CR-Prolog$_2$: 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 CR-Prolog with an improved semantics, and allowing ordered disjunction 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 often allows for a more concise, easier to read, representation of knowledge. We also 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, where ordered disjunction was initially introduced). Finally, we show how the new language can be used to represent complex knowledge and to perform fairly sophisticated reasoning tasks.

1 Introduction

In recent years, A-Prolog – the language of logic programs with the answer set semantics [9] – was shown to be a useful tool for knowledge representation and reasoning [8]. 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 [16,6,12,15] has allowed the use of A-Prolog for a diverse collection of applications [10,14,17,13].

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 [2] 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$_2$, an extension of CR-Prolog with an improved semantics, and allowing ordered disjunction [5] in the head of both regular rules and consistency-restoring rules (cr-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 [5] 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.

2 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:

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

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 [5]: 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

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

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

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

forces the agent to believe p, and program

$$p \times q \leftarrow \text{not } r. \\ s \leftarrow \text{not } s, p, \text{not } r.$$

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:

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

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

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

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

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

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

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

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 [11,7] – 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].

[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.

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:

$$\begin{cases} \%\ \text{transitive closure of predicate prefer} \\ m_{1a} : is\_preferred(R1, R2) \leftarrow prefer(R1, R2). \\ m_{1b} : is\_preferred(R1, R2) \leftarrow prefer(R1, R3), \\ \qquad\qquad\qquad\qquad\qquad\qquad is\_preferred(R3, R2). \\ \%\ \text{no circular preferences} \\ m_2 : \ \leftarrow is\_preferred(R, R). \\ \%\ \text{prohibit application of } R1 \text{ and } R2 \text{ if} \\ \%\ R1 \text{ is preferred to } R2 \\ m_3 : \ \leftarrow appl(R1), appl(R2), is\_preferred(R1, R2). \end{cases}$$

Let us compute the hard reduct of the following program:

$$\Pi_1 \begin{cases} r_1 : p \leftarrow \text{not}\ q. & r_5 : \ \leftarrow p, r. \\ r_2 : r \leftarrow \text{not}\ s. & \\ r_3 : q \leftarrow t. & r_6 : q \times s \overset{+}{\leftarrow} . \\ r_4 : s \leftarrow t. & r_7 : t \overset{+}{\leftarrow} . \end{cases}$$

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

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

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

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

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:

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

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 $C_3 \cap lit(\Pi_1)$.

Let us apply the above definitions to compute the answer sets of some sample programs.

Example 1. Consider the following program:

$$\Pi_2 \begin{cases} r_1 : p \leftarrow r, \text{not } q. \\ r_2 : r. \\ r_3 : s \overset{+}{\leftarrow} r. \end{cases}$$

Intuitively, r_3 should not be applied, since program $\Pi_2 \setminus \{r_3\}$ is consistent. Hence, the only answer set of Π_2 should be $\{p, r\}$. Let us check that this intuition is captured by our definition. The first element of $hr(\Pi_2)$ is $H_{\Pi_2} = H'_{\Pi_2} \cup \Pi_p$, where H'_{Π_2} is:

$$H'_{\Pi_2} \begin{cases} r_1 : p \leftarrow r, \text{not } q. \\ r_2 : r. \\ r'_3 : s \leftarrow r, appl(r_3). \end{cases}$$

The generalized answer sets of $hr(\Pi_2)$ are $C_1 = \{p, r\}$ and $C_2 = \{p, r, s, appl(r_3)\}$. Since the program does not contain preferences, C_1 and C_2 are the candidate answer sets of Π_2. Notice that $C_2 \cap lit(\Pi_2)$ is not an answer set of Π_2, since $C_1 \cap atoms(\{appl\}) \subset C_2$. Hence, C_1 is the only answer set of Π_2.

Example 2. Consider the following program:

$$\Pi_3 \begin{cases} r_1 : a \leftarrow p. & & \\ r_2 : a \leftarrow r. & r_{5a} : \leftarrow \text{not } a. & r_6 : p \times q \overset{+}{\leftarrow} . \\ r_3 : b \leftarrow q. & r_{5b} : \leftarrow \text{not } b. & r_7 : r \times s \overset{+}{\leftarrow} . \\ r_4 : b \leftarrow s. & & \end{cases}$$

The generalized answer sets of $hr(\Pi_3)$ are:

$$\begin{aligned} C_1 = \{ & prefer(choice(r_6, n_p), choice(r_6, n_q)), \\ & prefer(choice(r_7, n_r), choice(r_7, n_s)), \\ & appl(choice(r_6, n_p)), \\ & appl(choice(r_7, n_s)), p, s, a, b\} \end{aligned} \quad \begin{aligned} C_2 = \{ & prefer(choice(r_6, n_p), choice(r_6, n_q)), \\ & prefer(choice(r_7, n_r), choice(r_7, n_s)), \\ & appl(choice(r_7, n_r)), \\ & appl(choice(r_6, n_q)), r, q, a, b\} \end{aligned}$$

Since $C_1 \succ C_2$ and $C_2 \succ C_1$, Π_3 has no answer sets.

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.

3 CR-Prolog$_2$ and CR-Prolog

CR-Prolog$_2$ has two main advantages over CR-Prolog: the availability of ordered disjunction and the improved semantics, which solves some problems present in the semantics of CR-Prolog.

Ordered disjunction allows for a more concise, easier to read, representation of knowledge when the preference order on a set of alternatives is total. Consider the following program:

$$\begin{cases} r_1 : eat(meat) \overset{+}{\leftarrow} . & eat \leftarrow eat(X). \\ r_2 : eat(rice) \overset{+}{\leftarrow} . & starve \leftarrow hungry, \text{not } eat. \\ r_3 : eat(cracker) \overset{+}{\leftarrow} . & \leftarrow starve. \\ \quad prefer(r_1, r_2). & \\ \quad prefer(r_2, r_3). & \end{cases}$$

Since the preference order between the cr-rules is total, the program can be quite simplified by introducing ordered disjunction:

$$\left\{\begin{array}{l} eat(meat) \times eat(rice) \times eat(cracker) \stackrel{+}{\leftarrow} . \\ \quad eat \leftarrow eat(X). \\ starve \leftarrow hungry, \text{not } eat. \\ \qquad \leftarrow starve. \end{array}\right.$$

Moreover, ordered disjunction can be used to concisely represent a set of alternatives among which one *must* be chosen. It may be instructive to compare how A-Prolog, CR-Prolog, and CR-Prolog$_2$ can be used in this context. Suppose we want to encode the statement

"normally, hard-working graduate students receive A in the courses they take." (4)

We might simply represent it with a default:

$$\begin{array}{rl} received(S, a, C) \leftarrow & hard\_working(S), took(S, C), \\ & \text{not } \neg received(S, a, C). \end{array}$$

Consider now the additional information "Frank is an hard-working student, but he took AI, and did not receive A". It would seem natural to conclude that Frank *probably* received a B, but our simple representation of (4) does not yield this conclusion. Writing in A-Prolog a set of defaults that embodies the common-sense knowledge about this domain can prove tedious. A possible solution is:

$$\left\{\begin{array}{rl} better\_grade(a, b). & better\_grade(b, c). \\ received(S, G1, C) \leftarrow & hard\_working(S), took(S, C), \\ & \text{not } \neg received(S, G1, C), \\ & better\_grade(G2, G1), \\ & \neg received(S, G2, C). \\ received(S, a, C) \leftarrow & hard\_working(S), took(S, C), \\ & \text{not } \neg received(S, a, C). \end{array}\right. \tag{5}$$

The first rule says that, normally, an hard-working student takes grade $G1$ in the courses he attends, unless he takes a better grade. The second rule is needed because the previous statement does not apply to grade A, as there is no better grade than that. [2] Given the information:

$$\left\{\begin{array}{l} hard\_working(frank). \\ took(frank, ai). \\ \neg received(frank, a, ai). \end{array}\right. \tag{6}$$

the program correctly concludes $received(frank, b, ai)$. On the other hand, if information is represented as:

$$\left\{\begin{array}{l} hard\_working(frank). \\ took(frank, ai). \\ \leftarrow received(frank, a, ai). \end{array}\right. \tag{7}$$

the program still fails to yield the intuitive conclusion.

[2] Several options are possible in order to avoid writing the second default, but none of them seems completely satisfactory.

A better way to solve the problem is offered by CR-Prolog. Consider the following program:

$$\begin{cases} r(S,G,C) : received(S,G,C) \stackrel{+}{\leftarrow} took(S,C). \\ \\ prefer(r(S,a,C), r(S,b,C)) \leftarrow hard\_working(S). \\ prefer(r(S,b,C), r(S,c,C)) \leftarrow hard\_working(S). \\ \\ had\_grade(S,C) \leftarrow received(S,G,C). \\ \leftarrow took(S,C), \text{not } had\_grade(S,C). \end{cases} \tag{8}$$

This program yields the expected conclusion with both (6) and (7).

Now let us consider a possible CR-Prolog$_2$ representation of (4):

$$\begin{cases} received(S,a,C) \times \\ received(S,b,C) \times \\ received(S,c,C) \leftarrow took(S,C), hard\_working(S). \end{cases} \tag{9}$$

It is easy to see that this program yields the desired conclusion independently of whether the extra information is given in form (6) or (7). Moreover, the program is, in our opinion, easier to understand than the previous ones.[3]

Not only CR-Prolog$_2$ is often more concise than CR-Prolog: it also 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." (10)

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.[4] If preferences are ignored, both combinations are acceptable.

[3] It is worth noticing that, although ordered disjunction provides a concise representation, it cannot entirely replace the use of cr-rules. For example, (8) allows us to derive the grade received by graduate students who are *not* hard-working, while (9) does not yield any conclusion about them.

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

Statement (10) can be encoded by the following program, Π_4:

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

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 $prefer$)

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

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.

4 Comparison with LPOD

In [4], 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 [5], 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 [4,5]. In this section, we compare LPOD (under Pareto-preference) and CR-Prolog$_2$.

Consider program Π_{s_1} from [5]:

$$\begin{cases} \%\ \textit{Ice cream is preferred to cake.} \\ r_1 : ice\_cream \times cake. \\ \%\ \textit{Coffee is preferred to tea.} \\ r_2 : coffee \times tea. \\ \%\ \textit{We cannot have ice\_cream and cake together.} \\ \leftarrow ice\_cream, coffee. \end{cases}$$

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

$$\{ice\_cream, tea\} \text{ and } \{cake, coffee\}. \tag{11}$$

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

$$\begin{array}{l} \exists appl(r_1) \in C, appl(r_2) \in C' \text{ s.t.} \\ is\_preferred(r_1, r_2) \in C \cap C', \text{and} \\ \\ \neg\, \exists appl(r_3) \in C, appl(r_4) \in C' \text{ s.t.} \\ is\_preferred(r_4, r_3) \in C \cap C'. \end{array} \tag{12}$$

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}:

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

In Π_{s_2}, the desire to have ice_cream over cake is represented by:

- a cr-rule, r_5, that says that, "the agent may possibly give up his preference for ice_cream over cake".
- 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 (11).

Now suppose that we are more inclined to give up the preference about coffee and tea rather than the preference about ice_cream and cake. In the language of LPOD, such priorities over rules are represented by the *meta-preference* relation $>$.[5] The above information is represented by adding to Π_{s_1} the rule $r_5 > r_6$, which says that r_5 is preferred to r_6.

Adding the same type of knowledge using CR-Prolog$_2$ is, in our opinion, more straightforward, because it does not require the introduction of a new type of preference relation in the language.[6] The above information is represented in CR-Prolog$_2$ by adding to Π_{s_2} the rule $prefer(r_6, r_5)$, which says that belief sets obtained by giving up the preference on coffee and tea are more acceptable than the belief sets obtained by giving up the preference on ice_cream and cake.

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:[7]

[5] The connective used in [5] is $\succ$. In this paper we replace $\succ$ by $>$ to avoid confusion with the symbol introduced in Definition 7.
[6] It is still not sure if the two approaches are entirely equivalent. A thorough comparison is under way.
[7] We thank Richard Watson for noticing the problem with LPOD and suggesting the example.

Example 3. *"John prefers to go to a movie over watching tv. If he goes to the movies then he prefers eating popcorn over candy. Now popcorn is not available."*

$$\left\{\begin{array}{l}
\%\ \textit{John prefers to go to a movie over watching tv.} \\
movie \times tv. \\
\%\ \textit{At the movies, he prefers eating popcorn over candy.} \\
popcorn \times candy \leftarrow movie. \\
\%\ \textit{Popcorn is not available.} \\
\neg popcorn.
\end{array}\right.$$

Intuitively, John should prefer going to the movies. Since popcorn is not available, he will eat candy. Watching tv seems a less acceptable option. Under the LPOD semantics, however, the above program has two answer sets: $\{movie, candy\}$ and $\{tv\}$, in contrast to the intuition. The same program under CR-Prolog$_2$ semantics gives only one answer set, $\{movie, candy\}$, 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.)

5 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 [3]. 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 [1,2] for details on the formalization used.

Example 4. *"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:

$$\left\{\begin{array}{l}
\%\ \text{normally, a car's engine starts when the start key is turned,} \\
\%\ \text{unless there is a failure in start equipment.} \\
h(engine_on, T+1) \ \leftarrow o(turn_key, T), \\
\qquad\qquad\qquad\qquad\qquad \neg h(ab(start_equip), T). \\
\\
\%\ \text{battery being down causes failure in start equipment.} \\
h(ab(start_equip), T) \leftarrow h(battery_down, T).
\end{array}\right.$$

$$
\begin{cases}
\text{\% fuse being burnt causes failure in start equipment.} \\
\qquad h(ab(start\_equip), T) \leftarrow h(fuse\_burnt, T). \\
\\
\text{\% clutch sensor stuck causes failure in start equipment.} \\
\qquad h(ab(start\_equip), T) \leftarrow h(sensor\_stuck, T). \\
\\
\text{\% belt being loose causes failure in start equipment.} \\
\qquad h(ab(start\_equip), T) \leftarrow h(belt\_loose, T). \\
\\
\text{\% sometimes, battery is down or fuse is burnt,} \\
\text{\% the former being more likely than the latter} \\
r_{elec}(T): \quad h(battery\_down, T) \times h(fuse\_burnt, T) \stackrel{+}{\leftarrow} . \\
\\
\text{\% sometimes, clutch sensor is stuck or belt is loose,} \\
\text{\% the former being more likely than the latter} \\
r_{mech}(T): h(sensor\_stuck, T) \times h(belt\_loose, T) \stackrel{+}{\leftarrow} . \\
\\
\text{\% electrical failures are more likely than mechanical failures} \\
r_p(T): \qquad prefer(r_{elec}(T), r_{mech}(T)). \\
\\
\text{\% INERTIA} \\
\qquad h(F, T+1) \leftarrow h(F, T), \text{not } \neg h(F, T+1). \\
\qquad \neg h(F, T+1) \leftarrow \neg h(F, T), \text{not } h(F, T+1). \\
\text{\% REALITY CHECKS} \\
\qquad \leftarrow obs(F, T), \text{not } h(F, T). \\
\qquad \leftarrow obs(\neg F, T), \text{not } \neg h(F, T). \\
\text{\% AUXILIARY AXIOMS} \\
\qquad o(A, T) \leftarrow hpd(A, T). \\
\qquad h(F, 0) \leftarrow obs(F, 0). \\
\qquad \neg h(F, 0) \leftarrow obs(\neg F, 0).
\end{cases}
$$

Let us add the following history to the action description:

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

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:

$$
\begin{cases}
r_{batt}(T): \ h(battery\_down, T) \stackrel{+}{\leftarrow} hyp\_elec(T). \\
r_{fuse}(T): h(fuse\_burnt, T) \quad \stackrel{+}{\leftarrow} hyp\_elec(T). \\
r_{sens}(T): h(sensor\_stuck, T) \stackrel{+}{\leftarrow} hyp\_mech(T). \\
r_{belt}(T): \ h(belt\_loose, T) \qquad \stackrel{+}{\leftarrow} hyp\_mech(T).
\end{cases}
$$

$$\left\{\begin{array}{ll} & prefer(r_{batt}(T), r_{fuse}(T)). \\ & prefer(r_{sens}(T), r_{belt}(T)). \\ r_{elec}(T): & hyp\_elec(T) \stackrel{+}{\leftarrow} . \\ r_{mech}(T): & hyp\_mech(T) \stackrel{+}{\leftarrow} . \\ & prefer(r_{elec}(T), r_{mech}(T)). \end{array}\right.$$

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.

6 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.

7 Acknowledgments

The authors are very thankful to Micheal Gelfond for his suggestions. This work was partially supported by United Space Alliance under Research Grant 26-3502-21 and Contract COC6771311, and by NASA under grant NCC9-157.

References

1. 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.
2. 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, Mar 2003.
3. Chitta Baral and Michael Gelfond. Reasoning agents in dynamic domains. In *Workshop on Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, Jun 2000.
4. Gerhard Brewka. Logic programming with ordered disjunction. In *Proceedings of AAAI-02*, 2002.
5. Gerhard Brewka, Ilkka Niemela, and Tommi Syrjanen. Implementing ordered disjunction using answer set solvers for normal programs. In Sergio Flesca and Giovanbattista Ianni, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA 2002)*, Sep 2002.

6. Francesco Calimeri, Tina Dell'Armi, Thomas Eiter, Wolfgang Faber, Georg Gottlob, Giovanbattista Ianni, Giuseppe Ielpa, Christoph Koch, Nicola Leone, Simona Perri, Gerard Pfeifer, and Axel Polleres. The dlv system. In Sergio Flesca and Giovanbattista Ianni, editors, *Proceedings of the 8th European Conference on Artificial Intelligence (JELIA 2002)*, Sep 2002.
7. Michael Gelfond. Epistemic approach to formalization of commonsense reasoning. Technical Report TR-91-2, University of Texas at El Paso, 1991.
8. 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.
9. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–385, 1991.
10. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.
11. Antonis C. Kakas and Paolo Mancarella. Generalized stable models: a semantics for abduction. In *Proceedings of ECAI-90*, pages 385–391. IOS Press, 1990.
12. Fangzhen Lin and Yuting Zhao. Assat: Computing answer sets of a logic program by sat solvers. In *Proceedings of AAAI-02*, 2002.
13. Monica Nogueira. *Building Knowledge Systems in A-Prolog*. PhD thesis, University of Texas at El Paso, May 2003.
14. Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An a-prolog decision support system for the space shuttle. In Alessandro Provetti and Son Cao Tran, editors, *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, AAAI 2001 Spring Symposium Series, Mar 2001.
15. Enrico Pontelli, Marcello Balduccini, and F. Bermudez. Non-monotonic reasoning on beowulf platforms. In Veronica Dahl and Philip Wadler, editors, *PADL 2003*, volume 2562 of *Lecture Notes in Artificial Intelligence (LNCS)*, pages 37–57, Jan 2003.
16. Patrik Simons. *Computing Stable Models*, Oct 1996.
17. Timo Soininen and Ilkka Niemela. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, May 1999.