# Computing Answer Sets of CR-Prolog Programs

Marcello Balduccini
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
marcello.balduccini@ttu.edu

December 19, 2006

**Abstract**

CR-Prolog is an extension of the knowledge representation language A-Prolog. The extension is built around the introduction of *consistency-restoring rules* (cr-rules for short), and allows an elegant formalization of events or exceptions that are unlikely, unusual, or undesired. The flexibility of the language has been extensively demonstrated in the literature, with examples that include planning and diagnostic reasoning.

In this paper we present the design and implementation of an inference engine for CR-Prolog that is efficient enough to allow the practical use of the language for medium-size applications. The capabilities of the inference engine have been successfully demonstrated with experiments on an application independently developed for use by NASA.

## 1 Introduction

In recent years, A-Prolog – a knowledge representation language based on the answer set semantics [17] – was shown to be a useful tool for knowledge representation and reasoning (e.g. [24, 15, 7]). The language is expressive and has a well understood methodology of representing defaults, causal properties of actions and fluents, various types of incompleteness, etc. Over time, several extensions of A-Prolog have been proposed [12, 22, 15, 13, 4, 11, 9], aimed at improving event further the expressive power of the language.

One of these extensions, called CR-Prolog [4, 6], is built around the introduction of *consistency-restoring rules* (cr-rules for short). The intuitive idea behind cr-rules is that they are normally not applied, even when their body is satisfied. They are only applied if the regular program (i.e. the program consisting only of conventional A-Prolog rules) is inconsistent. The language also allows the specification of a partial preference order on cr-rules, intuitively regulating the application of cr-rules.

One of the most immediate uses of cr-rules is an elegant encoding of events or exceptions that are unlikely, unusual, or undesired (and preferences can be used to formalize the relative likelihood of these events and exceptions).

The flexibility of CR-Prolog has been extensively demonstrated in the literature [4, 1, 9, 2, 16, 5], with examples including planning and diagnostic reasoning. For example, in [4], cr-rules have been used to model exogenous actions that may occur, unobserved, and cause malfunctioning in a physical system. In [1, 5], cr-rules have been applied to the task finding high quality plans. The technique consists in encoding requirements that high quality plans must satisfy, and using cr-rules to formalize exceptions to the requirements, that should be considered only as a last resort. In [16], cr-rules are used to model interruptions of sequences of actions that an agent intends to perform.

Most of the uses of CR-Prolog in the literature are not strongly concerned with computation time, and use relatively simple prototypes of CR-Prolog inference engines. However, to allow the use of CR-Prolog for practical applications, an efficient inference engine is needed.

In this paper, we present the design and implementation of an inference engine for CR-Prolog that is efficient enough to allow the practical use of CR-Prolog for medium-size applications. The paper is organized as follows. In the next section, we introduce the syntax and semantics of CR-Prolog. Section 3 contains a description of the algorithm of the inference engine, whose soundness and completeness are proven in Section 4. In Section 5, we discuss interesting implementation issues. Finally, in Sections 6 and 7 we talk about related work draw conclusions.

# 2 CR-Prolog

Like A-Prolog, CR-Prolog is a knowledge representation language that allows the formalization of commonsense knowledge and reasoning. The consistency-restoring rules introduced in CR-Prolog allow the encoding of statements that should be used "as rarely as possible, and only if strictly necessary to obtain a consistent set of conclusions," with preferences intuitively determining which statements should be given precedence. The language has been shown to allow the elegant formalization of various sophisticated reasoning tasksthat are problematic to encode in A-Prolog.

The syntax of CR-Prolog is determined by a typed signature $\Sigma$ consisting of types, typed object constants, and typed function and predicate symbols. We assume that the signature contains symbols for integers and for the standard functions and relations of arithmetic. Terms are built as in first-order languages.

By *simple arithmetic terms* of $\Sigma$ we mean its integer constants. By *complex arithmetic terms* of $\Sigma$ we mean terms built from legal combinations of arithmetic functions and simple arithmetic terms (e.g. $3+2\cdot 5$ is a complex arithmetic term, but $3+\cdot\ 2\ 5$ is not).

Atoms are expressions of the form $p(t_1,\ldots,t_n)$, where $p$ is a predicate symbol with arity $n$ and $t$'s are terms of suitable types. Atoms formed by arithmetic relations are called *arithmetic atoms*. Atoms formed by non-arithmetic relations are called *plain*

*atoms.* We allow arithmetic terms and atoms to be written in notations other than prefix notation, according to the way they are traditionally written in arithmetic (e.g. we write $3 = 1 + 2$ instead of $= (3, +(1, 2))$).

Literals are atoms and negated atoms, i.e. expressions of the form $\neg p(t_1, \ldots, t_n)$. Literals $p(t_1, \ldots, t_n)$ and $\neg p(t_1, \ldots, t_n)$ are called *complementary*. By $\bar{l}$ we denote the literal complementary to $l$.

The syntax of the statements of CR-Prolog is defined as follows.

**Definition 2.1** *A* regular rule $\rho$ *is a statement of the form:*

$$r : h_1 \text{ OR } h_2 \text{ OR } \ldots \text{ OR } h_k \leftarrow l_1, l_2, \ldots l_m, \textit{not } l_{m+1}, \textit{not } l_{m+2}, \ldots, \textit{not } l_n. \quad (1)$$

*where $r$ is a term that uniquely denotes $\rho$ (called name of the rule), $l_1, \ldots, l_m$ are literals, and $h_i$'s and $l_{m+1}, \ldots, l_n$ are plain literals. We call $h_1$ OR $h_2$ OR $\ldots$ OR $h_k$ the* head *of the rule ($head(r)$); $l_1, l_2, \ldots l_m, not\ l_{m+1}, not\ l_{m+2}, \ldots, not\ l_n$ is its* body *($body(r)$), and $pos(r)$, $neg(r)$ denote, respectively, $\{l_1, \ldots, l_m\}$ and $\{l_{m+1}, \ldots, l_n\}$.*

The informal reading of the rule (in terms of the reasoning of a rational agent about its own beliefs) is the same used in A-Prolog: "if you believe $l_1, \ldots, l_m$ and have no reason to believe $l_{m+1}, \ldots, l_n$, then believe one of $h_1, \ldots, h_k$." The connective "not" is called *default negation.* To simplify the presentation, we allow the rule name to be omitted whenever possible.

A rule such that $k = 0$ is called *constraint*, and is considered a shorthand of:

$$false \leftarrow \text{not } false, l_1, l_2, \ldots l_m, \text{not } l_{m+1}, \text{not } l_{m+2}, \ldots, \text{not } l_n.$$

**Definition 2.2** *A* consistency-restoring rule *(or* cr-rule*) is a statement of the form:*

$$r : h_1 \text{ OR } h_2 \text{ OR } \ldots \text{ OR } h_k \stackrel{+}{\leftarrow} l_1, l_2, \ldots l_m, \textit{not } l_{m+1}, \textit{not } l_{m+2}, \ldots, \textit{not } l_n. \quad (2)$$

*where $r$, $h_i$'s and $l_i$'s are as before.*

The intuitive reading of a cr-rule is "if you believe $l_1, \ldots, l_m$ and have no reason to believe $l_{m+1}, \ldots, l_n$, then you *may possibly* believe one of $h_1, \ldots, h_k$." The implicit assumption is that this possibility is used as little as possible, and only to restore consistency of the agent's beliefs.

**Definition 2.3** *A* CR-Prolog program *is a pair $\langle \Sigma, \Pi \rangle$, where $\Sigma$ is a typed signature and $\Pi$ is a set of regular rules and cr-rules.*

In this paper we often denote programs of CR-Prolog by their second element. The corresponding signature is denoted by $\Sigma(\Pi)$. We also extend the basic operations on sets to programs in a natural way, so that, for example, $\Pi_1 \cup \Pi_2$ is the program whose signature and set of rules are the unions of the respective components of $\Pi_1$ and $\Pi_2$.

The terms, atoms and literals of a program $\Pi$ are denoted respectively by $terms(\Pi)$, $atoms(\Pi)$ and $literals(\Pi)$. Given a set of relations $\{p_1, \ldots, p_m\}$, $atoms(\{p_1, \ldots, p_m\}, \Pi)$ denotes the set of atoms from the signature of $\Pi$ formed by

every $p_i$. $literals(\{p_1,\ldots,p_m\},\Pi)$ is defined in a similar way. To simplify notation, we allow the use of $atoms(p,\Pi)$ as an abbreviation of $atoms(\{p\},\Pi)$ (and similarly for $literals$).

Given a CR-Prolog program, $\Pi$, the *regular part* of $\Pi$ is the set of its regular rules, and is denoted by $reg(\Pi)$. The set of cr-rules of $\Pi$ is denoted by $cr(\Pi)$.

**Example 2.1** *Consider the following program:*

$$\begin{cases} r_1 : p \text{ OR } q \stackrel{+}{\leftarrow} not\ r. \\ s. \end{cases}$$

*The regular part of the program, consisting of fact $s$, is consistent. Hence, the is no reason to apply the cr-rule, and the agent is only forced to believe $s$.*

**Example 2.2** *Now consider the program:*

$$\begin{cases} r_1 : p \text{ OR } q \stackrel{+}{\leftarrow} not\ r. \\ s. \\ \leftarrow not\ p, not\ q. \end{cases}$$

*This time, the regular part of the program is inconsistent. The cr-rule can be applied to restore consistency, and the agent is forced to believe either $\{s,p\}$ or $\{s,q\}$.*

It is also possible to have cases when different cr-rules can be applied, like in the following example.

**Example 2.3**

$$\begin{cases} r_1 : p \stackrel{+}{\leftarrow} not\ r. \\ r_2 : q \stackrel{+}{\leftarrow} not\ r. \\ s. \\ \leftarrow not\ p, not\ q. \end{cases}$$

*Again, the regular part of the program is inconsistent. Consistency can be restored by applying either $r_1$ or $r_2$, or both. Since cr-rules should be applied as little as possible, the last case is not considered. Hence, the agent is forced to believe either $\{s,p\}$ or $\{s,q\}$.*

When different cr-rules are applicable, it is possible to specify preferences on which one should be applied by means of atoms of the form

$$prefer(r_1, r_2),$$

where $r_1$, $r_2$ are names of cr-rules. The atom informally says "do not consider solutions obtained using $r_2$ unless no solution can be found using $r_1$." The next example shows the effect of the introduction of preferences in the program from Example 2.3.

**Example 2.4**

$$\begin{cases} r_1 : p \stackrel{+}{\leftarrow} not\ r. \\ r_2 : q \stackrel{+}{\leftarrow} not\ r. \\ prefer(r_1, r_2). \\ s. \\ \leftarrow not\ p, not\ q. \end{cases}$$

*The preference prevents the agent from applying $r_2$ unless no solution can be found using $r_1$. We have seen already that $r_1$ is sufficient to restore consistency. Hence, the agent has only one set of beliefs, $\{s, p, prefer(r_1, r_2)\}$*

Notice that our reading of the preference atom $prefer(r_1, r_2)$ rules out solutions in which $r_1$ and $r_2$ are applied simultaneously, as the use of $r_2$ is allowed only if no solution is obtained by applying $r_1$.

It is important to observe that the definition of the syntax of CR-Prolog does not allow the use of variables. As usual, we assume that programs containing variables (denoted by capital letters) are shorthands for the sets of their ground instantiations, obtained by substituting the variables with all the terms of appropriate type from the signature of the program. The approach is justified for the so called closed domains, i.e. domains satisfying the domain closure assumption [25] that all objects in the domain of discourse have names in the language of the program. An (A-Prolog oriented) investigation of open domains can be found in [8, 18].

In the rest of this section, we define the semantics of CR-Prolog. In the following discussion, $\Pi$ denotes an arbitrary CR-Prolog program. Also, for every $R' \subseteq cr(\Pi)$, $\theta(R')$ denotes the set of regular rules obtained from $R'$ by replacing every connective $\stackrel{+}{\leftarrow}$ with $\leftarrow$. Notice that the regular part of any CR-Prolog program is an A-Prolog program. We will begin by introducing some terminology.

An atom is in *normal form* if it is an arithmetic atom or if it is a plain atom and its arguments are either non-arithmetic terms or simple arithmetic terms. Notice that literals that are not in normal form can be mapped into literals in normal form by applying the standard rules of arithmetic. For example, $p(4+1)$ is mapped into $p(5)$. For this reason, in the following definition of the semantics of CR-Prolog, we assume that all literals are in normal form.

A literal $l$ is *satisfied* by a consistent set of plain literals $S$ (denoted by $S \models l$) if:

- $l$ is an arithmetic literal and is true according to the standard arithmetic interpretation;
- $l$ is a plain literal and $l \in S$.

If $l$ is not satisfied by $S$, we write $S \not\models l$. An expression not $l$, where $l$ is a plain literal, is satisfied by $S$ if $S \not\models l$. A set of literals and literals under default negation (not $l$) is satisfied by $S$ if each element of the set is satisfied by $S$. A rule is satisfied by $S$ if either its head is satisfied or its body is not satisfied.

Next, we introduce the transitive closure of relation $prefer$. To simplify the presentation, we use, whenever possible, the same term to denote both a rule and its name. For example, given rules $r_1, r_2 \in cr(\Pi)$, the fact that $r_1$ is preferred to $r_2$ will be expressed by a statement $prefer(r_1, r_2)$. Notice that this is made possible by the fact that rules are uniquely identified by their names.

**Definition 2.4** For every set of literals, $S$, from the signature of $\Pi$, and every $r_1$,$r_2$ from $cr(\Pi)$, $pref_S(r_1, r_2)$ is true iff

$prefer(r_1, r_2) \in S$, or

there exists $r_3 \in cr(\Pi)$ such that $prefer(r_1, r_3) \in S$ and $pref_S(r_3, r_2)$.

To see how the definition works, consider the following example.

**Example 2.5** *Given $S = \{prefer(r_1, r_2), prefer(r_2, r_3), a, q, p\}$ and $cr(\Pi)$ consisting of cr-rules $r_1, r_2, r_3$:*

- $pref_S(r_1, r_2)$ *holds (because $prefer(r_1, r_2) \in S$).*
- $pref_S(r_2, r_3)$ *holds (because $prefer(r_2, r_3) \in S$).*
- $pref_S(r_1, r_3)$ *holds (because $prefer(r_1, r_2) \in S$ and $pref_S(r_2, r_3)$ holds).*

The semantics of CR-Prolog is given in three steps. Intuitively, in the first step we look for combinations of cr-rules that restore consistency. Preferences are not considered, with the exception that solutions deriving from the simultaneous use of two cr-rules between which a preference exists are discarded.

**Definition 2.5** Let $S \subseteq literals(\Pi)$ and $R \subseteq cr(\Pi)$. $\mathscr{V} = \langle S, R \rangle$ is a *view* of $\Pi$ if:

1. $S$ is an answer set of $reg(\Pi) \cup \theta(R)$, and
2. for every $r_1$, $r_2$, if $pref_S(r_1, r_2)$, then $\{r_1, r_2\} \not\subseteq R$, and
3. for every $r$ in $R$, $body(r)$ is satisfied by $S$.

We denote the elements of $\mathscr{V}$ by $\mathscr{V}^S$ and $\mathscr{V}^R$ respectively. The cr-rules in $\mathscr{V}^R$ are said to be *applied.*

**Example 2.6** *Consider the program, $P_1$:*

$$\begin{cases} r_1 : t \stackrel{+}{\leftarrow} . \\ r_2 : p \stackrel{+}{\leftarrow} q. \\ r_3 : s \stackrel{+}{\leftarrow} . \\ r_4 : q \stackrel{+}{\leftarrow} . \\ \\ \leftarrow \mathit{not}\ t, \mathit{not}\ p, \mathit{not}\ s. \\ \\ prefer(r_1, r_3). \end{cases}$$

*First of all, notice that the regular part of the program is inconsistent. Hence, cr-rules are applied. According to Definition 2.5,*

$$\mathscr{V}_1 = \langle \{t, prefer(r_1, r_3)\}, \{r_1\} \rangle$$

*is a view of $P_1$. In fact: (1) $\mathscr{V}_1^S$ is an answer set of $reg(P_1) \cup \theta(\mathscr{V}_1^R)$; (2) $\{r_1, r_3\} \not\subseteq \mathscr{V}_1^R$; and (3) the body of $r_1$ is trivially satisfied. On the other hand,*

$$\mathscr{V}_x = \langle \{t, s, prefer(r_1, r_3)\}, \{r_1, r_3\} \rangle$$

*is not a view of $P_1$, because it does not satisfy condition (2) of the definition. In fact, $pref_{\mathscr{V}_1^S}(r_1, r_3)$ holds but $\{r_1, r_3\} \subseteq \mathscr{V}_1^R$. Similarly,*

$$\mathscr{V}_y = \langle \{t, prefer(r_1, r_3)\}, \{r_1, r_2\} \rangle$$

*is not a view of $P_1$. In this case, condition (3) of the definition is not satisfied, as the body of $r_2$ does not hold in $\mathscr{V}_1^S$. It is not difficult to show that the views of $P_1$ are (from now on, we omit preference atoms, whenever possible, to save space):*

$$\begin{array}{ll} \mathscr{V}_1 = \langle \{t\}, \{r_1\} \rangle & \mathscr{V}_2 = \langle \{t, q\}, \{r_1, r_4\} \rangle \\ \mathscr{V}_3 = \langle \{s\}, \{r_3\} \rangle & \mathscr{V}_4 = \langle \{s, q\}, \{r_3, r_4\} \rangle \\ \mathscr{V}_5 = \langle \{p, q\}, \{r_2, r_4\} \rangle & \mathscr{V}_6 = \langle \{s, p, q\}, \{r_2, r_3, r_4\} \rangle \\ \mathscr{V}_7 = \langle \{t, p, q\}, \{r_1, r_2, r_4\} \rangle & \end{array}$$

The second step in the definition of the answer sets of $\Pi$ consists in selecting the best views with respect to the preferences specified. Particular attention must be paid to the case when preferences are dynamic. The intuition is that we consider only preferences on which there is agreement in the views under consideration.

**Definition 2.6** For every pair of views of $\Pi$, $\mathscr{V}_1$ and $\mathscr{V}_2$, $\mathscr{V}_1$ *dominates* $\mathscr{V}_2$ if there exist $r_1 \in \mathscr{V}_1^R$, $r_2 \in \mathscr{V}_2^R$ such that $pref_{(\mathscr{V}_1^S \cap \mathscr{V}_2^S)}(r_1, r_2)$.

**Example 2.7** *Let us consider the views of program $P_1$ from Example 2.6. View $\mathscr{V}_1$ dominates $\mathscr{V}_3$: in fact, $\mathscr{V}_1^S \cap \mathscr{V}_3^S = \{prefer(r_1, r_3)\}$ and $pref_{\{prefer(r_1,r_3)\}}(r_1, r_3)$ obviously holds. On the other hand, $\mathscr{V}_1$ does not dominate $\mathscr{V}_5$, as neither $pref_{\{prefer(r_1,r_3)\}}(r_1, r_2)$ nor $pref_{\{prefer(r_1,r_3)\}}(r_1, r_4)$ hold.*

**Definition 2.7** A view, $\mathscr{V}$, is a candidate answer set of $\Pi$ if, for every view $\mathscr{V}'$ of $\Pi$,

$$\mathscr{V}' \text{ does not dominate } \mathscr{V}.$$

**Example 2.8** *According to the conclusions from Example 2.6, $\mathscr{V}_3$ is not a candidate answer of $P_1$, as it is dominated by $\mathscr{V}_1$. Conversely, it is not difficult to see that $\mathscr{V}_1$ is not dominated by any other view, and is therefore a candidate answer set. Overall, the candidate answer sets of $P_1$ are:*

$$\mathscr{V}_1 = \langle \{t\}, \{r_1\} \rangle \quad \mathscr{V}_2 = \langle \{t, q\}, \{r_1, r_4\} \rangle \quad \mathscr{V}_5 = \langle \{p, q\}, \{r_2, r_4\} \rangle$$

Finally, we select the candidate answer sets that are obtained by applying a minimal set (w.r.t. set-theoretic inclusion) of cr-rules.

**Definition 2.8** A set of literals, $S$, is an *answer set* of $\Pi$ if:

1. there exists $R \subseteq cr(\Pi)$ such that $\langle S, R \rangle$ is a candidate answer set of $\Pi$, and

2. for every candidate answer set $\langle S', R' \rangle$ of $\Pi$, $R' \not\subset R$.

**Example 2.9** *Consider $\mathcal{V}_1$ and $\mathcal{V}_2$ from the list of the candidate answer sets of $P_1$ from Example 2.8. Since ${\mathcal{V}_1}^R \subseteq {\mathcal{V}_2}^R$, $\mathcal{V}_2$ is not an answer set of $P_1$. According to Definition 2.8, the answer sets of $P_1$ are:*

$$\mathcal{V}_1 = \langle \{t\}, \{r_1\} \rangle \quad \mathcal{V}_5 = \langle \{p, q\}, \{r_2, r_4\} \rangle$$

# 3 The CRMODELS Algorithm

The algorithm for computing the answer sets of CR-Prolog programs is based on a generate-and-test approach. We begin our description of CRMODELS by presenting the algorithm at a high level of abstraction. Next, we increase the level of detail in various steps, until we have a complete specification of CRMODELS.

At a high level of abstraction, one answer set of a CR-Prolog program $\Pi$ can be computed as show below (Figure 1). Notice that, in the algorithm, $\perp$ is used to indicate the absence of a solution.

**Algorithm: CRMODELS$_1$**

**input:** $\Pi$: CR-Prolog program
**output:** one answer set of $\Pi$

var $i$: number of cr-rules to be applied

1. $i := 0$ { first we look for an answer set of $reg(\Pi)$ }
2. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
3. *repeat* { **inner loop** }
4. generate new view $\mathcal{V}$ of $\Pi$ s.t. $|{\mathcal{V}}^R| = i$; if none is found, $\mathcal{V} := \perp$
5. *if* $\mathcal{V}$ is candidate answer set of $\Pi$ *then* { test fails if $\mathcal{V} = \perp$ }
6. *return* ${\mathcal{V}}^S$ { answer set found }
7. *end if*
8. *until* $\mathcal{V} = \perp$
9. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
10. *done*
11. *return* $\perp$ { signal that no answer set was found }

Figure 1: Algorithm CRMODELS$_1$

The algorithm begins by looking for a view $\mathcal{V}$ such that $|{\mathcal{V}}^R| = 0$. If one is found, CRMODELS$_1$ checks that $\mathcal{V}$ is a candidate answer set of $\Pi$ (line 5). Notice that, because $|{\mathcal{V}}^R| = 0$, the condition of Definition 2.6 is never satisfied (as there is no $r \in {\mathcal{V}}^R$). Hence, if a view if found for $i = 0$, that view is a candidate answer set, which causes

the test at line 5 to succeed. Such a candidate answer set is also minimal w.r.t. set-theoretic inclusion on $\mathscr{V}^R$, which implies that $\mathscr{V}^S$ is an answer set of $\Pi$ according to Definition 2.8. Hence, the algorithm returns $\mathscr{V}^S$ and terminates.

Now let us consider what happens if no view is found for $i = 0$. According to line 4, $\mathscr{V}$ is set to $\bot$, which causes the test on line 5 to fail. Because the termination condition of the inner loop (line 8) is true, the loop terminates, $i$ is incremented and, assuming $\Pi$ contains at least one cr-rule, execution goes back to line 4, where a view $\mathscr{V}$ with $|\mathscr{V}^R| = 1$ is computed. It is important to notice[1] that, because of the iteration over increasing values of $i$ in the outer loop (lines 2–10), the first candidate answer set found by the algorithm is always guaranteed to be set-theoretically minimal (with respect to the set of cr-rules used). Hence, according to Definition 2.8, $\mathscr{V}^S$ is an answer set of $\Pi$. That explains why the return statement at line 6 is executed without further testing. If no candidate answer set is found for $i = 1$, the iterations of the outer loop continue for increasing values of $i$ until either a candidate answer set is found or the condition on line 2 becomes false (i.e. all possible combinations of cr-rules have been considered). In this case, the algorithm returns $\bot$.

In our approach, both the generation and the test steps (lines 4 and 5 in Figure 1) are reduced to the computation of answer sets of *A-Prolog programs*. To allow a compact representation of the A-Prolog programs involved in these steps, we introduce the following *macros*.

- A macro-rule of the form:

  $$\{p(X)\}\cdot \tag{3}$$

  informally says that any $X$ can have property $p$, and stands for the rules:

  $$p(X) \leftarrow \text{not } \neg p(X)\cdot$$
  $$\neg p(X) \leftarrow \text{not } p(X)\cdot$$

- A macro-rule of the form:

  $$\leftarrow \text{not } i\{p(X)\}j\cdot \tag{4}$$

  informally states that only between $i$ and $j$ $X$'s can have property $p$ and is expanded as follows. Let $t$ denote the cardinality of the ground atoms of the form $p(X)$ and $\Delta(m)$ denote the collection of inequalities:

  $$X_k \neq X_h \qquad\qquad \forall\, k,h \text{ s.t. } 1 \leq k \leq m, 1 \leq h \leq m, k \neq h\cdot$$

  The macro-rule stands for:

  $$\leftarrow p(X_1), p(X_2), \ldots, p(X_j), p(X_{j+1}), \Delta(j+1)\cdot$$
  $$\leftarrow \text{not } p(X_1), \text{not } p(X_2), \ldots, \text{not } p(X_{j-i}), \Delta(j-i)\cdot$$

[1] A refinement of this statement will be proven later in the paper.

Adopting a terminology similar to [22], we call (3) a *choice macro* and (4) a *cardinality macro*. The use of these macros allows us to keep a compact description of the programs, later in the paper, without committing to a particular extension of A-Prolog (and to its inference engine). Moreover, the structure of the macros is simple enough to allow their translation, at the time of the implementation of the algorithm, to more efficient expressions, such as choice rules and cardinality constraints [26, 22], statements about sets from the language of A-Prolog with sets [15], or statements about aggregates from the language of DLV [13].

Central to both steps is the notion of *hard reduct* of a CR-Prolog program, which we introduce next.

## 3.1 The Hard Reduct

The hard reduct of a CR-Prolog program $\Pi$, denoted by $hr(\Pi)$, maps $\Pi$ into an A-Prolog program. The importance of $hr(\Pi)$ is in the fact that *there is a one-to-one correspondence between the views of* $\Pi$ *and the answer sets of* $hr(\Pi)$, as shown by Lemmas 9 and 10 later in this paper.

The signature of $hr(\Pi)$ is obtained from the signature of $\Pi$ by the addition of predicate symbols $appl$, $is\_preferred$, $bodytrue$, $o\_appl$, $o\_is\_preferred$, $dominates$. For simplicity we assume that none of those predicate names occurs in the signature of $\Pi$. We also assume that the signature of $\Pi$ already contains the predicate name $prefer$. In the description of the hard reduct that follows, variable $R$, possibly indexed, ranges over the names of cr-rules.

**Definition 3.1 (Hard Reduct of $\Pi$)** *Let $\Pi$ be a CR-Prolog program. The hard reduct of $\Pi$, $hr(\Pi)$, consists of:*

1. *Every regular rule from $\Pi$.*
2. *For every cr-rule $r \in cr(\Pi)$ with head $h_1$ OR ... OR $h_k$ and body $l_1, \ldots l_m$, not $l_{m+1}, \ldots,$ not $l_n$, two rules:*

   $$h_1 \text{ OR } \ldots \text{ OR } h_k \leftarrow l_1, \ldots l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n, appl(r). \tag{5}$$

   *and*

   $$bodytrue(r) \leftarrow l_1, \ldots l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n. \tag{6}$$

3. *The* generator rule*, intuitively allowing the application of arbitrary sets of cr-rules:*

   $$\{appl(R)\}.$$

4. *A constraint prohibiting the application of a cr-rule when its the body is not satisfied (intuitively corresponding to condition (3) of Definition 2.5):*

   $$\leftarrow \textit{not } bodytrue(R), appl(R).$$

5. *Rules defining the transitive closure of relation* $prefer$*:*

$$is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_2)\cdot$$
$$is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_3), is\_preferred(R_3, R_2)\cdot$$

6. *A rule prohibiting the application of cr-rules* $r_1$ *and* $r_2$ *if* $r_1$ *is preferred to* $r_2$ *(intuitively corresponding to condition (2) of Definition 2.5):*

$$\leftarrow appl(R_1), appl(R_2), is\_preferred(R_1, R_2)\cdot$$

**Example 3.1** *Let us compute the hard reduct of the following program,* $P_2$*:*

$$\begin{cases} r_1 : p \overset{+}{\leftarrow} \textit{not } q\cdot \\ r_2 : s \overset{+}{\leftarrow} \cdot \\ \\ r_3 : \leftarrow \textit{not } p, \textit{not } s\cdot \\ \\ r_4 : prefer(r_1, r_2)\cdot \end{cases}$$

*According to item (1) above,* $hr(P_2)$ *contains the regular rules* $r_3$ *and* $r_4$*. For cr-rule* $r_1$*,* $hr(P_2)$ *contains:*

$$\begin{cases} p \leftarrow \textit{not } q, appl(r_1)\cdot \\ bodytrue(r_1) \leftarrow \textit{not } q\cdot \end{cases}$$

*For* $r_2$*,* $hr(P_2)$ *contains:*

$$\begin{cases} s \leftarrow appl(r_2)\cdot \\ bodytrue(r_2)\cdot \end{cases}$$

*Items (3 – 6) result in the addition of the rules:*

$$\begin{cases} \{appl(R)\}\cdot \\ \leftarrow \textit{not } bodytrue(R), appl(R)\cdot \\ \\ is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_2)\cdot \\ is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_3), is\_preferred(R_3, R_2)\cdot \\ \\ \leftarrow appl(R_1), appl(R_2), is\_preferred(R_1, R_2)\cdot \end{cases}$$

*The answer sets of* $hr(P_2)$ *are:*

$$\{p, appl(r_1), bodytrue(r_1), bodytrue(r_2), prefer(r_1, r_2), is\_preferred(r_1, r_2)\}$$
$$\{s, appl(r_2), bodytrue(r_1), bodytrue(r_2), prefer(r_1, r_2), is\_preferred(r_1, r_2)\}$$

*which correspond to the views:*

$$\mathcal{V}_1 = \langle\{p, prefer(r_1, r_2)\}, \{r_1\}\rangle \quad \mathcal{V}_2 = \langle\{s, prefer(r_1, r_2)\}, \{r_2\}\rangle$$

## 3.2 The Generation Step

In the generation step of the algorithm (line 4 from Figure 1), we find a view $\mathscr{V}$ of $\Pi$ such that $\mathscr{V}^R$ has a specified cardinality $i$ (the task of finding a *new* view satisfying the condition will be addressed later). The task is reduced to that of computing an answer set of $hr(\Pi)$ containing exactly $i$ occurrences of atoms of the form $appl(R)$. In turn, this is reduced to finding an answer set of the *$i$-generator of* $\Pi$, $\gamma_i(\Pi)$, defined below.

**Definition 3.2 ($i$-Generator of $\Pi$)** *Let $\Pi$ be a CR-Prolog program, and $i$ a non-negative integer such that $i \leq |cr(\Pi)|$. The* $i$-generator of $\Pi$ *is the program:*

$$hr(\Pi) \cup \{ \quad \leftarrow \textit{not } i\{appl(R)\}i \quad \}.$$

It is not difficult to show that $\gamma_i(\Pi)$ has the following properties (for more details, see Section 4):

- $M$ is an answer set of $\gamma_0(\Pi)$ iff $M \cap \Sigma(\Pi)$ is an answer set of $reg(\Pi)$.
- Every answer set of $\gamma_i(\Pi)$ is an answer set of $hr(\Pi)$.
- Every answer set $M$ of $\gamma_i(\Pi)$ contains exactly $i$ atoms of the form $appl(R)$.

**Example 3.2** *Consider program $P_2$ from Example 3.1. The $i$-generators for $P_2$ for various values of $i$ and the corresponding answer sets are as follows:*

- $\gamma_0(P_2) = hr(P_2) \cup \{ \quad \leftarrow \textit{not } 0\{appl(R)\}0 \quad \}.$

  *The program has no answer sets, since the constraint prevents any cr-rules from being applied and the regular part of $P_2$ is inconsistent.*

- $\gamma_1(P_2) = hr(P_2) \cup \{ \quad \leftarrow \textit{not } 1\{appl(R)\}1 \quad \}.$

  *The program allows the application of* 1 *cr-rule at a time. Its answer sets are:*

  $$\{p, appl(r_1), bodytrue(r_1), bodytrue(r_2), prefer(r_1, r_2), is\_preferred(r_1, r_2)\}$$
  $$\{s, appl(r_2), bodytrue(r_1), bodytrue(r_2), prefer(r_1, r_2), is\_preferred(r_1, r_2)\}$$

- $\gamma_2(P_2) := hr(P_2) \cup \{ \quad \leftarrow \textit{not } 2\{appl(R)\}2 \quad \}.$

  *The program is inconsistent. In fact, of the only two cr-rules in $P_2$, one is preferred to the other, and the constraint added to $hr(P_2)$ by item (6) of Definition 3.1 prevents the application of two cr-rules if one of them is preferred to the other.*

Intuitively, the task of generating a *new* view at each execution of line 4 of the algorithm can be accomplished, with $\gamma_i(\Pi)$, by keeping track of the answer sets of $\gamma_i(\Pi)$ found so far and by adding suitable constraints to prevent them from being generated again. More precisely, for each answer set $M$ that has already been found, we need a constraint:

$$\leftarrow \lambda(M), \nu(M).$$

where $\lambda(M)$ is the list of the literals that occur in $M$ and $\nu(M)$ is a list not $l_1$,not $l_2,\ldots,$not $l_k$ containing all the literals from the signature of $hr(\Pi)$ that do not belong to $M$. Let $U$ be the set of the constraints for all the answer sets that have already been found. It is not difficult to see that the answer sets of the program:

$$\gamma_i(\Pi) \cup U$$

correspond exactly to the "new" answer sets of $\gamma_i(\Pi)$.

## 3.3 The Test Step

The test step of the algorithm (line 5 from Figure 1) checks whether a view $\mathscr{V}$ found during the generation step is a candidate answer set of $\Pi$. Let $M$ be the answer set corresponding to $\mathscr{V}$. The test is reduced to checking whether a suitable A-Prolog program is consistent. The A-Prolog program is called the *tester* for $M$ w.r.t $\Pi$, and is defined below.

**Definition 3.3 (Tester for $M$ w.r.t. $\Pi$, $\tau(M,\Pi)$)** *Let $\Pi$ be a CR-Prolog program and $M$ be an answer set corresponding to a view $\mathscr{V}$ of $\Pi$. The* tester *for $M$ w.r.t. $\Pi$, $\tau(M,\Pi)$, contains:*

1. *The hard reduct of $\Pi$.*
2. *For each atom $appl(r) \in M$, a rule:*

$$o\_appl(r)\cdot$$

3. *For each atom $is\_preferred(r_1,r_2) \in M$, a rule:*

$$o\_is\_preferred(r_1,r_2)\cdot$$

4. *The rules:*

$$\begin{aligned} dominates \leftarrow\ & appl(R_1), o\_appl(R_2), \\ & is\_preferred(R_1,R_2), o\_is\_preferred(R_1,R_2)\cdot \\ \leftarrow\ & \textit{not}\ dominates\cdot \end{aligned}$$

Intuitively, relations $o\_appl$ and $o\_is\_preferred$ are used to store information about which cr-rules have been applied to obtain $M$ and which preferences hold in the model. The first rule of item (4) above embodies the conditions of Definition 2.6, while the constraint enforces Definition 2.7.

The following is a list of important properties of $\tau(M,\Pi)$ (see Section 4):

- If $M$ does not contain any atom formed by $appl$, $\tau(M,\Pi)$ is inconsistent.
- Every answer set of $\tau(M,\Pi)$ contains an answer set of $hr(\Pi)$ (they differ only by the atoms formed by relations $o\_appl$, $o\_is\_preferred$, and $dominates$).

- $M'$ is an answer set of $\tau(M,\Pi)$ iff the view corresponding to $M'$ dominates the view encoded by $M$.
- $\tau(M,\Pi)$ is inconsistent iff there exists no view of $\Pi$ that dominates the view, $\mathscr{V}$, encoded by $M$ (i.e. $\mathscr{V}$ is a candidate answer set according to Definition 2.7).

**Example 3.3** *Consider program $P_2$ from Example 3.1 and the answer set, $M$, of $\gamma_i(\Pi)$:*

$$\{s, appl(r_2), bodytrue(r_1), bodytrue(r_2), prefer(r_1,r_2), is\_preferred(r_1,r_2)\}\cdot$$

*The tester for $M$ w.r.t. $P_2$, $\tau(M,P_2)$ consists of $hr(P_2)$ together with (the constraint from item (4) of Definition 3.3 has been grounded for sake of clarity):*

$$\begin{cases} o\_appl(r_2)\cdot \\ o\_is\_preferred(r_1,r_2)\cdot \\ \\ dominates \leftarrow appl(r_1), appl(r_2), is\_preferred(r_1,r_2), o\_is\_preferred(r_1,r_2)\cdot \\ \leftarrow \textit{not } dominates\cdot \end{cases}$$

*It is not difficult to show that $\tau(M,P_2)$ has a unique answer set:*

$$\{p, appl(r_1), bodytrue(r_1), bodytrue(r_2), prefer(r_1,r_2), is\_preferred(r_1,r_2), \\ o\_appl(r_2), o\_is\_preferred(r_1,r_2), dominates\}$$

*In fact, view $\mathscr{V}_1 = \langle\{s\},\{r_2\}\rangle$ is no a candidate answer set, as it is dominated by $\mathscr{V}_2 = \langle\{p\},\{r_1\}\rangle$. On the other hand, $\tau(M',P_2)$, where $M'$ is the answer set encoding $\mathscr{V}_2$ is inconsistent, implying that $\mathscr{V}_2$ is a candidate answer set.*

## 3.4 The Algorithm

In this section we describe the complete CRMODELS algorithm. We begin by describing CRMODELS$_2$, a detailed version of CRMODELS$_1$. First of all, let us introduce some terminology.

Given an A-Prolog program $\Pi$, the set of the answer sets of $\Pi$ is denoted by $\alpha_*(\Pi)$. We also define an operator $\alpha_1(\Pi)$, which returns non-deterministically one of the answer sets of $\Pi$, or $\bot$ is $\Pi$ is inconsistent.

Recall from Section 3.2 that, given a set of literals $M$ from the signature of $hr(\Pi)$, $\lambda(M)$ denotes the list (as opposed to the set) of the literals that occur in $M$ and $\nu(M)$ is the list not $l_1$,not $l_2,\ldots,$not $l_k$ containing all the literals from the signature of $hr(\Pi)$ that do not belong to $M$.

Algorithm CRMODELS$_2$ is shown in Figure 2 below.

**Algorithm: CRMODELS$_2$**

**input:** $\Pi$: CR-Prolog program
**output:** the answer sets of $\Pi$

var:
$i$: number of cr-rules to be applied
$M$: a set of literals or $\bot$
$C$: a set of constraints

1. $C := \emptyset$
2. $i := 0$ { first we look for an answer set of $reg(\Pi)$ }
3. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
4. *repeat* { **inner loop** }
5. *if* $\gamma_i(\Pi) \cup C$ is inconsistent *then*
6. $M := \bot$
7. *else*
8. $M := \alpha_1(\gamma_i(\Pi) \cup C)$
9. *if* $\tau(M, \Pi)$ is inconsistent *then*
10. *return* $M \cap \Sigma(\Pi)$ { answer set found }
11. *end if*
12. $C := C \cup \{ \leftarrow \lambda(M), \nu(M) \cdot \}$
13. *end if*
14. *until* $M = \bot$
15. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
16. *done*
17. *return* $\bot$ { signal that no answer set was found }

Figure 2: Algorithm CRMODELS$_2$

The algorithm works as follows. At the time of the first execution of line 5, the consistency of $\gamma_0(\Pi)$ is checked ($C$ is $\emptyset$). From the properties of the $i$-generator (see Section 3.2), it follows that $\gamma_0(\Pi)$ is consistent iff $reg(\Pi)$ is consistent. If the test succeeds, $M$ is set to one of the answer sets of $\gamma_0(\Pi)$ and the consistency of $\tau(M, \Pi)$ is tested. Since no cr-rules were used to generate $M$ ($i$ is 0), $\tau(M, \Pi)$ must be inconsistent according to the properties of $\tau(M, \Pi)$ from Section 3.3. Hence, the restriction of $M$ to $\Sigma(\Pi)$ is returned and the algorithm terminates. Notice that the set returned corresponds to an answer set of $reg(\Pi)$, as expected.

If instead $\gamma_0(\Pi)$ is inconsistent, $M$ is set to $\bot$, the inner loop terminates and a new iteration of the outer loop is performed. When line 5 is executed again, $\gamma_1(\Pi)$ is checked for consistency. If the program is inconsistent, the algorithm proceeds to check $\gamma_2(\Pi)$, etc. On the other hand, if $\gamma_1(\Pi)$ is consistent, one of its answer sets is assigned to $M$ and consistency of $\tau(M, \Pi)$ is tested. If the program is inconsistent, it follows that $M$ encodes a candidate answer set (as well as an answer set, as explained at the beginning

of Section 3) and its restriction to $\Sigma(\Pi)$ is returned.

Finally, if instead $\tau(M,\Pi)$ is found to be consistent, the algorithm needs to prevent future computations of the answer sets of $\gamma_1(\Pi) \cup C$ (lines 5 and 8) from considering $M$ again. This is accomplished on line 12 by adding a suitable constraint to set $C$.

We are now ready to present the CRMODELS algorithm. Differently from CRMODELS$_1$

**Algorithm: CRMODELS**

**input:** $\Pi$: CR-Prolog program
**output:** the answer sets of $\Pi$

var:
$i$: number of cr-rules to be applied
$M$: a set of literals or $\bot$
$\mathscr{A}$: a set of answer sets of $\Pi$
$C, C'$: sets of constraints

1. $C := \emptyset$; $\mathscr{A} := \emptyset$
2. $i := 0$ { first we look for an answer set of $reg(\Pi)$ }
3. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
4. $C' := \emptyset$
5. *repeat* { **inner loop** }
6. *if* $\gamma_i(\Pi) \cup C$ is inconsistent *then*
7. $M := \bot$
8. *else*
9. $M := \alpha_1(\gamma_i(\Pi) \cup C)$
10. *if* $\tau(M,\Pi)$ is inconsistent *then* { answer set found }
11. $\mathscr{A} := \mathscr{A} \cup \{M \cap \Sigma(\Pi)\}$
12. $C' := C' \cup \{ \leftarrow \lambda(M \cap atoms(appl, hr(\Pi)))\cdot \}$
13. *end if*
14. $C := C \cup \{ \leftarrow \lambda(M), \nu(M)\cdot \}$
15. *end if*
16. *until* $M = \bot$
17. $C := C \cup C'$
18. $i := i+1$ { consider views obtained with a larger number of cr-rules }
19. *done*
20. *return* $\mathscr{A}$

Figure 3: Algorithm CRMODELS$_2$

and CRMODELS$_2$, CRMODELS computes *all* the answer sets of the program. Informally speaking, to do this we need to store additional information to ensure that the set of cr-rules applied at each generation step is minimal. The complete algorithm is shown in Figure 3.

With respect to CRMODELS$_2$, CRMODELS uses two new data-structures. Set $\mathscr{A}$ contains the answer sets of $\Pi$ found. Set $C'$ has a key role in guaranteeing the minimality of the candidate answer sets identified at step 10. As can be seen from line 12, every time an answer set of $\Pi$ is found, we add to $C'$ a constraint whose body contains the atoms of the form $appl(R)$ that occur in the answer set. The idea is to use $C'$ to prevent any strict superset of the corresponding cr-rules from being applied in the future generation steps (lines 6 and 9). However, particular attention must be paid to the way $C'$ is used, because each constraint in $C'$ can prevent the generation step from using *any* superset of the corresponding cr-rules — *not only the strict supersets.* This would affect the computation when multiple answer sets exist for a fixed choice of cr-rules (see Example 3.4 later). Therefore, the use of the constraints added to $C'$ during one iteration of the outer loop is delayed until the beginning of the following iteration, when the cardinality of the sets of cr-rules considered is increased by 1. This ensures that only the strict supersets of the constraints in $C'$ are considered at all times.

**Example 3.4** *To better understand the issue, consider what would happen, for instance, if we were to replace line* 14 *of the algorithm in Figure 3 with:*

14. $C := C \cup C' \cup \{ \leftarrow \lambda(M), \nu(M). \}$

*and used the resulting algorithm,* CRMODELS$\downarrow$*, to compute the answer sets of the program:*

$$P_3 = \begin{cases} p \leftarrow not\ q, r. \\ q \leftarrow not\ p, r. \\ \\ \leftarrow not\ p, not\ q. \\ \\ r_1 : r \stackrel{+}{\leftarrow}. \end{cases}$$

*Since the regular part of $P_3$ is inconsistent, the first iteration of the outer loop increments $i$, and does not alter the other data-structures. At the next execution of line* 6*, the test succeeds (as the application of $r_1$ makes the program consistent). Let us suppose that at line* 9 *$M$ is set to:*

$$\{p, appl(r_1), bodytrue(r_1)\}.$$

*Since $P_3$ does not contain any preference statements, the test on line* 10 *succeeds, and the following constraint, $c_1$, is added to $C'$:*

$$\leftarrow appl(r_1).$$

*Because of the change to line* 14*, the constraint is immediately added to $C$. Next, a new iteration of the inner loop is performed (because $M \neq \bot$). It is not difficult to see that this time $\gamma_i(P_3) \cup C$ is inconsistent: in fact, applying $r_1$ is the only way to restore consistency of $reg(P_3)$, but doing so is prevented by constraint $c_1$. Therefore, the algorithm terminates returning $\{p, appl(r_1), bodytrue(r_1)\}$ as the unique answer set of $P_3$. The algorithm is of course incomplete, since, according to the semantics of CR-Prolog, $P_3$ has a* second *answer set:*

$$\{q, appl(r_1), bodytrue(r_1)\}.$$

# 4 Properties of the CRMODELS Algorithm

In this section we prove various properties of CRMODELS for arbitrary finite CR-Prolog programs, including the algorithm's termination, soundness, and completeness.

To begin, we summarize some useful properties of A-Prolog programs. All of them are direct applications of the Splitting Set Lemma [20, 28]. Several variants of these properties are present in the literature, e.g. in [10, 3, 14].

**Lemma 1 (Choice Elimination)** *For every A-Prolog program $\Pi$, relation $p$ not occurring in the head of any rule of $\Pi$, and choice macro $\Pi_c$ of the form $\{p(X)\}$, $M$ is an answer set of $\Pi \cup \Pi_c$ iff $M$ is an answer set of $\Pi \cup (M \cap literals(p, \Pi))$.*

Let $literals^{\neg}(p, \Pi)$ denote the negative literals from the signature of $\Pi$ formed by relation $p$ (e.g. $\neg p(t)$). The following holds.

**Lemma 2 (Positive Choice Elimination)** *Let $\Pi$, $p$, and $\Pi_c$ be as in Proposition 1 and $M$ be a set of literals such that $M \cap literals^{\neg}(p, \Pi) = \emptyset$. If no literals from $literals^{\neg}(p, \Pi)$ occur in the rules of $\Pi$, then $M \cup \{\neg p(x) \,|\, p(x) \notin M\}$ is an answer set of $\Pi \cup \Pi_c$ iff $M$ is an answer set of $\Pi \cup (M \cap atoms(p, \Pi))$.*

**Lemma 3 (Constraint Elimination)** *For every A-Prolog program $\Pi$ and set of constraints $\Pi_k$ in the signature of $\Pi$, $M$ is an answer set of $\Pi \cup \Pi_k$ iff $M$ is an answer set of $\Pi$ satisfying the constraints in $\Pi_k$.*

Notice that the above Lemma holds also for cardinality macros.

**Lemma 4 (Definition Elimination)** *Let $\Pi$ be an A-Prolog program, $Q$ be a set of literals not occurring in the head or in the negative part of the body of any rule of $\Pi$, and $\Pi_d$ be a set of rules of the form:*

$$q \leftarrow \Gamma.$$

*where $q \in Q$ and no element of $Q$ occurs in the bodies of the rules of $\Pi_d$. Let $\Pi'$ be obtained from $\Pi$ by replacing every rule of the form*

$$l_0 \leftarrow l_1, \ldots, q, \ldots, l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n.$$

*with a rule:*

$$l_0 \leftarrow l_1, \ldots, \Gamma, \ldots, l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n.$$

*Then, $M$ is an answer set of $\Pi \cup \Pi_d$ iff $M \setminus Q$ is an answer set of $\Pi'$.*

The following lemma is useful in connecting relation $pref$ from Definition 2.4 and relation $is\_preferred$ from the hard reduct (Definition 3.1).

**Lemma 5** *For every CR-Prolog program $\Pi$, answer set $M$ of $hr(\Pi)$, and cr-rules $r_1$, $r_2$ from $\Pi$,*

$$pref_{S(M,\Pi)}(r_1, r_2) \textit{ iff } is\_preferred(r_1, r_2) \in M.$$

Now we prove the termination of the CRMODELS algorithm.

**Lemma 6** *For every CR-Prolog program $\Pi$ and integer $0 \leq j \leq |cr(\Pi)|$, if variable $i$ has value $j$, then the inner loop of* CRMODELS$(\Pi)$ *performs at most $|\alpha_*(\gamma_i(\Pi))|+1$ iterations.*

Proof. *We will prove the claim by showing that, at the end of iteration $|\alpha_*(\gamma_i(\Pi))|+1$ of the inner loop, the termination condition on line* 16 *is true.*

*Given a set of constraints $K = \{c_1, c_2, \ldots, c_m\}$, let $pos(K)$ denote $\{pos(c_1), \ldots, pos(c_m)\}$. Let also $C^k$ denote the value of variable $C$ at the beginning of the $k^{th}$ iteration of the inner loop.*

*It is not difficult to show that an invariant of the inner loop is:*

$$\alpha_*(\gamma_i(\Pi) \cup C^k) = \alpha_*(\gamma_i(\Pi)) \setminus pos(C^k)\cdot \tag{7}$$

*which implies:*

$$|\alpha_*(\gamma_i(\Pi) \cup C^k)| = |\alpha_*(\gamma_i(\Pi))| - |pos(C^k)|\cdot \tag{8}$$

*From the fact that $|pos(C^k)| = |C^k|$, (8) becomes:*

$$|\alpha_*(\gamma_i(\Pi) \cup C^k)| = |\alpha_*(\gamma_i(\Pi))| - |C^k|\cdot \tag{9}$$

*Let $n = |\alpha_*(\gamma_i(\Pi))|$ and assume that, for $i = j$, the inner loop didn't terminate before the $n+1^{th}$ iteration. From (9), it we obtain:*

$$|\alpha_*(\gamma_i(\Pi) \cup C^{n+1})| = n - |C^{n+1}|\cdot \tag{10}$$

*Since a new element is added to variable $C$ during each iteration of the inner loop, it follows that $|C^{n+1}| \geq n$. In other words, for some non-negative integer $h$:*

$$|C^{n+1}| = n + k\cdot \tag{11}$$

*From (10) and (11), we obtain:*

$$|\alpha_*(\gamma_i(\Pi) \cup C^{n+1})| = n - n - k\cdot \tag{12}$$

*Since $k$ is a non-negative integer, (12) implies:*

$$\alpha_*(\gamma_i(\Pi) \cup C^{n+1}) = \emptyset\cdot \tag{13}$$

*From 13 and the definition of operator $\alpha_1$:*

$$\alpha_1(\gamma_i(\Pi) \cup C^{n+1}) \textit{ is inconsistent}\cdot \tag{14}$$

*Hence, the condition of the $if$ statement on line* 6 *is satisfied and $M$ is assigned value $\bot$, thus making the termination condition of the inner loop true.*

$\diamond$

**Lemma 7** *For every CR-Prolog program* $\Pi$*, the inner loop of* CRMODELS($\Pi$) *terminates in exactly* $|cr(\Pi)|+1$ *iterations.*

Proof. *From Lemma 6, it follows that the inner loop terminates for any iteration of the outer loop. Since the value of variable* $i$ *in the algorithm increases by* 1 *after each termination of the inner loop, the termination condition of the outer loop (line* 3*) becomes false in exactly* $|cr(\Pi)|+1$ *iterations.*

◇

**Theorem 1 (Termination of CRMODELS($\Pi$))** CRMODELS($\Pi$) *terminates for any CR-Prolog program* $\Pi$*.*

Proof. *From Lemma 7, the outer loop terminates. When that happens,* CRMODELS($\Pi$) *returns the collection of sets of literals found and terminates.*

◇

To prove soundness and completeness, we introduce the following terminology and lemmas. Given a ground term $r$, $\rho(r,\Pi)$ denotes the cr-rule from $\Pi$ whose name is $r$. Also,

$S(M,\Pi) = M \cap literals(\Pi)$, and

$R(M,\Pi) = \{\rho(r,\Pi) \mid appl(r) \in M\}$.

**Lemma 8** *For every CR-Prolog program* $\Pi$*, if* $M$ *is an answer set of* $\gamma_i(\Pi)$*, then* $|R(M,\Pi)| = i$*.*

Proof. *By Lemma 3,* $M$ *satisfies the cardinality macro (see Definition 3.2):*

$$K = \{\ \leftarrow \textit{not}\ i\{appl(R)\}i. \tag{15}$$

*Hence, the body of* $K$ *is not be satisfied by* $M$*. By definition of the cardinality macros,* $M$ *does not satisfy the body of* $K$ *if* $|M \cap atoms(appl,\Pi)| = i$*. From the definition of* $R(M,\Pi)$*, it follows that* $|R(M,\Pi)| = i$*.*

◇

**Lemma 9** *For every CR-Prolog program* $\Pi$ *and integer* $0 \leq i \leq |cr(\Pi)|$*, if* $M$ *is an answer set of* $\gamma_i(\Pi)$*, then* $\langle S(M,\Pi), R(M,\Pi)\rangle$ *is a view of* $\Pi$*.*

Proof. *We need to prove that* $\langle S(M,\Pi), R(M,\Pi)\rangle$ *satisfies the conditions of Definition 2.5. Let us start by proving that item (1) holds, i.e. that*

$$S(M,\Pi) \textit{ is an answer set of } reg(\Pi) \cup \theta(R(M,\Pi)). \tag{16}$$

*Let* $T_1 = atoms(bodytrue, hr(\Pi))$*,* $K_1$ *the set of rules:*

$$\begin{cases} bodytrue(r) \leftarrow l_1,\ldots,\textit{not}\ l_n. \\ \leftarrow bodytrue(R), appl(R). \\ \\ \leftarrow \textit{not}\ i\{appl(R)\}i. \end{cases}$$

*from Definitions 3.1 and 3.2. From the hypothesis that* $M \in \alpha_*(\gamma_i(\Pi))$*, and Lemmas 4 and 3, it follows that, if* $M \in \alpha_*(\gamma_i(\Pi))$*, then*

$$(M \setminus T_1) \in \alpha_*(\gamma_i(\Pi) \setminus K_1)\cdot \tag{17}$$

*Let now* $T_2 = T_1$ *and* $K_2^-$ *be obtained from* $K_1$ *by the addition of the rule*

$$\{appl(R)\}\cdot$$

*from Definition 3.1 and* $K_2^+ = M \cap atoms(appl, hr(\Pi))$*. From (17) and Lemma 1, it follows that:*

$$(M \setminus T_2) \in \alpha_*(\gamma_i(\Pi) \setminus K_2^- \cup K_2^+)\cdot \tag{18}$$

*Next, let* $T_3 = T_2 \cup atoms(is\_preferred, hr(\Pi))$*,* $K_3^+ = K_2^+$*, and* $K_3^-$ *be* $K_2^-$ *together with rules (again, from Definition 3.1):*

$$\begin{cases} is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_2)\cdot \\ is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_3), is\_preferred(R_3, R_2)\cdot \\ \\ \leftarrow appl(R_1), appl(R_2), is\_preferred(R_1, R_2)\cdot \end{cases}$$

*From (18) and Lemmas 4 and 3, it follows that:*

$$(M \setminus T_3) \in \alpha_*(\gamma_i(\Pi) \setminus K_3^- \cup K_3^+)\cdot \tag{19}$$

*Finally, let* $T_4$ *be* $T_3 \cup atoms(appl, hr(\Pi))$*. From (19) and Lemma 4,*

$$(M \setminus T_4) \in \alpha_*(reg(\Pi) \cup \theta(R(M, \Pi)))\cdot \tag{20}$$

*Since* $M \setminus T_4 = S(M, \Pi)$*, (16) is proven.*

*Next, we prove that* $\langle S(M, \Pi), R(M, \Pi)\rangle$ *satisfies condition (2) of Definition 2.5. More precisely, we need to show that:*

$$\forall r_1, r_2 \;\; pref_{S(M,\Pi)}(r_1, r_2) \rightarrow \{\rho(r_1, \Pi), \rho(r_2, \Pi)\} \not\subseteq R(M, \Pi)\cdot \tag{21}$$

*Let us begin by proving that:*

$$\forall r_1, r_2 \;\; is\_preferred(r_1, r_2) \in M \rightarrow \{\rho(r_1, \Pi), \rho(r_2, \Pi)\} \not\subseteq M\cdot \tag{22}$$

*Consider the constraint from Definition 3.1:*

$$\leftarrow appl(R_1), appl(R_2), is\_preferred(R_1, R_2)\cdot$$

*Since* $M$ *is an answer set of* $\gamma_i(\Pi)$*, it is closed under the constraint. Hence, the following holds:*

$$\forall r_1, r_2 \;\; is\_preferred(r_1, r_2) \in M \rightarrow \{appl(r_1), appl(r_2)\} \not\subseteq M\cdot \tag{23}$$

*Equation (22) follows from (23), and the definitions of* $\rho(r, \Pi)$ *and* $R(M, \Pi)$*. Finally, from (22) and Lemma 4, we obtain (21).*

*To conclude, we prove that* $\langle S(M,\Pi), R(M,\Pi)\rangle$ *satisfies condition (3) of Definition 2.5, that is:*

$$\forall \rho \;\; \rho \in R(M,\Pi) \rightarrow S(M,\Pi) \models body(\rho)\cdot \tag{24}$$

*Consider the constraint:*

$$\leftarrow bodytrue(R), appl(R)\cdot \tag{25}$$

*from Definition 3.1. Since* $M$ *is closed under (25),*

$$\forall r \;\; appl(r) \in M \rightarrow bodytrue(r) \in M\cdot \tag{26}$$

*Consider now the rule:*

$$bodytrue(r) \leftarrow l_1,\ldots,\textit{not } l_n\cdot \tag{27}$$

*from Definition 3.1, where* $l_1,\ldots,$*not* $l_n$ *is the body of* $\rho(r,\Pi)$*. Since (27) is the only definition of* $bodytrue(r)$ *and* $M$ *is closed under (27), it follows that:*

$$\forall r \;\; bodytrue(r) \in M \textit{ iff } S(M,\Pi) \models body(\rho(r,\Pi))\cdot \tag{28}$$

*From (26) and (28) we obtain:*

$$\forall r \;\; appl(r) \in M \rightarrow S(M,\Pi) \models body(\rho(r,\Pi))\cdot \tag{29}$$

*Equation (24) follows from (29) and the definitions of* $\rho(r,\Pi)$ *and* $R(M,\Pi)$*.*

$\diamond$

**Lemma 10** *For every CR-Prolog program* $\Pi$*, if* $\langle \mathscr{S},\mathscr{R}\rangle$ *is a view of* $\Pi$*, then there exist* $M$ *and* $0 \leq i \leq |cr(\Pi)|$ *such that:*

1. $S(M,\Pi) = \mathscr{S}$*,* $R(M,\Pi) = \mathscr{R}$*, and*
2. $M$ *is an answer set of* $\alpha_*(\gamma_i(\Pi))$*.*

Proof. *Let* $i = |\mathscr{R}|$ *and* $M$ *be the union of* $\mathscr{S}$ *with:*

- $\{is\_preferred(r_1,r_2) \,|\, pref_{\mathscr{S}}(r_1,r_2)\}$*;*
- $\{appl(r) \,|\, \rho(r,\Pi) \in \mathscr{R}\} \cup \{\neg appl(r) \,|\, \rho(r,\Pi) \notin \mathscr{R}\}$*;*
- $\{bodytrue(r) \,|\, \mathscr{S} \models body(\rho(r,\Pi))\}$*.*

*We will prove that* $M$ *and* $i$ *satisfy conditions (1) and (2) above.*

*To begin, notice that* $S(M,\Pi) = \mathscr{S}$ *and* $R(M,\Pi) = \mathscr{R}$ *by construction of* $M$*. Hence, condition (1) is satisfied.*

*Next, we prove that condition (2) is satisfied. According to item (1) of Definition 2.5, the following holds:*

$$\mathscr{S} \in \alpha_*(reg(\Pi) \cup \theta(\mathscr{R}))\cdot \tag{30}$$

*Consider the following sets of rules from Definitions 3.1 and 3.2:*

$$K_1 = \left\{ \begin{array}{l} \leftarrow \textit{not } i\{appl(R)\}i. \end{array} \right.$$

$$K_2 = \left\{ \begin{array}{l} bodytrue(r) \leftarrow l_1, \ldots, \textit{not } l_n. \\ \leftarrow bodytrue(R), appl(R). \end{array} \right.$$

$$K_3 = \left\{ \begin{array}{l} \{appl(R)\}. \end{array} \right.$$

$$K_4 = \left\{ \begin{array}{l} is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_2). \\ is\_preferred(R_1, R_2) \leftarrow prefer(R_1, R_3), is\_preferred(R_3, R_2). \\ \\ \leftarrow appl(R_1), appl(R_2), is\_preferred(R_1, R_2). \end{array} \right.$$

*Let* $T_1 = \{appl(r) \,|\, \rho(r, \Pi) \in \mathscr{R}\} \cup \{\neg appl(r) \,|\, \rho(r, \Pi) \notin \mathscr{R}\}$. *From (30) and Lemma 4, it follows that:*

$$(\mathscr{S} \cup T_1) \in \alpha_*(\gamma_i(\Pi) \setminus (K_1 \cup K_2 \cup K_3 \cup K_4) \cup T_1). \tag{31}$$

*Let* $T_2 = \{is\_preferred(r_1, r_2) \,|\, pref_{\mathscr{S}}(r_1, r_2)\}$. *From (31) and Lemmas 3 and 4, we obtain:*

$$(\mathscr{S} \cup T_1 \cup T_2) \in \alpha_*(\gamma_i(\Pi) \setminus (K_1 \cup K_2 \cup K_3) \cup T_1). \tag{32}$$

*From (32) and Lemma 1, we conclude:*

$$(\mathscr{S} \cup T_1 \cup T_2) \in \alpha_*(\gamma_i(\Pi) \setminus (K_1 \cup K_2)). \tag{33}$$

*Now, let* $T_3 = \{bodytrue(r) \,|\, \mathscr{S} \models body(\rho(r, \Pi))\}$. *Then, (33) and Lemmas 4 and 3, imply:*

$$(\mathscr{S} \cup T_1 \cup T_2 \cup T_3) \in \alpha_*(\gamma_i(\Pi) \setminus K_1). \tag{34}$$

*Notice that* $|(\mathscr{S} \cup T_1 \cup T_2 \cup T_3) \cap atom(appl, hr(\Pi))| = i$ *by construction. Hence, from (34) and Lemma 3, we obtain:*

$$(\mathscr{S} \cup T_1 \cup T_2 \cup T_3) \in \alpha_*(\gamma_i(\Pi)). \tag{35}$$

*Since* $M = \mathscr{S} \cup T_1 \cup T_2 \cup T_3$ *by construction, we have proven that* $M \in \alpha_*(\gamma_i(\Pi))$.

$\diamond$

**Lemma 11** *For every CR-Prolog program* $\Pi$, *if* $M_1$ *is an answer set of* $\gamma_i(\Pi)$ *and* $\tau(M_1, \Pi)$ *is inconsistent, then* $\langle S(M_1, \Pi), R(M_1, \Pi)\rangle$ *is a candidate answer set of* $\Pi$.

Proof. *By Lemma 9,* $\mathscr{V}_1 = \langle S(M_1, \Pi), R(M_1, \Pi)\rangle$ *is a view of* $\Pi$. *Next, we prove by contradiction that* $\mathscr{V}_1$ *is a candidate answer set of* $\Pi$. *Let us assume that* $\mathscr{V}_1$ *is not a*

*candidate answer set and let us prove that the hypothesis of inconsistency of $\tau(M_1, \Pi)$ is contradicted.*

*By Definition 2.7, it follows that there exists a view, $\mathcal{V}_2$, of $\Pi$ such that:*

$$\mathcal{V}_2 \text{ dominates } \mathcal{V}_1. \tag{36}$$

*By applying Lemma 10 to $\mathcal{V}_2$, we obtain that:*

$$\exists M_2, i_2 \text{ s.t. } S(M_2, \Pi) = \mathcal{V}_2^S \wedge R(M_2, \Pi) = \mathcal{V}_2^R \wedge M_2 \in \alpha_*(\gamma_{i_2}(\Pi)). \tag{37}$$

*Notice that, from Definition 3.2 and Lemma 3, it follows that:*

$$M_2 \in \alpha_*(hr(\Pi)). \tag{38}$$

*We use $M_2$ to construct an answer set of $\tau(M_1, \Pi)$. Let $M_3$ consist of the union of $M_2$ and:*

- $\{o\_appl(r) \,|\, appl(r) \in M_1\}$;
- $\{o\_is\_preferred(r_1, r_2) \,|\, is\_preferred(r_1, r_2) \in M_1\}$;
- $\{dominates\}$.

*Let also $K$ denote the set of rules:*

$$\begin{aligned} dominates \leftarrow\ & appl(R_1), o\_appl(R_2), \\ & is\_preferred(R_1, R_2), o\_is\_preferred(R_1, R_2). \\ \leftarrow\ & \text{not } dominates. \end{aligned}$$

*From (38), Definition 3.3, and the construction of $M_3$, it is not difficult to see that $M_3 \setminus \{dominates\}$ is an answer set of $\tau(M_1, \Pi) \setminus K$. Hence, to prove that $M_3$ is an answer set of $\tau(M_1, \Pi)$, we need to show that $M_3$ is closed under the rules in $K$ and minimal w.r.t. set-theoretic inclusion.*

*That $M_3$ is closed under the rules in $K$ follows directly from its construction: in fact, the head of the first rule of $K$ is satisfied, and the body of the second rule is not.*

*Since $M_3 \setminus \{dominates\}$ is an answer set of $\tau(M_1, \Pi) \setminus K$, we prove the minimality of $M_3$ by showing that there is at least one ground instance of the first rule of $K$ whose body is satisfied by $M_3 \setminus \{dominates\}$.*

*By (36) and Definition 2.6:*

$$\exists r_1, r_2 \text{ s.t. } \rho(r_1, \Pi) \in \mathcal{V}_1^R \wedge \rho(r_2, \Pi) \in \mathcal{V}_1^R \wedge pref_{(\mathcal{V}_1^S \cap \mathcal{V}_2^S)}(r_2, r_1). \tag{39}$$

*From (39) and (37), it follows that:*

$$\begin{aligned} \exists r_1, r_2 \text{ s.t. } & \rho(r_1, \Pi) \in R(M_1, \Pi) \wedge \rho(r_2, \Pi) \in R(M_2, \Pi) \wedge \\ & pref_{(S(M_1, \Pi) \cap S(M_2, \Pi))}(r_2, r_1). \end{aligned} \tag{40}$$

*From (40) and the definition of $R(M, \Pi)$, we have:*

$$\begin{aligned} \exists r_1, r_2 \text{ s.t. } & appl(r_1) \in M_1 \wedge appl(r_2) \in M_2 \wedge \\ & pref_{(S(M_1, \Pi) \cap S(M_2, \Pi))}(r_2, r_1). \end{aligned} \tag{41}$$

*By construction of $M_3$, (41) implies:*

$$\exists r_1, r_2 \text{ s.t. } \{o\_appl(r_1), appl(r_2)\} \subseteq M_3 \wedge pref_{(S(M_1,\Pi)\cap S(M_2,\Pi))}(r_2, r_1). \quad (42)$$

*Notice that $M_1$ and $M_2$ are both answer sets of $hr(\Pi)$. Then, by (42) and Lemma 4 we have:*

$$\forall r_1, r_2 \;\; pref_{(S(M_1,\Pi)\cap S(M_2,\Pi))}(r_2, r_1) \text{ iff } is\_preferred(r_2, r_1) \in (M_1 \cap M_2). \quad (43)$$

*Hence, from (42), (43), and the construction of $M_3$, we obtain:*

$$\begin{array}{l}\exists r_1, r_2 \text{ s.t. } \{o\_appl(r_1), appl(r_2)\} \subseteq M_3 \wedge \\ \qquad \{is\_preferred(r_2, r_1), o\_is\_preferred(r_2, r_1)\} \subseteq M_3.\end{array} \quad (44)$$

*Equation (44) proves that there exists at least one instance of the first rule of $K$ whose body is satisfied. Hence, $M_3$ is minimal.*

*This concludes the proof that $M_3$ is an answer set of $\tau(M_1, \Pi)$. But the conclusion contradicts the hypothesis that $\tau(M_1, \Pi)$ is inconsistent. Therefore, $\langle S(M_1, \Pi), R(M_1, \Pi)\rangle$ is a candidate answer set of $\Pi$.*

$\diamond$

**Lemma 12** *For every CR-Prolog program $\Pi$, if $\langle \mathscr{S}, \mathscr{R}\rangle$ is a candidate answer set of $\Pi$, then there exist $M$ and $0 \leq i \leq |cr(\Pi)|$ such that:*

1. *$S(M, \Pi) = \mathscr{S}$, $R(M, \Pi) = \mathscr{R}$,*
2. *$M$ is an answer set of $\gamma_i(\Pi)$, and*
3. *$\tau(M, \Pi)$ is inconsistent.*

Proof. *Conditions (1) and (2) follow from Definition 2.7 and Lemma 10. Now let us prove that $\tau(M, \Pi)$ is inconsistent.*

*Let $\mathscr{V}_1$ denote $\langle \mathscr{S}, \mathscr{R}\rangle$. Since $\mathscr{V}_1$ is a candidate answer set of $\Pi$, from Definition 2.7 it follows that, for every view $\mathscr{V}_2$ of $\Pi$*

$$\mathscr{V}_2 \text{ does not dominate } \mathscr{V}_1. \quad (45)$$

*By Lemmas 10 and 9, for each view $\mathscr{V}_2$ of $\Pi$ there exist exactly one pair $M_2$, $i_2$ such that:*

$$\mathscr{V}_2 = \langle S(M_2, \Pi), R(M_2, \Pi)\rangle \;\wedge\; M_2 \in \alpha_*(\gamma_{i_2}(\Pi)). \quad (46)$$

*From (45) and (46), we obtain that, for every $i_2$ and answer set $M_2$ of $\gamma_{i_2}(\Pi)$:*

$$\langle S(M_2, \Pi), R(M_2, \Pi)\rangle \text{ does not dominate } \langle S(M, \Pi), R(M, \Pi)\rangle. \quad (47)$$

*From Definition 2.6, it follows that, for each $M_2$:*

$$\begin{array}{l}\neg\exists r_1, r_2 \text{ s.t. } \rho(r_1, \Pi) \in R(M, \Pi) \wedge \rho(r_2, \Pi) \in R(M_2, \Pi) \wedge \\ \qquad pref_{(S(M,\Pi)\cap S(M_2,\Pi))}(r_2, r_1).\end{array} \quad (48)$$

*By the definitions of $S(M,\Pi)$ and $R(M,\Pi)$, and Lemma 4, we obtain:*

$$\neg\exists r_1, r_2 \text{ s.t. } appl(r_1) \in M \wedge appl(r_2) \in M_2 \wedge \\ is\_preferred(r_2, r_1) \in (M \cap M_2)\cdot \tag{49}$$

*Let $K_1$ denote the constraint (see Definition 3.3):*

$$K_1 = \left\{ \leftarrow \text{not } dominates\cdot \right. \tag{50}$$

*and let $\tau^-$ denote $\tau(M,\Pi) \setminus K_1$.*

*Recall that, by construction, for every answer set $M_3$ of $\tau^-$, if $appl(r) \in M_1$, then $o\_appl(r) \in M_3$, and similarly for $is\_preferred(r_1, r_2)$ and $o\_is\_preferred(r_1, r_2)$. Also, notice that every $M_2$ is an answer set of $hr(\Pi)$, and $\tau^-$ is obtained from $hr(\Pi)$ by the addition of the rule:*

$$K_2 = \left\{ \begin{array}{l} dominates \leftarrow appl(R_1), o\_appl(R_2), \\ \qquad\qquad is\_preferred(R_1, R_2), o\_is\_preferred(R_1, R_2)\cdot \end{array} \right. \tag{51}$$

*and of appropriate definitions of $o\_appl$ and $o\_is\_preferred$. Then, by Lemma 4, for each $M_2$ there exists one and only one $M_3 \supseteq M_2$ such that $M_3 \in \alpha_*(\tau^-)$.*

*Hence, (49) becomes:*

$$\neg\exists r_1, r_2 \text{ s.t. } o\_appl(r_1) \in M_3 \wedge appl(r_2) \in M_3 \wedge \\ o\_is\_preferred(r_2, r_1) \in M_3 \wedge \\ is\_preferred(r_2, r_1) \in M_3\cdot \tag{52}$$

*Equation (52) implies that the body of rule (51) is not satisfied by any answer set of $\tau^-$. Hence, the answer sets of $\tau^-$ do not satisfy constraint (50). Since $\tau^- = \tau(M,\Pi) \setminus K_1$, from Lemma 3 it follows that $\tau(M,\Pi)$ has no answer set.*

$\diamond$

We are now ready to prove the two main theorems of this section. In the following discussion, we use CRMODELS($\Pi$) to denote the collection of sets of literals returned by the CRMODELS algorithm with CR-Prolog program $\Pi$ in input. Notice that Theorem 1 guarantees the existence of such set for any $\Pi$.

**Theorem 2 (Soundness)** *For every CR-Prolog program $\Pi$, if $J \in$ CRMODELS($\Pi$), then $J$ is an answer set of $\Pi$.*

Proof. *In the rest of this proof, $V^i$ denotes the value of variable $V$, from the algorithm, at the beginning of iteration $i$ of the* outer loop *(see Figure 3). When we need to refer to the value of $V$ at the beginning of a specific iteration of the inner loop, we use the notation $V^i_j$.*

*Let $n = |cr(\Pi)|$. By Lemma 7, the iteration indexes will range between 1 and $n+1$.*

*Let* $J \in \text{CRMODELS}(\Pi)$. *From the final step of the algorithm (line* 20 *in Figure 3), it follows that:*

$$J \in \mathscr{A}^{n+1}. \tag{53}$$

*The only update to* $\mathscr{A}$ *in the algorithm occurs on line* 11*, and consists of the addition of a new set of literals. The set of literals used for the update is computed on line* 9*. By observing line* 9*, we conclude that there exist iterations* $i$ *and* $j$*, of the outer and inner loops respectively, and* $M$*, such that* $J = M \cap \Sigma(\Pi)$*, and:*

$$M \in \alpha_*(\gamma_i(\Pi) \cup C_j^i)\cdot \tag{54}$$

*By (54), Lemma 3, and the observation that the contents of* $C$ *monotonically increases:*

$$M \in \alpha_*(\gamma_i(\Pi) \cup C^i), \tag{55}$$

$$M \in \alpha_*(\gamma_i(\Pi))\cdot \tag{56}$$

*From (53) and the observation that the condition of step* 10 *must be true, we obtain:*

$$\tau(M,\Pi) \text{ is inconsistent.} \tag{57}$$

*From (55), (57), and Lemma 11, it follows that:*

$$\langle S(M,\Pi), R(M,\Pi)\rangle \text{ is a candidate answer set of } \Pi\cdot \tag{58}$$

*Notice that* $S(M,\Pi) = J$*. To prove that* $J$ *is an answer set of* $\Pi$*, we need to prove that condition 2 of Definition 2.8 is satisfied, i.e. that:*

$$\text{for every candidate answer set } \langle S', R'\rangle \text{ of } \Pi, R' \not\subset R(M,\Pi)\cdot \tag{59}$$

*Proceeding by contradiction, let us assume the existence of a candidate answer set* $\langle S', R'\rangle$ *of* $\Pi$ *such that:*

$$R' \subset R(M,\Pi)\cdot \tag{60}$$

*By Lemma 12, there exist* $M'$*,* $0 \leq i' \leq |cr(\Pi)|$ *such that:* $S(M',\Pi) = S'$*,* $R(M',\Pi) = R'$*, and:*

$$S(M',\Pi) = S', R(M',\Pi) = R', \tag{61}$$

$$M' \in \alpha_*(\gamma_{i'}(\Pi)), \tag{62}$$

$$\tau(M',\Pi) \text{ is inconsistent.} \tag{63}$$

*We now prove that the existence of* $M'$ *contradicts the hypothesis that* $J \in \text{CRMODELS}(\Pi)$*. To do this, we show that* $M$ *cannot have been computed by line* 9*.*

*Let us begin by noticing that, from (62), (63), and line* 12 *of the algorithm:*

$$(C')^{i'+1} \supseteq \{\leftarrow \lambda(M' \cap atoms(appl, hr(\Pi)))\cdot\}\cdot \tag{64}$$

*Because of line* 17*, the following also holds:*

$$C^{i'+1} \supseteq \{\leftarrow \lambda(M' \cap atoms(appl, hr(\Pi)))\cdot\}\cdot \tag{65}$$

*By (65) and Lemma 3, the sets of literals computed at line* 9 *after iteration* $i'$ *of the outer loop do not contain* $M' \cap atoms(appl, hr(\Pi))$*. However, from (60), it follows that* $M$ *contains* $M' \cap atoms(appl, hr(\Pi))$*. Then, what we need to prove to have a contradiction is* $i > i'$*.*

*From (61) and (60), it follows that:*

$$|R(M', \Pi)| < |R(M, \Pi)|\cdot \tag{66}$$

*From (55), (62), and Lemma 8,*

$$|R(M', \Pi)| = i' \wedge |R(M, \Pi)| = i\cdot \tag{67}$$

*Equations (66) and (67) imply:*

$$i' < i\cdot \tag{68}$$

*Hence,* $M$ *does not satisfy* $C^i$*. By Lemma 3:*

$$M \notin \alpha_*(\gamma_i(\Pi) \cup C^i), \tag{69}$$

*which contradicts (55).*

$\diamond$

**Theorem 3 (Completeness)** *For every CR-Prolog program* $\Pi$*, if* $J$ *is an answer set of* $\Pi$*, then* $J \in \text{CRMODELS}(\Pi)$*.*

Proof. *As before,* $V^i$ *denotes the value of variable* $V$ *at the beginning of iteration* $i$ *of the outer loop, at* $V^i_j$ *denotes the value of* $V$ *at the beginning of iteration* $j$ *of the inner loop, during iteration* $i$ *of the outer loop.*

*Since* $J$ *is an answer set of* $\Pi$*, from Definition 2.8, we know that:*

$$\exists R_J \subseteq cr(\Pi) \textit{ s.t. } \langle J, R_J\rangle \textit{ is a candidate answer set of } \Pi\cdot \tag{70}$$

*By (70) and Lemma 12, there exist* $M$ *and* $i$ *such that:*

$$S(M) = J \wedge R(M) = R_J, \tag{71}$$

$$M \in \alpha_*(\gamma_i(\Pi)), \tag{72}$$

$$\tau(M, \Pi) \textit{ is inconsistent}\cdot \tag{73}$$

*Now let us prove that* $M$ *is also an answer set of* $\gamma_i(\Pi) \cup C^i_j$ *for some iteration* $j$ *of the inner loop. Let us proceed by contradiction, and assume that:*

$$\forall j\ M \notin \alpha_*(\gamma_i(\Pi) \cup C^i_j)\cdot \tag{74}$$

*where* $j$ *is an iteration of the inner loop.*

*By construction of the algorithm, $C^i = C^i_j$. Hence, (74) implies:*

$$M \notin \alpha_*(\gamma_i(\Pi) \cup C^i)\cdot \tag{75}$$

*By (75), (73), and Lemma 3, $M$ does not satisfy the constraints in $C^i$. By inspection of the algorithm (lines* 12*, and* 17*) we conclude that there exist $M'$ and $i' < i$ such that:*

$$M' \in \alpha_*(\gamma_{i'}(\Pi)), \tag{76}$$

$$R(M',\Pi) \subseteq R(M,\Pi), \tag{77}$$

$$\tau(M',\Pi) \textit{ is inconsistent}\cdot \tag{78}$$

*By (76), (78), and Lemma 11,*

$$\langle S(M',\Pi), R(M',\Pi)\rangle \textit{ is a candidate answer set of } \Pi\cdot \tag{79}$$

*From (77), Lemma 8, and $i' < i$, it follows that:*

$$R(M',\Pi) \subset R(M,\Pi)\cdot \tag{80}$$

*From (79), (80), and condition 2 of Definition 2.8, we conclude that $J$ is not an answer set of $\Pi$. Contradiction. Hence there exists an iteration $j$ of the inner loop such that:*

$$M \textit{ is an answer set of } \gamma_i(\Pi) \cup C^i_j\cdot \tag{81}$$

*For simplicity, let us assume that $M = \alpha_1(\gamma_i(\Pi) \cup C^i_j)$ (it this is not the case, it is not difficult to prove that the statement becomes true for some $j' > j$). Since $\tau(M,\Pi)$ is inconsistent (see (73), from steps* 10 *and* 11 *of the algorithm, it follows that $J \in \mathscr{A}^{i+1}$. Since $\mathscr{A}$ grows monotonically and, by Theorem 1, the algorithm terminates, $J \in$* CRMODELS$(\Pi)$.

$\diamond$

# 5 Implementing CRMODELS

In this section, we describe interesting issues involved in the implementation of CRMODELS, which for clarity we will refer to as CRMODELS$^I$.

Notice that the computation of answer sets (lines 6, 9, and 10 from Figure 3) is by far the most demanding task of the entire algorithm. Hence, reducing the time spent computing answer sets is critical to improve the practical applicability of CRMODELS$^I$.

To accomplish this, we have refined the implementation of the algorithm in two directions:

- Reducing the time spent in each computation of answer sets.
- Reducing the number of overall computations of answer sets.

The first goal can be achieved by storing and re-using the results of earlier computations. The second goal can be achieved by employing more sophisticated search techniques.

Let us now discuss the issue of storing and re-using earlier results. We begin by giving some background information on the implementation of the task of finding answer sets of A-Prolog programs. In CRMODELS$^I$, the computation of answer sets of A-Prolog programs is performed by calls to a state-of-the-art A-Prolog inference engine consisting of the pair of programs LPARSE [27] and SMODELS [21, 26, 22]. We refer to the inference engine as LPARSE–SMODELS, and, whenever possible, we abbreviate it to SMODELS.

Like all state-of-the-art inference engines for A-Prolog, SMODELS works by first computing the ground instance[2] of the program in input, and then finding the answer sets of the ground program. In the case of LPARSE–SMODELS pair, LPARSE grounds the input, while SMODELS computes the answer sets.

Let us now turn our attention to the time spent computing answer sets in a straightforward implementation of the CRMODELS algorithm (Figure 3). At each iteration of the inner loop, if the condition of line 6 is satisfied, CRMODELS makes three calls to both LPARSE and SMODELS (lines 6, 9, and 10). As a first obvious improvement, the computation at line 9 can be easily removed by caching the result from line 6.

An important observation that allows us to improve efficiency further is that programs $\gamma_i(\Pi)$ and $\tau(M,\Pi)$ do not differ much from each other. In fact, both are obtained by adding a few rules to $hr(\Pi)$ (see Sections 3.2 and 3.3), while $hr(\Pi)$ *itself does not change throughout the algorithm.* A variation of the main algorithm that is based on this observation is shown in Figure 4 below and is called CRMODELS$^I_1$. In the algorithm, $\gamma_i^{HR}(\Pi)$ denotes the program obtained from $\gamma_i(\Pi)$ by replacing $hr(\Pi)$ by its grounding $HR$. Similarly, $\tau^{HR}(M,\Pi)$ indicates the replacement of $hr(\Pi)$ in $\tau(M,\Pi)$. The call to the inference engine to test the consistency of $\tau^{HR}(M,\Pi)$ (step 9) has been made explicit for sake of clarity.

[2] Obtained by replacing the variables in the program with all the possible variable-free terms.

**Algorithm: CRMODELS$_1^I$**

**input:** $\Pi$: CR-Prolog program
**output:** the answer sets of $\Pi$

var:
$i$: number of cr-rules to be applied
$M$: a set of literals or $\bot$
$\mathscr{A}$: a set of answer sets of $\Pi$
$C, C'$: sets of constraints
$HR$: grounding of $hr(\Pi)$

1. $C := \emptyset$; $\mathscr{A} := \emptyset$
2. $HR := ground(hr(\Pi))$ { first we ground $hr(\Pi)$ }
3. $i := 0$ { first we look for an answer set of $reg(\Pi)$ }
4. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
5. $C' := \emptyset$
6. *repeat* { **inner loop** }
7. $M := \alpha_1(\gamma_i^{HR}(\Pi) \cup C)$
8. *if* $M \neq \bot$ *then*
9. *if* $\alpha_1(\tau^{HR}(M,\Pi)) = \bot$ *then* { answer set found }
10. $\mathscr{A} := \mathscr{A} \cup \{M \cap \Sigma(\Pi)\}$
11. $C' := C' \cup \{ \leftarrow \lambda(M \cap atoms(appl, hr(\Pi)))\cdot \}$
12. *end if*
13. $C := C \cup \{ \leftarrow \lambda(M), \nu(M)\cdot \}$
14. *end if*
15. *until* $M = \bot$
16. $C := C \cup C'$
17. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
18. *done*
19. *return* $\mathscr{A}$

Figure 4: Algorithm CRMODELS$_1^I$

Notice that both $\gamma_i^{HR}(\Pi)$ and $\tau^{HR}(M,\Pi)$ contain non-ground rules (see Sections 3.2 and 3.3). Hence, the computations at steps 7 and 9 still involve grounding, but on a significantly smaller program. To take advantage of the situation, we can use the "-g" option of LPARSE. This option allows the user to load a previously grounded program and add new (possibly non-ground) rules to it.

**Example 5.1** *Consider the following program, $P_4$.*

$p(1)\cdot \quad p(2)\cdot$
$q(X) \leftarrow p(X), \mathit{not}\ r(X)\cdot$

*Its ground instance, $P_4^g$, consists of the rules:*

$$p(1)\cdot \quad p(2)\cdot$$
$$q(1) \leftarrow p(1), \textit{not } r(1)\cdot$$
$$q(2) \leftarrow p(2), \textit{not } r(2)\cdot$$

*and can be computed*[3] *by the executing "*`lparse` $P_4$*".*

*Suppose now we would like to compute the ground instance of $P_4$ together with the following constraint, $K$:*

$$\leftarrow q(X), X > 1, \textit{not } d(X)\cdot$$

*Obviously, this can be done by invoking* LPARSE *on $P_4 \cup K$. However, if $P_4^g$ is already available, we can obtain the same result by executing "*`lparse -g` $P_4^g$ $K$*." The command line tells* LPARSE *to combine a ground program, $P_4^g$, with a possibly non-ground set of rules, $K$.*

In conclusion, lines 7 and 9 of Figure 4 can be implemented by calls to LPARSE of the form "`lparse -g` $HR\ \Pi_N$" where $\Pi_N$ is the set of rules to be added to $HR$ according to Sections 3.2 and 3.3 respectively. Unfortunately, experiments showed that LPARSE does not perform well on partially grounded programs. A possible explanation is that, although a relatively small amount of time is spent processing the previously grounded rules, the large number of them causes the total processing time to raise unacceptably.

A more efficient way to store and re-use the grounding of $hr(\Pi)$ is to use a specialized grounding algorithm for partially-ground programs, to be used at steps 7 and 9 instead of LPARSE. The specialized algorithm that we have developed is called SGA, and takes as input a ground program and a (possibly non-ground) set of rules, and returns the ground instance of the union of the two sets of rules.

Intuitively, given a program $\Pi$ and a (possibly non-ground) set of rules to be added to it, $\Pi_N$, SGA is applicable under the following conditions:

1. The heads of the rules of $\Pi_N$ are either empty (i.e. the rules are constraints), or fresh atoms.
2. The (non-empty) heads of the rules of $\Pi_N$ with non-empty body have arity 0.
3. The literals in the bodies of the rules of $\Pi_N$ with arity greater than 0, either are obtained from the signature of $\Pi$ or are facts in $\Pi_N$.

More precisely:

**Proposition 1** *Let $ground(\Pi)$ denote the grounding of $\Pi$. For every set of rules $\Pi$, $\Pi_N$, satisfying conditions (1)–(3):*

$$\text{SGA}(ground(\Pi), \Pi_N) = ground(\Pi \cup \Pi_N)\cdot \tag{82}$$

[3] For simplicity, in this example we ignore the simplifications that LPARSE performs on the program to improve the efficiency of the inference engine.

It is not difficult to show that the sets of rules added to $hr(\Pi)$ to obtain $\gamma_i(\Pi) \cup C$ and $\tau(M, \Pi)$ satisfy the premise of Proposition 1.

For efficiency, SGA contains a specific grounding procedure for each type of rule added to $hr(\Pi)$ by $\gamma_i(\Pi) \cup C$ and $\tau(M, \Pi)$. The approach is pretty straightforward and, to save space, we illustrate it by means of an example. Consider the following rule from $\tau(M, \Pi)$ (Section 3.3):

$$dominates \leftarrow appl(R_1), o\_appl(R_2), \\ is\_preferred(R_1, R_2), o\_is\_preferred(R_1, R_2).$$

Let us assume that atoms $o\_appl(r)$ and $o\_is\_preferred(r_1, r_2)$ from Section 3.3 have already been added to the grounding. Then, the grounding procedure for the rule above is:

1. Extract from $ground(hr(\Pi))$ all the ground instances of $is\_preferred(R_1, R_2)$.
2. For every corresponding instantiation of variables $R_1$ and $R_2$, $\langle r_1, r_2 \rangle$, generate the ground instance:

$$dominates \leftarrow appl(r_1), o\_appl(r_2), \\ is\_preferred(r_1, r_2), o\_is\_preferred(r_1, r_2).$$

In the actual implementation of SGA, the ground instances are encoded directly in the LPARSE output language [27], where ground literals are replaced by the corresponding indexes in the symbol table associated with the program. To merge the ground and non-ground sets of rules, SGA first augments the symbol table, for example with a new entry for atom $dominates$. Next, if $i_0$, $i_1$, $i_2$, $i_3$, $i_4$ are, respectively, the indexes of atoms $dominates$, $appl(r_1), \ldots, o\_is\_preferred(r_1, r_2)$ in the symbol table, then one of the ground instances of the rule above, generated by the implementation of SGA, is represented by the string:

$$1 \quad i_0 \quad 4 \quad 0 \quad i_1 \quad i_2 \quad i_3 \quad i_4$$

which denotes a "type 1 rule" (a conventional A-Prolog rule), with the literal with index $i_0$ (i.e. $dominates$) in the head, and 4 literals in the body, of which 0 under default negation, corresponding to indexes $i_1, \ldots, i_4$ (for more details on the LPARSE output format, the reader is invited to refer to [27]).

As expected, experiments showed that SGA is substantially more efficient than LPARSE in processing partially-ground programs. Figure 5 shows the algorithm, called CRMODELS$_2^I$, resulting from the adoption of SGA. CRMODELS$_2^I$ and CRMODELS$_1^I$ also differ in that the calls to $\alpha_1$ have been replaced by explicit calls to SMODELS and SGA. More precisely, SMODELS$_1$ denotes an invocation of the SMODELS algorithm to compute a single answer set and SGA$(HR, P)$ denotes a call to SGA with arguments $HR$ and $P \setminus HR$.

**Algorithm:** CRMODELS$_2^I$

**input:** $\Pi$: CR-Prolog program
**output:** the answer sets of $\Pi$

var:
$i$: number of cr-rules to be applied
$M$: a set of literals or $\bot$
$\mathscr{A}$: a set of answer sets of $\Pi$
$C, C'$: sets of constraints
$HR$: grounding of $hr(\Pi)$

1. $C := \emptyset$; $\mathscr{A} := \emptyset$
2. $HR :=$ LPARSE$(hr(\Pi))$ { first we ground $hr(\Pi)$ }
3. $i := 0$ { first we look for an answer set of $reg(\Pi)$ }
4. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
5. $C' := \emptyset$
6. *repeat* { **inner loop** }
7. $M :=$ SMODELS$_1$(SGA$(HR, \gamma_i^{HR}(\Pi) \cup C)$)
8. *if* $M \neq \bot$ *then*
9. *if* SMODELS$_1$(SGA$(HR, \tau^{HR}(M, \Pi))) = \bot$ *then*
{ answer set found }
10. $\mathscr{A} := \mathscr{A} \cup \{M \cap \Sigma(\Pi)\}$
11. $C' := C' \cup \{ \leftarrow \lambda(M \cap atoms(appl, hr(\Pi))) \cdot \}$
12. *end if*
13. $C := C \cup \{ \leftarrow \lambda(M), \nu(M) \cdot \}$
14. *end if*
15. *until* $M = \bot$
16. $C := C \cup C'$
17. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
18. *done*
19. *return* $\mathscr{A}$

Figure 5: Algorithm CRMODELS$_2^I$

The other direction taken to improve the efficiency of the implementation of CRMODELS consists in reducing the number of overall computations of answer sets by employing more sophisticated search techniques.

If we look at CRMODELS$_2^I$ from the standpoint of the search technique used, it is easy to see that the algorithm uses linear search over the number of cr-rules applied. Intuitively, performance could be improved by adopting a more sophisticated search strategy, such as binary search.

Unfortunately binary search is not suitable for a direct modification of the outer loop, because the sequentiality of the search procedure is essential to ensure the minimality

of the candidate answer sets found at line 9 of CRMODELS$_2^I$ (for details, refer to the final part of the proof of Theorem 2). However, binary search can be used to determine the smallest number of cr-rules for which a view exists (right now the search begins by setting $i = 0$ at line 3). After the number has been determined, linear search can be used. To employ binary search effectively, the following conditions must be met:

- We need to be able to determine the existence of a view whose number of cr-rules is in a specified *interval.*
- The time taken to look for a view in an interval must be roughly independent of the interval examined.

To meet the first condition, let us consider the program $\eta_{\langle j,i\rangle}(\Pi)$, obtained from $\gamma_i(\Pi)$ by replacing the rule (see Section 3.2):

$$\leftarrow \text{not } i\{appl(R)\}i.$$

with:

$$\leftarrow \text{not } j\{appl(R)\}i.$$

The answer sets of $\eta_{\langle j,i\rangle}(\Pi)$ have the following important property.

**Proposition 2** *For every CR-Prolog program* $\Pi$ *and integers* $0 \leq j \leq i \leq |cr(\Pi)|$*, if* $M$ *is an answer set of* $\eta_{\langle j,i\rangle}(\Pi)$*, then* $\langle S(M,\Pi), R(M,\Pi)\rangle$ *is a view of* $\Pi$ *and* $j \leq |R(M,\Pi)| \leq i$.

The proof of the proposition is a simple extension of the proofs of Lemmas 9 and 8.

According to Proposition 2, $\eta_{\langle j,i\rangle}(\Pi)$ can be used to meet the first condition above. The second condition is also met: experimental results in fact show that, when the input to CRMODELS$_2^I$ is a program of medium to large size, the time taken to compute one answer set of $\eta_{\langle j,i\rangle}(\Pi)$ is, within certain limits, independent of $j$ and $i$.

Figure 6 shows CRMODELS$^I$, the final version of the implementation of CRMODELS. CRMODELS$^I$ extends CRMODELS$_2^I$ by the addition of binary search. In the algorithm, $\eta^{HR}_{\langle j,i\rangle}(\Pi)$ denotes, as usual, the program obtained from $\eta_{\langle j,i\rangle}(\Pi)$ by replacing $hr(\Pi)$ with $HR$.

**Algorithm: CRMODELS$^I$**

**input:** $\Pi$: CR-Prolog program
**output:** the answer sets of $\Pi$

var:
$i$: number of cr-rules to be applied
$M$: a set of literals or $\bot$
$\mathscr{A}$: a set of answer sets of $\Pi$
$C, C'$: sets of constraints
$HR$: grounding of $hr(\Pi)$
$i_1, i_2$: binary search interval

1. $C := \emptyset$; $\mathscr{A} := \emptyset$
2. $HR :=$ LPARSE$(hr(\Pi))$ { first we ground $hr(\Pi)$ }
4. $i_1 := 0$; $i_2 := |cr(\Pi)|$
5. *while* $(i_1 < i_2)$ { **binary search loop** }
6. $i := (i_1 + i_2)/2$
7. $M :=$ SMODELS$_1$(SGA$(HR, \eta^{HR}_{\langle i_1, i\rangle}(\Pi))$)
8. *if* $M \neq \bot$ *then*
9. $i_2 := |M \cap atom(appl, hr(\Pi))| - 1$
10. *else*
11. $i_1 := i + 1$
12. *end if*
13. *done*
14. $i := i_1$ { The termination condition above implies $i_1 = i_2$ }
14. *while* $(i \leq |cr(\Pi)|)$ *do* { **outer loop** }
15. $C' := \emptyset$
16. *repeat* { **inner loop** }
17. $M :=$ SMODELS$_1$(SGA$(HR, \gamma^{HR}_i(\Pi) \cup C)$)
18. *if* $M \neq \bot$ *then*
19. *if* SMODELS$_1$(SGA$(HR, \tau^{HR}(M, \Pi))) = \bot$ *then* { answer set found }
20. $\mathscr{A} := \mathscr{A} \cup \{M \cap \Sigma(\Pi)\}$
21. $C' := C' \cup \{ \leftarrow \lambda(M \cap atoms(appl, hr(\Pi)))\cdot \}$
22. *end if*
23. $C := C \cup \{ \leftarrow \lambda(M), \nu(M)\cdot \}$
24. *end if*
25. *until* $M = \bot$
26. $C := C \cup C'$
27. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
28. *done*
29. *return* $\mathscr{A}$

Figure 6: Algorithm CRMODELS$^I$

It is interesting to notice that step 9 above is slightly non-standard, as $i_2$ is not assigned $i-1$. Step 9 exploits the fact that, when at step 7 an answer set is found, information becomes available about the number of cr-rules applied to obtain the answer set. Since this value can be easily shown to be less than or equal to $i$, it can be used in the update of $i_2$ and obtain a possibly smaller search interval.

The performance improvement yielded by the introduction of binary search is particularly evident when a large number of cr-rules needs to be applied before a view of $\Pi$ can be found, and when $\Pi$ is inconsistent. In particular, the inconsistency of a program with $n$ cr-rules can be detected in $log(n)$ calls to SMODELS, while with CRMODELS$_2^I$ the corresponding number of calls to SMODELS is $n$.

# 6 Related Work

There are no previous published results on the design and implementation of an inference engine for CR-Prolog. However, this paper builds on years of research on the topic, which resulted in various prototypes. In particular, here we extend previous work by Loveleen Kolvekar [19], where the first complete description of the CRMODELS algorithm was given. The algorithm presented here is a substantial simplification of the one from [19], with performance improvements that reduced the computation time of several orders of magnitude. We also made several improvements (in particular in terms of accuracy) and simplifications to the theoretical results.

# 7 Conclusions

In this paper we have described our design and implementation of an inference engine for CR-Prolog. The inference engine is aimed at allowing practical applications of CR-Prolog that require the efficient computation of the answer sets of medium-size programs.

The efficiency of CRMODELS has been demonstrated experimentally on 2000 planning problems by using a modified version of the experiment from [23]. The modification consisted in replacing the A-Prolog planning module from [23] with a CR-Prolog based module capable of finding plans that satisfy (if at all possible) 3 sets of non-trivial requirements, aimed at improving plan quality. The planning module has been tested both with and without preferences on the sets of requirements. The experiments have been successful (refer to [5] for a more detailed discussion of experiments and results): the average time to find a plan was about 200 seconds, against an average time of 10 seconds for the original A-Prolog planner[4], with an increase of about one order of magnitude in spite of the substantially more complex reasoning task (the quality of plans increased, depending on the parameters used to measure it, between 19% and 96%). Moreover, the average time obtained with the CR-Prolog planner was substantially lower than the limit of practical use by NASA for this application, which is 20 minutes.

[4] All the experiments were run on the same computer.

The implementation described in this paper is available for download from `http://www.krlab.cs.ttu.edu/Software/`, and corresponds to version 1.5 of the system.

# 8 Acknowledgments

The author would like to thank Michael Gelfond for his help. This research was partially supported by United Space Alliance under contract number NAS9-20000 and by NASA under contract number NASA-NNG05GP48G.

# References

[1] Marcello Balduccini. USA-Smart: Improving the Quality of Plans in Answer Set Planning. In *PADL'04*, Lecture Notes in Artificial Intelligence (LNCS), Jun 2004.

[2] Marcello Balduccini. *Answer Set Based Design of Highly Autonomous, Rational Agents.* PhD thesis, Texas Tech University, Dec 2005.

[3] Marcello Balduccini and Michael Gelfond. Diagnostic reasoning with A-Prolog. *Journal of Theory and Practice of Logic Programming (TPLP)*, 3(4–5):425–461, Jul 2003.

[4] Marcello Balduccini and Michael Gelfond. Logic Programs with Consistency-Restoring Rules. In Patrick Doherty, John McCarthy, and Mary-Anne Williams, editors, *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, pages 9–18, Mar 2003.

[5] Marcello Balduccini, Michael Gelfond, and Monica Nogueira. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence*, 2006. (to appear).

[6] Marcello Balduccini and Veena S. Mellarkod. CR-Prolog2: CR-Prolog with Ordered Disjunction. In *ASP03 Answer Set Programming: Advances in Theory and Implementation*, volume 78 of *CEUR Workshop proceedings*, Sep 2003.

[7] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving.* Cambridge University Press, Jan 2003.

[8] Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation. *Journal of Logic Programming*, 19(20):73–148, 1994.

[9] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Journal of Theory and Practice of Logic Programming (TPLP)*, 2005. (submitted).

[10] S. Brass and J. Dix. A Characterization of the Stable Semantics by Partial Evaluation. In *Proceedings of the 10th Workshop on Logic Programming*, Oct 1994.

[11] Gerhard Brewka, Ilkka Niemela, and Tommi Syrjanen. Logic Programs wirh Ordered Disjunction. 20(2):335–357, 2004.

[12] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Adding Weak Constraints to Disjunctive Datalog. In *Proceedings of the 1997 Joint Conference on Declarative Programming APPIA-GULP-PRODE'97*, 1997.

[13] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerard Pfeifer. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 03)*. Morgan Kaufmann, Aug 2003.

[14] Selim Erdogan and Vladimir Lifschitz. Definitions in answer set programming. In *Proceedings of LPNMR-7*, Jan 2004.

[15] Michael Gelfond. Representing Knowledge in A-Prolog. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408, pages 413–451. Springer Verlag, Berlin, 2002.

[16] Michael Gelfond. Going places - notes on a modular development of knowledge about travel. In *AAAI Spring 2006 Symposium on Knowledge Repositories*, 2006.

[17] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–385, 1991.

[18] M. Kaminski. A note on the stable model semantics of logic programs. *Artificial Intelligence*, 96(2):467–479, 1997.

[19] Loveleen Kolvekar. Developing an Inference Engine for CR-Prolog with Preferences. Master's thesis, Texas Tech University, Dec 2004.

[20] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming (ICLP94)*, pages 23–38, 1994.

[21] Ilkka Niemela and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence (LNCS)*, pages 420–429, 1997.

[22] Ilkka Niemela and Patrik Simons. *Extending the Smodels System with Cardinality and Weight Constraints*, pages 491–521. Logic-Based Artificial Intelligence. Kluwer Academic Publishers, 2000.

[23] Monica Nogueira. *Building Knowledge Systems in A-Prolog*. PhD thesis, University of Texas at El Paso, May 2003.

[24] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the Space Shuttle. In *PADL 2001*, pages 169–183, 2001.

[25] Raymond Reiter. *On Closed World Data Bases*, pages 119–140. Logic and Data Bases. Plenum Press, 1978.

[26] Patrik Simons. Extending the Stable Model Semantics with More Expressive Rules. In *Proceedings of the 5th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR-99)*, number 1730 in Lecture Notes in Artificial Intelligence (LNCS). Springer Verlag, Berlin, 1999.

[27] Tommi Syrjanen. Implementation of logical grounding for logic programs with stable model semantics. Technical Report 18, Digital Systems Laboratory, Helsinki University of Technology, 1998.

[28] Hudson Turner. Splitting a Default Theory. In *Proceedings of AAAI-96*, pages 645–651, 1996.