Logic Programs with Consistency-Restoring Rules

Marcello Balduccini and **Michael Gelfond**
Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
{balduccini, mgelfond}@cs.ttu.edu

Abstract

We present an extension of language A-Prolog by consistency-restoring rules with preferences, give the semantics of the new language, CR-Prolog, and show how the language can be used to formalize various types of commonsense knowledge and reasoning.

Introduction

In recent years, A-Prolog – a 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 (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 (Heljanko 1999; Balduccini, Gelfond, & Nogueira 2000; Nogueira *et al.* 2001; Soininen & Niemela 1999; Aiello & Massacci 2001).

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 we developed CR-Prolog - an extension of A-Prolog by *consistency-restoring rules* (cr-rules) with preferences - which is capable of such reasoning. To illustrate the problem and its solution we consider several reasoning tasks performed by an intelligent agent acting in dynamic domains in the sense of (Baral & Gelfond 2000).

The paper is structured as follows. We start with background material needed to understand modeling of dynamic system in A-Prolog. Next, we introduce the syntax and semantics of CR-Prolog, give examples of using CR-rules with preferences for representing agent's knowledge, and show how this knowledge can be used to perform fairly sophisticated reasoning tasks. Finally, we discuss related work and future research directions.

Modeling dynamic systems

In this section we briefly review the basic ideas of A-Prolog based modeling of dynamic systems. We assume that the dynamic system, consisting of an agent and its environment, satisfies the following simplifying conditions.

1. The environment can be viewed as a transition diagram whose states are sets of fluents (relevant properties of the domain whose truth values may depend on time) and whose arcs are labeled by actions.
2. The agent is capable of making correct observations, performing actions, and remembering the domain history.
3. Normally the agent is capable of observing all relevant exogenous events occurring in its environment.

The above assumptions determine the structure of the agent's knowledge base. It consists of three parts:

- An *action description*, which specifies the transition diagram representing possible trajectories of the system. It contains descriptions of domain's actions and fluents, together with the definition of possible successor states to which the system can move after an action a is executed in a state σ.
- A *recorded history*, which contains observations made by the agent together with a record of its own actions. It defines a collection of paths in the diagram which, from the standpoint of the agent, can be interpreted as the system's possible pasts. If the agent's knowledge is complete (e.g., it has complete information about the initial state and the occurrences of actions, and the system's actions are deterministic) then there is only one such path.
- A *collection of agent's goals*.

All this knowledge is used and updated by the agent who repeatedly executes steps of the following observe-think-act loop:

Observe-think-act loop

1. observe the world;
2. interpret the observations;
3. select a goal;
4. plan;
5. execute part of the plan.

We will illustrate this design with the following simple example:

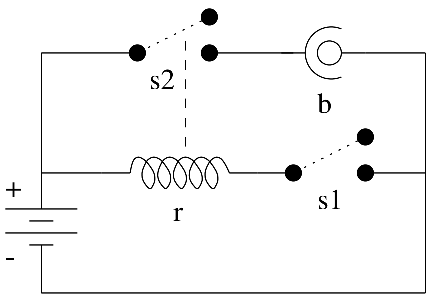

Figure 1: $\mathcal{AC}$

Example 1 Consider a system S consisting of an analog circuit $\mathcal{AC}$ from Figure 1. We assume that switches s_1 and s_2 are mechanical components which cannot become damaged. Relay r is a magnetic coil. If not damaged, it is activated when s_1 is closed, causing s_2 to close. Undamaged bulb b emits light if s_2 is closed. For simplicity of presentation we consider an agent capable of performing only one action, $close(s_1)$. The environment can be represented by two damaging exogenous[1] actions: brk, which causes b to become faulty, and srg, which damages r and also b assuming that b is not protected.

To model the system we introduce fluents:
$closed(SW)$ - switch SW is closed;
$ab(C)$ - component C is malfunctioning;
$prot(b)$ - bulb b is protected from power surges;
$active(r)$ - relay r is active;
$on(b)$ - bulb b is on.

The action description, AD, of S consists of the rules in the first three sections of program Π_d (Figure 2). Statement $h(f,t)$ says that "fluent f holds at time t"; $o(a,t)$ stands for "action a occurred at time t".

The first section of Π_d contains dynamic causal laws (Gelfond 2002) of S expressed in A-Prolog. Its first rule, which corresponds to a statement "a causes f if P" of action language A and its extensions, says that closing switch s_1 causes the switch to be closed. (We assume that all actions have unit durations.)

The second section of Π_d contains relations between fluents frequently referred to as domain constraints. In action languages they are normally described by statements "f if P". The first rule, for example, says that relay r is active if switch s_1 is closed and r is not malfunctioning.

The third section contains executability conditions of actions. The only rule of this section says that it is impossible to close switch s_1 if it is already closed.

[1] By *exogenous* actions we mean actions performed by the agent's environment. This includes natural events as well as actions performed by other agents.

To give the semantics of action description AD, we need to define states of the corresponding transition diagram T and its transition relation $\langle \sigma_0, a, \sigma_1 \rangle$. By a state we mean a complete and consistent collection of fluent literals of AD which satisfy its domain constraints. To define the transition relation we consider the program Π_i consisting of the first four sections of Π_d. Rules of the fourth part formalize the famous Inertia Axiom from (Hayes & McCarthy 1969) which says that "things tend to stay as they are". The simplicity of representation of this (and other) defaults is due to the *default (or stable) negation* "not" of A-Prolog. The transition relation of T is given by the following

Definition 1 (Successor State) Let Π_i be the action description plus the Inertia Axioms from Π_d, $\mathcal{T} = \langle \sigma_0, a, \sigma_1 \rangle$ be a transition, and $h(\sigma_0, 0) = \{h(f,0) \mid f \in \sigma_0\} \cup \{\neg h(f,0) \mid \neg f \in \sigma_0\}$. Each answer set, A, of $\Pi_i \cup h(\sigma_0,0) \cup o(a,0)$, defines a successor state, σ_1, in $\mathcal{T}$ as:

$$\sigma_1 = \{f \mid h(f,1) \in A\} \cup \{\neg f \mid \neg h(f,1) \in A\}.$$

Using this definition one can easily check that the execution of action $close(s_1)$ in state $\sigma_a = \{prot(b)\}$ moves the system to state $\sigma_b = \{closed(s_1), closed(s_2), on(b), prot(b)\}$. (Whenever possible we omit negative fluent literals from the representation of states.) Notice also that if $\sigma_0 = \{prot(b), closed(s_1)\}$ and a is $close(s_1)$, then the corresponding program has no answer set and hence a is not executable in σ_0.

This approach of defining T dates back to (Baral & Lobo 1997). The more commonly used alternative specifies T by a collection of statements in a "high level" action language with the successor relation defined in the style of McCain and Turner (McCain & Turner 1995; 1997; Turner 1997; Balduccini & Gelfond 2002). For a number of action languages, these approaches were shown to be equivalent. Since we believe that some knowledge of A-Prolog will be needed by a system designer and that the distance between statements in action languages and the corresponding logic programming rules is fairly short we use the A-Prolog based approach. Moreover, this approach is rather general and can be applied to action descriptions with defeasible causal laws, non-deterministic actions, etc. for which McCain-Turner style definitions are either non-existent or more complex.

Recorded history Γ_n (where n is the current time step) is given by a collection of statements of the form:

1. $obs(l,t)$ - 'fluent literal l was observed to be true at moment t';
2. $hpd(a,t)$ - action $a \in A$ was observed to happen at moment t

where t is an integer from the interval $[0,n)$.

The trajectories $\langle \sigma_0, \ldots \rangle$ of the system defined by Γ_n can be extracted from the answer sets of $\Pi_d \cup \Gamma_n$. Note that the axioms in the last two sections of Π_d establish the relationship between relations obs, hpd and h,o. The former correspond to undoubtedly correct observations while the latter correspond to predictions made by the agent, and may be defeated by further observations. The "reality checks" axioms ensure that the agent's predictions do not contradict his observations. One can easily check, for instance, that $\Gamma_1 = obs(\sigma_a, 0) \cup \{hpd(close(s_1), 0)\}$ defines the trajectory $\langle \sigma_a, close(s_1), \sigma_b \rangle$. Here $obs(\sigma, t) = \{l \mid l \in \sigma\}$. The program corresponding to history $\Gamma_1 = obs(\sigma_a, 0) \cup \{hpd(close(s_1), 0) \cup obs(\neg closed(s_1), 1)\}$ is inconsistent (thanks to the reality checks of Π_d), and hence Γ_1 defines no trajectories.

Finally, the goals of the agent can be represented by rules of the form:

$$\leftarrow \text{not } goal.$$

The agent will be able to use this knowledge for planning, diagnosing, consistency checking, and other tasks. For example the diagnostics agent in (Balduccini & Gelfond 2002) finds possible explanation of observations O by finding answer sets of the program $\Pi_d \cup O \cup DM_0$, where DM_0 is the following "diagnostic module":

$$o(A, T) \text{ or } \neg o(A, T) \leftarrow 0 \leq T < n,\ x\_act(A).$$

where $x\_act(A)$ is satisfied by exogenous actions (in our case $brks$ and srg). This is of course done only if the corresponding observations are inconsistent with the agent's predictions, i.e. when the program $\Pi_d \cup O$ is inconsistent. To see how this works let us go back to action description from Example 1.

Example 2 Suppose that initially all switches are open, the bulb is off and protected from power surges, and that the agent operating this device is given the goal of lighting the bulb. The agent starts at time 0 by observing the initial state O_0. Since $\Pi_d \cup O_0$ is consistent no explanation is required, the agent selects the goal of lighting the bulb, and generates a plan, $close(s_1)$, to achieve it. As shown in (Dimopoulos, Koehler, & Nebel 1997; Lifschitz 1999) this can be done by extracting occurrences of actions (i.e. statements of the form $o(a, t)$) from an answer set of program $\Pi_d \cup O_0 \cup PM$ where PM is

$$o(A, T) \text{ or } \neg o(A, T) \leftarrow n \leq T, a\_act(A).$$

where $a\_act(A)$ is satisfied by agent actions (in our case $close(s_1)$). The plan is executed at step 5 and the agent goes back to observe the world. Suppose that the agent discovers that the bulb is not lit, i.e. $O_1 = O_0 \cup \{hpd(close(s_1), 0), obs(\neg on(b), 1)\}$. Since $\Pi_d \cup O_1$ is inconsistent, the agent performs diagnosis at step 2 by computing the answer sets of $\Pi_d \cup O_1 \cup DM_0$. In this case, there is only one answer set, corresponding to the occurrence of action $brks$ at time 0. Since the bulb is broken, the agent has no way to achieve its goal. However, if more complex actions were introduced – like $repair(C)$, to repair a component – the agent could generate a new plan $repair(b), close(s_1)$, which would finally allow it to light the bulb.

%% **DYNAMIC CAUSAL LAWS**

$$\begin{array}{lcl}
h(closed(s_1), T+1) & \leftarrow & o(close(s_1), T). \\
 & & \\
h(ab(b), T+1) & \leftarrow & o(brks, T). \\
 & & \\
h(ab(r), T+1) & \leftarrow & o(srg, T). \\
 & & \\
h(ab(b), T+1) & \leftarrow & \neg h(prot(b), T), o(srg, T).
\end{array}$$

%% **DOMAIN CONSTRAINTS**

$$\begin{array}{lcl}
h(active(r), T) & \leftarrow & h(closed(s_1), T), \neg h(ab(r), T). \\
\neg h(active(r), T) & \leftarrow & h(ab(r), T). \\
\neg h(active(r), T) & \leftarrow & \neg h(closed(s_1), T). \\
 & & \\
h(closed(s_2), T) & \leftarrow & h(active(r), T). \\
 & & \\
h(on(b), T) & \leftarrow & h(closed(s_2), T), \neg h(ab(b), T). \\
 & & \\
\neg h(on(b), T) & \leftarrow & h(ab(b), T). \\
\neg h(on(b), T) & \leftarrow & \neg h(closed(s_2), T).
\end{array}$$

%% **EXECUTABILITY CONDITION**

$$\leftarrow o(close(s_1), T), h(closed(s_1), T).$$

%% **INERTIA**

$$\begin{array}{lcl}
h(F, T+1) & \leftarrow & h(F, T), \text{not } \neg h(F, T+1). \\
\neg h(F, T+1) & \leftarrow & \neg h(F, T), \text{not } h(F, T+1).
\end{array}$$

%% **REALITY CHECKS**

$$\begin{array}{l}
\leftarrow obs(F, T), \text{not } h(F, T). \\
\leftarrow obs(\neg F, T), \text{not } \neg h(F, T).
\end{array}$$

%% **AUXILIARY AXIOMS**

$$\begin{array}{lcl}
o(A, T) & \leftarrow & hpd(A, T). \\
h(F, 0) & \leftarrow & obs(F, 0). \\
\neg h(F, 0) & \leftarrow & obs(\neg F, 0).
\end{array}$$

Figure 2: Π_d

Expanding the language

Notice that checking consistency and finding a diagnosis in the above algorithms is achieved by two calls to lp-satisfiability checkers – inference engines computing answer sets of logic programs. Such multiple calls require the repetition of a substantial amount of computation (including grounding of the whole program). Even more importantly, we have no way to declaratively specify preferences between possible diagnoses, and hence may be forced to eliminate unlikely diagnoses by performing extra observations. To avoid these problems, we expand A-Prolog by *consistency-restoring* rules with preferences. We call the new language "CR-Prolog".

Cr-rule is a statement of the form:

$$r: \quad h_1 \text{ or } \ldots \text{ or } h_k \overset{+}{\leftarrow} \quad \begin{array}{l} l_1, \ldots, l_m, \\ \text{not } l_{m+1}, \ldots, \text{not } l_n \end{array} \tag{1}$$

where r is the name of the rule, and $h_1, \ldots, h_k, l_1, \ldots, l_n$ 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 $h_1, \ldots, h_k$. This possibility is used only if the agent has no way to obtain a consistent set of beliefs using regular rules only.

Consider for instance program Π_0:

$$\Pi_0 : \left\{ \begin{array}{l} a \leftarrow \text{not } b. \\ r_1 : b \overset{+}{\leftarrow} . \end{array} \right.$$

Π_0 has an answer set $\{a\}$, computed without the use of cr-rule r_1. Now consider $\Pi'_0 = \Pi_0 \cup \{\neg a.\}$. If r_1 is not used, Π'_0 in inconsistent. Consistency can be restored by using r_1 which allow the reasoner to believe in b, leading to the answer set $\{\neg a, b\}$.

Cr-rules can be used by an agent to remove the need for multiple calls to lp-satisfiable checkers. Consider the program from Example 1 with a new diagnostic module, DM_0^{cr}:

$$DM_0^{cr} \left\{ \; r(A,T): \quad o(A,T) \overset{+}{\leftarrow} T < n, x\_act(A). \right.$$

which say that some (unobserved) exogenous actions may possibly have occurred in the past. This fact can be used by the agent to restore consistency of his beliefs. To see how this is done consider recorded histories O_1, O_2, and O_3 with $n = 1$. (To save space, we will omit observations of negative fluent literals.)

$$O_1 : \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(prot(b), 0). \\ obs(on(b), 1). \end{array} \right. \qquad O_2 : \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(prot(b), 0). \\ obs(\neg on(b), 1). \end{array} \right.$$

$$O_3 : \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(\neg on(b), 1). \end{array} \right.$$

According to our semantics the answer set of $\Pi_d \cup O_1 \cup DM_0^{cr}$ contains no occurrences of exogenous actions - no cr-rules are used at this point; Consistency of the "regular part" of $\Pi_d \cup O_2 \cup DM_0^{cr}$ can be restored only by rule $r(brks, 0)$. The problem is explained by the occurrence of $brks$. $\Pi_d \cup O_3 \cup DM_0^{cr}$ has two answer sets, one obtained using $r(brks, 0)$, and the other obtained using $r(srg, 0)$. The agent concludes that either $brks$ or srg occurred at time 0.

The agent's selection of cr-rules can be guided by the preference relation $prefer(r_1, r_2)$ which says that sets of beliefs obtained by applying r_1 are preferred over those obtained by applying r_2.

For example, we might want to say that $brks$ occurs more often then srg and hence an explanation based on an unobserved past occurrence of $brks$ is preferred to one based on similar occurrence on srg. The rule

$$\Pi_d^p : \left\{ \; prefer(r(brks, T), r(srg, T)). \right.$$

formalizes this intuition. Given $\Pi_d \cup O_3 \cup DM_0^{cr} \cup \Pi_d^p$, our agent will use this rule to conclude that $brks$ occurred at time 0. It will not conclude that srg occurred, since this corresponds to a less preferred set of beliefs. The agent may derive that srg occurred only if additional information is provided, showing that $brks$ cannot have occurred. Now we give a precise definition of our language.

Syntax and Semantics

Let Σ be a signature containing symbols for constants, variables, functions, and predicates. Terms, atoms, and literal are defined as in FOL. 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 $lit(\Sigma)$. Let P be a set of predicate symbols from Σ. By $atoms(\Sigma, P)$ we denote the set of all atoms from $atoms(\Sigma)$ formed by predicate symbols from P. (Whenever possible we drop the first argument and simply write $atoms(P)$).

A *regular* rule of CR-Prolog is a statement of the form:

$$r: \quad h_1 \text{ or } \ldots \text{ or } h_k \leftarrow \quad \begin{array}{l} l_1, \ldots, l_m, \\ \text{not } l_{m+1}, \ldots, \text{not } l_n \end{array} \tag{2}$$

where $h_1, \ldots, h_k, l_1, \ldots, l_n$ are literals, and r is a term representing the name of the rule. (Names of regular rules are not needed for the definition of the semantics of programs with cr-rules, and can thus be safely dropped.) A *cr-rule* is a statement of the form (1). 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 2 An CR-Prolog program, Π, is a pair $\langle \Sigma, R \rangle$ consisting of signature Σ and a set R of rules of form (2) or (1).

The signature Σ is often denoted by $sig(\Pi)$; the set of atoms (literals) of $sig(\Pi)$ is denoted by $atoms(\Pi)$ ($lit(\Pi)$); $rules(\Pi)$ stands for the set of rules of Π. If ρ is a rule of Π then $head(\rho) = \{h_1, \ldots, h_k\}$; $body(\rho) = \{l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n\}$.

Programs of CR-Prolog are closely related to abductive logic programs (Kakas & Mancarella 1990; Kakas, Kowalski, & Toni 1998; 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*[2]. The semantics of the language is based on a trans-

[2]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 answer set as *minimal* if it is minimal with respect to this ordering.

formation $hr(\Pi)$ of its programs into abductive programs. The *hard reduct* $hr(\Pi) = \langle H_\Pi, atoms(\{appl\})\rangle$ is defined as follows:

1. $sig(H_\Pi) = \langle const(\Pi), \{appl, is\_preferred\} \cup pred(\Pi)\rangle$. We assume that $appl$ and $is\_preferred$ do not occur in $sig(\Pi)$.
2. every regular rule of Π belongs to H_Π;
3. for every consistency-restoring rule $\rho \in rules(\Pi)$, the following regular rule belongs to H_Π:

$$head(\rho) \leftarrow body(\rho), appl(r)$$

   where r is the name of rule ρ;
4. finally, if $prefer \in pred(\Pi)$, H_Π contains the following set of rules, denoted by Π_p:

$$\left\{\begin{array}{l} \text{\% transitive closure of predicate prefer} \\ m_{1a} : is\_preferred(R1, R2) \leftarrow prefer(R1, R2). \\ m_{1b} : is\_preferred(R1, R2) \leftarrow prefer(R1, R3), \\ \qquad\qquad\qquad\qquad\qquad\qquad is\_preferred(R3, R2). \\ \text{\% no circular preferences} \\ m_2 : \ \leftarrow is\_preferred(R, R). \\ \text{\% prohibits application of a lesser rule if} \\ \text{\% a better rule is applied} \\ m_3 : \ \leftarrow appl(R1), appl(R2), is\_preferred(R1, R2). \end{array}\right.$$

   where R_1, R_2, R_3 denote names of rules.

We need two preliminary definitions:

Definition 3 A set of literals, C, is a *candidate answer set* of Π if C is a minimal generalized answer set of $hr(\Pi)$.

Definition 4 Let C, D be candidate answer sets of Π. C is *better than* D ($C \prec D$) if

$$\begin{array}{l} \exists appl(r_1) \in C \ \ \exists appl(r_2) \in D \\ is\_preferred(r_1, r_2) \in C \cap D. \end{array} \tag{3}$$

We can now give the definition of answer set of a program.

Definition 5 Let C be a candidate answer set of Π, and $\hat{C}$ be $C \setminus atoms(\{appl, is\_preferred\})$. $\hat{C}$ is an answer set of Π if there exists no candidate answer set, D, of Π which is better than C.

One can notice that Definition 4 has some similarities with Pareto-optimality (Pareto 1896). In fact, our definition and (a suitably modified version of) Pareto-optimality appear[3] to yield the same answer sets in all cases, except when conflicting preferences have been specified. When this happens, our definition behaves in a more conservative way, returning no answer set instead of all candidate answer sets, like Pareto-optimality does.

Example 3 Consider the following program:

$$\Pi_1 \left\{\begin{array}{llll} r_1: & p & \leftarrow & r, \text{not } q. \\ r_2: & r. & & \\ & & & \\ r_3: & s & \stackrel{+}{\leftarrow} & r. \end{array}\right.$$

[3]This conclusion is based on experimental results. A formal verification of the statement is the subject of future research.

Intuitively, r_3 should not be applied, since program $\Pi_1 \setminus \{r_3\}$ is consistent. Hence, the only answer set of Π_1 should be $\{p, r\}$.

Let us check that this intuition is captured by our definition. The first element of $hr(\Pi_1)$ is $H_{\Pi_1} = H'_{\Pi_1} \cup \Pi_p$, where H'_{Π_1} is:

$$H'_{\Pi_1} \left\{\begin{array}{llll} r_1: & p & \leftarrow & r, \text{not } q. \\ r_2: & r. & & \\ & & & \\ r'_3: & s & \leftarrow & r, appl(r_3). \end{array}\right.$$

The only minimal generalized answer set of $hr(\Pi_1)$ is $C = \{p, r\}$. (Notice that $\{p, r, s, appl(r_3)\}$ is a generalized answer set of $hr(\Pi_1)$, but it is not minimal.) Hence, C is the unique answer set of Π_1.

Example 4 Consider the following program:

$$\Pi_2 \left\{\begin{array}{llll} r_1: & p & \leftarrow & \text{not } q. \\ r_2: & r & \leftarrow & \text{not } s. \\ r_3: & q & \leftarrow & t. \\ r_4: & s & \leftarrow & t. \\ & & & \\ r_5: & & \leftarrow & p, r. \\ & & & \\ r_6: & q \stackrel{+}{\leftarrow} . & & \\ r_7: & s \stackrel{+}{\leftarrow} . & & \\ r_8: & t \stackrel{+}{\leftarrow} . & & \\ & & & \\ r_9: & prefer(r_6, r_7). & & \end{array}\right.$$

(Notice that the program consisting of regular rules of Π_2 is inconsistent.) $H_{\Pi_2} = H'_{\Pi_2} \cup \Pi_p$, where H'_{Π_2} is:

$$H'_{\Pi_2} \left\{\begin{array}{llll} r_1: & p & \leftarrow & \text{not } q. \\ r_2: & r & \leftarrow & \text{not } s. \\ r_3: & q & \leftarrow & t. \\ r_4: & s & \leftarrow & t. \\ & & & \\ r_5: & & \leftarrow & p, r. \\ & & & \\ r'_6: & q & \leftarrow & appl(r_6). \\ r'_7: & s & \leftarrow & appl(r_7). \\ r'_8: & t & \leftarrow & appl(r_8). \\ & & & \\ r_9: & prefer(r_6, r_7). & & \end{array}\right.$$

The minimal generalized answer sets of $hr(\Pi_2)$ (and hence candidate answer sets of Π_2) are shown below (we skip the formed by $is\_preferred$):

$$\begin{array}{l} C_1 = \{prefer(r_6, r_7), appl(r_6), q, r\} \\ C_2 = \{prefer(r_6, r_7), appl(r_7), s, p\} \\ C_3 = \{prefer(r_6, r_7), appl(r_8), t, q, s\} \end{array}$$

According to Equation (3), $C_1 \prec C_2$. Hence, $\hat{C}_2$ is not an answer set of Π_2, and the answer sets of Π_2 are $\hat{C}_1$ and $\hat{C}_3$.

Example 5 Consider the following program:

$$\Pi_3 \left\{ \begin{array}{lllll} r_1: & a & \leftarrow & p. \\ r_2: & a & \leftarrow & r. \\ r_3: & b & \leftarrow & q. \\ r_4: & b & \leftarrow & s. \\ & & & \\ r_{5a}: & \leftarrow \text{not } a. & & \\ r_{5b}: & \leftarrow \text{not } b. & & \\ & & & \\ r_6: & p \stackrel{+}{\leftarrow}. & & \\ r_7: & q \stackrel{+}{\leftarrow}. & & \\ r_8: & r \stackrel{+}{\leftarrow}. & & \\ r_9: & s \stackrel{+}{\leftarrow}. & & \\ & & & \\ r_{10}: & prefer(r_6, r_7). & & \\ r_{11}: & prefer(r_8, r_9). & & \end{array} \right.$$

The candidate answer sets of Π_3 are:

$$\begin{array}{l} C_1 = \{prefer(r_6, r_7), prefer(r_8, r_9), \\ \qquad appl(r_6), appl(r_9), p, s, a, b\} \\ C_2 = \{prefer(r_6, r_7), prefer(r_8, r_9), \\ \qquad appl(r_8), appl(r_7), r, q, a, b\} \end{array}$$

Since $C_1 \prec C_2$ and $C_2 \prec C_1$, Π_3 has no answer set.

The reader can also check that the intuitive reasoning described in Example 1 is captured by our semantics, and Π_d, together with $O_1, \ldots, O_3$, entails the expected conclusions.

Applications of Consistency-Restoring Rules

Cr-rules 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 examples of such use.

Example 6 (Dynamic Preferences) Consider diagnostic module DM_0^{cr} used by the agent from Example 1 to solve its diagnostic problems. Suppose we would like to supply him with the following additional information: "*Bulbs blow-ups happen more frequently than power surges unless there is a storm in the area.* " This information can be encoded by the preferences:

$$DM_p : \left\{ \begin{array}{l} prefer(r(brks, T), r(srg, T)) \leftarrow \neg h(storm, 0). \\ prefer(r(srg, T), r(brks, T)) \leftarrow h(storm, 0). \end{array} \right.$$

Let $DM_1^{cr} = DM_0^{cr} \cup DM_p$, and consider recorded history O_4:

$$O_4 : \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(storm, 0). \\ obs(\neg on(b), 1). \end{array} \right.$$

(Recall that if $obs(f, 0)$ is not in O_4 then $obs(\neg f, 0)$ is.) Obviously O_4 requires an explanation. It is storming and therefore the intuitive explanation is $o(srg, 0)$. It is not difficult to check that this is indeed the case. The program $\Pi_d \cup O_4 \cup DM_1^{cr}$ has two candidate answer sets. Due to the second rule of DM_p only one of them, containing srg, is the answer set of the program and hence $o(srg, 0)$ is the explanation of O_4.

The results obtained by the use of cr-rules match the intuitive answers also in more complex cases. Consider recorded history O_5:

$$O_5 : \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(storm, 0) \text{ or } obs(\neg storm, 0). \\ \\ obs(\neg on(b), 1). \\ obs(\neg ab(b), 1). \end{array} \right.$$

This time, we do not know if there has been a storm at 0, but, at time 1, the bulb is observed to be *intact*. Common-sense should tell the agent that there was a power surge. Nothing can be said, however, on whether there has been a storm. Indeed one can check that the answer sets of $\Pi_d \cup O_5 \cup DM_1^{cr}$ contain sets of facts:

$$\begin{array}{c} \{obs(storm, 0), o(srg, 0)\} \\ \{obs(\neg storm, 0), o(srg, 0)\} \end{array}$$

which correspond to the intuitive answers.

To illustrate more complex interaction of planning and diagnostic reasoning let us consider the following elaboration of the Yale Shooting Scenario.

Example 7 *"John has a gun, which he can load. A loaded gun can be fired in order to kill the turkey. Sometimes, if bullets are wet, the gun misfires. Also, sometimes loading the gun fails if John is in a hurry. Finally, in some cases, the turkey could be too big to be killed by a gun."*

The knowledge contained in this story can be represented with the action description, Π_{y1}

$$\left\{ \begin{array}{lll} \multicolumn{3}{l}{\textit{% normally, shooting a loaded gun kills the turkey.}} \\ \quad h(dead, T+1) & \leftarrow & h(loaded(G), T), \\ & & \neg h(ab(shoot), T), \\ & & o(shoot(G), T). \\ \multicolumn{3}{l}{\textit{% shooting always unloads the gun.}} \\ \quad \neg h(loaded(G), T+1) & \leftarrow & h(loaded(G), T), \\ & & o(shoot(G), T). \\ \multicolumn{3}{l}{\textit{% normally, loading makes the gun loaded.}} \\ \quad h(loaded(G), T+1) & \leftarrow & o(load(G), T), \\ & & \neg h(ab(load), T). \\ \multicolumn{3}{l}{\textit{% wet bullets sometimes cause misfiring.}} \\ r_1(T) : h(ab(shoot), T) & \stackrel{+}{\leftarrow} & h(wet\_bullet, T), \\ & & o(shoot(G), T). \\ \multicolumn{3}{l}{\textit{% big turkeys sometimes do not die.}} \\ r_2(T) : h(ab(shoot), T) & \stackrel{+}{\leftarrow} & h(big\_turkey, T), \\ & & o(shoot(G), T). \\ \multicolumn{3}{l}{\textit{% loading sometimes fails if John is in a hurry.}} \\ r_3(T) : h(ab(load), T) & \stackrel{+}{\leftarrow} & h(hurrying, T), \\ & & o(load(G), T). \\ \multicolumn{3}{l}{\textit{% it is more likely for loading to fail than}} \\ \multicolumn{3}{l}{\textit{% for bullets to misfire.}} \\ \multicolumn{3}{l}{prefer(r_3(T_1), r_1(T_2)).} \end{array} \right.$$

(A particular gun becomes a parameter of our actions and a fluent $loaded(G)$ since later we look at scenario with two guns.)

If we add to Π_{y1} the history

$$\begin{cases} obs(wet\_bullet, 0). \\ obs(big\_turkey, 0). \\ obs(hurrying, 0). \\ \textit{\% Closed World Assumption on initial state} \\ \neg obs(F, 0) \leftarrow \text{not } obs(F, 0). \\ \\ hpd(load(g1), 0). \\ hpd(shoot(g1), 1). \end{cases}$$

we obtain a unique answer set, stating that the turkey is dead at time 2. Note that this conclusion does not require the application of cr-rules.

Let us now add the further observation

$$obs(\neg dead, 2).$$

Now, cr-rules are necessary. The resulting program has two answer sets, corresponding to one scenario in which John did not load the gun properly, and to one in which the gun fired as expected, but the turkey was too big to be killed. Notice that the other possible explanation – the wet bullets caused the gun to misfire – does not correspond to any answer set of the program because of the preference $prefer(r_3(T_1), r_1(T_2))$.

As was mentioned before, cr-rules can also be used in construction of a planning module of an agent. The following is an example of such a module which generates plans of *minimal length*.

Example 8 Consider the domain description from Example 7 and program Π_{y2} defined as the union of Π_{y1} and planning module PM_{cr}:

$$\begin{cases} r_4(T): & maxtime(T) \stackrel{+}{\leftarrow} n \leq T. \\ & prefer(r_4(T), r_4(T+1)). \\ \\ r_5(A,T): o(A,T) \stackrel{+}{\leftarrow} maxtime(MT), n \leq T < MT. \end{cases}$$

(Here n stands for the current time of the agent's history - in our case 0.) Cr-rule $r_4(T)$ says that any time can possibly be the maximum planning time of the agent. The second rule gives the preference to shortest plans. The last rules allows the agent the future use of any of his actions.

To illustrate the workings of this module let us consider an initial situation where the turkey is alive and the gun is unloaded; the agent has the goal of killing the turkey, represented as:

$$\begin{cases} goal \leftarrow h(dead, T). \\ \leftarrow \text{not } goal. \end{cases}$$

The goal does not hold at the current moment 0. To avoid inconsistency with the goal constraint the agent may use rules $r_5(A, 0), r_5(A, 1), \ldots$ where time ranges from 0 and the maximum planning time MT. Without the preference relation MT could have been determine by any rule from $r_4(0), r_4(1) \ldots$. The preference, however, forces the agent to select the shortest plan for achieving the goal, in our case $\{o(load(g1), 0), h(shoot(g1), 1)\}$. It may be worth noting a symmetry between rules $r_5(A,T)$ and $r(A,T)$ of our planning a diagnostic modules. The first allowing the agent to consider exogenous actions in the past while the second makes possible the use of its own actions in the future. It is also instructive to compare planning with cr-rules with that using traditional planning module, PM, discussed above. In the latter case a plan of minimum length can be found by multiple calls to lp-solver with $MT = 0, 1, \ldots$. The single call suffices with CR-Prolog planner.

The next example shows another advantage of PM_{cr} with respect to a traditional A-Prolog planner.

Example 9 Let us expand scenario from Example 8 by introducing a new gun, $g2$, which can be used exactly like $g1$. Initially, the turkey is alive, the guns are not loaded, bullets are not wet, the turkey is not big, and John is not hurrying. The goal is to find the sequences of actions which kill the turkey with the smallest number of actions.

Provided that variable G is allowed to range over $\{g1, g2\}$, Π_{y2} can be employed without modifications. It has now two answer sets, corresponding to plans

$$\{o(load(g1), 0), o(shoot(g1), 1)\}$$
$$\{o(load(g2), 0), o(shoot(g2), 1)\}$$

The traditional A-Prolog planner PM may return one of the two intended plans, as well as any of the others, such as:

$$\{o(load(g1), 0), o(load(g2), 0), o(shoot(g1), 1)\}$$
$$\{o(load(g2), 0), o(shoot(g2), 1), o(load(g1), 1)\}$$
$$\ldots$$

Removing these unintended plans is usually a non-trivial task. The use of cr-rules provide us with a rather simple way of doing exactly that.

Related Work

Programs of CR-Prolog closely resemble knowledge systems of (Inoue 1994) – pairs $\langle T, H \rangle$ of non-disjunctive programs in which T represents a background knowledge and H is a set of candidate hypothesis. Though syntactically and even semantically similar, programming methodologies of these two approaches differ considerably. The background theory T of knowledge system seems to be either a collection of integrity constraints or a collection of defaults whose credibility is higher than that of H. This is quite different from structuring of knowledge advocated in this paper. The use of rules from H differ depending on the use of the knowledge system. The emphasis seems to be on default reasoning, where hypothesis are interpreted as defaults and hence rules of H are fired whenever possible. This interferes with search for explanations, which normally favors some form of minimality and applies the rules sparingly. There are some suggestions of using knowledge systems for this purpose by applying different strategy for selection of rules. In our opinion these two types of reasoning are not easily combined together. (In some cases they may even require different representation of knowledge for each of the reasoning tasks.) The notion of knowledge system is further extended in (Sakama & Inoue 2000) by introducing priorities over elements of H viewed as defaults. The new work does not seem to change the methodology of knowledge representation of the original paper. Consequently even our priority relations are quite different from each other. In our future work we plan to investigate the precise relationship between this work and CR-Prolog.

Now let us look at a formalism from (Lin & You 2001) where the authors point out problems linked with the "classical" definition of abductive logic programs given by Kakas & Mancarella (1990), and propose a new definition of abduction. The problems underlined by Lin & You appear to be linked with the use of negation as failure in place of classical negation, which is typical of traditional logic programming, as well as of logic programming under the stable model semantics.

To see the problem consider program $P^1_{ly} = \{q \leftarrow a.\}$, set of abducibles $\mathcal{A} = \{a, b\}$, and observation q. The minimal abductive explanation for q given by $\langle P^1_{ly}, \mathcal{A}\rangle$ is $E_1 = \{a\}$. In (Lin & You 2001), the authors notice that E_1 is often seen as representing its completion under the Closed World Assumption, i.e. E_1 is really a shorthand for $E_2 = \{a, \neg b\}$. They, correctly, argue that such an interpretation is unintuitive - the Closed World Assumption should not be applied to b, since abducibles "are assumptions which one can make one way or the other" (Lin & You 2001).

We believe however that this criticism is not applicable to our formalism. In the semantics of CR-Prolog, abduction is used only to select sets of cr-rules needed to restore consistency of the reasoner's beliefs. Hence, according to our semantics, cr-rules normally do not apply, which justifies the Closed World Assumption for our abducibles - atoms formed by predicate $appl$.

Moreover, the problem primarily exists in languages not containing classical negation. The availability of classical negation in our language allows for more accurate formalization of knowledge and removes many difficulties caused by unjustified use of negation as failure. We will illustrate this point by the "canCross example" from (Lin & You 2001).

In this example, the authors consider a scenario in which a boat can be used to cross a river, if the boat is not leaking. If it is leaking, the river can be still crossed if there is a bucket available to scoop the water out of the boat. The agent observes that someone is crossing the river, and must explain this fact.

They formalize the domain with program P^2_{ly}

$$\begin{array}{l} canCross \leftarrow boat, \text{not } leaking. \\ canCross \leftarrow boat, leaking, hasBucket. \end{array}$$

and notice that, under the definition given by Kakas & Mancarella, the observation has a unique minimal explanation, $\{boat\}$. The closed world interpretation discussed above gives us a "complete" explanation $\{boat, \neg leaking, \neg hasBucket\}$, which is correct. Unfortunately, perfectly plausible explanations like $\{boat, leaking, hasBucket\}$ are not captured by this definition. The alternative would be to interpret $boat$ as incomplete explanation which could be completed by assigning values to other abducibles. As pointed out by the authors this does not work either since $\{boat, leaking, \neg hasBucket\}$ is not an explanation. In their paper the authors propose semantics of abductive programs aimed at solving this problem.

Let us now look at treatment of the same example in CR-Prolog. We propose the following formalization:

$$P^2_{cr} \left\{ \begin{array}{lrl} r_1 : & canCross & \leftarrow boat, \neg leaking. \\ r_2 : & canCross & \leftarrow boat, leaking, hasBucket. \\ & \multicolumn{2}{l}{\leftarrow \text{not } canCross.} \\ a_1 : & boat & \stackrel{+}{\leftarrow} . \\ a_2 : & \neg boat & \stackrel{+}{\leftarrow} . \\ a_3 : & leaking & \stackrel{+}{\leftarrow} . \\ a_4 : & \neg leaking & \stackrel{+}{\leftarrow} . \\ a_5 : & hasBucket & \stackrel{+}{\leftarrow} . \\ a_6 : & \neg hasBucket & \stackrel{+}{\leftarrow} . \end{array} \right.$$

Notice that rule r_1 uses classical negation (instead of the default negation used in the first formalization). This is certainly a more faithful translation of the problem's knowledge which contains nothing about beliefs of closed world assumptions. Notice also that the observation, represented as a constraint, becomes part of our knowledge. The reader can check that the program has two answer sets corresponding to explanations $E_1 = \{boat, \neg leaking\}$ and $E_2 = \{boat, leaking, hasBucket\}$. Note that these explanations do not depend on unspecified values of "abducible" - E_1 will remain a good intuitive explanation of the observation even after we discover whether there is a bucket in the boat.

It may also be instructive to see how preferences of CR-Prolog can be used to select some "preferred" explanations of our observation. For example, if we believe that boats are rarely leaking, we can introduce preference

$$prefer(a_4, a_3).$$

The new program, P^{cr}_3, will generate only explanation E_1. However, if later we were to observe that the boat is leaking, adding this piece of information to the program would make it retract E_1 and return E_2.

Another work relevant to our research is the introduction of *weak constraints* in DLV (Buccafurri, Leone, & Rullo 1997a; 1997b; Calimeri *et al.* 2002). Intuitively, a weak constraint is a constraint that can be violated, if this is needed to obtain an answer set of a program. To each weak constraint, a *weight* is assigned, indicating the cost of violating the constraint[4]. A preferred answer set of a program with weak constraints is one that minimizes the sum of the weights of

[4]To be precise, two different "costs", weight and level, are assigned to weak constraints, but in our discussion we only consider the weight, since even levels do not seem to solve the problem.

the constraints that the answer set violates. Consider for example program Π_{dlv} of DLV:

$$\left\{\begin{array}{l} a \text{ or } b. \\ :\sim a.\ [1:] \\ :\sim b.\ [2:] \end{array}\right.$$

where the first weak constraint (denoted by symbol $:\sim$) has weight 1 and the second has weight 2. In order to satisfy the first rule, the answer sets of Π_{dlv} must violate one of the constraints. Since violating the first constraint has a lower cost than violating the second, the preferred answer set of Π_{dlv} is $\{a\}$.

Weak constraints are of course similar to our cr-rules and weights can often play a role of preferences. The main disadvantage of using weak constraints instead of cr-rules is that weights induce a total order on the weak constraints of the program, as opposed to the partial order that can be specified on cr-rules. This seems to be a key difference in the formalization of some forms of common-sense knowledge, like the one from Example 6. To the best of our knowledge, there is no formalization of this domain in DLV with weak constraints, that, given recorded history O_5, concludes that there are two possible alternatives compatible with O_5:

$$\begin{array}{l} \{obs(storm, 0), o(srg, 0)\} \\ \{obs(\neg storm, 0), o(srg, 0)\} \end{array}$$

To see the problem, consider, for example, the following DLV formalization of our diagnostic module DM_1^{cr} with dynamic preferences: $DM_{wk} = DM_0 \cup \Pi_{wk}$, where Π_{wk} is:

$$\Pi_{wk} \left\{\begin{array}{l} :\sim o(brks, T), h(storm, 0).\ [4:] \\ :\sim o(srg, T), h(storm, 0).\ [1:] \\ :\sim o(brks, T), \neg h(storm, 0).\ [1:] \\ :\sim o(srg, T), \neg h(storm, 0).\ [4:] \end{array}\right.$$

The first two weak constraints say that, if a storm occurred, assuming that action $brks$ occurred has a cost of 4, while assuming that action srg occurred has a cost of 1. The last two weak constraints say that, if a storm did not occur, assuming that action $brks$ occurred has a cost of 1, while assuming that action srg occurred has a cost of 4. The selection of particular weights is fairly arbitrary, but it captures the corresponding dynamic preferences.

The only possible explanation of recorded history O_5 is the occurrence of srg at time 0. Hence, $\Pi_d \cup O_5 \cup DM_{wk}$ has two candidate answer sets, containing, as expected, the two set of facts above. Unfortunately, the answer set corresponding to the second set of facts has a total cost of 4, while the answer set corresponding to the first explanation has a cost of 1. This forces the reasoner to prefer the first answer set, and to assume, without any sufficient reason, the existence of a storm.

There are however some classes of programs of CR-Prolog which can be reduced to DLV programs with weak constraints. Study of such classes may be useful not only for improving our understanding of both formalisms, but also for using efficient computation engine of DLV for CR-Prolog computations.

Conclusions and Future Work

In this paper, we extended A-Prolog by cr-rules with preferences, gave the semantics of the new language, CR-Prolog, and demonstrated how it 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. Formalizations in CR-Prolog however seem to be natural, reasonably elaboration tolerant, and, we hope, can be efficiently implemented.

So far, we have implemented a naïve algorithm for computing answer sets of programs with cr-rules. We are currently working on the development and implementation of more efficient reasoning algorithms. We also plan to study the ramifications of the use of different preference relations in the definition of answer sets of CR-Prolog.

Acknowledgments

This work was partially supported by United Space Alliance under Research Grant 26-3502-21 and Contract COC6771311, and by NASA under Contracts 1314-44-1476 and 1314-44-1769. The authors would like to thank the anonymous reviewers for their useful comments which contributed to the improvement of this paper.

References

Aiello, L., and Massacci, F. 2001. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic*.

Balduccini, M., and Gelfond, M. 2002. Diagnostic reasoning with a-prolog. *Theory and Practice of Logic Programming*. (to appear).

Balduccini, M.; Gelfond, M.; and Nogueira, M. 2000. A-prolog as a tool for declarative programming. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE'2000)*, 63–72.

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

Baral, C., and Lobo, J. 1997. Defeasible specifications in action theories. In *Proceedings of IJCAI-97*, 1441–1446.

Buccafurri, F.; Leone, N.; and Rullo, P. 1997a. Adding weak constraints to disjunctive datalog. In *Proceedings of the 1997 Joint Conference on Declarative Programming APPIA-GULP-PRODE'97*.

Buccafurri, F.; Leone, N.; and Rullo, P. 1997b. Strong and weak constraints in disjunctive datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, volume 1265, 2–17.

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)*.

Dimopoulos, Y.; Koehler, J.; and Nebel, B. 1997. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the 4th European Conference on Planning*, volume 1348, 169–181.

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.

Hayes, P. J., and McCarthy, J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502.

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.

Inoue, K. 1994. Hypothetical reasoning in logic programs. *Journal of Logic Programming* 18(3):191–227.

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

Kakas, A. C.; Kowalski, R. A.; and Toni, F. 1998. *The role of abduction in logic programming*. Handbook of Logic in Artificial Intelligence and Logic Programming. Oxford University Press. 235–324.

Lifschitz, V. 1999. Answer set planning. In *Proceedings of IJCSLP 99*.

Lin, F., and You, J.-H. 2001. Abduction in logic programming: A new definition and an abductive procedure based on rewriting. In *International Joint Conference on Artificial Intelligence (IJCAI'01)*.

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

McCain, T., and Turner, H. 1995. A causal theory of ramifications and qualifications. *Artificial Intelligence* 32:57–95.

McCain, T., and Turner, H. 1997. Causal theories of action and change. In *Proceedings of AAAI-97*, 460–465.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An a-prolog decision support system for the space shuttle. In *AAAI Spring 2001 Symposium*.

Pareto, V. 1896. Cours d'economie politique professe a l'universite de lausanne.

Pontelli, E.; Balduccini, M.; and Bermudez, F. 2003. Nonmonotonic reasoning on beowulf platforms. In *PADL 2003*. (to appear).

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123:185–222.

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

Soininen, T., and Niemela, I. 1999. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*.

Turner, H. 1997. Reprenting actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming* 31(1-3):245–298.