend the completeness result to all finitary programs.

Theorem 5.1 *Let P be a finitary program, and let $(\bigwedge H \rightarrow \bigwedge G)\gamma$ be a ground skeptical consequence of P , where H and G are sequences of literals. Then the skeptical goal $(G | H)$ has a successful skeptical derivation from P with answer substitution θ more general than γ .*

$\tau(s, L, v, [V \mid R]) \leftarrow \tau(s', [v' \mid L], V, R)$	for all 5-tuples $\langle s, v, v', s', \text{right} \rangle$
$\tau(s, [V \mid L], v, R) \leftarrow \tau(s', L, V, [v' \mid R])$	for all 5-tuples $\langle s, v, v', s', \text{left} \rangle$
$\tau(s, L, v, R)$	if there exists no tuple for s and v .

Figure 2: A program deciding termination of a Turing machine with bounded tape

Proof. Let $F = (\bigwedge H \rightarrow \bigwedge G)\gamma$. Since F is a ground skeptical consequence of P , then F is also a skeptical consequence of $R(P, F)$ (by Theorem 3.5). Moreover, $R(P, F)$ is finite (by Proposition 4.2). Then, by the original completeness result [Bonatti, 1997], $(G \mid H)\gamma$ has a ground skeptical derivation from $R(P, F) \subseteq \text{Ground}(P)$. By standard lifting techniques (cf. [Lloyd, 1984]), a corresponding skeptical derivation of $(G \mid H)$ from P with answer substitution θ more general than γ can easily be obtained. ■

Example 5.2 Consider again the program P of Example 3.2. This finitary program has no stable model, because of its third rule. Figure 3 shows a successful skeptical derivation for the skeptical conclusion $p(X)$, with empty answer substitution (which means that $\forall X.p(X)$ is a skeptical consequence of P). The Restricted Split of type II introduces two subgoals, each with a new hypothesis $u(a)$ and $\neg u(a)$, respectively), obtained from an atom occurring in some odd-cycle. The Contradiction rule replaces the left-hand side of a goal with the negation of some hypothesis. Intuitively, the goal is proved by showing that the hypotheses on the right hand side cannot be satisfied. The Failure rule rewrites a goal with one of its *counter-supports*. In this example the counter-support of $u(a)$ is $u(a)$ itself (obtained by negating the unique support $\{\neg u(a)\}$ of $u(a)$). Resolution can be performed either with a program rule or with one of the hypotheses (treated as facts). Finally, the Success rule removes goals where there is nothing left to prove. As usual, the empty goal sequence is denoted by $\square$. ■

With the help of the resolution calculus it can be shown that a class of normal programs larger than finitary programs is Turing equivalent. The extended class—called *almost finitary*—admits the kind of rules typically used to generate inconsistencies, without the restrictions of Definition 4.1.

Definition 5.3 [Almost finitary programs] A normal program P is *almost finitary* iff it can be partitioned into two (disjoint) subsets, P_1 and P_2 , such that P_1 is finitary, and P_2 consists of rules of the form $A \leftarrow B_1, \dots, B_n, \neg A$. ■

Note that there is no restriction on P_2 's rule variables. Given a ground instance $(A \leftarrow B_1, \dots, B_n, \neg A)\theta$ of a rule in P_2 , $A\theta$ may depend on infinitely many ground atoms B (i.e., Condition 1 of Definition 4.1 can be violated).

Theorem 5.4 *The skeptical resolution calculus is complete (in the sense of Theorem 5.1) for all almost finitary programs.*

Corollary 5.5 *The set of skeptical consequences of the form $\exists(\bigwedge H \rightarrow \bigwedge G)$, where H and G are sequences of literals, is semi-decidable for almost finitary programs.*

Remark 5.6 For almost finitary programs, ground credulous inference is not semi-decidable. This means that *skeptical inference from almost finitary programs cannot be implemented*

through credulous inference. Only direct approaches such as the resolution calculus are possible.

6 Credulous automated reasoners

For function-free programs, there exist powerful automated reasoners based upon stable model generation, such as *SMODELS* [Niemelä and Simons, 1997] and *DLV* [Eiter *et al.*, 1997]. Internally, these engines operate on ground programs; for this reason they embody smart program instantiation routines. Such routines are applied before any other form of reasoning. It seems not difficult to extend the instantiation routines to deal with all finitary programs, so that given a ground goal G ¹, only the relevant (ground and finite) subprogram $R(P, G)$ (or an equivalent subset thereof)² is generated. The rest of the engine needs no modification. By Theorem 3.5, soundness and completeness are preserved. In this way, our results can be used to extend the applicability range of this class of automated reasoners, with no modification to the core of their reasoning mechanisms.³

7 Summary and related work

For the first time, semi-decidable fragments of the stable model semantics have been explored in depth, and related to the number of atoms involved in odd-cycles. The main contributions of this paper can be summarized by recalling that finitary programs (Def. 4.1) are computationally complete (Theorem 4.10), can deal with function symbols, and enjoy a compactness result (Theorem 4.4) that guarantees semi-decidability of inference (Theorem 4.6). Finitary programs are characterized by two conditions on their dependency graph that are minimal, in the sense that if any of them were dropped, then inference would not always be semi-decidable (Theorems 4.7, 4.8 and 4.9). We proved that the skeptical resolution calculus is complete for finitary and almost finitary programs (Theorems 5.1 and 5.4), and sketched how other engines for nonmonotonic reasoning can be extended to deal with finitary programs (Section 6). We are currently extending our results to larger classes of programs, and to partial stable models.⁴

The work on finitary programs contributes to providing nonmonotonic logics with classical proof- and model-

¹These engines can only deal with ground queries

²The smart instantiation routines remove some non-applicable rule instances.

³As noted by an anonymous referee, the same idea can be applied to function-free programs, in order to reduce the cost of program instantiation.

⁴The latter idea and evidence to its feasibility have been suggested by an anonymous referee.

$(p(X) \mid)$	(Initial goal)
$(p(X) \mid u(a))$	Restricted Split of type II
$(\neg u(a) \mid u(a))$	Contradiction rule
$(u(a) \mid u(a))$	Failure rule
$(\square \mid u(a))$	Resolution with hypothesis
$(p(X) \mid \neg u(a))$	Success rule
$(u(a) \mid \neg u(a))$	Contradiction rule
$(\neg u(a) \mid \neg u(a))$	Resolution with $u(a) \leftarrow \neg u(a)$
$(\square \mid \neg u(a))$	Resolution with hypothesis
$\square$	Success rule

Figure 3: A skeptical derivation

theoretic tools and results. Work in a similar direction comprises some pretty standard axiomatizations based on Hilbert-style systems [Levesque, 1990]⁵ and sequent calculi [Olivetti, 1992; Bonatti and Olivetti, 1997]. These axiomatizations for Default and Autoepistemic logic should be extended to first-order theories, perhaps adapting the techniques introduced in this paper. An infinitary proof-theory can be found in [Milnikel, 1999]. In [Rosati, 1999], issues related to decidable nonmonotonic reasoning in MKNF are investigated. In [Cenzer et al., 1999], sufficient conditions for the existence of recursively enumerable stable models are identified. They do not ensure that all stable models are r.e., so skeptical and credulous reasoning are not semi-decidable.

Acknowledgments

The author is grateful to the anonymous referees for their deep and careful reviews, and for their precious suggestions.

References

- [Bonatti and Olivetti, 1997] P. A. Bonatti and N. Olivetti. A sequent calculus for skeptical default logic. In *Proceedings of TABLEAUX'97*, number 1227 in LNAI, pages 107–121. Springer Verlag, 1997.
- [Bonatti, 1997] P. A. Bonatti. Resolution for skeptical stable semantics. *Journal of Automated Reasoning*. To appear. Preliminary version in [Dix et al., 1997].
- [Cenzer et al., 1999] D. Cenzer, J. B. Remmel, and A. Vanderbilt. Locally determined logic programs. In *Proc. of LPNMR'99*, number 1730 in LNAI, pages 34–48, Berlin, 1999. Springer Verlag.
- [Cholewiński et al., 1995] P. Cholewiński, V. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proc. of ICLP'95*. MIT Press, 1995.
- [Dix et al., 1997] J. Dix, U. Furbach, and A. Nerode, editors. *Proc. of LPNMR'97*, number 1265 in LNAI, Berlin, 1997. Springer Verlag.
- [Dung, 1992] P. M. Dung. On the relation between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 105:7–25, 1992.
- [Eiter and Gottlob, 1993] T. Eiter and G. Gottlob. Complexity results for disjunctive logic programming and applications to nonmonotonic logics. In *Proc. of ILPS'93*. MIT Press, 1993.
- [Eiter et al., 1997] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In [Dix et al., 1997], 1997.
- [Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th ICLP*, pages 1070–1080. MIT Press, 1988.
- [Gottlob, 1992] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.
- [Levesque, 1990] H. J. Levesque. All I know: a study in autoepistemic logic. *Artificial Intelligence*, 42:263–309, 1990.
- [Lifschitz and Turner, 1994] V. Lifschitz and H. Turner. Splitting a logic program. In *Proc. of ICLP'94*, pages 23–37. MIT Press, 1994.
- [Lifschitz, 1985] V. Lifschitz. Computing circumscription. In *Proceedings of IJCAI'85*, pages 121–127, 1985.
- [Lloyd, 1984] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1984.
- [Milnikel, 1999] R. S. Milnikel. *Nonmonotonic logic: a monotonic approach*. PhD thesis, Cornell University, 1999.
- [Niemelä and Simons, 1997] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In [Dix et al., 1997], 1997.
- [Olivetti, 1992] N. Olivetti. Tableaux and sequent calculus for minimal entailment. *Journal of Automated Reasoning*, 9:99–139, 1992.
- [Rosati, 1999] R. Rosati. Towards first-order nonmonotonic reasoning. In *Proc. of LPNMR'99*, number 1730 in LNAI, pages 332–346, Berlin, 1999. Springer Verlag.
- [Schlipf, 1990] J. Schlipf. The expressive power of the logic programming semantics. In *Proc. of PODS'90*, 1990.

⁵This axiomatization of Autoepistemic logic, formulated for first-order theories, is complete only for the propositional fragment.