Finitary Open Logic Programs

Piero A. Bonatti

Università di Napoli Federico II
bonatti@na.infn.it

Abstract. Open logic programs and open entailment have been recently proposed as an abstract framework for the verification of incomplete specifications based upon normal logic programs and the stable model semantics. Open programs generalize both standard ASP and Abduction Frameworks, but can be embedded into standard ASP, thereby enabling reasoning with open domains by means of standard engines. However, the embedding can be an infinite program, or it may contain function symbols. In this paper, we show how to deal effectively with these cases by exploiting the theory of finitary normal programs. We obtain interesting classes of open programs and abduction frameworks with unbounded open domains and function symbols, where inference and explanations are nonetheless computable.

1 Introduction

Open logic programs and open entailment have been recently proposed as an abstract framework for the verification of incomplete specifications based upon normal logic programs and the stable model semantics [Bonatti 2001].

An important example of incomplete specifications is given by compound security policies [Bonatti et al. 2000], where details such as the set of users and the formulation of certain subpolicies are tipically unknown at verification time. In this setting, it is interesting to verify whether the policy will necessarily satisfy privacy laws, all parties' requirements, etc.

Logic-based agents are a second important example. IMPACT agent programs [Subrahmanian et al. 2000] must satisfy a property called *conflict freedom.* This property depends on a variable state that extends the agent program. The state is unknown at verification time, and can be regarded as a runtime extension of the agent program. Conflict freedom should hold for all possible such extensions.

Incomplete specifications are modelled by identifying a set of predicates, called *open predicates*, that are not completely defined in the program. Different forms of entailment capture what could possibly be true and what must necessarily be true, across a space of possible complete specifications of the open predicates.

Open programs generalize both standard normal programs under the stable model semantics, and abduction frameworks [Bonatti 2002b]. Conversely, open programs can be simulated by normal programs, by suitably extending the vocabulary [Bonatti 2002b]. Therefore, in principle, one can use any state-of-the-art ASP engine to compute open inferences.

However, in general, the embedding may produce infinite programs, or programs with function symbols. In this case, some restrictions are needed in order to make inferences computable. Moreover, the existing reasoning techniques need to be adapted: a direct use of the existing engines forbids unbounded open domains (cf. [Bonatti 2002b]).

In this paper, these difficulties are tackled by applying the theory of *finitary programs* [Bonatti 2001b,Bonatti 2002] to handle function symbols and unbounded open domains. The main contributions of the paper include an extension of the notion of finitary programs to open logic programs, decidability and semidecidability results for open programs with function symbols and unbounded domains, and a new technique for computing explanations from abduction frameworks with open domains and function symbols.

The basic definitions concerning abduction frameworks, open programs, and finitary programs will be recalled in Section 2. Then finitary open programs and their properties will be introduced (Section 3). Section 4 shows how to handle infinite open programs, such as those needed to capture abduction frameworks with open domains. Finally, the main results and future work are summarized and discussed in Section 5.

2 Preliminaries

We assume the reader to be familiar with normal logic programs and the stable model semantics [Gelfond and Lifschitz 1988]. We say that a normal logic program is *consistent* if it has at least one stable model.

An *open program* is a triple $\langle P, F, O \rangle$ where P is a normal logic program, F is a set of function and constant symbols not occurring in P, and O is a set of predicate symbols. A *completion*[1] of $\langle P, F, O \rangle$ is a normal logic program P' such that

1. $P' \supseteq P$,
2. the constant and function symbols of P' occur in P or F,
3. if $r \in P' \setminus P$, then $head(r) \in O$.

The set of all the completions of $\langle P, F, O \rangle$ is denoted by $\mathsf{Comp}(P, F, O)$. There exist four kinds of *open inference*:

1. (Credulous open inference) $\langle P, F, O \rangle \models^c \Psi$ iff for some $P' \in \mathsf{Comp}(P, F, O)$, P' credulously entails Ψ.
2. (Skeptical open inference) $\langle P, F, O \rangle \models^s \Psi$ iff for all $P' \in \mathsf{Comp}(P, F, O)$, P' skeptically entails Ψ.

[1] Note that this notion of completion has nothing to do with the standard notion of completion derived from Clark's work. We have not been able to identify an alternative term to express correctly the idea of complete predicate specification.

3. (Mixed open inference I) $\langle P, F, O \rangle \models^{cs} \Psi$ iff for some consistent $P' \in \mathsf{Comp}(P, F, O)$, P' skeptically entails Ψ.
4. (Mixed open inference II) $\langle P, F, O \rangle \models^{sc} \Psi$ iff for each consistent $P' \in \mathsf{Comp}(P, F, O)$, P' credulously entails Ψ.

The four open entailments are pairwise dual and form a diamond-shaped lattice [Bonatti 2001].

Proposition 1. *(Duality) For all open programs Ω and all sentences Ψ,*

1. $\Omega \models^{c} \Psi$ *iff* $\Omega \not\models^{s} \neg\Psi$;
2. $\Omega \models^{sc} \Psi$ *iff* $\Omega \not\models^{cs} \neg\Psi$.

Proposition 2. *(Entailment lattice) Suppose there exists a consistent $P' \in \mathsf{Comp}(\Omega)$. Then, for all sentences Ψ,*

1. $\Omega \models^{s} \Psi$ *implies* $\Omega \models^{cs} \Psi$ *and* $\Omega \models^{sc} \Psi$;
2. $\Omega \models^{cs} \Psi$ *implies* $\Omega \models^{c} \Psi$;
3. $\Omega \models^{sc} \Psi$ *implies* $\Omega \models^{c} \Psi$.

An *abduction framework* is a pair $\langle T, A \rangle$, where T is a normal logic program and A is a set of *abducible predicates.* Let $\mathsf{Abducibles}(T, A)$ be the set of all ground atoms $p(t_1, \ldots, t_n)$ such that $p \in A$ and t_i belongs to the Herbrand domain of T $(1 \leq i \leq n)$.

Definition 1. *A* generalized stable model *of an abduction framework $\langle T, A \rangle$ is a stable model of $T \cup E$, for some $E \subseteq \mathsf{Abducibles}(T, A)$.*

An explanation *of a closed sentence Q (called* observation*) w.r.t. an abduction framework $\langle T, A \rangle$ is a set $E \subseteq \mathsf{Abducibles}(T, A)$ such that there exists a stable model M of $T \cup E$ that satisfies Q.*

The following definitions [Bonatti 2002b] extend abductive frameworks so that the existence of new domain elements can be abduced. In the rest of the paper, let $Sk = \{c_i \mid 0 \leq i \leq \omega\}$ be an arbitrary but fixed set of (skolem) constants not occurring in the given abductive frameworks.

Definition 2. *For each abduction framework $\langle T, A \rangle$, let $\mathsf{Abducibles}^{Sk}(T, A)$ be a set of ground atoms $p(t_1, \ldots, t_n)$ such that $p \in A$ and each t_i is a term built from the function and constant symbols occurring in T and Sk.*

$\mathsf{Abducibles}^{Sk}(T, A)$ will be called a set of open abducibles *of the abduction framework $\langle T, A \rangle$ (w.r.t. the set of skolem constants Sk).*

Note that the choice of Sk is irrelevant, as long as it does not intersect the vocabulary of T.

Definition 3. *An* open generalized stable model *of an abduction framework $\langle T, A \rangle$ is a stable model of $T \cup E$, for some $E \subseteq \mathsf{Abducibles}^{Sk}(T, A)$.*

An open explanation *of a closed sentence Q (called* observation*) w.r.t. an open abduction framework $\langle T, A \rangle$ is a set $E \subseteq \mathsf{Abducibles}^{Sk}(T, A)$ such that there exists a stable model M of $T \cup E$ that satisfies Q.*

Proposition 3. [Bonatti 2002b] *M is an open generalized stable model of an abductive framework $\langle T, A \rangle$ iff M is a stable model of some $P \in \mathsf{Comp}(T, Sk, A)$.*

The *(atom) dependency graph* of a program P is a labelled directed graph denoted by $DG(P)$, whose vertices are the ground atoms of P's language. Moreover, (i) there exists an edge labelled '+' (called positive edge) from B to A iff for some rule $R \in \mathsf{Ground}(P)$, $A \in \mathit{head}(R)$ and $B \in \mathit{body}(R)$; (ii) there exists an edge labelled '−' (called negative edge) from B to A iff for some rule $R \in \mathsf{Ground}(P)$, $A \in \mathit{head}(R)$ and $\neg B \in \mathit{body}(R)$.

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. Moreover, each atom depends positively on itself. If A depends positively (resp. negatively) on B we write $A \geq_+ B$ (resp. $A \geq_- B$). We write $A \geq B$ if either $A \geq_+ B$ or $A \geq_- B$. If both $A \geq_+ B$ and $A \geq_- B$ hold then we write $A \geq_\pm B$.

By *odd-cycle* we mean a cycle in the dependency graph with an odd number of negative edges. A ground atom is *odd-cyclic* if it occurs in an odd-cycle.

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 $r \in \mathsf{Ground}(P)$ whose head is in U. The set of rules $r \in \mathsf{Ground}(P)$ 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 $\mathsf{Ground}(P)$ each rule $A \leftarrow L_1, \ldots, L_n$ 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 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 $e_U(P \setminus b_U(P), I)$.*

The next definitions and results, taken from [Bonatti 2001b,Bonatti 2002], characterize finitary programs.

Definition 4 (Finitary programs). *We say a program P is* finitary *if the following conditions hold:*

1. *Each ground atom A in $DG(P)$ depends on finitely many ground atoms B. In other words, the cardinality of $\{B \mid A \geq B\}$ must be finite for all ground atoms A.*
2. *There are finitely many odd-cyclic atoms in $DG(P)$.*

Two examples of finitary programs can be found in figures 1 and 2. Many further examples of interesting finitary programs, showing that the above definition is compatible with a rather free use of function symbols and recursion, can be found in [Bonatti 2001b,Bonatti 2002].

The consequences of finitary programs can be computed by using only a finite fragment of their (potentially infinite) domain.

Definition 5 (Kernel atoms, Relevant universe and subprogram). *A* kernel atom *for a normal program P and a ground formula Q is either an odd-cyclic atom or an atom occurring in Q (note that kernel atoms are ground by definition). The set of kernel atoms for P and Q is denoted by $K(P,Q)$.*

The relevant universe *for P and Q, denoted by $U(P,Q)$, is the set of all ground atoms B such that some kernel atom for P and Q depends on B. In symbols:*

$$U(P,Q) = \{B \mid \textit{for some } A \in K(P,Q),\ A \geq B\}\,.$$

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

$$R(P,Q) = \{R \mid R \in \mathsf{Ground}(P) \textit{ and } \textit{head}(R) \in U(P,Q)\}\,.$$

In [Bonatti 2001b] it is proven that if P is finitary, then $R(P,Q)$ is finite and computable. Moreover, $R(P,Q)$ is all that is needed for query answering:

Lemma 1. *For all finitary normal programs P and all ground formulae Q, $R(P,Q)$ has a stable model M_Q iff P has a stable model M such that $M \cap U(P,Q) = M_Q$.*

Theorem 2. *For all finitary normal programs P and all ground formulae Q,*

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

Since $R(P,Q)$ is finite, it follows that all the above reasoning tasks are decidable. Moreover, it can be shown that if Q is a quantifier-free nonground formula, then credulous and skeptical entailment are semi-decidable and Turing-equivalent [Bonatti 2001b].

3 Finitary open programs

In this section we extend the theory of finitary programs from normal logic programs to open programs.

Each standard normal logic program P can be regarded as a degenerate open program $\langle P, \emptyset, \emptyset \rangle$. Therefore, in order to inherit the properties of finitary programs, the first element of each open program $\langle P, F, O \rangle$ should at least be finitary. In the general case, given that finitary programs are defined by two conditions on their dependency graph, the question is: how should the dependency graph be constructed, and how should we take into account the function and constant symbols in F?

A very natural approach consists in extending the vocabulary of P with the symbols of F. In the following, $\mathsf{Ground}_F(P)$ denotes the ground instantiation of P w.r.t. the vocabulary of P extended with the symbols in F.

Definition 6. *The* (atom) dependency graph *of an* open *program* $\langle P, F, O \rangle$ *is a labelled directed graph denoted by* $DG_F(P)$*, whose vertices are the ground atoms of* P*'s language extended with the function (and constant) symbols in* F*. Moreover, (i) there exists an edge labelled '+' (called positive edge) from* B *to* A *iff for some rule* $R \in \mathsf{Ground}_F(P)$*,* $A \in head(R)$ *and* $B \in body(R)$*; (ii) there exists an edge labelled '−' (called negative edge) from* B *to* A *iff for some rule* $R \in \mathsf{Ground}_F(P)$*,* $A \in head(R)$ *and* $\neg B \in body(R)$*.*

We say that A depends *on* B *in* $DG_F(P)$ *if* A *and* B *are nodes of* $DG_F(P)$ *and there exists a path from* B *to* A *in* $DG_F(P)$*.*

A ground atom A *is* odd-cyclic *in* $DG_F(P)$ *if* A *belongs to a cycle in* $DG_F(P)$ *containing an odd number of negative edges.*

Now we have the building blocks for the definition of finitary open programs.

Definition 7 (Finitary open programs). *We say that an open program* $\langle P, F, O \rangle$ *is* finitary *if the following conditions hold:*

1. *Each ground atom* A *depends on finitely many ground atoms* B *in* $DG_F(P)$*.*
2. *There are finitely many odd-cyclic atoms in* $DG_F(P)$*.*

Let us see the implications of this definition on the embedding of open programs into ASP [Bonatti 2002b]. Recall that the normal logic program $\Pi(P, F, O)$ corresponding to an open program $\langle P, F, O \rangle$ is defined as follows. For each predicate symbol $p \in O$, introduce a new distinct predicate symbol $\bar{p}$. Moreover, let U, S and $\bar{S}$ be three new predicate symbols distinct from the symbols $\bar{p}$. $\Pi(P, F, O)$ consists of the following rules, for all rules $H \leftarrow Body$ in P, for all n-ary function symbols f occurring in P or F, and for all $p \in O$:

$$\begin{array}{ll}
H \leftarrow Body, U(x_1), \ldots, U(x_n) & \text{where the } x_i\text{s are the variables of } H \leftarrow Body \\
U(f(x_1, \ldots, x_n)) \leftarrow U(x_1), \ldots, U(x_n), S(f) & \\
 & \\
S(f) & \text{if } f \notin F \\
S(f) \leftarrow \neg\bar{S}(f) & \text{if } f \in F \\
\bar{S}(f) \leftarrow \neg S(f) & \text{if } f \in F \\
 & \\
p(x_1, \ldots, x_n) \leftarrow \neg\bar{p}(x_1, \ldots, x_n), U(x_1), \ldots, U(x_n) & \\
\bar{p}(x_1, \ldots, x_n) \leftarrow \neg p(x_1, \ldots, x_n), U(x_1), \ldots, U(x_n) &
\end{array}$$

Intuitively, each extension of U (in each stable model of the above program) corresponds to the Herbrand domain of a program $P' \in \mathsf{Comp}(P, F, O)$. By filtering away the new predicates and the terms not occuring in the extension of U, one obtains the stable models of the completions $P' \in \mathsf{Comp}(P, F, O)$:

Theorem 3. [Bonatti 2002b] *Let*

$$M|_{P,F,O} = \{\, p(t_1, \ldots, t_n) \in M \mid p \textit{ occurs in } P \cup O \textit{ and } U(t_i) \in M \ (1 \le i \le n) \,\}.$$

M is a stable model of $\Pi(P, F, O)$ iff there exist $P' \in \mathsf{Comp}(P, F, O)$ and a stable model M' of P' such that $M|_{P,F,O} = M'$.

Note that the set W consisting of all the ground atoms $U(t)$, $S(t)$ and $\bar{S}(t)$ occurring in $\mathsf{Ground}(\Pi(P,F,O))$ is a splitting set of P. The following properties hold.

Proposition 4. *The bottom (normal) program $b_W(\Pi(P,F,O))$ (consisting of the definitions of U, S and $\bar{S}$) is finitary.*

To see this, note that for all rules defining U, S and $\bar{S}$, each variable in the body occurs also in the head (so these predicates are finitely recursive), and there are no odd-cyclic atoms.

Proposition 5. *If $\langle P,F,O \rangle$ is a finitary open program, then the (normal) program $\Pi(P,F,O) \setminus b_W(\Pi(P,F,O))$ is finitary.*

From these propositions it follows that $\Pi(P,F,O)$ is finitary, through the following lemmata.

Lemma 2. *Let U be a splitting set for a normal program P. If $b_U(P)$ and $P \setminus b_U(P)$ are finitary, then P is finitary, too.*

Theorem 4. *If $\langle P,F,O \rangle$ is a finitary open program, then $\Pi(P,F,O)$ is a finitary normal program.*

This result has several immediate corollaries, that extend the theoretical results of [Bonatti 2001b] to open programs. Perhaps, the most important of such results are the following.

Theorem 5. *Let $\langle P,F,O \rangle$ be a finitary open program. For all ground formulae Q, $R(\Pi(P,F,O),Q)$ is finite and*

1. *$\langle P,F,O \rangle \models^c \exists Q$ iff $R(\Pi(P,F,O),Q)$ credulously entails Q.*
2. *$\langle P,F,O \rangle \models^s \exists Q$ iff $R(\Pi(P,F,O),Q)$ skeptically entails Q.*

Corollary 1. *Let $\langle P,F,O \rangle$ be a finite, finitary open program. For all quantifier-free formulae Q with no occurrences of the new predicates $\bar{p}$ ($p \in O$), U, and $\bar{U}$,*

1. *checking whether $\langle P,F,O \rangle \models^c \exists Q$ is decidable if Q is ground, and semi-decidable (r.e.-complete) otherwise.*
2. *checking whether $\langle P,F,O \rangle \models^s \exists Q$ is decidable if Q is ground, and semi-decidable (r.e.-complete) otherwise.*

These results make it possible to effectively compute credulous and skeptical open inference, even if P or F contain (occurrences of) function symbols. For example, one may first compute $R(\Pi(P,F,O),Q)$ and then feed it to any state-of-the-art ASP engine.

A second interesting extension of the results of [Bonatti 2001b] tells us that the two conditions in Definition 7 are "minimal", in the sense that if any of them were dropped, then the decidability and semi-decidability results would not hold. To prove this, we can use the same counterexamples illustrated in [Bonatti 2001b], and exploit the fact that each normal program P is equivalent to the degenerate open program $\langle P, \emptyset, \emptyset \rangle$.

Last but not least, the standard embedding of finitary open programs into normal programs enjoys the following compactness theorem (unlike unrestricted normal programs, default logic and autoepistemic logic).

Definition 8. *An* unstable kernel *for an open program $\langle P, F, O \rangle$ is a subset K of $\mathsf{Ground}_F(P)$ with the following properties:*

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

Theorem 6 (Compactness). *Let $\langle P, F, O \rangle$ be a finitary open program. Then $\Pi(P, F, O)$ has no stable models iff $\langle P, F, O \rangle$ has a finite unstable kernel.*

4 Dealing with infinite open programs

Recall that each open abduction framework $\langle T, A \rangle$ is captured by an infinite open program $\langle T, Sk, A \rangle$, where T is the domain theory, A is the set of abducible predicates, and Sk is an infinite set of (skolem) constants (see Section 2). The elements of Sk may be included into the domain of the completions of $\langle T, Sk, A \rangle$ to abduce the existence of new individuals.

Unfortunately, the standard embedding Π of open programs into normal logic programs in this case produces an infinite program. Since the current implementation techniques apply only to finite (nonground) finitary programs, here we show an alternative (finite) embedding for the class of open programs $\langle P, F, O \rangle$ such that F is an infinite set of constants with cardinality $\aleph_0$, while P and O are finite. Given the relationships with abduction, we call this class *abductive open programs.*

To simplify the presentation, we assume that P contains no function symbols (recall, however, that Sk is an infinite set of constants).[2] Each abductive open program $\langle P, F, O \rangle$ can be captured by a normal logic program $\Pi'(P, F, O)$ defined as follows. For each predicate symbol $p \in O$, introduce a new distinct predicate symbol $\bar{p}$. Moreover, let U, and $\bar{U}$ be two new predicate symbols distinct from the symbols $\bar{p}$, and let c and s, respectively, be a constant and a unary function not occurring in P. $\Pi'(P, F, O)$ consists of the following rules, for all rules $H \leftarrow Body$ in P, for all n-ary function symbols f occurring in P or F, and

[2] If P contains function symbols, then the definition of U must be slightly changed to avoid hybrid terms $s^k(t)$, where t contains symbols occurring in P.

for all $p \in O$:

$$H \leftarrow Body, U(x_1), \ldots, U(x_n) \quad \text{where } x_1 \ldots x_n \text{ are the variables of } H \leftarrow Body$$

$$\begin{array}{l}
U(c) \leftarrow \neg\bar{U}(c) \\
\bar{U}(c) \leftarrow \neg U(c) \\
U(s(X)) \leftarrow U(X), \neg\bar{U}(s(X)) \\
\bar{U}(s(X)) \leftarrow \neg U(s(X)) \\
p(x_1, \ldots, x_n) \leftarrow \neg\bar{p}(x_1, \ldots, x_n), U(x_1), \ldots, U(x_n) \\
\bar{p}(x_1, \ldots, x_n) \leftarrow \neg p(x_1, \ldots, x_n), U(x_1), \ldots, U(x_n)
\end{array}$$

Clearly, $\Pi'(P, F, O)$ is finite, as required.

Note that the possible extensions of predicate U under the stable model semantics are the initial segments of the infinite sequence $\Sigma = c, s(c), s(s(c)), \ldots$ (including the empty sequence and Σ).

Therefore, each subset of F is isomorphic to some extension of U, in some stable model of $\Pi'(P, F, O)$. Then it can be proved that the models of the new embedding correspond faithfully to the stable models of the completed programs in $\mathsf{Comp}(P, F, O)$. More precisely, the new atoms must be filtered away, and each new term $s^k(c)$ must be matched with a suitable element of F.

Let $M|_{P,F,O}$ be defined as in Theorem 3. Let a *renaming function* over F be an injective substitution, preserving the symbols in P and O, and mapping the new elements occurring in Σ onto elements of F.

Theorem 7. *M is a stable model of $\Pi'(P, F, O)$ iff there exist $P' \in \mathsf{Comp}(P, F, O)$, a stable model M' of P' and a renaming function ρ such that*

- *the image of ρ restricted to $M(U)$ (the extension of U in M) equals the set of elements of F actually occurring in P'.*
- *$\rho(M|_{P,F,O}) = M'$.*

This proves that the new embedding can be used to compute open inference from abductive open programs.

Next we prove that if an abductive open program is finitary, then also the corresponding normal program (under the new embedding) is finitary.

Proposition 6. *Let W' be the set of ground atoms $U(t)$ and $\bar{U}(t)$ occurring in $\mathsf{Ground}(\Pi'(P, F, O))$.*

1. *W' is a splitting set for $\Pi'(P, F, O)$.*
2. *$b_{W'}(\Pi'(P, F, O))$ is finitary.*
3. *If $\langle P, F, O \rangle$ is a finitary open program, then $\Pi'(P, F, O) \setminus b_{W'}(\Pi'(P, F, O))$ is finitary.*

From the above proposition and Lemma 2 we have:

Theorem 8. *If $\langle P, F, O \rangle$ is a finitary open program then $\Pi'(P, F, O)$ is a finitary normal program.*

An interesting corollary of Theorem 7 relates the credulous and the skeptical consequences of the new embedding to their open counterparts.

Corollary 2. *Let $\langle P, F, O \rangle$ be an abductive open program. For all sentences Q with no occurrences of the new symbols $\bar{p}$ ($p \in O$), U, $\bar{U}$, c, and s,*

1. *$\langle P, F, O \rangle \models^c Q$ iff Q is a credulous consequence of $\Pi'(P, F, O)$.*
2. *$\langle P, F, O \rangle \models^s Q$ iff Q is a skeptical consequence of $\Pi'(P, F, O)$.*

As a consequence of this corollary, Theorem 8 and Theorem 2, we have that if $\langle P, F, O \rangle$ is finitary, then the above inferences are computable.

Corollary 3. *Let $\langle P, F, O \rangle$ be a finitary, abductive (infinite) open program. For all quantifier-free formulae Q with no occurrences of the new symbols $\bar{p}$ ($p \in O$), U, $\bar{U}$, c, and s,*

1. *checking whether $\langle P, F, O \rangle \models^c \exists Q$ is decidable if Q is ground, and semi-decidable (r.e.-complete) otherwise.*
2. *checking whether $\langle P, F, O \rangle \models^s \exists Q$ is decidable if Q is ground, and semi-decidable (r.e.-complete) otherwise.*

We conclude this section with two examples of standard abduction frameworks corresponding to finitary open programs with unbounded domains. The first domain theory illustrated in Figure 1 is a finitary program for model-based circuit diagnosis. Each atom `out(G)` models the output of gate `G`. The values of circuit inputs are specifed through the same predicate. For example the fact `out(y2)` states that the value of input `y2` is 1. The rules model the behavior of each gate, both under the assumption that the gate is working properly ($\neg\mathtt{ab}(G)$), and under the assumption that a fault exists ($\mathtt{ab}(G)$). This formalization makes the common assumption that in the absence of observable faults gates are not faulty. This program is stratified, so there are no odd-cycles. The set of abducible predicates is $A = \{\mathtt{ab}\}$.

The second domain theory, illustrated in Figure 2, can be used to find sequences of events that explain sequences of observations by abducing the predicate `do`. Most atoms with time argument $\mathtt{T}+1$ depend on atoms with time argument `T`. So the only source of cycles with negative edges are the rules for nondeterministic actions (predicates `fails` and `succeeds`). However, these are not odd-cycles.

An open version of this example (where the existence of new individuals can be abduced) can produce explanations that exploit blocks not explicitly mentioned in the domain theory.

Note the advantages of using function symbols and recursion. Infinite and structured domains (such as the set of all gates and time) can be modelled directly, in a natural way. An infinite number of problem instances can be encoded in one program, so there is no need for external components to build a specific encoding for each instance. More generally, all necessary pre- and post-processing can be carried out within the same language, and in principle such auxiliary computations can be interleaved naturally with proper problem solving activities. Once all instances are simultaneously encoded in one program, one can submit queries across (and relating) multiple instances. For further details on these topics, see [Bonatti 2001b,Bonatti 2002].

```
out(and(X, Y)) ← out(X), out(Y), ¬ab(and(X, Y))
out(and(X, Y)) ← ab(and(X, Y)), ¬out(X)
out(and(X, Y)) ← ab(and(X, Y)), ¬out(Y)

out(or(X, Y)) ← out(X), ¬ab(or(X, Y))
out(or(X, Y)) ← out(Y), ¬ab(or(X, Y))
out(or(X, Y)) ← ab(or(X, Y)), ¬out(X), ¬out(Y)

out(not(X)) ← ¬out(X), ¬ab(not(X))
out(not(X)) ← ab(not(X)), out(X)

out(y2)    /* other inputs implicitly negated */
```

Fig. 1. Model based diagnosis

5 Conclusions

A natural generalization of the notion of finitary programs captures a very expressive class of open programs where skeptical and credulous open consequences can be effectively computed, even if such programs allow a rather flexible use of function symbols and recursion.

As this work is still in progress, a number of things remain to be done. In a forthcoming paper we will show how the finitary program recognizer can be adapted to handle finitary open programs. As far as traditional abduction frameworks are concerned, no modification is needed. Moreover, we shall illustrate in detail how to handle effectively finite explanations in the presence of unbounded domains.

Then we are going to study how to reason with open programs where F is infinite *and* contains function symbols.

It is also interesting to see whether the current definition of finitary open programs can be relaxed in some way. A related question is what relations hold

```
/* Frame axiom */
holds(P, T + 1) ← holds(P, T), ¬ab(P, T)

/* Sample deterministic action */
holds(on_top(A, B), T + 1) ←
                    do(put_on(A, B), T),       /* action */
                    holds(is_clear(B), T), /* preconds */
                    holds(in_hand(A), T)

/* Sample nondeterministic action */
holds(in_hand(B), T + 1) ←
                    do(grasp(B), T),           /* action */
                    holds(is_clear(B), T), /* preconds */
                    ¬fails(grasp(B), T)

holds(on_table(B), T + 1) ←
                    do(grasp(B), T),           /* action */
                    holds(is_clear(B), T), /* preconds */
                    fails(grasp(B), T)

ab(on_top(B, C), T) ←
                    do(grasp(B), T),           /* action */
                    holds(is_clear(B), T)  /* preconds */

fails(grasp(B), T) ← ¬succeeds(grasp(B), T)
succeeds(grasp(B), T) ← ¬fails(grasp(B), T)
```

Fig. 2. Reasoning about actions

between the cycles in $DG(P)$ and those in $DG_F(P)$. Concerning the first condition in Definition 7, a relaxation technique based on *local variable binding by recursive domain predicates* will be illustrated in an extended version of the paper.

The complexity of reasoning with finitary open programs is still an open issue. Moreover, we do not know yet how to handle mixed open inference in the presence of infinite domains.

Acknowledgements The work reported in this paper is partially supported by the European Community within the Fifth (EC) Framework Programme under contract IST-FET-2001-37004 – WASP working group (on answer set programming).

References

[Bonatti 2002] P. A. Bonatti. Reasoning with infinite stable models II: Disjunctive programs. *Proc. of ICLP'02*, LNCS 2401, 333-346, Springer, 2002.

[Bonatti 2002b] P. A. Bonatti. Abduction, ASP and open logic programs. *Proc. of NMR'02*, Toulouse, 2002.

[Bonatti 2001] P. A. Bonatti. Reasoning with open programs. *Proc. of LPNMR'01*, pp. 147-159, LNAI 2173, Springer Verlag, 2001.

[Bonatti 2001b] P. A. Bonatti. Reasoning with infinite stable models. *Proc. of IJCAI*, Morgan Kaufmann, 2001.

[Bonatti et al. 2000] P.A. Bonatti, S. De Capitani Di Vimercate, P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security, 5(1), 2002*, to appear. Preliminary version in Proc. of the 7th ACM Conference on Computer and Communication Security, CCS'2000, Atene, 2000.

[Denecker and De Schreye 1998] D. Denecker, D. De Schreye. SLDNFA: An abductive procedure for abductive logic programs. *The Journal of Logic Programming*, 34(2):111-167, 1998.

[Eshghi and Kowalski 1989] K. Eshghi, R.A. Kowalski. Abduction compared with negation as failure. In *Proc. of the 6th Int.l Conf. on Logic Programming*, MIT Press, 1989.

[Gelfond and Lifschitz 1988] M. Gelfond, V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th ICLP*, pp.1070-1080, MIT Press, 1988.

[Kakas et al. 1992] A.C. Kakas, R.A. Kowalski, F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719-770, 1992.

[Kakas and Mancarella 1990] A.C. Kakas, P. Mancarella. Generalized Stable Models: a semantics for abduction. *Proc. of ECAI'90*, pp.385-391, 1990.

[Lifschitz and Turner 1994] V. Lifschitz, H. Turner. Splitting a logic program. In *Proc. ICLP'94*, pp.23-37, MIT Press, 1994.

[Niemelä and Simons 1997] I. Niemelä, P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach, A. Nerode (eds.), *Logic Programming and Nonmonotonic Reasoning: 4th international conference, LPNMR'97*, LNAI 1265, Springer Verlag, Berlin, 1997.

[Poole 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27-47, 1988.

[Satoh and Iwayama 1992] K. Satoh, N. Iwayama. A query evaluation method for abductive logic programming. In *Proc. of the Joint Int.l Conf. and Symposium on Logic Programming* (JICSLP'92), 671-685, MIT Press, 1992.

[Satoh and Iwayama 1991] K. Satoh, N. Iwayama. Computing abduction by using the TMS. In *Proc. of the Int.l Conf. on Logic Programming* (JICSLP'92), 504-ff, MIT Press, 1991.

[Shanahan 1989] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of IJCAI'89*, 1055-ff, 1989.

[Subrahmanian et al. 2000] V.S. Subrahmanian, P.A. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, R. Ross. *Heterogeneous Active Agents.* MIT Press, 2000.