Chapter 14

Logic Programming and Reasoning about Actions

[Chitta Baral & Michael Gelfond]

In this chapter we discuss how recent advances in logic programming can be used to *represent* and *reason* about actions and its impact on a dynamic world, which are necessary components of intelligent agents. Some of the specific issues that we consider are: the representation be tolerant to future updates and not repeatative, there may be relationships between objects in the world, exogenous actions may occur, we may have incomplete information about the world, and we may need to construct a plan for a given goal. In the process we introduce several action theories based on logic programs under the stable model semantics and discuss their gradual (and correct) transformation into executable programs.

14.1 Introduction

To perform nontrivial reasoning an intelligent agent situated in a changing domain needs the knowledge of causal laws that describe effects of actions that change the domain, and the ability to observe and record occurrences of these actions and the truth values of fluents at particular moments of time. One of the central problems of knowledge representation is the discovery of methods of representing this kind of information in a form allowing various types of reasoning about the dynamic world and at the same time tolerant of future updates. *The goal of this chapter is to demonstrate how recent advances in logic programming can be used to address this problem.* The early attempts on the use of logic programming for representing knowledge about dynamic domains can be found in [Eshghi, 1988a; Evans, 1989; Apt, 1990], among others. In these work the corresponding domains are described by general logic programs - collections of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{14.1}$$

where l_i's are atoms over some signature σ and *not* is a nonstandard logical connective called negation as failure. Due to the presence of this connective, the entailment relation between literals of σ and general logic programs is nonmonotonic, i.e., a literal l entailed by a program Π_1 is not necessarily entailed by a program Π_2 when $\Pi_1 \subset \Pi_2$. This property of the entailment makes logic programming a convenient tool for representing defaults [Reiter, 1980c], i.e., statements of the form "normally, (typically, as a rule) elements of a class a have property p." There are several defaults which seem to be frequently used in reasoning about dynamic domains. The most important one, known as the common-sense law of inertia [McCarthy, 1959; McCarthy, 1963; McCarthy and Hayes, 1969], says that normally things remain as they are. Any axiom describing the effect of an action on a state of the world represents an exception to this default. An agent reasoning about possible effects of his actions on the current state of the world uses these axioms to derive the changes that would occur in the current state after the execution of a particular action. The law of inertia is used to derive what does not change. The

problem of constructing a formal framework which would allow us to express and reason with the law of inertia is called the frame problem. The use of negation as failure leads to a simple solution of the frame problem for a broad class of dynamic domains. Unlike the initial attempts to solve the frame problem using circumscription [Shanahan, 1997], the logic programming solution avoids the existence of unintended models. *Moreover, some of the reasoning about dynamic domains can be performed by simply running the corresponding program under Prolog or one of its extensions, without developing any additional algorithms for nonmonotonic reasoning.*

In the last ten years we have witnessed several developments in the theory of logic programming which substantially improved its applicability to the theory of actions. Extensions of "classical" logic programming such as the use of abduction [Kakas *et al.*, 1993], disjunction [Lobo *et al.*, 1992; Gelfond and Lifschitz, 1991], and programs with two negation operators [Gelfond and Lifschitz, 1991] allowed the removal of the closed world assumption [Reiter, 1978] implicit in its initial framework. As a result logic programming became suitable for representing incomplete information [Gelfond, 1994; Denecker and De Schreye, 1993; Dung, 1993]. Discovery of declarative semantics of logic programs independent of the inference mechanism of Prolog allowed us to better understand the nature and mathematical properties of new logical connectives. This led to advances in development and implementations of inference mechanisms [Niemelä and Simons, 1997; Chen *et al.*, 1995; Eiter *et al.*, 2000a; Denecker and De Schreye, 1997] for enhanced logic programming languages. These and other advances facilitated a systematic development of formal theories of actions based on logic programming. There is a considerable body of work devoted to this subject. It can be roughly classified by *the ontology of actions and time*, by *the type of semantics of logic programming*, and by the type of the targeted interpreter, used in a particular work.

Ontology based differences can be traced to differences between two basic calculi proposed for formalization of actions: the Situation Calculus [McCarthy and Hayes, 1969; Reiter, 2001] and the Event Calculus [Kowalski and Sergot, 1986]. Even though originally the Situation Calculus was formulated in First-Order Logic, its logic programming counterparts appeared shortly after its introduction. The Event Calculus was originally formulated using a logic programming language. The relationship between the two formalisms is by now well understood [Van Belleghem *et al.*, 1995; Provetti, 1996a; Kowalski and Sadri, 1997]. There is also some work on combining the most important features of both approaches [Baral *et al.*, 1997; Kakas and Miller, 1997].

The differences in semantics are related to slightly different views on the utility of various patterns of default reasoning. Open logic programs [Denecker and De Schreye, 1993] seem to put particular emphasis on abduction. Logic programs under well-founded semantics [Van Gelder *et al.*, 1991; Alferes and Pereira, 1996; Brass *et al.*, 1998] are based on cautious approach to applying defaults which leads to the intended model of a program in which truth values of some literals may be undefined. Stable model semantics [Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1990] allows a form of reasoning by cases and has an epistemic flavor. Declaratively, logic programs (without disjunctions in the head) under stable model semantics can be viewed as subclasses of Reiter's default theories. The situation is not however as messy as it may appear to a reader not familiar with all these subtleties and fortunately, the semantics coincide for very large classes of programs. When it is not the case the relationships between different formalisms are rather well understood. For instance, for any program Π consistent from the standpoint of stable model semantics and any literal l, if l is a consequence of Π under the well-founded semantics then it is a consequence of Π under the stable model semantics.

Until recently, most formulations of reasoning about actions in logic programming were based on the underlying idea of using a Prolog like interpreter where queries, possibly containing variables, are asked with respect to a program and the answer substitution of the variables returned by the interpreter contained meaningful information such as a plan. Recently, the development of systems that generate stable models of logic programs [Niemelä and Simons, 1997; Eiter *et al.*, 2000a; Citrigno *et al.*, 1997] has led to formulations where meaningful information, such as a plan [Subrahmanian and Zaniolo, 1995; Dimopoulos *et al.*, 1997; Lifschitz, 1999; Son *et al.*, 2001] or a diagnosis [Gelfond *et al.*, 2001], are encoded by the stable models themselves.

In this chapter we will not attempt to discuss all these differences and advantages and disadvantages of different approaches. Instead we introduce several action theories based on logic programs under the stable model semantics and its generalizations. The emphasis will be on the methodology of development of these theories and on their gradual transformation into executable programs. Most of the results in this chapter are from previously published work. The only new and previously unpublished results in this chapter are Proposition 14.9.2 and Proposition 14.10.1. The rest of the paper is organized as follows. In Section 14.2, we give a brief overview of the stable model semantics of logic programs and notions such as 'splitting' and 'signing'. In Section 14.3 we give the basic notions of action languages and then progressively introduce action languages $\mathcal{A}_0$ (Section 14.4), and $\mathcal{A}_1$ (Section 14.10), query languages $\mathcal{Q}_0$(Section 14.5), and $\mathcal{Q}_1$ (Section 14.7) and algorithms to answer queries in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$ (Section 14.6 and Section 14.9), $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_1)$ (Section 14.8), and $\mathcal{L}(\mathcal{A}_1, \mathcal{Q}_1)$ (Section 14.11). Finally in Section 14.12, we show how logic programming can be used for planning in a model enumeration style.

14.2 Logic Programming

In this section we review necessary definitions and results from the theory of declarative logic programming. In addition to the negation as failure operator *not* [Clark, 1978a] of "classical" logic programming languages we consider two other connectives: classical (strong, explicit) negation ($\neg$) of [Gelfond and Lifschitz, 1990] and epistemic disjunction *or* of [Gelfond and Lifschitz, 1991]. Both connectives are needed to allow representation of various forms of incomplete information. There is no complete agreement on the nature and semantics of these connectives and their interrelation with negation as failure. Several different proposals were discussed in the literature. (See, for instance, Minker et al. [Lobo *et al.*, 1992], Pereira et al. [Pereira *et al.*, 1990], Dix [Dix, 1991], Przymusinski [Przymusinski, 1990], and Gelfond and Lifschitz [Gelfond and Lifschitz, 1991]). We will follow the answer set semantics* of [Gelfond and Lifschitz, 1991]. Applicability of this approach to representation of incomplete information is discussed in [Baral and Gelfond, 1994; Gelfond, 1994].

A disjunctive logic program (DLP) is a collection of rules of the form

$$l_0 \textbf{ or } \ldots \textbf{ or } l_k \leftarrow l_{k+1}, \ldots l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{14.2}$$

where each l_i is a literal, i.e. an atom possibly preceded by $\neg$, and *not* is the negation as failure. Expression on the left hand (right hand) side of $\leftarrow$ is called the *head* (the *body*) of the rule. Both, the head and the body of (14.2) can be empty. DLPs whose rules have $k = 0$, and whose l_i's are positive literals are referred to as *general logic programs*. When all the rules in a DLP have $k = 0$ then it is referred to as an *extended logic program* [Gelfond and Lifschitz, 1990; Pearce and Wagner, 1989].

Intuitively the rule 14.2 can be read as: if $l_{k+1}, \ldots, l_m$ are believed and it is not true that $l_{m+1}, \ldots, l_n$ are believed then at least one of $\{l_0, \ldots, l_k\}$ is believed. For a rule r of the form (14.2) the sets $\{l_0, \ldots, l_k\}$, $\{l_{k+1}, \ldots, l_m\}$ and $\{l_{m+1}, \ldots, l_n\}$ are referred to as $head(r)$, $pos(r)$ and $neg(r)$ respectively; $lit(r)$ stands for $head(r) \cup pos(r) \cup neg(r)$. For any DLP Π, $head(\Pi) = \bigcup_{r \in \Pi} head(r)$. For a set of predicates S, $Lit(S)$ denotes the set of literals with predicates from S. For a DLP Π, $Lit(\Pi)$ denotes the set of literals with predicates from the language of Π. When it is clear from the context, we write Lit instead of $Lit(\Pi)$. For sets of literals X and Y, we say Y is *complete* in X if for every literal $l \in X$, at least one of the complementary literals $l, \overline{l}$ belongs to Y.

A program determines a collection of *answer sets* – sets of ground literals representing possible beliefs of the program.

Definition 14.2.1 ([Gelfond and Lifschitz, 1991]). Let Π be a disjunctive logic program without variables. For any set S of literals, let Π^S be the logic program obtained from Π by deleting

(i) each rule that has a formula *not* l in its body with $l \in S$, and

*Recently, the language of logic programming with answer set semantics is referred to as A-Prolog or AnsProlog meaning 'answer set programming in logic'.

(ii) all formulas of the form *not* l in the bodies of the remaining rules. □

Definition 14.2.2. An *answer set* of a disjunctive logic program Π not containing *not* is a minimal (in a sense of set-theoretic inclusion) subset S of Lit such that

(i) for any rule l_0 **or** $\ldots$ **or** $l_k \leftarrow l_{k+1} \ldots l_m$ from Π, if $l_{k+1}, \ldots, l_m \in S$, then for some i, $0 \leq i \leq k$, $l_i \in S$;

(ii) if S contains a pair of complementary literals, then $S = Lit$.

A set S of literals is an answer set of an arbitrary disjunctive logic program Π if
S is an answer set of Π^S. □

A program* is consistent if it has an answer set not containing contradictory literals. As was shown in [Gelfond, 1994] if a program is consistent then all of its answer sets are consistent. A ground literal l is said to be *entailed* by a DLP Π, written as $\Pi \models l$, if it belongs to all of its answer sets.

In our further discussion we will need the following proposition about DLPs:

Proposition 14.2.1 ([Baral and Gelfond, 1994]). For any answer set S of a disjunctive logic program Π:

(a) For any ground instance of a rule of the type (14.2) from Π, if
$\{l_{k+1} \ldots l_m\} \subseteq S$ and
$\{l_{m+1} \ldots l_n\} \cap S = \emptyset$
then there exists an i, $0 \leq i \leq k$ such that $l_i \in S$.

(b) If S is a consistent answer set of Π and $l_i \in S$ for some $0 \leq i \leq k$ then there exists a ground instance of a rule from Π such that
$\{l_{k+1} \ldots l_m\} \subseteq S$, and
$\{l_{m+1} \ldots l_n\} \cap S = \emptyset$, and
$\{l_0 \ldots l_k\} \cap S = \{l_i\}$. □

We now review the definitions of "splitting" and "signing" which we use to analyze properties of the programs obtained by translating a domain description.

Definition 14.2.3 ([Turner, 1994]). Let Π be a DLP such that no rule in it has empty heads. Let S be a set of literals in the language of Π such that no literals in $head(\Pi)$ appears complemented in $head(\Pi)$. Let $\overline{S}$ denote $Lit \setminus S$. S is said to be a $signing$ for Π if each rule $r \in \Pi$ satisfies the following two conditions:

(i) $head(r) \cup pos(r) \subset S$ and $neg(r) \subset \overline{S}$, or
$head(r) \cup pos(r) \subset \overline{S}$ and $neg(r) \subset S$

(ii) If $head(r) \subset S$, then $head(r)$ is a singleton.

If a program has a signing, we say that it is signed. □

Definition 14.2.4 ([Turner, 1994]). Let Π be a program. If S is a signing for Π, then

$h_S(\Pi) = \{r \in \Pi \ : \ head(r) \subset S\}$,

$h_{\overline{S}}(\Pi) = \{r \in \Pi \ : \ head(r) \subset \overline{S}\}$. □

Proposition 14.2.2. Based on the restricted monotonicity theorem in [Turner, 1994]
Let Π_1 and Π_2 be programs in the same language, both with signing S. If $h_{\overline{S}}(\Pi_1) \subseteq h_{\overline{S}}(\Pi_2)$ and $h_S(\Pi_2) \subseteq h_S(\Pi_1)$, then

if $\Pi_1 \models l$ and $l \in \overline{S}$ then $\Pi_2 \models l$. □

*Henceforth by "program" we mean a disjunctive logic program.

Definition 14.2.5. *(Splitting set) [Lifschitz and Turner, 1994]*
A *splitting set* for a program Π is any set U of literals such that for every rule $r \in \Pi$, if $head(r) \cap U \neq \emptyset$ then $lit(r) \subset U$. If U is a splitting set for Π, we also say that U splits P. The set of rules $r \in \Pi$ such that $lit(r) \subset U$ is called the *bottom* of Π relative to the splitting set U and denoted by $b_U(\Pi)$. The subprogram $\Pi \setminus b_U(\Pi)$ is called *the top of* Π relative to U. □

Definition 14.2.6. *(Partial evaluation) [Lifschitz and Turner, 1994]*
The partial evaluation of a program Π with splitting set U w.r.t. a set of literals X is the program $e_U(\Pi, X)$ defined as follows. For each rule $r \in \Pi$ such that:

$$(pos(r) \cap U) \subset X \quad \wedge \quad (neg(r) \cap U) \cap X = \emptyset$$

put in $e_U(\Pi, X)$ all the rules $r^{'}$ that satisfy the following property:

$$head(r') = head(r),\ pos(r') = pos(r) \setminus U,\ neg(r') = neg(r) \setminus U$$

□

Definition 14.2.7. *(Solution) [Lifschitz and Turner, 1994]*
Let U be a splitting set for a program Π. A solution to Π w.r.t. U is a pair $\langle X, Y \rangle$ of literals such that:

- X is an answer set for $b_U(\Pi)$;
- Y is an answer set for $e_U(\Pi \setminus b_U(\Pi), X)$;
- $X \cup Y$ is consistent.

□

Lemma 14.2.1. *(Splitting Lemma) [Lifschitz and Turner, 1994]*
Let U be a splitting set for a program Π. A set A of literals is a consistent answer set for Π if and only if $A = X \cup Y$ for some solution $\langle X, Y \rangle$ to Π w.r.t. U. □

The concept of a well-moded program due to Dembinski and Maluszynski [Dembinski and Maluszynski, 1985] has proven to be useful for establishing various properties of logic programs. We will be using it in this chapter and hence to be self-complete we will define it here. We first need the following terminology:

By a *mode* for an n-ary predicate symbol p we mean a function d_p from $\{1, \ldots, n\}$ to the set $\{+, -\}$. If $d_p(i)$ = '+' the i is called an *input* position of p and if $d_p(i)$ = '-' the i is called an *output* position of p. We write d_p in the form $p(d_p(1), \ldots, d_p(n))$. Intuitively, queries formed by predicate p will be expected to have input positions occupied by ground terms. To simplify the notation, when writing an atom as $p(u, v)$, we assume that u is the sequence of terms filling in the input positions of p and that v is the sequence of terms filling in the output positions. By $l(u, v)$ we denote expressions of the form $p(u, v)$ or *not* $p(u, v)$; $var(s)$ denotes the set of all variables occurring in s. Assignment of modes to the predicate symbols of a program Π is called *input-output specification*.

A rule $p_0(t_0, s_{m+1}) \leftarrow l_1(s_1, t_1), \ldots, l_m(s_m, t_m)$ is called *well-moded* w.r.t. its input output specification if for $i \in [1, m+1]$, $var(s_i) \subseteq \bigcup_{j=0}^{i-1} var(t_j)$. In other words, a rule is well-moded if

i) every variable occurring in an input position of a body goal occurs either in an input position of the head or in an output position of an earlier body goal;

ii) every variable occurring in an output position of the head occurs in an input position of the head, or in an output position of a body goal.

A program is called well-moded w.r.t. its input-output specification if all its rules are.

In our analysis we will also be needing the following notion of acyclic programs [Apt, 1990].

Definition 14.2.8 ([Apt, 1990]). A general logic program Π is *acyclic* if there exists a mapping $|\ |$ from the Herbrand base of Π to the the set of natural numbers such that for every $A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ in the ground version of Π, and for every $1 \leq i \leq n$: $|A_0| > |A_i|$. □

14.2.1 Abductive logic programs

An alternative approach for reasoning with incomplete information is the formulation of abductive logic programs [Kakas and Mancarella, 1990a; Denecker and De Schreye, 1993; Baral and Gelfond, 1994], where predicates about which incompleteness is allowed is referred to as the *abducible* predicates or *open predicates*. An abductive logic program is a triple $\langle \Pi, A, O \rangle$, where A is the set of open predicates, Π is a general logic program with atoms of non-open predicates in its heads and O is a set of first order formulas. O is used to express *observations* and *constraints* in an abductive logic program. Abductive logic programs are characterized as follows:

Definition 14.2.9. Let $\langle \Pi, A, O \rangle$ be an abductive logic program. A set M of ground atoms is a *generalized stable model* of $\langle \Pi, A, O \rangle$ if there is a set of ground atoms Δ made up of predicates in A, such that M is a stable model of $\Pi \cup \Delta$ and M satisfies O.

For an atom f, we say $\langle \Pi, A, O \rangle \models_{abd} f$, if f belongs to all generalized stable models of $\langle \Pi, A, O \rangle$. For a negative literal $\neg f$, we say $\langle \Pi, A, O \rangle \models_{abd} \neg f$, if f does not belong to any of the generalized stable models of $\langle \Pi, A, O \rangle$. □

14.3 Action Languages: basic notions

Our description of dynamic domains will be based on the formalism of action languages. Such languages, first introduced in [Gelfond and Lifschitz, 1993], can be thought of as formal models of the part of the natural language that are used for describing the behavior of dynamic domains. An *action language* can be represented as the sum of two distinct parts: an "*action description language*", and an "*action query language*".

A set of propositions in an *action description language* describes the effects of actions on states. Mathematically, it defines a transition system with nodes corresponding to possible states and arcs labeled by actions from the given domain. An arc (σ_1, a, σ_2) indicates that execution of an action a in state σ_1 may result in the domain moving to the state σ_2. By a *path or history* of a transition system T we mean a sequence $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ such that for any $1 \leq i < n$, $(\sigma_i, a_{i+1}, \sigma_{i+1})$ is an arc of T. σ_0 and σ_n are called initial and final states of the path (or history) respectively.

An *action query language* serves for expressing properties of paths in a given transition system. *The syntax* of such a language is defined by two classes of syntactic expressions: *axioms* and *queries*. *The semantics* of an action query language is defined by specifying, for every transition system T, every set Γ of axioms, and every query Q, whether Q is a consequence of Γ in T.

In the next three sections we define a simple action language $\mathcal{L}_0$ which can be viewed as the sum of an action description language $\mathcal{A}_0$ and a query description language $\mathcal{Q}_0$. We assume a fixed signature Σ_0 which consists of two disjoint and nonempty sets of symbols, a set $\mathbf{F}$ of fluents, and a set $\mathbf{A}$ of actions. Signatures of this kind will be called *action signatures*. By fluent literals we mean fluents and their negations. Negation of $f \in \mathbf{F}$ will be denoted by $\neg f$. A set S of fluent literals is called complete (w.r.t. $\mathbf{F}$) if for any $f \in \mathbf{F}$ we have $f \in S$ or $\neg f \in S$. A *state* is represented by a complete and consistent set of fluent literals of Σ_0. A fluent literal l is said to be true or said to hold in a state s, if $l \in s$. A set S of fluent literals is said to be true or said to hold in a state s, if all element of S hold in s.

14.4 Action description language $\mathcal{A}_0$

Consider a fixed action signature Σ_0. The syntax of $\mathcal{A}_0$ is characterized by the following definition.

Definition 14.4.1. In the language $\mathcal{A}_0$,

1. A fluent literal is an expression of the form f or $\neg f$ where f is a fluent name,

2. Propositions (called *causal laws*) are expressions of the form

$$impossible\_if(a, [l_1, \ldots, l_n]) \tag{14.3}$$

$$causes(a, l_0, [l_1, \ldots, l_n]) \tag{14.4}$$

where a is an action name and l's are fluent literals. The former are called *executability conditions*, and the latter are called *dynamic causal laws*. Intuitively, the proposition (14.3) means that the action a is impossible to execute in a state s if the set of fluent literals $\{l_1, \ldots, l_n\}$ hold in s. Similarly, the proposition (14.4) means that if an action a is executed in a state s such that the set of fluent literals $\{l_1, \ldots, l_n\}$ hold in s then the fluent literal l_0 will hold in the subsequent state.

3. An *action description* is a set of causal laws. □

Given an action description $\mathcal{D}$, the semantics of $\mathcal{A}_0$ defines the transition system that is "described" by $\mathcal{D}$. More precisely

Definition 14.4.2. The transition system $T = \langle \mathcal{S}, \mathcal{R} \rangle$ *described* by $\mathcal{D}$ is defined as follows:

1. $\mathcal{S}$ is the collection of all complete and consistent sets of fluent literals of Σ_0,

2. $\mathcal{R}$ is the set of all triples (σ, a, σ'), where $\sigma, \sigma' \in \mathcal{S}$, such that $\mathcal{D}$ does not contain a proposition of the form $impossible\_if(a, [l_1, \ldots, l_n])$ with $[l_1, \ldots, l_n] \subseteq \sigma$ and

$$E(a, \sigma) \subseteq \sigma' \subseteq E(a, \sigma) \cup \sigma \tag{14.5}$$

where $E(a, \sigma)$ stands for the set of all fluent literals l_0 for which there is a dynamic causal law $causes(a, l_0, [l_1, \ldots, l_n])$ in $\mathcal{D}$ such that $[l_1, \ldots, l_n] \subseteq \sigma$.

3. For any $\sigma \in \mathcal{S}$, if there does not exist a proposition of the form $impossible\_if(a, [l_1, \ldots, l_n])$ with $[l_1, \ldots, l_n] \subseteq \sigma$ then there exists a σ' such that $(\sigma, a, \sigma') \in \mathcal{R}$. (Note that this (σ, a, σ') must satisfy condition (2).)

We say that an action a is *prohibited* in a state σ if there is no σ' such that $(\sigma, a, \sigma') \in \mathcal{R}$. The transition system T described by $\mathcal{D}$ is called the *causal model* of $\mathcal{D}$. A domain description with no causal model is called *inconsistent*. □

Intuitively, (14.5) together with the requirement that σ' be complete and consistent says that the immediate effects of action a in state σ must be in σ', and in addition (accounting for inertia) all other fluents in σ must remain unchanged in σ'. Moreover, for $\mathcal{A}_0$, σ' satisfying (14.5), if exists, is unique. An example of a domain description that is inconsistent is $\{causes(a, f, [\,]), causes(a, \neg f, [\,])\}$. That is because for an arbitrary σ there does not exist a σ' such that (σ, a, σ') satisfies the condition (2). Thus condition (3) is violated.

Example 14.4.1. Let us consider a collection of vehicles which can move between different locations. The corresponding signature Σ_0 consists of two sets of object constants, $v_1, \ldots, v_n$ and $l_1, \ldots, l_m$; the set of fluents of the form $at(v, l)$ which stands for "the vehicle v is located at location l", and a set of actions $move(v, l_1, l_2)$ where l_1 and l_2 are different locations. The effects of these actions can be defined by the following set $\mathcal{D}_0$ of causal laws:

$$\mathcal{D}_0 \left\{ \begin{array}{l} causes(move(v, l_1, l_2), at(v, l_2), [\,]). \\ causes(move(v, l_1, l_2), \neg at(v, l_3), [\,]). \\ impossible\_if(move(v, l_1, l_2), [\neg at(v, l_1)]). \\ \text{where } v\text{'s are vehicles and } l_1, l_2, \text{ and } l_3 \text{ are locations and } l_3 \neq l_2. \end{array} \right.$$

To actually specify these causal laws to a computer program we will use a DLP. We assume the existence of complete lists of vehicles and locations given by collection of atoms $vehicle(v_1), \ldots$ and $location(l_1), \ldots$, and define the causal laws by rules with variables:

$$\Pi_{\mathcal{D}_0} \begin{cases} causes(move(V, L_1, L_2), at(V, L_2), [\,]) :- & vehicle(V), \\ & location(L_1), \\ & location(L_2). \\ causes(move(V, L_1, L_2), \neg at(V, L_3), [\,]) :- & vehicle(V) \\ & location(L_1), \\ & location(L_2), \\ & location(L_3), \\ & L_3 \neq L_2. \\ impossible\_if(move(V, L_1, L_2), [\neg at(V, L_1)]) :- & vehicle(V) \\ & location(L_1), \\ & location(L_2). \end{cases}$$

We say a causal law $c \in D_0$ iff it is entailed by the above program.

The Figure 14.1 shows the transition system T_0 described by $\mathcal{D}_0$. (For simplicity we assumed that the signature of $\mathcal{D}_0$ contains names for one vehicle and two locations.)

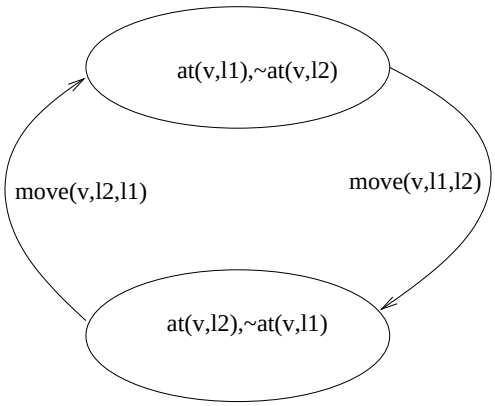

Figure 14.1: Transition system T_0

It is worth noticing that according to our description there are states in which a vehicle can occupy more than one location. Similarly, one location may contain more than one vehicle. Later we show how these possibilities – if necessary – can be eliminated. □

14.5 Query description language $\mathcal{Q}_0$

The query language $\mathcal{Q}_0$ over an action signature Σ_0 consists of two types of expressions: axioms and queries. Axioms of $\mathcal{Q}_0$ are of the form

$$initially(l) \tag{14.6}$$

where l is a fluent literal. A collection of axioms describes the set of fluents which are true in (the state corresponding to) the initial situation*. A set of axioms of $\mathcal{Q}_0$ is said to be *initial state complete* if for all fluents f either $initially(f)$ or $initially(\neg f)$ is in the set.

A query of $\mathcal{Q}_0$ is a statement of the form

$$holds\_after(l, \alpha) \tag{14.7}$$

where l is a fluent literal and α is a sequence of actions. The statement says that α is executable in the initial situation and, if it were executed, then the fluent literal l would be true afterwards. To give the semantics of $\mathcal{Q}_0$ we need the following definition.

Definition 14.5.1. Let T be a transition system over action signature Σ_0. We say that

(i) a history $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ *satisfies* an axiom $initially(l)$ if $l \in \sigma_0$,

(ii) a query $Q = holds\_after(l, [a_n, \ldots, a_1])$ is a *consequence* of a set Γ of axioms with respect to T if, for every history H of T of the form $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ that satisfies all axioms in Γ, $l \in \sigma_n$. In this case we say Q holds in H.

Let $\mathcal{D}$ be an action description and T be the transition system defined by $\mathcal{D}$. We say that a query Q is a *consequence* of a set Γ of axioms in $\mathcal{D}$ (symbolically, $\Gamma \models_{\mathcal{D}} Q$) if Q is a consequence of Γ with respect to T. □

To illustrate the definition let us consider the following example.

Example 14.5.1. Let $\mathcal{D}_0$ be the action description from Example 14.4.1 and consider the set Γ_0 of axioms of the form

$$\Gamma_0 \left\{ \begin{array}{l} (a)\ initially(at(v_1, l_1)), initially(at(v_2, l_2)), \ldots \\ (b)\ initially(\neg at(v_i, l_j)), \text{where } i \neq j. \end{array} \right.$$

Obviously, Γ_0 gives a *complete* description of the initial situation. I.e., there is only one state in T_0 which satisfies all the axioms from Γ_0. It can then be shown that

(i) $\Gamma_0 \models_{\mathcal{D}_0} holds\_after(at(v_1, l_3), [move(v_1, l_1, l_3)])$,

(ii) $\Gamma_0 \models_{\mathcal{D}_0} holds\_after(\neg at(v_1, l_i), [move(v_1, l_1, l_3)])$, for any location l_i different from l_3,

(iii) $\Gamma_0 \models_{\mathcal{D}_0} holds\_after(at(v_2, l_2), [move(v_1, l_1, l_3)])$, and

(iv) $\Gamma_0 \models_{\mathcal{D}_0} holds\_after(\neg at(v_2, l_i), [move(v_1, l_1, l_3)])$, for any location l_i different from l_2.

Similar to Example 14.4.1 the axioms of Γ_0 can be more concisely defined by replacing facts of the form (b) by the DLP below:

$$\Pi_{\Gamma_0} \left\{ \begin{array}{lll} & initially(at(v_1, l_1)). & \\ & initially(at(v_2, l_2)). & \\ & \ldots & \\ initially(\neg at(V, L)) :- & vehicle(V), & \\ & location(L), & \\ & \textit{not } initially(at(V, L)). & \end{array} \right.$$

□

In the next section we give additional DLP rules that are needed to compute $\models_{\mathcal{D}_0}$ with respect to Γ_0.

*By situation we mean an executable sequence of actions. The initial situation corresponds to the empty sequence.

14.6 Answering queries in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$

In this section we address the question of computing the consequences of axioms of $\mathcal{Q}_0$. We limit ourselves to sets of axioms which are initial state complete. (We will lift this restriction in Section 14.9.) The consequences of an action description $\mathcal{D}$ and a set Γ of axioms will be computed by a general logic program Π_{00} together with $\mathcal{D}$ and Γ as a set of facts. Π_{00} consists of the following rules:

1. Executability of Actions:

$$\Pi^1_{00} \left\{ \begin{array}{lll} impossible([A|S]) & :- & \\ & & impossible(S). \\ impossible([A|S]) & :- & \\ & & impossible\_if(A,P), \\ & & holds\_after\_list(P,S). \\ executable(S) & :- & \\ & & not\ impossible(S). \end{array} \right.$$

   The atom $holds\_after\_list(p,s)$ says that the sequence s of actions is executable in the initial situation and, if it were to be executed, then all fluent literals from the list p would necessarily be true afterwards. The statement $impossible(s)$ ($executable(s)$) says that the sequence s of action can not (can) be executed in the initial situation. Intuitively, the use of negation as failure in the third rule is justified by the completeness of information about impossibility of actions and about truth and falsity of fluents. Formal justification for these and other axioms will be provided by Proposition 14.6.1 below.

2. The Effect Axioms:

$$\Pi^2_{00} \left\{ \begin{array}{lll} holds\_after(L,[\,]) & :- & \\ & & initially(L). \\ holds\_after(L,[A|S]) & :- & \\ & & executable([A|S]), \\ & & causes(A,L,C), \\ & & holds\_after\_list(C,S). \end{array} \right.$$

   These axioms define the effects of actions on a state (corresponding to a situation) based on causal laws and on truth and falsity of fluents.

3. The List Axioms:

$$\Pi^3_{00} \left\{ \begin{array}{lll} holds\_after\_list([],\_). & & \\ holds\_after\_list([L|Rest],S) & :- & \\ & & holds\_after(L,S), \\ & & holds\_after\_list(Rest,S). \end{array} \right.$$

   The axioms in this group define the auxiliary relation $holds\_after\_list(p,\alpha)$.

4. The Inertia Axiom:

$$\Pi^4_{00} \left\{ \begin{array}{lll} holds\_after(L,[A|S]) & :- & \\ & & executable([A|S]), \\ & & holds\_after(L,S), \\ & & not\ ab(L,A,S). \end{array} \right.$$

This is the inertia axiom mentioned in the introduction. It has a form of default which says that *normally*, things remain as they are. The atom $ab(l, a_n, [a_{n-1}, \ldots, a_1])$ says that the inertia axiom shall not be applied to establish the truth value of l after the execution of $[a_1, \ldots, a_{n-1}, a_n]$. This is a common way of representing defaults in a logic programming framework, where we represent 'normally ps are qs', but r's are an exception to this rule, by writing:

$q(X) \leftarrow p(X), not\ ab(X)$
$ab(X) \leftarrow r(X)$

5. Cancellation Axioms

$$\Pi_{00}^5 \left\{ \begin{array}{lll} ab(L, A, S) & :- & \\ & & contrary(L, NL), \\ & & causes(A, NL, C), \\ & & holds\_after\_list(C, S) \end{array} \right.$$

The above axiom stops the application of the inertia axiom – that establishes the truth of a fluent literal l in situation $[a|s]$, if there are causal laws which cause l to become false in the situation $[a|s]$.

6. Auxiliary

$$\Pi_{00}^6 \left\{ \begin{array}{lll} contrary(neg(F), F) & :- & \\ & & fluent(F). \\ contrary(F, neg(F)) & :- & \\ & & fluent(F). \end{array} \right.$$

The following proposition gives conditions for soundness and completeness of $\Pi_{00} \cup \mathcal{D} \cup \Gamma$ with respect to the entailment $\models_{\mathcal{D}}$ from a set of complete and consistent axioms Γ. Given a consistent (but possibly incomplete) set of axioms Γ, by $c(\Gamma)$ we denote the set $\{\Gamma' :$ such that $\Gamma \subseteq \Gamma'$ and Γ' is complete $\}$. In the following proposition and through the rest of this paper, for a logic program Π, we will often denote the set $\{Q : \Pi \cup \Gamma \cup \mathcal{D} \models Q\}$ by $\Pi(\Gamma \cup \mathcal{D})$.

Proposition 14.6.1. For any consistent action description $\mathcal{D}$ and an initial state complete set of axioms Γ, $\Gamma \models_{\mathcal{D}} holds\_after(l, s)$ iff $holds\_after(l, s) \in \Pi_{00}(\mathcal{D} \cup \Gamma)$. □

sketch. The proof follows from the following two lemmas. In each of these two lemmas $\mathcal{D}$ is a consistent action description and Γ is an initial state complete set of axioms. □

Lemma 14.6.1 ([Apt, 1990]). $\Pi_{00}(\mathcal{D} \cup \Gamma)$ is acyclic. □

Proof:
The following level mapping $|\ |$ shows that the program $\Pi_{00} \cup \Gamma \cup \mathcal{D}$ satisfies the conditions for acyclicity.

Let c be the number of fluent literals in the language plus 1; p be a list of fluent literal, f be a fluent literal, and s be a sequence of actions.

For any action a, $|a| = 1$, $|[\,]| = 1$, and for any list $[a|r]$ of actions $|[a|r]| = |r| + 1$.

Also, for any fluent literal f, $|f| = 1$, $|[\,]| = 1$, and for any list $[f|p]$ of fluent literals $|[f|p]| = |p| + 1$.

$|holds\_after\_list(p, s)| = 6c * |s| + |p| + 4$

$|holds\_after(f, s)| = 6c * |s| + |f| + 4$

$|executable(s)| = 6c * |s| + 3$

$|impossible(s)| = 6c * |s| + 2$

$|ab(f,a,s)| = 6c * |s| + 5c + |f| + 1$

$|contrary(f,g)| = 1$, and all other atoms are mapped to 0. □

From the properties of acyclic programs [Apt, 1990] it follows that $\Pi_{00} \cup \mathcal{D} \cup \Gamma$ has a unique answer set.

Lemma 14.6.2. Let $H = \sigma_0, a_1, \sigma_1, \ldots a_n, \sigma_n$ be a history of the transition system defined by $\mathcal{D}$ that satisfy the axioms in Γ and let M be the answer set of $\Pi_{00} \cup \mathcal{D} \cup \Gamma$. Then,
for all i, $0 \leq i \leq n$, $f \in \sigma_i$ iff $holds\_ater(f, [a_i, \ldots, a_1])^*$ is true in M. □

Proposition 14.6.1 reduces the question of computing the consequence relation $\models_{\mathcal{D}}$ to computing entailment with respect to the logic program $\Pi_{00} \cup \mathcal{D} \cup \Gamma$. Computation with respect to a logic program depends on the interpreter used for making inferences. Since Prolog is the most popular logic programming language to date, we now consider using the Prolog interpreter, and view the program $\Pi_{00} \cup \mathcal{D} \cup \Gamma$ as a Prolog program with variables.

Proposition 14.6.2. The program $\Pi_{00} \cup \Gamma \cup \mathcal{D}$ is computable by the Prolog interpreter. I.e., The inference due to the Prolog interpreter on $\Pi_{00} \cup \Gamma \cup \mathcal{D}$ viewed as a Prolog program is sound and complete with respect to the answer set semantics of $\Pi_{00} \cup \Gamma \cup \mathcal{D}$. □

sketch. Let us start by listing the questions which need to be addressed to prove this proposition. First it is well known that for some programs the Prolog interpreter may produce unsound results. This may happen because of *the absence of the occur-check* which, in some cases, is necessary for soundness of the SLDNF resolution, or because the *interpreter may flounder*, i.e. may select for resolution a goal of the form *not* q where q contains an uninstantiated variable. Second, the interpreter *may fail to terminate*. Even if we show that for any $X \in \Pi_{00}$ and ground query q, the interpreter which takes $\Pi_{00} \cup X$ and q as an input terminates, does not flounder, and does not require the occur-check, the soundness of our result is guaranteed only with respect to the *unsorted* grounding of Π_{00}, i.e. the grounding of Π_{00} by terms of signature Σ_u obtained from signature Σ_0 by removing types and type information. In what follows we briefly discuss how these questions can be addressed. In particular we give hints about why (i) the program is occur-check free, (ii) it does not flounder, and (iii) it terminates. A proof based on our hints will be similar to the proofs in Section 7 of [Baral *et al.*, 1997]. □

- *Occur-check free:* To show that our program is occur-check free we use the result by Apt and Pellegrini in [Apt and Pellegrini, 1994] where they showed that if Π is well-moded [Dembinski and Maluszynski, 1985] for some input-output specification and there is no rule in Π whose head contains more than one occurrence of the same variable in its output positions then Π is occur-check free w.r.t. any ground query q. It can be shown that the following input-output specification, where '+' denotes input and '-' denotes output, indeed satisfies the above property. (For further details on this property please see [Dembinski and Maluszynski, 1985; Apt and Pellegrini, 1994; Baral *et al.*, 1997].)

  $impossible(+)$
  $executable(+)$
  $holds\_after(-,+)$
  $holds\_after\_list(-,+)$
  $ab(+,+,+)$
  $contrary(+,-)$
  $initially(-)$
  $causes(+,-,-)$
  $impossible\_if(+,-)$
  $fluent(+)$

*When $i = 0$, the list $[a_i, \ldots, a_1]$ denotes the empty list $[\,]$.

- *Does not flounder:* To show this property we can use another theorem from [Apt and Pellegrini, 1994] which was also independently discovered by Stroetman [Stroetman, 1993]: if Π is well-moded (for some input-output specification) and all predicate symbols occurring under *not* in Π are moded completely by input then a ground query $\pi(q)$ to Π does not flounder. The only two predicate symbols occurring under *not* in $\Pi_{00} \cup \Gamma \cup \mathcal{D}$ are $impossible$ and ab, and as required by the above mentioned condition, they both are moded completely by input.
- *Terminates:* Since $\Pi_{00} \cup \mathcal{D} \cup \Gamma$ is acyclic (From Lemma 14.6.1), termination follows as a property of acyclic programs [Apt, 1990]. □

To conclude the proof it suffices to apply the main result of [McCain and Turner, 1994] which reduces entailment in typed groundings of programs to entailment in their untyped groundings. □

14.7 Query language $\mathcal{Q}_1$

In this section we expand query language $\mathcal{Q}_0$ to be able to talk about the events that have actually taken place and about hypothetical actions that may be part of a history. The letters $t_0, t_1, \ldots$ are called *actual situations* and used to denote time points in the actual evolution of the system. If such evolution is caused by consecutive actions $a_1, \ldots, a_n$ then t_0 corresponds to the initial situation and t_k where $k \leq n$ corresponds to the end of the execution of $a_1, \ldots, a_k$.

The domain's past evolution is described by a set Γ of axioms in the query language $\mathcal{Q}_1$, which are expressions of the form

$$occurs\_at(a, t_k) \tag{14.8}$$

$$holds\_at(l, t_k) \tag{14.9}$$

The axiom (14.8) says that the action a has been executed in actual situation t_k; the axiom (14.9) indicates that l is true after a sequence of k consecutive actions has been actually executed. The proposition $initially(l)$ will be often used as a shorthand for $holds\_at(l, t_0)$. The axioms of Γ can be viewed as *observations*. Besides the actual situations, we have a special situation t_c that we refer to as the *current situation*, or the *current moment of time*. If there is a situation t_k with an axiom $occurs\_at(a, t_k)$, such that for every axiom in Γ with a situation t_i in it, $i \leq k$, then t_c is the situation t_{k+1}, otherwise t_c is the situation t_{max}, where max is the maximum j, such that there is an axiom about t_j in Γ.

Queries of $\mathcal{Q}_1$ are expressions of the form (14.9)

and of the form

$$currently(l) \tag{14.10}$$

$$holds\_after(l, [a_n, \ldots, a_1], t). \tag{14.11}$$

The query (14.10) states that l holds at the current moment of time. The query (14.11) is hypothetical and is read as: "sequence $a_1, \ldots, a_n$ of actions is executable in the situation t, and if it were executed, then fluent literal l would be true afterwards. If t is an actual situation that happened in the past 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. The following definitions refines the intuition behind the meaning of propositions of $\mathcal{Q}_1$.

Definition 14.7.1. Let T be a transition system, $H = \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ be a history of T, and Σ be a *situation map*, a mapping from situations to positive integers such that $i < j$ implies $\Sigma(t_i) < \Sigma(t_j)$, $\Sigma(t_0) = 0$ and for all t_i, $\Sigma(t_i) \leq \Sigma(t_c)$. We say that

- (H, Σ) *satisfies* an axiom $occurs\_at(a, t_k)$ if $a = a_{\Sigma(t_k)+1}$,

- (H, Σ) *satisfies* an axiom $holds\_at(l, t_k)$ if $l \in \sigma_{\Sigma(t_k)}$.

A pair (H, Σ) of history H of T and a situation map Σ is called a *model* of a set of axioms Γ if (H, Σ) satisfies all axioms from Γ and there does not exist a proper prefix H' of H such that for some situation map Σ', the pair (H', Σ') satisfies all axioms from Γ. Γ is called *consistent* if it has a model.

- a query $holds\_at(l, t_k)$ is a *consequence* of a set Γ of axioms with respect to T if, for every pair of history $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ and situation map Σ of T that is a model of Γ, $l \in \sigma_{\Sigma(t_k)}$;
- a query $currently(l)$ is a *consequence* of a set Γ of axioms with respect to T if, for every pair of history $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ and situation map Σ of T that is a model of Γ, $l \in \sigma_{\Sigma(t_c)}$;
- a query $holds\_after(l, [a'_m, \ldots, a'_1], t_k)$ is a *consequence* of a set Γ of axioms with respect to T if, for every pair of history $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ and situation map Σ of T that is a model of Γ, $a'_1, \ldots, a'_m$ is executable in $\sigma_{\Sigma(t_k)}$ and for any history $\sigma_0, a_1, \sigma_1, \ldots, \sigma_{\Sigma(t_k)-1}, a_{\Sigma(t_k)}, \sigma'_0, a'_1, \sigma'_1, \ldots, a'_m, \sigma'_m$ of T such that $\sigma'_0 = \sigma_{\Sigma(t_k)}$, $l \in \sigma'_m$.

As before, $\Gamma \models_{\mathcal{D}} Q$ will mean that the query Q is the consequence of Γ in the transition system T described by the action description $\mathcal{D}$. □

It can be shown that if a history $H = \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ and situation map Σ of T is a model of Γ then $\Sigma(t_c) = n$, as otherwise the part of the history after $\sigma_{\Sigma(t_c)}$ can be eliminated from the history without affecting the truth of the axioms, thus contradicting the conditions of being a model.

Example 14.7.1. Let $\mathcal{D}_0$ be the action description from Example 14.4.1 and consider the set Γ_1 of axioms of the form

$$\Gamma_1 \left\{ \begin{array}{l} initially(at(v_1, l_1)). \\ initially(at(v_2, l_2)). \\ \quad \ldots \\ initially(\neg at(V, L)) \ :- \ vehicle(V), \\ \qquad\qquad\qquad\qquad location(L), \\ \qquad\qquad\qquad\qquad not\ initially(at(V, L)). \\ \\ occurs\_at(move(v_1, l_1, l_2), t_0). \end{array} \right.$$

It can be shown that

$\Gamma_1 \models_{\mathcal{D}_0} currently(at(v_1, l_2))$

$\Gamma_1 \models_{\mathcal{D}_0} currently(\neg at(v_1, l_1))$

$\Gamma_1 \models_{\mathcal{D}_0} currently(at(v_2, l_2))$

$\Gamma_1 \models_{\mathcal{D}_0} holds\_after(at(v_1, l_1), [move(v_2, l_2, l_3)], t_0)$

$\Gamma_1 \models_{\mathcal{D}_0} holds\_after(at(v_2, l_2), [move(v_2, l_2, l_3)], t_0)$

$\Gamma_1 \models_{\mathcal{D}_0} holds\_after(at(v_2, l_2), [move(v_2, l_2, l_3)], t_1)$

The last two queries contain counterfactual and hypothetical statements respectively. Let us now consider $\Gamma_2 = \Gamma_1 \cup \{holds\_at(at(v_2, l_3), t_2)\}$. Clearly,

$\Gamma_2 \models_{\mathcal{D}_0} currently(\neg at(v_2, l_2))$

This demonstrates that the consequence relation of $\mathcal{Q}_1$ is nonmonotonic with respect to Γ. This is of course not surprising because the definition of the corresponding consequence relation incorporates the closed world assumption which roughly says that *no actions occur except those needed to explain*

observations of Γ. Notice also that the only possible history which satisfies axioms from Γ_2 starts at the fully defined initial state σ_0 and consists of the two actions $move(v_1, l_1, l_2)$ and $move(v_2, l_2, l_3)$. If we were to allow queries of the form (14.8) we would be able to conclude that Γ_2 entails $occurs\_at(move(v_2, l_2, l_3))$. □

In the rest of this paper, when using $\mathcal{Q}_1$, we will make some completeness assumptions.

- (i) Γ specifies a *complete initial situation*, and
- (ii) for all models (H, Σ) of Γ, with $H = \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$

  $\Sigma(t_i) = i$ and
  $occurs\_at(a_j, t_i) \in \Gamma$ iff $\Sigma(t_i) = j - 1$.

  In the above case we say that Γ specifies a *complete observation about the history* with respect to the transition system T.

Thus if a set of axioms Γ and a transition system T satisfy the above two assumptions then they have exactly one model (H, Σ), with $H = \sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$, where $\sigma_0 = \{l : initially(l) \in \Gamma\}$, and for $1 \leq i \leq n$, $occurs\_at(a_i, t_{i-1}) \in \Gamma$ and $\Sigma(t_i) = i$.

14.8 Answering queries in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_1)$

As in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$ the queries in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_1)$ will be answered by computing a program Π_{01} – similar to that of Π_{00} – together with action description $\mathcal{D}$ and axioms Γ.

The program Π_{01} will consist of the following rules:

1. Executability of Actions:

$$\Pi^1_{01} \left\{ \begin{array}{lll} impossible([A|S], T) & :- & \\ & & impossible(S, T). \\ impossible([A|S], T) & :- & \\ & & impossible\_if(A, P), \\ & & holds\_after\_list(P, S, T). \\ executable(S, T) & :- & \\ & & not\ impossible(S, T). \end{array} \right.$$

   These axioms are similar to that of Π_{00}. The difference is the existence of the new parameter T which stands for an actual situation. The statement $impossible([a_n, \ldots, a_1], t)$ says that the sequence $a_1, \ldots, a_n$ of actions cannot be executed in the actual situation t. The meaning of $possible$ and $hold\_after$ are also similar.

2. The Effect Axioms:

$$\Pi^2_{01}\left\{\begin{array}{lll}
holds_after(L,[\,],t_0) & :- & \\
 & & initially(L).\\
holds_after(L,[A|S],T) & :- & \\
 & & executable([A|S],T),\\
 & & causes(A,L,C),\\
 & & holds_after_list(C,S,T).\\
holds_after(L,[\,],T) & :- & \\
 & & holds_at(L,T).\\
holds_at(L,T_k) & :- & \\
 & & next(T_k,T_{k-1}),\\
 & & occurs_at(A,T_{k-1})\\
 & & holds_after(L,[A],T_{k-1}).\\
currently(L) & :- & \\
 & & current(T),\\
 & & holds_at(L,T).\\
current(T_k) & :- & \\
 & & next(T_k,T_{k-1}),\\
 & & occurs_at(A,T_{k-1}),\\
 & & nothing_happend(T_k).\\
something_happend(T) & :- & \\
 & & occurs_at(A,T).\\
nothing_happend(T) & :- & \\
 & & not\ something_happend(T).
\end{array}\right.$$

The first axiom in this group is similar to the corresponding axiom in Π_{00}. The second and the third axioms express the relationship between relations $holds\_after$ and $holds\_at$. The last four axioms define the current situation. The use of negation as failure is justified by our completeness assumption and is responsible for the non-monotonicity of our program with respect to queries of the form $currently(L)$.

3. The List Axioms: (Similar to Π^3_{00}.)

$$\Pi^3_{01}\left\{\begin{array}{lll}
holds_after_list([],_,T). & & \\
holds_after_list([L|Rest],S,T) & :- & \\
 & & holds_after(L,S,T),\\
 & & holds_after_list(Rest,S,T).
\end{array}\right.$$

4. The Inertia Axiom: (Similar to Π^4_{00}.)

$$\Pi^4_{01}\left\{\begin{array}{lll}
holds_after(L,[A|S],T) & :- & \\
 & & executable([A|S]),\\
 & & holds_after(L,S,T),\\
 & & not\ ab(L,A,S,T).
\end{array}\right.$$

5. Cancellation Axioms: (Similar to Π^5_{00}).)

$$\Pi^5_{01}\left\{\begin{array}{lll}
ab(L,A,S,T) & :- & \\
 & & contrary(L,NL),\\
 & & causes(A,NL,C),\\
 & & holds_after_list(C,S,T)
\end{array}\right.$$

6. Auxiliary rules Π^6_{01}: Same as Π^6_{00}.

The following proposition gives conditions for soundness and completeness of $\Pi_{01} \cup \mathcal{D} \cup \Gamma$.

Proposition 14.8.1. For any consistent action description $\mathcal{D}$, a consistent set of axioms Γ that specifies a complete initial situation and also a complete observation of the history with respect to the transition system of $\mathcal{D}$, and a query Q of $\mathcal{Q}_1$, $\Gamma \models_{\mathcal{D}} Q$ iff $\Pi_{01} \cup \mathcal{D} \cup \Gamma \models Q$. □

The proof of the above proposition is similar to the proof in [Baral *et al.*, 1997].

As in Section 14.6 we can show that the Prolog interpreter's inferencing by viewing $\Pi_{01} \cup \mathcal{D} \cup \Gamma$ as a Prolog program is sound and complete with respect to its answer set semantics. The proof of this results is similar to the proof of Proposition 14.6.2 and the proofs in Section 7 of [Baral *et al.*, 1997]. It will require us to show the following:

- *Occur-check free:* It can be shown that the following input-output specification, where '+' denotes input and '-' denotes output, satisfies the well-moded property that guarantees that the program is occur-check free.
  $impossible(+,+)$
  $executable(+,+)$
  $holds\_after(-,+,+)$
  $holds\_after\_list(-,+,+)$
  $ab(+,+,+,+)$
  $contrary(+,-)$
  $initially(-)$
  $causes(+,-,-)$
  $impossible\_if(+,-)$
  $holds\_at(-,+)$
  $currently(-)$
  $current(-)$
  $something\_happened(+)$
  $nothing\_happened(+)$
  $next(-,-)$
  $occurs\_at(-,-)$
  $fluent(+)$
- *Does not flounder:* To show this property, as in the proof sketch of Proposition 14.6.2 we can use the theorems from [Apt and Pellegrini, 1994; Stroetman, 1993] which states: if Π is well-moded (for some input-output specification) and all predicate symbols occurring under *not* in Π are moded completely by input then a ground query $\pi(q)$ to Π does not flounder. The only predicate symbols occurring under *not* in $\Pi_{01} \cup \mathcal{D} \cup \Gamma$ are $impossible$, ab, and $something\_happened$ and as required by the above mentioned condition, they both are moded completely by input.
- *Terminates:* To prove termination, we can use the acyclicity condition of [Apt, 1990]. We can show the acyclicity of $\Pi_{01} \cup \mathcal{D} \cup \Gamma$ by defining the following level mapping $|\ |$.

  Let c be the number of fluent literals in the language plus 1; p be a list of fluent literal, f be a fluent literal, s be a sequence of actions, t_i's be time points and t_{max} be the current time plus 1.

  For any action a, $|a| = 1$, $|[\,]| = 1$, and for any list $[a|r]$ of actions $|[a|r]| = |r| + 1$.

  For any fluent literal f, $|f| = 1$, $|[\,]| = 1$, and for any list $[f|p]$ of fluent literals $|[f|p]| = |p| + 1$.

  For any time point t_i, $|t_i| = i + 1$.

  $|holds\_after\_list(p, s, t)| = 10c * |t| + 4c * |s| + |p| + 4$

$|holds\_after(f,s,t)| = 10c * |t| + 4c * |s| + |f| + 4$

$|executable(s)| = 10c * |t| + 4c * |s| + 3$

$|impossible(s)| = 10c * |t| + 4c * |s| + 2$

$|ab(f,a,s,t)| = 10c * |t| + 4c * (|s| + 1) + |f| + 1$

$|holds\_at(f,t)| = 10c * |t| + |f| + 3$

$|currently(L)| = 15ct_{max}$

$|current(t)| = |t| + 3$

$|nothing\_happened| = |t| + 2$

$|something\_happened(t)| = |t| + 1$

$|contrary(f,g)| = 1$, and all other atoms are mapped to 0.

□

14.9 Incomplete axioms

In this section we discuss how to answer a query Q when the corresponding set Γ of axioms is incomplete. For simplicity we limit our discussion to the language $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$. First let us notice that the program Π_{00} from Section 14.6 can not be used for this purpose. Indeed, consider the following example:

Example 14.9.1. Let $\mathcal{D}_0$ be the action description from Example 14.4.1 and consider the set Γ_3 of axioms of the form

$$\Gamma_3 \left\{ \begin{array}{l} initially(at(v_1, l_1)). \\ initially(\neg at(v_1, l_2)). \\ initially(at(v_2, l_2)). \\ initially(\neg at(v_2, l_1)). \end{array} \right.$$

Let us also assume that our domain contains exactly three vehicles, v_1, v_2 and v_3, and two locations l_1 and l_2. The axioms specify positions of v_1 and v_2 and say nothing about the position of v_3. It can be shown that $\Pi_{00} \cup \mathcal{D} \cup \Gamma_3)$ entails query $Q = holds\_after(at(v_3, l_1), move(v_3, l_2, l_1))$, and hence the answer to Q is *yes*. The answer is incorrect, since Γ_3 has a model in which v_3 is initially located at position l_1. The action $move(v_3, l_2, l_1)$ is impossible in this position and hence $\Pi_{00} \cup \mathcal{D}_0 \cup \Gamma_3$ should entail neither Q nor $\neg Q$.

□

14.9.1 A sound but (possibly) incomplete formulation

There are several possible ways to modify Π_{00} to make it sound. The first modification, Π_{001}, is obtained from Π_{00} by replacing two groups of axioms as follows:

1. Executability of Actions:

$$\Pi^1_{001} \left\{ \begin{array}{lll} may\_be\_impossible([A|S]) & :- & \\ & & may\_be\_impossible(S). \\ may\_be\_impossible([A|S]) & :- & \\ & & impossible\_if(A, P), \\ & & not\ fail\_after(P, S). \\ executable(S) & :- & \\ & & not\ may\_be\_impossible(S). \end{array} \right.$$

2. Cancellation Axioms

$$\Pi^5_{001}\left\{\begin{array}{lll} ab(L,A,S) & :- & \\ & & contrary(L,NL),\\ & & causes(A,NL,C),\\ & & not\ fail\_after(C,S). \end{array}\right.$$

and adding the axiom

3. Falsification Axiom

$$\Pi^7_{001}\left\{\begin{array}{lll} fail\_after(C,S) & :- & \\ & & member(L,C),\\ & & contrary(NL,L),\\ & & holds\_after(NL,S). \end{array}\right.$$

Thus, let Π_{001} *be the set of axioms* $\Pi^1_{001} \cup \Pi^2_{00} \cup \Pi^3_{00} \cup \Pi^4_{00} \cup \Pi^5_{001} \cup \Pi^6_{00} \cup \Pi^7_{001}$.

It can be checked that neither query Q from Example 14.9.1 nor its negation is entailed by $\Pi_{001} \cup \mathcal{D}_0 \cup \Gamma_3$ and hence the answer to Q is *unknown*. The correctness of this answer is not an accident as it can be shown that Π_{001} is sound with respect to the consequence relation in $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$.

Proposition 14.9.1. For any consistent action description $\mathcal{D}$, consistent set of axioms Γ, and a query Q of $\mathcal{Q}_0$, if $Q \in \Pi_{001}(\mathcal{D} \cup \Gamma)$ then $\Gamma \models_{\mathcal{D}} Q$. □

sketch. From Definition 14.5.1 and Proposition 14.6.1 we have that to prove this proposition it suffices to show that if

$$Q \in \Pi_{001}(\mathcal{D} \cup \Gamma) \tag{14.12}$$

then

$$Q \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} \Pi_{00}(\mathcal{D} \cup \hat{\Gamma}) \tag{14.13}$$

Using the splitting set theorem we can simplify program Π_{001} by removing all the occurrences of literals formed by predicate symbols *causes* and *contrary*. It can be checked that the resulting program is signed and therefore, according to Proposition 14.2.2, monotonic. To conclude the proof it suffices to check that for a complete set of axioms $\hat{\Gamma}$, $\Pi_{001}(\mathcal{D} \cup \hat{\Gamma}) = \Pi_{00}(\mathcal{D} \cup \hat{\Gamma})$. □ □

The following example shows that for some action descriptions Π_{001} is incomplete.

Example 14.9.2. Let $\mathcal{D}_1$ be an action description

$causes(a, f, [p])$.
$causes(a, f, [\neg p])$.

Then it can be checked that $\emptyset \models_{\mathcal{D}_1} holds\_after(f, [a])$ while $holds\_after(f, [a]) \notin \Pi_{001}(\mathcal{D}_1)$ □

14.9.2 Soundness and completeness results for STRIPS action descriptions

Now let us consider *STRIPS action descriptions*, i.e. action descriptions consisting of causal laws of the form $causes(a, l_0, [\,])$ and $impossible\_if(a, [l_1, \ldots, l_n])$.* Notice that the action description from Example 14.9.1 belongs to this class.

Proposition 14.9.2. For any consistent STRIPS action description $\mathcal{D}$, any consistent set of axioms Γ, and a query Q of $\mathcal{Q}_0$, $Q \in \Pi_{001}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

sketch. It can be shown that for any Γ, $\Pi_{001}(\mathcal{D} \cup \Gamma)$ is categorical, i.e., it has a unique answer set. Let us denote this set by $A(\Gamma)$. The *if* part of the program follows immediately from Proposition 14.9.1. To prove the *only if* part we will first demonstrate that for any sequence α of actions

$$\text{if } executable(\alpha) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \text{ then } executable(\alpha) \in A(\Gamma) \tag{14.14}$$

We use induction on the length $|\alpha|$ of α. The base, $|\alpha| = 0$, follows immediately from the executability axioms (Π^1_{001}) of Π_{001}. Let $\alpha = [a|\beta]$,

$$executable(\alpha) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \tag{14.15}$$

and assume that (14.14) holds for β. Suppose now that

$$executable(\alpha) \notin A(\Gamma) \tag{14.16}$$

From (14.15) and the Executability axioms of Π_{001} we can conclude that

$$executable(\beta) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \tag{14.17}$$

By inductive hypotheses this implies that

$$executable(\beta) \in A(\Gamma) \tag{14.18}$$

From (14.16), (14.18), and the Executability axioms we conclude that there are fluent literals $l_1, \ldots, l_n$ such that

$$impossible\_if(a, [l_1, \ldots, l_n]) \in \mathcal{D} \tag{14.19}$$

and

$$fail\_after([l_1, \ldots, l_n], \beta) \notin A(\Gamma) \tag{14.20}$$

From (14.20) and the Falsification axiom we conclude that for any l_i $(1 \leq i \leq n)$

$$holds\_after(\overline{l_i}, \beta) \notin A(\Gamma) \tag{14.21}$$

Let

$$M = \{l_j : l_j \text{ satisfies (14.21) and } holds\_after(l_j, \beta) \notin A(\Gamma)\} \tag{14.22}$$

be the set of all fluent literals from the body of the causal law (14.19) whose truth values after the execution of β are undetermined. Since $\mathcal{D}$ is the STRIPS action description we can check (using the Effect, Inertia, and Cancellation axioms) that for any $\gamma = [a_1|\gamma_1]$ if

$$executable(\gamma) \in A(\Gamma) \text{ and } holds\_after(l, \gamma_1) \in A(\Gamma)$$

*This is an extension of a standard STRIPS representation language [Poole *et al.*, 1998]. Add and delete lists of this language correspond to causal laws of the type $causes(a, f, [\,])$ and $causes(a, \neg f, [\,])$ respectively. The precondition statement of STRIPS for an action a consists of a collection $p_1, \ldots, p_n$ of atomic fluents that need to be true for the action to be executable. In our action description language this corresponds to n statements of the form $impossible(\neg p_i)$ for all $1 \leq i \leq n$. Unlike the original STRIPS representation the STRIPS action descriptions allows to specify the effects of actions for incomplete descriptions of states.

then

$$holds\_after(l,\gamma) \in A(\Gamma) \text{ or } holds\_after(\overline{l},\gamma) \in A(\Gamma) \tag{14.23}$$

i.e., once the value of a fluent literal becomes determined it stays determined. From this observation and the construction of M we conclude that for any $l_j \in M$

$$initially(\overline{l_j}) \notin \Gamma \tag{14.24}$$

and for any action a_k from β

$$causes(a_k, l_j, [\,]) \notin \mathcal{D} \text{ and } causes(a_k, \overline{l_j}, [\,]) \notin \mathcal{D} \tag{14.25}$$

Let us now consider an extension $\hat{\Gamma}_0$ of Γ containing statements $initially(l_j)$ for any $l_j \in M$. From (14.16) we have that the body of (14.19) contains no contrary literals. This, together with (14.24) implies that $\hat{\Gamma}_0$ is consistent. From construction of $\hat{\Gamma}_0$, (14.25), and the Inertia axiom we have that

$$holds\_after(l_j,\beta) \in A(\hat{\Gamma}_0) \tag{14.26}$$

for all $l_j \in M$ and hence

$$executable(\alpha) \notin A(\hat{\Gamma}_0) \tag{14.27}$$

which contradicts our assumption (14.15). Hence

$$executable(\alpha) \in A(\Gamma) \tag{14.28}$$

To complete the proof we again use induction on α. The base case is obvious. Consider

$$holds\_after(l,[a|\beta]) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \tag{14.29}$$

This implies that

$$executable([a|\beta]) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \tag{14.30}$$

and hence, by (14.14),

$$executable(\alpha) \in A(\Gamma) \tag{14.31}$$

To show that

$$holds\_after(l,[a|\beta]) \in A(\Gamma) \tag{14.32}$$

we first consider the case when

$$causes(a,l,[\,]) \in \mathcal{D} \tag{14.33}$$

Then (14.29) follows immediately from (14.31) and the effect axioms. If (14.33) does not hold then (14.29) implies that

$$holds\_after(l,\beta) \in \bigcap_{\hat{\Gamma} \in c(\Gamma)} A(\hat{\Gamma}) \tag{14.34}$$

and hence, by the inductive hypothesis,

$$holds\_after(l,\beta) \in A(\Gamma) \tag{14.35}$$

Now (14.32) follows immediately from (14.31), (14.35), and the Inertia axioms. □ □

14.9.3 A general sound and complete formulation

The next modification of Π_{001} is obtained by adding to Π_{001} the following Initial Situation Axioms.

$$initially(l) \text{ :}-not\ initially(\bar{l}) \tag{14.36}$$

for every fluent literal l. Let us denote the resulting program by Π_{002}. Intuitively, the addition of these axioms corresponds to forcing the program to consider possible values of all fluents in the initial situation and do reasoning by cases, if necessary. To better understand these rules let us go back to action description $\mathcal{D}_1$ from Example 14.9.2. It can be checked that the program $\Pi_{002}(\mathcal{D}_1)$ has two answer sets, A_1 and A_2. Suppose that A_1 does not contain $initially(p)$. Then by rule (14.36), it must contain $initially(\neg p)$. Using the second causal law from Example 14.9.2 and the effect axioms we can conclude that A_1 contains $holds\_after(f,[a])$. Similarly we can show that A_2 contains $initially(p), holds\_after(f,[a])$, and therefore $holds\_after(f,[a]) \in \Pi_{002}(\mathcal{D}_1)$. This informal argument can easily be made precise. Moreover, the answer to a query $holds\_after(f,[a])$ can be computed by an extension of XSB, called SLG [Chen *et al.*, 1995] which allows reasoning with multiple answer sets. The following theorem shows that Π_{002} adequately represents entailment relation of $\mathcal{L}(\mathcal{A}_0, \mathcal{Q}_0)$.

Proposition 14.9.3. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_0$, any consistent set of axioms Γ, and a query Q of $\mathcal{Q}_0$, $Q \in \Pi_{002}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

Proof: Follows from using the splitting lemma (Lemma 14.2.1) and Proposition 14.6.1.

14.9.4 A sound and complete formulation using disjunction

Let us obtain Π_{003} from Π_{002} by replacing (14.36) by the following.

$$initially(f) \textbf{ or } initially(neg(f)) \leftarrow \tag{14.37}$$

We can now show that the following holds.

Proposition 14.9.4. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_0$, any consistent set of axioms Γ, and a query Q of $\mathcal{Q}_0$, $Q \in \Pi_{003}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

Proof: Follows from using splitting and Proposition 14.6.1.

An alternative approach is to use the formulation of abductive logic programs [Kakas and Mancarella, 1990a; Denecker and De Schreye, 1993] for which an interpreter [Denecker and De Schreye, 1997] exists. Other alternatives were suggested in [Kartha, 1993; Dung, 1993].

14.9.5 A sound and complete formulation using abduction

$$\Pi^2_{004}\left\{\begin{array}{lll}
holds_after(L,[\,]) & :- & \\
 & & fluent(L),\\
 & & initially(L).\\
holds_after(L,[A|S]) & :- & \\
 & & fluent(L),\\
 & & executable([A|S]),\\
 & & causes(A,L,C),\\
 & & holds_after_list(C,S).\\
holds_after(neg(L),S) & :- & \\
 & & fluent(L),\\
 & & executable(S),\\
 & & not\ holds_after(L,S),
\end{array}\right.$$

$$\Pi^4_{004}\left\{\begin{array}{ll} holds\_after(L,[A|S]) & :- \\ & \quad fluent(L), \\ & \quad executable([A|S]), \\ & \quad holds\_after(L,S), \\ & \quad not\ ab(L,A,S). \end{array}\right.$$

$$\Pi^5_{004}\left\{\begin{array}{ll} ab(L,A,S) & :- \\ & \quad fluent(L), \\ & \quad contrary(L,NL), \\ & \quad causes(A,NL,C), \\ & \quad holds\_after\_list(C,S). \end{array}\right.$$

Let Π_{004} be the general logic program consisting of Π^1_{00}, Π^2_{004}, Π^3_{00}, Π^4_{004}, Π^5_{004}, and Π^6_{00}. Given a set of axioms Γ, let Γ^* denote the conjunction of atoms in the following set: $\{initially(f)\ : initially(f) \in \Gamma$ and f is a fluent $\} \cup \{\neg initially(f)\ : initially(neg(f)) \in \Gamma$ and f is a fluent $\}$. Let C be the following formula $\forall f.\neg initially(neg(f))$. Let us now consider the abductive logic program $\langle \Pi_{004} \cup \mathcal{D}, \{initially\}, \Gamma^* \wedge C\rangle$. Intuitively, the constraint $\Gamma^* \wedge C$ force $\Delta \subset atoms(initially)$ to be facts about positive fluents only, and in such a way that it is consistent with Γ.

Proposition 14.9.5. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_0$, any consistent set of axioms Γ, and a query Q of $\mathcal{Q}_0$,

$\langle \Pi_{004} \cup \mathcal{D}, \{initially\}, \Gamma^* \wedge C\rangle \models_{abd} Q$ iff $\Gamma \models_{\mathcal{D}} Q$. □

sketch. The proof follows from the following three lemmas. In each of these lemmas $\mathcal{D}$ is a consistent action description and Γ is a consistent (but possibly incomplete) set of axioms. □

Lemma 14.9.1. Let $H = \sigma_0, a_1, \sigma_1, \ldots a_n, \sigma_n$ be a history of the transition system of $\mathcal{D}$ that satisfy the axioms in Γ and let M be a generalized stable model of $\langle \Pi_{004} \cup \mathcal{D}, \{initially\}, \Gamma^* \wedge C\rangle$ such that $\sigma_0 = \{f : initially(f) \in M\} \cup \{\neg f : initially(f) \notin M\}$. Let $[a_0, \ldots, a_1]$ denote the list $[\,]$. Then,
For all i, $0 \leq i \leq n$, $f \in \sigma_i$ iff $holds\_ater(f, [a_i, \ldots, a_1])$ is true in M. □

The above lemma can be proved by induction on i.

Lemma 14.9.2. For every history $H = \sigma_0, a_1, \sigma_1, \ldots a_n, \sigma_n$ of the transition system of $\mathcal{D}$ that satisfy the axioms in Γ there exists a generalized stable model M_H of $\langle \Pi_{004} \cup \mathcal{D}, \{initially\}, \Gamma^* \wedge C\rangle$ such that $\sigma_0 = \{f : initially(f) \in M_H\} \cup \{\neg f : initially(f) \notin M_H\}$. □

Lemma 14.9.3. For every generalized stable model M of $\langle \Pi_{004} \cup \mathcal{D}, \{initially\}, \Gamma^* \wedge C\rangle$ there exists a history $H_M = \sigma_0, a_1, \sigma_1, \ldots a_n, \sigma_n$ of the transition system of $\mathcal{D}$ that satisfy the axioms in Γ such that $\sigma_0 = \{f : initially(f) \in M\} \cup \{\neg f : initially(f) \notin M\}$ □

14.10 Action description language $\mathcal{A}_1$

In this section we consider an extension $\mathcal{A}_1$ of action description language $\mathcal{A}_0$ from Section 14.4. As before we consider a fixed action signature Σ_0. Propositions of $\mathcal{A}_1$ are expressions of the form

$$impossible\_if(a, [l_1, \ldots, l_n]) \tag{14.38}$$

$$causes(a, l_0, [l_1, \ldots, l_n]) \tag{14.39}$$

$$causes(l_0, [l_1, \ldots, l_n]) \tag{14.40}$$

The first two propositions are exactly those allowed in $\mathcal{A}_0$. The last proposition says that, in the action domain being described, whenever $l_1, \ldots, l_n$ are caused, l_0 is caused. Propositions of this form are

called *static causal laws** To better understand the use of these laws for representing knowledge about effects of actions let us go back to Example 14.4.1. The transition diagram of the domain description $\mathcal{D}_0$ from this example contains states in which the same vehicle occupies more than one location. This possibility can be eliminated if we assume that a vehicle in our domain can not be in two locations at the same time. This information can be represented in $\mathcal{A}_1$ by static causal laws of the form

$causes(\neg at(v, l_1), [at(v, l_2)])$

where v is a vehicle and l_1 and l_2 are different locations. As before, this can be written as a logic programming rule

$$\begin{array}{lll} causes(\neg at(V, L_1), [at(v, L_2)]) & :- & \\ & & vehicle(V) \\ & & location(L_1) \\ & & location(L_2) \\ & & diff(L_1, L_2). \end{array}$$

Inclusion of this law makes the dynamic causal law

$causes(move(v, l_1, l_2), \neg at(v, l_3), [\,])$.

of $\mathcal{D}_0$ redundant and therefore it can be removed. Let us consider an action description $\mathcal{D}_2$ of $\mathcal{A}_1$ given below:

$$\mathcal{D}_2 \left\{ \begin{array}{l} causes(move(v, l_1, l_2), at(v, l_2), [\,]). \\ causes(\neg at(v, l_1), [at(v, l_2)]) \\ impossible\_if(move(v, l_1, l_2), [\neg at(v, l_1)]). \\ \text{where } v\text{'s are vehicles and } l_1, and l_2 \text{ are locations and } l_1 \neq l_2. \end{array} \right.$$

Intuitively, $\mathcal{D}_2$ describes the same transition diagram as in Figure 14.1, if we assume a single vehicle v and two locations l_1 and l_2.

We are now ready to define the semantics of $\mathcal{A}_1$ (based on the characterization in [McCain and Turner, 1995]) that uses the following terminology and notations. A set s of literals is closed under a set Z of static causal laws if s includes the head, l_0, of every static causal law (14.40) such that $\{l_1, \ldots, l_n\} \subseteq S$. The set $Cn_Z(s)$ of *consequences* of s under Z is the set of all literals that contain s and is closed under Z. Let $\mathcal{D}$ be an action description in $\mathcal{A}_1$. The transition system $T = \langle \mathcal{S}, \mathcal{R} \rangle$ *described* by $\mathcal{D}$ is defined as follows.

1. $\mathcal{S}$ is the collection of all complete and consistent sets of fluent literals of Σ_0 closed under the static laws of $\mathcal{D}$,
2. $\mathcal{R}$ is the set of all triples (σ, a, σ') such that $\mathcal{D}$ does not contain a proposition of the form $impossible\_if(a, [l_1, \ldots, l_n])$ such that $[l_1, \ldots, l_n] \subseteq \sigma$ and

$$\sigma' = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma')) \tag{14.41}$$

where Z is the set of all static causal laws of $\mathcal{D}$, and $E(a, \sigma)$ is the set of the heads l_0 of dynamic causal laws $causes(a, l_0, [l_1, \ldots, l_n]$ of $\mathcal{D}$ such that $\{l_1, \ldots, l_n\} \subseteq s$. The argument of $Cn(Z)$ in (14.41) is the union of the set $E(a, s)$ of the "direct effects" of a with the set $s \cap s'$ of facts that are "preserved by inertia". The application of $Cn(Z)$ adds the "indirect effects" to this union.

The following example shows that addition of static causal laws substantially increases expressive power of our language.

*The paper [Marek and Truszczyński, 1994] was perhaps the first work that inspired later logic programming and default logic based formulations of static causal laws [Baral, August 1994; Baral, 1997; Baral, 1995; McCain and Turner, 1995; McCain and Turner, 1997a; Turner, 1997]. Alternative formulations of causality while reasoning about actions were suggested in [Lin, 1995; Thielscher, 1997a; Lifschitz, 1997a].

Example 14.10.1. Let $\mathcal{D}_3$ be an action description

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

The transition system T_2 described by $\mathcal{D}_3$ is represented by Figure 14.2.

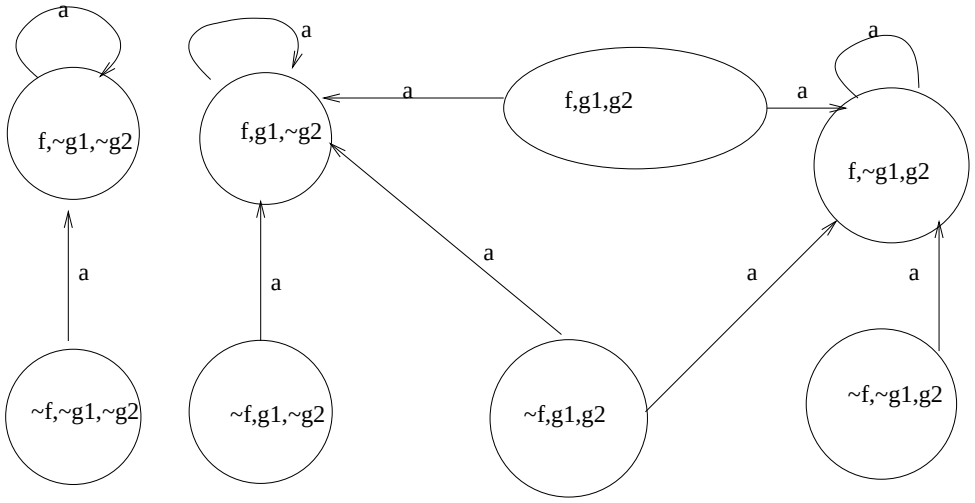

Figure 14.2: Transition Diagram

The diagram is nondeterministic and therefore cannot be described by a domain description of $\mathcal{A}_0$. □

We now give some conditions which guarantee that an action description $\mathcal{D}$ of $\mathcal{A}_1$ is deterministic, i.e., describes a deterministic transition system. Let R be a collection of static causal laws of $\mathcal{D}$. For any action a and a state σ, by $E^*(a, \sigma)$ we will denote the closure of direct effects, $E(a, \sigma)$, of executing a in σ with respect to R. We will say that $\mathcal{D}$ is *separable* if for any a and σ such that a is executable in σ, if $r \in R$ and $body(r) \cap E^*(a, \sigma) \neq \emptyset$ then $body(r) \subseteq E^*(a, \sigma)$.

Proposition 14.10.1. Any separable action description $\mathcal{D}$ of $\mathcal{A}_1$ is deterministic. □

Proof. Let $T = \langle \mathcal{S}, \mathcal{R} \rangle$ be a transition system described by $\mathcal{D}$. We need to show that for any action a and states $\sigma, \sigma^1, \sigma^2 \in \mathcal{S}$ if

1. $(\sigma, a, \sigma^1) \in \mathcal{R}$ and $(\sigma, a, \sigma^2) \in \mathcal{R}$ then

2. $\sigma^1 = \sigma^2$.

From definition (14.41) of $\mathcal{R}$ and the assumption (1) we have

3. $\sigma^1 = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma^1))$

which is equivalent to

4. $\sigma^1 = Cn_Z(E^*(a, \sigma) \cup (\sigma \cap \sigma^1))$.

By separability of $\mathcal{D}$ (4) is equivalent to

5. $\sigma^1 = E^*(a, \sigma) \cup Cn_Z(\sigma \cap \sigma^1)$.

Since σ and σ_1 are states they are closed under the rules of Z and hence (5) is equivalent to

6. $\sigma^1 = E^*(a, \sigma) \cup (\sigma \cap \sigma^1)$.

Similarly, we can show that

7. $\sigma^2 = E^*(a, \sigma) \cup (\sigma \cap \sigma^2)$.

To prove (2) let us assume that $l \in \sigma^1$. By (6) we have that

8. $l \in E^*(a, \sigma)$ or

9. $l \in \sigma$.

If (8) holds then from (7) we have that $l \in \sigma_2$. If (8) does not hold then we have (9). Since states are complete and consistent sets of literals this implies that $l \in \sigma^2$. This completes the proof. □

14.11 Answering queries in $\mathcal{L}(\mathcal{A}_1, \mathcal{Q}_0)$ and $\mathcal{L}(\mathcal{A}_1, \mathcal{Q}_1)$

In this section we illustrate the use of logic programming for computing consequences of domain descriptions of $\mathcal{L}(\mathcal{A}_1, \mathcal{Q}_0)$ and $\mathcal{L}(\mathcal{A}_1, \mathcal{Q}_1)$. As in Section 14.6 we assume that $\mathcal{D}$ is consistent and make some completeness assumptions about Γ. The corresponding programs Π_{10} and Π_{11} are obtained from Π_{00} and Π_{01} respectively by adding the following rules:

- Π_{10} is Π_{00} plus the following two rules.

$$
\begin{array}{l}
holds\_after(L, S) \quad :- \\
\qquad\qquad executable(S), \\
\qquad\qquad causes(L, C), \\
\qquad\qquad holds\_after\_list(C, S).
\end{array}
$$

$$
\begin{array}{l}
ab(L, A, S) \quad :- \\
\qquad\qquad contrary(L, NL), \\
\qquad\qquad causes(NL, C), \\
\qquad\qquad holds\_after\_list(C, [A|S])
\end{array}
$$

- Π_{11} is Π_{01} plus the following two rules.

$$
\begin{array}{l}
holds\_after(L, S, T) \quad :- \\
\qquad\qquad executable(S, T), \\
\qquad\qquad causes(L, C), \\
\qquad\qquad holds\_after\_list(C, S, T).
\end{array}
$$

$$
\begin{array}{l}
ab(L, A, S, T) \quad :- \\
\qquad\qquad contrary(L, NL), \\
\qquad\qquad causes(NL, C), \\
\qquad\qquad holds\_after\_list(C, [A|S], T)
\end{array}
$$

Proposition 14.11.1. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_1$, any consistent set of axioms Γ that specifies a complete initial situation, and a query Q of $\mathcal{Q}_0$, $Q \in \Pi_{10}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

Proposition 14.11.2. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_1$, a consistent set of axioms Γ that specifies a complete initial situation and also a complete observation of the history with respect to the transition system of $\mathcal{D}$, and a query Q of $\mathcal{Q}_1$, $Q \in \Pi_{11}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

The proofs of Propositions 14.11.1 and 14.11.2 are similar to that of Propositions 14.6.1 and 14.8.1 respectively.

As before this does not work if Γ is not complete. The complete initial situation assumption can be removed by expanding Π_{11} by the rules:

$$\begin{array}{lll} initially(l) & :- & \\ & & contrary(l,\bar{l}), \\ & & not\ initially(\bar{l}) \end{array}$$

where $contrary(l,\bar{l})$ holds iff l and $\bar{l}$ are contrary fluent literals. The resulting program will be denoted by Π_{110}.

Proposition 14.11.3. For any consistent action description $\mathcal{D}$ of $\mathcal{A}_1$, a consistent set of axioms Γ that specifies a complete observation of the history with respect to the transition system of $\mathcal{D}$, and a query Q of $\mathcal{Q}_1$, $Q \in \Pi_{110}(\mathcal{D} \cup \Gamma)$ iff $\Gamma \models_{\mathcal{D}} Q$. □

Proof: Follows from using splitting and Proposition 14.11.2.

The difficulty of the computation of $\Pi_{11} \cup \mathcal{D} \cup \Gamma$ and $\Pi_{110} \cup \mathcal{D} \cup \Gamma$ is dependent on whether D describes a deterministic transition diagram. If it does then there will be a single stable model of the program and we can use the XSB interpreter. Otherwise, there may be multiple stable models of the program and we would have to use interpreters such as the Smodels and DLV systems.

14.12 Planning using model enumeration

The DLPs in the previous sections are most appropriate for *verifying* if a particular fluent literal is true after the execution of a sequence of actions. They can be used for planning by using interpreters that do answer extraction. In this section we show how the DLPs can be adapted so that planning can be done through model enumeration.

In the model enumeration approach [Subrahmanian and Zaniolo, 1995] each stable model of our program corresponds to a particular hypothetical evolution of the world. We guess a minimal plan length for a given goal and that information is part of the program. The stable models where the goal is not true at the guessed plan length are eliminated by adding appropriate constraints to the program. The stable models that are not weeded out give us plans that achieve the given goals at the guessed plan length. To make sure that each stable model of our program corresponds to a possible evolution of the world we have executability axioms, effect axioms, inertia axioms, etc., with the modification that instead of situations we use plan length or time as the basis of how the world evolves. This approach to planning has recently been called as *answer-set planning* [Lifschitz, 1999], where *answer-sets* is a more general term for stable models. Answer-set planning is a particular instance of the more general notion of *answer set programming* where queries with respect to a logic program are answered through the bottom-up approach of generating answer sets and evaluating the query with respect to them rather than through the top down approach of unification and resolution. One advantage of the answer set programming [Marek and Truszczyński, 1999; Niemelä, 1999; Eiter *et al.*, 2000b] approach is that it takes advantage of multiplicity of answer sets by treating them as a solution space, and allows us to implement the brave semantics (i.e., entailment with respect to some answer set rather than all answer sets) of logic programs.

We now give an example of how planning is done with respect to our vehicle example. In Section 14.12.1 we describe a downtown with one-way streets and do planning to go from one location to another. In Section 14.12.2 we allow observations, and do planning from the current situation.

14.12.1 Navigating a downtown with one-way streets

Consider the one-way streets in Anymetro USA given in Figure 14.3. The driver of vehicle v would like to go from the point l_3 to point l_2. Following are the domain dependent and domain independent axioms that the driver has.

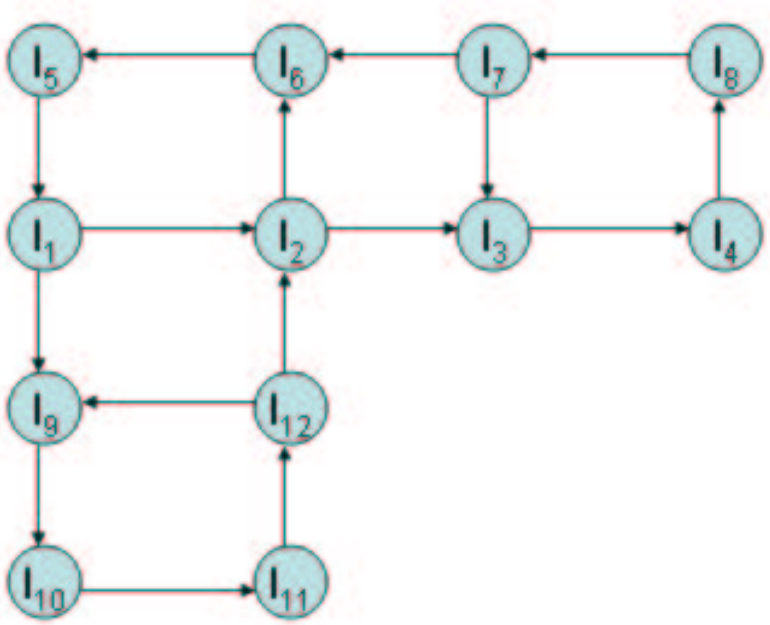

Figure 14.3: One-way streets in downtown Anymetro, USA

1. The domain dependent part

   (a) The initial street description:

   $initially(edge(l_1, l_2)).$

   $initially(edge(l_2, l_3)).$

   $initially(edge(l_3, l_4)).$

   $initially(edge(l_4, l_8)).$

   $initially(edge(l_8, l_7)).$

   $initially(edge(l_7, l_6)).$

   $initially(edge(l_6, l_5)).$

   $initially(edge(l_5, l_1)).$

   $initially(edge(l_2, l_6)).$

   $initially(edge(l_7, l_3)).$

   $initially(edge(l_1, l_9)).$

   $initially(edge(l_9, l_{10})).$

$initially(edge(l_{10}, l_{11})).$

$initially(edge(l_{11}, l_{12})).$

$initially(edge(l_{12}, l_2)).$

$initially(edge(l_{12}, l_9)).$

(b) The initial position of the vehicle

$initially(at(v, l_3)).$

(c) The goal state

$finally(at(v, l_2)).$

(d) When actions are not executable

$impossible\_if(move(V, L1, L2), neg(at(V, L1))).$

$impossible\_if(move(V, L1, L2), neg(edge(L1, L2))).$

(e) The effect of actions

$causes(move(V, L1, L2), at(V, L2)).$

$causes(move(V, L1, L2), neg(at(V, L1))).$

2. The domain independent part

(a) Defining $goal(T)$: The goal is true in time T.

$$\begin{aligned} not\_goal(T) :- \quad & time(T), \\ & finally(X), \\ & not\ holds(X, T). \\ goal(T) :- \quad & time(T), \\ & not\ not\_goal(T). \end{aligned}$$

(b) Eliminating models which do not have a plan of the given length.

$$:- \quad not\ goal(length).$$

(c) Defining contrary

$contrary(F, neg(F)).$
$contrary(neg(F), F).$

(d) What holds in time point 1.

$$\begin{aligned} holds(F, 1) :- \quad & initially(F). \\ holds(neg(F), 1) :- \quad & not\ holds(F, 1). \end{aligned}$$

(e) Effect axiom

$$\begin{aligned} holds(F, T+1) :- \quad & T < length, \\ & executable(A, T), \\ & occurs(A, T), \\ & causes(A, F). \end{aligned}$$

(f) Inertia

$$\begin{aligned} holds(F, T+1) :- \quad & contrary(F, G), \\ & T < length, \\ & holds(F, T), \\ & not\ holds(G, T+1). \end{aligned}$$

(g) We need rules that define executability in terms of the $impossible\_if$ conditions given in the domain dependent part. These rules are:

$$\begin{aligned} not\_executable(A,T) :- \quad & impossible\_if(A,B), \\ & holds(B,T) \\ executable(A,T) :- \quad & not\ not\_executable(A,T). \end{aligned}$$

(h) What actions are possible at each time point? A simple formulation of this could be to encode that at any time point all executable actions are possible if the goal is not reached.

$$\begin{aligned} possible(A,T) :- \quad & action(A), \\ & executable(A,T), \\ & not\ goal(T). \end{aligned}$$

(i) Occurrences of actions

$$\begin{aligned} occurs(A,T) :- \quad & action(A), \\ & possible(A,T), \\ & not\ not\_occurs(A,T). \\ not\_occurs(A,T) :- \quad & action(A), \\ & action(AA), \\ & occurs(AA,T), \\ & A \neq AA. \end{aligned}$$

When the above program is given to the interpreter Smodels [Niemelä and Simons, 1997] one of the stable models that is generated has the following literals describing a plan.

$occurs(move(v,l_3,l_4),1).$
$occurs(move(v,l_4,l_8),2).$
$occurs(move(v,l_8,l_7),3).$
$occurs(move(v,l_7,l_6),4).$
$occurs(move(v,l_6,l_5),5).$
$occurs(move(v,l_5,l_1),6).$
$occurs(move(v,l_1,l_2),7).$

We refer to the domain independent part of the above program as $\Pi_{00.planning}$. The following proposition states the correctness of the program $\Pi_{00.planning}$ for planning when we are given a consistent domain description and an initial state complete set of axioms.

Proposition 14.12.1. Let D be a consistent domain description in $\mathcal{A}_0$ and Γ be an initial state complete set of axioms in $\mathcal{Q}_0$. Let $length$ be a positive integer and G be a set of fluent literals that we want to be true in the goal state.

(i) If there is a sequence of actions $a_1,\ldots,a_{length}$ such that for each literal l in G, $\Gamma \models_D holds\_after(l,[a_{length},\ldots,a_1])$, then $\Pi_{00.planning} \cup D \cup \Gamma \cup \{finally(l) : l \in G\}$ has an answer set with $\{occurs(a_1,1),\ldots,occurs(a_{length},length)\}$ as the set of facts about $occurs$ in it.

(ii) If $\Pi_{00.planning} \cup D \cup \Gamma \cup \{finally(l) : l \in G\}$ has an answer set with $\{occurs(a_1,1),\ldots,occurs(a_{length},length)\}$ as the set of facts about $occurs$ in it then for each literal l in G, $\Gamma \models_D holds\_after(l,[a_{length},\ldots,a_1])$. □

A specific instance of the above proposition is the case where Γ consists of the rules in part 1(a) and 1(b) above and $\mathcal{D}$ consists of the rules in part 1(d) and 1(e) above.

14.12.2 Downtown navigation: planning while driving

Consider the case that an agent uses the planner in the previous section and makes a plan. It now executes part of the plan, where it moves from l_3 to l_4 and l_4 to l_8, and then hears in the radio that *an accident occurred between point* l_1*and* l_2 and that section of the street is blocked. The agent now has

to make a new plan from where it is to its destination. To be able to encode observations and make plans from the current situation we need to add the following to our program in the previous section.

1. The domain dependent part
   (a) The observations

   $occurs\_at(move(v, l_3, l_4), 1).$

   $occurs\_at(move(v, l_4, l_8), 2).$

   $occurs\_at(acc(l_1, l_2), 3).$

   (b) Exogenous actions

   $causes(acc(X, Y), neg(edge(X, Y))).$

2. The domain independent part
   (a) Relating $occurs\_at$ and $occurs$

   $occurs(A, T) :- \quad occurs\_at(A, T).$

With these additions one of the plans generated by Smodels is as follows:

$occurs(move(v, l_8, l_7), 4).$
$occurs(move(v, l_7, l_6), 5).$
$occurs(move(v, l_6, l_5), 6).$
$occurs(move(v, l_5, l_1), 7).$
$occurs(move(v, l_1, l_9), 8).$
$occurs(move(v, l_9, l_{10}), 9).$
$occurs(move(v, l_{10}, l_{11}), 10).$
$occurs(move(v, l_{11}, l_{12}), 11).$
$occurs(move(v, l_{12}, l_2), 12).$

Although it does not matter in the particular example described above, we should separate the set of actions to $agent\_actions$ and $exogenous\_actions$, and in the planning module require that while planning we only use $agent\_actions$. This can be achieved by replacing the two rules about $occurs$ by the following rules.

$$\begin{aligned} occurs(A, T) :- & \quad occurs\_at(A, T). \\ occurs(A, T) :- & \quad agent\_action(A), \\ & \quad possible(A, T), \\ & \quad not\ not\_occurs(A, T). \\ not\_occurs(A, T) :- & \quad not\ occurs(A, T). \\ :- & \quad occurs(A, T), occurs(AA, T), A \neq AA. \end{aligned}$$

14.13 Concluding Remarks

In this chapter we presented a series of logic programming (with stable model semantics and its generalizations) based action theories with increasing expressibility, and with special emphasis on (i) using an independent automata based semantics for defining correctness, (ii) developing executable programs, and (iii) dealing with incompleteness. These aspects have been among our main interests in the last 8-9 years. Some of the other aspects of logic programming based reasoning about actions that we and other researchers worked on but which we did not discuss here are: reasoning about concurrent actions [Baral and Gelfond, 1997], reasoning with narratives [Baral *et al.*, 1997; Pinto and R.Reiter, 1993], using action theories to develop an agent architecture [Baral *et al.*, 1997], and reasoning about resources [Holldobler and Thielscher, 1993]. An important work which we would

like mention here is [Lin, 1997] where Lin gives semantics of the cut operator of Prolog using an action theory.

Amongst the emerging areas, one of the most important is the area of model based planning using logic programming. Starting with the development of the S-models [Niemelä and Simons, 1997] and the work by Dimopolous et al. [Dimopoulos *et al.*, 1997], there has been a lot of recent research [McCain and Turner, 1998; Lifschitz, 1999; Erdem and Lifschitz, 1999; Baral and Gelfond, 2000] in this area. In Section 14.12 we gave a quick introduction to this.

In terms of related future and ongoing work, some of the questions that are being currently addressed and not elaborated in this chapter are: (i) using domain knowledge[Son *et al.*, 2001] and heuristics [Balduccini *et al.*, 2000; Gelfond, 2001] in model based planning using logic programming. (ii) using action theories to develop a notion of diagnosis [Baral and Gelfond, 2000; Gelfond *et al.*, 2001], (iii) using interpreters that can accommodate disjunctive logic programs (such as the DLV interpreter [Eiter *et al.*, 2000a]) to develop planners that generate plans with sensing actions, and (iv) developing more general results about when a transition function is deterministic.