CR-MODELS: An Inference Engine for CR-Prolog

Marcello Balduccini

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

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 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 [8] – was shown to be a useful tool for knowledge representation and reasoning (e.g. [7,5]). 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, aimed at improving event further the expressive power of the language.

One of these extensions, called CR-Prolog [3], 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 [3,1,6,4], with examples including planning and diagnostic reasoning. For example, in [3], cr-rules have been used to model exogenous actions that may occur, unobserved, and cause malfunctioning in a physical system. In [1,4], 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.

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 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 the description of the algorithm of the inference engine. Finally, in Section 4 we talk about related work and 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 tasks that are problematic to encode in A-Prolog.

The syntax of CR-Prolog is determined by a typed signature Σ 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 Σ we mean its integer constants. By *complex arithmetic terms* of Σ 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 1. *A* regular rule ρ *is a statement of the form:*

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

where r is a term that uniquely denotes ρ (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 ... OR h_k the head *of the rule (head(r)); $l_1,l_2,\ldots l_m,\mathit{not}\ l_{m+1},\mathit{not}\ l_{m+2},\ldots,\mathit{not}\ l_n$ 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. *A* consistency-restoring rule *(or* cr-rule*) is a statement of the form:*

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

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 3. *A* CR-Prolog program *is a pair $\langle \Sigma, \Pi \rangle$, where Σ is a typed signature and Π 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 Π_1 and Π_2.

The terms, atoms and literals of a program Π 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 Π 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, Π, the *regular part* of Π is the set of its regular rules, and is denoted by $reg(\Pi)$. The set of cr-rules of Π is denoted by $cr(\Pi)$.

Example 1.

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

The regular part of the program (consisting of the last two rules) 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 $\{s,q\}$.[1]

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 1.

[1] The examples in this section are only aimed at illustrating the features of the language, and not its usefulness. Please refer to e.g. [3,1] for more comprehensive examples.

Example 2.

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

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.

As usual, we assume that rules containing variables are shorthands for the sets of their ground instances

Now we define the semantics of CR-Prolog. In the following discussion, Π 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: (1) l is an arithmetic literal and is true according to the standard arithmetic interpretation; or (2) 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 4. For every set of literals, S, from the signature of Π, and every r_1,r_2 from $cr(\Pi)$, $pref_S(r_1, r_2)$ is true iff (1) $prefer(r_1, r_2) \in S$, or (2) 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 3. 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 5. Let $S \subseteq literals(\Pi)$ and $R \subseteq cr(\Pi)$. $\mathscr{V} = \langle S, R \rangle$ is a *view* of Π 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 4. Consider the program, P_1:

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

The regular part of the program is inconsistent. According to Definition 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):

$$\begin{array}{ll} \mathscr{V}_1 = \langle \{t\}, \{r_1\} \rangle & \mathscr{V}_2 = \langle \{t, q\}, \{r_1, r_4\} \rangle \\ \mathscr{V}_3 = \langle \{s\}, \{r_3\} \rangle & \mathscr{V}_4 = \langle \{s, q\}, \{r_3, r_4\} \rangle \\ \mathscr{V}_5 = \langle \{p, q\}, \{r_2, r_4\} \rangle & \mathscr{V}_6 = \langle \{s, p, q\}, \{r_2, r_3, r_4\} \rangle \\ \mathscr{V}_7 = \langle \{t, p, q\}, \{r1, r_2, r_4\} \rangle & \end{array}$$

The second step in the definition of the answer sets of Π 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 6. For every pair of views of Π, $\mathscr{V}_1$ 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 5. Let us consider the views of program P_1 from Example 4. 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 7. A view, $\mathscr{V}$, is a candidate answer set of Π if, for every view $\mathscr{V}'$ of Π, $\mathscr{V}'$ does not dominate $\mathscr{V}$.

Example 6. According to the conclusions from Example 4, $\mathscr{V}_3$ 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: $\mathscr{V}_1 = \langle\{\{t\},\{r_1\}\rangle$ $\mathscr{V}_2 = \langle\{\{t,q\},\{r_1,r_4\}\rangle$ $\mathscr{V}_5 = \langle\{\{p,q\},\{r_2,r_4\}\rangle$ $\mathscr{V}_7 = \langle\{\{t,p,q\},\{r1,r_2,r_4\}\rangle$.

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 8. A set of literals, S, is an *answer set* of Π if:

1. there exists $R \subseteq cr(\Pi)$ such that $\langle S,R\rangle$ is a candidate answer set of Π, and
2. for every candidate answer set $\langle S',R'\rangle$ of Π, $R' \not\subset R$.

Example 7. Consider $\mathscr{V}_1$ and $\mathscr{V}_2$ from the list of the candidate answer sets of P_1 from Example 6. Since $\mathscr{V}_1^R \subseteq \mathscr{V}_2^R$, $\mathscr{V}_2$ is not an answer set of P_1. According to Definition 8, the answer sets of P_1 are: $\mathscr{V}_1 = \langle\{\{t\},\{r_1\}\rangle$ $\mathscr{V}_5 = \langle\{\{p,q\},\{r_2,r_4\}\rangle$.

It is worth pointing out how the above definitions deal with cyclic preferences. For simplicity, let us focus on static preferences. Let r be a cr-rule that occurs in the preference cycle. It is not difficult to see that, for any view $\mathscr{V}$, $pref_{(\mathscr{V}^S \cap \mathscr{V}^S)}(r,r)$ holds. This prevents any view where r is used from being a candidate answer set. Hence, the cr-rules involved in preference cycle cannot be used to restore consistency.

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 Π can be computed as show below (Figure 1). Notice that, in the algorithm, $\perp$ is used to indicate the absence of a solution. The algorithm begins by looking for a view $\mathscr{V}$ such that $|\mathscr{V}^R| = 0$. If one is found, CRMODELS$_1$ checks that $\mathscr{V}$ is a candidate answer set of Π (line 5). Notice that, because $|\mathscr{V}^R| = 0$, the condition of Definition 6 is never satisfied (as there is no $r \in \mathscr{V}^R$). 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 Π according to Definition 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 $\perp$, 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 Π contains at least one cr-rule, execution goes back to line 4, where a view $\mathscr{V}$ with

Algorithm: CRMODELS$_1$
input: Π: CR-Prolog program
output: one answer set of Π
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 Π s.t. $|\mathcal{V}^R| = i$; if none is found, $\mathcal{V} := \bot$
5. *if* $\mathcal{V}$ is candidate answer set of Π *then* { test fails if $\mathcal{V} = \bot$ }
6. *return* $\mathcal{V}^S$ { answer set found }
7. *end if*
8. *until* $\mathcal{V} = \bot$
9. $i := i + 1$ { consider views obtained with a larger number of cr-rules }
10. *done*
11. *return* $\bot$ { signal that no answer set was found }

Figure 1. Algorithm CRMODELS$_1$

$|\mathcal{V}^R| = 1$ is computed. It is important to notice[2] 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 8, $\mathcal{V}^S$ is an answer set of Π. 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: $\{p(X)\}$. informally says that any X can have property p, and stands for the rules: $p(X) \leftarrow \text{not } \neg p(X)$. $\neg p(X) \leftarrow \text{not } p(X)$.
- A macro-rule of the form: $\leftarrow \text{not } i\{p(X)\}j$. 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: $X_k \neq X_h$ for every k, h such that $1 \leq k \leq m, 1 \leq h \leq m, k \neq h$. The macro-rule stands for:

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

We call the former a *choice macro* and the latter a *cardinality macro*. These macros allow for compact programs without committing to a particular extension of A-Prolog (and to its inference engine). Moreover, the structure of the macros is simple enough

[2] A refinement of this statement is proven in [2].

to allow their translation, at the time of the implementation of the algorithm, to more efficient expressions, specific of the inference engine used.

Central to the execution of steps 5 and 6 of the algorithm is the notion of *hard reduct*. The hard reduct of a CR-Prolog program Π, denoted by $hr(\Pi)$, maps Π 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 Π and the answer sets of $hr(\Pi)$* [2].

The signature of $hr(\Pi)$ is obtained from the signature of Π 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 Π. We also assume that the signature of Π 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 9 (Hard Reduct of Π). *Let Π be a CR-Prolog program. The hard reduct of Π, $hr(\Pi)$, consists of:*

1. *Every regular rule from Π.*
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 OR ... OR $h_k \leftarrow l_1,\ldots l_m, not\ l_{m+1},\ldots,not\ l_n, appl(r)$. and $bodytrue(r) \leftarrow l_1,\ldots l_m, not\ l_{m+1},\ldots,not\ l_n$.*
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 5):*

$$\leftarrow not\ bodytrue(R), appl(R).$$

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

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

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 5):*

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

Example 8. Let us compute the hard reduct of the following program, P_2:

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

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 $\{p \leftarrow \text{not } q, appl(r_1).\ \ bodytrue(r_1) \leftarrow \text{not } q.\}$. For r_2, $hr(P_2)$ contains $\{s \leftarrow appl(r_2).\ \ bodytrue(r_2).\}$. Items (3 – 6) result in the addition of the rules:

$$\{appl(R)\}. \qquad \leftarrow \text{not } bodytrue(R), appl(R).$$
$$is\_preferred(R_1,R_2) \leftarrow prefer(R_1,R_2). \qquad \leftarrow appl(R_1), appl(R_2), is\_preferred(R_1,R_2).$$
$$is\_preferred(R_1,R_2) \leftarrow prefer(R_1,R_3), is\_preferred(R_3,R_2).$$

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

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

corresponding to the views $\mathscr{V}_1 = \langle\{p, prefer(r_1,r_2)\},\{r_1\}\rangle, \mathscr{V}_2 = \langle\{s, prefer(r_1,r_2)\},\{r_2\}\rangle$.

In the generation step of the algorithm (line 4 from Figure 1), we find a view $\mathscr{V}$ of Π 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* Π, $\gamma_i(\Pi)$, defined below.

Definition 10 (i-Generator of Π). *Let Π be a CR-Prolog program, and i a non-negative integer such that $i \leq |cr(\Pi)|$. The* i-generator of Π *is the program:* $hr(\Pi) \cup \{ \leftarrow not\ i\{appl(R)\}i. \}$.

It is not difficult to show that $\gamma_i(\Pi)$ has the following properties [2]: (1) M is an answer set of $\gamma_0(\Pi)$ iff $M \cap \Sigma(\Pi)$ is an answer set of $reg(\Pi)$; (2) Every answer set of $\gamma_i(\Pi)$ is an answer set of $hr(\Pi)$; (3) Every answer set M of $\gamma_i(\Pi)$ contains exactly i atoms of the form $appl(R)$.

Example 9. Consider program P_2 from Example 8. The i-generators for P_2 for various values of i and the corresponding answer sets are as follows:

- $\gamma_0(P_2) = hr(P_2) \cup \{ \leftarrow \text{not } 0\{appl(R)\}0 \}$.
  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.
- $\gamma_1(P_2) = hr(P_2) \cup \{ \leftarrow \text{not } 1\{appl(R)\}1 \}$.
  The program allows the application of 1 cr-rule at a time. Its answer sets are:

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

- $\gamma_2(P_2) = hr(P_2) \cup \{ \leftarrow \text{not } 2\{appl(R)\}2 \}$.
  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 9 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 $\gamma_i(\Pi)$.

The test step of the algorithm (line 5 from Figure 1) checks whether a view $\mathcal{V}$ found during the generation step is a candidate answer set of Π. Let M be the answer set corresponding to $\mathcal{V}$. 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 Π, and is defined below.

Definition 11 (Tester for M w.r.t. Π, $\tau(M,\Pi)$). *Let Π be a CR-Prolog program and M be an answer set corresponding to a view $\mathcal{V}$ of Π. The* tester *for M w.r.t. Π, $\tau(M,\Pi)$, contains:*

1. *The hard reduct of Π.*
2. *For each atom $appl(r) \in M$, a rule: $o\_appl(r)$.*
3. *For each atom $is\_preferred(r_1,r_2) \in M$, a rule: $o\_is\_preferred(r_1,r_2)$.*
4. *The rules:*

$$\begin{array}{l} dominates \leftarrow appl(R_1), o\_appl(R_2), \\ \qquad\qquad is\_preferred(R_1,R_2), o\_is\_preferred(R_1,R_2). \\ \leftarrow not\ dominates. \end{array}$$

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 6, while the constraint enforces Definition 7.

The following is a list of important properties of $\tau(M,\Pi)$ [2]: (1) If M does not contain any atom formed by $appl$, $\tau(M,\Pi)$ is inconsistent; (2) 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 $dominates$); (3) M' is an answer set of $\tau(M,\Pi)$ iff the view corresponding to M' dominates the view encoded by M; (4) $\tau(M,\Pi)$ is inconsistent iff there exists no view of Π that dominates the view, $\mathcal{V}$, encoded by M (i.e. $\mathcal{V}$ is a candidate answer set according to Definition 7).

Example 10. Consider program P_2 from Example 8 and the answer set, M, of $\gamma_i(\Pi)$:

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

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 11 has been grounded for sake of clarity):

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

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

We can now describe the complete CRMODELS algorithm. We need the following terminology. Given an A-Prolog program Π, the set of the answer sets of Π is denoted by $\alpha_*(\Pi)$. We also define an operator $\alpha_1(\Pi)$, which non-deterministically returns one of the answer sets of Π, or $\bot$ is Π is inconsistent. Recall 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 is shown in Figure 2 below. Notice that, differently from CRMODELS$_1$, CRMODELS *computes all the answer sets of the program*. The answer sets of the program are stored in the set $\mathscr{A}$. The algorithm works as follows. At the time

Algorithm: CRMODELS
input: Π: CR-Prolog program
output: the answer sets of Π
var i: number of cr-rules to be applied
M: a set of literals or $\bot$
$\mathscr{A}$: a set of answer sets of Π
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. $C' := C' \cup \{ \leftarrow \lambda(M \cap atoms(appl, hr(\Pi))). \}$
13. *end if*
14. $C := C \cup \{ \leftarrow \lambda(M), \nu(M). \}$
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 2. Algorithm CRMODELS

of the first execution of line 6, the consistency of $\gamma_0(\Pi)$ is checked (C is $\emptyset$). From the properties of the i-generator, 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 $\tau(M,\Pi)$. Hence, the restriction of M to $\Sigma(\Pi)$ is added to the set of answer sets of Π, $\mathscr{A}$. 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 6 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 6 and 9) from considering M again. This is accomplished on line 14 by adding a suitable constraint to set C.

As the algorithm computes all the answer sets of the program, CRMODELS needs to ensure that the set of cr-rules applied at each generation step is minimal. Set C' has a key role in this. As can be seen from line 12, every time an answer set of Π 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. 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.

Let us stress that the implementation correctly deals with preference cycles, discussed at the end of Section 2: for any cr-rule r from a preference cycle and any M such that $appl(r) \in M$, there always exists an answer set of $\tau(M,\Pi)$ (it contains M itself, together with appropriate definitions of *o_appl* and *o_is_preferred*). Hence, cr-rules from preference cycles cannot be used by the implementation to restore consistency.

The following theorems guarantee termination, soundness, and completeness of CRMODELS. The proofs cannot be shown because of space restrictions, but can be found, together with the description of the implementation of the algorithm, in [2].

Theorem 1. CRMODELS(Π) *terminates for any CR-Prolog program* Π.

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

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

4 Related Work and Conclusions

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. Here we extend previous work by L. Kolvekar [9], where the first description of the CRMODELS algorithm was given. The algorithm and theoretical results presented here are a substantial simplification of the ones from [9].

In this paper we have described our design 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 [10]. The modification consisted in replacing the A-Prolog planning module from [10] 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 [4] 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[3], 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 for practical use by NASA, which is 20 minutes.

The proofs of the theorems in this paper and a discussion on the implementation of CRMODELS can be found in [2]. An implementation of the algorithm is available for download from `http://www.krlab.cs.ttu.edu/Software/`.

The author would like to thank Michael Gelfond for his help. This research was partially supported, over time, by United Space Alliance contract NAS9-20000, NASA contract NASA-NNG05GP48G, and ATEE/DTO contract ASU-06-C-0143.

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. Computing Answer Sets of CR-Prolog Programs. Technical report, Texas Tech University, 2006. http://krlab.cs.ttu.edu/∼marcy/bib.php.
3. 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.
4. Marcello Balduccini, Michael Gelfond, and Monica Nogueira. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence*, 2006.
5. Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, Jan 2003.
6. Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Journal of Theory and Practice of Logic Programming (TPLP)*, 2005. (submitted).
7. 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.
8. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–385, 1991.
9. Loveleen Kolvekar. Developing an Inference Engine for CR-Prolog with Preferences. Master's thesis, Texas Tech University, Dec 2004.
10. Monica Nogueira. *Building Knowledge Systems in A-Prolog*. PhD thesis, University of Texas at El Paso, May 2003.

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