Chapter 1

REASONING AGENTS IN DYNAMIC DOMAINS

Chitta Baral
Department of Computer Sc. and Engg.
Arizona State University, Tempe, AZ 85287
chitta@asu.edu

Michael Gelfond
Department of Computer Sc.
Texas Tech University, Lubbock, TX 79409
michael.gelfond@coe.ttu.edu

Abstract The paper discusses an architecture for intelligent agents based on the use of *A-Prolog* - a language of logic programs under the answer set semantics. *A-Prolog* is used to represent the agent's knowledge about the domain and to formulate the agent's reasoning tasks. We outline how these tasks can be reduced to answering questions about properties of simple logic programs and demonstrate the methodology of constructing these programs.

Keywords:
intelligent agents, logic programming and nonmonotonic reasoning.

1. INTRODUCTION

This paper is a report on the attempt by the authors to better understand *the design of software components of intelligent agents capable of reasoning, planning and acting in a changing environment.* The class of such agents includes, but is not limited to, intelligent mobile robots, softbots, immobots, intelligent information systems, expert systems, and decision-making systems. The ability to design intelligent agents (IA) is crucial for such diverse tasks as space exploration, intelligent communication with the Internet, and development of various types of control systems. Despite the substantial progress in the work on IA achieved in the last decade we are still far from a clear understanding of the basic principles and techniques needed for their design. The problem is complicated by the fact that IA differ from more traditional software systems in several important aspects:

• An agent could have a large amount of knowledge about the domain in which it is intended to act, and about its own capabilities and goals.

• It should be able to frequently expand this knowledge by new information coming from observations, communication with other agents, and awareness of its own actions.

• All this knowledge cannot be explicitly represented in the agent's memory. This implies that the agent should be able to reason, i.e. to extract knowledge stored there implicitly.

• Finally, the agent should be able to use its knowledge and its ability to reason to rationally plan and execute its actions.

These observations imply that solid theoretical foundations of agent design should be based on theories of knowledge representation and reasoning. Work reported in this paper is based on two such theories: theory of logic programming and nonmonotonic reasoning (Baral and Gelfond, 1994; Lifschitz, 1996; Marek and Truszczyński, 1993) and theory of actions and change (Sandewall, 1998).

The latter develops the ontology and basic relations needed for modeling the agent's domain. The former provides a logic for representing and reasoning with the domain knowledge. This logic is more expressive than classical first-order predicate calculus (Dantsin et al., 1997). This additional expressibility is needed to represent defaults, causal relations, various forms of transitive closures, etc. The entailment relation of the logic is nonmonotonic, i.e. it allows the reasoner to withdraw previously made conclusions when new information becomes available. This property substantially simplifies the process of assimilating new information. There are efficient systems implementing rather general reasoning algorithms developed in logic programming community. In this paper we show how these systems can be used to implement specific planning, explanation finding, and plan checking algorithms in a simple observe-think-act style agent architecture.

There are several other approaches which use logic as a basis for agent design. They differ primarily by the languages and the logics in which the agent's knowledge is specified, by the algorithms used by the agent to perform its reasoning tasks, and by the agent's architecture. For instance, the Toronto School of Cognitive Robotics (Levesque et al., 1997) uses modified and substantially expanded versions of the situation calculus of (McCarthy and Hayes, 1969) to specify the agent knowledge. Despite occasional use of nonmonotonic entailment (primarily in the form of circumscription (McCarthy, 1980)) the Toronto School seems to prefer to stay as close as possible to the entailment relation and reasoning algorithms of classical logic. In their approach the agent's behavior is determined by a program written in a programming language Golog (Levesque et al., 1997) or one of its variants. Such a program allows procedural constructs such as sequences, loops, conditionals, and has non-deterministic operators and test conditions where the non-determinism is resolved by the interpreter so that the overall plan is executable. To interpret and execute these constructs the agent uses the situation calculus based entailment relation described above. The main program can be complemented by an execution monitor capable of modifying plans invalidated by exogenous actions (DeGiacomo et al., 1998), as well as by other special purpose reasoning modules. In (Kowalski, 1995; Kowalski and Sadri, 1999), the authors investigate an architecture based on a variant of observe-think-act cycle. The internal state of the agent is determined by a collection of standard logic programming rules, integrity constraints, condition-actions rules, etc. The approach is somewhat independent of a particular entailment relation but the authors seem to favor entailments associated with the Clark's completion (Clark, 1978). The thinking part of a cycle is performed by a combination of traditional logic programming algorithms, abduction (Denecker and De Schreye, 1998; Kakas et al., 1998) and forward reasoning by means of integrity constraints.

Even though all these approaches develop in parallel they share many common insights and ideas. Sometimes this is a result of direct influence and sometimes (and probably more often) the similarities are determined by the common subject of study. The technical differences between the approaches are however rather large and we believe that the complete understanding of their pros and cons will require a substantial amount of further work. This paper is aimed at explaining our approach and we do not make any attempt at the comparison.

2. MODELING THE AGENT

In this paper we adopt the following simplifying assumptions about the agents and their environments:

1. The dynamics of the agent's environment is viewed as a (possibly infinite) transition diagram whose states are sets of fluents (i.e.,

statements whose truth depends 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.

These assumptions hold in many realistic domains and are suitable for a broad class of applications. In many domains, however, the effects of actions and the truth of observations can only be known with a substantial degree of uncertainty which cannot be ignored in the modeling process. In this case our basic approach can be expanded by introducing probabilistic information. We prefer to start our investigation with the simplest case and work our way up the ladder of complexity only after this simple case is reasonably well understood. The assumptions above determine the structure of the agent's knowledge base. It consists of two parts. The *first part*, called an *action description*, specifies the transition diagram of the agent. It contains descriptions of actions and fluents relevant to the domain together with the definition of possible successor states to which the system can move after an action a is executed in a state σ. Due to the size of the diagram the problem of finding its concise specification is far from being trivial. Its solution requires a good understanding of the nature of causal effects of actions in the presence of complex interrelations between fluents. An additional level of complexity is added by the fact that, unlike standard mathematical reasoning, causal reasoning is nonmonotonic, i.e., new information about the domain can cause a reasoning agent to withdraw its previously made conclusions. Nonmonotonicity is partly caused by the need for representing defaults, i.e., statements of the form "Normally, objects of type A have property P". A default that is often used in causal reasoning is the so called inertia axiom (McCarthy and Hayes, 1969). It says that "Normally, actions do not change the values of fluents". The problem of finding a concise and accurate representation of this default, known as the Frame Problem, substantially influenced AI research during the last twenty years (Shanahan, 1997). The *second part* of the agent's knowledge contains observations made by the agent together with a record of its own actions. It defines a collection of paths in the diagram which can be interpreted as the domain's possible trajectories traversed so far. If the agent's knowledge is complete (i.e., it has complete information about the initial state and the action occurrences) and its actions are deterministic then there is only one such path. We believe that a good theory of agents should contain a logical language (equipped with a consequence relation) which would be capable of representing both types of the agent's knowledge.

A-Prolog

In this paper the language of choice is *A-Prolog* - a language of logic programs under the answer set semantics (Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1991). An *A-Prolog* program consists of a signature Σ and a collection of rules of the form

$$head \leftarrow body \tag{1.1}$$

where the *head* is empty or consists of a literal l_0 and *body* is of the form $l_1, \dots, l_m, not\ l_{m+1}, \dots, not\ l_n$ where l_i's are literals over Σ. A literal is an atom p or its negation $\neg p$. If the *body* is empty we replace $\leftarrow$ by a period. While $\neg p$ says that p is false, *not* p has an epistemic character and can be read as "there is no reason to believe that p is true". The symbol *not* denotes a non-standard logical connective often called *default negation* or *negation as failure.* An *A-Prolog* program Π can be viewed as a specification given to a rational agent for constructing beliefs about possible states of the world. Technically these beliefs are captured by the notion of *answer set* of a program Π. By $ground(\Pi)$ we denote a program obtained from Π by replacing variables by the ground terms of Σ. By answer sets of Π we mean answer sets of $ground(\Pi)$. If Π consists of rules not containing default negation then its answer set S is the smallest set of ground literals of Σ which satisfies two conditions:

1. S is closed under the rules of $ground(\Pi)$, i.e. for every rule (1.1) in Π, either there is a literal l in its body such that $l \notin S$ or its non-empty head $l_0 \in S$.

2. if S contains an atom p and its negation $\neg p$ then S contains all ground literals of the language.

It is not difficult to show that there is at most one set $Cn(\Pi)$ satisfying these conditions.

Now let Π be an arbitrary ground program of *A-Prolog.* For any set S of ground literals of its signature Σ, let Π^S be the program obtained from Π by deleting

(i) each rule that has an occurrence of *not* l in its body with $l \in S$,

(ii) all occurrences of *not* l in the bodies of the remaining rules.

Then S *is an answer set of* Π *if*

$$S = Cn(\Pi^S). \tag{1.2}$$

In this paper we limit our attention to *consistent* programs, i.e. programs with at least one consistent answer set. Let S be an answer set of Π.

A ground literal l is *true* in S if $l \in S$; *false* in S if $\neg l \in S$. This is expanded to conjunctions and disjunctions of literals in a standard way. A query Q is *entailed* by a program Π ($\Pi \models Q$) if Q is true in all answer sets of Π. Queries $l_1 \wedge \ldots \wedge l_n$ and $\overline{l_1} \vee \ldots \vee \overline{l_n}$ (where for an atom p, $\overline{p}$ denotes $\neg p$, and $\overline{\neg p}$ denotes p) are called complementary. If Q and $\overline{Q}$ are complementary queries then Π's answer to Q is *yes* if $\Pi \models Q$; *no* if $\Pi \models \overline{Q}$, and *unknown* otherwise.

Here are some examples. Assume that signature Σ contains two object constants a and b. The program

$$\Pi_1 = \{\neg p(X) \leftarrow not\ q(X). \qquad q(a).\}$$

has the unique answer set $S = \{q(a), \neg p(b)\}$. The program

$$\Pi_2 = \{p(a) \leftarrow not\ p(b). \qquad p(b) \leftarrow not\ p(a).\}$$

has two answer sets, $\{p(a)\}$ and $\{p(b)\}$. The programs

$$\Pi_3 = \{p(a) \leftarrow not\ p(a).\} \text{ and } \Pi_4 = \{p(a). \quad \leftarrow p(a).\}$$

have no answer sets.

It is easy to see that programs of *A-Prolog* are *nonmonotonic*. ($\Pi_1 \models \neg p(b)$ but $\Pi_1 \cup \{q(b).\} \not\models \neg p(b)$.) *A-Prolog* is closely connected with more general nonmonotonic theories. In particular, as was shown in (Marek and Truszczyński, 1989; Gelfond and Lifschitz, 1991), there is a simple and natural mapping of programs of *A-Prolog* into a subclass of Reiter's default theories (Reiter, 1980). (Similar results are also available for Autoepistemic Logic of Moore (Moore, 1985).)

Specifying transition diagrams

To describe the agent's transition diagram, the domain's history and the type of queries available to the agent we use action languages (Gelfond and Lifschitz, 1992) which can be viewed as formal models of parts of natural language that are used for talking about the effects of actions. A particular action language reflects properties of a domain and the abilities of an agent. In this paper we define a class of action languages $\mathcal{AL}$ which combine ideas from (Baral, 1995; McCain and Turner, 95; Baral et al., 1995; Baral et al., 1997; Guinchiglia and Lifschitz, 1998). Languages $\mathcal{AL}(\Sigma)$ from this class will be parameterized with respect to a signature Σ which we normally assume fixed and omit from our notation. The simplicity of $\mathcal{AL}$ as a language makes it suitable for illustrating our methodology, which is a primary goal of this paper. The methodology however is applicable to languages with much richer ontology.

We follow a slightly modified view of (Lifschitz, 1997) and divide an action language into three parts: *action description language*, *history description language*, and *query language*. We start with defining the action description part of $\mathcal{AL}$, denoted by $\mathcal{AL}_d$. Its signature Σ will consist of two disjoint, non-empty sets of symbols: the set $\mathbf{F}$ of fluents and the set $\mathbf{A}$ of *elementary actions*. A set $\{a_1, \ldots, a_n\}$ of elementary actions is called a *compound* action. It is interpreted as a collection of elementary actions performed simultaneously. By actions we mean both, elementary and compound actions. By *fluent literals* we mean fluents and their negations. By $\bar{l}$ we denote the fluent literal complementary to l. A set S of fluent literals is called *complete* if for any $f \in \mathbf{F}$, $f \in S$ or $\neg f \in S$. An action description of $\mathcal{AL}_d(\Sigma)$ is a collection of propositions of the form

1. $causes(a_e, l_0, [l_1, \ldots, l_n])$,
2. $caused(l_0, [l_1, \ldots, l_n])$, and
3. $impossible\_if(a, [l_1, \ldots, l_n])$

where a_e and a are elementary and arbitrary actions respectively and $l_0, \ldots, l_n$ are fluent literals from Σ. The first proposition says that, if the action a were to be executed in a situation in which $l_1, \ldots, l_n$ hold, the fluent literal l_0 will be caused to hold in the resulting situation. Such propositions are called *dynamic causal laws*. (The restriction on a_e being elementary is not essential and can be lifted. We require it to simplify the presentation). The second proposition, called a *static causal law*, says that, in an arbitrary situation, the truth of fluent literals, $l_1, \ldots, l_n$, (often called the body of (2)) is sufficient to cause the truth of l_0. The last proposition says that action a can not be performed in any situation in which $l_1, \ldots, l_n$ hold. Notice that here a can be compound, e.g. $impossible(\{a_1, a_2\}, [\,])$ means that elementary actions a_1 and a_2 cannot be performed concurrently. An action description $\mathcal{A}$ of $\mathcal{AL}_d$ defines a transition diagram describing effects of actions on the possible states of the domain. A *state* is a complete and consistent set σ of fluent literals such that σ is *closed* under the static causal laws of $\mathcal{A}$, i.e. for any static causal law (2) of $\mathcal{A}$, if $\{l_1, \ldots, l_n\} \subseteq \sigma$ then $l_0 \in \sigma$. States serve as the nodes of the transition diagram. Nodes σ_1 and σ_2 are connected by a directed arc labeled by an action a if σ_2 may result from executing a in σ_1. (To simplify the diagram we do not distinguish between links labeled by an elementary action a_e and ones labeled by $\{a_e\}$.) The set of all states that may result from executing a in a state σ will be denoted by $res(a, \sigma)$. Defining this set for action descriptions of increasingly complex action languages seems to be one of the main issues in the development of action theories. Several fixpoint definitions of this set were recently given by different authors; see for instance (McCain and Turner, 95; Lin, 95; Guinchiglia and Lifschitz, 1998). The definition we give in this paper is based on the semantics of *A-Prolog* and is similar in spirit to the work in (Baral, 1995). First let us consider the following program:

$$
\left.\begin{array}{lllll}
1. & \mathit{holds\_at}(L,T') & \leftarrow & \mathit{next}(T,T'), \\
 & & & \mathit{causes}(A_e,L,P), \\
 & & & \mathit{occurs\_at}(A_e,T), \\
 & & & \mathit{hold\_at}(P,T). \\
2. & \mathit{holds\_at}(L,T) & \leftarrow & \mathit{caused}(L,P), \\
 & & & \mathit{hold\_at}(P,T). \\
3. & \mathit{holds\_at}(L,T') & \leftarrow & \mathit{next}(T,T'), \\
 & & & \mathit{holds\_at}(L,T), \\
 & & & \mathit{not}\ \mathit{holds\_at}(\overline{L},T'). \\
4. & \mathit{hold\_at}([\,],T). & & \\
5. & \mathit{hold\_at}([L|\mathit{Rest}],T) & \leftarrow & \mathit{holds\_at}(L,T), \\
 & & & \mathit{hold\_at}(\mathit{Rest},T). \\
6. & \neg \mathit{occurs\_at}(A_e,T) & \leftarrow & \mathit{not}\ \mathit{occurs\_at}(A_e,T). \\
7. & \neg \mathit{occurs\_at}(A_c,T) & \leftarrow & \mathit{member}(A_e,A_c), \\
 & & & \neg \mathit{occurs\_at}(A_e,T). \\
8. & \neg \mathit{occurs\_at}(A_c,T) & \leftarrow & \mathit{member}(A_e,\overline{A_c}), \\
 & & & \mathit{occurs\_at}(A_e,T). \\
9. & \mathit{occurs\_at}(A_c,T) & \leftarrow & \mathit{not}\ \neg \mathit{occurs\_at}(A_c,T). \\
10. & & \leftarrow & \mathit{impossible\_if}(A,P), \\
 & & & \mathit{hold\_at}(P,T), \\
 & & & \mathit{occurs\_at}(A',T), \\
 & & & \mathit{subset}(A,A').
\end{array}\right\} \Pi(N)
$$

We use A_e, A_c, and A as variables for elementary, compound, and arbitrary actions respectively; L and P are variables for fluent literals and (finite) sets of fluent literals, and T and T' are variables for integers from some interval $[0, N]$. Integers from this interval will be later interpreted as time points. The program uses the list notation $[\,]$ and predicate symbols *member* and *subset* denoting the standard membership and proper subset relations; $\mathit{member}(A_e, \overline{A_c})$ holds when A_e is an elementary action not belonging to A_c. Rules one and two of $\Pi(N)$ describe the effects of causal laws. The predicate symbol $\mathit{holds\_at}(L,T)$ denotes the relation "A fluent literal L is true at moment T"; $\mathit{hold\_at}(P,T)$ denotes the expansion of this relation to sets of fluents; $\mathit{occurs\_at}(A,T)$ indicates that an action A occurs at moment T. The relation $\mathit{next}(T,T')$ is satisfied by two consecutive moments of time from the interval $[0, N]$. Rule three encodes the law of inertia. The default nature of this law is nicely captured by the use of default negation of *A-Prolog*. Rules four and five define $hold\_at(P, T)$. Normally, the program $\Pi(N)$ will be used in conjunction with a complete list of elementary actions which occurred in the domain. The completeness is expressed by the closed world assumption for elementary actions encoded by rule six. Rules seven and eight define the complete list of compound actions which do not occur at moment T. Rule nine uses the closed world assumption to define compound action which does occur at that moment. (It is easy to see that if $a_1, \dots, a_k$ is the complete list of elementary actions which occur at moment T then the only compound action which occur at this moment is $\{a_1, \dots, a_k\}$.) The last rule with the empty head is used to insure that impossible actions are indeed impossible.

By $\Pi(1)$ we denote the program obtained from $\Pi(N)$ by replacing the time variables by 0 and 1. Now we define the transition relation determined by an action description $\mathcal{A}$.

Definition 1 For any action a and state σ_1, a state σ_2 is a *successor state* of a on σ_1 if there is an answer set S of

$$\Pi(1) \cup \mathcal{A} \cup \{holds\_at(l, 0) \ : \ l \in \sigma_1\} \cup \{occurs\_at(a_i, 0) \ : \ a_i \in a\}$$

such that $\sigma_2 = \{l \ : \ holds\_at(l, 1) \in S\}$.

It is essential to notice that Definition 1 allows us to *reduce computing successor states of the transition diagram representing our dynamic domain to computing answer sets of comparatively simple logic programs.* For domains without concurrent actions this definition is equivalent to one from (McCain and Turner, 95). The idea however is not limited to $\mathcal{AL}_d$ and can be used to define a larger range of transition functions.

We say that an action description $\mathcal{A}$ of $\mathcal{AL}_d$ is *deterministic* if for any state σ and any action a from Σ there is at most one successor state, i.e., the cardinality of the set $res(a, \sigma)$ is at most one. The following example shows that action descriptions of $\mathcal{AL}_d$ can be nondeterministic.

Example 1 Let $\mathcal{A}_1$ be an action description

$causes(a, f, [\ ]). \quad caused(\neg g_1, [f, g_2]). \quad caused(\neg g_2, [f, g_1]).$

Using the Definition 1 it is easy to show that the transition diagram of $\mathcal{A}_1$ can be given as follows:

$res(a, \{f, g_1, \neg g_2\}) = \{\{f, g_1, \neg g_2\}\}$
$res(a, \{f, \neg g_1, g_2\}) = \{\{f, \neg g_1, g_2\}\}$

$res(a, \{f, \neg g_1, \neg g_2\}) = \{\{f, \neg g_1, \neg g_2\}\}$
$res(a, \{\neg f, g_1, g_2\}) = \{\{f, g_1, \neg g_2\}, \{f, \neg g_1, g_2\}\}$
$res(a, \{\neg f, g_1, \neg g_2\}) = \{\{f, g_1, \neg g_2\}\}$
$res(a, \{\neg f, \neg g_1, g_2\}) = \{\{f, \neg g_1, g_2\}\}$
$res(a, \{\neg f, \neg g_1, \neg g_2\}) = \{\{f, \neg g_1, \neg g_2\}\}$

Action descriptions not containing static causal laws are deterministic and hence the addition of such laws adds to expressive power of the language. The following proposition gives a sufficient condition guaranteeing that this property holds.

Proposition 1 *Any action description $\mathcal{A}$ of $\mathcal{AL}_d$ with static causal laws containing at most one literal in the body is deterministic.*

The proposition is meant to illustrate how the syntactic form of action description can be used to learn important properties of the corresponding transition relations. Similar results in somewhat different context can be found in (Pinto, 1999; Lin, 2000).

Specifying the history

We now describe a language $\mathcal{AL}_h$ for specifying the history of the agent's domain. To do that we expand action signature Σ by integers $0, 1, 2, \ldots$, which are used to denote time points in the actual evolution of the system. If such evolution is caused by a sequence of consecutive actions $a_1, \ldots, a_n$ then 0 corresponds to the initial situation and k ($0 < k \leq n$) corresponds to the end of the execution of $a_1, \ldots, a_k$.

The domain's past is described by a set, Γ, of axioms of the form

1. $happened(a, k)$.
2. $observed(l, k)$.

where a's are elementary actions. The first axiom says that (elementary) action a has been executed at time k; the second axiom indicates that fluent literal l was observed to be true at time k. The axioms of Γ will be often referred to as *observations*. Every set Γ of observations uniquely defines the *current moment of time*, $t_c(\Gamma)$. If $\Gamma = \emptyset$ then $t_c(\Gamma) = 0$; If every occurrence, t', of time in Γ is less than or equal to some t such that $happened(a, t) \in \Gamma$ then $t_c(\Gamma) = t + 1$; otherwise, $t_c = max(\{t : observed(l, t) \in \Gamma\})$. ($\Gamma$ will be omitted whenever possible.)

A set of axioms defines the collection of paths in a transition diagram T which can be interpreted as possible histories of the domain represented by T. (As usual, by a path we mean a sequence $\langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$, where σ_0 is a state and for $1 \leq i \leq n$, $\sigma_i \in res(a_i, \sigma_{i-1})$.) If our knowledge about the initial situation is complete and actions are deterministic then there is only one such path. A pair $\langle \mathcal{A}, \Gamma \rangle$, where $\mathcal{A}$ is an action description and Γ is a set of observations, is called a *domain description.* The following definition refines the intuition behind the meaning of observations.

Definition 2 Let $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ be a domain description with the current time n, T be the transition diagram defined by $\mathcal{A}$, and $H = \langle \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n \rangle$ be a path in T. We say that H is a *possible history* of $\mathcal{D}$ if

1. action a_i is the i-th action in H iff $a_i = \{a : happened(a, i{-}1) \in \Gamma\}$.

2. If $observed(l, k) \in \Gamma$ then $l \in \sigma_k$.

A domain description $\mathcal{D}$ is called *consistent* if it has a non-empty set of possible histories.

Queries and the consequence relation

Let us now assume that at each moment of time an agent maintains a knowledge base in the form of a domain description $\mathcal{D}$. Various reasoning tasks of an agent can be reduced to answering queries about properties of the agent's domain. The corresponding query language, $\mathcal{L}_q$, includes the following queries:

1. $holds\_at(l, t)$.

2. $currently(l)$.

3. $holds\_after(l, [a_n, \ldots, a_1], t)$.

Query (1) asks if fluent literal l holds at time t. Query (2) asks if l holds at the current moment of time. The last query is hypothetical and is read as: "Is it true that a sequence $a_1, \ldots, a_n$ of actions could have been executed at the time $0 \leq t \leq t_c$, and if it were, then fluent literal l would be true immediately afterwards". If $t < t_c$ and the sequence $a_1, \ldots, a_n$ is different from the one that actually occurs at t then the corresponding query expresses a counterfactual. If $t = t_c$ then the query expresses a hypothesis about the system's future behavior. (In this case we often omit t from the query and simply write $holds\_after(l, [a_n, \ldots, a_1])$.) The definition below formalizes this intuition. Let $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ be a domain description with the current moment n, and T be the transition diagram defined by $\mathcal{A}$.

1. a query $holds\_at(l, t)$ is a *consequence* of $\mathcal{D}$ if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of $\mathcal{D}$, $0 \leq t \leq n$, and we have $l \in \sigma_t$;

2. a query $currently(l)$ is a *consequence* of $\mathcal{D}$ if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of $\mathcal{D}$, we have $l \in \sigma_n$;

3. a query $holds\_after(l, [a'_m, \dots, a'_1], t)$ is a *consequence* of $\mathcal{D}$ if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of $\mathcal{D}$, $0 \leq t \leq n$ and $a'_1, \dots, a'_m$ is executable in σ_t and for any path $\sigma'_0, a'_1, \sigma'_1, \dots, a'_m, \sigma'_m$ of T such that $\sigma'_0 = \sigma_t$, $l \in \sigma'_m$.

The consequence relation between query Q and domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ will be denoted by $\Gamma \models_{\mathcal{A}} Q$.

Architecture for intelligent agents

We now demonstrate how the notion of consistency of a domain description and the consequence relation $\Gamma \models_{\mathcal{A}} Q$ can be used to describe an architecture of an intelligent agent. In this architecture the set of elementary actions is divided into two parts: the actions which can be performed by an agent and exogenous actions performed by nature or other agents in the domain. We assume that at each moment t of time the agent's memory contains domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ and a partially ordered set $\mathcal{G}$ of agent's *goals*. By a goal we mean a finite set of fluent literals the agent wants to make true. Partial ordering corresponds to comparative importance of goals. The agent operates under the assumption that it was able to observe all the exogenous actions. This assumption can however be contradicted by observations which may force the agent to conclude that some exogenous actions in the past remained unobserved. An agent will repeatedly execute the following steps

1. observe the world and incorporate the observations in Γ;

2. select one of the most important goal $g \in \mathcal{G}$ to be achieved;

3. find plan $a_1, \dots, a_n$ to achieve g;

4. execute a_1;

During the first step the agent does two things. First it observes exogenous actions $\{b_1, \dots, b_k\}$ which happen at the current moment of time, t_0. These observations are recorded by simply adding the statements $happened(b_i, t_0)$ $(1 \leq i \leq k)$ to the current set of observations, Γ_0. Now the agent's history is encoded in the new set of axioms, Γ' with the current moment of time, t'. Second, the agent observes the truth of

some fluent literals, O. If the agent observed all the exogenous actions which happen at time t_0 then the domain description $\langle \mathcal{A}, \Gamma' \cup \mathbf{O} \rangle$ where $\mathbf{O} = \{observed(l_i, t') : l_i \in O \text{ and } l_i \text{ is observed to be true at } t'\}$ will be consistent. Otherwise it may be inconsistent. In the latter case the new observations should be explained by assuming some occurrences of unobserved exogenous actions. This suggests that instead of adding observations to the domain description[1] the agent should first check their consistency and, if necessary, provide a suitable explanation. In precise terms, by *explanation* of unexplained observations O we mean a collection of statements
$H_0 = \{happened(a_i, t_k) : t_k < t_c,\ a_i \text{ is an elementary exogenous action}\}$
such that

$$\Gamma \cup H_0 \cup \mathbf{O} \text{ is consistent} \tag{1.3}$$

If O can be explained in several possible ways the agent should be supplied with some mechanism of selecting a most plausible one. (To simplify the discussion we assume that this is possible). Let us denote the set of axioms of the agent after the first step of the loop by Γ_1 and its current time by t_1. During the second stage the agent selects a goal $g \in G$ which will, at least for a while, determine its future actions. The goals in G may have priorities which depend on the current state of the domain and hence the agent may change its immediate goal during the next execution of the loop. Meanwhile however it goes to step three and looks for a plan to achieve g. This planning problem can be reduced to finding a sequence α of actions such that

$$\Gamma \models_{\mathcal{A}} holds\_after(g, \alpha) \tag{1.4}$$

After (1.4) is solved and a plan $\alpha = a_1, \ldots, a_n$ is found the agent proceeds to execute the first action, a_1, records $occurs\_at(a_1, t_1)$ in the domain history, and goes back to step one of the loop. Of course this architecture is only valid if computation performed during the execution of the body of the loop is fast enough to not allow the possibility of occurrence of exogenous actions, that may change the relevant characteristics of the domain, before a_1 is performed. In many situations the process can be made considerably faster by pre-computing solutions of (1.3) and (1.4) for certain goals. A solution for (1.4) for instance can be stored in the agent's memory as rules of the form, say,

$$\begin{array}{ll} if\ \Gamma \models_{\mathcal{A}} currently(l) & then\ execute(a_1) \\ & else\ \ execute(a_2) \end{array}$$

Agents whose ability to plan is limited to the use of such rules are called *reactive*. Otherwise, we characterize them as *deliberative*. Normally, agents should have both[2] deliberative and reactive components. Notice, however, that even reactive actions depend on the current knowledge of an agent and the choice of such actions may require a certain amount of reasoning. (Of course, checking if l holds in the current situation requires substantially less effort than planning or explaining observations.)

Example 2 Consider a domain with two locations, *home* and *airport*, and two objects, *money* and a *ticket*. An agent, Jack, is capable of driving from his current location to the other one and of getting money and ticket. The last action however is possible only if Jack has money and is located at the airport. Jack views the world in terms of two fluents, $jack\_at(L)$ and $has\_jack(O)$. The only exogenous action relevant to Jack is that of losing an object. Let us construct an action description $\mathcal{A}$ of Jack's world and use it to model Jack's behavior. Even though all the steps in the example can be easily proven the discussion will be informal. Later we will comment on the ways to automate all the reasoning steps.

Types :

$$\begin{array}{ll} location(home). & object(money). \\ location(airport). & object(ticket). \end{array}$$

Fluents :

$$\begin{array}{lcl} fluent(jack\_at(L)) & \leftarrow & location(L). \\ fluent(has\_jack(O)) & \leftarrow & object(O). \end{array}$$

Actions :

$$\begin{array}{lcl} agent\_action(drive\_to(L)) & \leftarrow & location(L). \\ agent\_action(get(O)) & \leftarrow & object(O). \\ exogenous\_action(lose(O)) & \leftarrow & object(O). \end{array}$$

Causal Laws :

$$\left.\begin{array}{l} impossible\_if(drive\_to(L), [jack\_at(L)]). \\ causes(drive\_to(L), jack\_at(L), []). \\ impossible\_if(get(ticket), [\neg has\_jack(money)]). \\ impossible\_if(get(ticket), [\neg jack\_at(airport)]). \\ causes(get(O), has\_jack(O), []). \\ causes(lose(O), \neg has\_jack(O), []). \\ caused(\neg jack\_at(L1), [jack\_at(L2), L1 \neq L2]). \\ impossible\_if([get(O), drive\_to(L)], []). \\ impossible\_if([get(O_1), lose(O_2)], []). \end{array}\right\} \mathcal{A}$$

Let us now assume that Jack's goal is to have a ticket. He starts with step one of the algorithm and records the following observations:

$$\left.\begin{array}{l}\textbf{Axioms}:\\ observed(has\_jack(money), 0)\\ observed(\neg has\_jack(ticket), 0)\\ observed(jack\_at(home), 0).\end{array}\right\}\Gamma_0$$

Since Jack has only one goal the selection is trivial. He goes to the planning stage, solves equation

$\Gamma_0 \models_{\mathcal{A}} holds\_after(has\_jack(ticket), X)$

and finds a plan, $\alpha_1 = [drive\_to(airport), get(ticket)]$. (We assume that he has enough time to find the shortest plan). Jack proceeds by executing the first action of this plan and recording this execution in the history of domain. The new collection of axioms is as follows:

$\Gamma_1 = \Gamma_0 \cup \{happened(drive\_to(airport), 0)\}$.

Now the current time is 1 and Jack continues his observations. Suppose he observes that his money is missing. This observation contradicts $\langle \mathcal{A}, \Gamma_1 \rangle$ and hence needs an explanation. The only explanation, obtained by solving equation

$\Gamma_1 \cup Y \cup \{observed(\neg has\_jack(money), 1)\}$ is consistent,

is $Y = \{happened(lose(money), 0)\}$, and hence

$\Gamma_2 = \Gamma_1 \cup \{happened(lose(money), 0), observed(\neg has\_jack(money), 1)\}$.

Assuming that Jack's goal is not changed, he proceeds to solve the equation $\Gamma_2 \models_{\mathcal{A}} holds\_after(has\_jack(ticket), X)$, finds a new plan, $\alpha_2 = [get(money), get(ticket)]$, gets money, goes back to step one and hopefully proceeds toward his goal without further interruptions.

3. REASONING ALGORITHMS

The logic programming community has developed a large number of reasoning algorithms which range from the SLDNF resolution implemented in traditional Prolog systems to the XSB resolution (Chen et al., 1995) implementing the well-founded semantics (Van Gelder et al., 1991), and comparatively recent techniques which can be used for computing answer sets of *A-Prolog* (Cholewiński et al., 1996; Niemela and Simons, 1997; Faber et al., 1999; Wang and Zaniolo, 2000). The latter form the basis for answer set programming advocated in (Niemela, 1999; Marek and Truszczyński, 1999). In this section we illustrate how these algorithms can be used to implement the above architecture and to perform the agent's reasoning tasks.

Planning

In this subsection we discuss model-theoretic planning using *A-Prolog*. In this approach, the answer sets of the program encode possible plans. We slightly differ from the previous work on answer set planning by emphasizing the use of "planning modules" to restrict the kind of plans that we are looking for. These modules can allow concurrency or force sequentiality of plans, specify preference between actions, incorporate reactive components into deliberative planning, etc. They can also contain a domain dependent control information.

We start with assuming that the corresponding domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ is *consistent and deterministic*, the set Γ of axioms is *complete*, i.e. uniquely determines the initial situation, and the goal g is a *set of fluent literals*. We also assume that time is represented by integers from $[0, N]$, that m is the maximum length of a plan the agent is willing to search for, and that $t_c + m \leq N$.

To find a solution of equation (1.4) we consider a program

$$\Pi_d = \Pi(N) \cup \mathcal{A} \cup \Gamma \cup R$$

where R consists of rules

$occurs\_at(A, T) \leftarrow happened(A, T).$

$holds\_at(L, T) \leftarrow observed(L, T).$

$agent\_action(a).$ (for any agent action a.)

Next we discuss how to construct *planning modules* of agents with different degrees of sophistication. We start with a simple planning module, $PM_0(m)$:

$$\left.\begin{array}{llcl}
(p1) & found & \leftarrow & t_c \leq T < t_c + m, \\
 & & & hold_at(g, T). \\
(p2) & & \leftarrow & not\ found. \\
(p3) & occurs_at(A, T) & \leftarrow & t_c \leq T < t_c + m, \\
 & & & not\ hold_at(g, T), \\
 & & & agent_action(A), \\
 & & & not\ \neg occurs_at(A, T).
\end{array}\right\} PM_0(m)$$

Let

$$P_0(m) = \Pi_d \cup PM_0(m)$$

The use of $P_0(m)$ for planning is based on the following observation:

Let $0 \leq k < m$. A sequence $\alpha = a_0, \ldots, a_k$ of actions is a plan for g iff there is an answer set S of $P_0(m)$ such that for any a_i from α,

$a_i = \{a : \ a$ is elementary and $occurs\_at(a, t_c + i) \in S\}$. (Due to the space limitations we cannot give a proof of this statement. The idea is however rather simple. One can notice that at any future moment t such that the goal is not yet satisfied the rule (p3) above and the rule (6) from program $\Pi(N)$ generates all possible sets of occurrences of the agent's actions at t. Other rules of Π_d reject those which do not lead to the plan.)

A plan of minimal length can be found by checking existence of answer sets of $P_0(1), P_0(2), \dots, P_0(m)$ or by varying the depth of a plan in some other way.

Similar techniques can be used to look for plans in *incomplete domain descriptions*. If, for instance, the values of fluents $f_1, \dots, f_k$ in the initial situation are unknown the planning can be done as follows:

1. Expand $P_0(m)$ by the rules:

$$\begin{array}{lcl} observed(f_i, 0) & \leftarrow & not\ observed(\neg f_i, 0). \\ observed(\neg f_i, 0) & \leftarrow & not\ observed(f_i, 0). \end{array}$$

2. Find an answer set S of the resulting program P'_0;

3. Let X be the set of propositions of the form $occurs\_at(a, t)$ from S and check if $P'_0 \cup X \models_{\mathcal{A}} g$. If the test succeeds then X defines a plan. Otherwise the process can be repeated, i.e. we can look for another answer set. If all answer sets are analyzed and a plan is still not found it does not exist. Alternatively, we can look for conditional plans, incorporate testing of values of fluents into planning or use a variety of other techniques.

A-Prolog can be used to implement planning modules which are different from $PM_0(m)$. For instance, the module $PM_1(m)$ consisting of the first two rules of $PM_0(m)$ and the rules

$$\left.\begin{array}{lcl} occurs\_at(A, T) & \leftarrow & t_c \leq T < t_c + m, \\ & & agent\_action(A), \\ & & not\ other\_occurs(A, T), \\ & & not\ hold\_at(g, T). \\ other\_occurs(A, T) & \leftarrow & t_c \leq T < t_c + m, \\ & & agent\_action(A_1), A \neq A_1, \\ & & occurs\_at(A_1, T). \end{array}\right\} PM_1$$

restricts the agent's attention to sequential plans. (These rules can be viewed as *A-Prolog* implementation of the choice operator from (Saccà and Zaniolo, 1997).) As before, finding a sequential plan for g of the length bounded by m can be reduced to computing answer sets of

$$P_1(m) = \Pi_d \cup PM_1(m).$$

To illustrate a slightly more sophisticated planning module let us go back to Example 2 and assume that Jack always follows a prudent policy of reporting the loss of his money to the police. This knowledge should be incorporated in Jack's planning module which can be done by adding to it the following rules:

$$\begin{array}{lcl} occurs\_at(report, t_c) & \leftarrow & T_0 < t_c \\ & & happened(lose(money), T_0), \\ & & not\ reported\_after(T_0). \\ reported\_after(T_0) & \leftarrow & T_0 < T < t_c, \\ & & happened(report, T). \end{array}$$

The resulting rules allow the agent to create plans which include a simple reactive component.

Now suppose that Jack can buy his ticket either using a credit card or paying cash, and that he prefers to pay cash if he has enough. To represent this information we introduce two new objects, *cash* and *card* and two static causal laws $caused(has\_jack(money), [has\_jack(card)])$ and $caused(has\_jack(money), [has\_jack(cash)])$. Jack's preference will be represented by adding the following rules to his planning module:

$$\begin{array}{lcl} occurs\_at(pay(cash), T) & \leftarrow & occurs\_at(get(O), T), \\ & & holds\_at(has\_jack(cash), T). \\ occurs\_at(pay(card), T) & \leftarrow & occurs\_at(get(O), T), \\ & & holds\_at(\neg has\_jack(cash), T). \end{array}$$

From now on Jack will plan to pay with cash if possible and use the credit card only as the last resort. Planning modules can also be used to incorporated heuristic information needed to speed up the planning process. Preliminary experiments show that this allows a very substantial increase in the efficiency of the planning process but a more systematic investigation is needed to make really precise and general claims. It also will be very interesting to do a serious study of the relationship between the planning methods described in this section and satisfiability planning (Kautz and Selman, 1992).

Explaining Observations

Now we demonstrate how answer set programming can be used for finding explanations, i.e. for solving equation 1.3. This can be achieved by finding answer sets of a program

$$E_1(m) = \Pi_d \cup EM_0(m)$$

where m determines the time interval in the past the agent is willing to consider in its search for explanations and $EM_0(m)$ is an explanation module consisting of rules

$$\left.\begin{array}{lll}
unexplained & \leftarrow & member(l, O), \\
 & & not\ holds_at(l, t_c). \\
 & \leftarrow & unexplained. \\
occurs_at(A, T) & \leftarrow & t_c - m \leq T < t_c, \\
 & & exogenous_action(A), \\
 & & not\ \neg occurs_at(A, T). \\
\neg occurs_at(A, T) & \leftarrow & t_c - m < T < t_c, \\
 & & exogenous_action(A), \\
 & & not\ occurs_at(A, T).
\end{array}\right\} EM_0(m)$$

Let S be an answer set of $E_1(m)$ and let $H_0(m)$ be the set of statements of the form $happened(a_i, t)$ such that $t_c - m \leq t < t_c$, a_i is an elementary exogenous action, $occurs\_at(a_i, t) \in S$ and $happened(a_i, t) \notin S$ (i.e. this occurrence of a_i has not been explicitly observed). Then $H_0(m)$ is an explanation of O. Moreover, any explanation of O of the depth m can be obtained in this way.

As in the case of the planning modules, the explanation module $EM_0(m)$ can be elaborated and tailored to a particular domain. For instance possible explanations for certain observations may be represented in the agent's knowledge base by a list of statements of the form

$$poss\_exp(l, a)$$

where l is a fluent literal and a is an exogenous action. (Often such a list can be extracted automatically from the corresponding action description). The following explanation module, $EM_1(m)$ uses this information to explain a single unexplained observation l by an occurrence of a single exogenous action in the interval $[t_0, t_c)$ where $t_0 = t_c - m$:

$$\left.\begin{array}{lll}
 & \leftarrow & not\ holds_at(l, t_c). \\
impossible(A, T) & \leftarrow & impossible_if(A, P), \\
 & & hold_at(P, T). \\
occurs_at(A, T) & \leftarrow & t_0 \leq T < t_c, \\
 & & poss_exp(l, A), \\
 & & not\ impossible(A, T), \\
 & & not\ other_occurs(A, T). \\
other_occurs(A, T) & \leftarrow & t_0 \leq T, T' < t_c, \\
 & & T' \neq T, \\
 & & occurs_at(A, T'), \\
 & & not\ happened(A, T'). \\
other_occurs(A, T) & \leftarrow & t_0 \leq T, T' < t_c, \\
 & & A \neq A', \\
 & & occurs_at(A', T'), \\
 & & not\ happened(A', T').
\end{array}\right\} EM_1(m)$$

Let S be an answer set of a program

$$E_2(m) = \Pi_d \cup EM_1(m)$$

and let $H_0(m)$ be the set of statements of the form $happened(a_i, t)$ such that $t_0 \leq t < t_c$, a_i is an elementary exogenous action, $occurs\_at(a_i, t) \in S$ and $happened(a_i, t) \notin S$. Then $H_0(m)$ is an explanation of O. Moreover, any explanation of O consisting of an occurrence of a single elementary exogenous action in the interval $[t_0, t_c]$ can be obtained in this way.

Checking the entailment

Now let us consider an agent's reactive component. Implementation of this component requires checking the condition

$$\Gamma \models_{\mathcal{A}} currently(l) \tag{1.5}$$

It can be shown that this can be reduced to checking the condition

$$\Pi_d \models currently(l) \tag{1.6}$$

As before, this task can be accomplished by the constraint satisfaction approach in answer set programming. For a very broad class of deterministic domain descriptions this condition can be also checked by using more traditional Prolog or XSB systems with a query $currently(l)$ and a simple modification of program Π_d as an input. The modification requires addition of information about types of variables in the bodies of some rules. It is needed to make sure that reasoning done by Prolog (XSB) on Π_d is sound and complete with respect to queries of the type $currently(l)$. (The corresponding proof follows the lines of the similar proof in (Baral et al., 1997) and demonstrates that for this input the Prolog interpreter always terminates, does not flounder, and does not require the occur check. This allows to reduce the proof of soundness and completeness of the interpreter to soundness and completeness of SLDNF resolution.) Ramifications of the choice of inference engine used for solving this and other problems are not entirely clear and require further investigation. We believe however that in this particular case diversity can be beneficial.

Finally, let us look at the situation in which the agent found a plan $a_1, \ldots, a_n$, executed action a_1, modified Γ to record new observations and discovered that its goal, g, is not changed. In this case it seems natural to start the planning with simply checking if

$$\Gamma \models_{\mathcal{A}} holds\_after(g, [a_n, \ldots, a_2]) \tag{1.7}$$

As before for complete, deterministic, and acyclic domain descriptions (Watson, 1999a) this can be done by running a Prolog interpreter on the query

$$holds\_after(g, [a_n, \ldots, a_2], T) \tag{1.8}$$

and program Π_c obtained by expanding Π_d by rules

$$\left.\begin{array}{lll}
impossible(A, S, T) & \leftarrow & impossible_if(A, P) \\
 & & hold_after(P, S, T). \\
possible(A, S, T) & \leftarrow & not\ impossible(A, S, T). \\
holds_after(L, [], T) & \leftarrow & holds_at(L, T). \\
holds_after(L, [A|S], T) & \leftarrow & possible(A, S, T), \\
 & & causes(A, L, P), \\
 & & hold_after(P, S, T). \\
holds_after(L, S, T) & \leftarrow & caused(L, P), \\
 & & hold_after(P, S, T). \\
holds_after(L, [A|S], T) & \leftarrow & possible(A, S, T), \\
 & & holds_after(L, S, T), \\
 & & not\ holds_after(\overline{L}, [A|S], T).
\end{array}\right\}$$

As before typing information should be added to the program to guarantee soundness and completeness of this method.

4. CONCLUSION

We have proposed a model of a simple intelligent agent acting in a changing environment. A characteristic feature of this model is its extensive use of a declarative language *A-Prolog*. The language is used for describing the agent's knowledge about the world and its own abilities as well as for precise mathematical characterization of the agent's tasks. We demonstrated how the agent's reasoning tasks can be formulated as questions about programs of *A-Prolog* and how these questions can be answered using various logic programming algorithms. A high expressiveness of the language allows one to represent rather complex domains not easily expressible in more traditional languages. The corresponding knowledge bases have a reasonably high degree of elaboration tolerance. Recent advances in logic programming theory allow one to prove correctness of these algorithms for a broad range of domains. New logic programming systems allow their implementation. Four such systems, CCALC, DeRes, DLV, and SMODELS, were demonstrated at the LBAI workshop. These systems were used to find plans from Example 2 as well as to solve much more complex tasks. Some experimental results aimed at testing the efficiency of the system are rather encouraging. The current rate of improvement of the systems performance and rapid advances in our understanding of methodology of programming in *A-Prolog* allow us to believe in the practicality of this approach. Some applications using XSB, DLV, Smodels and CCALC can be found in (Watson, 1999b; Soininen and Niemela, 1999; Erdem et al., 2000; McCain and Turner, 98; Cui et al., 1999) and the systems can be reached from http://www.cs.utexas.edu/users/tag/.

Acknowledgments

We would like to thank V. Lifschitz, E. Erdem, M. Nogueira, J. Minker and the referees for their useful comments.

Notes

1. To the best of our knowledge the idea that "observations should be added together with possible explanations" was first published in (Reiter, 1995).

2. More details on the combination of reactive and deliberative reasoning in the context of action based languages can be found in (Baral and Son, 1998). It also contains more detailed comparison to the reactive aspects of ConGolog (De Giacomo et al., 1997).

References

Baral, C. (1995). Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 2017–2023. Morgan Kaufmann.

Baral, C. and Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148.

Baral, C., Gelfond, M., and Provetti, A. (1995). Representing Actions I: Laws, Observations and Hypothesis. In *Proc. of AAAI 95 Spring Symposium on Extending Theories of Action: Formal theory and practical applications.*

Baral, C., Gelfond, M., and Provetti, A. (1997). Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming*, 31(1-3):201–243.

Baral, C. and Son, T. (1998). Relating theories of actions and reactive control. *Electronic transactions on Artificial Intelligence*, 2(3-4).

Chen, W., Swift, T., and Warren, D. (1995). Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–201.

Cholewiński, P., Marek, W., and Truszczyński, M. (1996). Default reasoning system DeRes. In Aiello, L., Doyle, J., and Shapiro, S., editors, *Proc. of KR 96*, pages 518–528. Morgan Kaufmann.

Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York.

Cui, B., Swift, T., and Warren, D. (1999). A case study in using preference logic grammars for knowledge representation. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proc. of LPNMR 99*, pages 206–220. Springer.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (1997). Complexity and expressive power of logic programming. In *Proc. of 12th annual IEEE conference on Computational Complexity*, pages 82–101.

De Giacomo, G., Lesperance, Y., and Levesque, H. (1997). Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI 97*, pages 1221–1226. Morgan Kaufmann.

DeGiacomo, G., Reiter, R., and Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proc. of KR 98*, pages 453–464. Morgan Kaufmann.

Denecker, M. and De Schreye, D. (1998). SLDNFA: an abductive procedure for normal abductive logic programs. *Journal of Logic Programming*, 34(2):111–167.

Erdem, E., Lifschitz, V., and Wong, M. (2000). Wire routing and satisfiability planning. In *Proc. CL-2000 (to appear)*.

Faber, W., Leone, N., and Pfeifer, G. (1999). Pushing goal derivation in DLP computations. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proc. of LPNMR 99*, pages 177–191. Springer.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. and Bowen, K., editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–387.

Gelfond, M. and Lifschitz, V. (1992). Representing actions in extended logic programs. In Apt, K., editor, *Joint International Conference and Symposium on Logic Programming.*, pages 559–573. MIT Press.

Guinchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. MIT Press.

Kakas, A., Kowalski, R., and Toni, F. (1998). The role of abduction in logic programming. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press.

Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proc. of ECAI-92*, pages 359–363.

Kowalski, R. (1995). Using metalogic to reconcile reactive with rational agents. In Apt, K. and Turini, F., editors, *Meta-logics and logic programming*, pages 227–242. MIT Press.

Kowalski, R. and Sadri, F. (1999). From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25:391–419.

Levesque, H., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84.

Lifschitz, V. (1996). Foundations of declarative logic programming. In Brewka, G., editor, *Principles of Knowledge Representation*, pages 69–128. CSLI Publications.

Lifschitz, V. (1997). Two components of an action language. *Annals of Math and AI*, 21(2-4):305–320.

Lin, F. (2000). From causal theories to successor state axioms and STRIPS like systems. In *Proc. of AAAI 2000 (to appear)*.

Lin, F. (95). Embracing causality in specifying the indirect effects of actions. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 1985–1993. Morgan Kaufmann.

Marek, W. and Truszczyński, M. (1989). Stable semantics for logic programs and default reasoning. In Lusk, E. and Overbeek, R., editors, *Proc. of the North American Conf. on Logic Programming*, pages 243–257. MIT Press.

Marek, W. and Truszczyński, M. (1993). *Nonmonotonic Logic: Context dependent reasoning*. Springer.

Marek, W. and Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V., Trucszczynski, M., and Warren, D., editors, *The Logic Programming Paradigm: a 25-Year perspective*, pages 375–398. Springer.

McCain, N. and Turner, H. (95). A causal theory of ramifications and qualifications. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 1978–1984. Morgan Kaufmann.

McCain, N. and Turner, H. (98). Satisfiability planning with causal theories. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proc. of KR 98*, pages 212–223. Morgan Kaufmann.

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1, 2):27–39,171–172.

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

Moore, R. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94.

Niemela, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–271.

Niemela, I. and Simons, P. (1997). Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429. Springer.

Pinto, J. (1999). Compiling Ramification Constraints into Effect Axioms. *Computational Intelligence*, 15(3):280–307.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132.

Reiter, R. (1995). On specifying database updates. *Journal of Logic Programming*, 25:25–91.

Saccà, D. and Zaniolo, C. (1997). Deterministic and non-deterministic stable models. *Journal of Logic and Computation*, 7(5):555–579.

Sandewall, E. (1998). Special issue. *Electronic Transactions on Artificial Intelligence*, 2(3-4):159–330. http://www.ep.liu.se/ej/etai/.

Shanahan, M. (1997). *Solving the frame problem: A mathematical investigation of the commonsense law of inertia*. MIT press.

Soininen, T. and Niemela, I. (1999). Developing a declarative rule language for applications in product configuration. In Gupta, G., editor, *Proc. of Practical Aspects of Declarative Languages '99*, volume 1551, pages 305–319. Springer.

Van Gelder, A., Ross, K., and Schlipf, J. (1991). The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650.

Wang, H. and Zaniolo, C. (2000). Nonmonotonic Reasoning in LDL++. In Minker, J., editor, *This volume*. Kluwer.

Watson, R. (1999a). *Action languages and domain modeling*. PhD thesis, University of Texas at EL Paso.

Watson, R. (1999b). An application of action theory to the space shuttle. In Gupta, G., editor, *Proc. of Practical Aspects of Declarative Languages '99*, volume 1551, pages 290–304. Springer.