Languages for Planning

Participating Units: University of Genova, ITC-irst, University of Trento

Abstract. Planning is a traditional research area in AI. The last decade has witnessed a large body of work in the development of languages and algorithms for planning, up to the point where complex, realistic problems can be modeled and solved in seconds. At the same time, thanks to the generality of the problem formulation, several applicative problems can be recasted into planning. This is also true in the SOC area, where planning can be used to automatically compose distributed services to provide new services satisfying domain and user specified requirements. In this deliverable, we focus on languages for specifying domain and user requirements. The first correspond to languages for the specification of planning domains, and latter correspond to languages for the specification of user's goals.

Document Identifier	Deliverable D4.2
Project	MIUR-FIRB project RBNE0195K5 "Knowledge Level Automated Software Engineering"
Version	v1.0
Date	May 24, 2006
State	Final
Distribution	Public

Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’I-struzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A..

Executive Summary

Planning is a traditional research area in AI. The last decade has witnessed a large body of work in the development of languages and algorithms for planning, up to the point where complex, realistic problems can be modeled and solved in seconds. At the same time, thanks to the generality of the problem formulation, several applicative problems can be recasted into planning. This is also true in the SOC area, where planning can be used to automatically compose distributed services to provide new services satisfying domain and user specified requirements.

In this deliverable we recap the research performed by the partners of the KLASE project to design advanced languages for the specification of domain and user requirements. Domain requirements specify the underlying scenario within which the different actors operate and the capabilities and interactions among the different actors. In the planning terminology, they correspond to the specification of the underlying scenario within which actors operate. User requirements represents the goals that, given the requirements imposed by the domain, the actors have to satisfy. In the SOC framework, actors correspond to services and the issue of finding a plan satisfying the domain and user requirements correspond to determine a service composition meeting all the requirements.

In this deliverable, we first report on the research conducted on languages for the specification of planning scenarios (domain constraints), and then on the languages for the specification of goals (user's requirements).

Contents

Chapter 1

Languages for specifying planning scenarios

1.1 Introduction

The problem of describing changes caused by the execution of actions is as old as logic-based Artificial Intelligence. In the "advice taker" paper by John McCarthy [McC59], effects of actions are described by first-order formulas, such as

$$\mathit{did}(\mathit{go}(\mathit{desk}, \mathit{car}, \mathit{walking})) \supset \mathit{at}(I, \mathit{car}).$$

Several other approaches to describing effects of actions have been proposed later, including the situation calculus [MH69], the STRIPS language [FN71] and the event calculus [KS86].

A large body of research has been devoted to the frame problem—the problem of describing what does *not* change when actions are performed [Sha97]. This work has led to the emergence of several formal theories of nonmonotonic reasoning, including circumscription [McC80], [McC86] and default logic [Rei80].

Several researchers addressed the frame problem by distinguishing between causal and noncausal propositions. When we say that E is an effect of an action A, we assert not only a material implication ("if A has just been executed then E holds") but also the existence of a causal relationship between A and E ("the execution of A causes E"). In the natural sciences, it is common practice to disregard distinctions like this. In AI, surprisingly, formal theories of causal reasoning that stress this distinction have turned out to be quite useful.

In our causal logic, we distinguish between being true and having a cause, but we do not attempt to talk about what a cause may be. Strictly speaking, there is no way to express in our formal notation that the execution of an action A is the cause for E; what we write instead is that *there exists* a cause for E if A has just been executed. Syntactically, this can be expressed by combining material implication $\supset$ with a modal operator C that is used to express the existence of a cause:

$$A \supset \mathsf{C}E \tag{1.1}$$

[Gef90], [Tur99]. The syntax defined we use here is more limited. We use "causal rules" of the form

$$F \Leftarrow G$$

where F and G are formulas of classical logic, in place of the combination

$$G \supset \mathsf{C}F$$

("there is a cause for F to be true if G is true"). In particular, we can express that there is a cause for F to be true by writing the causal rule

$$F \Leftarrow \top.$$

The noncausal assertion that F is true can be expressed by

$$\bot \Leftarrow \neg F.$$

(Symbols $\top$ and $\bot$ are 0-place propositional connectives.) Expression (1.1) can be written as the causal rule

$$E \Leftarrow A.$$

We will define a causal theory to be a set of causal rules.

Our semantics of causal theories can be informally summarized as follows: every fact that is caused obtains, and vice versa. This assertion is made precise in Proposition 1 (Section 1.2.3). Its second half—every fact that obtains is caused—expresses the "principle of universal causation," a rather strong philosophical commitment that is rewarded by mathematical simplicity in the semantics of causal theories. We will soon see how the principle of universal causation can be "disabled" for selected formulas.

To illustrate the idea of universal causation, we will show how to use it, in an informal way, to find the models of a simple causal theory. Take the following example:

$$\begin{array}{l} p \Leftarrow q, \\ q \Leftarrow q, \\ \neg q \Leftarrow \neg q. \end{array} \tag{1.2}$$

Here p, q are atoms. Which interpretations (that is, truth-valued functions on $\{p, q\}$) are models of (1.2)?

Causal rules (1.2) assert that, under various conditions, there is a cause for p, there is a cause for q, and there is a cause for $\neg q$. But no rule in (1.2) may lead to the conclusion that there is a cause for $\neg p$. Consequently,

$\neg p$ is not caused.

Since we require that every true formula be caused, we can conclude that

$$\neg p \text{ is not true}$$

or, equivalently,

$$p \text{ is true.}$$

Again because every true formula is caused, it must be the case that

$$p \text{ is caused.}$$

The only way to use rules (1.2) to establish that p is caused is to refer to rule $p \Leftarrow q$, which says: p is caused if q is true. Consequently,

$$q \text{ is true.}$$

This informal argument shows that the interpretation making both p and q true is the only possible model of (1.2). And it is indeed a model, because, under this interpretation, the first two rules of (1.2) guarantee the existence of causes both for p and for q.

The last two rules of (1.2) illustrate the point made above about relaxing the principle of universal causation—they "disable" it for formula q. These rules express that if q is true then there is a cause for this, and, on the other hand, if q is false then there is a cause for that.

In the next section, we define the semantics of causal theories and study the special case of "definite" theories. In Section 1.3 we show how to use definite theories to describe action domains; the Monkey and Bananas problem[1] is used as the main example. "Causal laws"—higher-level notation for describing actions, similar to the action language $\mathcal{C}$ from [GL98]—are introduced in Section 1.4, and further examples illustrating the expressive possibilities of definite causal laws are discussed in Section 1.5. In Section 1.8 we compare this work with other research on nonmonotonic reasoning and knowledge representation. Proofs of theorems are presented in Section 1.9. We conclude by discussing the relation of this work to the idea of elaboration tolerance [McC03]. Section 1.6 relates formulas of "multi-valued propositional signatures" to usual propositional formulas. Finally, Section 1.7 is a list of useful abbreviations for special kinds of causal laws.

1.2 Causal Theories

1.2.1 Formulas

The class of formulas defined below is similar to the class of propositional formulas, but a little bit more general: we will allow atomic parts of a formula to be equalities of the

[1]A monkey wants to get a bunch of bananas hanging from the ceiling. He can reach the bananas by first pushing a box to the empty place under the bananas and then climbing on top of the box.

kind found in constraint satisfaction problems. This is convenient when formulas are used to talk about states of a system. For instance, to describe the location of a person in an apartment, we can use equalities like

$$\mathit{Loc}\!=\!\mathit{Kitchen},\ \mathit{Loc}\!=\!\mathit{LivingRoom},\ \mathit{Loc}\!=\!\mathit{Bathroom},\ \mathit{Loc}\!=\!\mathit{Bedroom}.$$

The effect of walking to the kitchen can be described by saying that it makes the first of these atomic formulas true, so that the others become false.

A *(multi-valued propositional) signature* is a set σ of symbols called *constants*, along with a nonempty finite set $\mathit{Dom}(c)$ of symbols, disjoint from σ, assigned to each constant c. We call $\mathit{Dom}(c)$ the *domain* of c. An *atom* of a signature σ is an expression of the form $c\!=\!v$ ("the value of c is v") where $c \in \sigma$ and $v \in \mathit{Dom}(c)$. A *formula* of σ is a propositional combination of atoms.

To distinguish formulas of the usual propositional logic from formulas of a multi-valued signature, we will call them "classical."

An *interpretation* of σ is a function that maps every element of σ to an element of its domain. An interpretation I *satisfies* an atom $c\!=\!v$ (symbolically, $I \models c\!=\!v$) if $I(c) = v$. The satisfaction relation is extended from atoms to arbitrary formulas according to the usual truth tables for the propositional connectives.

The following definitions are standard in logic. A *model* of a set X of formulas is an interpretation that satisfies all formulas in X. If X has a model, it is said to be *consistent*, or *satisfiable*. If every model of X satisfies a formula F then we say that X *entails* F and write $X \models F$. Two sets of formulas are *equivalent* to each other if they have the same models.

A *Boolean* constant is one whose domain is the set $\{\mathbf{f}, \mathbf{t}\}$ of truth values. A *Boolean* signature is one whose constants are Boolean. If c is a Boolean constant, we will sometimes use c as shorthand for the atom $c\!=\!\mathbf{t}$. When the syntax and semantics defined above are restricted to Boolean signatures and to formulas that do not contain $\mathbf{f}$, they turn into the usual syntax and semantics of classical propositional formulas. Checking satisfiability (and entailment) for formulas of a multi-valued propositional signature can be easily reduced to the satisfiability problem for the classical case, as discussed in Section 1.6.

Recall that, according to the definition, an atom is an equality whose left-hand side is a constant c, and whose right-hand side is an element of the domain of c. An expression of the form $c\!=\!d$, where both c and d are constants, will be understood as an abbreviation for the disjunction

$$\bigvee_{v \in \mathit{Dom}(c) \cap \mathit{Dom}(d)} (c\!=\!v \wedge d\!=\!v).$$

The symbol $\neq$ will be used to abbreviate the negation of an equality of either kind.

For example, consider the use of a multi-valued signature to describe possible states in the Monkey and Bananas problem. A state can be described by specifying

- the current locations of the monkey, the bananas, and the box,
- whether or not the monkey is on the box, and
- whether or not the monkey has the bananas.

Assume that the possible locations of the monkey, the bananas, and the box are L_1, L_2, L_3. A signature that would allow us to talk about states consists of the constants

$$\mathit{Loc}(x) \qquad (x \in \{\mathit{Monkey}, \mathit{Bananas}, \mathit{Box}\}) \tag{1.3}$$

whose domain is $\{L_1, L_2, L_3\}$, and the Boolean constants

$$\mathit{HasBananas},\ \mathit{OnBox}. \tag{1.4}$$

This signature has $3^3 \cdot 2^2$ interpretations. The interpretations that satisfy

$$\mathit{HasBananas} \supset \mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Bananas})$$

and

$$\mathit{OnBox} \supset \mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box})$$

represent the possible states of the system.

1.2.2 Causal Rules: Syntax

Begin with a multi-valued propositional signature σ. By a *(causal) rule* we mean an expression of the form $F \Leftarrow G$ ("F is caused if G is true"), where F and G are formulas of σ, called the *head* and the *body* of the rule. Rules with the head $\perp$ are called *constraints*. A *causal theory* is a set of causal rules.

The examples below come from the representation of the Monkey and Bananas domain that will be discussed in Section 1.3.2. Since that theory deals with a sequence $s_0, \ldots, s_m$ of states rather than an individual state, the supply of constants used in these rules is richer than the signature introduced in Section 1.2.1. For a fixed nonnegative integer m—the length of "histories" that we want to describe—introduce $m + 1$ copies of the constants in that signature, one corresponding to the each of the states s_i. Each copy is formed by putting a "time stamp" i: in front of each of the constants:

$$i\!:\!\mathit{Loc}(x),\ i\!:\!\mathit{HasBananas},\ i\!:\!\mathit{OnBox} \tag{1.5}$$

($x \in \{\mathit{Monkey}, \mathit{Bananas}, \mathit{Box}\}$; $i \in \{0, \ldots, m\}$). We also need Boolean constants representing the execution of actions:

$$i\!:\!\mathit{Walk}(l) \tag{1.6}$$

(expressing that between states s_i and s_{i+1} the monkey walks to location l),

$$i\!:\!\mathit{PushBox}(l) \tag{1.7}$$

(the monkey pushes the box to location l), and

$$i\!:\!\textit{ClimbOn},\ i\!:\!\textit{ClimbOff},\ i\!:\!\textit{GraspBananas}, \tag{1.8}$$

where $l \in \{L_1, L_2, L_3\}$ and $i \in \{0, \ldots, m-1\}$.[2] Intuitively, a function mapping constants (1.6)–(1.8) to truth values is understood as the execution of all actions that are mapped to **t**; the actions with the same time stamp i are executed concurrently.

The effects of actions can be described by rules such as

$$\begin{array}{l} i\!+\!1\!:\!\textit{Loc}(\textit{Monkey})\!=\!l \Leftarrow i\!:\!\textit{Walk}(l), \\ i\!+\!1\!:\!\textit{HasBananas} \Leftarrow i\!:\!\textit{GraspBananas}. \end{array} \tag{1.9}$$

For instance, the last rule says that there is a cause for the monkey to have the bananas in state s_{i+1} if he grasped the bananas between states s_i and s_{i+1}.

Restrictions on the executability of actions can be described by constraints, for instance:

$$\begin{array}{l} \bot \Leftarrow i\!:\!(\textit{GraspBananas} \wedge \neg \textit{OnBox}), \\ \bot \Leftarrow i\!:\!(\textit{GraspBananas} \wedge \textit{Loc}(\textit{Monkey}) \!\neq\! \textit{Loc}(\textit{Bananas})). \end{array} \tag{1.10}$$

In the bodies of the last two rules, we use the following convention:

$$i\!:\!(F \wedge G)$$

stands for

$$(i\!:\!F) \wedge (i\!:\!G)$$

and similarly for the other propositional connectives.

1.2.3 Causal Rules: Semantics

Now we will show how to extend the concept of a model, defined in Section 1.2.1 for sets of formulas, to sets of causal rules.

Let T be a causal theory, and let I be an interpretation of its signature. The *reduct* T^I of T relative to I is the set of the heads of all rules in T whose bodies are satisfied by I. We say that I is a *model* of T if I is the unique model of T^I.

Intuitively, T^I is the set of formulas that are caused, according to the rules of T, under interpretation I. If this set has no models or more than one model, then, according to the definition above, I is not considered a model of T. If T^I has exactly one model, but that model is different from I, then I is not a model of T either. The only case when I is a model of T is when I satisfies every formula in the reduct, and no other interpretation does.

[2]The action of climbing off the box is not required to solve the traditional form of the Monkey and Bananas problem, but is included in the version of the domain discussed here. It is needed, for instance, if the monkey wants to bring the bananas back to his original location.

If a causal theory T has a model, we say that it is *consistent*, or *satisfiable*. If every model of T satisfies a formula F then we say that T *entails* F and write $T \models F$.

As an example, take

$$\sigma = \{c\},\ \mathit{Dom}(c) = \{1, \ldots, n\}$$

for some positive integer n, and let the only rule of T be

$$c{=}1 \Leftarrow c{=}1. \tag{1.11}$$

The interpretation I defined by $I(c) = 1$ is a model of T. Indeed,

$$T^I = \{c{=}1\},$$

so that I is the only model of T^I. Furthermore, T has no other models. Indeed, for any interpretation J such that $J(c) \neq 1$, T^J is empty, and I is a model of T^J different from J.

It follows that causal theory (1.11) entails $c{=}1$.

Consider now what happens if we add the rule

$$c{=}2 \Leftarrow \top \tag{1.12}$$

to this theory. The reduct of the extended theory relative to any interpretation includes the atom $c{=}2$. Consequently, the interpretation assigning 2 to c is the only possible model of the extended theory. It is easy to see that this is indeed a model.

The extended theory does not entail $c{=}1$; it entails $c{=}2$. This example shows that the logic introduced above is nonmonotonic. Intuitively, rule (1.11) expresses that 1 is “the default value” of c, and rule (1.12) overrides this default.

If the rule

$$c{=}2 \Leftarrow c{=}2 \tag{1.13}$$

is added to (1.11) instead of (1.12), we will get a causal theory with two models. This theory entails $c{=}1 \vee c{=}2$.

Let us apply the definition of a model to the causal theory T with rules (1.2), assuming that the Boolean constants p, q are the only elements of the underlying signature. Consider, one by one, all interpretations of that signature:

- $I_1(p) = I_1(q) =$ **t**. The reduct consists of the heads of the first two rules of (1.2): $T^{I_1} = \{p, q\}$. Since I_1 is the unique model of T^{I_1}, it is a model of T.
- $I_2(p) =$ **f**, $I_2(q) =$ **t**. The reduct is the same as above, and I_2 is not a model of the reduct. Consequently, I_2 is not a model of T.
- $I_3(p) =$ **t**, $I_3(q) =$ **f**. The only element of the reduct is the head of the third rule of (1.2): $T^{I_3} = \{\neg q\}$. It has 2 models. Consequently, I_3 is not a model of T.

- $I_4(p) = I_4(q) = \mathbf{f}$. The reduct is the same as above, so that I_4 is not a model of T either.

We see that I_1 is the only model of T.

In the discussion of each of the examples above, it was essential that the underlying signature does not include any constants that do not occur in the rules. Adding a constant to the signature of a causal theory without adding any rules containing that constant in the head would render the theory inconsistent (unless the domain of the new constant was a singleton).

The following proposition provides an alternative characterization of the semantics of causal theories:

Proposition 1 *An interpretation I is a model of a causal theory T if and only if for every formula F,*

$$I \models F \quad \textit{iff} \quad T^I \models F\,.$$

In particular, we see that in a model I of a causal theory, every fact that obtains is caused, in the sense that every formula satisfied by I is entailed by the set of formulas caused under I according to the rules of the theory. This assertion is a precise form of the principle of universal causation discussed in the introduction.

The following fact relates causal rules to corresponding material implications:

Proposition 2 *If a causal theory T contains a causal rule $F \Leftarrow G$ then T entails $G \supset F$.*

The satisfiability problem for multi-valued propositional formulas clearly belongs to class NP. The semantics of causal theories is apparently more complex:

Proposition 3 *The problem of determining that a finite causal theory is consistent is Σ_2^P-complete.*

In Section 1.2.6 we define a special kind of causal theories for which the satisfiability problem is in class NP.

1.2.4 Equivalent Transformations of Causal Theories

A negated atom $c \neq v$ is equivalent to the disjunction

$$\bigvee_{w \in Dom(c)\setminus\{v\}} c = w.$$

Consequently, any formula of a multi-valued propositional signature can be rewritten in "positive disjunctive normal form"—as a disjunction of conjunctions of atoms. Similarly, any such formula is equivalent to a conjunction of disjunctions of atoms.

Furthermore, if the head of a causal rule is a conjunction, or if its body is a disjunction, then the rule can be broken into simpler rules:

Proposition 4 *(i) Replacing a rule*

$$F \wedge G \Leftarrow H$$

in a causal theory by the rules

$$F \Leftarrow H,\ G \Leftarrow H$$

does not change the set of models. (ii) Replacing a rule

$$F \Leftarrow G \vee H$$

in a causal theory by the rules

$$F \Leftarrow G,\ F \Leftarrow H$$

does not change the set of models.

Taken together, these facts allow us to replace any rule in a causal theory by several rules whose heads are disjunctions of atoms, and whose bodies are conjunctions of atoms.

1.2.5 Constraints

As we saw in Section 1.2.3, adding a rule to a causal theory may affect the theory nonmonotonically—it can get new models. The proposition below shows that the effect of adding constraints (rules with head $\bot$) is monotonic.

We say that an interpretation I *violates* a constraint $\bot \Leftarrow F$ if I satisfies F.

Proposition 5 *Let T_1 and T_2 be causal theories of a signature σ, such that every rule in T_2 is a constraint. An interpretation of σ is a model of $T_1 \cup T_2$ iff it is a model of T_1 and does not violate any of the constraints T_2.*

In other words, the effect of adding a set of constraints to a causal theory is simply to eliminate the models that violate at least one of these constraints.

For instance, the theory whose rules are (1.11) and (1.13) has two models: $I_1(c) = 1$ and $I_2(c) = 2$. The constraint

$$\bot \Leftarrow c{=}2 \vee c{=}3 \tag{1.14}$$

is violated by I_2. Consequently, the theory with rules (1.11), (1.13) and (1.14) has one model, I_1.

Corollary 1 *A causal theory T entails a formula F iff the theory $T \cup \{\bot \Leftarrow F\}$ is inconsistent.*

1.2.6 Definite Theories

A causal theory T is *definite* if

- the head of every rule of T is an atom or $\bot$, and
- no atom is the head of infinitely many rules of T.

For instance, causal theory (1.11) is definite. Causal theory (1.2) is, strictly speaking, not definite, but it can be turned into a definite theory by replacing $\neg q$ in the head of the last rule with the equivalent atom:

$$\begin{array}{l} p \Leftarrow q, \\ q \Leftarrow q, \\ q{=}\mathbf{f} \Leftarrow \neg q. \end{array} \tag{1.15}$$

The "completion" process described below reduces the problem of finding a model of a definite causal theory to the problem of finding a model of a set of formulas.

Take a definite causal theory T of a signature σ. We say that an atom $c = v$ of σ is *trivial* if the domain of c is a singleton. For each nontrivial atom A, the *completion formula* for A is the formula

$$A \equiv G_1 \vee \cdots \vee G_n$$

where $G_1, \ldots, G_n$ ($n \geq 0$) are the bodies of the rules of T with head A. The *completion* of T is obtained by taking the completion formulas for all nontrivial atoms of σ, along with the formula $\neg F$ for each constraint $\bot \Leftarrow F$ in T.

Proposition 6 *The models of a definite causal theory are precisely the models of its completion.*

For instance, the completion of (1.11) is

$$\begin{array}{ll} c{=}1 \equiv c{=}1, & \\ c{=}v \equiv \bot & \quad (v \in \mathit{Dom}(c) \setminus \{1\}) \end{array} \tag{1.16}$$

if $|\mathit{Dom}(c)| > 1$. Otherwise the atom $c = 1$ is trivial, and the completion is empty. In both cases, the only model of the completion is defined by $I(c) = 1$. As discussed in Section 1.2.3, this is the only model of (1.11).

After adding rule (1.12), the completion turns into

$$\begin{array}{ll} c{=}1 \equiv c{=}1, & \\ c{=}2 \equiv \top, & \\ c{=}v \equiv \bot & \quad (v \in \mathit{Dom}(c) \setminus \{1, 2\}). \end{array}$$

The only model of these formulas is defined by $I(c) = 2$.

The completion of (1.15) is

$$\begin{array}{l} p \equiv q, \\ p{=}\mathbf{f} \equiv \bot, \\ q \equiv q, \\ q{=}\mathbf{f} \equiv \neg q. \end{array} \tag{1.17}$$

The last two equivalences are identically true and can be dropped. The only model of the first two equivalences is the interpretation I_1 (mapping both p and q to **t**) that was found in Section 1.2.3 to be the only model of (1.2).

Here are two more examples of the use of completion. First, we will show how to turn any set X of formulas into a causal theory that has the same models as X. The rules of this theory are

- $A \Leftarrow A$ for every nontrivial atom A, and
- the constraints $\bot \Leftarrow \neg F$ for every $F \in X$.

The completion of this theory consists of the formulas $A \equiv A$ for nontrivial atoms A and the formulas $\neg\neg F$ for all $F \in X$. Clearly, the completion is equivalent to X.

Second, definite theories can be used to express the "closed-world assumption," as follows. Take a Boolean signature σ. The assumption that the elements of σ are false by default can be expressed by the rules

$$\neg c \Leftarrow \neg c \qquad (c \in \sigma) \tag{1.18}$$

(if c is false then there is a cause for this). If, for some subset S of σ, we combine (1.18) with the rules

$$c \Leftarrow \top \qquad (c \in S),$$

we will get a causal theory whose only model is the interpretation I that maps the constants in S to **t** and all other constants to **f**. Indeed, the completion of this theory consists of the formulas

$$\begin{array}{ll} c \equiv \top & (c \in S), \\ c \equiv \bot & (c \in \sigma \setminus S), \\ c{=}\mathbf{f} \equiv \neg c & (c \in \sigma), \end{array}$$

and I is the only model of these formulas.

The assertion of Proposition 6 would be incorrect if we did not restrict the completion process to nontrivial atoms. Consider, for instance, the causal theory whose signature consists of one constant c with the domain $\{0\}$, and whose set of rules is empty. If the definition of completion were extended to trivial atoms then the completion of this theory would be $c{=}0 \equiv \bot$, which is inconsistent.

Proposition 6 shows that the satisfiability problem for finite definite causal theories belongs to class NP. It is clearly NP-complete.

1.3 Actions and Change

1.3.1 A Very Simple Example

Before presenting a formalization of the Monkey and Bananas domain in causal logic, we consider a much simpler domain: a system whose state is determined by one truth-valued parameter p. The only available action a changes the value of this parameter to **t**.

Assume first that we are only interested in histories of length 1, that is to say, in pairs of successive states s_0, s_1. In the spirit of the examples from Section 1.2.2, the system under consideration can be described by a causal theory using the Boolean constants $0:p$, $1:p$ and $0:a$. The effect of executing the action is described by the causal rule

$$1:p \Leftarrow 0:a. \tag{1.19}$$

Rule (1.19) by itself does not give an adequate description of the system, because it does not tell us

(i) how to determine the value of p in the initial state s_0,

(ii) how to determine whether action a is executed,

(iii) how to determine the value of p in the final state s_1 if a is not executed.

The answer to (i) is that the initial state of the system is arbitrary. We will exempt $0:p$ from the principle of universal causation by rules similar to the last two rules of (1.2):

$$\begin{aligned} 0:p &\Leftarrow 0:p, \\ 0:\neg p &\Leftarrow 0:\neg p. \end{aligned} \tag{1.20}$$

Whichever causes determine the initial state of the system, they are outside the theory; $0:p$ is "exogenous." (According to the convention introduced at the end of Section 1.2.2, $0:\neg p$ stands for $\neg 0:p$.)

The answer to (ii) is similar: whichever causes determine whether or not the action is executed, they are outside the theory; $0:a$ is exogenous as well:

$$\begin{aligned} 0:a &\Leftarrow 0:a, \\ 0:\neg a &\Leftarrow 0:\neg a. \end{aligned} \tag{1.21}$$

The answer to (iii) is that, when action a is not executed, the value of p in state s_1 is determined by "commonsense inertia"—it is the same as in state s_0. This idea can be expressed by the rules

$$\begin{aligned} 1:p &\Leftarrow (0:p) \wedge (1:p), \\ 1:\neg p &\Leftarrow (0:\neg p) \wedge (1:\neg p). \end{aligned} \tag{1.22}$$

The first rule says that if the value of p is **t** both in state s_0 and in state s_1 then there is a cause for it to be **t** in state s_1. Intuitively, inertia is the cause. (To put it differently, this rule says that p is true by default in state s_1 if it is true in state s_0.) The second rule expresses a similar condition on the value **f**. Rules (1.22) illustrate the solution to the frame problem adopted in our framework.

To find models of causal theory (1.19)–(1.22), we compute the completion of this theory and translate it into classical propositional logic, as this is done for theory (1.2) in Section 1.2.6 and Section 1.6. The result is

$$\begin{array}{l}
0\!:\!p \equiv 0\!:\!p, \\
\neg\, 0\!:\!p \equiv \neg\, 0\!:\!p, \\
1\!:\!p \equiv 0\!:\!a \vee (0\!:\!p \wedge 1\!:\!p), \\
\neg\, 1\!:\!p \equiv \neg\, 0\!:\!p \wedge \neg\, 1\!:\!p, \\
0\!:\!a \equiv 0\!:\!a, \\
\neg\, 0\!:\!a \equiv \neg\, 0\!:\!a.
\end{array}$$

The first two lines and the last two lines are tautologies. The third line can be rewritten as the conjunction of two implications:

$$\begin{array}{l}
1\!:\!p \supset 0\!:\!a \vee 0\!:\!p, \\
0\!:\!a \supset 1\!:\!p.
\end{array}$$

The fourth line is equivalent to

$$0\!:\!p \supset 1\!:\!p.$$

The conjunction of these three implications can be rewritten in the form of an explicit definition of $1\!:\!p$ in terms of $0\!:\!a$ and $0\!:\!p$:

$$1\!:\!p \equiv 0\!:\!a \vee 0\!:\!p.$$

(The end value of p is **t** iff action a is executed or the initial value of p is **t**.) This calculation shows that causal theory (1.19)–(1.22) has 4 models, corresponding to the possible combinations of the values of $0\!:\!p$ and $0\!:\!a$.

To get a causal theory whose models correspond to the histories of the same simple domain whose length is m ($m \geq 0$), we introduce Boolean constants $i\!:\!p$ for $i = 0, \ldots, m$ and $i : a$ for $i = 0, \ldots, m-1$. The value of $i : p$ characterizes state s_i—it gives the value of the parameter p in that state. The value of $i : a$ characterizes the event occurring between states s_i and s_{i+1}—it tells us whether that event included the execution of action a. Rules (1.19)–(1.22) are generalized as follows:

$$\begin{array}{l}
i{+}1\!:\!p \Leftarrow i\!:\!a, \\
0\!:\!p \Leftarrow 0\!:\!p, \\
0\!:\!\neg p \Leftarrow 0\!:\!\neg p, \\
i\!:\!a \Leftarrow i\!:\!a, \\
i\!:\!\neg a \Leftarrow i\!:\!\neg a, \\
i{+}1\!:\!p \Leftarrow (i\!:\!p) \wedge (i{+}1\!:\!p), \\
i{+}1\!:\!\neg p \Leftarrow (i\!:\!\neg p) \wedge (i{+}1\!:\!\neg p)
\end{array} \qquad (1.23)$$

($i = 0, \dots, m-1$). Call this causal theory *SD*$_m$ (for “simple domain, histories of length m”). For instance, *SD*$_0$ consists of just two rules

$$\begin{array}{l} 0{:}p \Leftarrow 0{:}p, \\ 0{:}\neg p \Leftarrow 0{:}\neg p. \end{array} \tag{1.24}$$

Theory *SD*$_0$ has two models, corresponding to the possible states of the system. Theory *SD*$_1$ is identical to (1.19)–(1.22). For any m, the completion of *SD*$_m$ can be rewritten as m equivalences

$$i{+}1 : p \equiv (i{:}a) \vee (i{:}p) \qquad (0 \leq i < m).$$

It follows that *SD*$_m$ has 2^{m+1} models, each characterized by the truth values assigned to the constants $0{:}p$ and $i{:}a$ ($i = 0, \dots, m-1$).

1.3.2 Monkey and Bananas

We will now describe a causal theory *MB*$_m$ ($m \geq 0$) whose models represent all possible histories of length m in the Monkey and Bananas domain. The signature of this theory is listed in (1.5)–(1.8). Recall that it consists of symbols (1.3) and (1.4) prefixed by $i:$ and symbols of the forms

$$\mathit{Walk}(l),\ \mathit{PushBox}(l),\ \mathit{ClimbOn},\ \mathit{ClimbOff},\ \mathit{GraspBananas} \tag{1.25}$$

prefixed by $i{:}$ with $i < m$. Here and below, l ranges over $\{L_1, L_2, L_3\}$, and i ranges over $\{0, \dots, m\}$.

As discussed in Section 1.2.1, the possible states of the Monkey and Bananas system are described by the interpretations of constants (1.3), (1.4) that satisfy two conditions: if the monkey has the bananas then the monkey and the bananas are at the same location; if the monkey is on the box then the monkey and the box are at the same location. In the causal theory *MB*$_m$ we postulate the following causal counterparts of these conditions. If the monkey has the bananas then there is a cause for the bananas to be at the same place where the monkey is:

$$i{:}\mathit{Loc}(\mathit{Bananas}){=}l \Leftarrow i{:}(\mathit{HasBananas} \wedge \mathit{Loc}(\mathit{Monkey}){=}l). \tag{1.26}$$

If the monkey is on the box then there is a cause for the monkey to be at the same place where the box is:

$$i{:}\mathit{Loc}(\mathit{Monkey}){=}l \Leftarrow i{:}(\mathit{OnBox} \wedge \mathit{Loc}(\mathit{Box}){=}l). \tag{1.27}$$

We will see soon why including these rules is convenient.

The next group of rules describes the effects and preconditions of walking:

$$\begin{array}{l} i{+}1{:}\mathit{Loc}(\mathit{Monkey}){=}l \Leftarrow i{:}\mathit{Walk}(l), \\ \bot \Leftarrow i{:}(\mathit{Walk}(l) \wedge \mathit{Loc}(\mathit{Monkey}){=}l), \\ \bot \Leftarrow i{:}(\mathit{Walk}(l) \wedge \mathit{OnBox}) \end{array} \tag{1.28}$$

($i < m$). We have seen the first of these rules in Section 1.2.2. The other two rules are similar to constraints (1.10).

The effect of walking on the location of the monkey is not the only possible effect of this action: if the monkey has the bananas then walking will affect the location of the bananas as well. This second effect of walking can be described by the rules

$$i{+}1 : \mathit{Loc}(\mathit{Bananas}) = l \Leftarrow i : (\mathit{HasBananas} \wedge \mathit{Walk}(l)).$$

But we will not include these rules, because they would be redundant: in the presence of (1.26) the change in the location of the bananas is an indirect effect, or "ramification," of walking (and of any other action that affects the location of the monkey). The possibility of this simplification is what makes (1.26) an attractive postulate. Similarly, in the presence of (1.27), a change in the location of the monkey is an indirect effect of any action that affects the location of the box, if it is executed when the monkey is on the box. (Scenarios like this are possible in the enhanced version of the Monkey and Bananas domain described in Section 1.3.3.)

Pushing the box has two effects and three preconditions:

$$\begin{array}{l} i{+}1 : \mathit{Loc}(\mathit{Box}) = l \Leftarrow i : \mathit{PushBox}(l), \\ i{+}1 : \mathit{Loc}(\mathit{Monkey}) = l \Leftarrow i : \mathit{PushBox}(l), \\ \bot \Leftarrow i : (\mathit{PushBox}(l) \wedge \mathit{Loc}(\mathit{Monkey}) = l), \\ \bot \Leftarrow i : (\mathit{PushBox}(l) \wedge \mathit{OnBox}), \\ \bot \Leftarrow i : (\mathit{PushBox}(l) \wedge \mathit{Loc}(\mathit{Monkey}) \neq \mathit{Loc}(\mathit{Box})) \end{array} \tag{1.29}$$

($i < m$). The descriptions of the other actions have a similar structure:

$$\begin{array}{l} i{+}1 : \mathit{OnBox} \Leftarrow i : \mathit{ClimbOn}, \\ \bot \Leftarrow i : (\mathit{ClimbOn} \wedge \mathit{OnBox}), \\ \bot \Leftarrow i : (\mathit{ClimbOn} \wedge \mathit{Loc}(\mathit{Monkey}) \neq \mathit{Loc}(\mathit{Box})), \\ \\ i{+}1 : \neg \mathit{OnBox} \Leftarrow i : \mathit{ClimbOff}, \\ \bot \Leftarrow i : (\mathit{ClimbOff} \wedge \neg \mathit{OnBox}), \\ \\ i{+}1 : \mathit{HasBananas} \Leftarrow i : \mathit{GraspBananas}, \\ \bot \Leftarrow i : (\mathit{GraspBananas} \wedge \mathit{HasBananas}), \\ \bot \Leftarrow i : (\mathit{GraspBananas} \wedge \neg \mathit{OnBox}), \\ \bot \Leftarrow i : (\mathit{GraspBananas} \wedge \mathit{Loc}(\mathit{Monkey}) \neq \mathit{Loc}(\mathit{Bananas})) \end{array} \tag{1.30}$$

($i < m$). Some of these rules are familiar to us from Section 1.2.2.

The next group of rules expresses that some combinations of actions cannot be executed concurrently:

$$\begin{array}{l} \bot \Leftarrow i : (\mathit{Walk}(l) \wedge \mathit{PushBox}(l)), \\ \bot \Leftarrow i : (\mathit{Walk}(l) \wedge \mathit{ClimbOn}), \\ \bot \Leftarrow i : (\mathit{PushBox}(l) \wedge \mathit{ClimbOn}), \\ \bot \Leftarrow i : (\mathit{ClimbOff} \wedge \mathit{GraspBananas}) \end{array} \tag{1.31}$$

$(i < m)$.[3]

To complete the list of postulates, we need to add

- rules expressing that the initial state is exogenous, which are similar to (1.20),
- rules expressing that the execution of actions is exogenous, which are similar to (1.21), and
- inertia rules, which are similar to (1.22).

To say that the initial state is exogenous, we postulate

$$0\!:\!c\!=\!v \Leftarrow 0\!:\!c\!=\!v, \tag{1.32}$$

where c is any of the symbols (1.3), (1.4), and $v \in \mathit{Dom}(0\!:\!c)$. To say that the execution of actions is exogenous, we postulate

$$i\!:\!c\!=\!v \Leftarrow i\!:\!c\!=\!v, \tag{1.33}$$

where c is any of the symbols (1.25), $i < m$, and $v \in \{\mathbf{f}, \mathbf{t}\}$. The inertia rules are

$$i\!+\!1\!:\!c\!=\!v \Leftarrow (i\!:\!c\!=\!v) \wedge (i\!+\!1\!:\!c\!=\!v), \tag{1.34}$$

where c is any of the symbols (1.3), (1.4); $i < m$; $v \in \mathit{Dom}(i\!:\!c)$.

We define MB_m as the causal theory whose rules are (1.26)–(1.34). The models of this theory correspond to the possible histories of the Monkey and Bananas domain of length m.

1.3.3 Concurrent Actions and Multiple Agents

Causal theories can be used to describe the concurrent execution of actions by several agents. Consider, for instance, the enhancement of the Monkey and Bananas domain that involves several monkeys. Each monkey can walk, push the box, climb on and off, and grasp the bananas, except that two monkeys cannot both have the bananas simultaneously, or be on top of the box simultaneously.

In the modification of MB_m described below, α and β range over a fixed finite nonempty set M of symbols—the names of the monkeys. The signature consists of the symbols

$$i\!:\!\mathit{Loc}(x),\ i\!:\!\mathit{HasBananas}(\alpha),\ i\!:\!\mathit{OnBox}(\alpha) \tag{1.35}$$

[3]Equivalently, we could prohibit the concurrent execution of *all* possible pairs of actions. For the pairs of actions not included in (1.31), the fact that their concurrent execution is impossible follows from the other postulates. For instance, actions $\mathit{Walk}(l)$ and $\mathit{Walk}(l_1)$ for $l \neq l_1$ cannot be executed at the same time because their effects are incompatible. Actions $\mathit{Walk}(l)$ and $\mathit{ClimbOff}$ cannot be executed at the same time because their preconditions are incompatible.

where $x \in M \cup \{\mathit{Bananas}, \mathit{Box}\}$, and the symbols

$$\begin{gathered} i\!:\!\mathit{Walk}(\alpha, l),\ i\!:\!\mathit{PushBox}(\alpha, l), \\ i\!:\!\mathit{ClimbOn}(\alpha),\ i\!:\!\mathit{ClimbOff}(\alpha),\ i\!:\!\mathit{GraspBananas}(\alpha) \end{gathered} \tag{1.36}$$

for $i < m$. As before, the domain of $i : \mathit{Loc}(x)$ is $\{L_1, L_2, L_3\}$; the other constants are Boolean.

The first group of rules consists of generalizations of (1.26) and (1.27)

$$\begin{aligned} &i\!:\!\mathit{Loc}(\mathit{Bananas})\!=\!l \Leftarrow i\!:\!(\mathit{HasBananas}(\alpha) \wedge \mathit{Loc}(\alpha)\!=\!l), \\ &i\!:\!\mathit{Loc}(\alpha)\!=\!l \Leftarrow i\!:\!(\mathit{OnBox}(\alpha) \wedge \mathit{Loc}(\mathit{Box})\!=\!l) \end{aligned}$$

and the new constraints

$$\bot \Leftarrow i\!:\!(\mathit{HasBananas}(\alpha) \wedge \mathit{HasBananas}(\beta)), \tag{1.37}$$

$$\bot \Leftarrow i\!:\!(\mathit{OnBox}(\alpha) \wedge \mathit{OnBox}(\beta)) \tag{1.38}$$

($\alpha \neq \beta$). Next, we generalize rules (1.28)–(1.31):

$$\begin{aligned} &i\!+\!1\!:\!\mathit{Loc}(\alpha)\!=\!l \Leftarrow i\!:\!\mathit{Walk}(\alpha, l), \\ &\bot \Leftarrow i\!:\!(\mathit{Walk}(\alpha, l) \wedge \mathit{Loc}(\alpha)\!=\!l), \\ &\bot \Leftarrow i\!:\!(\mathit{Walk}(\alpha, l) \wedge \mathit{OnBox}(\alpha)), \end{aligned}$$

$$\begin{aligned} &i\!+\!1\!:\!\mathit{Loc}(\mathit{Box})\!=\!l \Leftarrow i\!:\!\mathit{PushBox}(\alpha, l), \\ &i\!+\!1\!:\!\mathit{Loc}(\alpha)\!=\!l \Leftarrow i\!:\!\mathit{PushBox}(\alpha, l), \\ &\bot \Leftarrow i\!:\!(\mathit{PushBox}(\alpha, l) \wedge \mathit{Loc}(\alpha)\!=\!l), \\ &\bot \Leftarrow i\!:\!(\mathit{PushBox}(\alpha, l) \wedge \mathit{OnBox}(\alpha)), \\ &\bot \Leftarrow i\!:\!(\mathit{PushBox}(\alpha, l) \wedge \mathit{Loc}(\alpha)\!\neq\!\mathit{Loc}(\mathit{Box})), \end{aligned}$$

$$\begin{aligned} &i\!+\!1\!:\!\mathit{OnBox}(\alpha) \Leftarrow i\!:\!\mathit{ClimbOn}(\alpha), \\ &\bot \Leftarrow i\!:\!(\mathit{ClimbOn}(\alpha) \wedge \mathit{OnBox}(\alpha)), \\ &\bot \Leftarrow i\!:\!(\mathit{ClimbOn}(\alpha) \wedge \mathit{Loc}(\alpha)\!\neq\!\mathit{Loc}(\mathit{Box})), \end{aligned}$$

$$\begin{aligned} &i\!+\!1\!:\!\neg\mathit{OnBox}(\alpha) \Leftarrow i\!:\!\mathit{ClimbOff}(\alpha), \\ &\bot \Leftarrow i\!:\!(\mathit{ClimbOff}(\alpha) \wedge \neg\mathit{OnBox}(\alpha)), \end{aligned}$$

$$\begin{aligned} &i\!+\!1\!:\!\mathit{HasBananas}(\alpha) \Leftarrow i\!:\!\mathit{GraspBananas}(\alpha), \\ &\bot \Leftarrow i\!:\!(\mathit{GraspBananas}(\alpha) \wedge \mathit{HasBananas}(\beta)), \\ &\bot \Leftarrow i\!:\!(\mathit{GraspBananas}(\alpha) \wedge \neg\mathit{OnBox}(\alpha)), \\ &\bot \Leftarrow i\!:\!(\mathit{GraspBananas}(\alpha) \wedge \mathit{Loc}(\alpha)\!\neq\!\mathit{Loc}(\mathit{Bananas})), \end{aligned}$$

$$\begin{aligned} &\bot \Leftarrow i\!:\!(\mathit{Walk}(\alpha, l) \wedge \mathit{PushBox}(\alpha, l)), \\ &\bot \Leftarrow i\!:\!(\mathit{Walk}(\alpha, l) \wedge \mathit{ClimbOn}(\alpha)), \\ &\bot \Leftarrow i\!:\!(\mathit{PushBox}(\alpha, l) \wedge \mathit{ClimbOn}(\alpha)), \\ &\bot \Leftarrow i\!:\!(\mathit{ClimbOff}(\alpha) \wedge \mathit{GraspBananas}(\alpha)) \end{aligned}$$

($i < m$). Finally, we include rules (1.32) and (1.34) for c of any of the forms

$$\mathit{Loc}(x),\ \mathit{HasBananas}(\alpha),\ \mathit{OnBox}(\alpha)$$

and rule (1.33) for c of any of the forms

$$\mathit{Walk}(\alpha, l),\ \mathit{PushBox}(\alpha, l),\ \mathit{ClimbOn}(\alpha),\ \mathit{ClimbOff}(\alpha),\ \mathit{GraspBananas}(\alpha).$$

This causal theory MB_m^M entails, for instance, that two monkeys cannot climb the box simultaneously:

$$\neg\, i : (\mathit{ClimbOn}(\alpha) \wedge \mathit{ClimbOn}(\beta))$$

($i < m, \alpha \neq \beta$). To prove this fact, note that, by Proposition 2, MB_m^M entails each of the formulas

$$\begin{array}{l}(i : \mathit{ClimbOn}(\alpha)) \supset (i+1 : \mathit{OnBox}(\alpha)),\\ (i : \mathit{ClimbOn}(\beta)) \supset (i+1 : \mathit{OnBox}(\beta)),\\ \neg((i+1 : \mathit{OnBox}(\alpha)) \wedge (i+1 : \mathit{OnBox}(\beta))).\end{array}$$

If M is a singleton then MB_m^M is essentially identical to MB_m.

1.3.4 Making Rules Defeasible

Using auxiliary constants, we can make causal rules "defeasible." Consider, for instance, the constraint (1.38) in MB_m^M that prohibits more than one monkey on top of the box at the same time. We can make this constraint defeasible by rewriting it as

$$\bot \Leftarrow i : (\mathit{OnBox}(\alpha) \wedge \mathit{OnBox}(\beta) \wedge \neg \mathit{Ab}_1(\alpha, \beta)), \tag{1.39}$$

where $i : \mathit{Ab}_1(\alpha, \beta)$ are "abnormality constants"—new Boolean constants assumed to be false by default:

$$i : \neg \mathit{Ab}_1(\alpha, \beta) \Leftarrow i : \neg \mathit{Ab}_1(\alpha, \beta). \tag{1.40}$$

(Rule (1.40) is similar to the formalization (1.18) of the closed-world assumption.) The causal theory obtained from MB_m^M by replacing (1.38) with (1.39) and (1.40) entails the conjunctive terms $\neg \mathit{Ab}_1(\alpha, \beta)$ that were added to the body of (1.38). In this sense, including these terms did not make (1.38) weaker. But the limitations expressed by the modified rule can be retracted by adopting additional postulates. Imagine, for instance, that David is a baby monkey, so small that there is enough room for him on top of the box even when another monkey is there. We can express this additional information by adding the rules

$$\begin{array}{l} i : \mathit{Ab}_1(\mathit{David}, \alpha) \Leftarrow \top,\\ i : \mathit{Ab}_1(\alpha, \mathit{David}) \Leftarrow \top.\end{array} \tag{1.41}$$

The extended theory entails $i : \neg \mathit{Ab}_1(\alpha, \beta)$ only when $\alpha, \beta \neq \mathit{David}$.

As another example, consider the rules describing the effect of pushing the box:

$$\begin{array}{c} i{+}1 : \mathit{Loc}(\mathit{Box}) = l \;\Leftarrow\; i : \mathit{PushBox}(\alpha, l), \\ i{+}1 : \mathit{Loc}(\alpha) = l \;\Leftarrow\; i : \mathit{PushBox}(\alpha, l). \end{array} \tag{1.42}$$

We can make them defeasible by rewriting them as

$$\begin{array}{c} i{+}1 : \mathit{Loc}(\mathit{Box}) = l \;\Leftarrow\; i : (\mathit{PushBox}(\alpha, l) \wedge \neg \mathit{Ab}_2(\alpha)), \\ i{+}1 : \mathit{Loc}(\alpha) = l \;\Leftarrow\; i : (\mathit{PushBox}(\alpha, l) \wedge \neg \mathit{Ab}_2(\alpha)), \end{array} \tag{1.43}$$

where $i : \mathit{Ab}_2(\alpha)$ for $i < m$ are new Boolean constants, and adding the rules

$$i : \neg \mathit{Ab}_2(\alpha) \;\Leftarrow\; i : \neg \mathit{Ab}_2(\alpha). \tag{1.44}$$

The modification (1.43) of rules (1.42) will be convenient if we decide later to enhance the description of the domain by allowing the action *PushBox* to be unsuccessful. Imagine that one of the monkeys, Goliath, is so heavy that the other monkeys cannot move the box while Goliath sits on it. We can retract this special case of (1.43) by postulating

$$i : \mathit{Ab}_2(\alpha) \;\Leftarrow\; i : \mathit{OnBox}(\mathit{Goliath}). \tag{1.45}$$

Adding this rule does not make the action $\mathit{PushBox}(\alpha, l)$ nonexecutable when Goliath is on the box; it only retracts the assumption about the effect of pushing the box on the locations of the box and the monkey in this special case. The box with Goliath on top will stay, by inertia, where it was, and so will the monkey. This example shows how assumptions about effects of actions can be qualified by adding more postulates.

What if Goliath is trying to climb the box while α is trying to push it? To express that α does not succeed in this case either, we postulate

$$i : \mathit{Ab}_2(\alpha) \;\Leftarrow\; i : \mathit{ClimbOn}(\mathit{Goliath}). \tag{1.46}$$

In the next state, Goliath will be on top of the box, but the location of α and the location of the box will not change.

1.4 Language $\mathcal{C}$+

1.4.1 Action Description Languages

Let us return to the causal theories SD_m describing a simple action domain in Section 1.3.1. Models of these theories can be visualized as paths in a "transition system"—the graph shown in Figure 1.1. The two vertices of the graph represent states; in one state, the value of the parameter p is **f**, in the other it is **t**. The edges represent transitions between states; the action a is executed in two transitions, and is not executed in the other two.

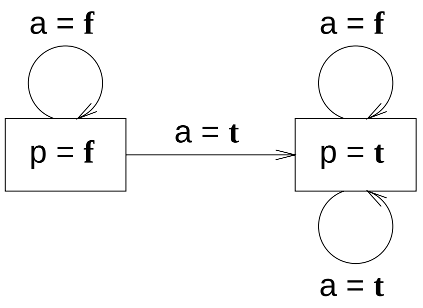


Figure 1.1: The transition system corresponding to the causal theories *SD*$_m$.

There is a simple 1–1 correspondence between the models of *SD*$_m$ and the paths of length m in this transition system. For instance, the model I of *SD*$_2$ defined by

$$I(0\!:\!p) = \mathbf{f},\ I(0\!:\!a) = \mathbf{f},\ I(1\!:\!p) = \mathbf{f},\ I(1\!:\!a) = \mathbf{t},\ I(2\!:\!p) = \mathbf{t} \qquad (1.47)$$

corresponds to the path

$$\langle p = \mathbf{f},\ a = \mathbf{f},\ p = \mathbf{f},\ a = \mathbf{t},\ p = \mathbf{t}\rangle$$

(start at the vertex $p = \mathbf{f}$, follow the loop labeled $a = \mathbf{f}$, and then follow the edge labeled $a = \mathbf{t}$ to the vertex $p = \mathbf{t}$). The models of *SD*$_0$ correspond to the vertices of the transition system, and the models of *SD*$_1$ correspond to its edges.

The sequence of causal theories *MB*$_m$ (Section 1.3.2) can be represented by a graph also. The models of *MB*$_0$ correspond to the vertices of that graph, and the models of *MB*$_1$ correspond to its edges. More generally, the models of *MB*$_m$ correspond to the paths of length m. The same is true for the generalization *MB*$_m^M$ of this sequence, where M is a fixed set of symbols (Section 1.3.3), and for the modifications of *MB*$_m^M$ introduced in Section 1.3.4.

Action description languages are formal languages for describing transition systems. We will introduce here an action description language, called $\mathcal{C}$+, and define how to turn any action description D in this language into an infinite sequence $D_0, D_1, \dots$ of causal theories. We will see there is an action description in $\mathcal{C}$+, called *SD*, that corresponds to the sequence *SD*$_m$ defined in Section 1.3.1, and that there is also an action description *MB* that corresponds to the sequence *MB*$_m$ from Section 1.3.2. The generalizations and modifications of this sequence introduced in Sections 1.3.3 and 1.3.4 can be represented in $\mathcal{C}$+ as well. We will show that for any action description D, the theories D_0 and D_1 characterize the vertices and edges of a transition system such that there is a 1–1 correspondence between the paths of length m in this system and the models of D_m.

The "higher-level" notation of $\mathcal{C}$+ is more concise than the notation of causal theories. This fact, along with the availability of a transition system semantics, makes $\mathcal{C}$+ an attractive formalism for describing actions.

1.4.2 From Action Descriptions to Causal Theories

Begin with a multi-valued signature (see Section 1.2.1) partitioned into *fluent* constants and *action* constants. The fluent constants are assumed to be further partitioned into *simple* and *statically determined.*

Intuitively, fluent constants denote "fluents," or parameters characterizing a state. Action constants denote parameters characterizing an event leading from one state to another. For instance, in the action description *SD* introduced below, p is a simple fluent constant, and a is an action constant. Note that action constants are not required to be Boolean; one possible use of non-Boolean action constants is discussed in Section 1.5.6. The need to distinguish between simple and statically determined fluent constants is discussed in Section 1.4.4.

A *fluent formula* is a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{1.48}$$

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (1.48) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{1.49}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law.

Fluent dynamic laws (1.49) are the most important element of the language, because they can be used for describing direct effects of actions. If c is a Boolean action constant, we express that F is an effect of executing c by the fluent dynamic law

$$\textbf{caused } F \textbf{ if } \top \textbf{ after } c,$$

which can be abbreviated as

$$c \textbf{ causes } F.$$

Static laws can be used to talk about causal dependencies between fluents in the same state, and action dynamic laws can express causal dependencies between concurrently executed actions.

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

The formula F in a causal law (1.48) or (1.49) is called the *head.* Note that statically determined constants are allowed in the heads of static laws, but not in the heads of dynamic laws. This explains the term "statically determined."

For any action description D and any nonnegative integer m, the causal theory D_m is defined as follows. The signature of D_m consists of the pairs $i\!:\!c$ such that

- $i \in \{0, \dots, m\}$ and c is a fluent constant of D, or
- $i \in \{0, \dots, m-1\}$ and c is an action constant of D.

The domain of $i\!:\!c$ is the same as the domain of c. By $i\!:\!F$ we denote the result of inserting $i\!:$ in front of every occurrence of every constant in a formula F, and similarly for a set of formulas. The rules of D_m are:

$$i\!:\!F \Leftarrow i\!:\!G \tag{1.50}$$

for every static law (1.48) in D and every $i \in \{0, \dots, m\}$, and for every action dynamic law (1.48) in D and every $i \in \{0, \dots, m-1\}$;

$$i{+}1\!:\!F \Leftarrow (i{+}1\!:\!G) \land (i\!:\!H) \tag{1.51}$$

for every fluent dynamic law (1.49) in D and every $i \in \{0, \dots, m-1\}$;

$$0\!:\!c{=}v \Leftarrow 0\!:\!c{=}v \tag{1.52}$$

for every simple fluent constant c and every $v \in \mathit{Dom}(c)$.

Note that the definition of D_m treats simple fluent constants and statically determined fluent constants in different ways: rules (1.52) are included only when c is simple.

In the example (1.53) below, the following abbreviations are used. If c is an action constant, the expression

$$\mathbf{exogenous}\ c$$

stands for the action dynamic laws

$$\mathbf{caused}\ c{=}v\ \mathbf{if}\ c{=}v$$

for all $v \in \mathit{Dom}(c)$. If c is a simple fluent constant, the expression

$$\mathbf{inertial}\ c$$

stands for the fluent dynamic laws

$$\mathbf{caused}\ c{=}v\ \mathbf{if}\ c{=}v\ \mathbf{after}\ c{=}v$$

for all $v \in \mathit{Dom}(c)$. These abbreviations for causal laws are part of a longer list given in Section 1.7.

By *SD* we denote the following action description:

$$\begin{array}{l} a\ \mathbf{causes}\ p, \\ \mathbf{exogenous}\ a, \\ \mathbf{inertial}\ p. \end{array} \tag{1.53}$$

The causal theory *SD*$_m$ generated from this action description as defined above is identical to the causal theory (1.23) denoted by *SD*$_m$ in Section 1.3.1 (to be precise, it turns into that theory after dropping the trivial conjunctive terms $\top$ in some formulas, and replacing the atoms of the form $c = \mathbf{f}$ with $\neg c$). Indeed, the first line of (1.53) corresponds to the rules

$$i{+}1 : p \Leftarrow i : a,$$

the second line to

$$\begin{aligned} i : a &\Leftarrow i : a, \\ i : \neg a &\Leftarrow i : \neg a, \end{aligned}$$

and the third to

$$\begin{aligned} i{+}1 : p &\Leftarrow (i : p) \wedge (i{+}1 : p), \\ i{+}1 : \neg p &\Leftarrow (i : \neg p) \wedge (i{+}1 : \neg p). \end{aligned}$$

The remaining two lines of (1.23)

$$\begin{aligned} 0 : p &\Leftarrow 0 : p, \\ 0 : \neg p &\Leftarrow 0 : \neg p \end{aligned}$$

come from (1.52).

1.4.3 Monkey and Bananas in $\mathcal{C}$+

The action description *MB* is shown in Figures 1.2, 1.3. The first two causal laws in this description, corresponding to rules (1.26) and (1.27), are examples of static laws (1.48). According to Section 1.7,

$$\textbf{nonexecutable } c \textbf{ if } F$$

stands for the fluent dynamic law

$$\textbf{caused } \bot \textbf{ if } \top \textbf{ after } c \wedge F.$$

The sequence of causal theories corresponding to action description *MB* is identical to the sequence *MB*$_m$ defined in Section 1.3.2.

Figure 1.4 shows the modification of the signature of *MB* needed to include a set M of names of monkeys and to make some of the causal rules defeasible, as in Section 1.3.4. Note that the fluent constants *Ab*$_1(\alpha, \beta)$ are classified here as statically determined; this is essential because the corresponding causal theories do not contain the rules

$$0 : Ab_1(\alpha, \beta) \Leftarrow 0 : Ab_1(\alpha, \beta)$$

that would have been included, as a special case of (1.52), if these constants were simple. The symbols *Ab*$_2(\alpha)$ are classified as action constants, and not as fluent constants; this is essential because the signature of the corresponding causal theory contains the constants $i : Ab_2(\alpha)$ for $i < m$, but not for $i = m$. Like the other action constants, *Ab*$_2(\alpha)$

Notation: x ranges over $\{Monkey, Bananas, Box\}$; l ranges over $\{L_1, L_2, L_3\}$.

Simple fluent constants:	Domains:
$Loc(x)$	$\{L_1, L_2, L_3\}$
$HasBananas, OnBox$	Boolean

Action constants:	Domains:
$Walk(l)$, $PushBox(l)$, $ClimbOn$, $ClimbOff$, $GraspBananas$	Boolean

Causal laws:

$\mathbf{caused}\ Loc(Bananas)=l\ \mathbf{if}\ HasBananas \wedge Loc(Monkey)=l$
$\mathbf{caused}\ Loc(Monkey)=l\ \mathbf{if}\ OnBox \wedge Loc(Box)=l$

$Walk(l)\ \mathbf{causes}\ Loc(Monkey)=l$
$\mathbf{nonexecutable}\ Walk(l)\ \mathbf{if}\ Loc(Monkey)=l$
$\mathbf{nonexecutable}\ Walk(l)\ \mathbf{if}\ OnBox$

$PushBox(l)\ \mathbf{causes}\ Loc(Box)=l$
$PushBox(l)\ \mathbf{causes}\ Loc(Monkey)=l$
$\mathbf{nonexecutable}\ PushBox(l)\ \mathbf{if}\ Loc(Monkey)=l$
$\mathbf{nonexecutable}\ PushBox(l)\ \mathbf{if}\ OnBox$
$\mathbf{nonexecutable}\ PushBox(l)\ \mathbf{if}\ Loc(Monkey)\neq Loc(Box)$

Figure 1.2: Action description *MB*, Part 1.

describes an event leading from one state to another, rather than a state: its value tells us whether the execution of the action $PushBox(\alpha, l)$ during that event is “abnormal.” In one way, however, $Ab_2(\alpha)$ is different from the other action constants in this example: we would not include

$$\mathbf{exogenous}\ Ab_2(\alpha)$$

in the set of postulates. This is because the corresponding causal theories do not contain the rules

$$i\!:\!Ab_2(\alpha) \Leftarrow i\!:\!Ab_2(\alpha).$$

Using an abbreviation introduced in Section 1.7, we can write the causal laws corresponding to causal rules (1.37) as

$$\mathbf{constraint}\ \neg(HasBananas(\alpha) \wedge HasBananas(\beta))$$

ClimbOn **causes** *OnBox*
nonexecutable *ClimbOn* **if** *OnBox*
nonexecutable *ClimbOn* **if** $Loc(Monkey) \neq Loc(Box)$

ClimbOff **causes** $\neg$*OnBox*
nonexecutable *ClimbOff* **if** $\neg$*OnBox*

GraspBananas **causes** *HasBananas*
nonexecutable *GraspBananas* **if** *HasBananas*
nonexecutable *GraspBananas* **if** $\neg$*OnBox*
nonexecutable *GraspBananas* **if** $Loc(Monkey) \neq Loc(Bananas)$

nonexecutable $Walk(l) \wedge PushBox(l)$
nonexecutable $Walk(l) \wedge ClimbOn$
nonexecutable $PushBox(l) \wedge ClimbOn$
nonexecutable $ClimbOff \wedge GraspBananas$

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Figure 1.3: Action description *MB*, Part 2.

($\alpha \neq \beta$). The pair of rules (1.39), (1.40) can be written as

$$\textbf{constraint } \neg(OnBox(\alpha) \wedge OnBox(\beta)) \textbf{ unless } Ab_1(\alpha, \beta)$$

($\alpha \neq \beta$). The pair (1.43), (1.44) becomes

$$PushBox(\alpha, l) \textbf{ causes } Loc(Box) = l \textbf{ unless } Ab_2(\alpha),$$
$$PushBox(\alpha, l) \textbf{ causes } Loc(\alpha) = l \textbf{ unless } Ab_2(\alpha).$$

The "exception" rules (1.41), (1.45) and (1.46) turn into the static laws

$$\textbf{caused } Ab_1(David, \alpha),$$
$$\textbf{caused } Ab_1(\alpha, David)$$

and the action dynamic laws

$$\textbf{caused } Ab_2(\alpha) \textbf{ if } OnBox(Goliath),$$
$$\textbf{caused } Ab_2(\alpha) \textbf{ if } ClimbOn(Goliath).$$

Notation: α, β range over M; x ranges over $M \cup \{\mathit{Bananas}, \mathit{Box}\}$; l ranges over $\{L_1, L_2, L_3\}$.

Simple fluent constants:	Domains:
$\mathit{Loc}(x)$	$\{L_1, L_2, L_3\}$
$\mathit{HasBananas}(\alpha), \mathit{OnBox}(\alpha)$	Boolean
Statically determined fluent constants:	Domains:
$\mathit{Ab}_1(\alpha, \beta)$	Boolean
Action constants:	Domains:
$\mathit{Walk}(\alpha, l)$, $\mathit{PushBox}(\alpha, l)$, $\mathit{ClimbOn}(\alpha)$, $\mathit{ClimbOff}(\alpha)$,	
$\mathit{GraspBananas}(\alpha)$, $\mathit{Ab}_2(\alpha)$	Boolean

Figure 1.4: The signature of the action description corresponding to the sequence of causal theories from Section 1.3.4.

1.4.4 From Action Descriptions to Transition Systems

Consider an action description D with a set σ^{fl} of fluent constants and a set σ^{act} of action constants. We will define now the transition system represented by D, using the first two members D_0, D_1 of the sequence of causal theories corresponding to D. In this definition, we identify an interpretation I in the sense of Section 1.2.1 with the set of atoms that are satisfied by this interpretation, that is to say, with the set of atoms of the form $c = I(c)$. This convention allows us to represent any interpretation of the signature of D_m in the form

$$(0\!:\!s_0) \cup (0\!:\!e_0) \cup (1\!:\!s_1) \cup (1\!:\!e_1) \cup \cdots \cup (m\!:\!s_m) \qquad (1.54)$$

where $s_0, \ldots, s_m$ are interpretations of σ^{fl}, and $e_1, \ldots, e_{m-1}$ are interpretations of σ^{act}. For instance, for the interpretation I defined by (1.47),

$$\begin{aligned} I &= \{0\!:\!p = \mathbf{f},\ 0\!:\!a = \mathbf{f},\ 1\!:\!p = \mathbf{f},\ 1\!:\!a = \mathbf{t},\ 2\!:\!p = \mathbf{t}\} \\ &= 0\!:\!\{p = \mathbf{f}\} \cup 0\!:\!\{a = \mathbf{f}\} \cup 1\!:\!\{p = \mathbf{f}\} \cup 1\!:\!\{a = \mathbf{t}\} \cup 2\!:\!\{p = \mathbf{t}\}. \end{aligned}$$

A *state* is an interpretation s of σ^{fl} such that $0 : s$ is a model of D_0. States are the vertices of the transition system represented by D. A *transition* is a triple $\langle s, e, s' \rangle$, where s and s' are interpretations of σ^{fl} and e is an interpretation of σ^{act}, such that $0\!:\!s \cup 0\!:\!e \cup 1\!:\!s'$ is a model of D_1. Transitions correspond to the edges of the transition system: for every transition $\langle s, e, s' \rangle$, it contains an edge from s to s' labeled e. These labels e will be called *events*.

For example, the vertices of the transition system in Figure 1.1 are $\{p = \mathbf{f}\}$ and $\{p = \mathbf{t}\}$; accordingly, $0\!:\!\{p = \mathbf{f}\}$ and $0\!:\!\{p = \mathbf{t}\}$ are the models of causal theory (1.24).

The transitions of that transition system are

$$\begin{array}{c}\langle\{p=\mathbf{f}\},\{a=\mathbf{f}\},\{p=\mathbf{f}\}\rangle,\\ \langle\{p=\mathbf{f}\},\{a=\mathbf{t}\},\{p=\mathbf{t}\}\rangle,\\ \langle\{p=\mathbf{t}\},\{a=\mathbf{f}\},\{p=\mathbf{t}\}\rangle,\\ \langle\{p=\mathbf{t}\},\{a=\mathbf{t}\},\{p=\mathbf{t}\}\rangle;\end{array}$$

accordingly, the interpretations

$$\begin{array}{c}0\!:\!\{p=\mathbf{f}\}\cup 0\!:\!\{a=\mathbf{f}\}\cup 1\!:\!\{p=\mathbf{f}\},\\ 0\!:\!\{p=\mathbf{f}\}\cup 0\!:\!\{a=\mathbf{t}\}\cup 1\!:\!\{p=\mathbf{t}\},\\ 0\!:\!\{p=\mathbf{t}\}\cup 0\!:\!\{a=\mathbf{f}\}\cup 1\!:\!\{p=\mathbf{t}\},\\ 0\!:\!\{p=\mathbf{t}\}\cup 0\!:\!\{a=\mathbf{t}\}\cup 1\!:\!\{p=\mathbf{t}\}\end{array}$$

are the models of causal theory (1.19)–(1.22).

The definition of the transition system above implicitly relies on the following property of transitions:

Proposition 7 *For any transition $\langle s,e,s'\rangle$, s and s' are states.*

The validity of this proposition depends on the fact that in the definition of a fluent dynamic law (Section 1.4.2) the head is not allowed to contain statically determined fluent constants. Indeed, imagine that this limitation is lifted. Then we will be able to form an action description D that consists of two causal laws: the static law

$$\textbf{caused}\ p\ \textbf{if}\ p$$

and the fluent dynamic law

$$\textbf{caused}\ \neg p\ \textbf{if}\ \top\ \textbf{after}\ p$$

where p is a statically determined Boolean fluent constant. For this D, causal theory D_0 is

$$0\!:\!p \Leftarrow 0\!:\!p$$

and causal theory D_1 is

$$\begin{array}{c}0\!:\!p \Leftarrow 0\!:\!p,\\ 1\!:\!p \Leftarrow 1\!:\!p,\\ 1\!:\!\neg p \Leftarrow 0\!:\!p.\end{array}$$

The last member $\{p=\mathbf{f}\}$ of the transition $\langle\{p=\mathbf{t}\},\emptyset,\{p=\mathbf{f}\}\rangle$ is not a state.

This fact explains the need to distinguish between fluent constants of two kinds—simple, that are allowed in the heads of dynamic laws of D, and statically determined, for which causal rules (1.52) are not automatically included in D_m.

The correspondence between the models of D_m and paths in the transition system represented by D can be described as follows:

Proposition 8 *For any $m>0$, an interpretation (1.54) of the signature of D_m is a model of D_m iff each triple $\langle s_i,e_i,s_{i+1}\rangle$ $(0\leq i<m)$ is a transition.*

1.4.5 Causal Logic as a Subset of $\mathcal{C}$+

In Section 1.4.2 we defined the semantics of $\mathcal{C}$+ by showing how to turn any action description into a sequence of causal theories. A reduction in the opposite direction is possible also. Any causal theory T can be turned into an action description by treating every constant of T as a statically determined fluent constant, and rewriting every causal rule

$$F \Leftarrow G$$

as the static law

$$\textbf{caused } F \textbf{ if } G.$$

It is easy to see that the states of the transition system represented by this action description are identical to the models of T.

We will agree to identify a causal theory with the corresponding action description. We will treat any causal theory, in other words, as an action description without simple fluent constants, without action constants, and without dynamic laws. This convention allows us to specify causal theories using the abbreviations for static laws introduced in Section 1.7. For instance, (1.2) can be now written as

$$\begin{array}{l}\textbf{caused } p \textbf{ if } q,\\ \textbf{exogenous } q,\end{array} \tag{1.55}$$

and (1.11) can be written as

$$\textbf{default } c=1.$$

1.4.6 Rigid Constants

A fluent constant c in the signature of an action description D is *rigid* (relative to D) if, for every transition $\langle s, e, s'\rangle$ in the transition system represented by D, $s'(c) = s(c)$. Intuitively, rigid constants represent the fluents whose values are not affected by any events.

Imagine, for instance, that the monkey in the Monkey and Bananas domain is unable to push the box. This assumption can be expressed by adding the causal laws

$$\textbf{nonexecutable } \mathit{PushBox}(l)$$

to the action description *MB* shown in Figures 1.2, 1.3. This modification makes the fluent constant $\mathit{Loc}(\mathit{Box})$ rigid. The corresponding transition system consists of 3 disconnected parts: $s(\mathit{Loc}(\mathit{Box}))$ is L_1 for the states s in the first part, L_2 in the second part, and L_3 in the third. No edges of the transition system lead from one part to another.

Other examples of action descriptions with rigid constants are given by the extensions of *MB* that contain symbols for the size and the material of the box, or for the species and gender of the monkey.

According to Section 1.7, the expression

$$\textbf{rigid}\ c$$

stands for the set of causal laws

$$\textbf{constraint}\ c{=}v\ \textbf{after}\ c{=}v,$$

that is to say,

$$\textbf{caused}\ \bot\ \textbf{if}\ \neg(c{=}v)\ \textbf{after}\ c{=}v$$

for all $v \in \mathit{Dom}(c)$. It is clear that c is rigid relative to any action description containing these laws.

One of the reasons why rigid constants are interesting is that, under some conditions, their presence allows us to make the causal theories D_m more compact, which can be computationally advantageous. Let R be a set of fluent constants that are rigid relative to D. Denote by D_m^R the causal theory whose constants and causal rules are obtained from the constants and causal rules of D_m by dropping the time stamps before each constant from R. For any interpretation I of the signature of D_m, by I^R we denote the interpretation of the signature of D_m^R defined by the formulas

$$\begin{array}{ll} I^R(c) = I(0:c) & \text{if } c \in R, \\ I^R(i{:}c) = I(i{:}c) & \text{if } c \notin R. \end{array}$$

Proposition 9 *If*

(i) every constant in R is statically determined, and

(ii) for every causal law in D that contains a constant from R in the head, all constants occurring in this law belong to R,

then the mapping $I \mapsto I^R$ is a 1-1 correspondence between the models of D_m and the models of D_m^R.

Thus dropping the time stamps in front of the rigid constants from R does not affect the meaning of D_m if, first, R contains no simple constants, and second, no constant from R “causally depends” on a constant that does not belong to R.

The following example shows that the assertion of Proposition 9 would be incorrect without the first condition. Take D to be

$$\begin{array}{l} \textbf{rigid}\ p, \\ \textbf{default}\ p \end{array}$$

where p is a Boolean simple fluent, and let $R = \{p\}$. Then D_1 is

$$\begin{array}{l}
\bot \Leftarrow (1\!:\!p) \wedge \neg(0\!:\!p),\\
\bot \Leftarrow \neg(1\!:\!p) \wedge (0\!:\!p),\\
0\!:\!p \Leftarrow 0\!:\!p,\\
1\!:\!p \Leftarrow 1\!:\!p,\\
0\!:\!\neg p \Leftarrow 0\!:\!\neg p
\end{array}$$

and D_1^R is

$$\begin{array}{l}
\bot \Leftarrow p \wedge \neg p,\\
\bot \Leftarrow \neg p \wedge p,\\
p \Leftarrow p,\\
\neg p \Leftarrow \neg p.
\end{array}$$

The interpretation $\{p = \mathbf{f}\}$ is a model of D_1^R, but it does not have the form I^R for any model I of D_1.

The following example shows that the assertion of Proposition 9 would be incorrect without the second condition. Take D to be (1.55), where p and q are statically determined fluent constants, and let $R = \{p\}$. Then D_1 is

$$\begin{array}{l}
0\!:\!p \Leftarrow 0\!:\!q,\\
1\!:\!p \Leftarrow 1\!:\!q,\\
0\!:\!q \Leftarrow 0\!:\!q,\\
1\!:\!q \Leftarrow 1\!:\!q,\\
\neg 0\!:\!q \Leftarrow \neg 0\!:\!q,\\
\neg 1\!:\!q \Leftarrow \neg 1\!:\!q
\end{array}$$

and D_1^R is

$$\begin{array}{l}
p \Leftarrow 0\!:\!q,\\
p \Leftarrow 1\!:\!q,\\
0\!:\!q \Leftarrow 0\!:\!q,\\
1\!:\!q \Leftarrow 1\!:\!q,\\
\neg 0\!:\!q \Leftarrow \neg 0\!:\!q,\\
\neg 1\!:\!q \Leftarrow \neg 1\!:\!q.
\end{array}$$

The interpretation $\{p = \mathbf{t}, 0:q = \mathbf{f}, 1:q = \mathbf{t}\}$ is a model of D_1^R, but it does not have the form I^R for any model I of D_1.

1.5 Expressive Possibilities of the Definite Fragment of $\mathcal{C}$+

An action description D is *definite* if

- the head of every causal law of D is an atom or $\bot$, and

- no atom is the head of infinitely many causal laws of D.

This is similar to the definition of a definite causal theory in Section 1.2.6. For any definite action description D, the corresponding causal theories D_m are definite also, so that many questions about transition systems described in the definite fragment of $\mathcal{C}$+ can be answered by reduction to propositional reasoning.

The examples of action descriptions discussed in Section 1.4 are definite (or can be made definite by replacing formulas $\neg c$ in the heads of causal laws by atoms $c = \mathbf{f}$). In this section we give a few other examples illustrating the expressive possibilities of the definite fragment of $\mathcal{C}$+.

1.5.1 Actions with Conditional Effects

Effects of an action can be "conditional": they may be caused by executing the action in some states, but not in others. As an example of representing conditional effects in $\mathcal{C}$+, we formalize in Figure 1.5 an enhancement of the well-known "Yale shooting" example [HM87] in which the effect of shooting depends on how the gun is aimed. As defined in Section 1.7, the expression

$$\mathit{Shoot}\ \mathbf{causes}\ \neg \mathit{Alive}(x)\ \mathbf{if}\ \mathit{Target}{=}x$$

in this action description stands for the fluent dynamic law

$$\mathbf{caused}\ \neg \mathit{Alive}(x)\ \mathbf{if}\ \top\ \mathbf{after}\ \mathit{Shoot} \wedge \mathit{Target}{=}x.$$

1.5.2 Nondeterministic Actions

When Jack goes to work, he either walks there or drives his car. We view walking and driving as two ways of executing the same action. The effect of that action on Jack's location is deterministic, but its effect on the location of his car is not.

The representation of this example in Figure 1.6 describes the nondeterministic effect of $\mathit{Go}(l)$ by the expression

$$\mathit{Go}(l)\ \mathbf{may\ cause}\ \mathit{Loc}(\mathit{Car}){=}l\ \mathbf{if}\ \mathit{Loc}(\mathit{Car}){=}\mathit{Loc}(\mathit{Jack}).$$

According to Section 1.7, it stands for the fluent dynamic law

$$\begin{aligned}\mathbf{caused}\ \mathit{Loc}(\mathit{Car}){=}l\ \mathbf{if}\ \mathit{Loc}(\mathit{Car}){=}l&\\ \mathbf{after}\ \mathit{Go}(l) \wedge \mathit{Loc}(\mathit{Car}){=}\mathit{Loc}(\mathit{Jack})&.\end{aligned}$$

By placing a copy of the head $\mathit{Loc}(\mathit{Car}) = l$ after **if** we say that this is not a necessary effect of the action, but merely a possible one. If this condition holds in the resulting state then there is a cause for this.

Notation: x ranges over $\{$*Turkey1*,*Turkey2*$\}$.

Simple fluent constants:	Domains:
Loaded, *Alive*(x)	Boolean
Target	$\{$*Turkey1*, *Turkey2*, *None*$\}$

Action constants:	Domains:
Load, *Aim*(x), *Shoot*	Boolean

Causal laws:

Load **causes** *Loaded*
Load **causes** *Target*$=$*None*

Aim(x) **causes** *Target*$=x$

Shoot **causes** $\neg$*Alive*(x) **if** *Target*$=x$
Shoot **causes** $\neg$*Loaded*
nonexecutable *Shoot* **if** $\neg$*Loaded*

nonexecutable *Aim*$(x) \wedge$ *Shoot*

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Figure 1.5: Shooting turkeys.

In the transition system represented by this action description we can find two different edges that start at the same state and are labeled by the same event. For instance, there are two edges that start at

$$\{Loc(Jack)=Home, Loc(Car)=Home\}$$

and have the label

$$\{Go(Home)=\mathbf{f}, Go(Work)=\mathbf{t}\}.$$

One of them leads to

$$\{Loc(Jack)=Work, Loc(Car)=Home\}$$

(walking), and the other to

$$\{Loc(Jack)=Work, Loc(Car)=Work\}$$

(driving).

Notation: x ranges over $\{Jack, Car\}$; l ranges over $\{Home, Work\}$.

Simple fluent constants:	Domains:
$Loc(x)$	$\{Home, Work\}$
Action constants:	Domains:
$Go(l)$	Boolean

Causal laws:

$Go(l)$ **causes** $Loc(Jack)\!=\!l$
$Go(l)$ **may cause** $Loc(Car)\!=\!l$ **if** $Loc(Car)\!=\!Loc(Jack)$
nonexecutable $Go(l)$ **if** $Loc(Jack)\!=\!l$

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Figure 1.6: Going to work.

1.5.3 Interaction between Concurrently Executed Actions

In a standard example of interacting actions, two agents lift the opposite ends of a table upon which various objects have been placed [Ped87, Section 3]. If one end of the table has been raised, the objects on the table will fall off. But if both ends are lifted simultaneously, the objects on the table will remain fixed.

A formalization of this domain in $\mathcal{C}$+ is shown in Figure 1.7. The change in the value of the fluent *OnTable* is treated here as an indirect effect of changes in the positions of the ends of the table.

1.5.4 Noninertial Fluents

Consider a pendulum that moves from its leftmost position to the rightmost and back, with each swing taking one unit of time. We would like to describe the effect of holding the pendulum steady in its current position for the duration of one unit of time.

The action description in Figure 1.8 describes the "default" behavior of the pendulum by the expressions

$$\textbf{default } Right \textbf{ after } \neg Right$$
$$\textbf{default } \neg Right \textbf{ after } Right$$

Notation: x ranges over $\{\mathit{LeftEnd}, \mathit{RightEnd}\}$.

Simple fluent constants:	Domains:
$\mathit{Level}(x)$	$\{\mathit{Low}, \mathit{High}\}$
OnTable	Boolean

Action constants:	Domains:
$\mathit{Lift}(x)$	Boolean

Causal laws:

$\mathit{Lift}(x)$ **causes** $\mathit{Level}(x) = \mathit{High}$
nonexecutable $\mathit{Lift}(x)$ **if** $\mathit{Level}(x) = \mathit{High}$

caused $\neg\mathit{OnTable}$ **if** $\mathit{Level}(\mathit{LeftEnd}) \neq \mathit{Level}(\mathit{RightEnd})$

exogenous c	for every action constant c
inertial c	for every simple fluent constant c

Figure 1.7: Lifting the ends of the table.

Simple fluent constant:	Domain:
Right	Boolean

Action constant:	Domain:
Hold	Boolean

Causal laws:

Hold **causes** *Right* **if** *Right*
Hold **causes** $\neg$*Right* **if** $\neg$*Right*

default *Right* **after** $\neg$*Right*
default $\neg$*Right* **after** *Right*

exogenous *Hold*

Figure 1.8: Holding the pendulum.

that stand for the dynamic laws

$$\begin{array}{l}\textbf{caused } \mathit{Right} \textbf{ if } \mathit{Right} \textbf{ after } \neg \mathit{Right},\\ \textbf{caused } \neg\mathit{Right} \textbf{ if } \neg\mathit{Right} \textbf{ after } \mathit{Right}.\end{array}$$

(Section 1.6). They are used here instead of the dynamic laws

$$\textbf{inertial } \mathit{Right},$$

that is to say, instead of

$$\begin{array}{l}\textbf{caused } \mathit{Right} \textbf{ if } \mathit{Right} \textbf{ after } \mathit{Right},\\ \textbf{caused } \neg\mathit{Right} \textbf{ if } \neg\mathit{Right} \textbf{ after } \neg\mathit{Right}.\end{array}$$

This is an example of a simple fluent constant that is not postulated to be inertial.

1.5.5 Defined Fluents

In the process of designing an action description, we may want to introduce a fluent constant that is defined in terms of other fluent constants. To see how this can be done, imagine that we want to expand action description *MB* (Figures 1.2, 1.3) by the new Boolean fluent constant *NextToBox* that can serve as shorthand for the formula $\mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box})$.

In the extended action description MB' we make the new fluent constant *NextToBox* statically determined, and include two static laws containing that constant in the head:

$$\begin{array}{l}\textbf{caused } \mathit{NextToBox} \textbf{ if } \mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box}),\\ \textbf{default } \neg\mathit{NextToBox}\end{array} \tag{1.56}$$

(there is a cause for *NextToBox* to be true if the monkey and the box are at the same location; otherwise, *NextToBox* is false).

It is easy to see that the completion of MB'_m can be obtained from the completion of MB_m by adding the formulas

$$i : (\mathit{NextToBox} \equiv \mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box}))$$

($i = 0, \dots, m$). This observation shows that the transition system represented by *MB* is isomorphic to the transition system represented by MB': every state of the former can be turned into the corresponding state of the latter by assigning to *NextToBox* the truth value equal to the truth value of the formula $\mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box})$.

In the presence of (1.56), any occurrences of $\mathit{Loc}(\mathit{Monkey}) = \mathit{Loc}(\mathit{Box})$ in the bodies of the causal laws of *MB* can be equivalently replaced with *NextToBox*. For instance,

$$\textbf{nonexecutable } \mathit{PushBox}(l) \textbf{ if } \mathit{Loc}(\mathit{Monkey}) \neq \mathit{Loc}(\mathit{Box})$$

can be rewritten as

$$\mathbf{nonexecutable}\ \textit{PushBox}(l)\ \mathbf{if}\ \neg\textit{NextToBox}.$$

Proposition 4(ii) shows that the first of the causal laws (1.56) can be equivalently replaced with

$$\mathbf{caused}\ \textit{NextToBox}\ \mathbf{if}\ \textit{Loc}(\textit{Monkey})=l \wedge \textit{Loc}(\textit{Box}) = l$$

($l = L_1, L_2, L_3$).

The fact that the transition systems represented by *MB* and *MB*$'$ are isomorphic to each other depends on the assumption that the new fluent constant is statically determined. If we made *NextToBox* a simple fluent constant then the causal theories *MB*$'$ would have been different because of the instance

$$0\!:\!\textit{NextToBox} \Leftarrow 0\!:\!\textit{NextToBox}$$

of (1.52).

1.5.6 Describing Actions by Attributes

Executing the action *Publish* causes the Boolean fluent *HasPublications* to become true:

$$\textit{Publish}\ \mathbf{causes}\ \textit{HasPublications}. \tag{1.57}$$

Imagine that we want to enhance an action description containing this causal law by distinguishing between publications of different kinds. The fluent *HasPublications* becomes true no matter whether a conference paper or a journal article has been published, but in the second case the action has one more effect—the fluent *HasJournalPublications* becomes true also. This distinction can be expressed by switching from the notation *Publish* for the action in question to the more elaborate notation *Publish*(k), where k ranges over the types of publications *Conference, Journal.* In the modified action description, (1.57) turns into

$$\textit{Publish}(k)\ \mathbf{causes}\ \textit{HasPublications} \tag{1.58}$$

for both possible values of k, and additionally we postulate

$$\textit{Publish}(\textit{Journal})\ \mathbf{causes}\ \textit{HasJournalPublications}. \tag{1.59}$$

If we later decide to distinguish between publications in terms of their length, the notation for the action will need to be amended again. Instead of *Publish*(k), we can denote the action, for instance, by *Publish*(k, l), where the number of pages l ranges over an initial segment of positive integers. Causal laws (1.58) and (1.59) will turn then into

$$\begin{array}{l}\textit{Publish}(k,l)\ \mathbf{causes}\ \textit{HasPublications},\\ \textit{Publish}(\textit{Journal},l)\ \mathbf{causes}\ \textit{HasJournalPublications},\end{array} \tag{1.60}$$

Notation: l ranges over $\{1, \ldots, 100\}$.

Simple fluent constants:	Domains:
HasPublications, *HasJournalPublications*,	
HasLongPublications	Boolean
Action constants:	Domains:
Publish	Boolean
Kind	$\{Conference, Journal, None\}$
Length	$\{1, \ldots, 100, None\}$

Causal laws:

$\mathbf{constraint}$ *HasJournalPublications* $\supset$ *HasPublications*
$\mathbf{constraint}$ *HasLongPublications* $\supset$ *HasPublications*

Publish **causes** *HasPublications*

$\mathbf{always}$ $Kind = None \equiv \neg Publish$
Publish **causes** *HasJournalPublications* **if** $Kind = Journal$

$\mathbf{always}$ $Length = None \equiv \neg Publish$
Publish **causes** *HasLongPublications* **if** $Length = l$ for $l > 30$

$\mathbf{exogenous}$ c for every action constant c

$\mathbf{inertial}$ c for every simple fluent constant c

Figure 1.9: Publishing papers.

and we also will be able to say

$$Publish(k, l) \textbf{ causes } HasLongPublications \qquad (l > 30)$$

to express that publishing a paper that is over 30 pages causes the fluent *HasLongPublications* to become true.

This example illustrates the need to modify notation for actions by introducing additional arguments, and to change the form of causal laws, that often arises when we want to introduce additional distinctions. An alternative way to represent such distinctions, which is sometimes preferable, is based on expressing them by “attributes.” In Figure 1.9, the kind and length of a publication are treated as attributes of the action *Publish*. The attributes *Kind* and *Length* are action constants—their values, like the value of *Publish*, characterize an event occurring between two states. The domain of every attribute of an action includes the special value *None*, and the attribute is required to take that value (to be "undefined") if and only if the action is not executed. For instance, we postulate

$$\textbf{always } \mathit{Kind}\!=\!\mathit{None} \equiv \neg \mathit{Publish}.$$

According to Section 1.6, this expression stands for the fluent dynamic law

$$\textbf{caused } \bot \textbf{ if } \top \textbf{ after } \neg(\mathit{Kind}\!=\!\mathit{None} \equiv \neg \mathit{Publish}).$$

1.6 Reducing Multi-Valued Formulas to Classical Formulas

If we disregard the internal structure of atoms $c\!=\!v$ of a multi-valued propositional signature σ (Section 1.2.1) then any formula of this signature becomes a classical formula, and it can be evaluated relative to any truth-valued function on the set of atoms. To find a model of a set X of formulas of signature σ means to find a truth-valued function I on the set of atoms that is a model of X in the sense of classical propositional logic and has the following additional property: for each $c \in \sigma$ there exists a unique $v \in \mathit{Dom}(c)$ such that $I(c\!=\!v) = \mathbf{t}$. This property can be expressed by saying that I satisfies the formulas

$$\bigvee_v c\!=\!v \;\wedge \bigwedge_{v\neq w} \neg(c\!=\!v \wedge c\!=\!w) \tag{1.61}$$

for all $c \in \sigma$. Consequently, to find a model of X means to find a model of the union of X with formulas (1.61) in the sense of classical propositional logic.

If some constants in σ are Boolean then the translation to classical propositional logic can be made more economical: for a Boolean constant c, we can replace every occurrence of $c\!=\!\mathbf{f}$ with $\neg c$, drop the corresponding formula (1.61), and then drop $c\!=\!\mathbf{f}$ from the set of atoms. For instance, the models of (1.17) can be computed without adding any of the formulas (1.61) if we rewrite (1.17) in the form

$$\begin{array}{l} p \equiv q, \\ \neg p \equiv \bot, \\ q \equiv q, \\ \neg q \equiv \neg q. \end{array}$$

A similar simplification can be used for any constant with a two-element domain.

The number of conjunctive terms

$$\neg(c\!=\!v \wedge c\!=\!w) \qquad (v \neq w) \tag{1.62}$$

in formula (1.61) is quadratic in the cardinality of $\mathit{Dom}(c)$, which makes this formula computationally unattractive when the domain of c is large. Instead of (1.62), we can use a definitional extension of this set of formulas whose size is linear in the number of elements $v_1, \dots, v_n$ of *Dom*(c):

$$\begin{array}{ll}
p_1 \equiv c{=}v_1 & \neg(p_1 \wedge c{=}v_2) \\
p_2 \equiv p_1 \vee c{=}v_2 & \neg(p_2 \wedge c{=}v_3) \\
\dots & \dots \\
p_{n-1} \equiv p_{n-2} \vee c{=}v_{n-1} & \neg(p_{n-1} \wedge c{=}v_n)
\end{array}$$

where $p_1, \dots, p_{n-1}$ are new Boolean constants.

1.7 Abbreviations for Causal Laws

1. A static law or an action dynamic law of the form

$$\textbf{caused } F \textbf{ if } \top$$

can be written as

$$\textbf{caused } F.$$

2. A fluent dynamic law of the form

$$\textbf{caused } F \textbf{ if } \top \textbf{ after } H$$

can be written as

$$\textbf{caused } F \textbf{ after } H.$$

3. A static law of the form

$$\textbf{caused } \bot \textbf{ if } \neg F$$

can be written as

$$\textbf{constraint } F.$$

4. A fluent dynamic law of the form

$$\textbf{caused } \bot \textbf{ if } \neg F \textbf{ after } G$$

can be written as

$$\textbf{constraint } F \textbf{ after } G.$$

5. An expression of the form

$$\textbf{rigid } c$$

where c is a fluent constant stands for the set of causal laws

$$\textbf{constraint } c{=}v \textbf{ after } c{=}v$$

for all $v \in Dom(c)$.

6. A fluent dynamic law of the form

$$\textbf{caused } \bot \textbf{ after } \neg F$$

can be written as

$$\textbf{always } F.$$

7. A fluent dynamic law of the form

$$\textbf{caused } \bot \textbf{ after } F \wedge G$$

where F is an action formula can be written as

$$\textbf{nonexecutable } F \textbf{ if } G. \tag{1.63}$$

8. An expression of the form

$$F \textbf{ causes } G \textbf{ if } H \tag{1.64}$$

where F is an action formula stands for the fluent dynamic law

$$\textbf{caused } G \textbf{ after } F \wedge H$$

if G is a fluent formula,[4] and for the action dynamic law

$$\textbf{caused } G \textbf{ if } F \wedge H$$

if G is an action formula.

9. An expression of the form

$$\textbf{default } F \textbf{ if } G \tag{1.65}$$

stands for the causal law

$$\textbf{caused } F \textbf{ if } F \wedge G.$$

10. An expression of the form

$$\textbf{default } F \textbf{ if } G \textbf{ after } H$$

stands for the fluent dynamic law

$$\textbf{caused } F \textbf{ if } F \wedge G \textbf{ after } H.$$

[4]It is clear that the expression in the previous line is a fluent dynamic law only when G does not contain statically determined constants. Similar remarks can be made in connection with many of the abbreviations introduced below.

The part

$$\textbf{if } G$$

in this abbreviation can be dropped if G is $\top$.

11. An expression of the form

$$\textbf{exogenous } c \textbf{ if } G \tag{1.66}$$

where c is a constant stands for the set of causal laws

$$\textbf{default } c{=}v \textbf{ if } G$$

for all $v \in$ *Dom*(c).

12. An expression of the form

$$F \textbf{ may cause } G \textbf{ if } H \tag{1.67}$$

where F is an action formula stands for the fluent dynamic law

$$\textbf{default } G \textbf{ after } F \wedge H$$

if G is a fluent formula, and for the action dynamic law

$$\textbf{default } G \textbf{ if } F \wedge H$$

if G is an action formula.

13. An expression of the form

$$\textbf{inertial } c \textbf{ if } G \tag{1.68}$$

where c is a fluent constant stands for the set of fluent dynamic laws

$$\textbf{default } c{=}v \textbf{ after } c{=}v \wedge G$$

for all $v \in$ *Dom(c)*.

14. If any of the abbreviations (1.64)–(1.68) ends with

$$\textbf{if } \top$$

then this part of the expression can be dropped.

15. The expression obtained by appending

$$\textbf{unless } c$$

where c is a Boolean statically determined fluent constant to a static law

$$\textbf{caused } F \textbf{ if } G \tag{1.69}$$

stands for the pair of static laws

$$\begin{array}{l}\textbf{caused } F \textbf{ if } G \wedge \neg c,\\ \textbf{default } \neg c.\end{array} \tag{1.70}$$

16. The expression obtained by appending

$$\textbf{unless } c$$

where c is a Boolean action constant to an action dynamic law (1.69) stands for the pair of action dynamic laws (1.70).

17. The expression obtained by appending

$$\textbf{unless } c$$

where c is a Boolean action constant to a fluent dynamic law

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H$$

stands for the pair of dynamic laws

$$\begin{array}{l}\textbf{caused } F \textbf{ if } G \textbf{ after } H \wedge \neg c,\\ \textbf{default } \neg c.\end{array}$$

1.8 Related Work

1.8.1 Default Logic

A causal theory of a Boolean signature can be viewed as a default theory in the sense of [Rei80]. (The syntax and semantics of propositional default theories are reviewed in Section 1.9.10.) Let us agree to identify a causal rule $F \Leftarrow G$ with the default

$$\frac{:\ G}{F}. \tag{1.71}$$

In the statement of the following proposition, we identify an interpretation I with the set of formulas satisfied by I.

Proposition 10 *Let T be a causal theory of a Boolean signature. An interpretation I is a model of T iff I is an extension for T in the sense of default logic.*

This theorem shows that causal rules are essentially propositional defaults without prerequisites and with a single justification, as long as we are only interested in the extensions that correspond to interpretations (that is to say, in the extensions that are consistent and complete).

For instance, causal theory (1.2) corresponds to the default theory

$$\frac{:\ q}{p},\ \frac{:\ q}{q},\ \frac{:\ \neg q}{\neg q}.$$

This theory has two extensions: the set of all consequences of p, q and the set of all consequences of $\neg q$. The first extension is complete, and it corresponds to the only model of (1.2).

This translation into default logic can be applied, in particular, to causal theories of the form D_m (Section 1.4). Given an action description D whose signature is Boolean, and a nonnegative integer m, consider the following set of defaults:[5]

$$\frac{:\ (i\!:\!G)}{i\!:\!F}$$

for every static law (1.48) in D and every $i \in \{0, \ldots, m\}$, and for every action dynamic law (1.48) in D and every $i \in \{0, \ldots, m-1\}$;

$$\frac{:\ (i{+}1\!:\!G) \wedge (i\!:\!H)}{i{+}1\!:\!F} \tag{1.72}$$

for every fluent dynamic law (1.49) in D and every $i \in \{0, \ldots, m-1\}$;

$$\frac{:\ (0\!:\!c{=}v)}{0\!:\!c{=}v}$$

for every simple fluent constant c and every $v \in \mathit{Dom}(c)$. According to Proposition 10, an interpretation I is a model of D_m if and only if it is an extension for this set of defaults. Alternatively, the second conjunctive term in the justification of (1.72) can be turned into a prerequisite, so that this default becomes

$$\frac{(i\!:\!H)\ :\ (i{+}1\!:\!G)}{i{+}1\!:\!F}.$$

As it turns out, if the translation of the fluent dynamic laws of D is modified in this way, the complete, consistent extensions for the default theory remain the same. (Proof is straightforward using the Splitting Sequence Theorem for default logic from [Tur96].)

It is interesting to apply the modified translation to the dynamic laws $\mathbf{inertial}\ c$ (Section 1.4). For Boolean c, this expression stands for the pair of dynamic laws

$$\begin{array}{l}\mathbf{caused}\ c\ \mathbf{if}\ c\ \mathbf{after}\ c,\\ \mathbf{caused}\ \neg c\ \mathbf{if}\ \neg c\ \mathbf{after}\ \neg c.\end{array}$$

[5]Note that in these defaults the colon is used in two ways: to separate justifications from prerequisites, as in default logic, and to separate time stamps from formulas, as described in Section 1.2.2.

In application to these laws, the modified translation gives the normal defaults

$$\frac{(i\!:\!c)\ :\ (i\!+\!1\!:\!c)}{i\!+\!1\!:\!c},\ \frac{(i\!:\!\neg c)\ :\ (i\!+\!1\!:\!\neg c)}{i\!+\!1\!:\!\neg c}.$$

This observation shows that the approach to the frame problem we adopt is closely related to the "frame default" from [Rei80, Section 1.1.4].

1.8.2 Logic Programming

The embedding of causal logic into default logic (Section 1.8.1) assumes that the underlying signature is Boolean. If we assume, in addition, that the head of every rule is an atom ($c\!=\!\mathbf{t}$ or $c\!=\!\mathbf{f}$), then a causal theory can be also translated into the language of logic programs under the answer set semantics [GL91]. Using the equivalent transformations discussed in Section 1.2.4, such a theory can be rewritten as a set of rules of the form

$$l_0 \Leftarrow l_1 \wedge \cdots \wedge l_n \tag{1.73}$$

where $l_0, \dots, l_n$ ($n \geq 0$) are literals (c or $\neg c$). Let us agree to identify causal rule (1.73) with the logic programming rule

$$l_0 \leftarrow \mathit{not}\ \overline{l_1}, \dots, \mathit{not}\ \overline{l_n} \tag{1.74}$$

($\bar{l}$ stands for the literal complementary to l). In the statement of the following proposition, we identify an interpretation I with the set of literals satisfied by I.

Proposition 11 *Let T be a causal theory whose rules have the form (1.73). An interpretation I is a model of T iff I is an answer set for T in the sense of logic programming.*

This theorem shows that causal rules of the form (1.73) can be viewed as nondisjunctive rules under the answer set semantics, as long as we are only interested in consistent and complete answer sets.

Recall that an expression in the body of a rule in a logic program corresponds to a justification in a default if that expression contains negation as failure, and it corresponds to a conjunctive term of the prerequisite if it does not [GL91, Section 5]. From this perspective, rule (1.74) is similar to default (1.71): the body of (1.74) does not include expressions without negation as failure, and (1.71) does not have a prerequisite. On the other hand, the body of (1.74) may have several occurrences of negation as failure, but (1.71) has only one justification.

The completion process defined in Section 1.2.6 is a modification of the syntactic transformation proposed in [Cla78] as a semantics of negation as failure. For a Boolean signature, this process differs from the propositional case of Clark's completion in that it introduces completion formulas not only for atoms in the sense of classical propositional logic, but also for negative literals (atoms of the form $c = \mathbf{f}$). This is why the completion of causal theories was called "literal completion" when it was first described in [MT97].

This observation shows that it is easy to reduce the propositional case of Clark's completion to completion in the sense of causal logic. Let T be a finite set of rules of the form

$$a \Leftarrow l_1 \wedge \cdots \wedge l_n$$

where a is an atom in the sense of classical propositional logic, and $l_1, \ldots, l_n$ ($n \geq 0$) are literals. Clark's completion of T can be obtained by applying the completion procedure defined in Section 1.2.6 to the union of T with the set of rules

$$\neg a \Leftarrow \neg a$$

for all atoms a. Indeed, in the presence of these additional rules, the completion formulas for the negative literals are tautologies $\neg a \equiv \neg a$.

As a semantics of logic programming, completion is known to lead sometimes to unintuitive results. For instance, under the completion semantics, the rules

$$\begin{array}{l} q(x,x) \Leftarrow \top \\ q(x,y) \Leftarrow p(x,z) \wedge q(z,y) \end{array} \tag{1.75}$$

fail to express that q is the reflexive transitive closure of p.[6] For this reason, the analogy between causal theories and logic programs can be misleading. For instance, the union of (1.75) with the closed-world assumption

$$\neg q(x,y) \Leftarrow \neg q(x,y)$$

fails to express in causal logic that q is the reflexive transitive closure of p.

1.8.3 Action Languages

Language $\mathcal{C}$+ is a new addition to the family of formal languages for describing actions, which started with STRIPS [FN71] and ADL [Ped94]. Gelfond and Lifschitz [GL93] related language $\mathcal{A}$ (which is essentially the propositional fragment of ADL) to logic programming. Baral and Gelfond [BG97] extended this work to a language that permits the concurrent execution of actions (including the "difficult" cases, such as the example discussed in Section 1.5.3).

A similar result for a language with static causal laws is proved in [Tur97]. That work, along with the theory of nonmonotonic causal reasoning presented in [MT97], has led to the design of language $\mathcal{C}$ [GL98], which is the immediate predecessor of $\mathcal{C}$+.

[6]Example: combine (1.75) with rules $p(1,1) \Leftarrow \top$ and $p(2,2) \Leftarrow \top$, and let x, y, z range over $\{1,2\}$. Clark's completion of this program has an unintended model in which q is identically true.

1.8.4 Other Research on Representing Actions

This line of research is part of a long line of research on representing actions, and, in particular, of research on the frame problem in the presence of actions with indirect effects. A critical discussion of this work and a comprehensive bibliography can be found in [Sha97].

The explicit definition of $1:p$ in terms of $0:p$ and $0:a$ in Section 1.3.1 is similar to successor state axioms introduced by Reiter [Rei91] as a generalization of earlier work by Pednault [Ped89].

Among the research on the nonmonotonic logic of causation, besides the papers mentioned in the introduction, Fangzhen Lin's proposal to circumscribe the predicate *Caused* is particularly relevant [Lin95, Lin96]. Although that approach uses a different model of nonmonotonic reasoning, our formalism is similar to it in the sense that both theories talk about "being caused," but not about what a cause is. It is also interesting to note that the circumscriptions used by Lin can be often reduced to a form of completion, just like definite causal theories (Section 1.2.6).

An alternative way to apply causation to describing indirect effects of actions is proposed in [Thi97].

Nonmonotonic causal theories are similar in some ways to temporal action logics [DGKK98], which were inspired by the ideas of [San95]. The Causal Calculator can be compared to VITAL ("Visualization and Implementation of Temporal Action Logics")[7]—a powerful software system for planning and for query answering in action domains.

The use of action attributes (Section 1.5.6) is an old idea in linguistic representations, originally due to Davidson [Dav67].

1.9 Proofs of Theorems

1.9.1 Proof of Proposition 1

Proposition 1 *An interpretation I is a model of a causal theory T if and only if, for every formula F,*

$$I \models F \quad \textit{iff} \quad T^I \models F\,.$$

[7]`http://www.ida.liu.se/~jonkv/vital/` .

Proof

$$\begin{array}{lll} I \text{ is a model of } T & \text{iff} & I \text{ is the unique model of } T^I \\ & \text{iff} & \text{for every atom } c{=}v,\ I \models c{=}v \text{ iff } T^I \models c{=}v \\ & \text{iff} & \text{for every formula } F,\ I \models F \text{ iff } T^I \models F. \end{array}$$

1.9.2 Proof of Proposition 2

Proposition 2 *If a causal theory T contains a causal rule $F \Leftarrow G$ then T entails $G \supset F$.*

Proof Let I be a model of T. If $I \models G$ then $F \in T^I$, and consequently $I \models F$.

1.9.3 Proof of Proposition 3

Our proof of Σ_2^P-hardness is based on an embedding of disjunctive logic programs (without classical negation) into causal theories, and on the investigation of the complexity of disjunctive logic programs in [EG93]. We will describe the reduction first.

Consider a Boolean signature σ. A *disjunctive logic program* over σ is a set of *(dlp) rules* of the form

$$a_1; \cdots ; a_k \leftarrow b_1, \ldots, b_m, \textit{not } c_1, \ldots, \textit{not } c_n \tag{1.76}$$

where $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n \in \sigma,\ k \geq 1,\ m, n \geq 0$. Let us identify each dlp rule (1.76) with the causal rule

$$b_1 \wedge \cdots \wedge b_m \supset a_1 \vee \cdots \vee a_k \Leftarrow \neg c_1 \wedge \cdots \wedge \neg c_n \,. \tag{1.77}$$

Let P be a disjunctive logic program over σ. If we agree to identify an interpretation I of σ with the set $\{c \in \sigma : I(c) = \mathsf{t}\}$, then we can say that I is an *answer set* for P if I is a minimal model of P^I. Although this is an unusual way of stating the definition of an answer set, its equivalence to the standard definition from [GL91] (in the absence of classical negation) is easily verified.

Lemma *Let P be a disjunctive logic program over σ. An interpretation I of σ is an answer set for P iff I is a model of the causal theory*

$$P \cup \{\, \neg c \Leftarrow \neg c \,:\, c \in \sigma \,\}\,.$$

This lemma too is easily verified. It is similar to an embedding of disjunctive programs into default logic due to Sakama and Inoue [SI93].

Proposition 3 *The problem of determining that a finite causal theory is consistent is Σ_2^P-complete.*

Proof We first establish Σ_2^P membership. Let T be a causal theory. If we know that $I \models T^I$, then it is an NP problem to determine that I is not a model of T: simply guess an interpretation J, and verify (in polynomial time) that $J \neq I$ and $J \models T^I$. Hence determining that T is consistent belongs to Σ_2^P: guess an interpretation I, verify (in polynomial time) that $I \models T^I$, and then use an NP-oracle to determine that I is a model of T.

Σ_2^P-hardness (for causal theories with Boolean signatures) follows from the lemma above, since the problem of existence of an answer set for a finite disjunctive logic program is Σ_2^P-hard [EG93, Corollary 3.8].

1.9.4 Proof of Proposition 4

Proposition 4 *(i) Replacing a rule*

$$F \wedge G \Leftarrow H \tag{1.78}$$

in a causal theory by the rules

$$F \Leftarrow H,\ G \Leftarrow H \tag{1.79}$$

does not change its models. (ii) Replacing a rule

$$F \Leftarrow G \vee H \tag{1.80}$$

in a causal theory by the rules

$$F \Leftarrow G,\ F \Leftarrow H \tag{1.81}$$

does not change its models.

Proof Let causal theory T_r be obtained from a causal theory T by replacing a rule of form (1.78) with the rules (1.79). Take any interpretation I, and consider two cases.

Case 1: $I \models H$. Then rule (1.78) guarantees that $F \wedge G \in T^I$, while rules (1.79) guarantee that $F, G \in T_r^I$. Otherwise T^I and T_r^I are identical. Hence I is the unique model of T^I iff I is the unique model of T_r^I.

Case 2: $I \not\models H$. Then $T^I = T_r^I$.

In both cases, I is a model of T iff I is a model of T_r.

Now let T_r be obtained from a causal theory T by replacing a rule of form (1.80) with the rules (1.81). For any interpretation I, $T^I = T_r^I$, and so T and T_r have the same models.

1.9.5 Proof of Proposition 5

Proposition 5 *Let T_1 and T_2 be causal theories of a signature σ, such that every rule in T_2 is a constraint. An interpretation of σ is a model of $T_1 \cup T_2$ iff it is a model of T_1 and does not violate any of the constraints T_2.*

Proof Let I be an interpretation of σ. Consider two cases.

Case 1: I violates a constraint of T_2. Then $T_2^I = \{\bot\}$ and so

$$I \not\models T_1^I \cup T_2^I = (T_1 \cup T_2)^I.$$

Hence I is not a model of $T_1 \cup T_2$, which proves the claim in this case.

Case 2: I violates no constraint of T_2. Then $T_2^I = \emptyset$, so

$$T_1^I = T_1^I \cup T_2^I = (T_1 \cup T_2)^I.$$

Consequently $T_1 \cup T_2$ and T_1 have the same models, which proves the claim in this case.

1.9.6 Proof of Proposition 6

Proposition 6 *The models of a definite causal theory are precisely the models of its completion.*

Proof Let T be a definite causal theory. Assume that I is a model of T. It follows that I violates no constraint of T, and thus satisfies every formula in the completion of T that is obtained from a constraint. It remains to show that I satisfies the completion formula for every nontrivial atom A. Consider two cases.

Case 1: $A \in T^I$. Since T is definite and $I \models T^I$, T^I is a set of atoms true in I. So I satisfies A, which is the left-hand side of the completion formula for A. Since $A \in T^I$, there is a rule with head A whose body is true in I. Hence I also satisfies the right-hand side of the completion formula for A.

Case 2: $A \notin T^I$. So there is no rule in T with head A whose body is true in I, which shows that I doesn't satisfy the right-hand side of the completion formula for A. It remains to show that $I \not\models A$. Since T^I is a set of atoms whose unique model is I, every nontrivial atom true in I belongs to T^I. Since A is a nontrivial atom that doesn't belong to T^I, we can conclude that A is false in I.

Proof in the other direction is similar.

1.9.7 Proof of Proposition 7

Proposition 7 *For any transition $\langle s, e, s'\rangle$, s and s' are states.*

Proof Let $X = 0\!:\!s \cup 0\!:\!e \cup 1\!:\!s'$ be a model of D_1. We need to show that $0\!:\!s$ and $0\!:\!s'$ are models of D_0. By $i\!:\!\sigma^{fl}$ we denote the set of constants of the form $i\!:\!c$ where $c \in \sigma^{fl}$.

To show that $0 : s$ is a model of D_0, observe that D_0 is the part of D_1 consisting of rules (1.50) for static laws with $i = 0$ and rules (1.52). The reduct D_0^X is a set of formulas over $0\!:\!\sigma^{fl}$ and every formula from D_1^X with a constant from $0\!:\!\sigma^{fl}$ belongs to D_0^X. Since

X is the unique model of D_1^X, we can conclude that $0\!:\!s$ is the unique model of D_0^X. But $D_0^X = D_0^{0:s}$, so that $0\!:\!s$ is a model of D_0.

Next we show that $0\!:\!s'$ is a model of D_0. Let T be the part of D_1 consisting of rules (1.50) for static laws with $i = 1$, rules (1.50) for action dynamic laws with $i = 0$, and rules (1.51) with $i = 0$. Let $\Gamma = T^X$. It is straightforward to verify that Γ is a set of formulas over $1\!:\!\sigma^{fl}$ and that every formula from D_1^X with a constant from $1\!:\!\sigma^{fl}$ belongs to Γ. Since X is the unique model of D_1^X, we can conclude that $1\!:\!s'$ is the unique model of Γ. Let Γ_0 be the set of formulas over $0 : \sigma^{fl}$ obtained from Γ by replacing each time stamp $1\!:$ with $0\!:$. Then $0\!:\!s'$ is the unique model of Γ_0. We need to show that $0\!:\!s'$ is also the unique model of $D_0^{0:s'}$. Observe first that every formula in $D_0^{0:s'}$ that does not belong to Γ_0 is an atom from $0 : s'$ that came to the reduct from one of the rules (1.52) of D_0. Hence $0 : s'$ satisfies $D_0^{0:s'}$. Due to the presence of rules (1.52) in D_0, any interpretation that satisfies $D_0^{0:s'}$ must agree with $0\!:\!s'$ on simple fluent constants. On the other hand, the formulas in Γ_0 that do not belong to $D_0^{0:s'}$ do not contain statically determined constants, because their counterparts in Γ came from the heads of dynamic laws. Consequently any interpretation that satisfies $D_0^{0:s'}$ must agree with $0 : s'$ on statically determined fluent constants. It follows that $0\!:\!s'$ is the unique model of $D_0^{0:s'}$, so that $0\!:\!s'$ is a model of D_0.

1.9.8 Proof of Proposition 8

Proposition 8 *For any $m > 0$, an interpretation (1.54) of the signature of D_m is a model of D_m iff each triple $\langle s_i, e_i, s_{i+1}\rangle$ $(0 \le i < m)$ is a transition.*

Proof We understand the notation $i : \sigma^{fl}$ as in the previous proof, and the meaning of $i\!:\!\sigma^{act}$ is similar.

Take a model X of D_m, represent it in the form (1.54), and take any $j \in \{0, \dots, m-1\}$. We need to show that $0\!:\!s_j \cup 0\!:\!e_j \cup 1\!:\!s_{j+1}$ is a model of D_1.

Let T be the subset of D_m consisting of rules (1.50) for static laws with $i = j + 1$, rules (1.50) for action dynamic laws with $i = j$, and rules (1.51) with $i = j$. Let $\Gamma = T^X$. It is straightforward to verify that Γ is a set of formulas over $j\!:\!\sigma^{act} \cup\ j\!+\!1\!:\!\sigma^{fl}$, and that every formula from D_m^X with a constant from $j\!:\!\sigma^{act} \cup\ j\!+\!1\!:\!\sigma^{fl}$ belongs to Γ. Since X is the unique model of D_m^X, it follows that $j\!:\!e_j \cup\ j\!+\!1\!:\!s_{j+1}$ is the unique model of Γ. Let Γ_0 be the set of formulas over $0\!:\!\sigma^{act} \cup\ 1\!:\!\sigma^{fl}$ obtained from Γ by replacing $j :$ with $0 :$ and $j + 1\!:$ with $1\!:$. Then $0\!:\!e_j \cup\ 1\!:\!s_{j+1}$ is the only interpretation of $0\!:\!\sigma^{act} \cup\ 1\!:\!\sigma^{fl}$ that satisfies Γ_0.

The proof of the previous proposition is easily adapted to show that s_j is a state, which is to say that $0 : s_j$ is a model of D_0. That is, $0 : s_j$ is the unique model of $D_0^{0:s_j} = D_0^{0:s_j \cup\ 0:e_j \cup\ 1:s_{j+1}}$.

It remains to observe that $D_0^{0:s_j \cup\ 0:e_j \cup\ 1:s_{j+1}} \cup \Gamma_0 = D_1^{0:s_j \cup\ 0:e_j \cup\ 1:s_{j+1}}$, so that $0\!:\!s_j \cup 0\!:\!e_j \cup 1\!:\!s_{j+1}$ is the unique model of this set of formulas, and, consequently, a model of D_1.

For the other direction, assume that, for each $j \in \{0, \ldots, m-1\}$, the triple $\langle s_j, e_j, s_{j+1}\rangle$ is a transition. We need to show that the corresponding interpretation X of form (1.54) is a model of D_m.

For each j, let T_j be the subset of D_m consisting of rules (1.50) for static laws with $i = j+1$, rules (1.50) for action dynamic laws with $i = j$, and rules (1.51) with $i = j$. Notice that $D_m = D_0 \cup T_0 \cup \cdots \cup T_{m-1}$. Let $\Gamma_j = T_j^X$. Of course $D_m^X = D_0^X \cup \Gamma_0 \cup \cdots \cup \Gamma_{m-1}$.

For any such j, since $\langle s_j, e_j, s_{j+1}\rangle$ is a transition, $0\!:\!s_j \cup 0\!:\!e_j \cup 1\!:\!s_{j+1}$ is the unique model of $D_1^{0:s_j\,\cup\,0:e_j\,\cup\,1:s_{j+1}} = D_0^{0:s_j} \cup T_0^{0:s_j\,\cup\,0:e_j\,\cup\,1:s_{j+1}}$. Reasoning much as before, it follows that $0\!:\!e_j \cup 1\!:\!s_{j+1}$ is the unique model $T_0^{0:s_j\,\cup\,0:e_j\,\cup\,1:s_{j+1}}$. This is equivalent to saying that $j\!:\!e_j \cup j{+}1\!:\!s_{j+1}$ is the unique model $T_j^{j:s_j\,\cup\,j:e_j\,\cup\,j+1:s_{j+1}} = \Gamma_j$.

From the previous proposition, we can conclude also that $0 : s_0$ is the unique model of D_0^X.

Finally, since $D_m^X = D_0^X \cup \Gamma_0 \cup \cdots \cup \Gamma_{m-1}$, we can conclude that X is the unique model of D_m^X, and thus a model of D_m.

1.9.9 Proof of Proposition 9

Proposition 9 *If*

(i) every constant in R is statically determined, and

(ii) for every causal law in D that contains a constant from R in the head, all constants occurring in this law belong to R,

then the mapping $I \mapsto I^R$ is a 1-1 correspondence between the models of D_m and the models of D_m^R.

Proof Using the fact that $s'(c) = s(c)$ for every $c \in R$ and for every transition $\langle s, e, s'\rangle$, it is easy to verify that if I is a model of D_m, then I^R is a model of D_m^R. In other words, the mapping $I \mapsto I^R$ is a function from the set of models of D_m into the set of models of D_m^R. It is also easy to see that the function is 1-1. It remains to show that the function is onto.

Take any model J of D_m^R. Define the interpretation I of the signature of D_m as follows:

$$I(i\!:\!c) = \begin{cases} J(c) & \text{if } c \in R, \\ J(i\!:\!c) & \text{otherwise.} \end{cases}$$

Then

$$I(0\!:\!c) = \cdots = I(m\!:\!c) \tag{1.82}$$

for all $c \in R$, and $I^R = J$. We will check that I is a model of D_m, that is to say, the only model of D_m^I.

For any formula F of the signature of D_m, let F^R be the result of dropping the time stamps in front of all constants from R in F. It is easy to see that I satisfies a formula F iff J satisfies F^R. It follows that

$$(D_m^R)^J = \{F^R : F \in D_m^I\}. \tag{1.83}$$

Since J is a model of $(D_m^R)^J$, it follows that I is a model of D_m^I. It remains to show that D_m^I has no other models.

Take any model I_1 of D_m^I. By (1.83), I_1^R is a model of $(D_m^R)^J$. Since J is the only model of $(D_m^R)^J$, $I_1^R = J$. Let i_0 be a number from $\{0, \dots, m\}$. Define the interpretation I_2 of the signature of D_m as follows:

$$I_2(i\!:\!c) = \begin{cases} I_1(i_0\!:\!c) & \text{if } c \in R, \\ I_1(i\!:\!c) & otherwise. \end{cases}$$

We want to show that I_2 is a model of D_m^I as well. Since I_1 is a model of D_m^I, I_2 satisfies all formulas from D_m^I that do not contain constants from R. Consider a formula from D_m^I that contains at least one constant from R. Since all constants in R are statically determined, this formula has the form $i : F$ for some static causal law (1.48) in D and some time stamp i such that I satisfies $i : G$. By condition (ii), all constants occurring in F, G belong to R. For every constant c from R, I assigns to $i\!:\!c$ and $i_0\!:\!c$ the same value; consequently, I satisfies $i_0 : G$, so that $i_0 : F$ belongs to D_m^I. It follows that I_1 satisfies $i_0 : F$. Since I_2 does not differ from I_1 on the constants occurring in this formula, it follows that I_2 satisfies $i_0 : F$. For every constant c from R, I_2 assigns to $i\!:\!c$ and $i_0\!:\!c$ the same value; consequently, I_2 satisfies $i : F$. We have established that I_2 indeed satisfies D_m^I.

In view of this fact, it follows from (1.83) that I_2^R satisfies $(D_m^R)^J$. We can conclude that $I_2^R = J$. So, for every $c \in R$, $I_2(0 : c) = J(c)$, and consequently $I_1(i_0 : c) = J(c)$. Since i_0 was arbitrary, we have proved that

$$I_1(0\!:\!c) = \dots = I_1(m\!:\!c) \tag{1.84}$$

for every $c \in R$. And since $I^R = J = I_1^R$, it follows from (1.82) and (1.84) that $I_1 = I$. This shows that D_m^I has no models other than I.

1.9.10 Proof of Proposition 10

A *(propositional) default theory* is a set of *defaults* of the form

$$\frac{F : G_1, \dots, G_k}{H} \tag{1.85}$$

where each of $F, G_1, \dots, G_k, H$ is a classical propositional formula and $k \geq 0$. If formula F in (1.85) is $\top$, it is often omitted. If in addition $k = 0$, we identify the default (1.85) with the formula H.

Let D be a default theory and E a logically closed set of classical formulas. By D^E we denote the *reduct* of D relative to E, defined as follows.

$$D^E = \left\{ \frac{F}{H} : \text{for some default (1.85) in } D\text{, none of } \neg G_1, \dots, \neg G_k \text{ belong to } E \right\}$$

E is an *extension* for D if E is the least logically closed set of classical formulas such that for every $\frac{F}{H} \in D^E$ if $F \in E$ then $H \in E$.

Where appropriate below, we identify an interpretation I with the set of formulas true in it, and we identify a causal rule $F \Leftarrow G$ with the default $\frac{:G}{F}$.

Proposition 10 *Let T be a causal theory of a Boolean signature. An interpretation I is a model of T iff I is an extension for T in the sense of default logic.*

Proof Let I be an interpretation. Notice that for any formula G, $I \models G$ iff $\neg G \notin I$. It follows that the reduct of causal theory T relative to I is the same as the reduct of default theory T relative to I. It remains to observe that I is the unique model of T^I iff I is the least logically closed set of formulas that contains T^I, that is to say, iff I is an extension for T.

1.9.11 Proof of Proposition 11

Here we are concerned with a different class of logic programs than in the proof of Proposition 3. These programs consist of rules of the form (1.74). For such programs, there is a well-known correspondence between answer sets and the extensions of a corresponding default theory, obtained by replacing each logic programming rule (1.74) with the default

$$\frac{: l_1, \dots, l_n}{l_0}$$

[GL91, Section 5]. We are interested in a special case of this correspondence. For convenience, where appropriate, we identify an interpretation with either the set of literals true in it or the set of formulas true in it. Under this convention, we can observe that an interpretation I is an answer set for a logic program whose rules have form (1.74) iff I is an extension for the corresponding default theory.

Proposition 11 *Let T be a causal theory whose rules have the form (1.73). An interpretation I is a model of T iff I is an answer set for T in the sense of logic programming.*

Proof In light of the previous observation, Proposition 11 follows from Proposition 10.

Chapter 2

Languages for specifying planning goals

2.1 Introduction

Several applications (e.g., space and robotics) often require planning in non-deterministic domains and for extended goals: actions are modeled with different outcomes that cannot be predicted at planning time, and goals are not simply states to be reached ("reachability goals"), but also conditions on the whole plan execution paths ("extended goals"). This planning problem is challenging. It requires planning algorithms that take into account the possibly different action outcomes, and that deal in practice with the explosion of the search space. Moreover, goals should take into account non-determinism and express behaviors that should hold for all paths and behaviors that should hold only for some paths.

The work described in [PT01] and [PBT01] proposes an approach to this problem, and shows that the approach is theoretically founded (a formal notion of when a plan satisfies an extended goal is provided) and practical (an implementation of the planning algorithm is experimentally evaluated on a parametric domain). In those papers, CTL is used as the language for expressing goals. CTL [Eme90] is a well-known and widely used language for the formal verification of properties by model checking. It provides the ability to express temporal behaviors that take into account the non-determinism of the domain. In CTL, this is done by means of a universal quantifier (A) and of an existential quantifier (E). The former requires a property to hold in all the execution paths, the latter in at least one. However, in many practical cases, we found CTL inadequate for planning, since it cannot express different kinds of goals that are relevant for applications in non-deterministic domains. Consider, for instance, the classical goal of reaching a state that satisfies a given condition p. If the goal should be reached in spite of non-determinism ("strong reachability"), we can use the CTL formula $\mathrm{AF}\, p$, intuitively meaning that for all its execution paths (A), the plan will finally (F) reach a state where p holds. As an example, we could strongly require that a mobile robot reaches a given room. However, in many applications, strong reachability cannot be satisfied. In this case, a reasonable requirement for a plan is that *"it should do everything that is possible to achieve a given condition"*. For instance, the robot should actually try – "do its best" – to reach the room. Unfortunately, in CTL it is impossible to express such goal: the weaker condition imposed by the existential path quantifier $\mathrm{EF}\, p$ only requires that the plan has one of the many possible executions that reach a state where p holds. Moreover, if we have plans that actually try to achieve given goals, their executions should raise a failure on the paths in which the goals are not reachable anymore. Goals should be able to express conditions for controlling and recovering from failure. For instance, a reasonable goal for a robot can be "try to reach a room and, in case of failure, do reach a different room". Again, CTL offers the possibility to compose goals with disjunctions, like $\mathrm{EF}\, p \vee \mathrm{AF}\, q$. However, this formula does not capture the idea of failure recovery: there is no preference in which is the main goal to be achieved (e.g., $\mathrm{EF}\, p$), and which is the recovery goal (e.g., $\mathrm{AF}\, q$) to be achieved just in case of failure.

Here we define a new goal language designed to express extended goals for planning in non-deterministic domains. The language has been motivated from a set of applications we have been involved in (see, e.g., [ACG$^+$01]). The language provides basic goals for expressing conditions that the system should *guarantee* to reach or maintain, and conditions that the system should *try* to reach or maintain. Basic goals can be concatenated by taking into account possible failures. For instance, it is possible to specify goals that we should achieve as a reaction to failure, and goals that we should repeat to achieve until a failure occurs.

We extend the planning algorithm described in [PT01, PBT01] to deal with the new language and implement it in the planner MBP [BCP$^+$01a]. We evaluate the algorithm on the domain described in [PBT01]. Given the new goal language, the algorithm generates better plans than those generated for CTL goals, maintaining a comparable performance.

2.2 Background on Planning for Extended Goals

We review some basic definitions presented in [PT01, PBT01].

Definition 1 *A (non-deterministic)* planning domain $\mathcal{D}$ *is a tuple* $(\mathcal{B}, \mathcal{Q}, \mathcal{A}, \rightarrow)$*, where* $\mathcal{B}$ *is the finite set of (basic) propositions,* $\mathcal{Q} \subseteq 2^{\mathcal{B}}$ *is the set of states,* $\mathcal{A}$ *is the finite set of actions, and* $\rightarrow\ \subseteq \mathcal{Q} \times \mathcal{A} \times \mathcal{Q}$ *is the transition relation, that describes how an action leads from one state to possibly many different states. We write* $q \xrightarrow{a} q'$ *for* $(q, a, q') \in \rightarrow$*.*

We require that relation $\rightarrow$ is total, i.e., for every $q \in \mathcal{Q}$ there is some $a \in \mathcal{A}$ and $q' \in \mathcal{Q}$ such that $q \xrightarrow{a} q'$. We denote with $\mathrm{Act}(q) \triangleq \{a : \exists q'.\ q \xrightarrow{a} q'\}$ the set of the actions that can be performed in state q, and with $\mathrm{Exec}(q, a) \triangleq \{q' : q \xrightarrow{a} q'\}$ the set of the states that can be reached from q performing action $a \in \mathrm{Act}(q)$.

A simple domain is shown in Fig. 2.1. It consists of a building of five rooms, namely

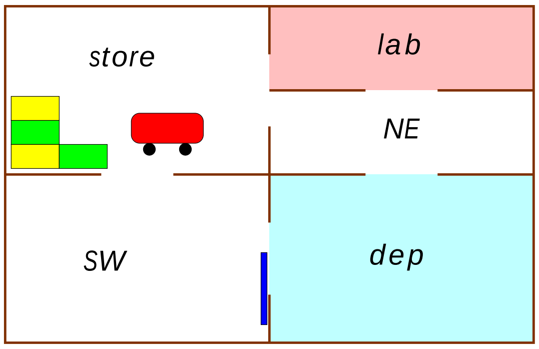


Figure 2.1: A simple navigation domain

a $store$, a department dep, a laboratory lab, and two passage rooms SW and NE. A robot can move between the rooms. The intended task of the robot is to deliver objects from the store to the department, avoiding the laboratory, that is a dangerous room. For the sake of simplicity, we do not model explicitly the objects, but only the movements of the robot. Between rooms SW and dep, there is a door that the robot cannot control. Therefore, an $east$ action from room SW successfully leads to room dep only if the door is open. Another non-deterministic outcome occurs when the robot tries to move $east$ from the $store$: in this case, the robot may end non-deterministically either in room NE or in room lab. The transition graph for the domain is represented in Fig. 2.2. For all states in the domain, we assume to have an action *wait* (not represented in the figure) that leaves the state unchanged.

In order to deal with extended goals, plans allow for specifying actions to be executed that depend not only on the current state of the domain, but also on the *execution context*, i.e., on an "internal state" of the executor, which can take into account, e.g., previous execution steps.

Definition 2 *A* plan *for a domain $\mathcal{D}$ is a tuple $\pi = (C, c_0, act, ctxt)$, where C is a set of contexts, $c_0 \in C$ is the initial context, $act : \mathcal{Q} \times C \rightharpoonup \mathcal{A}$ is the action function, and $ctxt : \mathcal{Q} \times C \times \mathcal{Q} \rightharpoonup C$ is the context function.*

If we are in state q and in execution context c, then $act(q, c)$ returns the action to be executed by the plan, while $ctxt(q, c, q')$ associates to each reached state q' the new execution context. Functions *act* and *ctxt* may be partial, since some state-context pairs are never reached in the execution of the plan. For instance, consider the plan π_1:

$$\begin{array}{ll}
act(store, c_0) = south & ctxt(store, c_0, SW) = c_0 \\
act(SW, c_0) = east & ctxt(SW, c_0, dep) = c_0 \\
 & ctxt(SW, c_0, SW) = c_1 \\
act(dep, c_0) = wait & ctxt(dep, c_0, dep) = c_0 \\
act(SW, c_1) = north & ctxt(SW, c_1, store) = c_1 \\
act(store, c_1) = south & ctxt(store, c_1, SW) = c_1
\end{array}$$

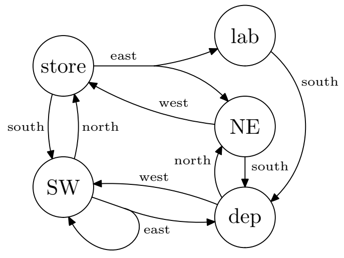


Figure 2.2: The transition graph of the navigation domain

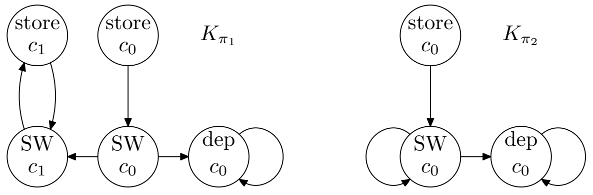


Figure 2.3: Two examples of execution structures

According to this plan, the robot moves south, then it moves east; if dep is reached, it waits there forever; otherwise it continues to move north and south forever.

The executions of a plan can be represented by an *execution structure*, i.e, a Kripke Structure [Eme90] that describes all the possible transitions from (q, c) to (q', c') that can be triggered by executing the plan.

Definition 3 *The* execution structure *of plan π in a domain $\mathcal{D}$ from state q_0 is the structure $K = \langle S, R, L \rangle$, where:*

- $S = \{(q, c) : act(q, c)$ *is defined*$\}$,
- $R = \{\big((q, c), (q', c')\big) \in S \times S : q \xrightarrow{a} q$ *with* $a = act(q, c)$ *and* $c' = ctxt(q, c, q')\}$
- $L(q, c) = \{b : b \in q\}$.

The execution structure K_{π_1} of plan π_1 is shown in Fig. 2.3.

Extended goals in [PT01, PBT01] are expressed with CTL formulas, and the standard semantics for CTL [Eme90] is used to define when a goal g is true in a state (q, c) of the execution structure K (written $(q, c) \models g$).

Definition 4 *Let π be a plan for domain $\mathcal{D}$ and K be the corresponding execution structure. Plan π satisfies goal g from initial state q_0, written $\pi, q_0 \models g$, if $(q_0, c_0) \models g$. Plan π satisfies goal g from the set of initial states Q_0 if $\pi, q_0 \models g$ for each $q_0 \in Q_0$.*

For instance, goal $\mathrm{AG}\,\neg lab \wedge \mathrm{EF}\, dep$ ("the robot should never enter the lab and should have a possibility of reaching the dep") is satisfied by plan π_1 from the initial state $store$.

2.3 The EaGLe Goal Language

2.3.1 Syntax and Semantics

We propose the following language that overcomes some main limitations of CTL as a goal language. Let $\mathcal{B}$ be the set of *basic propositions*. The *propositional formulas* $p \in \mathcal{P}rop$ and the *extended goals* $g \in \mathcal{G}$ over $\mathcal{B}$ are defined as follows:

$$\begin{aligned} p &:= \top \mid \bot \mid b \mid \neg p \mid p \wedge p \mid p \vee p \\ g &:= p \mid g \textbf{ And } g \mid g \textbf{ Then } g \mid g \textbf{ Fail } g \mid \textbf{Repeat } g \mid \\ &\quad\ \textbf{DoReach } p \mid \textbf{TryReach } p \mid \textbf{DoMaint } p \mid \textbf{TryMaint } p \end{aligned}$$

We now provide some intuitions and motivations for this language (and illustrate the intended meaning of the constructs on the example in Fig. 2.2). We often need a plan that is guaranteed to achieve a given goal. This can be done with operators **DoReach** (that specifies a property that should be reached) and **DoMaint** (that specifies a property that should be maintained true). However, in several domains, no plans exist that satisfy these strong goals. This is the case, for instance, for the goal **DoMaint** $\neg lab$ **And DoReach** dep, if the robot is initially in $store$. We need therefore to weaken the goal, but at the same time we want to capture its *intentional aspects*, i.e., we require that the plan *"does everything that is possible"* to achieve it. This is the intended meaning of **TryReach** and **TryMaint**. These operators are very different from operators EF and EG in CTL, which require plans that have just a possibility to achieve the goal. In the example, consider the goal **DoMaint** $\neg lab$ **And TryReach** dep. According to our intended meaning, this goal is not satisfied by plan π_1, that just tries one time to go east. Instead, the goal is satisfied by a plan π_2 that first moves the robot $south$ to room SW, and then keeps trying to go $east$, until this action succeeds and the dep is reached. CTL goal $\mathrm{AG}\,\neg lab \wedge \mathrm{EF}\, dep$ is satisfied by both plans π_1 and π_2: indeed, the execution structures corresponding to these plans (see Fig. 2.3) have a path that reaches dep. Plan π_1 is not valid if we specify CTL goal $\mathrm{AG}\,\neg lab \wedge \mathrm{AG}\,\mathrm{EF}\, dep$, that requires that there is always a path that leads to dep (this is a "strong cyclic" reachability goal). Also in this case, however, the intentional aspect is lost; an acceptable plan for the CTL goal (but not for the original goal) is the one that tries one time to go $east$ and, if this move fails, goes back to the $store$ before trying again.

In several applications, goals should specify reactions to failures. For instance, a main goal for a mobile robot might be to continue to deliver objects to a given room, but when this is impossible, the robot should not give up, but recover from failure by delivering the current object to a next room. Consider, for instance, the goal

$$(\textbf{TryMaint } \neg lab \textbf{ Fail DoReach } store) \textbf{ And DoReach } dep.$$

In order to satisfy it from the $store$, the robot should first go $east$. If room lab is entered after this move, then goal **TryMaint** $\neg lab$ fails, and the robot has to go back to the $store$ to satisfy recovery goal **DoReach** $store$. This is very different from CTL goal

$\mathrm{AG}\,(lab \rightarrow \mathrm{AF}\,store)$ that requires that if the robot ends in the lab it goes to the store. Indeed, this goal does not specify the fact that the lab should be avoided and that going back to the $store$ is only a recovery goal. The CTL goal is satisfied as well by a plan that "intentionally" leads the robot into the lab, and then takes it back to the $store$.

Construct **Fail** is used to model recovery from failure. Despite its simplicity, this construct is rather flexible. In combination with the other operators of the language, it allows for representing both failures that can be detected at planning time and failures that occurs at execution time. Consider, for instance, goal **DoReach** dep **Fail DoReach** $store$. Sub-goal **DoReach** dep requires to find a plan that is guaranteed to reach room dep. If such a plan exists from a given state, then sub-goal **DoReach** dep cannot fail, and the recovery goal **DoReach** $store$ is never considered. If there is no such plan, then sub-goal **DoReach** dep cannot be satisfied from the given state and the recovery goal **DoReach** $store$ is tried instead. In this case, the failure of the primary goal **DoReach** dep in a given state can be decided *at planning time*. Consider now the goal

TryReach dep **Fail DoReach** $store.$

In this case, sub-goal **TryReach** dep requires to find a plan that *tries* to reach room dep. During the execution of the plan, a state may be reached from which it is not possible to reach the dep. When such a state is reached, goal **TryReach** dep fails and the recovery goal **DoReach** $store$ is considered. That is, in this case, the failure of the primary goal **TryReach** dep is decided *at execution time*.

We remark that the goal language does not allow for arbitrary nesting of temporal operators, e.g., it is not possible to express goals like **DoReach** (**TryMaint** p). However, it provides goal **Repeat** g, that specifies that sub-goal g should be achieved cyclically, until it fails. In our experience, the **Repeat** construct replaces most of the usages of nesting.

We now provide a formal semantics for the new goal language that captures the desired intended meaning. Due to the features of the new language, the semantics of goals cannot be defined following the approach used for CTL [Eme90], that associates to each formula the set of states that satisfy it. Instead, for each goal g, we associate to each state s of the execution structure two sets $\mathcal{S}_g(s)$ and $\mathcal{F}_g(s)$ of finite paths in the execution structure. They represent the paths that lead to a success or to a failure in the achievement of goal g from state s. We say that an execution structure satisfies a goal g from state s_0 if $\mathcal{F}_g(s_0) = \emptyset$, that is, if no failure path exists for the goal. Let us consider, for instance, the case of goal $g =$ **TryReach** dep. Set $\mathcal{S}_g(s)$ contains the paths that lead from s to states where dep holds, while set $\mathcal{F}_g(s)$ contains the paths that lead from s to states where dep is not reachable anymore. In the case of execution structure K_{π_1} of Fig. 2.3, we have $\mathcal{S}_g(store, c_0) = \{((store, c_0), (SW, c_0), (dep, c_0))\}$ and $\mathcal{F}_g(store, c_0) = \{((store, c_0), (SW, c_0), (SW, c_1))\}$. We remark that we do not consider $((store, c_0), (SW, c_0), (dep, c_0), (dep, c_0))$ as a success path, since its prefix

$$((store, c_0), (SW, c_0), (dep, c_0))$$

is already a success path. In the case of execution structure K_{π_2} of Fig. 2.3, $\mathcal{F}_g(store, c_0) = \emptyset$, while $\mathcal{S}_g(store, c_0)$ contains all paths $((store, c_0), (SW, c_0), \ldots, (SW, c_0), (dep, c_0))$; that is, we take into account that a success path may stay in (SW, c_0) for an arbitrarily number of steps before the dep is eventually reached. Paths

$$((store, c_0), (SW, c_0), \ldots, (SW, c_0))$$

are neither success paths nor failure paths: indeed, condition dep is never satisfied along these paths, but they can be extended to success paths. According to this semantics, goal **TryReach** dep is not satisfied by execution structure K_{π_1}, while it is satisfied by execution structure K_{π_2}.

Some notations on paths are now in order. We use ρ to denote infinite paths and σ to denote finite paths in the execution structure. We represent with $\text{first}(\sigma)$ the first state of path σ and with $\text{last}(\sigma)$ the last state of the path. We write $s \in \sigma$ if state s appears in σ. We represent with $\sigma; \sigma'$ the path obtained by the concatenation of σ and σ'; concatenation is defined only if $\text{last}(\sigma) = \text{first}(\sigma')$. We write $\sigma \leq \sigma'$ if σ is a prefix of σ', i.e., if there is some path σ'' such that $\sigma; \sigma'' = \sigma'$. As usual, we write $\sigma < \sigma'$ if $\sigma \leq \sigma'$ and $\sigma \neq \sigma'$. Finally, given a set Σ of finite paths, we define the set of minimal paths in Σ as follows: $\min(\Sigma) \equiv \{\sigma \in \Sigma : \forall \sigma'. \sigma' < \sigma \implies \sigma' \notin \Sigma\}$.

We now define sets $\mathcal{S}_g(s)$ and $\mathcal{F}_g(s)$. The definition is by induction on the structure of g.

Goal $g = p$ is satisfied if condition p holds in the current state, while it fails if condition p does not hold. Formally, if $s \models p$, then $\mathcal{S}_g(s) = \{(s)\}$ and $\mathcal{F}_g(s) = \emptyset$. Otherwise, $\mathcal{S}_g(s) = \emptyset$ and $\mathcal{F}_g(s) = \{(s)\}$.

Goal $g =$ **TryReach** p has success when a state is reached that satisfies p. It fails if condition p cannot be satisfied in any of the future states. Formally, $\mathcal{S}_g(s) = \min\{\sigma : \text{first}(\sigma) = s \wedge \text{last}(\sigma) \models p\}$ and $\mathcal{F}_g(s) = \min\{\sigma : \text{first}(\sigma) = s \wedge \forall s' \in \sigma. s' \not\models p \wedge \forall \sigma' \geq \sigma. \text{last}(\sigma') \not\models p\}$.

Goal $g =$ **DoReach** p requires that condition p is eventually achieved despite non-determinism. If this is the case, then the successful paths are those that end in a state satisfying p. If there is some possible future computation from a state s along which p is never achieved, then goal g fails immediately in s. Formally, if there is some infinite path ρ from s such that $s' \not\models p$ for each $s' \in \rho$, then $\mathcal{S}_g(s) = \emptyset$ and $\mathcal{F}_g(s) = \{(s)\}$. Otherwise, $\mathcal{S}_g(s) = \min\{\sigma : \text{first}(\sigma) = s \wedge \text{last}(\sigma) \models p\}$ and $\mathcal{F}_g(s) = \emptyset$.

Goal $g =$ **TryMaint** p. Maintainability goals express conditions that should hold forever. No finite success path can thus be associated to maintainability goals. On the other hand, these goals may have failure paths. Goal $g =$ **TryMaint** p fails in all those states where condition p does not hold. Formally, $\mathcal{S}_g(s) = \emptyset$ and $\mathcal{F}_g(s) = \min\{\sigma : \text{first}(\sigma) = s \wedge \text{last}(\sigma) \not\models p\}$.

Goal g = **DoMaint** p fails immediately in all those states that do not guarantee that condition p can be maintained forever. Formally, if $s' \models p$ holds for all states s' reachable from s, then $\mathcal{F}_g(s) = \emptyset$. Otherwise $\mathcal{F}_g(s) = \{(s)\}$. In both cases, $\mathcal{S}_g(s) = \emptyset$.

Goal $g = g_1$ **Then** g_2 requires to satisfy sub-goal g_1 first and, once g_1 succeeds, to satisfy sub-goal g_2. Goal g_1 **Then** g_2 succeeds when also goal g_2 succeeds. It fails when either goal g_1 or goal g_2 fail. Formally, $\mathcal{S}_g(s) = \{\sigma_1;\sigma_2 : \sigma_1 \in \mathcal{S}_{g_1}(s) \wedge \sigma_2 \in \mathcal{S}_{g_2}(\mathrm{last}(\sigma_1))\}$ and $\mathcal{F}_g(s) = \{\sigma_1 : \sigma_1 \in \mathcal{F}_{g_1}(s)\} \cup \{\sigma_1;\sigma_2 : \sigma_1 \in \mathcal{S}_{g_1}(s) \wedge \sigma_2 \in \mathcal{F}_{g_2}(\mathrm{last}(\sigma_1))\}$.

Goal $g = g_1$ **Fail** g_2 permits to recover from failure. Sub-goal g_1 is tried first. If it succeeds, then the whole goal g_1 **Fail** g_2 succeeds. If sub-goal g_1 fails, then g_2 is taken into account. Formally, $\mathcal{S}_g(s) = \{\sigma_1 : \sigma_1 \in \mathcal{S}_{g_1}(s)\} \cup \{\sigma_1;\sigma_2 : \sigma_1 \in \mathcal{F}_{g_1}(s) \wedge \sigma_2 \in \mathcal{S}_{g_2}(\mathrm{last}(\sigma_1))\}$ and $\mathcal{F}_g(s) = \{\sigma_1;\sigma_2 : \sigma_1 \in \mathcal{F}_{g_1}(s) \wedge \sigma_2 \in \mathcal{F}_{g_2}(\mathrm{last}(\sigma_1))\}$.

Goal $g = g_1$ **And** g_2 succeeds if both sub-goals succeed, and fails if one of the sub-goals fails. Formally, $\mathcal{S}_g(s) = \min\{\sigma : \exists\sigma_1 \le \sigma.\sigma_1 \in \mathcal{S}_{g_1}(s) \wedge \exists\sigma_2 \le \sigma.\sigma_2 \in \mathcal{S}_{g_2}(s)\}$ and $\mathcal{F}_g(s) = \min(\mathcal{F}_{g_1}(s) \cup \mathcal{F}_{g_2}(s))$.

Goal g = **Repeat** g' requires to achieve goal g in a cyclic way. That is, as soon as goal g' has success, a new instance of goal g' is activated. Goal **Repeat** g' fails as soon as one of the instances of goal g' fails. Formally, $\mathcal{S}_g(s) = \emptyset$ and $\mathcal{F}_g(s) = \{\sigma_0;\sigma_1;\ldots;\sigma_n;\sigma' : \sigma' \in \mathcal{F}_{g'}(\mathrm{first}(\sigma')) \wedge \exists\sigma'_i \in \mathcal{S}_{g'}(\mathrm{first}(\sigma'_i)).\ \sigma_i = \sigma'_i;(\mathrm{last}(\sigma'_i),\mathrm{last}(\sigma_i))\}$.

We can now define the notion of a plan that satisfies a goal expressed in the new language.

Definition 5 *Let π be a plan for domain $\mathcal{D}$ and K be the corresponding execution structure. Plan π satisfies goal $g \in \mathcal{G}$ from initial state q_0, written $\pi, q_0 \models g$, if $\mathcal{F}_g(q_0, c_0) = \emptyset$ for execution structure K. Plan π satisfies goal g from the set of initial states Q_0 if $\pi, q_0 \models g$ for each $q_0 \in Q_0$.*

2.3.2 Outline of the algorithm

In this subsection we extend to the new goal language the symbolic planning algorithm defined in [PBT01] for CTL. The algorithm is based on control automata. They are used to encode the requirements expressed by the goal, and are then exploited to guide the search of the plan. The states of a control automaton define the different goals that the plan intends to achieve in different instants; they correspond to the contexts of the plan under construction. The transitions of the control automaton define constraints on the behaviors that the states of the domain should exhibit in order to be *compatible* with the control states, i.e., to satisfy the corresponding goals. In this subsection, we first define a mapping from the new goal language into (an enriched version of) control automata. The mapping we propose departs substantially from the construction defined in [PBT01].

Then we show how control automata can be used to guide the search of the plan. In the search phase, the algorithms of [PBT01] can be re-used with minor changes.

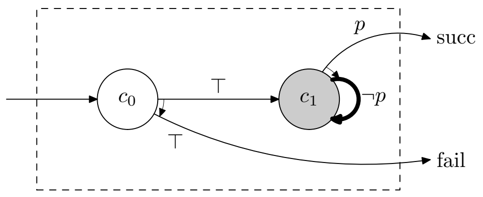
Construction of the control automaton

Definition 6 *A* control automaton *is a tuple* (C, c_0, T, RB)*, where:*

- C *is the set of* control states*;*
- $c_0 \in C$ *is the* initial control state*;*
- $T(c) = \langle t_1, t_2, \ldots, t_m \rangle$ *is the list of* transitions *for control state* c*; each transition* t_i *is either* normal*, in which case* $t_i \in \mathcal{P}rop \times (C \times \{\circ, \bullet\})^*$*; or* immediate*, in which case* $t_i \in \mathcal{P}rop \times (C \cup \{\mathrm{succ}, \mathrm{fail}\})$.
- $RB = \{rb_1, \ldots, rb_n\}$*, with* $rb_i \subseteq C$*, is the set of* red blocks.

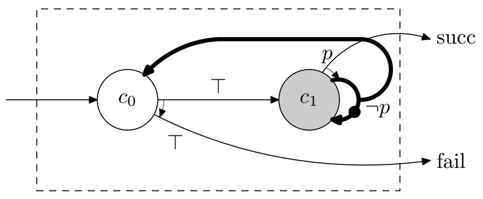
A list of transitions $T(c)$ is associated to each control state c: each transition is a possible behavior that a state can satisfy in order to be compatible with the control state c. The order of the list represents the preference among these transitions. The transitions of a control automaton are either *normal* or *immediate*. The former transitions correspond to the execution of an action in the plan. The latter ones describe updates in the control state that do not correspond to the execution of an action; they resemble ϵ-transitions of classical automata theory. The *normal transitions* are defined by a condition (that is, a formula in $\mathcal{P}rop$) and by a list of target control states. Each target control state is marked either by a $\circ$ or by a $\bullet$. State s satisfies a normal transition $(p, \langle (c'_1, k'_1), \ldots, (c'_n, k'_n) \rangle)$ (with $k'_i \in \{\circ, \bullet\}$) if it satisfies condition p and if there is some action a from s such that: (i) all the next states reachable from s doing action a are compatible with some of the target control states; and (ii) some next state is compatible with each target state marked as $\bullet$. The *immediate transitions* are defined by a condition and by a target control state. A state satisfies an immediate transition (p, c') if it satisfies condition p and if it is compatible with the target control state c'. Special target control states succ and fail are used to represent success and failure: all states are compatible with success, while no state is compatible with failure. The *red blocks* of a control automaton represent sets of control states where the execution cannot stay forever. Typically, a red block consists of the set of control states in which the execution is trying to achieve a given condition, as in the case of a reachability goal. If an execution persists inside such a set of control states, then the condition is never reached, which is not acceptable for a reachability goal. In the control automaton, a red block is used to represent the fact that any valid execution should eventually leave these control states.

We now describe the control automata corresponding to the goal language. Rather than providing the formal definition of the control automata, we represent them using a graphical notation. We start with goal **DoReach** p:



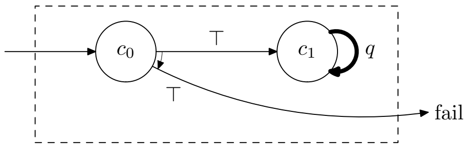
The control automaton has two control states: c_0 (the initial control state) and c_1. There are two transitions leaving control state c_1. The first one, guarded by condition p, is a success transition that corresponds to the cases where p holds in the current domain state. The second transition, guarded by condition $\neg p$, represents the case where p does not hold in the current state, and therefore, in order to achieve goal **DoReach** p, we have to assure that the goal can be achieved in *all the next* states. We remark that the second transition is a normal transition since it requires the execution of an action in the plan; the first transition, instead, is immediate. In the diagrams, we distinguish the two kinds of transitions by using thin arrows for the immediate ones and thick arrows for the normal ones. A domain state is compatible with control state c_1 only if it satisfies goal **DoReach** p, that is, if condition p holds in the current state (first transition from c_1), or if goal **DoReach** p holds in all the next states (second transition from c_1). According to the semantics of **DoReach** p, it is not possible for an execution to stay in state c_1 forever, as this corresponds to the case where condition p is never reached. That is, set $\{c_1\}$ is a red block of the control automaton. In the diagrams, states that are in a red block are marked in gray. Control state c_0 takes into account that it is not always possible to assure that condition p will be eventually reached, and that if this is not the case, then goal **DoReach** p fails. The precedence order among the two transitions from control state c_0, represented by the small circular arrow between them, guarantees that the transition leading to a failure is followed only if it is not possible to satisfy the constraints of control state c_1.

The control automaton for **TryReach** p is the following:



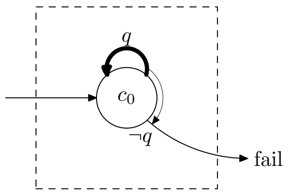
The difference with respect to the control automaton for **DoReach** p is in the transition from c_1 guarded by condition $\neg p$. In this case we do not require that goal **TryReach** p holds for *all* the next states, but only for *some* of them. Therefore, the transition has two possible targets, namely control states c_1 (corresponding to the next states were we expect to achieve **TryReach** p) and c_0 (for the other next states). The semantics of goal **TryReach** p requires that there should be always *at least one* next state that satisfies **TryReach** p; that is, target c_1 of the transition is marked by $\bullet$ in the control automaton. This "non-emptiness" requirement is represented in the diagram with the $\bullet$ on the arrow leading back to c_1. The preferred transition from control state c_0 is the one that leads to c_1. This ensures that the algorithm will try to achieve goal **TryReach** p whenever possible.

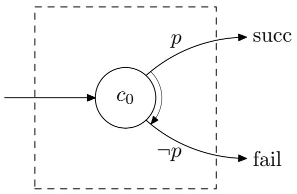
The control automaton for **DoMaint** q is the following:



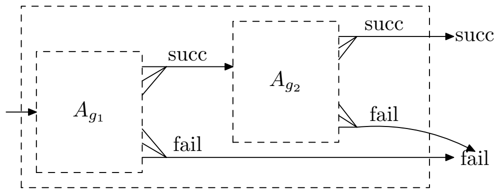
The transition from control state c_1 guarantees that a state s is compatible with control state c_1 only if s satisfies condition q and all the next states of s are also compatible with c_1. Control state c_1 is not gray. Indeed, a valid computation that satisfies **DoMaint** q remains in control state c_1 forever.

The control automaton for the remaining basic goals **TryMaint** q and p are, respectively:

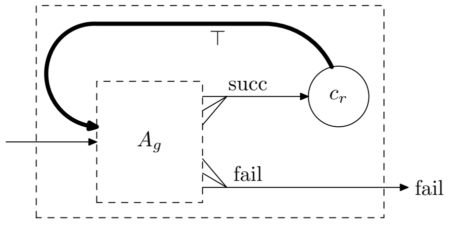




The control automata for the compound goals are defined compositionally, by combining the control automata of their sub-goal. The control automaton for goal g_1 **Then** g_2 is the following:

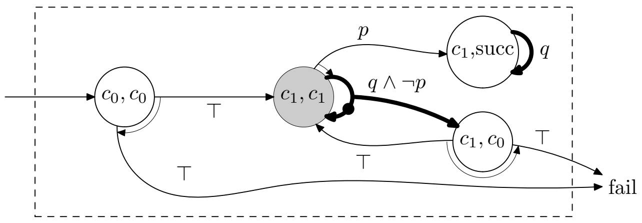


The initial state of the compound automaton coincides with the initial state of automaton A_{g_1}, and the transitions that leave the A_{g_1} with success are redirected to the initial state of A_{g_2}. The control automaton for goal g_1 **Fail** g_2 is defined similarly. The difference is that in this case the transitions that leave A_{g_1} with *failure* are redirected to the initial state of A_{g_2}. The control automaton for goal **Repeat** g is as follows:



A new control state c_r is added to the automaton. This guarantees that, if goal g has been successfully fulfilled in state s, then the achievement of a new instance of goal g starts from the next states, not directly in s.

The control automaton for goal g_1 **And** g_2 has to perform a parallel simulation of the control automata for the two sub-goals, and to accept only those behaviors that are accepted separately by A_{g_1} and by A_{g_2}. Technically, this corresponds to take the *synchronous product* $A_{g_1} \otimes A_{g_2}$ of the control automata for the sub-goals. Here, we skip the formal definition of synchronous product: it is simple, but is long and rather technical, since it has to take into account several special cases. Here, we present a simple example of synchronous product. The control automaton for goal **DoMaint** q **And TryReach** p (i.e., the synchronous product of the control automata A_g for goal **DoMaint** q and $A_{g'}$ for **TryReach** p) is as follows:



The control states of the product automaton are couples (c, c'), where c is a control state of A_g and c' is a control state of $A_{g'}$. The extra control state $(c_1, succ)$ is added to take into account the case where goal **TryReach** p has been successfully fulfilled, but goal **DoMaint** q is still active. The transitions from control states (c, c') are obtained by combining in a suitable way the transitions of A_g from c and the transitions of $A_{g'}$ from c'. For instance, in our example, the normal transition with condition $q \wedge \neg p$ from control state (c_1, c_1) corresponds to the combination of the two normal transitions of the control automata for **DoMaint** q and **TryReach** p.

Plan search

Once the control automaton corresponding to a goal has been built, the planning algorithm performs a search in the domain, guided by the strategy encoded in the control automaton. During this search, a set of domain states is associated to each state of the control automaton. Intuitively, these are the states that are compatible with the control state, i.e., for which a plan exists for the goal encoded in the control state. Initially, all the domain states are considered compatible with all the control states, and this initial assignment is then iteratively refined by discarding those domain states that are recognized incompatible with a given control state. Once a fix-point is reached, the information gathered during the search is exploited in order to extract a plan. We remark that iterative refinement and plan extraction are independent from the goal language. In fact, for them we have been able to re-use the algorithms of [PBT01] with minor changes.

In the iterative refinement algorithm, the following conditions are enforced: **(C1)** a domain state s is associated to a control state c only if s can satisfy the behavior described by some transition of c; and **(C2)** executions from a given state s are not acceptable if they stay forever inside a red block. In each step of the iterative refinement, either a control state is selected and the corresponding set of domain states is refined according to **(C1)**; or a red block is selected and all the sets of domain states associated to its control states are refined according to **(C2)**. The refinement algorithm terminates when no further refinement step is possible, that is, when a fix-point is reached.

The core of the refinement step resides is function $\textit{ctxt-assoc}_A(c, \gamma)$. It takes as input the control automaton $A = (C, c_0, T, RB)$, a control state $c \in C$, and the current association $\gamma : C \to 2^{\mathcal{Q}}$, and returns the new set of domain states to be associated to c. It is defined as follows:

$$\begin{aligned}&\textit{ctxt-assoc}_A(c,\gamma) \triangleq \\ &\qquad \big\{q \in \mathcal{Q} : \exists t \in T(c).\ q \in \textit{trans-assoc}_A(t,\gamma)\big\}.\end{aligned}$$

According to this definition, a state q is compatible with a control state c if it satisfies the conditions of some transition t from that control state. If $t = (p, c')$ is an immediate transition, then:

$$\textit{trans-assoc}_A(t,\gamma) \triangleq \big\{q \in \mathcal{Q} :\ q \models p\ \wedge\ q \in \gamma(c')\big\}.$$

where we assume that $\gamma(\text{fail}) = \emptyset$ and $\gamma(\text{succ}) = \mathcal{Q}$. That is, in the case of an immediate transition, we require that q satisfies property p and that it is compatible with the new control state c' according to the current association γ. If $t = (p, \langle (c'_1, k'_1), \ldots, (c'_n, k'_n)\rangle)$ is a normal transition, then:

$$\begin{aligned}\textit{trans-assoc}_A(t,\gamma) \triangleq \big\{&q \in \mathcal{Q} :\ q \models p\ \wedge\ \exists a \in \text{Act}(q). \\ &(q,a) \in \textit{gen-pre-image}\big((\gamma(c'_1), k'_1), \ldots, (\gamma(c'_n), k'_n)\big)\big\}\end{aligned}$$

where:

$$\begin{aligned}&\textit{gen-pre-image}\big((Q_1,k_1),\ldots(Q_n,k'_n)\big) \triangleq \\ &\qquad \big\{(q,a) : \exists Q'_1 \subseteq Q_1 \ldots Q'_k \subseteq Q_k. \\ &\qquad\qquad \text{Exec}(q,a) = Q'_1 \cup \cdots \cup Q'_k\ \wedge \\ &\qquad\qquad Q'_i \cap Q'_j \text{ if } i \neq j \wedge Q'_i \neq \emptyset \text{ if } k_i = \bullet\big\}.\end{aligned}$$

Also in the case of normal transitions, $\textit{trans-assoc}_A$ requires that q satisfies property p. Moreover, it requires that there is some action a such that the next states $\text{Exec}(q,a)$ satisfy the following conditions: (i) all the next states are compatible with some of the target control states, according to association γ; and (ii) some next state is compatible with each target control state marked as $\bullet$. These two conditions are enforced by requiring that the state-action pair (q,a) appears in the *generalized pre-image* of the sets of states

$\gamma(c'_i)$ associated by γ to the target control states c'_i. The generalized pre-image, that is computed by function *gen-pre-image*, can be seen as a combination and generalization of the *strong-pre-image* and *multi-weak-pre-image* used in [PBT01].

Function *ctxt-assoc* is used in the refinement steps corresponding to **(C1)** as well as in the refinement steps corresponding to **(C2)**. In the former case, the refinement step simply updates $\gamma(c)$ to the value of *ctxt-assoc*$_A(c, \gamma)$. In the latter case, the refinement should guarantee that any valid execution eventually leaves the control states in the selected red block rb. To this purpose, the empty set of domain states is initially associated to the control states in the red block; then, iteratively, one of the control states $c \in rb$ is chosen, and its association $\gamma(c)$ is updated to *ctxt-assoc*$_A(c, \gamma)$; these updates terminate when a fix-point is reached, that is, when $\gamma(c) =$ *ctxt-assoc*$_A(c, \gamma)$ for each $c \in rb$. In this way, a least fix-point is computed, which guarantees that a domain state is associated to a control state in the red block only if there is a plan from that domain state that leaves the red block in a finite number of actions.

Once a stable association γ from control states to sets of domain states is built for a control automaton, a plan can be easily obtained. The contexts for the plan correspond to the states of the control automaton. The information necessary to define functions act and $ctxt$ is implicitly computed during the execution of the refinement steps. Indeed, function *trans-assoc* defines the possible actions $a = act(q, c)$ to be executed in the state-context pair (q, c), namely the actions that satisfy the constraints of one of the normal transitions of the control automaton. Moreover, function *gen-pre-image* defines the next execution context $ctxt(q, c, q')$ for any possible next state $q' \in \mathrm{Exec}(q, a)$. The preference order among the transitions associated to each control state is exploited in this phase, in order to guarantee that the extracted plan satisfies the goal in the best possible way.

Implementation and experimental evaluation

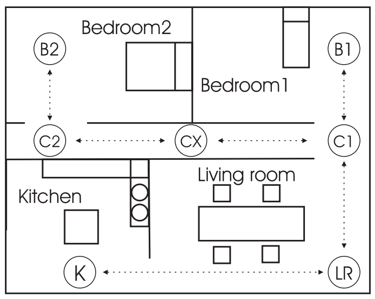
We have implemented the algorithm in the MBP planner [BCP$^+$01a]. In the plan search phase, we exploit the BDD-based symbolic techniques [BCM$^+$92] provided by MBP, thus allowing for a very efficient exploration of large domains. We have done some preliminary experiments for testing the expressiveness of the new goal language and for comparing the performance of the new algorithm w.r.t. the one presented in [PBT01] for CTL goals. For the experiments, we have used the test suite for extended goals made available in the MBP package (see `http://sra.itc.it/tools/mbp`). The tests are based on the robot navigation domain proposed by [KBSD97]. The small domain described in Fig. 2.1 is a very trivial instance of this class of domains. Some of the tests in the suite correspond to domains with more than 10^8 states.

The results of the tests are very positive. We have been able to express all the CTL goals in the test suite using the new goal language. Moreover, in spite of the fact that the domain was proposed for CTL goals, the quality of the generated plans is much higher in the case of the new goal language. For instance, in the case of **TryReach** goals, the plans generated by the new algorithm lead the robot to the goal room avoiding the useless moves that are present in the plans generated from the corresponding CTL goals. The new algorithm compares well with the CTL planning algorithm also in terms of performance: despite the more difficult task, in all the experiments the time required by the new algorithm for building the plan is comparable with the time required for CTL goals. In the worst cases, the new algorithm requires about twice the time of the algorithm for CTL, while in other cases the new algorithm behaves slightly better than the old one. The experiments show that the overhead for building a more complex control automaton is irrelevant w.r.t. the symbolic search phase. The slight differences in performances are mainly due to the use of different sequences of BDD operations in the search phase.

2.3.3 Concluding remarks

We have described a new language for extended goals for planning in non-deterministic domains. This language builds on and overcomes the limits of the work presented in [PT01], where extended goals are expressed in CTL. The main new features w.r.t. CTL are the capability of representing the "intentional" aspects of goals and the possibility of dealing with failure. We have defined a formal semantics for the new language that is very different from the standard CTL semantics. We have extended the planning algorithm described in [PBT01] to the new goal language. Our experimental evaluation shows that the new algorithm generates better plans than those generated from CTL goals, maintaining a comparable performance.

As far as we know, this language has never been proposed before, neither in the AI planning literature, nor in the field of automatic synthesis (see, e.g., [Var95]). The language is somehow related to languages for tactics in theorem proving. In deductive planning, tactics have been used to express plans, rather than extended goals (see [SB93, ST00]). The aim of these works is significantly different from ours. Besides [PT01], other works in planning have dealt with the problem of extended goals. Most of these works are based on extensions of LTL, see, e.g., [KBSD97]. Similarly to CTL, LTL cannot capture the goals defined in our language. Several other works only consider the case of deterministic domains (see, e.g., [dGV99, BK00]). In [BBG96] a past version of LTL is used to define temporally extended rewarding functions that can be used to provide "preferences" among temporally extended goals for MDP planning. This work shares with us the motivations for dealing with the problem of generating acceptable plans when it is impossible to satisfy a "most preferred" goal. In the framework of [BBG96], preferences are expressed with "quantitative measures", like utility functions. We propose a complementary approach where the goal language is enriched with logical operators (e.g., **TryReach** and **Fail**) that can be used to specify sort of "qualitative preferences". The comparison of the expressive power and of the practical applicability of the two approaches is an interesting topic for future evaluation. Moreover, the combination of the two approaches is a compelling topic for future research.



robot: { C1, CX, C2, B1, B2, LR, K }
bed1 : { ok, not-ok }
bed2 : { ok, not-ok }
lr : { clean, not-clean }
food : { ready, not-ready }
gas : { open, closed }

Figure 2.4: A simple domain.

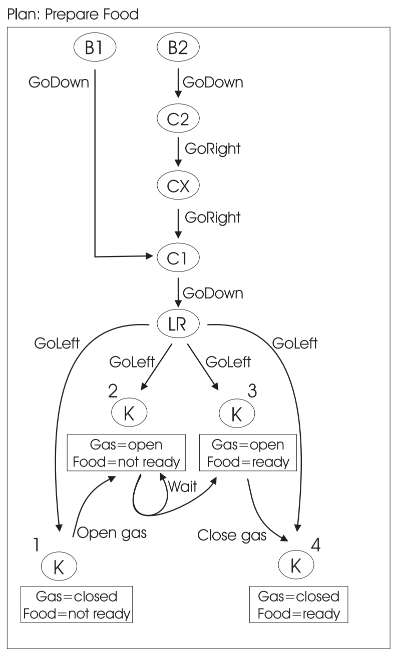
Further possible directions for future work are the extension of the language with arbitrary nesting of temporal operators (in the style of CTL or LTL), an extension to planning under partial observability [BCPT02] and to adversarial planning [JVB01]. More important, we plan to evaluate in depth the proposed approach in the real applications that originally motivated our work.

2.4 The EaGLe++ Goal Language

2.4.1 Motivating example

A simple planning domain is shown in Fig. 2.4. It consists of a flat of five rooms, where a robot can move carrying out tasks like making beds, cleaning the rooms, or cooking. A state of the system is defined by a position of the robot and by the other properties described in Figure 2.4(right). The actions in the domain consist of movement actions of the robot (GoUp, GoDown, GoLeft, GoRight), of a Wait action, and of specific activities that the robot can perform in each room. In the kitchen it can turn on and off the gas (OpenGas, CloseGas); in the living room it can dust (Clean); in a bedroom it can perform action MakeBed. The domain is non-deterministic: bed can be unmade without the direct intervention of the robot; a clean room may become dirty; and the gas has to be turned on for an arbitrary amount of time for the food to become cooked.

Figure 2.5 shows a plan that carries out the task of cooking. The left part of the picture gives a graphical representation of the plan, while the right part describes it according to Definition 2. Depending on its starting position, the robot has to reach the kitchen. In the kitchen, the behavior of the robot changes depending on the state of the gas (open, closed) and of the food (ready, not-ready). We remark the usage of action Wait, which may non-deterministically lead to a state where the food is ready if the gas is open. A single context c_0 is sufficient to describe the whole plan in this case. Indeed, the current state of the domain completely defines the next action according to the plan. An additional context $succ$ is used only to denote a successful plan termination.



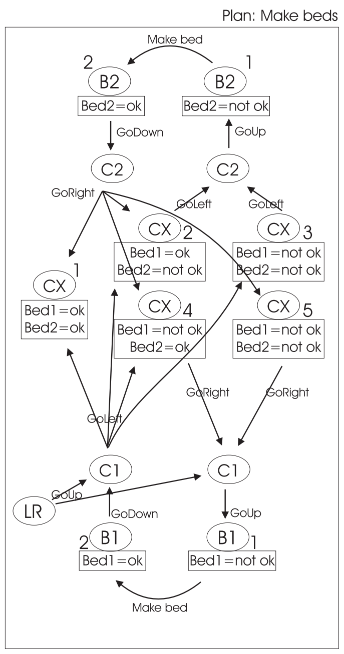
act(B2,c_0) = GoDown	ctxt(B2,c_0,C2) = c_0
act(C2,c_0) = GoRight	ctxt(C2,c_0,CX) = c_0
act(CX,c_0) = GoRight	ctxt(C2,c_0,C1) = c_0
act(B1,c_0) = GoDown	ctxt(C1,c_0,LR) = c_0
act(C1,c_0) = GoDown	ctxt(B2,c_0,LR) = c_0
act(LR,c_0) = GoLeft	ctxt(LR,c_0,K$_1$) = c_0
	ctxt(LR,c_0,K$_2$) = c_0
	ctxt(LR,c_0,K$_3$) = c_0
	ctxt(LR,c_0,K$_4$) = $succ$
act(K$_1$,c_0) = OpenGas	ctxt(K$_1$,c_0,K$_2$) = c_0
act(K$_2$,c_0) = Wait	ctxt(K$_2$,c_0,K$_2$) = c_0
	ctxt(K$_2$,c_0,K$_3$) = c_0
act(K$_3$,c_0) = CloseGas	ctxt(K$_3$,c_0,K$_4$) = $succ$

Figure 2.5: A plan for preparing food.

The goal of the second plan, depicted in Figure 2.6, is to reach a situation where both the beds are made. Due to the non-determinism in the domain, after the robot has made a bed, it can be unmade. For this reason, the robot keeps visiting the two bedrooms until both beds are ready. The decision of what to do is taken when the robot is in CX: if both beds are made, the robot can halt. If only one bed is unmade, the robot moves to the corresponding room and makes the bed. If both beds are unmade, then the robot moves to the bedroom it has not visited for more time. That is, if the beds get unmade continuously, the robot alternates between the two rooms. We remark that two contexts c_0 and c_1 are used in the plan. They are necessary to remember the last room visited by the robot.

We often need a plan that is guaranteed to achieve a given goal. In EaGLe this can be done with operators **DoReach** (that specifies a property that should be reached) and **DoMaint** (that specifies a property that should be maintained true).

However, in several domains, no plans exist that satisfy these strong goals. This is the case, for instance, for the goals **DoReach** $(bed1 = ok \wedge bed2 = ok)$. Indeed, if beds get continuously unmade, it is impossible for the robot to guarantee that a state is eventually reached where both beds are made. We need therefore to weaken the goal, but at the same time we want to capture its *intentional aspects*, i.e., we require that the plan *"does everything that is possible"* to achieve it. This is the intended meaning of **TryReach** and **TryMaint**. In the example, a possible goal could be **TryReach** $(bed1 = ok \wedge bed2 = ok)$, requiring to the robot to do its best to reach a state where both beds are made. Another example is goal **TryReach** $(food = ready \wedge gas = closed)$. Also in this case, a weak



act(C2,c_0) = GoUp	ctxt(C2,c_0,B2$_1$) = c_0
	ctxt(C2,c_0,B2$_2$) = c_1
act(B2$_1$,c_0) = MakeBed	ctxt(B2,c_0,B2$_2$) = c_1
act(B2$_2$,c_1) = GoDown	ctxt(B2$_2$,c_1,C2) = c_1
act(C2,c_1) = GoRight	...
...	
act(C1,c_1) = GoUp	ctxt(C1,c_1,B1$_1$) = c_1
	ctxt(C1,c_1,B1$_2$) = c_0
act(B1$_1$,c_1) = MakeBed	ctxt(B1,c_1,B1$_2$) = c_0
act(B1$_2$,c_0) = GoDown	ctxt(B1$_2$,c_0,C1) = c_0
act(C1,c_0) = GoLeft	...
...	

Figure 2.6: A plan for making beds.

goal is required, since, according to the domain, it might be possible that the food gets never cooked (i.e., the executing gets stacked in context K2 of Figure 2.5).

Goal **Repeat** $($**DoReach** $(bed1 = ok)$ **Then DoReach** $(bed2 = ok))$, requiring that the two beds are cyclically made, illustrates two other operators of EaGLe. Operator **Then** requires to achieve goals in sequence, while operator **Repeat** specifies that a goal should be achieved cyclically. The plan shown in Figure 2.6 satisfies this goal.

Construct **Fail** is used to model recovery from failure. Despite its simplicity, this construct is rather flexible. In combination with the other operators of the language, it allows for representing both failures that can be detected at planning time and failures that occurs at execution time. Consider, for instance, goal **DoReach** a **Fail DoReach** b. Sub-goal **DoReach** a requires to find a plan that is guaranteed to reach condition a. If such a plan exists from a given state, then sub-goal **DoReach** a cannot fail, and the recovery goal **DoReach** b is never considered. If there is no such plan, then sub-goal **DoReach** a cannot be satisfied from the given state and the recovery goal **DoReach** b is tried instead. In this case, the failure of the primary goal **DoReach** a in a given state can be decided *at planning time*. Consider now goal **TryReach** a **Fail DoReach** b. In this case, sub-goal **TryReach** a requires to find a plan that *tries* to reach condition a. During the execution of the plan, a state may be reached from which it is not possible to achieve a. When such a state is reached, goal **TryReach** a fails and the recovery goal **DoReach** b is considered. That is, in this case, the failure of the primary goal **TryReach** a is decided *at execution time*.

2.4.2 The EaGLe++ Language

The EaGLe language, allows for expressing planning goals that take into account several aspects, like nondeterminism, "intentional" aspects, and failure recovery, that are not captured by other languages for temporal goals. Still, the language is not adequate to define the complex goals typical of realistic domains, where different tasks need to be performed, and interferences among these tasks need to be taken into account. In the example of Figure 2.4, for instance, the robot need to carry out different tasks, like cooking, cleaning up, making beds... While each task can be defined by a simple EaGLe goal, as shown in the previous subsection, the definition of a suitable combination of these tasks requires different mechanisms. In this subsection, we define EaGLe++, an extension of EaGLe that allows for defining such compositions of elementary goals.

The EaGLe++ goal language is defined by the following grammar, where p and g are, respectively, propositional formulas and EaGLe goals, and id are generic identifiers:

$$\begin{aligned} eagle{+}{+} &:= mode^{*}\ \ \textbf{Start}\ id\ \ (\textbf{Trans}\ id, event \rightarrow behav)^{*} \\ mode &:= \textbf{Mode}\ id : g \mid \textbf{Mode}\ id : g\ \textbf{SafeCondition}\ p \\ event &:= \textbf{Event}\ p \mid \textbf{Done} \mid \textbf{Fail} \\ behav &:= \textbf{Ignore} \mid \textbf{Complete}, id \mid \textbf{Forget}, id \mid \textbf{Restart}, id \mid \textbf{Remember}, id \end{aligned}$$

The language is based on the concepts of modes, events, behaviours, and transitions. A **Mode** represents a task, or an operational modality, of the plan. It is defined by an identifier that is used to denote the modality, by an EaGLe goal, and by an optional safety condition, specifying constraints on the valid states from which it is possible to suspend or leave the execution of the given task. One of the modes is identified as the **Start** mode. An **Event** defines a condition that triggers a change in the active mode. In our case, an **Event** is a proposition specifying a condition on the domain. Two additional events **Done** and **Fail** correspond to the cases the execution of a modality terminates successfully or is aborted.

The transitions (**Trans**) define the changes in the modalities triggered by the events. A transition has the form **Trans** $i, e \rightarrow b$, where i is the identifier of the current modality, e is the event, and b is a behaviour specifying the reaction to the event. In the simplest case the event can be ignored (behaviour **Ignore**) and the modality is not changed. Another behaviour is **Complete**$, i'$, which requires to complete the task for the current modality, and only then to move to the new modality i'. The three other possible behaviours (**Forget**, **Restart**, **Remember**) correspond to the cases an immediate reaction to the event is required, and the execution of the current modality has to be interrupted. They differ in the way the current modality is restored after the new modality i' is completed. The intent of behaviour **Forget**$, i'$ is to specify that the current modality need not be re-entered, that is, we can safely forget the current task. Behavior **Restart**$, i'$ forces to re-start from the beginning the execution of the task corresponding to the current modality, after modality i' has been completed. Behavior **Remember**$, i'$ also specifies that we need to go back to the current modality after i' has been completed, however we do not have to re-start it from the beginning, but rather to complete it from the current context in the plan. If the goal of the current modality is **DoReach** a **Then DoReach** b, and a has been already achieved, then the execution goes back to the first goal **DoReach** a if the modality is left with a **Restart** behaviour, and continues with **DoReach** b in the case of a **Remember** behavior.

Finally, we remark that, in the base of behaviours **Forget**, **Restart**, and **Remember**, the current modality is not immediately left if a **SafeCondition** is specified for it. Instead, an auxiliary plan is first triggered that reached a state satisfying the safety condition. This is useful to specify, e.g., that it is possible to leave the task of cooking, if more urgent activities need to be done, but that the gas has to be turned off in this case.

An example of a (simple) EaGLe++ goal is the following:

Mode $PrepareFood$: **TryReach** $(gas = closed \wedge food = ready)$
  SafeCondition$(gas = closed)$
Mode $MakeBeds$: **TryReach** $(bed1 = ok \wedge bed2 = ok)$
Start $PrepareFood$
Trans $PrepareFood,$ **Event**$(time = 10{:}00AM) \rightarrow$ **Restart**$, MakeBeds$
Trans $PrepareFood,$ **Done** $\rightarrow$ **Forget**$, MakeBeds$

This goal requires to perform two tasks in sequence, namely cooking and tidying up the bedrooms. However, if too much time is spent cooking (condition $time = 10{:}00AM$), then we want to suspend the task of preparing food, make the beds, and then restart cooking. Moreover, the gas cannot be left open when we leave the kitchen. A plan satisfying this EaGLe++ goal is reported in Figure 2.7.[1]It consists of a combination of the two plans already presented in Figures 2.5 and 2.6. However, some extra activities are defined in order to manage the passage from a modality to the other one. The passage from modality $PrepareFood$ to modality $MakeBeds$, in particular, requires the introduction of additional actions. Notably, actions are introduced in the plan for turning off the gas when needed. These actions are required in order to satisfy the **SafeCondition** of modality $PrepareFood$.

We remark that there is no easy way to write the EaGLE++ goal described above using EaGLe. Indeed, this would require inserting the description of the behavior in case of event $time = 10{:}00AM$ inside the formula for goal $PrepareFood$. It is easy to see that this encoding can be obtained only at the cost of obtaining a formula that is more complex and where the distinction among modalities is lost.

[1]Some simplifications have been done in order to make the plan more readable. First, the event $time = 10{:}00AM$ is considered only for a subset of the states in the plan for $PrepareFood$. Second, once the plan for $MakeBeds$ ends, the execution goes back unconditionally to modality $PrepareFood$, while it should go back only if mode $PrepareFood$ was left due to event $time = 10{:}00AM$.

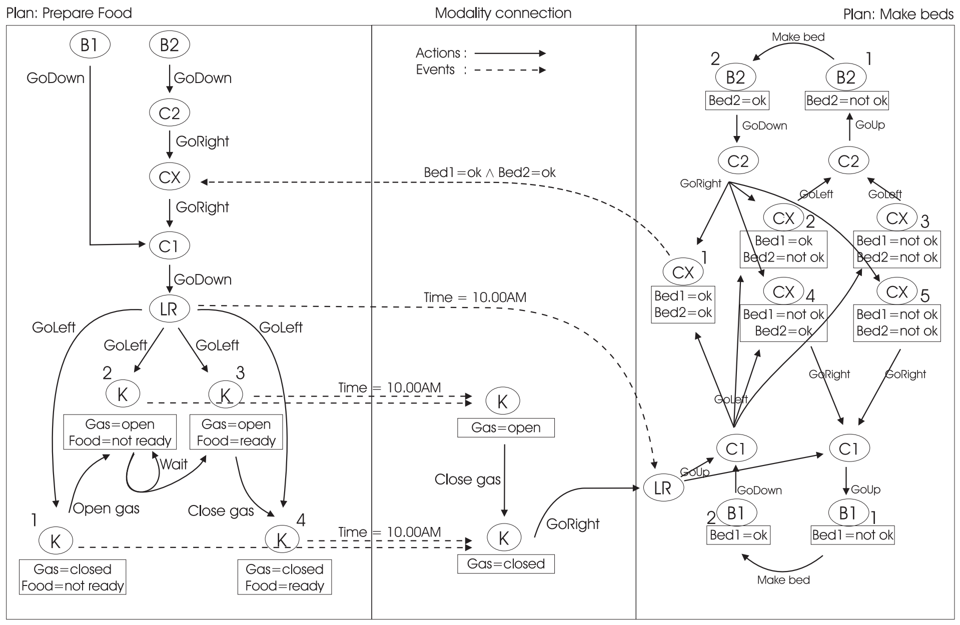


Figure 2.7: A composed plan.

2.4.3 Implementation and Experiments

A planning algorithm for EaGLe++ has been implemented in the MBP Planner [BCP$^+$01b]. This implementation is based on the algorithms already developed for EaGLe [DLPT02]. EaGLe uses the BDD-based symbolic techniques provided by MBP itself to represent the domain. The goals, instead, are translated in a particular form of automaton, called control automaton, suitable to apply the planning algorithms. Essentially, the states of a control automaton codify the different tasks that need to be performed to achieve a goal, and correspond to the different contexts of the plan that is being built (see Definition 2). The control automaton is exploited to guide the (BDD-based) exploration of the domain and the construction of the plan.

In [Mat04] it is shown how the planning algorithms developed for EaGLe can be extended to EaGLe++. The key idea is to construct the control automata for the EaGLe goals associated to the single modalities, and then to combine these basic control automata according to the behaviours governing the modality transitions. This way, the planning algorithm already developed for EaGLe applies also to EaGLe++ goals.

We have evaluated the feasibility of the approach through a set of tests. In the tests, we have considered a complex version of the domain described in Figure 2.4, with more rooms and non-determinism, and with more composite actions and events. The results of these experiments are reported in Figure 2.8. For each test we report the result of the planning task (i.e., whether a plan exists for the goal of not), the time required to complete the planning task, and the number of contexts in the generated control automaton. The latter information is a rough measure of the complexity of the associated EaGLe++ goal.

Description	Plan	Time (sec.)	Contexts
EstinguishFire	NO	0.018	4
EstinguishFire2	YES	0.272	4
WashCloths	YES	3.103	4
MakeBeds	YES	0.150	7
AnswerDoor	YES	0.174	7
IronCloths	NO	41.476	8
IronCloths2	YES	154.515	8
Event-Lunchtime-Restart	YES	289.700	36
Event-Lunchtime-Complete	YES	333.819	50
Event-Lunchtime-Forget	YES	323.130	63
Event-Lunchtime-Remember	YES	3600.514	891
All-Domain	YES	7561.762	1923

Figure 2.8: Results of the experiments.

The first 7 tests are on small goals with one or two modalities. The next four tests are on larger goals, combining different modalities. They differ in the behaviour associated to the event lunchtime, namely, the current activity can be restarted after lunch, completed before lunch, forgotten, or resumed after lunch. The last test, finally, correspond to the complete goal specifying all activities of the robot in the domain. This goal is rather complex, as it has 10 different modes and 4 possible events that lead to complex patterns of execution of the different modalities.

2.4.4 Conclusions

We have addressed the problem of defining a language that allows for specifying complex planning goals in terms of the different operational modes that should become active during the execution of the plan. The new goal language, called EaGLe++, is based on EaGLe, an expressive language for describing temporally extended goals for non-deterministic domains. The experiments conducted show that planning for EaGLe++ is possible, also in the case of rather complex goals. However, the performance tends to become worse when the number of modalities in the goal grows.

Bibliography

[ACG$^+$01] L. Carlucci Aiello, A. Cesta, E. Giunchiglia, M. Pis tore, and P. Traverso. Planning and verification techniques for the high level programming and monitoring of autonomous robotic devices. In *Proceedings of the ESA Workshop on On Board Autonomy*. ESA Press, 2001.

[BBG96] F. Bacchus, C. Boutilier, and A. Grove. Rewarding Behaviors. In *Proc. of AAAI'96*. AAAI-Press, 1996.

[BCM$^+$92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BCP$^+$01a] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *Proc. of IJCAI'01 workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[BCP$^+$01b] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *Proceeding of ICAI-2001 workshop on Planning under Uncertainty and Incomplete Information*, pages 93–97, Seattle, USA, August 2001.

[BCPT02] P. Bertoli, A. Cimatti, M. Pistore, and P. Traverso. Plan validation for extended goals under partial observability (preliminary report). In *Proc. of the AIPS'02 Workshop on Planning via Model Checking*, April 2002.

[BG97] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31, 1997.

[BK00] F. Bacchus and F. Kabanza. Using Temporal Logic to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[Cla78] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Dav67] Donald Davidson. The logical form of action sentences. In *The Logic of Decision and Action*, pages 81–120, 1967.

[DGKK98] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL: Temporal action logics language specification and tutorial. *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, 3(015), 1998. Available at `http://www.ep.liu.se/ea/cis/1998/015/`.

[dGV99] G. de Giacomo and M.Y. Vardi. Automata-Theoretic Approach to Planning with Temporally Extended Goals. In *Proc. ECP'99*, 1999.

[DLPT02] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *Proc. AAAI'02*, 2002.

[EG93] Thomas Eiter and Georg Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In *Proceedings of International Logic Programming Symposium (ILPS)*, pages 266–278, 1993.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier, 1990.

[FN71] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[Gef90] Hector Geffner. Causal theories for nonmonotonic reasoning. In *Proc. AAAI-90*, pages 524–530, 1990.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GL93] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[GL98] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI*, pages 623–630, 1998.

[HM87] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[JVB01] R. M. Jensen, M. M. Veloso, and M. H. Bowling. OBDD-based optimistic and strong cyclic adversarial planning. In *Proc. of ECP'01*, 2001.

[KBSD97] F. Kabanza, M. Barbeau, and R. St-Denis. Planning Control Rules for Reactive Agents. *Artificial Intelligence*, 95(1):67–113, 1997.

[KS86] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Lin95] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. IJCAI-95*, pages 1985–1991, 1995.

[Lin96] Fangzhen Lin. Embracing causality in specifying the indeterminate effects of actions. In *Proc. AAAI-96*, 1996.

[Mat04] A. Mattioli. Pianificazione per Goal Estesi: Controllo di Agenti Capaci di Operare in Modalità Multiple. Master's thesis, Università di Trento, 2004.

[McC59] John McCarthy. Programs with common sense. In *Proc. Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office. Reproduced in [McC90].

[McC80] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence,* 13:27–39,171–172, 1980. Reproduced in [McC90].

[McC86] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986. Reproduced in [McC90].

[McC90] John McCarthy. *Formalizing Common Sense: Papers by John McCarthy*. Ablex, Norwood, NJ, 1990.

[McC03] John McCarthy. Elaboration tolerance, 2003. In progress. Available at `http://www-formal.stanford.edu/jmc/elaboration.html` .

[MH69] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969. Reproduced in [McC90].

[MT97] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465, 1997.

[PBT01] M. Pistore, R. Bettin, and P. Traverso. Symbolic techniques for planning with extended goals in non-deterministic domains. In *Proc. of ECP'01*, 2001.

[Ped87] Edwin Pednault. Formulating multi-agent, dynamic world problems in the classical planning framework. In Michael Georgeff and Amy Lansky, editors, *Reasoning about Actions and Plans*, pages 47–82. Morgan Kaufmann, San Mateo, CA, 1987.

[Ped89] Edwin Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald Brachman, Hector Levesque, and Raymond Reiter, editors, *Proc. First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[Ped94] Edwin Pednault. ADL and the state-transition model of action. *Journal of Logic and Computation*, 4:467–512, 1994.

[PT01] M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. of IJCAI'01*. AAAI Press, 2001.

[Rei80] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Rei91] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[San95] Erik Sandewall. *Features and Fluents*, volume 1. Oxford University Press, 1995.

[SB93] W. Stephan and S. Biundo. A New Logical Framework for Deductive Planning. In *Proc. of IJCAI'93*, 1993.

[Sha97] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[SI93] Chiaki Sakama and Katsumi Inoue. Relating disjunctive logic programs to default theories. In *LPNMR*, pages 266–282, 1993.

[ST00] L. Spalazzi and P. Traverso. A Dynamic Logic for Acting, Sensing and Planning. *Journal of Logic and Computation*, 10(6):787–821, 2000.

[Thi97] Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997.

[Tur96] Hudson Turner. Splitting a default theory. In *Proc. AAAI-96*, pages 645–651, 1996.

[Tur97] Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.

[Tur99] Hudson Turner. A logic of universal causation. *Artif. Intell.*, 113(1-2):87–123, 1999.

[Var95] M. Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proc. of CAV'95*, 1995.