Chapter 1

GETTING TO THE AIRPORT: THE OLDEST PLANNING PROBLEM IN AI

Vladimir Lifschitz
University of Texas at Austin

Norman McCain
Baker University

Emilio Remolina
University of Texas at Austin

Armando Tacchella
Università di Genova

Abstract The problem discussed in this paper is described in a 1959 paper by John McCarthy as follows: *Assume that I am seated at my desk at home and I wish to go to the airport. My car is at my home also. The solution of the problem is to walk to the car and drive the car to the airport.* In the spirit of what is now known as the logic approach to AI, McCarthy proposed to address this problem by first giving "a formal statement of the premises" that a reasoning program would use to draw the relevant conclusions. Our goal here is to take a careful look at this episode from the early history of AI and to identify some of the logical and algorithmic ideas related to the airport problem that have emerged over the years.

1. INTRODUCTION

The problem discussed in this paper is described in [McCarthy, 1959] as follows:

> Assume that I am seated at my desk at home and I wish to go to the airport. My car is at my home also. The solution of the problem is to walk to the car and drive the car to the airport.

In the spirit of what is now known as the logic approach to AI, McCarthy proposed to address this problem by first giving "a formal statement of the premises" that a reasoning program would use to draw the relevant conclusions. Forty years ago, these premises could not be identified and formalized in a satisfactory way: the advances in knowledge representation and in formal commonsense reasoning that are required for this purpose came much later.

Our goal here is to take a careful look at this episode from the early history of AI and to identify some of the logical and algorithmic ideas related to the airport problem that have emerged over the years. We show that some of the relevant ideas are of a recent origin. Just a few years ago, formalizing the airport domain and turning this formalization into an input for a planner would be more difficult than it is today.

The attempt to describe the airport domain in [McCarthy, 1959] includes, among others, the following logical formulas:

$$\begin{gathered} at(I, desk), \\ at(desk, home), \\ at(car, home), \\ at(home, county), \\ at(airport, county), \end{gathered} \tag{1.1}$$

$$at(x, y) \wedge at(y, z) \supset at(x, z), \tag{1.2}$$

$$\begin{aligned} & walkable(x) \wedge at(y, x) \wedge at(z, x) \wedge at(I, x) \\ & \qquad \supset can(go(y, z, walking)), \\ & drivable(x) \wedge at(y, x) \wedge at(z, x) \wedge at(car, x) \wedge at(I, car) \\ & \qquad \supset can(go(y, z, driving)), \end{aligned} \tag{1.3}$$

$$\begin{gathered} walkable(home), \\ drivable(county), \end{gathered} \tag{1.4}$$

$$did(go(x, y, z)) \supset at(I, y). \tag{1.5}$$

The intention of formulas (1.3) and (1.4) is to explain why one can walk to the car but not to the airport by postulating that the home is "walkable" (it is so small that one can walk between any two locations within the home), but the county is only "drivable."

According to McCarthy's idea, a reasoning program would deduce, from these and other axioms, a formula corresponding to the first step of the plan:

$$do(go(desk, car, walking)). \tag{1.6}$$

"The deduction of the last proposition initiates action."

In the course of the discussion of this proposal at the Teddington Conference on the Mechanization of Thought Processes in 1959, Yehoshua Bar-Hillel raised a few objections, and these objections are discussed in Section 2. below. Many other difficulties transpired years later; the best known of them is the frame problem. (How do we know that after the execution of the first step of the plan—walking to the car—the car is still at home? Perhaps it has moved to another part of the world by that time, and driving to the airport became impossible!) We talk about the frame problem and other difficulties in Section 3..

The second half of the paper demonstrates that today all these problems—with one notable exception, discussed in Section 2.4—can be successfully resolved. In Section 4. we formalize the airport domain using recent work on action languages [Giunchiglia and Lifschitz, 1998]. In Section 5. we show how the "causal calculator" written by one of us (NMC) can be used to generate McCarthy's two-step solution.

2. BAR-HILLEL'S OBJECTIONS

2.1 TRANSITIVITY OF "AT"

Bar-Hillel's first objection against McCarthy's proposed formalization of the airport example deals with the assumption that the relation *at* is transitive (1.2):

> . . . since he takes both "$at(I, desk)$" and "$at(desk, home)$" as premises, I presume—though this is never made quite clear—that *at* means something like being-a-physical-part-or-in-the-immediate-spatial-neighborhood-of. But then the relation is clearly not transitive. If A is in the immediate spatial neighborhood of B, and B in the immediate spatial neighborhood of C, than A need not be in the immediate spatial neighborhood of C. Otherwise, everything would turn out to be in the immediate spatial neighborhood of everything, which is surely not Dr. McCarthy's intention.

McCarthy said in reply that he

> was not trying to formalize the sentence form, A is at B, as it is used in English: *at* merely was intended to serve as a convenient mnemonic for the relation between a place and a subplace.

(See [McCarthy, 1990], pages 17 and 20.)

In spite of this clarification, it seems to us that Bar-Hillel's objection is well taken. To sit at the desk does not mean to occupy a subplace of the desk. The symbol *at* can be understood to represent the relation between a place and a subplace in the last 4 of axioms (1.1), but not in $at(I, desk)$.

One way out of this difficulty is to replace this axiom by

$$at(I, neighborhood(desk)),$$

where *neighborhood* represents a function that maps every region to a slightly larger region—its "immediate neighborhood." We would write, for instance,

$$at(scissors, desk)$$

to express that the scissors are in a drawer (or perhaps on top) of the desk, but $at(I, desk)$ would not be an acceptable postulate. The difference between $at(I, car)$ and $at(I, neighborhood(car))$ could be used to distinguish between being inside the car and standing next to it.

Alternatively, we can justify the first of axioms (1.1) as given above if we agree to understand the symbol *desk* to represent an immediate neighborhood of the desk. This convention would not apply to the other symbols: *car*, as before, denotes the part of space occupied by the car, rather than a neighborhood of the car, and similarly for *I*, *home*, *airport* and *county*.

The formalization given in Section 4. below uses this simpler (although admittedly ad hoc) solution.

2.2 ARRIVING AT A PLAN BY DEDUCTION

About the expected conclusion (1.6) Bar-Hillel said:

> It sounds rather incredible that the machine could have arrived at its conclusion—which, in plain English, is "Walk from your desk to your car!"—by sound deduction. This conclusion surely could not possibly follow from the premise in any serious sense. Might it not be occasionally cheaper to call a taxi and have it take you over to the airport?

(See [McCarthy, 1990], pages 17–18.)

Indeed, the idea that a plan is a deductive consequence of the statement of the planning problem seems to presuppose the uniqueness of the plan: the problem is solved *only if this particular plan is executed*. In algebra, if 5 is a solution to a given equation, we do not expect $x = 5$ to be "deducible" from the equation in any sense—unless 5 is the only solution.

The relationship between a formal statement of the problem and a plan is not the provability relation. It is better to say that the plan "satisfies" the statement of the problem in some sense. Whatever the precise meaning of this word is, it's essential that several different objects may "satisfy" the statement of a planning problem. (And there are no objects with this property if the problem is not solvable.)

In deductive planning, for instance, plans are represented by ground terms of the situation calculus [Green, 1969]. A problem is described by specifying, first, an axiomatic theory T that describes the initial situation and the effects of actions, and, second, a goal condition $G(s)$. A situation term t "satisfies" $\langle T, G(s)\rangle$ if T entails $G(t)$.

In satisfiability planning [Kautz and Selman, 1992], plans are represented by propositional interpretations, and the satisfaction relation in question is essentially the satisfaction relation of propositional logic. Answer set planning [Subrahmanian and Zaniolo, 1995, Dimopoulos *et al.*, 1997, Lifschitz, 1999a] is similar, except that its underlying satisfaction relation is nonmonotonic.

2.3 THE TIME FACTOR

Further in Bar-Hillel's comments, we read:

> Let me also point out that in the example the time factor has never been mentioned, probably for the sake of simplicity. But clearly this factor is here so important that it could not possibly be disregarded without distorting the whole argument. Does not the solution depend, among thousands of other things, also upon the time of my being at my desk, the time at which I have to be at the airport, the distance from the airport, the speed of my car, etc.

(See [McCarthy, 1990], page 18.)

Over the years, several temporal formalisms were proposed for use in formalizing actions, beginning with the situation calculus [McCarthy and Hayes, 1969]. As to the description of the airport domain in [McCarthy, 1959], it is not correct to say that the time factor is ignored there altogether: the use of *did* in (1.5) hints at a time difference between action $go(x, y, z)$ and its effect $at(I, y)$. This syntactic mechanism helps us distinguish the effects of an action, as in (1.5), from its preconditions, as in (1.3). But it does not allow us to talk about the execution of several actions in a row. In this respect, it is similar to the language of STRIPS operators [Fikes and Nilsson, 1971], to ADL [Pednault, 1987] and to other action description languages [Gelfond and Lifschitz, 1998].

The temporal formalism used in this paper is action language $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998], reviewed in Section 4.1 below. This language has a simple syntax but is substantially more expressive than STRIPS.

2.4 IDENTIFYING THE RELEVANT PREMISES

Bar-Hillel said also in his comments:

> A deductive argument, where you have first to find out what are the relevant premises, is something that many humans are not always able to carry out successfully. I do not see the slightest reason to believe that, at present, machines should be able to perform things that humans find trouble in doing. I do not think there could possibly exist a program which would, given any problem, divide all facts in the universe into those which are and those which are not relevant for that problem. Developing such a program seems to me by 10^{10} orders of magnitude more difficult than, say, the Newell—Simon problem of developing a heuristic for deduction in the propositional calculus. This cavalier way of jumping over orders of magnitude only tends to becloud the issue and throw doubt on ways of thinking for which I have a great deal of respect. By developing a powerful program language, you may have paved the way for the first step in solving problems of the kind treated in your example, but the claim of being well on the way towards their solution is a gross exaggeration. This was the major point of my objections.

(See [McCarthy, 1990], page 19.)

Today, admittedly, AI researchers have about as little to say on this subject as they did in 1959. There is no need to explain that, in setting up the formalization of the airport example below, the task of identifying the relevant premises was not automated.

3. OTHER DIFFICULTIES

3.1 GOING TO A REGION

The destination of the action *go* may be a large “region” that has subregions (rather than a specific “location,” as in the case of the move action in the blocks world). This leads to an interesting question. If I go to a region y then, according to (1.5), fluent $at(I, y)$ becomes true; but what about the effect of this action on fluent $at(I, y')$, where y' is a part of y? For instance, when I go home, how does this action affect fluent $at(I, desk)$?

We see here three choices.

One is to say that fluent $at(I, desk)$ may remain false or may become true, depending on the particular way of executing the action of going home. The action becomes nondeterministic.

Second, we can say that this fluent always remains false. Going home is understood then as going to an unspecified location within the home that is disjoint from all the subregions of *home* that have names in the language (*desk* and *car*, in the initial state of the airport problem).

Finally, we can say that going home sometimes "involves" going to the desk, and sometimes it does not. In the first case, fluent $at(I, desk)$ becomes true; in the second case, it does not. This view is adopted in the formalization below.

The idea that an execution of an action of type A_1 "involves" an execution of an action of type A_2 may be understood in two ways. We may think that there is a single event which instantiates both the A_1 and A_2 action types, or we may think that there are two events appropriately related—they at least must happen concurrently. Formally, we are not committed to either interpretation. When we say that the action A_1 occurs, we mean that some event occurs that instantiates the A_1 action type. When we say that the action A_2 also occurs, we mean that some event occurs that instantiates the A_2 action type. Whether these are the same event or two is not an issue that we address.[1] The view that a single event may instantiate distinct action types or descriptions has been articulated by the philosopher Donald Davidson [1967].

3.2 NEED FOR NONMONOTONIC REASONING

The discovery of the frame problem and the invention of the nonmonotonic formalisms that are capable of solving it may have been the most significant events so far in the history of research on reasoning about actions. A large part of this story is described in [Shanahan, 1997].

In simple cases, the frame problem can be solved monotonically [Schubert, 1990], but in the case of the airport domain the use of nonmonotonic reasoning seems almost imperative. The reason for this is that going to a region may have rather complicated ramifications. Axiom (1.5) describes the direct effect of going to y: in the resulting state, I am at y. But this action has also indirect effects. First, for every region y' that includes y, I am at y'. Second, for every region y' that is disjoint from y, I am not at y'. Formal nonmonotonic reasoning helps us describe these indirect effects in a concise way.

A nonmonotonic solution to the frame problem provides a formalization of the informal principle called the "commonsense law of inertia." In application to a fluent f, this principle asserts that, as time goes by, f tends to keep the same value that it had in the past. The approach to expressing inertia adopted in action language $\mathcal{C}$ is outlined in Section 4.1.

Another use of nonmonotonic reasoning in our formalization of the airport example has to do with the closed world assumption. Axioms (1.4) give little information about the extents of the predicates *walkable* and *drivable*; they do not tell us, for instance, whether the airport has any of these two properties, or whether the county is walkable. We can observe that if a region x is so small that one can walk (or drive), in principle, between any two locations within it, then any subregion of x has the same property. The axiom expressing this idea will allow us to derive, for instance, $\mathit{drivable}(\mathit{airport})$ from the axioms $\mathit{drivable}(\mathit{county})$ and $\mathit{at}(\mathit{airport}, \mathit{county})$. It also will allow us to derive $\mathit{drivable}(\mathit{home})$. (We understand $\mathit{drivable}(X)$ to mean only that the distances within X are not too large, not necessarily that there are no obstacles against driving within X.) Even so, we will not be able to prove any negative facts about *walkable* or *drivable*. One way to overcome this difficulty is to postulate the closed world assumption for these two predicates: by default, a region is not walkable and not drivable. This idea, just like the idea of inertia, calls for the use of a nonmonotonic formalism.

4. AIRPORT DOMAIN IN ACTION LANGUAGE $\mathcal{C}$

4.1 LANGUAGE

According to [Giunchiglia and Lifschitz, 1998] (see also [Gelfond and Lifschitz, 1998, Section 6]), a description of an action domain in action language $\mathcal{C}$ is a set of propositions of two kinds: *static laws*

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

and *dynamic laws*

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

Here F, G and U are logical formulas. The atomic subformulas of F and G are names of propositional fluents. U may contain, in addition, names of "elementary actions"; these names are used as atomic formulas also. An assignment of truth values to the names of elementary actions represents, intuitively, the composite action that consists in executing concurrently, during some fixed time interval, all elementary actions to which the value t is assigned. According to the semantics of $\mathcal{C}$, every action description represents a "transition system"—a directed graph whose edges correspond to the transitions caused by the execution of actions.

To illustrate the use of $\mathcal{C}$ notation, consider a few examples.

1. The effect of $\mathit{go}(x, y, z)$ on $\mathit{at}(I, y)$ that McCarthy represented by formula (1.5) would be expressed in $\mathcal{C}$ by

$$\mathit{go}(x, y, z) \textbf{ causes } \mathit{at}(I, y).$$

This expression stands for the dynamic law

$$\textbf{caused } \mathit{at}(I, y) \textbf{ if } \mathit{true} \textbf{ after } \mathit{go}(x, y, z).$$

The general definition of the abbreviation **causes**, as well as the definitions of other abbreviations for $\mathcal{C}$ propositions, can be found in [Gelfond and Lifschitz, 1998, Section 6].

2. Consider now the first of formulas (1.3), which expresses a sufficient condition for the executabilty of $\mathit{go}(y, z, \mathit{walking})$. In $\mathcal{C}$, a similar condition might be written as

$$\begin{array}{l}\textbf{nonexecutable } \mathit{go}(y, z, \mathit{walking}) \\ \qquad\qquad \textbf{if } \neg\exists x(\mathit{walkable}(x) \wedge \mathit{at}(y, x) \wedge \mathit{at}(z, x) \wedge \mathit{at}(I, y))\end{array}$$

which stands for the dynamic law

$$\begin{array}{l}\textbf{caused}\ \mathit{false}\ \textbf{if}\ \mathit{true}\\ \quad\textbf{after}\ \neg\exists x(\mathit{walkable}(x)\wedge \mathit{at}(y,x)\wedge \mathit{at}(z,x)\wedge \mathit{at}(I,y))\wedge \mathit{go}(y,z,\mathit{walking}).\end{array}$$

Actions described in $\mathcal{C}$ are presumed to be executable if there is no evidence to the contrary, and the **nonexecutable** construct provides a way to express such evidence. The version of $\mathcal{C}$ described in the papers mentioned above and used in this paper does not permit quantifiers. To comply with this limitation, $\exists x$ can be replaced by a finite disjunction over the regions represented in the language.

3. The closed world assumption for *walkable* (Section 3.2) can be expressed in $\mathcal{C}$ by

$$\textbf{default}\ \neg\mathit{walkable}(x)$$

which stands for the static law

$$\textbf{caused}\ \neg\mathit{walkable}(x)\ \textbf{if}\ \neg\mathit{walkable}(x).$$

The role of this proposition in $\mathcal{C}$ is similar to the role of the normal default

$$\frac{:\ \neg\mathit{walkable}(x)}{\neg\mathit{walkable}(x)}$$

in default logic [Reiter, 1980]. Intuitively, it says that, whenever x is not walkable, there is a cause for this; when x is walkable, however, this fact requires a special explanation provided by "positive" postulates such as

$$\textbf{caused}\ \mathit{walkable}(\mathit{home}).$$

This expression, similar to the first of postulates (1.4), stands for the static law

$$\textbf{caused}\ \mathit{walkable}(\mathit{home})\ \textbf{if}\ \mathit{true}.$$

4. One possible counterpart of (1.2) in $\mathcal{C}$ is

$$\textbf{always}\ \mathit{at}(x,y)\wedge \mathit{at}(y,z)\supset \mathit{at}(x,z)$$

which stands for the static law

$$\textbf{caused}\ \mathit{false}\ \textbf{if}\ \neg(\mathit{at}(x,y)\wedge \mathit{at}(y,z)\supset \mathit{at}(x,z)).$$

This proposition eliminates the states of the world in which *at* is nontransitive, so that any action whose effect is to make (1.2) false is nonexecutable. This version of (1.2) is a "qualification constraint" in the sense of [Lin and Reiter, 1994].

Another possibility is to postulate the static law

$$\textbf{caused}\ \mathit{at}(x,z)\ \textbf{if}\ \mathit{at}(x,y)\wedge \mathit{at}(y,z).$$

This is the version used in our formalization of the airport example. Besides eliminating the "bad" states, it allows us to draw conclusions about indirect effects of actions. It tells us, in particular, that any action which makes $at(x,y) \wedge at(y,z)$ true has also another effect: it makes $at(x,z)$ true. For instance, if I go to a subregion y of z then one ramification of this is that $at(I,z)$ becomes true; this was discussed in Section 3.2.

5. The commonsense law of inertia is not a "built-in" feature of $\mathcal{C}$, but the inertia assumption can be easily expressed in this language. For instance,

$$\textbf{inertial}\ at(x,y)$$

stands for dynamic law

$$\textbf{caused}\ at(x,y)\ \textbf{if}\ at(x,y)\ \textbf{after}\ at(x,y).$$

If $at(x,y)$ was true and remained true after executing an action then there is a cause for this. Whenever this fluent changes its value from t to f, a causal explanation is required. Such explanations are provided by other causal laws—the dynamic and static laws describing the direct and indirect effects of actions on $at(x,y)$.

4.2 FORMALIZATION

In our formalization of the airport example, action names are expressions of the form $go(X,M)$ where the metavariable X ranges over the symbols for regions

$$i,\ desk,\ car,\ home,\ airport,\ county \tag{1.7}$$

and M is either *walking* or *driving*. This is more concise than McCarthy's notation $go(x,y,z)$: we suppress the "source" argument x and keep only the "destination" argument and the "mode" argument.

We use fluent names of three kinds:

$$at(X,Y)$$

where each of the argument positions is occupied by one of symbols (1.7), and

$$walkable(U),\ drivable(U)$$

where U is one of the symbols in (1.7) that represent "big areas":

$$home,\ airport,\ county. \tag{1.8}$$

Although $walkable(U)$ and $drivable(U)$ are formally treated as fluents, their truth values are not going to be affected by any actions.

The first group of postulates consists of the static laws describing properties of *at*:

$$\begin{array}{ll}\textbf{caused}\ at(X,Z)\ \textbf{if}\ at(X,Y) \wedge at(Y,Z), & \\ \textbf{caused}\ \neg at(X,Z)\ \textbf{if}\ at(X,Y) \wedge disjoint(Y,Z) & (Y \neq Z), \\ \textbf{caused}\ \neg at(X,X). & \end{array}$$

Here X, Y, Z range over (1.7), and $disjoint(Y,Z)$ stands for

$$\neg at(Y,Z) \wedge \neg at(Z,Y). \tag{1.9}$$

This abbreviation is motivated by the stipulation that the regions occupied by objects (1.7) never overlap: if two different regions Y, Z from that list satisfy (1.9) then they are disjoint.

The next two groups of postulates describe the direct effects of actions:

$$\begin{array}{l} \mathit{go}(X, M) \textbf{ causes } \mathit{at}(i, X), \\ \mathit{go}(X, \mathit{driving}) \textbf{ causes } \mathit{at}(\mathit{car}, X) \end{array}$$

and their preconditions:

$$\begin{array}{l} \textbf{nonexecutable } \mathit{go}(X, M) \textbf{ if } \mathit{at}(i, X), \\ \textbf{nonexecutable } \mathit{go}(X, \mathit{driving}) \textbf{ if } \neg \mathit{at}(i, \mathit{car}), \\ \textbf{nonexecutable } \mathit{go}(X, \mathit{walking}) \textbf{ if } \neg \bigvee_U (\mathit{walkable}(U) \wedge \mathit{at}(i, U) \wedge \mathit{at}(X, U)), \\ \textbf{nonexecutable } \mathit{go}(X, \mathit{driving}) \textbf{ if } \neg \bigvee_U (\mathit{drivable}(U) \wedge \mathit{at}(i, U) \wedge \mathit{at}(X, U)). \end{array}$$

Here U ranges over the names of big areas (1.8).

We also postulate two restrictions on the concurrent execution of actions. First, it is impossible to walk and to drive simultaneously:

$$\textbf{nonexecutable } \mathit{go}(X, \mathit{walking}) \wedge \mathit{go}(Y, \mathit{driving}).$$

Second, going to a region involves going to all supersets of that region (see Section 3.1):

$$\textbf{nonexecutable } \mathit{go}(X, M) \wedge \neg \mathit{go}(Y, M) \textbf{ if } \mathit{at}(X, Y) \wedge \neg \mathit{at}(i, Y).$$

The next group of postulates provides information related to the sizes of objects:

$$\begin{array}{l} \textbf{always } \neg \mathit{at}(\mathit{car}, \mathit{desk}), \\ \textbf{caused } \mathit{walkable}(\mathit{home}), \\ \textbf{caused } \mathit{walkable}(U) \textbf{ if } \mathit{at}(U, V) \wedge \mathit{walkable}(V), \\ \textbf{caused } \mathit{drivable}(\mathit{county}), \\ \textbf{caused } \mathit{drivable}(U) \textbf{ if } \mathit{at}(U, V) \wedge \mathit{drivable}(V). \end{array}$$

The first of them expresses that the car is too big to be fully contained in the immediate neighborhood of the desk. This fact is relevant because it explains why it would be impossible to get back from the airport to the desk in one step, by driving not followed by walking. (Alternatively, this could be explained by the impossibility of driving inside the home, in spite of the fact that the home is small enough to be "drivable.")

Finally, we need to postulate that *at* satisfies the commonsense law of inertia

$$\textbf{inertial } \mathit{at}(X, Y), \neg \mathit{at}(X, Y)$$

and that *walkable* and *drivable* are normally false:

$$\begin{array}{l} \textbf{default } \neg \mathit{walkable}(U), \\ \textbf{default } \neg \mathit{drivable}(U). \end{array}$$

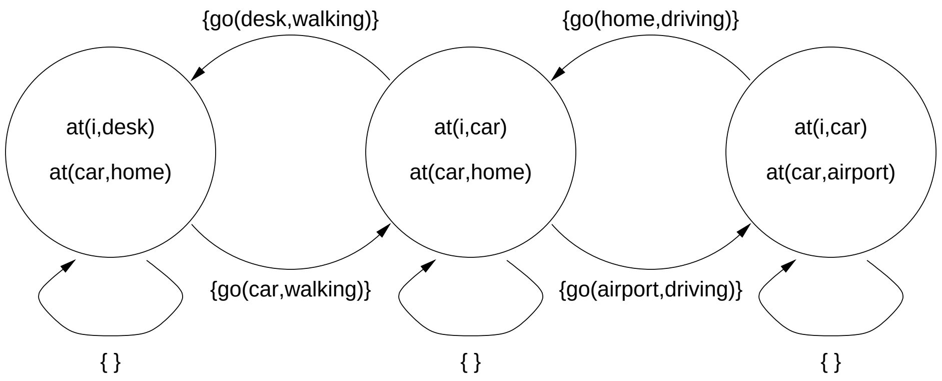

Figure 1.1 Part of the transition system describing the airport domain

4.3 TRANSITION SYSTEM

The semantics of $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998] defines how the propositions above describe a transition system. Every state of this transition system is characterized by an assignment of truth values to the fluent names

$$at(X,Y),\ walkable(U),\ drivable(U).$$

One of the states, s_0, is the initial state of the airport problem. The fluents that are true in state s_0 are

$$\begin{gathered} walkable(home), \\ drivable(airport),\ drivable(county),\ drivable(home), \\ at(airport, county),\ at(home, county), \\ at(desk, county),\ at(i, county),\ at(car, county), \\ at(desk, home), \end{gathered} \tag{1.10}$$

$$at(i, desk),\ at(i, home),\ at(car, home). \tag{1.11}$$

Fluents (1.10) are not affected by any of the actions $go(X,M)$, so that they hold not only in state s_0 but also in all states that are reachable from s_0 in the transition system. Fluents (1.11) may become false after the execution of a sequence of actions.

Figure 1.1 shows state s_0 along with the two states that can be reached from it by executing actions. State s_0 is shown on the left. There are two actions that can be executed in this state: doing nothing (the loop labeled { }) and walking to the car. In the new state shown in the center, one can either do nothing, or walk back to the desk, or drive to the airport (the state on the right). In this last state, one can either do nothing or drive back home.

5. SOLVING THE PLANNING PROBLEM

5.1 CAUSAL CALCULATOR AND SATISFIABILITY PLANNING

The Causal Calculator[2] is an implementation of the propositional causal logic from [McCain and Turner, 1997]. Domain descriptions in $\mathcal{C}$ are translated by the Causal Calculator first into the language of causal theories and then into propositional logic. The models of the propositional theory correspond to paths in the transition system described by the original domain description in $\mathcal{C}$.

Planning is carried out by satisfiability checking, as proposed in [Kautz and Selman, 1992]. The Causal Calculator conjoins the formulas describing an initial state and goal to the propositional theory, and then calls a propositional solver, such as SATO [Zhang, 1997], to find a model. If a model is found, it corresponds to a possible history of the domain, and the assignments it makes to the action symbols correspond to a plan. For deterministic domains, any plan found in this way is guaranteed to be valid [McCain and Turner, 1998].

A $\mathcal{C}$ input file for the Causal Calculator consists of declarations, propositions in language $\mathcal{C}$ (or, more often, schemas with metavariables whose instances are propositions in $\mathcal{C}$), problems (for instance, planning problems) and comments. Among its declarations, a $\mathcal{C}$ input file usually contains a directive to include the "standard" file `C.t` which contains rewrite rules for translating from $\mathcal{C}$ into the language of causal theories, as well as various sorts, variables, constants, and domain independent causal laws that have been found useful in formalizing action domains.

5.2 AIRPORT DOMAIN IN THE CAUSAL CALCULATOR

The input file for the airport domain (`airport-domain.t`) shown below corresponds to the formalization of the domain in $\mathcal{C}$ described in Section 4.2. The sort names `inertialFluent`, `defaultFalseFluent`, and `action` that appear below in the `constants` declaration are declared in `C.t`. The declarations

```
:- constants
 at(object,object)              :: inertialFluent ;
 walkable(object), drivable(object)
                                :: defaultFalseFluent.
```

are equivalent to the declarations

```
:- constants
 at(object,object)                   :: fluent ;
 walkable(object), drivable(object)  :: fluent.
```

together with the schemas postulated at the end of Section 4.2:

```
inertial at(X,Y), -at(X,Y).
default -walkable(X).
default -drivable(X).
```

(X and Y are meta-variables of sort `object`). Hopefully, these remarks will suffice to enable the reader to understand the following file.[3]

```
%%% File 'airport-domain.t'

:- macros
 disjoint(#1,#2) ->
         (-((#1)=(#2)) && -at(#1,#2) && -at(#2,#1)).

:- include 'C.t'.

:- sorts
 object >> region; mode.

:- variables
 X,Y,Z    :: object;
 U,V      :: region;
 M        :: mode.

:- constants
 i, car, desk                     :: object;
 home, airport, county            :: region;
 walking, driving                 :: mode;
 at(object,object)                :: inertialFluent;
 walkable(region), drivable(region)
                                  :: defaultFalseFluent;
 go(object,mode)                  :: action.

% Properties of at

caused  at(X,Z) if at(X,Y) && at(Y,Z).
caused -at(X,Z) if at(X,Y) && disjoint(Y,Z).
caused -at(X,X).

% Effects of actions

go(X,M)       causes at(i,X).
go(X,driving) causes at(car,X).

% Action preconditions

nonexecutable go(X,M) if at(i,X).

nonexecutable go(X,driving) if -at(i,car).
nonexecutable go(X,walking)
     if - (\/ U: (walkable(U) && at(i,U) && at(X,U))).

nonexecutable go(X,driving)
     if - (\/ U: (drivable(U) && at(i,U) && at(X,U))).

% Restrictions on the concurrent execution of actions

nonexecutable go(X,walking) && go(Y,driving).

nonexecutable go(X,M) && -go(Y,M)
           if at(X,Y) && -at(i,Y).

% Sizes of objects

always -at(car,desk).

caused walkable(home).
caused walkable(U) if at(U,V) && walkable(V).

constant walkable(U).

caused drivable(county).
caused drivable(U) if at(U,V) && drivable(V).

constant drivable(U).
```

5.3 PLANNING

Besides the description of the airport domain shown above, the input given to the Causal Calculator includes the description of the planning problem:

```
%%% File ’airport-problem.t’

:- include ’airport-domain.t’.

:- plan
facts ::
 0: at(i,desk),
 0: at(desk,home),
 0: at(car,home),
 0: at(home,county),
 0: at(airport,county),
 0: -at(desk,car),
 0: disjoint(home,airport);
```

```
goal ::
 2: at(i,airport).
```

Each of the lines beginning with the time stamp `0` is an initial condition. These conditions express that s_0 (Section 4.3) is the initial state of the planning problem. The line beginning with the time stamp `2` tells us that the goal is to make $at(i, airport)$ true after the execution of 2 actions.[4]

Given this input file, the Causal Calculator produces the following output:

```
calling sato...
run time (seconds)                0.08

0.  drivable(airport)  drivable(county)  drivable(home)
walkable(home)  at(airport,county)  at(car,county)
at(car,home)  at(desk,county) at(desk,home)
at(home,county)  at(i,county)  at(i,desk)  at(i,home)

ACTIONS: go(car,walking)

1.  drivable(airport)  drivable(county)  drivable(home)
walkable(home)  at(airport,county)  at(car,county)
at(car,home)  at(desk,county)  at(desk,home)
at(home,county)  at(i,car)  at(i,county)  at(i,home)

ACTIONS: go(airport,driving)

2.  drivable(airport)  drivable(county)  drivable(home)
walkable(home)  at(airport,county)  at(car,airport)
at(car,county)  at(desk,county)  at(desk,home)
at(home,county)  at(i,airport)  at(i,car)  at(i,county)
```

The output shows the SATO runtime, the actions to be executed at times 0 and 1, and the fluents that hold at each of the time instants 0, 1, 2 when the plan is executed.

6. CONCLUSION

The solution to the airport problem presented in this paper is based on a number of ideas—some old, some new—coming from several areas of logic-based AI.

Nonmonotonic Reasoning. As a nonmonotonic formalism, the input language of the Causal Calculator can be viewed as a subset of default logic in the sense of Reiter [1980]. The process of “literal completion” [McCain and Turner, 1997] that translates rules of causal logic into propositional formulas is a modification of the completion semantics of negation as failure in logic programming [Clark, 1978] that treats positive and negative literals in a symmetric way, in the style of [Gelfond and Lifschitz, 1991]. As discussed in [Lifschitz, 1997, Section 3], the formalization of the closed world assumption in causal logic can be viewed as the use of circumscription [McCarthy, 1986].

Frame Problem. The solution to the frame problem incorporated in $\mathcal{C}$ and in the Causal Calculator is related to the method of [Reiter, 1980, Section 1.1.4]. Claims to the opposite [Hanks and McDermott, 1987] notwithstanding, Reiter's solution turned out to be completely satisfactory after the rest of the postulates were formulated in the right way ([Turner, 1997, Section 5.2]; see also [Lifschitz, 1999b, Section 3]). The frame problem becomes more difficult in the presence of actions with indirect effects. This issue, known as the "ramification problem," was clarified over the last decade on the basis of the ideas of [Geffner, 1990] and [Lin and Reiter, 1994].

Action Languages. The idea of an action language as a language for describing transition systems was articulated by Pednault [1994]. After action language $\mathcal{A}$ was defined and related to logic programming in [Gelfond and Lifschitz, 1993], it was extended and modified by several authors. Language $\mathcal{C}$ is one of the most expressive action description languages introduced so far.

Satisfiability Planning. The success of this idea (due to Kautz and Selman [1992, 2000]) is determined by the remarkable progress in the development of fast propositional solvers. "Between 1991 and 1996 the size of hard satisfiability problems that could be feasibly solved grew from ones involving less than 100 variables to ones involving over 10,000 variables" [Selman *et al.*, 1997]. On the other hand, the Causal Calculator demonstrates that the method is applicable to planning problems described in very expressive languages. This last fact is particularly essential in case of the airport problem whose difficulty is representational rather than algorithmic.

The approach to actions and planning described in this paper represents only one of several streams of research in this area. References to the work not mentioned here can be found in [Shanahan, 1997] and in recent issues of the *Electronic Transactions on AI: Reasoning about Actions and Change*.[5]

Acknowledgements

We are grateful to Esra Erdem, Jack Minker and Hudson Turner for constructive criticisms. This work was partially supported by National Science Foundation under grant IIS-9732744.

Notes

1. The issue is, however, of some importance. The number of distinct actions in a plan is one useful measure of its quality. In order to count actions we have to be able to tell when we have two and when we have one.

2. `http://www.cs.utexas.edu/users/tag/cc/` . See [Lifschitz, 2000] for other examples of the use of this system.

3. Both this file and the file `airport-problem.t` are available on-line at `http://www.cs.utexas.edu/users/tag/ccalc/ccalc.1.23/examples/airport/`.

4. If the number of steps required to solve a planning problem is unknown, the Causal Calculator can explore a range of possibilities. For instance, by writing `goal :: 5-20` we instruct the Causal Calculator to look first for a plan of length 5; if there is no such plan, try 6, etc., up to 20.

5. `http://www.ida.liu.se/ext/etai/rac/` .

References

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

Donald Davidson. The logical form of action sentences. In *The Logic of Decision and Action*, pages 81–120. University of Pittsburgh Press, 1967.

Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in non-monotonic logic programs. In Sam Steel and Rachid Alami, editors, *Proc. European Conf. on Planning 1997*, pages 169–181. Springer-Verlag, 1997.

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.

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

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

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

Michael Gelfond and Vladimir Lifschitz. Action languages.[1] *Electronic Transactions on AI*, 3, 1998.

Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press, 1998.

Cordell Green. Application of theorem proving to problem solving. In Donald Walker and Lewis Norton, editors, *Proc. IJCAI*, pages 219–240. The MITRE Corporation, 1969.

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

Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

Henry Kautz and Bart Selman. Encoding domain and control knowledge for propositional planning. 2000. This volume.

Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

Vladimir Lifschitz. Answer set planning. In *Proc. ICLP-99*, pages 23–37, 1999.

Vladimir Lifschitz. Success of default logic. In Hector Levesque and Fiora Pirri, editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 208–212. Springer-Verlag, 1999.

Vladimir Lifschitz. Missionaries and cannibals in the causal calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pages 85–96, 2000.

Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4:655–678, 1994.

Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465, 1997.

Norman McCain and Hudson Turner. Satisfiability planning with causal theories. In Anthony Cohn, Lenhart Schubert, and Stuart Shapiro, editors, *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 212–223, 1998.

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 [McCarthy, 1990].

John McCarthy. Programs with common sense. In *Proc. Teddington Conf. on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationery Office. Reproduced in [McCarthy, 1990].

John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986. Reproduced in [McCarthy, 1990].

John McCarthy. *Formalizing Common Sense: Papers by John McCarthy*. Ablex, Norwood, NJ, 1990.

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.

Edwin Pednault. ADL and the state-transition model of action. *Journal of Logic and Computation*, 4:467–512, 1994.

Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyburg, R. Loui, and G. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer, 1990.

Bart Selman, Henry Kautz, and David McAllester. Ten challenges in propositional reasoning and search. In *Proc. IJCAI-97*, pages 50–54, 1997.

Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

V.S. Subrahmanian and Carlo Zaniolo. Relating stable models and AI planning domains. In *Proc. ICLP-95*, 1995.

Hudson Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.

Hantao Zhang. An efficient propositional prover. In *Proc. CADE-97*, 1997.