CR-Prolog with Ordered Disjunction

Marcello Balduccini and **Veena Mellarkod**
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
{balduccini, mellarko}@cs.ttu.edu

Abstract

We present CR-Prolog$_2$, an extension of A-Prolog with cr-rules and ordered disjunction. CR-rules can be used to formalize various types of common-sense knowledge and reasoning, that, to the best of our knowledge, have no formalization in A-Prolog. The use of ordered disjunction often allows for a very concise, easy to read, representation of knowledge. We also show how CR-Prolog$_2$ can be used to represent preferences intended both as strict preferences, and as desires.

Introduction

In recent years, A-Prolog – the language of logic programs with the answer set semantics (Gelfond & Lifschitz 1991) – was shown to be a useful tool for knowledge representation and reasoning (Gelfond 2002). The language is expressive and has a well understood methodology of representing defaults, causal properties of actions and fluents, various types of incompleteness, etc. The development of efficient computational systems (Cholewinski, Marek, & Truszczynski 1996; Simons 1996; Calimeri *et al.* 2002; Lin & Zhao 2002; Pontelli, Balduccini, & Bermudez 2003) has allowed the use of A-Prolog for a diverse collection of applications. Some of the applications include: a decision support system for space shuttle flight controllers. The system is planned to be used by flight controllers to find plans for the operation of the Reactive Control System of the space shuttle, as well as checking correctness of existing plans (Nogueira *et al.* 2001; Nogueira 2003); detection of deadlocks using a reduction of the problem to the computation of stable models of logic programs (Heljanko 1999). Other important applications are in planning, product configuration, bounded model checking, wire routing and modeling in hybrid systems etc.

It seems however that A-Prolog lacks the ability to gracefully perform the reasoning needed for certain types of conflict resolution, e.g. for finding the best explanations of unexpected observations. To solve the problem, in (Balduccini & Gelfond 2003b) the authors introduced CR-Prolog – an extension of A-Prolog by *consistency-restoring rules* (cr-rules) with preferences.

In this paper we present CR-Prolog$_2$, a variant of CR-Prolog with an improved semantics, and allowing ordered disjunction (Brewka 2002; Brewka, Niemela, & Syrjanen 2002) in the head of both regular rules and consistency-restoring rules. The new semantics yields intuitive conclusions in cases when CR-Prolog would give unintuitive results. The use of ordered disjunction, when the preference order on a set of alternatives is total, allows for a more concise, easier to read, representation of knowledge. The flexibility of the preference relation in CR-Prolog$_2$ is such that meta-preferences from LPOD (Brewka, Niemela, & Syrjanen 2002) can be encoded in CR-Prolog$_2$ using directly its preference relation, rather than requiring the definition of a new type of preference. We show how CR-Prolog$_2$ can be used to represent preferences intended both as strict preferences (like in CR-Prolog), and as desires (like in LPOD).

The paper is structured as follows. We start with the syntax and semantics of CR-Prolog$_2$. Next, we compare the new language with CR-Prolog and LPOD, and show how the new language can be used to represent complex knowledge and to perform fairly sophisticated reasoning tasks. Finally, we summarize the paper and draw conclusions.

Syntax and Semantics

Let Σ be a signature containing symbols for constants, variables, functions, and predicates (denoted by $const(\Sigma)$, $var(\Sigma)$, $func(\Sigma)$ and $pred(\Sigma)$, respectively). Terms, atoms, and literals are defined as usual. Literals and terms not containing variables are called *ground*. The sets of ground terms, atoms and literals over Σ will be denoted by $terms(\Sigma)$, $atoms(\Sigma)$, and $lit(\Sigma)$.

Definition 1 A *head expression* is either an epistemic disjunction of literals (h_1 or h_2 or $\ldots$ or h_k, with $k \geq 0$) or an ordered disjunction of literals ($h_1 \times h_2 \times \ldots \times h_k$, with $k > 1$).

Definition 2 A *regular* rule of CR-Prolog$_2$ is a statement of the form:

$$r: \quad \mathcal{H} \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n \qquad (1)$$

where $\mathcal{H}$ is a head expression, $l_1, \ldots, l_n$ are literals, and r is a term representing the name of the rule. If $\mathcal{H}$ is an epistemic disjunction, the intuitive reading of the rule is as usual. If $\mathcal{H}$ is an ordered disjunction, the intuitive meaning of the rule is (Brewka, Niemela, & Syrjanen 2002): if the body of the rule is satisfied by the agent's beliefs, then the agent must believe the first (leftmost) element of $\mathcal{H}$, if possible; otherwise it must believe the second element, if possible; ... otherwise, it must believe the last element of $\mathcal{H}$.

For example, program

$$r_1:\ p \text{ or } q \leftarrow \text{not } r.$$

yields two possible conclusions: $\{p\}$ and $\{q\}$. On the other hand, program

$$r_1:\ p \times q \leftarrow \text{not } r.$$

forces the agent to believe p, and program

$$\begin{array}{l} r_1:\ p \times q \leftarrow \text{not } r. \\ r_2:\ s \leftarrow \text{not } s, p, \text{not } r. \end{array}$$

forces the agent to believe q (since believing p is made impossible by the second rule).

Definition 3 A cr-rule is a statement of the form:

$$r:\ \ \mathcal{H} \stackrel{+}{\leftarrow} l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n \tag{2}$$

where r is the name of the rule, $\mathcal{H}$ is a head expression, and $l_1, \ldots, l_n$ are literals. The rule says that if $l_1, \ldots, l_m$ belong to a set of agent's beliefs and none of $l_{m+1}, \ldots, l_n$ belongs to it then the agent "may possibly" believe one of the elements of the head expression. This possibility is used only if the agent has no way to obtain a consistent set of beliefs using regular rules only. If $\mathcal{H}$ is an ordered disjunction, the preference order that the agent uses to select an element from the head expression goes from left to right, as for regular rules. If $\mathcal{H}$ is an epistemic disjunction, then all the elements are equally preferable.

For example, program

$$\begin{array}{l} r_1:\ p \text{ or } q \stackrel{+}{\leftarrow} \text{not } r \\ r_2:\ s. \end{array}$$

only forces the agent to believe s (the cr-rule need not be applied, since the program containing only the second rule is consistent). On the other hand, program

$$\begin{array}{l} r_1:\ p \text{ or } q \stackrel{+}{\leftarrow} \text{not } r \\ r_2:\ s. \\ r_3:\ \leftarrow \text{not } p, \text{not } q. \end{array}$$

forces the agent to believe either $\{s, p\}$ or $\{s, q\}$. If finally we want the agent to prefer conclusion p to conclusion q when possible, we write

$$\begin{array}{l} r_1:\ p \times q \stackrel{+}{\leftarrow} \text{not } r \\ r_2:\ s. \\ r_3:\ \leftarrow \text{not } p, \text{not } q. \end{array}$$

which yields a unique set of beliefs, $\{s, p\}$. Notice though that adding new information to the above program, for example a new rule $\leftarrow p$, forces the agent to retract the previous conclusions, and believe $\{s, q\}$.

We will use the term *rule* to denote both regular rules and cr-rules. As usual, non-ground rules are intended as schemata for their ground counterparts.

Definition 4 *Preferences between cr-rules* are expressed by atoms of the form $prefer(r_1, r_2)$. If all preferences in a program are expressed as facts, we say that the program employs *static preferences*. Otherwise, preferences are *dynamic*.

Definition 5 A *CR-Prolog*$_2$ *program*, Π, is a pair $\langle \Sigma, R \rangle$ consisting of signature Σ and a set R of rules of form (1) or (2). We require that $func(\Sigma)$ does not contain $choice$, and that $pred(\Sigma)$ contains $prefer$ and does not contain $appl$, $fired$, and $is\_preferred$. Signature Σ is denoted by $sig(\Pi)$; $const(\Pi)$, $func(\Pi)$, $pred(\Pi)$, $atoms(\Pi)$ and $lit(\Pi)$ are shorthands for $const(sig(\Pi))$, $func(sig(\Pi))$, $pred(sig(\Pi))$, $atoms(sig(\Pi))$ and $lit(sig(\Pi))$, respectively. Let P be a set of predicate symbols from Σ. By $atoms(\Pi, P)$ we denote the set of all atoms from $atoms(\Pi)$ formed by predicate symbols from P. (Whenever possible we drop the first argument and simply write $atoms(P)$). The set of rules of Π is denoted by $rules(\Pi)$. If ρ is a rule of Π then $head(\rho) = \mathcal{H}$, and $body(\rho) = \{l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n\}$.

Programs of CR-Prolog$_2$ are closely related to abductive logic programs (Kakas & Mancarella 1990; Gelfond 1991) – pairs $\langle \Pi, \mathcal{A} \rangle$ where Π is a program of A-Prolog and $\mathcal{A}$ is a set of atoms, called *abducibles*[1]. The semantics of the language is based on a transformation $hr(\Pi)$ of its programs into abductive programs.

Definition 6 The *hard reduct* $hr(\Pi) = \langle H_\Pi, atoms(H_\Pi, \{appl\}) \rangle$ is defined as follows:

1. *let N be a set of new constant symbols, such that every element of N is uniquely associated with an atom from $sig(\Pi)$. We will denote the constant associated with atom a by n_a;*
2. *$sig(H_\Pi)$ extends $sig(\Pi)$ so that: $const(H_\Pi) = const(\Pi) \cup N$, $func(H_\Pi) = func(\Pi) \cup \{choice\}$, and $pred(H_\Pi) = pred(\Pi) \cup \{appl, fired, is\_preferred\}$.*
3. *let R_Π be set of rules obtained from Π by replacing every cr-rule, ρ, with a rule:*
$$r : head(\rho) \leftarrow\ body(\rho), appl(r)$$
where r is the name of ρ. Notice that R_Π contains only regular rules;
4. the set of rules of H_Π is obtained from R_Π by replacing every rule, ρ, such that $head(\rho) = h_1 \times h_2 \times \ldots \times h_k$, with the following rules: (r is the name of rule ρ)
   (a) $h_i \leftarrow body(\rho), appl(choice(r, n_{h_i}))$ for $1 \leq i \leq k$;
   (b) $fired(r) \leftarrow appl(choice(r, n_{h_i}))$ for $1 \leq i \leq k$;
   (c) $prefer(choice(r, n_{h_i}), choice(r, n_{h_{i+1}}))$ for $1 \leq i < k$;
   (d) $\leftarrow body(\rho), \text{not } fired(r)$.
5. *H_Π also contains the following set of rules, denoted by Π_p:*

$$\left\{ \begin{array}{lll} \multicolumn{3}{l}{\%\ \textit{transitive closure of predicate prefer}} \\ m_{1a}: & is\_preferred(R1, R2) & \leftarrow\ \ prefer(R1, R2). \\ m_{1b}: & is\_preferred(R1, R2) & \leftarrow\ \ prefer(R1, R3), \\ & & \qquad is\_preferred(R3, R2). \\ \multicolumn{3}{l}{\%\ \textit{no circular preferences}} \\ m_2: & \leftarrow is\_preferred(R, R). & \\ \multicolumn{3}{l}{\%\ \textit{prohibit application of } R1 \textit{ and } R2 \textit{ if}} \\ \multicolumn{3}{l}{\%\ R1 \textit{ is preferred to } R2} \\ m_3: & \multicolumn{2}{l}{\leftarrow appl(R1), appl(R2), is\_preferred(R1, R2).} \end{array} \right.$$

[1]Recall that the semantics of an abductive program is given by the notion of *generalized answer set* – an answer set $M(\Delta)$ of $\Pi \cup \Delta$ where $\Delta \subseteq \mathcal{A}$; $M(\Delta_1) < M(\Delta_2)$ if $\Delta_1 \subset \Delta_2$. We refer to an answer set as *minimal* if it is minimal with respect to this ordering.

Let us compute the hard reduct of the following program:

$$\Pi_1 \left\{ \begin{array}{llcllll} r_1: & p & \leftarrow & \text{not } q. & r_5: & & \leftarrow \; p, r. \\ r_2: & r & \leftarrow & \text{not } s. & & & \\ r_3: & q & \leftarrow & t. & r_6: & q \times s \stackrel{+}{\leftarrow}. & \\ r_4: & s & \leftarrow & t. & r_7: & t \stackrel{+}{\leftarrow}. & \end{array} \right.$$

$H_{\Pi_1} = H'_{\Pi_1} \cup \Pi_p$, where H'_{Π_1} is:

$$\left\{ \begin{array}{lrcl} \multicolumn{4}{l}{r_1 : p \leftarrow \text{not } q. \quad r_2 : r \leftarrow \text{not } s. \quad r_3 : q \leftarrow t.} \\ \multicolumn{4}{l}{r_4 : s \leftarrow t. \qquad\quad r_5 : \leftarrow p, r.} \\ & & & \\ r_{6a}^1 : & q & \leftarrow & appl(choice(r_6, n_q)), appl(r_6). \\ r_{6a}^2 : & s & \leftarrow & appl(choice(r_6, n_s)), appl(r_6). \\ r_{6b}^1 : & fired(r_6) & \leftarrow & appl(choice(r_6, n_q)), appl(r_6). \\ r_{6b}^2 : & fired(r_6) & \leftarrow & appl(choice(r_6, n_s)), appl(r_6). \\ r_{6c} : & \multicolumn{3}{l}{prefer(choice(r_6, n_q), choice(r_6, n_s)).} \\ r_{6d} : & & \leftarrow & appl(r_6), \text{not } fired(r_6). \\ r'_7 : & t & \leftarrow & appl(r_7). \end{array} \right.$$

The generalized answer sets of $hr(\Pi_1)$ are: (we omit the atoms formed by $is\_preferred$ and $fired$):

$$C_1 = \{prefer(choice(r_6, n_q), choice(r_6, n_s)), appl(r_6), appl(choice(r_6, n_q)), q, r\}$$
$$C_2 = \{prefer(choice(r_6, n_q), choice(r_6, n_s)), appl(r_6), appl(choice(r_6, n_s)), s, p\}$$
$$C_3 = \{prefer(choice(r_6, n_q), choice(r_6, n_s)), appl(r_7), t, q, s\}$$
$$C_4 = \{prefer(choice(r_6, n_q), choice(r_6, n_s)), appl(r_6), appl(choice(r_6, n_q)), q, r, appl(r_7), t, s\}$$
$$C_5 = \{prefer(choice(r_6, n_q), choice(r_6, n_s)), appl(r_6), appl(choice(r_6, n_s)), q, appl(r_7), t, s\}$$

Intuitively, not all the generalized answer sets appear equally appealing w.r.t the preferences expressed in the program. The following definition formalizes this idea.

Definition 7 Let Π be a CR-Prolog$_2$ program, and C, C' be generalized answer sets of $hr(\Pi)$. We say that C *dominates* C' (and write $C \succ C'$) if:

$$\exists appl(r_1) \in C, appl(r_2) \in C' \quad \text{s.t.} \quad is\_preferred(r_1, r_2) \in C \cap C'. \tag{3}$$

To see how this definition works, let us apply it to the generalized answer sets of program Π_1 above. According to Equation (3), $C_1 \succ C_2$. In fact $appl(choice(r_6, n_q))$ belongs to C_1, $appl(choice(r_6, n_s))$ belongs to C_2, and $prefer(choice(r_6, n_q), choice(r_6, n_s))$ belongs to $C1$ and $C2$. In a similar way, $C_4 \succ C_5$.

If a generalized answer set is dominated by another, it means that it is not as "good" as the other w.r.t. some preference contained in the program. Consider C_2, for example: since it is dominated by C_1, the intuition suggests that C_2 should be excluded from the belief sets of the agent. Generalized answer sets that are equally acceptable w.r.t. the preferences are called candidate answer sets, as stated by the next definition.

Definition 8 Let Π be a CR-Prolog$_2$ program, and C be a generalized answer set of $hr(\Pi)$. We say that C *is a candidate answer set of* Π if there exists no generalized answer set, C', of $hr(\Pi)$ such that $C' \succ C$.

Hence, C_2 and C_5 above are not candidate answer sets of Π_1, while C_1, C_3, and C_4 are. Now let us compare C_1 and C_4. Set C_1 is obtained by abducing $appl(r_6)$ and $appl(choice(r_6, n_s))$. Set C_4 is obtained by abducing $appl(r_6)$, $appl(choice(r_6, n_s))$ and $appl(r_7)$. According to the intuition, $appl(r_7)$ is abduced unnecessarily, which makes C_4 less acceptable than C_1. We discard belief sets such as C_4 by applying a minimality criterion based on set-theoretic inclusion on the abducibles present in each set. The remaining sets are the answer sets of the program.

Definition 9 Let Π be a CR-Prolog$_2$ program, and C be a candidate answer set of Π. We say that $C \cap lit(\Pi)$ *is an answer set of* Π if there exists no candidate answer set, C', of Π such that $C' \cap atoms(\{appl\}) \subset C$.

Since $C_1 \cap atoms(\{appl\}) \subset C_4$, $C_4 \cap lit(\Pi_1)$ is not an answer set of Π_1. In conclusion, the answer sets of Π_1 are $C_1 \cap lit(\Pi_1)$ and $C_3 \cap lit(\Pi_1)$.
As the reader may have noticed, the names of rules can be safely omitted when they are not used to specify preferences. In the rest of the paper, we will omit them when possible.

CR-Prolog$_2$ and CR-Prolog

CR-Prolog$_2$ has two main advantages over CR-Prolog: the major conciseness, due to ordered disjunction (see Example 2), and the improved semantics, which allows to derive the correct conclusions in cases when CR-Prolog returns unintuitive conclusions. To understand when CR-Prolog may give unintuitive results, consider the following situation:

> "We need to take full-body exercise. Full-body exercise is achieved either by combining swimming and ball playing, or by combining weight lifting and running. We prefer running to swimming and ball playing to weight lifting, but we are willing to ignore our preferences, if that is the only way to obtain a solution to the problem." (4)

According to the intuition, the problem has no solution unless preferences are ignored. In fact, we can either combine weight lifting and running, or combine swimming and ball playing, but each option is at the same time better and worse than the other according to different points of view.[2] If preferences are ignored, both combinations are acceptable.

Statement (4) can be encoded by the following program, Π_4:

$$\left\{ \begin{array}{ll} r_r : run \stackrel{+}{\leftarrow}. & full\_body\_exercise \leftarrow lift\_weights, run. \\ r_s : swim \stackrel{+}{\leftarrow}. & full\_body\_exercise \leftarrow swim, play\_ball. \\ r_p : play\_ball \stackrel{+}{\leftarrow}. & \leftarrow \text{not } full\_body\_exercise. \\ r_w : lift\_weights \stackrel{+}{\leftarrow}. & \\ & \\ prefer(r_r, r_s) \leftarrow \text{not } ignore\_prefs. & prefer(r_p, r_w) \leftarrow \text{not } ignore\_prefs. \\ r_z : ignore\_prefs \stackrel{+}{\leftarrow}. & \end{array} \right.$$

The generalized answer sets of $hr(\Pi_4)$ are: (we show only the atoms formed by run, $swim$, $play\_ball$, $lift\_weights$, $ignore\_prefs$, and $prefer$)

$G_1 : \{lift\_weights, run, prefer(r_r, r_s), prefer(r_p, r_w)\}$
$G_2 : \{swim, play\_ball, prefer(r_r, r_s), prefer(r_p, r_w)\}$
$G_3 : \{ignore\_prefs, lift\_weights, run\}$
$G_4 : \{ignore\_prefs, swim, play\_ball\}$
$G_5 : \{ignore\_prefs, lift\_weights, run, swim\}$
$G_6 : \{ignore\_prefs, lift\_weights, run, play\_ball\}$
$G_7 : \{ignore\_prefs, swim, play\_ball, lift\_weights\}$
$G_7 : \{ignore\_prefs, swim, play\_ball, run\}$
$G_8 : \{ignore\_prefs, lift\_weights, run, swim, play\_ball\}$

[2] Both alternatives are valid if we intend preferences as *desires*, instead of strict preferences. See the next section for a discussion on this topic.

Under the semantics of CR-Prolog, G_1 and G_2 are the only minimal generalized answer sets. Since $G_1 \succ G_2$ and $G_2 \succ G_1$, Π_4 has no answer sets.

Under the semantics of CR-Prolog$_2$, G_1 and G_2 dominate each other, which leaves only $G_3, \ldots, G_8$ as candidate answer sets. Since G_3 and G_4 are both minimal w.r.t. the abducibles present in each candidate answer set, they are both answer sets of Π_4, like intuition suggested.

The reason for this difference is that, in the semantics of CR-Prolog, set-theoretic minimization occurs *before* the comparison of belief sets w.r.t. the preferences. In CR-Prolog$_2$, on the other hand, generalized answer sets are first of all compared w.r.t. the preference relation, and only later set-theoretic minimization is applied. In our opinion, giving higher relevance to the preference relation is a better choice (as confirmed by the previous example), since preferences are explicitly given by the programmer.

Comparison with LPOD

In (Brewka 2002), the author introduces logic programs with ordered disjunction (LPOD). The semantics of LPOD is based on the notion of preferred answer sets. In a later paper (Brewka, Niemela, & Syrjanen 2002), the authors introduce the notion of Pareto-preference between belief sets and show that this criterion gives more intuitive results that the other criteria described in (Brewka 2002; Brewka, Niemela, & Syrjanen 2002). In this section, we compare LPOD (under Pareto-preference) and CR-Prolog$_2$.

Consider program Π_{s_1} from (Brewka, Niemela, & Syrjanen 2002):

$$\left\{\begin{array}{l} \textit{\% Have ice cream, if possible; otherwise, have cake.} \\ r_1: \quad ice\_cream \times cake. \\ \textit{\% Have coffee if possible, otherwise, have tea.} \\ r_2: \quad coffee \times tea. \\ \textit{\% It is impossible to have ice\_cream and cake together.} \\ \leftarrow ice\_cream, coffee. \end{array}\right.$$

The preferred answer sets of Π_{s_1} in LPOD are:

$$\{ice\_cream, tea\} \text{ and } \{cake, coffee\}. \tag{5}$$

There are no answer sets of Π_{s_1} according to the semantics of CR-Prolog$_2$. The difference between the two semantics depends on the fact that Pareto optimality was introduced to satisfy desires and it looks for a set of solutions that satisfy as many desires as possible. On the other hand, our preference criterion corresponds to a more strict reading of the preferences.

In order to make it easy to understand the relationship between the two types of preference, we restate the Pareto criterion in the context of CR-Prolog$_2$.

Definition 10 Let Π be a CR-Prolog$_2$ program, and C, C' be generalized answer sets of $hr(\Pi)$. We say that C *Pareto-dominates* C' (written as $C \succ_p C'$), if

$$\exists appl(r_1) \in C, appl(r_2) \in C' \text{ s.t. } is\_preferred(r_1, r_2) \in C \cap C', \text{and}$$
$$\neg\, \exists appl(r_3) \in C, appl(r_4) \in C' \text{ s.t. } is\_preferred(r_4, r_3) \in C \cap C'. \tag{6}$$

Definition 11 Let Π be a CR-Prolog$_2$ program, C be a generalized answer set of $hr(\Pi)$. We say that C *is a Pareto-candidate answer set of* Π if there exists no generalized answer set, C', of $hr(\Pi)$ such that $C' \succ_p C$.

(Notice that Pareto-domination is essentially a restatement of the Pareto criterion, in the context of CR-Prolog$_2$. Also, Pareto-candidate answer sets essentially correspond to preferred answer sets.)

Now, to see the difference between Definitions 7 and 10, consider a program, Π, and generalized answer sets, C_1 and C_2, such that C_1 dominates C_2 and vice-versa. Notice that they do not Pareto-dominate each other. Under our semantics, none of them is a candidate answer set of Π. However, using Pareto-domination, C_1 and C_2 are incomparable and thus, both are eligible as Pareto-candidate answer sets (whether they really are Pareto-candidate answer sets, depends on the other generalized answer sets).

In a sense, Definition 7 enforces a clearer representation of knowledge and of preferences. However, that does not rule out the possibility of representing desires in CR-Prolog$_2$. The defeasible nature of desires is represented by means of cr-rules. For example, the program Π_{s_1} can be rewritten as follows, Π_{s_2}:

$$\left\{\begin{array}{lll} r_{1a}: & ice\_cream \stackrel{+}{\leftarrow}. & solid \leftarrow ice\_cream. \\ r_{1b}: & cake \stackrel{+}{\leftarrow}. & solid \leftarrow cake. \\ r_{2a}: & coffee \stackrel{+}{\leftarrow}. & liquid \leftarrow coffee. \\ r_{2b}: & tea \stackrel{+}{\leftarrow}. & liquid \leftarrow tea. \\ \leftarrow & ice\_cream, coffee. & \leftarrow \text{not } solid. \\ & & \leftarrow \text{not } liquid. \\ & & \\ r_3: & prefer(r_{1a}, r_{1b}) \leftarrow \text{not } \neg prefer(r_{1a}, r_{1b}). & \\ r_4: & prefer(r_{2a}, r_{2b}) \leftarrow \text{not } \neg prefer(r_{2a}, r_{2b}). & \\ r_5: & \neg prefer(r_{1a}, r_{1b}) \stackrel{+}{\leftarrow}. & r_6: \neg prefer(r_{2a}, r_{2b}) \stackrel{+}{\leftarrow}. \end{array}\right.$$

In Π_{s_2}, the desire to have ice_cream over cake is represented by: (1) a cr-rule, r_5, that says that "the agent may possibly give up his preference for ice_cream over cake"; (2) a default, r_3, saying that "the agent normally prefers ice_cream over cake". In a similar way, we represent the desire for coffee over tea. The answer sets of the above program are (we show only the atoms from cr-rules r_{1a}-r_{2b}) $\{ice\_cream, tea\}$ and $\{cake, coffee\}$, which correspond to (5).

There are also some programs for which the semantics of LPOD seems to yield unintuitive results, while the semantics of CR-Prolog$_2$ gives results that agree with the intuition. Consider the following example:[3]

Example 1 *"A television show conducts a game where the first winner is offered a prize of \$200,000 and the second winner is offered a prize of \$100,000. John wants to play, if possible. Otherwise he will give up. If he plays he wants to gain \$200,000 if possible; otherwise, \$100,000. He is told*

[3]We thank Richard Watson for pointing out the problem with LPOD.

that he cannot win the first prize.

% John wants to play, if possible. Otherwise, give up.
$play \times give\_up.$
% If he plays, he prefers gaining \$200,000 over \$100,000.
$gain(200,000) \times gain(100,000) \leftarrow play.$
% He is told that, he cannot gain \$200,000.
$\leftarrow gain(200,000).$

Intuitively, John should play and gain \$100,000. Giving up without even trying seems a less acceptable option. Under the LPOD semantics, however, the above program has two answer sets: $\{play, gain(100,000)\}$ and $\{give\_up\}$, in contrast to the intuition. The same program under CR-Prolog$_2$ semantics gives only one answer set, $\{play, gain(100,000)\}$, which corresponds to the intuitive result. (The unintuitive result by LPOD, we believe, may be caused by the fact that degree 1 is assigned to rules whose body is not satisfied.)

Applications of CR-Prolog$_2$

CR-Prolog$_2$ can be used to encode types of common-sense knowledge which, to the best of our knowledge, have no natural formalization in A-Prolog. In this section, we give an example of such use, and show how the alternative formalization in CR-Prolog is less elegant and concise.

In the example that follows we consider a diagnostic reasoning task performed by an intelligent agent acting in dynamic domains in the sense of (Baral & Gelfond 2000). Since space limitations do not allow us to give a complete introduction on the modeling of dynamic systems in A-Prolog and its extensions, we refer the reader to (Balduccini & Gelfond 2003a; 2003b) for details on the formalization used.

Example 2 *"A car's engine starts when the start key is turned, unless there is a failure with some equipment responsible for starting the engine. There can be electrical failures, such as battery down or fuse burnt; or mechanical failures, such as clutch sensor stuck or belt loose. In general, the electrical failures are more likely than the mechanical failures. Among the electrical failures, battery down is more likely than fuse burnt. Among the mechanical failures, clutch sensor stuck is more likely than belt loose."*

The knowledge contained in this story can be represented by the following action description, Π_c:

% normally, a car's engine starts when the start key is
% turned, unless there is a failure in start equipment.
$h(engine\_on, T+1) \leftarrow o(turn\_key, T), \neg h(ab(start\_equip), T).$

% battery being down causes failure in start equipment.
$h(ab(start\_equip), T) \leftarrow h(battery\_down, T).$
% fuse being burnt causes failure in start equipment.
$h(ab(start\_equip), T) \leftarrow h(fuse\_burnt, T).$

% clutch sensor stuck causes failure in start equipment.
$h(ab(start\_equip), T) \leftarrow h(sensor\_stuck, T).$

% belt being loose causes failure in start equipment.
$h(ab(start\_equip), T) \leftarrow h(belt\_loose, T).$

% sometimes, battery is down or fuse is burnt,
% the former being more likely than the latter
$r_{elec}(T) : h(battery\_down, T) \times h(fuse\_burnt, T) \overset{+}{\leftarrow}.$

% sometimes, clutch sensor is stuck or
% belt is loose, the former being more
% likely than the latter
$r_{mech}(T) : h(sensor\_stuck, T) \times h(belt\_loose, T) \overset{+}{\leftarrow}.$

% electrical failures are more likely
% than mechanical failures
$r_p(T) : prefer(r_{elec}(T), r_{mech}(T)).$

% INERTIA
$h(F, T+1) \leftarrow h(F, T), \text{not } \neg h(F, T+1).$
$\neg h(F, T+1) \leftarrow \neg h(F, T), \text{not } h(F, T+1).$
% REALITY CHECKS
$\leftarrow obs(F, T), \text{not } h(F, T).$
$\leftarrow obs(\neg F, T), \text{not } \neg h(F, T).$
% AUXILIARY AXIOMS
$o(A, T) \leftarrow hpd(A, T).$
$h(F, 0) \leftarrow obs(F, 0).$
$\neg h(F, 0) \leftarrow obs(\neg F, 0).$

Let us add the following history to the action description:

$obs(\neg engine\_on, 0).$
$hpd(turn\_key, 0).$
$obs(\neg engine\_on, 1).$
% CWA on initial observations
$obs(\neg F, 0) \leftarrow \text{not } obs(F, 0).$

The observation at time 1 is unexpected, and causes the program to be inconsistent (because of the *reality checks*), unless at least one cr-rule is applied; because of the preference encoded by $r_p(T)$, the preferred way to restore consistency is by applying $r_{elec}(0)$; of the two options contained in the head of $r_{elec}(0)$, $h(battery\_down, 0)$ is the preferred one. Clearly, adding the belief $h(battery\_down, 0)$ restores consistency of the program, and explains the unexpected observation.

It is worth noticing that the statements encoded by rules $r_{elect}(T)$, $r_{mech}(T)$ and $r_p(T)$ of Π_c can be also represented without ordered disjunction. The three rules are replaced by:

$$
\begin{cases}
r_{batt}(T): & h(battery\_down, T) \overset{+}{\leftarrow} hyp\_elec(T).\\
r_{fuse}(T): & h(fuse\_burnt, T) \overset{+}{\leftarrow} hyp\_elec(T).\\
r_{sens}(T): & h(sensor\_stuck, T) \overset{+}{\leftarrow} hyp\_mech(T).\\
r_{belt}(T): & h(belt\_loose, T) \overset{+}{\leftarrow} hyp\_mech(T).\\
 & prefer(r_{batt}(T), r_{fuse}(T)).\\
 & prefer(r_{sens}(T), r_{belt}(T)).\\
r_{elec}(T): & hyp\_elec(T) \overset{+}{\leftarrow}.\\
r_{mech}(T): & hyp\_mech(T) \overset{+}{\leftarrow}.\\
 & prefer(r_{elec}(T), r_{mech}(T)).
\end{cases}
$$

The new program has (essentially) the same answer sets as the previous one. This shows that the rules with ordered disjunction allow for a more concise and elegant representation of knowledge.

Conclusions

In this paper, we extended CR-Prolog by ordered disjunction and an improved semantics, gave the semantics of the new language, and demonstrated how it differs from CR-Prolog and LPOD. We also showed how CR-Prolog$_2$ can be used to formalize various types of common-sense knowledge and reasoning. We could not find natural A-Prolog formalizations for some of the examples in the paper, and formalizations in CR-Prolog were often less elegant and concise (besides giving sometimes unintuitive results). In comparison with CR-Prolog, we believe that the new features of CR-Prolog$_2$ make it possible to write formalizations that are more natural, and reasonably elaboration tolerant. In comparison with LPOD, CR-Prolog$_2$ appears more expressive (because of the availability of cr-rules and epistemic disjunction), and, in some cases, yields more intuitive results than LPOD.

References

Balduccini, M., and Gelfond, M. 2003a. Diagnostic reasoning with a-prolog. *Journal of Theory and Practice of Logic Programming (TPLP)* 3(4–5):425–461.

Balduccini, M., and Gelfond, M. 2003b. Logic programs with consistency-restoring rules. In Doherty, P.; McCarthy, J.; and Williams, M.-A., eds., *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series.

Baral, C., and Gelfond, M. 2000. Reasoning agents in dynamic domains. In *Workshop on Logic-Based Artificial Intelligence*. Kluwer Academic Publishers.

Brewka, G.; Niemela, I.; and Syrjanen, T. 2002. Implementing ordered disjunction using answer set solvers for normal programs. In Flesca, S., and Ianni, G., eds., *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA 2002)*.

Brewka, G. 2002. Logic programming with ordered disjunction. In *Proceedings of AAAI-02*.

Calimeri, F.; Dell'Armi, T.; Eiter, T.; Faber, W.; Gottlob, G.; Ianni, G.; Ielpa, G.; Koch, C.; Leone, N.; Perri, S.; Pfeifer, G.; and Polleres, A. 2002. The dlv system. In Flesca, S., and Ianni, G., eds., *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA 2002)*.

Cholewinski, P.; Marek, V. W.; and Truszczynski, M. 1996. Default reasoning system deres. In *International Conference on Principles of Knowledge Representation and Reasoning*, 518–528. Morgan Kaufmann.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 365–385.

Gelfond, M. 1991. Epistemic approach to formalization of commonsense reasoning. Technical Report TR-91-2, University of Texas at El Paso.

Gelfond, M. 2002. Representing knowledge in A-Prolog. In Kakas, A. C., and Sadri, F., eds., *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408, 413–451. Springer Verlag, Berlin.

Heljanko, K. 1999. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae* 37(3):247–268.

Kakas, A. C., and Mancarella, P. 1990. Generalized stable models: a semantics for abduction. In *Proceedings of ECAI-90*, 385–391. IOS Press.

Lin, F., and Zhao, Y. 2002. Assat: Computing answer sets of a logic program by sat solvers. In *Proceedings of AAAI-02*.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An a-prolog decision support system for the space shuttle. In Provetti, A., and Tran, S. C., eds., *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, AAAI 2001 Spring Symposium Series.

Nogueira, M. 2003. *Building Knowledge Systems in A-Prolog*. Ph.D. Dissertation, University of Texas at El Paso.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Nonmonotonic reasoning on beowulf platforms. In Dahl, V., and Wadler, P., eds., *PADL 2003*, volume 2562 of *Lecture Notes in Artificial Intelligence (LNCS)*, 37–57.

Simons, P. 1996. *Computing Stable Models*.