Credulous Resolution for Answer Set Programming[*]

Piero A. Bonatti
Dip. di Scienze Fisiche, Sezione di Informatica
Università di Napoli Federico II
Complesso Universitario di Monte S. Angelo
Via Cinthia, I-80126 Napoli, Italy
bonatti@na.infn.it

Enrico Pontelli and **Tran Cao Son**
Department of Computer Science
New Mexico State University
MSC CS, PO Box 30001
Las Cruces, NM 88003, USA
epontell|tson@cs.nmsu.edu

Abstract

The paper presents a calculus based on resolution for credulous reasoning in Answer Set Programming. The new approach allows a top-down and goal directed resolution, in the same spirit as traditional SLD-resolution. The proposed credulous resolution can be used in query-answering with non-ground queries and with non-ground, and possibly infinite, programs. Soundness and completeness results for the resolution procedure are proved for large classes of logic programs. The resolution procedure is also extended to handle some traditional syntactic extensions used in Answer Set Programming, such as choice rules and constraints. The paper also describes an initial implementation of a system for credulous reasoning in Answer Set Programming.

Introduction

Answer Set Programming (ASP) (Niemelä 1999; Marek & Truszczyński 1999) emerges from research in logic programming and non-monotonic reasoning. Using ASP, one can solve a problem by *(i)* encoding the problem as a logic program P; *(ii)* computing an answer set of P using an answer set solver, such as SMODELS (Simons *et al.* 2002) and DLV (Eiter *et al.* 2003a)); and *(iii)* extracting the solution from the answer set. The key idea is to ensure that the answer sets of the program developed in step *(i)* correspond one-to-one to solutions of the original problem.

An important observation is that, frequently, an *arbitrary* solution to the problem is required—or just the knowledge that a solution exists. For example, one coloring in a graph coloring problem or one assignment of variables satisfying a SAT instance might be enough. In ASP, this translates to the interest in performing *credulous* reasoning, i.e., exploring entailment w.r.t. an arbitrary answer set of a program.

ASP has been applied to several important areas—e.g., diagnosis for the space shuttle (Balduccini *et al.* 2001); LTL model checking (Heljanko & Niemelä 2003); security protocols verification (Carlucci *et al.* 2001). The improvements of ASP solvers with respect to the representation languages and implementation techniques enable ASP to be competitive to other state-of-the-art technologies in several domains (e.g., (Dovier *et al.* 2008)).

One of the main difficulties in applying ASP to real-world applications lies in the requirement that the input program provided to the solver is propositional (i.e., ground). Programs with variables need to be grounded before their answer sets can be computed, and their grounding has to result in a finite collection of rules. This limits our ability to use ASP in several practical domains and/or makes the development of ASP programs cumbersome. For example:

- We are prevented from directly using popular recursive predicate definitions, such as the various list predicates (`member`, `append`, `length`, etc.) available in Prolog.
- We cannot deal with many ASP programs which have a finitely representable answer set or a partial answer set containing sufficient information to address the user's query. For example, the program $\{P(A) \leftarrow;\, P(F(x)) \leftarrow Q(x)\}$ has a unique answer set $\{P(A)\}$; the program $\{P(A) \leftarrow;\, P(F(x)) \leftarrow P(x)\}$ has a partial answer set $\{P(A), P(F(A))\}$ that allows one to conclude that $P(F(A))$ is credulously entailed by the program. Both programs cannot be handled by current answer set solvers.
- We cannot deal with programs whose ground versions are very large; for example, the encoding of a typical planning problem in ASP often contains several hundreds of actions and may require a large number of steps to reach the goal, leading to ground instances with several hundreds of thousands of rules (Son & Pontelli 2007).

Extensive research has been conducted to address the above limitations. For example, the work in (Eiter *et al.* 1997) investigates a compact representation of answer sets for non-ground programs. Recently, a direct definition of answer sets for non-ground logic programs has been proposed (Ferraris *et al.* 2007). However, implementation following these definitions has yet to be developed. The work of (Bonatti 2001) addresses the issue of reasoning with non-ground and possibly infinite programs. It presents a sound and complete algorithm for *skeptical reasoning* (*skeptical resolution*) that works for a large class of non-ground ASP programs, called *finitary programs*. A skeptical derivation allows goal-directed reasoning; at each step, a pair of a goal and a set of hypotheses is transformed into a new pair of a new goal and hypotheses according to some inference rules. The process can either fail or succeed. When a derivation suc-

[*]This work has been supported by NSF grants 0420407 and 0220590.

ceeds, its initial goal is considered a skeptical consequence of the program given the initial set of hypotheses.[1] While applicable to several domains (e.g., security protocol verification (Bonatti 2001)), skeptical reasoning is not suited for applications favoring credulous reasoning, e.g., combinatorial problems and planning. Moreover, skeptical resolution in (Bonatti 2001) allows for reasoning by contradiction, which admits the answer *'yes'* when the program is inconsistent. This is not appropriate for those ASP applications where a negative answer should be given if the program is inconsistent. Other calculi (based on tableaux) have been presented in the literature (e.g., (Fitting 1994; Gebser & Schaub 2006)).

The goal of this paper is to develop a calculus for credulous resolution for non-ground and/or infinite logic programs. The calculus should not use the contradiction rule and should be sound and complete for large useful classes of infinite and/or non-ground ASP programs. Proofs of the results can be found in (Bonatti *et al.* 2008).

Preliminaries

Let $\mathcal{L}$ be a first order language. A positive literal is an atom in $\mathcal{L}$. A negative literal is of the form $\mathtt{not}\,A$, where A is an atom. A *literal* is a positive or a negative literal. A *normal logic program* is a set of rules of the form $A \leftarrow L_1, \ldots, L_n$ where A is an atom and each L_i is a literal. A is called the *head* and $L_1, \ldots, L_n$ the *body* of the rule. We use the notation $\mathsf{head}(r) = \{A\}$ and $\mathsf{body}(r) = \{L_1, \ldots, L_n\}$. Given a set of literals S, let $\mathsf{pos}(S)$ be the set of positive literals in S, and $\mathsf{neg}(S)$ be the set of atoms A such that $\mathtt{not}\,A \in S$. These functions are extended to rules as follows: $\mathsf{pos}(r) = \mathsf{pos}(\mathsf{body}(r))$ and $\mathsf{neg}(r) = \mathsf{neg}(\mathsf{body}(r))$. A set of ground atoms M satisfies a ground atom A (resp. a ground negative literal $\mathtt{not}\,A$) if $A \in M$ (resp. $A \notin M$). This is denoted by $M \models A$ (resp. $M \models \mathtt{not}\,A$). $M \models S$ for a set of ground literals S iff $M \models A$ for every $A \in S$.

A *ground* rule is a ground instantiation of some rule. $\mathsf{Ground}(P)$ denotes the set of all ground rules of the program P. The *Gelfond-Lifschitz transformation* P^I of a program P w.r.t. an Herbrand interpretation I is obtained by removing from $\mathsf{Ground}(P)$ all the rules containing a negative literal $\mathtt{not}\,B$ such that $B \in I$, and by removing from the remaining rules all negative literals. An interpretation M is an *answer set* of P if M is the minimal Herbrand model of P^M (Gelfond & Lifschitz 1988).

The *atom dependency graph* of a program P is a labeled directed graph $DG(P)$ whose vertices are the ground atoms from the language of P. $DG(P)$ contains an edge labeled '+' (a *positive* edge) from A to B iff there is a rule $r \in \mathsf{Ground}(P)$ such that $A \in \mathsf{head}(r)$ and $B \in \mathsf{pos}(r)$. $DG(P)$ contains an edge labeled '-' (a *negative* edge) from A to B iff there is $r \in \mathsf{Ground}(P)$ such that $A \in \mathsf{head}(r)$ and $\mathtt{not}\,B \in \mathsf{neg}(r)$.

An atom A *depends positively* (resp. *negatively*) on B if there is a path from A to B in $DG(P)$ 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 $A \geq B$ to denote that A depends positively or negatively on B. If both $A \geq_+ B$ and $A \geq_- B$ hold then we write $A \geq_\pm B$. A program is *order consistent* if there are no infinite chains $A_1 \geq_\pm A_2 \geq_\pm \ldots$ Order consistent programs always have an answer set. An *odd-cycle* is a cycle in $DG(P)$ containing an odd number of negative edges. Note that order consistency implies that there are no odd-cycles.

A program is *finitely recursive* iff each ground atom depends on finitely many ground atoms. A program is *finitary* if it is finitely recursive and has finitely many odd-cycles.

SLD derivations are defined as in (Lloyd 1987), assuming that a computation rule that does not select negative literals is used. The following definitions are from (Bonatti 2001).

Definition 1. *A* support *for a ground atom A (from P) is a set of ground negative literals $\{L_1, \ldots, L_n\}$ such that there exists an SLD derivation of A from* $\mathsf{Ground}(P)$ *whose last resolvent is $L_1, \ldots, L_n$.*

Definition 2. *A* ground countersupport *for a ground atom A (from P) is a set of atoms C such that:* (1) *For each support S of A there exists $B \in C$ such that* $\mathtt{not}\,B \in S$*;* (2) *For each $B \in C$ there exists a support S of A such that* $\mathtt{not}\,B \in S$*.*

Example 1. The Yale Shooting Problem can be represented by the following program (in all the rules, $t \geq 0$ represents a time step; H and O mean $Holds$ and $Occurs$):

$$H(\neg Loaded, 0) \leftarrow \tag{1}$$
$$H(Alive, 0) \leftarrow \tag{2}$$
$$H(\neg Alive, t+1) \leftarrow O(Shoot, t), H(Loaded, t) \tag{3}$$
$$H(\neg Loaded, t+1) \leftarrow O(Shoot, t) \tag{4}$$
$$H(Loaded, t+1) \leftarrow O(Load, t) \tag{5}$$
$$H(f, t+1) \leftarrow H(f, t), \mathtt{not}\,H(\neg f, t+1) \tag{6}$$
$$H(\neg f, t+1) \leftarrow H(\neg f, t), \mathtt{not}\,H(f, t+1) \tag{7}$$

where the last two rules encode the frame axiom for every fluent $f \in \{Loaded, Alive\}$. This program can be used for hypothetical reasoning. For planning, we will add the rules:[2]

$$O(Shoot, t) \leftarrow \mathtt{not}\,O(Load, t) \tag{8}$$
$$O(Load, t) \leftarrow \mathtt{not}\,O(Shoot, t) \tag{9}$$

Let us denote with P_{YST} the above program. It is easy to see that $\{\mathtt{not}\,O(Load, t)\}$ is a support for $O(Shoot, t)$ and $\{O(Load, t)\}$ is a countersupport for $O(Shoot, t)$. $\triangle$

The following theorem (Bonatti 2001) relates the notions of support and countersupport to answer sets of a program.

Theorem 1. *If M is an answer set and A is a ground atom, then:* (1) *$A \in M$ iff A has a support S s.t. $M \models S$;* (2) *$A \notin M$ iff A has a ground countersupport C s.t. $M \models C$.*

Example 2. The program P_{YST} has an answer set A which contains $\{O(Load, 0), O(Shoot, 1), H(\neg Alive, 2)\}$ and does not contain $\{O(Shoot, 0), O(Load, 1)\}$. In this answer

[1]When the initial set of hypotheses is empty, the goal is a consequence of the program.

[2]We have simplified the action generation rules in the examples.

set, $O(Load, 0) \in A$ and $A \models \{\texttt{not}\, O(Shoot, 0)\}$, a support for $O(Load, 0)$. Likewise, $O(Shoot, 0) \notin A$ and $A \models \{O(Load, 0)\}$, a countersupport of $O(Shoot, 0)$. A support for $H(\neg Alive, 2)$ is $\{\texttt{not}\, O(Shoot, 0), \texttt{not}\, O(Load, 1)\}$. Thus, $H(\neg Alive, 2)$ is credulously entailed by P_{YST}.

The program admits another answer set which contains $\{O(Load, 0), O(Load, 1), H(Alive, 2)\}$ and does not contain $H(\neg Alive, 2)$. In other words, P_{YST} credulously entails $H(\neg Alive, 2)$ but it does not skeptically entail it. $\triangle$

Credulous Resolution

In this section, we define a calculus for credulous reasoning in ASP. Like the skeptical resolution of (Bonatti 2001), credulous resolution also works with *goals with hypotheses* (or *h-goals*). Each h-goal is a pair $(G \mid H)$, where G is a standard goal, i.e., a finite set of literals, and H is a set of negative literals. Intuitively, a h-goal $(G \mid H)$ asks whether G is true if H were assumed to be true—i.e., whether there is an answer set M which satisfies both G and H. We use $\Box$ to denote an empty goal, satisfied by any answer set.

Example 3. *The pair $(\{H(\neg Alive, 2)\} \mid \Box)$ is a h-goal for the program P_{YST}, which asks whether there exists an answer set of the program containing $H(\neg Alive, 2)$.*

We will also need the notion of a *derivation* which is a sequence of h-goals $\Delta = (G_0 \mid H_0)\ (G_1 \mid H_1) \dots (G_n \mid H_n) \dots$ (or $\Delta = \langle (G_i \mid H_i) \rangle_{i=0}^{k}$, for short). We will next present the rules for credulous reasoning. Since these rules are complex in the general case (non-ground programs), we will specialize them for different classes of logic programs. We start with the case of ground normal programs.

Ground Normal Programs

Let P be a program and $(G \mid H)$ be a h-goal. Theorem 1 indicates that if P has an answer set M which satisfies both G and H then we can create a new h-goal $(G' \mid H')$, which is also satisfied by M, by replacing elements in G and H by one of their supports or countersupports. This process can be repeated and creates a *ground credulous derivation*.

Definition 3. *A* ground credulous derivation *(*c-derivation*) of a ground h-goal $(G \mid H)$ from P is a (possibly infinite) derivation $\Delta = \langle (G_i \mid H_i) \rangle_{i=0}^{k}$ with $G_0 = G$, $H_0 = H$, and some of the following conditions hold for every i:*

- *(**R**) (Resolution rule) for some positive literal A in G_i and some rule $r \in \mathsf{Ground}(P)$ such that $\mathsf{head}(r) = A$ we have that $G_{i+1} = (G_i \setminus \{A\}) \cup \mathsf{body}(r)$ and $H_{i+1} = H_i$*
- *(**H**) (Resolution with hypothesis) for some negative literal $\texttt{not}\, A \in G_i \cap H_i$ we have that $G_{i+1} = G_i \setminus \{\texttt{not}\, A\}$ and $H_{i+1} = H_i$*
- *(**F**) (Failure rule) for some negative literal $\texttt{not}\, A$ in $G_i \setminus H_i$ and some ground countersupport C_A for A (from P), we have that $G_{i+1} = (G_i \setminus \{\texttt{not}\, A\}) \cup C_A$ and $H_{i+1} = H_i \cup \{\texttt{not}\, A\}$*

Intuitively, a c-derivation represents a top-down computation, starting with a goal $(G \mid H)$. The first element of the derivation is the h-goal $(G \mid H)$ which needs to be solved, as in other proof procedures (e.g., traditional SLD resolution). At each step of the derivation, one of the three rules (**R**), (**H**), or (**F**), is used to transform the h-goal into a new h-goal. We illustrate some computations in the Table 1.

$(G_i \mid H_i)$	Rule applied
$(\{(H(\neg Alive, 2)\} \mid \Box)$	(**R**), rule (3), $t = 1$
$(\{O(Shoot, 1), H(Loaded, 1)\} \mid \Box)$	(**R**), rule (8), $t = 1$
$(\{\texttt{not}\, O(Load, 1), H(Loaded, 1)\} \mid \Box)$	(**F**), $\texttt{not}\, O(Load, 1)$
$(\{O(Shoot, 1), H(Loaded, 1)\} \mid \{\texttt{not}\, O(Load, 1)\})$	(**R**), rule (8), $t = 1$
$(\{\texttt{not}\, O(Load, 1), H(Loaded, 1)\} \mid \{\texttt{not}\, O(Load, 1)\})$	(**H**)
$(\{H(Loaded, 1)\} \mid \{\texttt{not}\, O(Load, 1)\})$	

Table 1: Computation steps w.r.t. P_{YST}

A c-derivation is *consistent* iff for all pair of h-goals $(G_i \mid H_i)$ and $(G_j \mid H_j)$ in the sequence, $\mathsf{pos}(G_i) \cap \mathsf{neg}(G_j) = \emptyset$. A c-derivation is *successful* if it is consistent, finite, and the last h-goal has the form $(\Box \mid H)$, for some set of negative literals H. The following theorem shows that credulous resolution is sound for order-consistent programs while completeness is guaranteed for finitely recursive programs.

Theorem 2 (Ground Soundness and Completeness)**.**

- ○ *Let P be a ground, order consistent program. If $(G \mid \Box)$ has a successful c-derivation from P, then G is a credulous consequence of P.*
- ○ *Let P be a ground finitely recursive program. If G is a credulous consequence of P, then $(G \mid \Box)$ has a successful c-derivation from P.*

Observe that finitely recursive, odd-cycle free programs are order consistent but not vice versa. It is also worth mentioning that the class of finitely recursive, odd-cycle free programs is large enough to cover many interesting predicates often used in ASP (Bonatti 2004).

Continuing with the derivation in Table 1, we have:

$(G_i \mid H_i)$	Rule applied
$(\{H(Loaded, 1)\} \mid \{\texttt{not}\, O(Load, 1)\})$	(**R**), (5), $t = 0$
$(\{O(Load, 0)\} \mid \{\texttt{not}\, O(Load, 1)\})$	(**R**), (9), $t = 0$
$(\{\texttt{not}\, O(Shoot, 0)\} \mid \{\texttt{not}\, O(Load, 1)\})$	(**F**)
$(\{O(Load, 0)\} \mid \{\texttt{not}\, O(Shoot, 0), \texttt{not}\, O(Load, 1)\})$	repeat 2nd step then apply (**H**)
$(\Box \mid \{\texttt{not}\, O(Shoot, 0), \texttt{not}\, O(Load, 1)\})$	

Table 2: A successful c-derivation for $(\{(H(\neg Alive, 2)\} \mid \Box)$

Non-Ground Programs

Credulous resolution can be naturally extended to non-ground programs and goals, similarly to the case of skeptical resolution. Rule (**R**) (resolution) is extended classically: positive literals are *unified* with rule heads, and the resulting most general unifier is applied to the new h-goal. Negation as failure is based on a non-ground version of countersupport. A *non-ground countersupport* for a (possibly non-ground) atom A from a program P is a pair (θ, C), where θ is a substitution and C a set of atoms such that, for all grounding substitutions γ of $A\theta$, $C\theta\gamma$ is a ground countersupport for $A\theta\gamma$ from P.

A *non-ground derivation* from P is a sequence $\Gamma = \langle (G_i \mid H_i \mid \theta_i) \rangle_{i=0}^{k}$ where G_i, H_i are (possibly non-ground) sets of literals, and θ_i is a substitution. We call a triple $(G_i \mid H_i \mid \theta_i)$ an *extended h-goal.*

A non-ground *credulous derivation* (ngc-derivation) of a (possibly non-ground) h-goal $(G \mid H)$ from P is a non-ground derivation $\Delta = \langle (G_i \mid H_i \mid \theta_i) \rangle_{i=0}^{k}$ with $G = G_0$, $H = H_0, \theta_0 = \epsilon$, and for all extended h-goals $(G_i \mid H_i \mid \theta_i)$ in Δ some of the following conditions hold:

(R) (Resolution rule) for some positive literal A in G_i and a new variant r of some rule in P, μ is the most general unifier (mgu) of $\mathsf{head}(r)$ and $A\theta_i$, $G_{i+1} = (G_i \setminus \{A\}) \cup \mathsf{body}(r)$, $H_{i+1} = H_i \cup \{A\}$, and $\theta_{i+1} = \theta_i \mu$.[3]

(H) (Resolution with hypothesis) for some negative literal $\mathtt{not}\, A$ in G_i and some $\mathtt{not}\, B$ in H_i such that μ is the mgu of $\mathtt{not}\, A\theta_i$ and $\mathtt{not}\, B\theta_i$, then $G_{i+1} = G_i \setminus \{\mathtt{not}\, A\}$, $H_{i+1} = H_i$, and $\theta_{i+1} = \theta_i \mu$.

(F) (Failure) for a literal $\mathtt{not}\, A$ in G_i and some non-ground countersupport (σ, C_A) for $A\theta_i$, we have $G_{i+1} = G_i \setminus \{\mathtt{not}\, A\} \cup C_A$, $H_{i+1} = H_i \cup \{\mathtt{not}\, A\}$, and $\theta_{i+1} = \theta_i \sigma$.

A derivation is finite if, after k steps, it reaches an extended h-goal $(\Box \mid H_k \mid \theta_k)$. In this case, σ is a *computed answer substitution* if there exists a substitution γ such that $\sigma = \theta_i \gamma$ and $\mathsf{Ground}(H_k \sigma)$ is a consistent set of ground literals.

A ngc-derivation $\Delta = \langle G_i \mid H_i \mid \theta_i \rangle_{i=0}^{k}$, is *successful* if it is finite and admits at least one computed answer substitution. Theorem 2 can be extended to the non-ground case:

Theorem 3 (Non-ground Soundness and Completeness)**.**

- *Let P be a (possibly non-ground) program such that* $\mathsf{Ground}(P)$ *is order consistent, and let Δ be a successful ngc-derivation of $(G \mid \Box)$ from P, where θ is a corresponding computed answer substitution. Then there exists an answer set M of* $\mathsf{Ground}(P)$ *such that $M \models \forall(G\theta)$.*
- *Let P be a finitely recursive, odd-cycle-free program. If G is a ground credulous consequences of P and G' is such that $G = G'\gamma$ for some substitution γ, then there is a successful ngc-derivation of $(G' \mid \Box)$ from P and a corresponding computed answer substitution θ such that $G = G'\theta\sigma$ for some substitution σ.*

Specializations of Credulous Resolution

The class of finitely recursive, odd-cycle-free programs is large enough for us to consider several programs. However, odd-cycles often appear in ASP programs when choice atoms and/or constraints (Simons *et al.* 2002) are used. E.g., if the theory in Example 1 has an additional action, say $Wait$, then the group of rules for action generation (8)-(9) should be changed to the two rules

$$O(a,t) \leftarrow \mathtt{not}\, No(a,t) \qquad\qquad No(a,t) \leftarrow \mathtt{not}\, O(a,t)$$

and the set of constraints

$$\leftarrow \quad \mathtt{not}\, No(Shoot,t), \mathtt{not}\, No(Load,t), \mathtt{not}\, No(Wait,t) \quad (10)$$
$$\leftarrow \quad a \neq b, O(a,t), O(b,t) \quad (11)$$

where a is an action variable and *No* means "not occurs." However, each constraint represents an odd-cycle. As discussed in (Bonatti 2004), odd-cycles caused by constraints can be dealt with by introducing new atoms. E.g., we can introduce a new predicate $Bad(t)$ and rewrite (10) as

$$\begin{array}{rcl} Bad(t) & \leftarrow & \mathtt{not}\, No(Shoot,t), \mathtt{not}\, No(Load,t), \\ & & \mathtt{not}\, No(Wait,t) \end{array} \quad (12)$$

together with the rule: $Bad(t+1) \leftarrow Bad(t)$.

The program with the predicate Bad does not have odd-cycles if the set of rules without constraints creates a program without odd-cycles. Furthermore, answer sets of the original programs are exactly those which do not contain any atom of the form $Bad(t)$. Thus, to discover whether G is a credulous consequence of the original program, we need to find a successful credulous derivation of the goal $G, \mathtt{not}\, Bad(K)$ where K is an appropriate constant, which can be determined given G; for example, if $G = H(\neg Alive, 2)$ then $K = 2$.

This shows that credulous resolution can be applied to programs with constraints. The disadvantage of this method is the exponential number of countersupports for $Bad(T)$ as T grows. This is not desirable for an implementation of credulous resolution. In this section, we will present two specializations of the credulous resolution for programs with choice atoms and constraints. For simplicity of the presentation, we will focus on ground (possibly infinite) programs.

Programs with Choice Atoms

Choice atoms have been introduced in (Simons *et al.* 2002) and have proved to be very useful in knowledge representation. In our example, we can replace the rules for selecting an action with the single choice atom (T is a time point)

$$1\{O(a,T) : Action(a)\}1 \leftarrow$$

Formally, a *choice atom* is of the form $L\{A_1, \ldots, A_n\}U$ where each A_i is a ground atom[4], $n \geq 1$, and L and U are non-negative integers, $L \leq U$. An atom A is called a *choice-literal* if there exists a choice atom $L\{A_1, \ldots, A_n\}U$ and $A \in \{A_1, \ldots, A_n\}$. A *choice rule* is of the form

$$L\{A_1, \ldots, A_n\}U \leftarrow L_1, \ldots, L_m \quad (m \geq 0) \quad (13)$$

where $L\{A_1, \ldots, A_n\}U$ is a choice atom and each L_i is a literal. For simplicity, in the rest of the discussion we will assume $L = U = 1$. The techniques used in this paper can be easily adapted to the general case.

A *program with choice rules* (or *c-program*) is a set containing both standard and choice rules. The notion of $\mathsf{Ground}(P)$ is extended to program with choice rules in the usual way. The atom dependency graph $DG^c(P)$ is defined as follows. The set of nodes of $DG^c(P)$ is the set of atoms occurring in P. There is a positive edge from A to B if there is a ground rule r in $\mathsf{Ground}(P)$ such that $B \in \mathsf{pos}(r)$ and A occurs in $\mathsf{head}(r)$. There is a negative edge from A to B if there is a ground rule r in $\mathsf{Ground}(P)$ such that $B \in \mathsf{neg}(R)$ and A occurs in $\mathsf{head}(R)$.

Definition 4. *A c-program P is* finitary *if* (1) *P is finitely recursive w.r.t. $DG^c(P)$,* (2) *$DG^c(P)$ is odd-cycle free,* (3) *there is no cycle in $DG^c(P)$ which contains some choice-literal, and* (4) *each choice-literal occurs in the head of at most one rule.*

[3]For two substitutions σ, θ, the composition $\sigma\theta$ is such that for every term/atom t, $\sigma\theta(t) = \theta(\sigma(t))$.

[4]The discussion in this section can be easily extended to literals.

Theorem 4. *Every finitary c-program is consistent.*

Let P be a finitary c-program. Let R_P be the program obtained from P by replacing each rule r of the form (13) in P by the set of rules $\{r_1, \ldots, r_n\}$ where $\mathsf{head}(r_i) = A_i$ and $\mathsf{body}(r_i) = \mathsf{body}(r) \cup \{\mathtt{not}\, A_j \mid j \neq i\}$.

Let P be a finitary c-program and A be a ground atom in the language of P. In the following we define supports and countersupports of A in P as the corresponding supports and countersupports of A in R_P. Thus, atom A depends on a finite and finitary program whose odd-cycles are disjoint. In fact, each odd-cycle of R_P involves a set of specific atoms which do not appear in other odd-cycles. This property allows us to extend the notion of a ground credulous derivation to programs with choice rules. The only modification to Definition 3 is the introduction of the resolution rule:

*(***RC***) (Resolution with choice rule) there is a positive literal A in G_i, and there is a rule in* $\mathsf{Ground}(P)$ $1\{A_1, \ldots, A_n\}1 \leftarrow L_1, \ldots, L_m$ *where* $A = A_k \in \mathsf{head}(R)$*, then* $G_{i+1} = (G_i \setminus \{A\}) \cup \mathsf{body}(r)$ *and* $H_{i+1} = H_i \cup \{\mathtt{not}\, A_1, \ldots, \mathtt{not}\, A_n\} \setminus \{\mathtt{not}\, A_k\}$.

The notions of consistent and successful derivations are unchanged. We can extend Theorem 2 to program with choice rules as follows.

Theorem 5. *Let P be an finitary c-program and G be a ground goal. G is a credulous consequence of P iff $(G \mid \Box)$ has a successful credulous derivation from P.*

Programs with Constraints

We will now extend the credulous resolution to deal with constraints. A constraint is a clause of the form

$$\leftarrow L_1, \ldots, L_n$$

where L_i are literals. Intuitively, a constraint is used to reject all answer sets M such that $M \models L_1, \ldots, L_n$. For simplicity, we will denote a constraint by the set of literals $\{L_1, \ldots, L_n\}$. Let us write $P = P^+ \cup P^c$, where P^c are the constraints in P and $P^+ = P \setminus P^c$. Since answer sets of a program P are answer sets of P^+ which satisfy the constraints in P^c, one way to deal directly with constraints in resolution is to manage the constraints at each step of the derivation.

For a literal L, $\bar{L}$ denotes its complementary literal; i.e., if L is an atom, then $\bar{L} = \mathtt{not}\, L$, if L is $\mathtt{not}\, A$ then $\bar{L} = A$. The notation can be extended to set of literals—$\bar{S} = \{\bar{L} | L \in S\}$. Given a set of constraints C and a literal L, let

$$\begin{array}{rcl} relev(C, L) & = & \{c \mid c \textit{ is a constraint in } C \textit{ and } L \in c\} \\ antirel(C, L) & = & \{c \mid c \textit{ is a constraint in } C \textit{ and } \bar{L} \in c\} \end{array}$$

In order to include constraints in the computation, let us replace the concept of h-goal with the notion of *constraint h-goal (ch-goal).* A ch-goal has the form $(G \mid H \mid C)$, where $(G \mid H)$ is a h-goal while C is a collection of constraints. Intuitively, C are the constraints that have not been satisfied by the derivation yet. The notion of derivation presented next will drop from C those constraints that are satisfied by the current selected literal (i.e., if L is the selected literal, we drop each constraint X s.t. $\bar{L} \in X$); it will force the derivation to take into consideration additional literals to satisfy those constraints that are in danger of being violated.

A *ground constraint credulous derivation* of a ground h-goal $(G \mid H)$ from P is a (possibly infinite) sequence of ch-goals

$$(G_0 \mid H_0 \mid C_0)\,(G_1 \mid H_1 \mid C_1) \ldots (G_k \mid H_k \mid C_k) \ldots$$

where $G_0 = G$, $H_0 = H$, $C_0 = P^c$, and such that for all ch-goals $(G_i \mid H_i \mid C_i)$ in the sequence (but the last one, if any) some of the following conditions hold:

(**R**) (Resolution rule) for a positive literal A in G_i, a rule $r \in \mathsf{Ground}(P)$ such that $\mathsf{head}(r) = A$, and a finite collection of literals $S \subseteq \bigcup_{c \in relev(C_i, A)} (c \setminus \{A\})$ such that $\forall c \in relev(C_i, A).\ c \cap S \neq \emptyset$, we have that

$$\begin{array}{rcl} G_{i+1} & = & (G_i \setminus \{A\}) \cup body(r) \cup \bar{S} \\ H_{i+1} & = & H_i \\ C_{i+1} & = & C_i \setminus antirel(C_i, A) \end{array}$$

(**H**) (Resolution with hypothesis) for a negative literal $\mathtt{not}\, A \in G_i \cap H_i$ we have that $G_{i+1} = G_i \setminus \{\mathtt{not}\, A\}$, $C_{i+1} = C_i$ and $H_{i+1} = H_i$

(**F**) (Failure rule) for a literal $\mathtt{not}\, A$ in $G_i \setminus H_i$, a ground countersupport C_A for A (from P), and a finite set of literals $S \subseteq \bigcup_{c \in relev(C_i, \mathtt{not}\, A)} (c \setminus \{\mathtt{not}\, A\})$ such that $\forall c \in relev(C_i, \mathtt{not}\, A).\ c \cap S \neq \emptyset$, we have that

$$\begin{array}{rcl} G_{i+1} & = & (G_i \setminus \{\mathtt{not}\, A\}) \cup C_A \cup \bar{S} \\ H_{i+1} & = & H_i \cup \{\mathtt{not}\, A\} \\ C_{i+1} & = & C_i \setminus antirel(C_i, \mathtt{not}\, A) \end{array}$$

A computation is *consistent* if $pos(G_i) \cap neg(G_j) = \emptyset$ for each i, j. A computation is *strongly successful* if it is finite, consistent, and the last ch-goal is of the form $(\Box \mid H \mid \emptyset)$. A computation is *weakly successful* if it is finite, consistent, and the last ch-goal is of the form $(\Box \mid H \mid S)$.

Observe that constraints encode odd cycles in $DG(P)$. We say that P is an *order consistent constraint program* if P^+ is order consistent.

Theorem 6 (Strong Soundness and Completeness)**.** *Let P be a ground order consistent constraint program; if $(G \mid \Box)$ has a strongly successful derivation, then G is a credulous consequence of P.*
Let P be a ground finitary program with constraints. If G is a credulous consequence of P, then there is a set of literals G' such that $G \cup G'$ has a strongly successful derivation.

The intuition is that we may need to prove some additional literals in order to ensure that all constraints in P^c are reached and satisfied by the derivation. In an alternative perspective, more in-line with a *"CLP-style"* view of derivation, C_i can be viewed as a *constraint store*, and the removal of constraints as a specialized form of propagation and constraint consistency. A weak computation terminating in the ch-goal $(\Box \mid H \mid C)$ provides an "implicit" representation of a solution—where a solution is an answer set that is compatible with the assumptions H and satisfies the constraints C. Work is in progress to revisit the notion of derivation from this perspective.

A Preliminary Implementation

We developed a first (unoptimized) implementation to conduct some preliminary experiments. Non-ground credulous

Yale Shooting	n=4	n=9	n=10	n=100	n=200
c-resolution	0.01	0.01	0.01	0.21	0.67
smodels	0.03	0.07	0.09	0.48	1.01
Towers of Hanoi	n=3	n=4	n=5	n=6	n=7
c-resolution	0.059	0.311	2.229	28.433	t.o.
smodels	8.355	t.o.	t.o.	t.o.	t.o.
Bomb in the Toilet	p=4, t=4				
c-resolution	0.10				
smodels	90.402				
Block World	no. of blocks=3				
c-resolution	1.055				
smodels	0.521				
Synth Planning	time steps=30				
c-resolution	0.425				
smodels	1.244				

Table 3: Execution time in seconds

resolution is implemented in XSB Prolog, a Prolog system capable of memoization of goals. Memoization is used to guarantee termination in the presence of positive cycles. Termination in the presence of negative cycles is guaranteed by giving rule (**H**) higher priority than (**F**). In this implementation, non-ground countersupport computation is complete only for safe programs (i.e., each variable should occur in at least one positive literal in the body of a rule). We also introduced treatment of constraints, as in the scheme described in this paper. Tabulation is also needed to factorize common subproofs, thereby leading to significant speedups. XSB supports two tabulation strategies, called *local* and *batched*. The latter was used as it significantly outperformed the former, both in terms of time and memory requirements.

We tested our prototype on a selection of benchmarks. These include the *Yale Shooting Problem* with varying number of time points (i.e., the initial goal is $\mathtt{H}(\neg\mathtt{alive}, n)$), the *Towers of Hanoi* with 3 poles and a varying number n of disks (the number of actions required to solve the Towers of Hanoi with n disks is $2^n - 1$), the *Bomb in the Toilet* planning problem (4 packets, 4 toilets), in its conformant version without any initial knowledge, a block world planning problem and a synthetic planning problem with a very large search space. The experiments have been run on a dual core with a 2.6 GHz clock and 4GB RAM, running Windows Vista and cygwin. Response times are reported in Table 3 (time in seconds, t.o. denotes timeout after 200sec.). We also report, for comparison, the corresponding time to detect an answer set satisfying the goal using SMODELS.[5]

Conclusion and Future Work

In this paper, we presented a goal-oriented top-down notion of derivation that captures the concept of credulous consequence under answer set semantics. The notion of derivation enables a goal-oriented view of answer set semantics, differently from the bottom-up approach traditionally used in most existing ASP systems. This approach does not require grounding of programs and it allows us to handle computations with infinite and/or non-ground programs. We have proved correctness and completeness for large classes of programs and extended the derivation to allow the presence of constraints and choice rules. A preliminary implementation in XSB Prolog has also been developed.

This work represents the first step in this promising direction. We are currently investigating more effective implementation schemes. We are also interested in revisiting the concept of derivation as an instance of a CLP framework, which is expected to allow a more effective handling of ASP constraints and the ability to elegantly extend ASP with other forms of reasoning (e.g., numerical constraints).

[5]Observe that SMODELS relies on a radically different computation model and it is not goal-oriented.

References

Balduccini, M.; *et al.* 2001. The USA-Advisor: A Case Study in Answer Set Planning. *LPNMR*, Springer Verlag.

Bonatti, P. A. 2001. Resolution for skeptical stable model semantics. *J. Autom. Reasoning* 27(4):391–421.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence Journal* 156:75–111.

Bonatti, P. A; *et al.* 2008. Credulous Resolution for ASP. Technical Report, NMSU–CS–2008–001. `www.cs.nmsu.edu/CSWS/php/techReports.php`.

Carlucci, L.; *et al.* 2001. Verifying Security Protocols as Planning in Logic Programming. *ACM TOCL* 2(4).

Dovier, A.; *et al.* 2008. An Empirical Study of CLP and Answer Set Programming. *JETAI* (to appear).

Eiter, T.; *et al.* 1997. Computing non-ground representations of stable models. *LPNMR*, 198–217. Springer Verlag.

Eiter, T.; *et al.* 2003a. A Logic Programming Approach to Knowledge State Planning, II: The DLV$^{\mathcal{K}}$ System. *Artificial Intelligence* 144(1-2):157–211.

Ferraris, P.; *et al.* 2007. A New Perspective on Stable Models. In *IJCAI*, 372–379.

Fitting, M. 1994. Tableaux for Logic Programming. *J. Autom. Reasoning* 13(2).

Gebser, M., and Schaub, T. 2006. Tableau Calculi for ASP. *ICLP*, Springer Verlag.

Gelfond, M., and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. *JICSLP*, MIT Press.

Heljanko, K., and Niemelä, I. 2003. Bounded LTL model checking with stable models. *TPLP* 3(4,5):519–550.

Lloyd, J. 1987. *Foundations of logic programming*. Springer Verlag.

Marek, V., and Truszczyński, M. 1999. Stable Models as an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm*, 375–398, Springer Verlag.

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and AI* 25(3,4):241–273.

Simons, P.; *et al.* 2002. Extending and Implementing the Stable Model Semantics. *AIJ*, 138(1–2):181–234.

Son, T. C., and Pontelli, E. 2007. Planning for Biochemical Pathways. In *SEA'07 Workshop*.

Son, T. C.; *et al.* 2005. An Approximation of Action Theories of $\mathcal{AL}$ and its Application to Conformant Planning. In *LPNMR*, 172–184, Springer Verlag.