Formalizing Sensing Actions - a transition function based approach

Tran Cao Son
Knowledge Systems Laboratory
Computer Science Department
Stanford University, Stanford, CA 94305
tson@ksl.stanford.edu

Chitta Baral
Department of Computer Science Engineering
Arizona State University
Tempe, Arizona 85287
chitta@asu.edu

July 19, 2000

Abstract

In presence of incomplete information about the world we need to distinguish between the state of the world and the state of the agent's knowledge about the world. In such a case the agent may need to have at its disposal sensing actions that change its state of knowledge about the world and may need to construct more general plans consisting of sensing actions and conditional statements to achieve its goal. In this paper we first develop a high level action description language that allows specification of sensing actions and their effects in its domain description and allows queries with conditional plans. We give provably correct translations of domain description in our language to axioms in first-order logic, and relate our formulation to several earlier formulations in the literature. We then analyze the state space of our formulation and develop several sound approximations that have much smaller state spaces. Finally we define regression of knowledge formulas over conditional plans.

1 Introduction and motivation

Unlike actions that change the world, sensing or knowledge producing actions change what the agent knows about the world. Consider the following example of a high security door. The action of pushing the door (*push_door*) – when executed in a world where the (lock of the) door is initially unlocked and not jammed – will change the world so that after the action is performed the door is open. The same action if executed when the door is locked will jam the door. Similarly, the action of flipping the lock (*flip_lock*) will unlock a locked door and lock an unlocked door. On the other hand the *sensing action* of checking the lock of the door (*check_if_locked*) will result in the agent knowing if the door is locked or not.

Sensing actions play an important role when an agent needs to plan in presence of incomplete information. Consider the case when our agent initially (i.e., in the initial situation) does not know if the door is locked or not, but knows that the door is not jammed and is not open and its goal is to open the door. We will assume that the only actions it can perform are the ones described earlier: *check_if_locked*, *flip_lock* and *push_door*. We now argue that the agent can not just construct a classical plan – consisting of a sequence of actions – that will always (i.e., regardless of what the real state of the world is) succeed in reaching the agent's goal.

Let us first consider the plan P_1 consisting of *push_door*. This plan will not work if the door is initially locked. In fact it will jam the door, and no subsequent action sequence will result in the door being open. Let us now consider the plan P_2 consisting of *flip_lock*; *push_door*. This plan will not work if the door is initially unlocked. In fact it will also jam the door, and no subsequent action sequence will result in the door being open. Therefore, neither P_1, nor P_2, and nor any plan that starts with P_1 and P_2 will work in both cases. This, together with the fact that the action *check_if_locked* does not change the world and a sequence of *flip_lock*s is equivalent to zero or a single *flip_lock*, is enough to conclude that there does not exist a classical plan that will work for all possible initial situations.

The following simple conditional plan P_3,

IF $\neg$*door_locked* THEN *push_door* ELSE *flip_lock*;*push_door*

is not appropriate either. That is because the agent not knowing whether *door_locked* is true or not can not execute this plan. A correct conditional plan, P_4, that will always achieve the goal uses the sensing action *check_if_locked*, and is as follows:

check_if_locked;
IF $\neg$*door_locked* THEN *push_door* ELSE *flip_lock*;*push_door*;

Thus sensing actions are very important for planning in presence of incomplete information. In the past, sensing actions have been formalized in [Moo79, Moo85, SL93, Haa86, LTM97] and planning in presence of incomplete information has been studied in [EHW$^+$92, PS92, PC96, GB94, GEW96, GW96, Lev96, KOG92, SW98, WAS98, GB96]. To motivate our work we now briefly review the earlier formalizations of sensing actions.

1.1 Moore's formalization

To the best of our knowledge sensing actions were first formalized by Moore in his dissertation [Moo79] and in some of his later papers; for example, [Moo85]. Moore uses possible world semantics to represent knowledge and treats the accessibility relation between worlds as a fluent when reasoning about sensing and non-sensing actions.

- He describes how the knowledge of an agent may change after executing a non-sensing action a, by defining the accessibility relation between the worlds that may be reached after executing the action a.

  According to him, for any two possible worlds w_1 and w_2 such that w_2 is the result of the execution of a in w_1, the worlds that are compatible with what the agent knows in w_2 are exactly the worlds that are the result of executing a in some world that is compatible with what the agent knows in w_1. This can be formally written as follows:

$$\begin{array}{ll} \forall w_1, w_2.(w_2 = do(a, w_1) \supset & \\ \qquad\qquad \forall w_3. & (acc(w_2, w_3) \equiv \\ & \exists w_4.acc(w_1, w_4) \wedge w_3 = do(a, w_4))) \end{array} \tag{1.1}$$

  The above formula (and the next formula) is a simplified version of Moore's original formula. Here we use the function *do* from situation calculus[1], use $acc(w, w')$ to denote that w' is accessible from (or is compatible with) w, and assume a single agent world.

---

[1] $do(a, w)$ denotes the world reached after executing the action a in the world w.

- He also describes how the knowledge of an agent may change after executing a sensing action $sense_f$, by defining accessibility relation between the worlds that may be reached after executing $sense_f$.

  Suppose $sense_f$ is an action that the agent can perform to know if f is true or not. Then for any world represented by w_1 and w_2 such that w_2 is the result of $sense_f$ happening in w_1, the world that is compatible with what the agent knows in w_2 are exactly those worlds that are the result of $sense_f$ happening in some world that is compatible with what the agent knows in w_1, and in which f has the same truth value as in w_2. This can be formally written as follows:

$$\begin{array}{lll} \forall w_1, w_2. & (w_2 = & do(sense_f, w_1) \supset \\ & \forall w_3. & ((acc(w_2, w_3) \equiv \\ & & \exists w_4.acc(w_1, w_4) \wedge w_3 = do(sense_f, w_4) \wedge f(w_2) \equiv f(w_3))) \end{array} \tag{1.2}$$

1.2 Scherl and Levesque's formalization

Scherl and Levesque [SL93] adapted Moore's formulation to situation calculus and proved several important results about their formulation such as: knowledge-producing actions do not affect fluents other than the knowledge fluent; and that actions that are not knowledge-producing only affect the knowledge fluent as appropriate. They also showed how regression can be applied to knowledge-producing actions.

Their slight simplification of Moore's formulation is given by the following two formulas: (Note that in their use of the relation K, which we will follow in the rest of the paper, the arguments are reversed from their normal modal logic use. I.e., $K(\mathsf{s}', \mathsf{s})$ is read as "the situation s' is accessible from the situation s". Also, *situation* is a term constructed by repeated application of *do* to the initial situation S_0.)

$$K(\mathsf{s}'', do(a, \mathsf{s})) \equiv (\exists \mathsf{s}'.K(\mathsf{s}', \mathsf{s}) \wedge \mathsf{s}'' = do(a, \mathsf{s}')) \tag{1.3}$$

$$K(\mathsf{s}'', do(sense_f, \mathsf{s})) \equiv (\exists \mathsf{s}'.K(\mathsf{s}', \mathsf{s}) \wedge \mathsf{s}'' = do(sense_f, \mathsf{s}') \wedge f(\mathsf{s}') \equiv f(\mathsf{s})) \tag{1.4}$$

1.3 Our simplification

One of our goals in this paper is to make it easy to visualize the state space we have to deal with when searching for plans in presence of sensing actions and incomplete information. Many formulations of planning (for example, most research on decision theoretic planning) often assume the existence of a transition function defining a transition between states – a collection of fluents – due to actions, and do not necessarily depend on a logical formulation defining this function. *The questions that we would like to answer are: What is a "state" when we need to distinguish between the state of the world and the state of the knowledge of an agent? How are state transitions due to actions – both sensing and non-sensing – defined?*

To answer the first question we introduce the notion of a *c-state* (or combined state) which is a pair consisting of:

(i) the real state of the world, s; and

(ii) the state of the agent's knowledge about the world given by the set of states Σ, that the agent thinks it may be in.

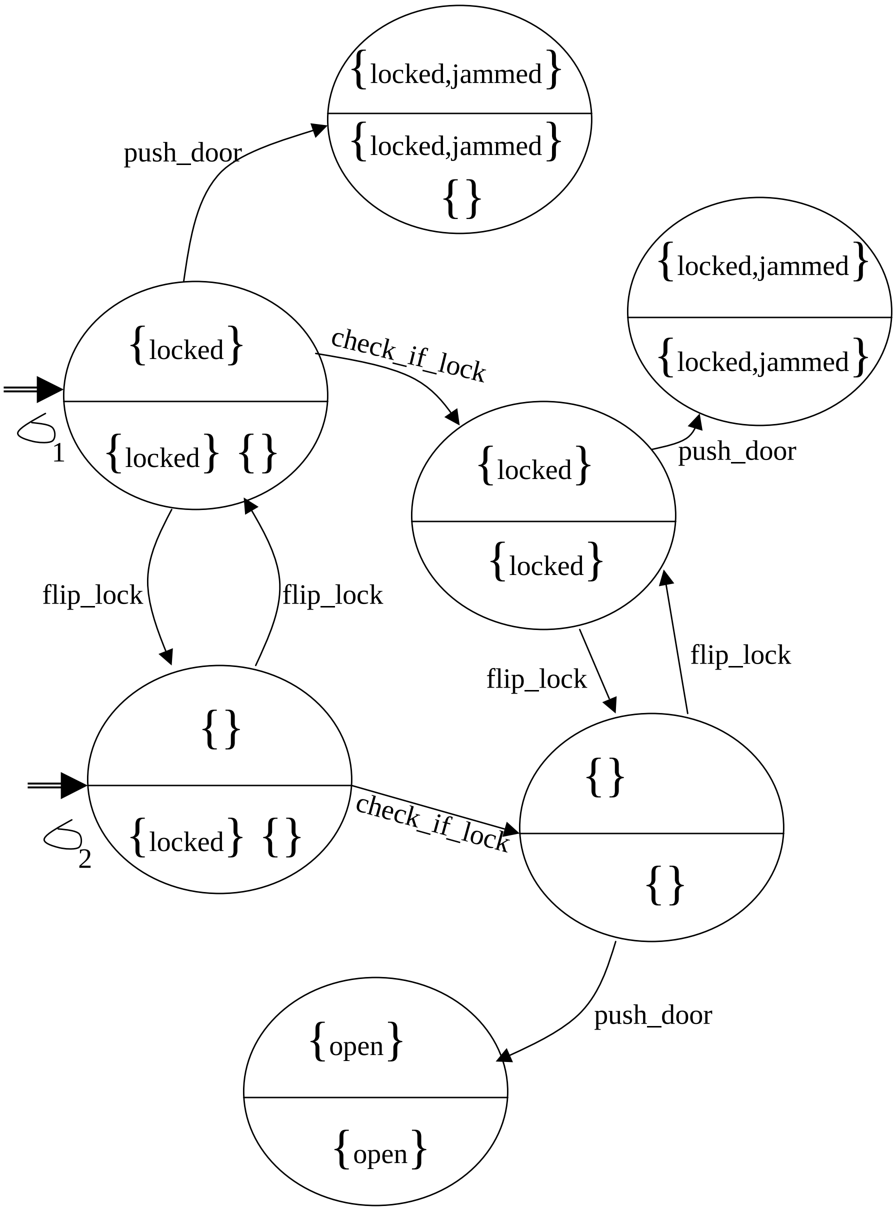
The transition between c-states due to actions – denoted by $\Phi(a, \langle s, \Sigma\rangle)$ – can then be defined in terms of the original transition between states (defined using the function Res) in the following way:

- If a is a non-sensing action then for any c-state $\sigma = \langle s, \Sigma\rangle$, $\Phi(a, \sigma)$ is defined as the pair $\langle Res(a, s), \{s' | s' = Res(a, s'') \text{ for some } s'' \in \Sigma\}\rangle$.
- If $sense_f$ is a sensing action that senses the fluent f then for any c-state $\sigma = \langle s, \Sigma\rangle$, $\Phi(sense_f, \sigma)$ is defined as the pair $\langle s, \{s' | s' \in \Sigma \text{ such that } f \in s \text{ iff } f \in s'\}\rangle$.

Consider our example in the beginning of this section. The two possible initial c-states – with explicit representation of negative fluents – for this example are:

$\sigma_1 = \langle\{locked\}, \{\{locked\}, \{\}\}\rangle$, and $\sigma_2 = \langle\{\}, \{\{locked\}, \{\}\}\rangle$.

In Figure 1 we give a fragment of the state space diagram of this example illustrating how transitions take place between one c-state to another because of actions.



For a logical formalization of the above we simplify Moore's and Scherl and Levesque's formulation by assuming that we only need to proceed from the K relation about the initial situation to possible future situations. The formulas (1.3) and (1.4) can then be modified as follows:

$$K(do(a, \mathsf{s}'), do(a, \mathsf{s})) \equiv K(\mathsf{s}', \mathsf{s}) \tag{1.5}$$

$$K(do(sense_f, \mathsf{s}'), do(sense_f, \mathsf{s})) \equiv (K(\mathsf{s}', \mathsf{s}) \wedge f(\mathsf{s}') \equiv f(\mathsf{s})) \tag{1.6}$$

Using the above two formulas, successor state axioms about actions [Rei91], and information about the initial situation, we can then reason about what is known to be true in a future situation. We discuss this formulation in further detail in Section 2.3.

1.4 Our goals

Our first goal in this paper is to augment the high-level language $\mathcal{A}$ [GL92, GL93] to allow specifications and reasoning about sensing actions. We will call the new language $\mathcal{A}_K$. The semantics of domain descriptions in $\mathcal{A}_K$ will be defined using the transition functions introduced in the previous sub-section. The motivation behind doing this is the simplicity of high level languages and the fact that no knowledge about particular logics is necessary to understand the concept. But we pay the price of being less general than when the formalization is done in a standard logical language (classical logic possibly augmented with circumscription, logic programming, default logic, etc.). But then later we give formalizations in logic, *and prove the correctness of our logical formalization* with respect to our original formalization. Thus our initial formalization using a high level language – which is simpler to follow – can play the role of a benchmark for formalizations in standard logical languages.

Our second goal, and perhaps the most important aspect of this paper, is to develop approximations of the language $\mathcal{A}_K$. The motivation behind that is the possible state space explosion in $\mathcal{A}_K$. In presence of n fluents, we will have 2^n possible states and 2^{2^n+n} possible *c-states*. We develop several approximations with much smaller state space (3^n) but with varying complexity in computing transitions. We then show the soundness of these approximations.

Finally, we relate our formulations with earlier formulations of sensing actions – in particular with Scherl and Levesque's [SL93] formulation and Lobo et al.'s [LTM97] formulation, and show that: (i) When we translate domain descriptions in our language to Scherl and Levesque's formulation we obtain similar conclusions, and (ii) When we make certain assumptions about our knowledge about the initial state then domain descriptions in our language have the same semantics as that of the semantics defined by Lobo et al. [LTM97]. We also discuss some of the earlier work on planning with sensing actions [GW96, Gol97, GB94, GB96], compare the formulations there with that of ours, and briefly describe earlier work on regression and adapt a simplified version of regression from [SL93] to define regression with respect to conditional plans.

2 The language $\mathcal{A}_K$

In this section we introduce $\mathcal{A}_K$ – an extension of the language $\mathcal{A}$ in [GL93] – which allows reasoning about sensing actions. (Strictly speaking, $\mathcal{A}_K$ is a variation of $\mathcal{A}$ instead of an extension, as unlike in $\mathcal{A}$, we do not allow observations or hypothesis about non-initial situations in our domain descriptions. Moreover, our language has two components [Lif97, BGP97]: one which defines domain descriptions and another which defines queries.)

2.1 Syntax of $\mathcal{A}_K$

We begin with two disjoint nonempty sets of symbols, called *fluent names* (or *fluents*) and *action names* (or *actions*). A *fluent literal* is either a fluent name or a fluent name preceded by $\neg$. For a fluent f, by $\overline{\neg f}$ we mean f, and by $\overline{f}$ we mean $\neg f$.

2.1.1 Domain descriptions in $\mathcal{A}_K$

A *v-proposition* (value proposition) is an expression of the form

$$\textbf{initially} \quad f \tag{2.1}$$

where f is a fluent literal. Intuitively, the above v-proposition means that the fluent literal f is initially *known to be true.* (In $\mathcal{A}$, where v-propositions describe the initial state of the world instead of what the agent knows about the initial state of the world, the above proposition has a slightly different meaning. There, the above proposition means that the fluent literal f is true in the initial state of the world.)

Two v-propositions **initially** f and **initially** g are said to be *contradictory* if $f = \overline{g}$.

An *ef-proposition* (effect proposition) is an expression of the form

$$a \ \textbf{causes} \ f \ \textbf{if} \ p_1, \ldots, p_n \tag{2.2}$$

where a is an action, and each of $f, p_1, \ldots, p_n$ $(n \geq 0)$ is a fluent literal. The set of fluent literals $\{p_1, \ldots, p_n\}$ is referred to as the *precondition* of the ef-proposition and f is referred to as the *effect* of this ef-proposition. Intuitively this proposition conveys the meaning that f is guaranteed to be true after the execution of an action a in any state of the world where $p_1, \ldots, p_n$ are true. If $n = 0$, we will drop the **if** part and simply write a **causes** f.

Two ef-propositions with preconditions $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$ respectively are said to be *contradictory* if they describe the effect of the same action a on complementary f's, and $\{p_1, \ldots, p_n\} \cap \{\overline{q_1}, \ldots, \overline{q_m}\} = \emptyset$.

An *ex-proposition* (executability proposition) is an expression of the form

$$\textbf{executable} \ a \ \textbf{if} \ p_1, \ldots, p_n \tag{2.3}$$

where a is an action, and each of $p_1, \ldots, p_n$ $(n \geq 0)$ is a fluent literal. Intuitively, this proposition conveys the meaning that the action a is executable in any state of the world where $p_1, \ldots, p_n$ are true. If $n = 0$, we will drop the **if** part and simply write **executable** a.

A *k-proposition* (knowledge proposition) is an expression of the form

$$a \ \textbf{determines} \ p \tag{2.4}$$

where a is an action and p is a fluent. Intuitively, the above proposition conveys the meaning that if a is executed in a situation, then in the resulting situation the truth value of p becomes known.

A *proposition* is a v-proposition, ef-proposition, ex-proposition or a k-proposition.

A *domain description* is a set of propositions, which does not contain

(i) contradictory v-propositions; or

(ii) contradictory ef-propositions.

Actions occurring in ef-propositions and k-propositions are called non-sensing actions and sensing actions, respectively. In this paper – to avoid distraction from the main points – we make the further assumption that the set of sensing actions and the set of non-sensing actions are disjoint. Following is an example of a domain description in our language.

Example 1 Let us consider an agent who has to *disarm* a bomb which can only be done safely – i.e., without *exploding* – if a special lock on the bomb has been switched off (*locked*); otherwise it explodes. The agent can determine if the lock is locked or not by *looking* at the lock. He can also *turn* the lock from the *locked* position to the *unlocked* position and vice-versa. He can only execute the above actions if the bomb has not exploded. Initially, the agent knows that the bomb is not disarmed and is not exploded. We can describe the above story by the following domain description.

$$\left.\begin{array}{l}
\textbf{initially}\ \neg disarmed \\
\textbf{initially}\ \neg exploded \\
disarm\ \textbf{causes}\ exploded\ \textbf{if}\ \neg locked \\
disarm\ \textbf{causes}\ disarmed\ \textbf{if}\ locked \\
turn\ \textbf{causes}\ \neg locked\ \textbf{if}\ locked \\
turn\ \textbf{causes}\ locked\ \textbf{if}\ \neg locked \\
look\ \textbf{determines}\ locked \\
\textbf{executable}\ look\ \textbf{if}\ \neg exploded \\
\textbf{executable}\ turn\ \textbf{if}\ \neg exploded \\
\textbf{executable}\ disarm\ \textbf{if}\ \neg exploded
\end{array}\right\} = D_1$$

2.1.2 Queries in $\mathcal{A}_K$

As discussed in Section 1, in the presence of incomplete information and knowledge producing actions, we need to extend the notion of a plan from a sequence of actions so as to allow conditional statements. In the following definition we formalize the notion of a conditional plan.

Definition 1 (Conditional Plan)

1. An empty sequence of action, denoted by [ ], is a conditional plan.

2. If a is an action then a is a conditional plan.

3. If $c_1, \ldots, c_n$ ($n \geq 1$) are conditional plans and $\varphi_1, \ldots, \varphi_n$ are conjunction of fluent literals, (which are mutually exclusive but not necessarily exhaustive) then the following is a conditional plan. (We refer to such a plan as a *case plan*).

   Case

   $\varphi_1 \rightarrow c_1$

   ...

   $\varphi_n \rightarrow c_n$

   Endcase

4. If c_1, c_2 are conditional plans then $c_1; c_2$ is a conditional plan.

5. Nothing else is a conditional plan. □

Intuitively, the case plan is a case statement where the agent evaluates the various φ_i's with respect to its knowledge. If it knows that φ_i is true for some i it executes the corresponding c_i. If none of the φ_i's are true then the case plan fails and the execution of the conditional plan which contains this case plan also fails.

There are two kind of queries that we can ask our domain descriptions. They are of the form:

$$\textbf{Knows } \varphi \textbf{ after } c \tag{2.5}$$

$$\textbf{Kwhether } \varphi \textbf{ after } c \tag{2.6}$$

where c is a conditional plan and φ is a fluent formula. Intuitively, the first query is about asking if a domain description entails that the fluent formula φ will be known to be true after executing the conditional plan c in the initial situation, and the second query is about asking if a domain description entails that the fluent formula φ will be known to be true or known to be false after executing the conditional plan c in the initial situation

2.2 Semantics of $\mathcal{A}_K$

In $\mathcal{A}_K$, we have three kinds of states: a world state (often referred to as a *state*) representing the state of the world, a *knowledge state* (or a k-state), representing the state of the knowledge of the agent, and a *combined state* (or a c-state) that is a pair consisting of a world state, and a k-state. As mentioned earlier, the semantics of domain descriptions in $\mathcal{A}_K$ are defined in terms of models which are pairs consisting of an initial c-state and a transition function that maps pairs of actions and c-states into c-states.

In the following we will use small letters beginning from s (possibly with indexes) to denote world states, uppercase Greek letters like Σ (possibly with indexes) to denote k-states, and lowercase Greek letters like σ, δ (possibly with indexes) to denote c-states. The letter c (possibly with indexes) will be used exclusively to denote conditional plans while α (possibly with indexes) will be used to denote a sequence of actions.

A *state* s is a set of fluents and a *k-state* is a set of states. A *combined state* (or *c-state*) of an agent is a pair $\langle s, \Sigma \rangle$ where s is a state and Σ is a k-state. Intuitively, the state s in a c-state $\langle s, \Sigma \rangle$ is the real state of the world whereas Σ is the set of possible states which an agent believes it might be in. We say a c-state $\sigma = \langle s, \Sigma \rangle$ is *grounded* if $s \in \Sigma$. Intuitively, grounded c-states correspond to the assumption that the world state belongs to the set of states that the agent believes it may be in.

Given a fluent f and a state s, we say that f holds in s (f is true in s) if $f \in s$; $\neg f$ holds in s (f is false in s) if $f \notin s$. The truth of a propositional fluent formula w.r.t. s is defined as usual. We say two states s and s' *agree on a fluent* f if ($f \in s$ iff $f \in s'$). Given a c-state $\sigma = \langle s, \Sigma \rangle$, we say that a fluent f is *known to be true* (resp. *known to be false*) in $\langle s, \Sigma \rangle$ if f is true (resp. false) in every state $s' \in \Sigma$; and f is *known* in $\langle s, \Sigma \rangle$, if f is known to be true or known to be false in $\langle s, \Sigma \rangle$. Given a fluent formula φ, we say that φ is known to be true (resp. false) in a c-state $\langle s, \Sigma \rangle$ if φ is true (resp. false) in every state $s' \in \Sigma$.

An action a is executable in a state s, if there exists an ex-proposition **executable** a **if** $p_1, \ldots, p_n$ in D such that $p_1, \ldots, p_n$ hold in s.

For an action a and a state s, if a is executable in s, we define

$E_a^+(s) = \{f \mid f$ is a fluent and there exists an ef-proposition "a **causes** f **if** $p_1, \ldots, p_n$" $\in D$ such that $p_1, \ldots, p_n$ hold in $s\}$,

$E_a^-(s) = \{f \mid f$ is a fluent and there exists an ef-proposition "a **causes** $\neg f$ **if** $p_1, \ldots, p_n$" $\in D$ such that $p_1, \ldots, p_n$ hold in $s\}$, and

$$Res(a, s) = s \cup E_a^+(s) \setminus E_a^-(s).$$

If a is not executable in s, we say that $Res(a, s)$ is undefined.

Intuitively, $Res(a, s)$ is the state resulting from executing a in s. Since we do not allow contradictory ef-propositions in our domain description, for any pair of an action a and a state s, $E_a^+(s)$ and $E_a^-(s)$ are disjoint and uniquely determined. Thus Res is a deterministic function.

We are now ready to define Φ, the transition function between c-states.

Definition 2 A function Φ from actions and c-states into c-states is called a *transition function* of D if for all c-state $\sigma = \langle s, \Sigma \rangle$ and action a,

1. if a is not executable in s then $\Phi(a, \sigma)$ is undefined, denoted by $\Phi(a, \sigma) = \perp$;
2. if a is executable in s and a is a non-sensing action, then

   $$\Phi(a, \sigma) = \langle Res(a, s), \{s' \mid s' = Res(a, s'') \text{ for some } s'' \in \Sigma \text{ such that } a \text{ is executable in } s''\} \rangle;$$

   and
3. if a is executable in s and a is a sensing action whose k-propositions are
   a **determines** f_1 ... a **determines** f_m, then

   $$\Phi(a, \sigma) = \langle s, \{s' \mid s' \in \Sigma \text{ such that } s \text{ and } s' \text{ agree on each } f_i, (i \leq m), \text{ and } a \text{ is executable in } s'\} \rangle.$$

Since Res is a deterministic function, it is easy to show the following:

Proposition 1 *Every domain description D possesses a unique transition function Φ.* □

Notice that our definition of the transition function Φ does not stipulate any special requirement on how the Res function is defined. Thus, any action description language [BG97, KL94, Tur97] with a semantics depending on a state transition function like Res can be extended to allow sensing actions. Therefore, several of the other features of action description languages such as multi-valued fluents [GKL97], ramification [LR94, KL94], causality [Lin95, MT95, Bar95], concurrent actions [LS92, BG93, BG97], can be directly added to our framework. For example, to extend our formulation to multi-valued fluents, we have to: (i) extend our propositions to be able to denote different values of the fluents, and (ii) extend our notion of states to be interpretations of the fluents. The definition of transition function will remain the same, except that the notion of s and s' agreeing on a fluent f would now mean that s and s' have the same value of f. To keep our focus on the main issue of formalizing sensing actions, we do not include these features in our formulation, as they can be directly added when desired.

Definition 3

1. A state s is called an *initial state* of a domain description D if for every value proposition of the form "**initially** p" (resp. "**initially** $\neg p$") in D, p is true (resp. false) in s.

2. A c-state $\langle s_0, \Sigma_0 \rangle$ is an *initial c-state* of D if s_0 is an initial state and Σ_0 is a set of initial states of D.

We say an initial c-state $\sigma_0 = \langle s_0, \Sigma_0 \rangle$ is *complete* if Σ_0 is the set of all initial states. Intuitively, the completeness of initial c-states express the assumption that our agent has complete knowledge about what it knows and does not know about the initial state. We will refer to this as the *complete awareness assumption*[2]. Even though, we believe that this assumption should not be used indiscriminately, *since it reduces the number of initial c-states, we will use it in most of our examples.*

Definition 4 A model of a domain description D is a pair (σ_0, Φ) such that σ_0 is a grounded initial c-state of D and Φ is a transition function of D. A model (σ_0, Φ) is called *rational* if σ_0 is complete.

Since the transition function Φ as defined so far can only tell us which c-state is reached after executing *an action* in a given c-state, we need to extend the function to be able to reason – beyond action sequences – about conditional plans. We call it the extended function of Φ and define it as follows.

Definition 5 Let D be a domain description and Φ be its transition function. The extended transition function of D, denoted by $\hat{\Phi}$, which maps pairs of conditional plans and c-states into c-states, is defined as follows.

1. $\hat{\Phi}([], \sigma) = \sigma$;
2. For an action a, $\hat{\Phi}(a, \sigma) = \Phi(a, \sigma)$;
3. For $c =$ Case
   $\varphi_1 \rightarrow c_1$
   $\dots$
   $\varphi_n \rightarrow c_n$
   Endcase,

$$\hat{\Phi}(c, \sigma) = \begin{cases} \hat{\Phi}(c_i, \sigma) \text{ if } \varphi_i \text{ is known to be true in } \sigma, \\ \\ \bot \quad \text{if none of } \varphi_1, \dots, \varphi_n \text{ is known to be true in } \sigma; \end{cases}$$

4. For $c = c_1; c_2$, where c_1, c_2 are conditional plans, $\hat{\Phi}(c, \sigma) = \hat{\Phi}(c_2, \hat{\Phi}(c_1, \sigma))$;
5. $\hat{\Phi}(c, \bot) = \bot$ for every conditional plan c.

We say that a conditional plan c is executable in a c-state σ if $\hat{\Phi}(c, \sigma) \neq \bot$. [3]

We are now ready to define the entailment relation for domains of $\mathcal{A}_K$.

Definition 6 Let D be a domain description, c be a conditional plan, and φ be a fluent formula. We say,

---

[2] Turner [Tur94] used a similar assumption called "complete initial situation assumption" according to which each model of his logic programming formulation of actions would have complete information about the initial state.

[3] It is easy to see that for every pair of a c-state σ and an action a, $\hat{\Phi}(c, \sigma) = \bot$ or there exists a unique c-state σ' such that $\hat{\Phi}(c, \sigma) = \sigma'$.

(i) $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c if c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ for every model (σ_0, Φ) of D;

(ii) $D \models_{\mathcal{A}_K}$ **Kwhether** φ **after** α if c is executable in σ_0 and φ is known to be true or known to be false in $\hat{\Phi}(\alpha, \sigma_0)$ for every model (σ_0, Φ) of D.

Rational entailment of queries w.r.t. D – denoted by $\models^r_{\mathcal{A}_K}$ – is defined similarly by only considering rational models of D. $\Box$

The following examples elucidates the above definitions.

Example 2 Let D_2 be the domain description consisting of the following propositions.

$$\left.\begin{array}{l} \textbf{initially } f \\ a \textbf{ causes } \neg f \\ sense_g \textbf{ determines } g \\ \textbf{executable } a \\ \textbf{executable } sense_g \end{array}\right\} = D_2$$

Let $s_1 = \{f, g\},\ s_2 = \{f\},\ s_3 = \{g\},\ s_4 = \emptyset$.

There are two possible complete initial c-states of D_2: $\sigma_1 = \langle s_1, \{s_1, s_2\}\rangle$ and $\sigma_2 = \langle s_2, \{s_1, s_2\}\rangle$. Let Φ be the transition function of D_2. We then have:

$$\begin{array}{l} \hat{\Phi}([a], \sigma_1) = \Phi(a, \sigma_1) = \langle s_3, \{s_3, s_4\}\rangle \\ \hat{\Phi}([a; sense_g], \sigma_1) = \hat{\Phi}([sense_g], \langle s_3, \{s_3, s_4\}\rangle) = \langle s_3, \{s_3\}\rangle \\ \\ \hat{\Phi}([a], \sigma_2) = \langle s_4, \{s_3, s_4\}\rangle \\ \hat{\Phi}([a; sense_g], \sigma_2) = \hat{\Phi}([sense_g], \langle s_4, \{s_3, s_4\}\rangle) = \langle s_4, \{s_4\}\rangle. \end{array}$$

Since g is known to be true in $\langle s_3, \{s_3\}\rangle$ and known to be false in $\langle s_4, \{s_4\}\rangle$, we can conclude that $D_2 \models^r_{\mathcal{A}_K}$ **Kwhether** g **after** $[a, sense_g]$.

However, $D_2 \not\models^r_{\mathcal{A}_K}$ **Kwhether** g **after** $[a]$, because g is not known to be true or known to be false in $\langle s_3, \{s_3, s_4\}\rangle$. Furthermore,

$D_2 \not\models^r_{\mathcal{A}_K}$ **Knows** g **after** $[a, sense_g]$, and

$D_2 \not\models^r_{\mathcal{A}_K}$ **Knows** $\neg g$ **after** $[a, sense_g]$. $\Box$

In the following example we consider conditional plans.

Example 3 Let us consider the domain description D_1 from Example 1. The states of D_1 are:

$$\begin{array}{ll} s_1 = \emptyset. & s_5 = \{disarmed\} \\ s_2 = \{locked\} & s_6 = \{disarmed, exploded\} \\ s_3 = \{exploded\} & s_7 = \{disarmed, locked\} \\ s_4 = \{locked, exploded\} & s_8 = \{disarmed, locked, exploded\} \end{array}$$

The set of initial states of D_1 is $\Sigma_0 = \{s_1, s_2\}$ and the two complete initial c-states of D_1 are $\sigma_1 = \langle s_1, \Sigma_0\rangle$ and $\sigma_2 = \langle s_2, \Sigma_0\rangle$. Let Φ be the transition function of D_1. Thus, by Definition 4, D_1

has two rational models: (σ_1, Φ) and (σ_2, Φ). We have:

$$\left.\begin{array}{ll}
\hat{\Phi}([look], \sigma_1) = \langle s_1, \{s_1\}\rangle & \hat{\Phi}([look], \sigma_2) = \langle s_2, \{s_2\}\rangle \\
\hat{\Phi}([look; disarm], \sigma_1) = \langle s_3, \{s_3\}\rangle & \hat{\Phi}([look; disarm], \sigma_2) = \langle s_7, \{s_7\}\rangle \\
\hat{\Phi}([look; turn], \sigma_1) = \langle s_2, \{s_2\}\rangle & \hat{\Phi}([look; turn], \sigma_2) = \langle s_1, \{s_1\}\rangle \\
\hat{\Phi}([look; turn; disarm], \sigma_1) = \langle s_7, \{s_7\}\rangle & \hat{\Phi}([look; turn; disarm], \sigma_2) = \langle s_3, \{s_3\}\rangle
\end{array}\right\} \quad (2.7)$$

Based on the above computation we have the following:

$D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** *disarmed* **after** $[look; disarm]$ and

$D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** *disarmed* **after** $[look; turn; disarm]$.

In Proposition 9 (Appendix A) we show that there exists no sequence of actions α of D_1 such that $D_1 \models^r_{\mathcal{A}_K}$ **Knows** $disarmed \wedge \neg exploded$ **after** α.

Let us now consider the conditional plan:

$$\left.\begin{array}{l}
look; \\
\left.\begin{array}{l}
\text{Case} \\
\neg locked \to turn \\
\quad locked \to [\,] \\
\text{Endcase}
\end{array}\right\} = c_1 \\
disarm
\end{array}\right\} = c$$

We will show that $D_1 \models^r_{\mathcal{A}_K}$ **Knows** $disarmed \wedge \neg exploded$ **after** c.

From the definition of $\hat{\Phi}$ and the computation of $\hat{\Phi}$ in (2.7), we have the following:

$$\begin{array}{rcl}
\hat{\Phi}(c, \sigma_1) & = & \hat{\Phi}(c_1; disarm, \hat{\Phi}(look, \sigma_1)) = \hat{\Phi}(c_1; disarm, \Phi(look, \sigma_1)) \\
& = & \hat{\Phi}(c_1; disarm, \langle s_1, \{s_1\}\rangle) = \hat{\Phi}(disarm, \hat{\Phi}(c_1, \langle s_1, \{s_1\}\rangle)) \\
& = & \hat{\Phi}(disarm, \hat{\Phi}(turn, \langle s_1, \{s_1\}\rangle)) \\
& & \quad \text{(because } \neg locked \text{ is known to be true in } \langle s_1, \{s_1\}\rangle) \\
& = & \hat{\Phi}(disarm, \Phi(turn, \langle s_1, \{s_1\}\rangle)) \\
& = & \hat{\Phi}(disarm, \langle s_2, \{s_2\}\rangle)) = \Phi(disarm, \langle s_2, \{s_2\}\rangle)) \\
& = & \langle s_7, \{s_7\}\rangle
\end{array}$$

and

$$\begin{array}{rcl}
\hat{\Phi}(c, \sigma_2) & = & \hat{\Phi}(c_1; disarm, \hat{\Phi}(look, \sigma_2)) = \hat{\Phi}(c_1; disarm, \Phi(look, \sigma_2)) \\
& = & \hat{\Phi}(c_1; disarm, \langle s_2, \{s_2\}\rangle) = \hat{\Phi}(disarm, \hat{\Phi}(c_1, \langle s_2, \{s_2\}\rangle)) \\
& = & \hat{\Phi}(disarm, \hat{\Phi}([\,], \langle s_2, \{s_2\}\rangle)) \\
& & \quad \text{(because } locked \text{ is known to be true in } \langle s_2, \{s_2\}\rangle) \\
& = & \hat{\Phi}(disarm, \langle s_2, \{s_2\}\rangle)) = \Phi(disarm, \langle s_2, \{s_2\}\rangle)) \\
& = & \langle s_7, \{s_7\}\rangle.
\end{array}$$

So, $\hat{\Phi}(c, \sigma_1) = \langle s_7, \{s_7\}\rangle$ and $\hat{\Phi}(c, \sigma_2) = \langle s_7, \{s_7\}\rangle$. Since $disarmed \wedge \neg exploded$ is known to be true in the c-state $\langle s_7, \{s_7\}\rangle$, by Definition 6, $D_1 \models^r_{\mathcal{A}_K}$ **Knows** $disarmed \wedge \neg exploded$ **after** c. □

2.3 Translating domain descriptions to first order theories

In this section we give a translation of domain descriptions (D) in $\mathcal{A}_K$ to theories in first-order logic ($R(D)$), and then show that the translation is sound and complete w.r.t. $\mathcal{A}_K$ when answering queries in the language of $\mathcal{A}_K$. Our translation from D into $R(D)$ is inspired by the translation in Kartha [Kar93], and uses axioms and notations from [SL93] and [Rei91]. In this section we use the standard notation of having variables start with small letters and constants start with capital letters. To be consistent we use the same notation for domain descriptions.

Let us consider a domain description D. Assume that D contains

1. $\mathbf{n}$ sensing actions $K_1, \ldots, K_{\mathbf{n}}$ with the k-propositions K_i **determines** F_i for $(1 \leq i \leq \mathbf{n})$, and
2. $\mathbf{m}$ value-propositions **initially** G_i for $(1 \leq i \leq \mathbf{m})$.

For simplicity, we also assume that each action A in D occurs in at least one executability condition and each sensing action K_i occurs in only one k-proposition. Then, the domain description D can be translated into a many-sorted theory $R(D)$ as follows.

Objects of $R(D)$ are of the sorts: *action, fluent*, and *situation.* To distinguish with states - which are often denoted by s (possibly with subscripts) - in the previous sections, we use s or S (possibly with subscripts) to denote situations. The vocabulary (signature) of $R(D)$ consists of the following:

- a constant S_0 of type situation;
- constants A of type "action" which correspond to different actions from D (one constant for each action);
- constants F of type "fluent" which correspond to different fluents from D (one constant for each fluent);
- a function symbol *do* of the type $\langle action \times situation \rightarrow situation \rangle$;
- a predicate symbol *Holds* of the type $\langle fluent, situation \rangle$;
- a predicate symbol K of the type $\langle situation, situation \rangle$;

We will need the following notations.

- For a fluent F, $Holds(\neg F, \mathsf{s})$ stands for $\neg Holds(F, \mathsf{s})$.
- For a conjunction of literals $\varrho = P_1 \wedge \ldots \wedge P_n$, $Holds(\varrho, \mathsf{s})$ denotes $\bigwedge_{i=1}^{n} Holds(P_i, \mathsf{s})$.
- For each fluent F and action A,

$$\gamma_F^+(A, \mathsf{s}) \overset{def}{\equiv} \bigvee_{\text{“}A \textbf{ causes } F \textbf{ if } \varrho\text{”} \in D} Holds(\varrho, \mathsf{s}),$$

$$\gamma_F^-(A, \mathsf{s}) \overset{def}{\equiv} \bigvee_{\text{“}A \textbf{ causes} \neg F \textbf{ if } \varrho\text{”} \in D} Holds(\varrho, \mathsf{s}), \text{ and}$$

$$Poss(A, \mathsf{s}) \overset{def}{\equiv} \bigvee_{\text{“}\textbf{executable } A \textbf{ if } \varrho\text{”} \in D} Holds(\varrho, \mathsf{s}).$$

The axioms of $R(D)$ are described below.

1. The successor state axiom – using Reiter's formulation in [Rei91] – for an ordinary fluent F and an action A is given by:

$$Poss(A, \mathsf{s}) \supset [Holds(F, do(A, \mathsf{s})) \equiv \gamma_F^+(A, \mathsf{s}) \vee (Holds(F, \mathsf{s}) \wedge \neg\gamma_F^-(A, \mathsf{s}))] \tag{2.8}$$

2. The successor state axiom for K (borrowed from [SL93]) and an action A is given by:

$$\begin{gathered} Poss(A, \mathsf{s}) \supset [K(\mathsf{s}'', do(A, \mathsf{s})) \equiv \\ \exists \mathsf{s}' \ (K(\mathsf{s}', \mathsf{s}) \wedge Poss(A, \mathsf{s}') \wedge (\mathsf{s}'' = do(A, \mathsf{s}'))) \wedge \\ ((\bigwedge_{j=1}^{\mathbf{n}} A \neq K_j) \vee (\bigvee_{j=1}^{\mathbf{n}} (A = K_j \wedge Holds(F_j, \mathsf{s}) \equiv Holds(F_j, \mathsf{s}'))))] \end{gathered} \tag{2.9}$$

   where, recall that, $K_1, \ldots, K_{\mathbf{n}}$ are the sensing actions in D that determine $F_1, \ldots, F_{\mathbf{n}}$ respectively.

3. For $i = 1, \ldots, \mathbf{m}$, $R(D)$ contains

$$Holds(G_i, \mathsf{S}_0) \tag{2.10}$$

   where, recall that, $G_1, \ldots, G_{\mathbf{m}}$, are the only fluent literals known to be true in the initial state.

4. The following axioms are for the accessibility relation in the initial situation:

$$K(\mathsf{s}, \mathsf{S}_0) \supset \bigwedge_{i=1}^{\mathbf{m}} Holds(G_i, \mathsf{s}). \tag{2.11}$$

   and

$$K(\mathsf{S}_0, \mathsf{S}_0). \tag{2.12}$$

5. The domain closure assumption (DCA) for fluents:

$$\bigvee_{F \in \mathbf{F}} f = F.$$

6. The domain closure assumption (DCA) for actions:

$$\bigvee_{A \in \mathbf{A}} a = A.$$

7. The unique name assumption (UNA) for fluents:

$$\bigwedge_{F_1, F_2 \in \mathbf{F} \ F_1, F_2 \text{ distinct}} F_1 \neq F_2.$$

8. The unique name assumption (UNA) for actions:

$$\bigwedge_{A_1, A_2 \in \mathbf{A} \ A_1, A_2 \text{ distinct}} A_1 \neq A_2.$$

We now relate the entailment in D and the entailment in $R(D)$, for queries regarding fluent values after a sequence of actions. We use the following notation:

- $Holds(\varphi, \mathsf{s})$ is a shorthand for a corresponding formula of $Holds$ with only fluents as its first argument. For example, $Holds(f_1 \vee f_2, \mathsf{s})$ denotes $Holds(f_1, \mathsf{s}) \vee Holds(f_2, \mathsf{s})$. Similarly, $Holds(f_1 \wedge f_2, \mathsf{s})$ denotes $Holds(f_1, \mathsf{s}) \wedge Holds(f_2, \mathsf{s})$, and as we mentioned before $Holds(\neg f, \mathsf{s})$ denotes $\neg Holds(f, \mathsf{s})$.
- $Knows(\varphi, \mathsf{S})$ denotes the formula: $\forall \mathsf{s}'(K(\mathsf{s}', \mathsf{S}) \supset Holds(\varphi, \mathsf{s}'))$, and
- for a sequence of actions $\alpha = [a_1; \ldots; a_k]$

  $do([], \mathsf{s})$ denotes s,
  $do(\alpha, \mathsf{s})$ denotes $do(a_k, do(a_{k-1}, \ldots, do(a_1, \mathsf{s})))$,
  $Poss([], \mathsf{s}) \equiv true$, and
  $Poss(\alpha, \mathsf{s})$ denotes $\bigwedge_{i=1}^{k} Poss(a_i, do([a_1; \ldots; a_{i-1}], \mathsf{s}))$.

Proposition 2 Let D be a domain description, φ be a fluent formula, and α be a sequence of actions of D. Then,

$$D \models_{\mathcal{A}_K} \textbf{Knows } \varphi \textbf{ after } \alpha \quad \text{iff} \quad R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0).$$

Proof. In Appendix B. □

Our next step is to relate D and $R(D)$ for queries with conditional plans. For that we introduce a three-sorted predicate $Apply(c, \mathsf{s}, \mathsf{s}')$, whose intuitive meaning is that the conditional plan c executed in situation s takes us to the situation s'. For example, let c be the conditional plan in Example 3, and s be a situation where $\neg locked$ holds in the real world. Then $Apply(c, \mathsf{s}, do(disarm, do(turn, do(look, \mathsf{s}))))$ will be true. Intuitively, this means that when c is executed in s, we reach the situation $do(disarm, do(turn, do(look, \mathsf{s})))$, or if c were to be executed in s, then the action sequence that would be executed from left to right is $look; turn; disarm$.

The definition of 'apply' is similar to the formula 'Rdo' in [Lev96]. In our formulation, we will represent a case plan as a list of pairs of conditions and conditional plans using three constructor functions: one that constructs a list, another that constructs a pair, and one that constructs a case plan. Any conditional plan can be represented as a list of actions and case plans. For example, the conditional plan c in Example 3 is represented by $[look;\ case([([\neg locked], [turn]), ([locked], [])]);\ disarm]$. We now define $Apply$ as a nested abnormality theory (NAT) [Lif95] block.

$$\begin{array}{l}
B_{\text{Apply}} = \\
\ \{\min \quad Apply \ : \\
\qquad\qquad Apply([], \mathsf{s}, \mathsf{s}) \\
\qquad\qquad Poss(a, \mathsf{s}) \wedge Apply(\alpha, do(a, \mathsf{s}), \mathsf{s}') \supset Apply([a|\alpha], \mathsf{s}, \mathsf{s}') \\
\qquad\qquad \neg Poss(a, \mathsf{s}) \supset Apply([a|\alpha], \mathsf{s}, \bot) \\
\qquad\qquad Apply([case([])|c], \mathsf{s}, \bot) \\
\qquad\qquad Apply(c, \bot, \bot) \\
\qquad\qquad Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{s}, \mathsf{s}') \wedge Apply(c'', \mathsf{s}', \mathsf{s}'') \supset Apply([case([(\varphi, c)|r'])|c''], \mathsf{s}, \mathsf{s}'') \\
\qquad\qquad \neg Knows(\varphi, \mathsf{s}) \wedge Apply([case(r')|c''], \mathsf{s}, \mathsf{s}')) \supset Apply([case([(\varphi, c)|r'])|c''], \mathsf{s}, \mathsf{s}') \\
\ \}
\end{array}$$

In the above nested abnormality theory c and c'' are conditional plans while r' is a list of pairs of conditions and conditional plans. (Note that $case(r')$ will denote a conditional plan.). The above NAT defines the predicate *Apply* using circumscription and can be equivalently written as $Circ(T; Apply)$, where T is the set of seven axioms following "*min Apply* :" in B_{Apply}. That is, we consider only models of T in which the predicate *Apply* is minimized. This guarantees that every situation is the result of execution of a conditional plan from the initial situation. For more on nested abnormal theories, please see Appendix E.

The NAT B_{Apply} can be defined in words as follows:

- $Apply([], \mathsf{s}, \mathsf{s})$ is true, for all s.
- For all $a, \alpha, \mathsf{s}, \mathsf{s}'$, $Apply([a|\alpha], \mathsf{s}, \mathsf{s}')$ is true if $Apply(\alpha, do(a, \mathsf{s}), \mathsf{s}') \wedge Poss(a, \mathsf{s})$ is true.
- For all $a, \alpha, \mathsf{s}, \mathsf{s}'$, $Apply([a|\alpha], \mathsf{s}, \bot)$ is true if $Apply(\alpha, do(a, \mathsf{s}), \mathsf{s}') \wedge \neg Poss(a, \mathsf{s})$ is true.
- $Apply([case([])|c], \mathsf{s}, \bot)$ is true for all c and s.
- $Apply(c, \bot, \bot)$ is true for all c.
- For all $\varphi, \mathsf{s}, \mathsf{s}', \mathsf{s}'', c, r', c''$, $Apply([case([(\varphi, c)|r'])|c''], \mathsf{s}, \mathsf{s}'')$ is true if $Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{s}, \mathsf{s}') \wedge Apply(c'', \mathsf{s}', \mathsf{s}'')$ is true.
- For all $\varphi, \mathsf{s}, \mathsf{s}', c, r', c''$, $Apply([case([(\varphi, c)|r'])|c''], \mathsf{s}, \mathsf{s}')$ is true if $\neg Knows(\varphi, \mathsf{s}) \wedge Apply([case(r')|c''], \mathsf{s}, \mathsf{s}'))$ is true.
- If none of the above rules is applicable then $Apply(c, \mathsf{s}, \mathsf{s}')$ is false.

We now explain how the above definition entails $Apply([a_1; a_2; a_3], \mathsf{s}, do(a_3, do(a_2, do(a_1, \mathsf{s}))))$, assuming that $Poss([a_1; a_2; a_3], \mathsf{s})$ is true. We have that

- $Apply([a_1; a_2; a_3], \mathsf{s}, do(a_1, do(a_2, do(a_3, \mathsf{s}))))$ is true if $Poss(a_1, \mathsf{s})$ and $Apply([a_2; a_3], do(a_1, \mathsf{s}), do(a_3, do(a_2, do(a_1, \mathsf{s}))))$ is true. (using the second rule)
- $Apply([a_2; a_3], do(a_1, \mathsf{s}), do(a_1, do(a_2, do(a_3, \mathsf{s}))))$ is true if $Poss(a_2, do(a_1, \mathsf{s}))$ and $Apply([a_3], do(a_2, do(a_1, \mathsf{s})), do(a_3, do(a_2, do(a_1, \mathsf{s}))))$ is true. (using the second rule)
- $Apply([a_3], do(a_2, do(a_1, \mathsf{s})), do(a_1, do(a_2, do(a_3, \mathsf{s}))))$ is true if $Poss([a_1; a_2; a_3], \mathsf{s})$ and $Apply([], do(a_3, do(a_2, do(a_1, \mathsf{s}))), do(a_3, do(a_2, do(a_1, \mathsf{s}))))$ is true. (using the second rule)
- $Apply([], do(a_3, do(a_2, do(a_1, \mathsf{s}))), do(a_3, do(a_2, do(a_1, \mathsf{s}))))$ is true. (using the first rule)

Proposition 3 Let D be a domain description and $R(D)$ be the corresponding first order theory. Let c be a conditional plan and φ be a fluent formula. Then,

$$D \models \textbf{Knows}\ \varphi\ \textbf{after}\ \ c \quad \text{iff} \quad R(D) \cup B_{\mathrm{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \bot\ .$$

Proof. In Appendix B. □

We would like to point out that the above proposition also holds for a slightly different translation $R_1(D)$, where we use the following simpler successor state axiom – based on the formulas (1.5) and (1.6) of Section 1.3 – instead of the successor state axiom (2.9):

$$\begin{array}{c} Poss(x, \mathsf{s}) \wedge Poss(x, \mathsf{s}') \supset [K(do(x, \mathsf{s}'), do(x, \mathsf{s})) \equiv \\ (K(\mathsf{s}', \mathsf{s}) \wedge ((\bigwedge_{j=1}^{\mathbf{n}} x \neq K_j) \vee (\bigvee_{j=1}^{\mathbf{n}} (x = K_j \wedge Holds(F_j, \mathsf{s}) \equiv Holds(F_j, \mathsf{s}'))))] \end{array} \tag{2.13}$$

2.4 State space analysis

In this section we analyze the size of the state space, when reasoning in $\mathcal{A}_K$.

• It is easy to see that when we have n fluents, we will have 2^{2^n+n} c-states and 2^{2^n+n-1} grounded c-states.

• Now suppose out of the n fluents, in the initial situation we do not know the truth value of p ($p \leq n$) fluents. I.e., we know the truth value of $n-p$ fluents. Then in all initial c-states $\langle s, \Sigma \rangle$, the size of Σ will be less than 2^p. It follows from the definition of the transition function and the fact that we do not have any knowledge loosing actions that any c-state that can be reached by executing a sequence of actions in the initial c-state will also have the size of its Σ less than 2^p. Taking this into account the size of the reachable (from the initial c-states) state space will be:

$$\binom{2^n}{1} + 2 \times \binom{2^n}{2} + \ldots + 2^p \times \binom{2^n}{2^p}$$

which is larger than 2^{2^p}.

• If we consider the formulations in [Moo85, SL93] the 'states' will be Kripke models. In that case for n fluents, we will have at least 2^n different possible worlds, and the accessibility relation will be a subset of $2^n \times 2^n = 2^{2n}$ elements. Thus the total number of different Kripke models will be $2^n \times 2^{2^{2n}} = 2^{2^{2n}+n}$.

• Recently complexity results about planning in presence of incomplete information have been developed in [BKT99]. One of the results is that the polynomial plan existence problem is Σ_2P complete in presence of incomplete knowledge about the initial situation and the restriction that sensing actions are executed a limited number (bounded by a constant) of times, when looking for feasible (polynomial length) plans. Without the restrictions the complexity is higher.

The tremendously large size of the state space for $\mathcal{A}_K$ and also for the formulations in [SL93, LTM97], and the above mentioned complexity results necessitates search for (provably sound) approximations that have a more manageable state space and a lower complexity. This is our focus in the next section.

3 Approximating $\mathcal{A}_K$

In this section we define several approximations of the semantics of $\mathcal{A}_K$. In our approximations we will use 3-valued states, which we will call *a-states* (or approximate states), to represent the state of knowledge of an agent. An a-state will be normally represented by a pair $\langle T, F \rangle$, where T and F are disjoint sets of fluents. Intuitively, T (resp. F) is the set of fluents which are true (resp. false) in the state $\langle T, F \rangle$. An a-state $\langle T, F \rangle$ is said to be *complete* if $T \cup F$ is the set of all the fluents in the domain description. Often we will abuse notation to represent a complete a-state $\langle T, F \rangle$, by just T. Let $\sigma_1 = \langle T_1, F_1 \rangle$ and $\sigma_2 = \langle T_2, F_2 \rangle$ be two a-states. We say that an a-state $\langle T_1, F_1 \rangle$ extends the a-state $\langle T_2, F_2 \rangle$, denoted by $\sigma_2 \preceq \sigma_1$, if $T_2 \subseteq T_1$ and $F_2 \subseteq F_1$. If σ_1 extends σ_2, we also say that σ_1 is an extension of σ_2. $\sigma_1 \cap \sigma_2$ will denote the pair $\langle T_1 \cap T_2, F_1 \cap F_2 \rangle$ and $\sigma_1 \setminus \sigma_2$ denotes the set $(T_1 \setminus T_2) \cup (F_1 \setminus F_2)$. For a set of fluents X we write $X \setminus \langle T, F \rangle$ to denote $X \setminus (T \cup F)$.

Given a fluent f and an a-state $\sigma = \langle T, F \rangle$, we say that f is *true* (resp. *false*) in σ if $f \in T$ (resp. $f \in F$); and f is *known* (resp. *unknown*) in σ if $f \in T \cup F$ (resp. $f \not\in T \cup F$). A positive (resp. negative) fluent literal f is said to hold in $\langle T, F \rangle$ if $f \in T$ (resp. $\overline{f} \in F$).

We are now ready to define several approximations for $\mathcal{A}_K$. The difference between the approximations is based on how much case analysis is done to reason about actions when the agent has incomplete knowledge about the world. We start with the 0-Approximation where no case analysis is done.

3.1 0-Approximation

Let D be a domain description, $\langle T, F\rangle$ be an a-state, and f be a fluent in D. f (resp. $\neg f$) is said to *possibly hold* in $\langle T, F\rangle$ if $f \notin F$ (resp. $f \notin T$). A set of fluents $\{f_1, \dots, f_n\}$ is said to *possibly hold* in $\langle T, F\rangle$ if for all i, f_i possibly holds in $\langle T, F\rangle$. An action a is said to be *0-executable* in an a-state $\langle T, F\rangle$ if there exists an ex-proposition **executable** a **if** $p_1, \dots, p_n$, such that $p_1, \dots, p_n$ hold in $\langle T, F\rangle$. We now introduce several notations.

- $e_a^+(\langle T, F\rangle) = \{f \mid f$ is a fluent and there exists "a **causes** f **if** $p_1, \dots, p_n$" in D such that $p_1, \dots, p_n$ hold in $\langle T, F\rangle\}$.
- $e_a^-(\langle T, F\rangle) = \{f \mid f$ is a fluent and there exists "a **causes** $\neg f$ **if** $p_1, \dots, p_n$" in D such that $p_1, \dots, p_n$ hold in $\langle T, F\rangle\}$.
- $F_a^+(\langle T, F\rangle) = \{f \mid f$ is a fluent and there exists "a **causes** f **if** $p_1, \dots, p_n$" in D such that $p_1, \dots, p_n$ possibly hold in $\langle T, F\rangle\}$.
- $F_a^-(\langle T, F\rangle) = \{f \mid f$ is a fluent and there exists "a **causes** $\neg f$ **if** $p_1, \dots, p_n$" in D such that $p_1, \dots, p_n$ possibly hold in $\langle T, F\rangle\}$.
- $K(a, \langle T, F\rangle) = \{f \mid f$ is a fluent and "a **determines** f" in $D\}$.

Intuitively, $e_a^+(\langle T, F\rangle)$ (resp. $e_a^-(\langle T, F\rangle)$) is the set of fluents that *must be true* (resp. $false$) after executing a in $\langle T, F\rangle$; $F_a^+(\langle T, F\rangle)$ (resp. $F_a^-(\langle T, F\rangle)$) is the set of fluents that *may be true* (resp. $false$) after executing a in $\langle T, F\rangle$; and $K(a, \langle T, F\rangle)$ is the set of fluents which become known after executing the action a in $\langle T, F\rangle$.

We define the result function of D in the 0-Approximation, denoted by Res_0, as follows.

$$Res_0(a, \langle T, F\rangle) = \langle T \cup e_a^+(\langle T, F\rangle) \setminus F_a^-(\langle T, F\rangle), F \cup e_a^-(\langle T, F\rangle) \setminus F_a^+(\langle T, F\rangle)\rangle.$$

We illustrate these definitions in the next example.

Example 4 For the domain description D_1 from Example 1, the initial a-state is
$\sigma_0 = \langle \emptyset, \{disarmed, exploded\}\rangle$.

Since neither $locked$ nor $\neg locked$ holds in σ_0, we have that

$$e^+_{disarm}(\sigma_0) = \emptyset \qquad e^-_{disarm}(\sigma_0) = \emptyset$$

$$e^+_{turn}(\sigma_0) = \emptyset \qquad e^-_{turn}(\sigma_0) = \emptyset$$

Since $locked$ and $\neg locked$ possibly hold in σ_0, we have that

$$F^+_{disarm}(\sigma_0) = \{exploded, disarmed\} \qquad F^-_{disarm}(\sigma_0) = \emptyset$$

$$F^+_{turn}(\sigma_0) = \{locked\} \qquad F^-_{turn}(\sigma_0) = \{locked\} \qquad \text{and}$$

$$K(look, \sigma_0) = \{locked\}$$

Since there is no ef-proposition whose action is *look*, we have that $e^+_{look}(\sigma_0) = e^-_{look}(\sigma_0) = F^+_{look}(\sigma_0) = F^-_{look}(\sigma_0) = \emptyset$. Hence,

$$Res_0(disarm, \sigma_0) = \langle \emptyset, \emptyset \rangle$$

$$Res_0(turn, \sigma_0) = \langle \emptyset, \{disarmed, exploded\} \rangle$$

$$Res_0(look, \sigma_0) = \langle \emptyset, \{disarmed, exploded\} \rangle$$

$\Box$

In the above example, even though *disarmed* and *exploded* were *false* in σ_0, after executing *disarm* they become unknown. On the face of it this is counter to the intuition behind the frame problem, where the values of fluents remain unchanged from one situation to another, unless the action in between changes them. In this case the action *disarm* has two effect propositions, neither of which is applicable as their preconditions ($\neg$*locked* and *locked* respectively) do not hold. So a naive application of the frame axiom would lead us to conclude that *disarmed* and *exploded* remain *false* in the situation after executing *disarm* in σ_0. But such a conclusion is *not sound*, as it is possible that in the real world *locked* was *true* and thus after executing *disarm*, *disarmed* became *true*. Based on this possibility, we can not just have *disarmed* to be true in the resultant situation either, as this would be unsound if $\neg$*locked* was true in the real world instead. Thus taking into account the two possibilities, we can reason that the agent will not know whether *disarmed* is *true* or *false* after executing *disarm*. Thus, the resultant a-state should have *disarmed* as unknown. *Our not so straightforward definition of* Res_0*, encodes this skeptical reasoning.* We now use Res_0 to define the transition function Φ_0. Again, executing an action might result in an undefined a-state, denoted by $\perp$.

Definition 7 Given a domain description D, the 0-transition function Φ_0 of D is defined as follows:

- If a is not 0-executable in σ, then $\Phi_0(a, \sigma) = \{\perp\}$;
- If a is 0-executable in σ and a is a non-sensing action then $\Phi_0(a, \sigma) = \{Res_0(a, \sigma)\}$; and
- If a is 0-executable in σ and a is a sensing action then $\Phi_0(a, \sigma) = \{\sigma' \mid \sigma \preceq \sigma'$ and $K(a, \sigma) \setminus \sigma = \sigma' \setminus \sigma\}$. $\Box$

In the above definition, the transition due to a sensing action results in a set of a-states, each corresponding to a particular set of sensing results. The condition that all elements of $\sigma' \setminus \sigma$ are from $K(a, \sigma)$ makes sure that only fluents that are sensed are the ones for which we have a k-proposition and the condition that all elements of $K(a, \sigma)$ are in $\sigma' \setminus \sigma$ makes sure that all fluents mentioned in the k-propositions for that action have a *true* or *false* value in σ'. If we were to allow actions to be able to both sense and change the world, then $\Phi_0(a, \sigma)$ for such an action can be succinctly defined as: $\Phi_0(a, \sigma) = \{\sigma' \mid \sigma'$ extends $Res_0(a, \sigma)$ and $\sigma' \setminus Res_0(a, \sigma) = K(a, \sigma) \setminus Res_0(a, \sigma)\}$.

Let Φ_0 be a 0-transition function of D. The 0-extended transition function $\hat{\Phi}_0$ which maps pairs of conditional plans and a-states into set of a-states is defined next.

Definition 8

1. $\hat{\Phi}_0([], \sigma) = \{\sigma\}$;
2. $\hat{\Phi}_0(a, \sigma) = \Phi_0(a, \sigma)$;

3. For a case plan

$$
\begin{array}{l}
c = \text{Case} \\
\quad \varphi_1 \rightarrow p_1 \\
\quad \vdots \\
\quad \varphi_n \rightarrow p_n \\
\text{Endcase}
\end{array}
$$

$$\hat{\Phi}_0(c,\sigma) = \begin{cases} \hat{\Phi}_0(p_j,\sigma) \text{ if } \varphi_j \text{ holds in } \sigma, \\ \{\bot\} \quad \text{if none of } \varphi_1,\ldots,\varphi_n \text{ holds in } \sigma; \end{cases}$$

4. For two conditional plans c_1 and c_2, $\hat{\Phi}_0([c_1;c_2],\sigma) = \bigcup_{\sigma' \in \hat{\Phi}_0(c_1,\sigma)} \hat{\Phi}_0(c_2,\sigma')$;

5. $\hat{\Phi}_0(c,\bot) = \{\bot\}$ □

A conditional plan c is *0-executable* in an a-state σ if $\bot \notin \hat{\Phi}_0(c,\sigma)$. An a-state σ_0 is called an *initial a-state* of D if for any fluent literal f, f holds in σ_0 iff " **initially** f" is in D. It is easy to see that for each domain description, the initial a-state is unique.

Definition 9 Given a domain description D, a 0-model is a pair (σ_0, Φ_0) where σ_0 is the initial a-state of D and Φ_0 is a 0-transition function of D. □

Similarly to Proposition 1, we can prove that the 0-transition function Φ_0 of D is unique. In the next definition, we define our first approximate entailment relation, the 0-entailment ($\models_0$), based on the 0-model.

Definition 10 Let D be a domain description, φ be a fluent formula, and c be a conditional plan in D. We say

- $D \models_0$ **Knows** φ **after** c if c is 0-executable in σ_0 and φ holds in every a-state belonging to $\hat{\Phi}_0(c,\sigma_0)$ for every 0-model (σ_0,Φ_0) of D; and
- $D \models_0$ **Kwhether** φ **after** c if c is 0-executable σ_0 and φ is known in every a-state belonging to $\hat{\Phi}_0(c,\sigma_0)$ for every 0-model (σ_0,Φ_0) of D. □

Example 5 For the domain description D_1 we have that

$$
\begin{array}{lcl}
\Phi_0(disarm,\sigma_0) & = & \{\langle \emptyset, \emptyset \}\rangle\} \\[1em]
\Phi_0(turn,\sigma_0) & = & \{\langle \emptyset, \{disarmed, exploded\}\rangle\} \\[1em]
\Phi_0(look,\sigma_0) & = & \{\langle \{locked\}, \{disarmed, exploded\}\rangle, \\
 & & \langle \emptyset, \{locked, disarmed, exploded\}\rangle\}
\end{array}
$$

Thus $D_1 \models_0$ **Kwhether** *locked* **after** *look* but
$D_1 \not\models_0$ **Knows** *locked* **after** *look* and $D_1 \not\models_0$ **Knows** $\neg$*locked* **after** *look*. □

In the next example we show that the conditional plan for disarming the bomb in Example 3 can also be analyzed using the 0-Approximation.

Example 6 Let us reconsider the domain D_1 and the conditional plan of Example 3.

$$\left.\begin{array}{l} \mathit{look}; \\ \quad \left.\begin{array}{l} \text{Case} \\ \neg \mathit{locked} \rightarrow \mathit{turn} \\ \quad \mathit{locked} \rightarrow [\,] \\ \text{Endcase} \end{array}\right\} = c_1 \\ \mathit{disarm} \end{array}\right\} = c$$

We have that the initial a-state of D is $\sigma_0 = \langle \emptyset, \{\mathit{disarmed}, \mathit{exploded}\}\rangle$.

To prove that $D_1 \models_0 \textbf{Knows}\ \mathit{disarmed} \wedge \neg \mathit{exploded}\ \textbf{after}\ c$, we compute $\hat{\Phi}_0(c, \sigma_0)$ as follows.

First, since $K(\mathit{look}, \sigma_0) = \{\mathit{locked}\}$ we have that $\Phi_0(\mathit{look}, \sigma_0) = \{\sigma_1, \sigma_2\}$ where

$\sigma_1 = \langle \{\mathit{locked}\}, \{\mathit{disarmed}, \mathit{exploded}\}\rangle$ and $\sigma_2 = \langle \emptyset, \{\mathit{disarmed}, \mathit{exploded}, \mathit{locked}\}\rangle$.

Hence,

$$\begin{array}{rcl} \hat{\Phi}_0(c, \sigma_0) & = & \bigcup_{\sigma' \in \Phi_0(\mathit{look}, \sigma_0)} \hat{\Phi}_0(c_1; \mathit{disarm}, \sigma') \\ & = & \hat{\Phi}_0(c_1; \mathit{disarm}, \sigma_1) \cup \hat{\Phi}_0(c_1; \mathit{disarm}, \sigma_2) \end{array}$$

Since *locked* holds in σ_1 and $\neg \mathit{locked}$ holds in σ_2, we have that

$\hat{\Phi}_0(c_1; \mathit{disarm}, \sigma_1) = \hat{\Phi}_0(\mathit{disarm}, \sigma_1)$ and $\hat{\Phi}_0(c_1; \mathit{disarm}, \sigma_2) = \bigcup_{\sigma' \in \hat{\Phi}_0(\mathit{turn}, \sigma_2)} \hat{\Phi}_0(\mathit{disarm}, \sigma')$.

Furthermore, $\hat{\Phi}_0(\mathit{disarm}, \sigma_1) = \{\langle \{\mathit{disarmed}, \mathit{locked}\}, \{\mathit{exploded}\}\rangle\}$ and

$\hat{\Phi}_0(\mathit{turn}, \sigma_2) = \Phi_0(\mathit{turn}, \sigma_2) = \{\langle \{\mathit{locked}\}, \{\mathit{disarmed}, \mathit{exploded}\}\rangle\} = \{\sigma_1\}$.

Thus, $\hat{\Phi}_0(c_1; \mathit{disarm}, \sigma_2) = \hat{\Phi}_0(\mathit{disarm}, \sigma_1) = \{\langle \{\mathit{disarmed}, \mathit{locked}\}, \{\mathit{exploded}\}\rangle\}$.

In summary, we have that $\hat{\Phi}_0(c, \sigma_0) = \{\langle \{\mathit{disarmed}, \mathit{locked}\}, \{\mathit{exploded}\}\rangle\}$ which implies that $D_1 \models_0 \textbf{Knows}\ \mathit{disarmed} \wedge \neg \mathit{exploded}\ \textbf{after}\ c$. □

Although 0-Approximation can correctly analyze the above example, it has weaknesses and it can not entail many queries entailed by the $\mathcal{A}_K$ semantics. The following example illustrates this.

Example 7 Let us consider the domain D_3 with the following causal rules;

$$\left.\begin{array}{l} a\ \textbf{causes}\ f\ \textbf{if}\ g \\ a\ \textbf{causes}\ f\ \textbf{if}\ \neg g \\ \textbf{executable}\ a \end{array}\right\} = D_3$$

The initial a-state of D_3 is $\sigma_0 = \langle \emptyset, \emptyset \rangle$. Intuitively, we would expect that $\textbf{Knows}\ f\ \textbf{after}\ a$ is entailed by D_3 and this entailment holds for $\models^r_{\mathcal{A}_K}$. However, $\hat{\Phi}_0(a, \sigma_0) = \Phi_0(a, \sigma_0) = \{\langle \emptyset, \emptyset \rangle\}$ because $e^+_a(\sigma_0) = e^-_a(\sigma_0) = F^-_a(\sigma_0) = \emptyset$ and $F^+_a(\sigma_0) = \{f\}$. This means that $D_3 \not\models_0 \textbf{Knows}\ f\ \textbf{after}\ a$. □

In the above example, by doing case analysis we can intuitively conclude that f should be true after executing a in the initial situation. I.e., we analyze that in the initial situation g could be either true or false, and in both cases we can conclude that f will be true after executing a. The reasoning mechanism in the 0-Approximation lacks any such case analysis. In the next section we introduce the notion of 1-Approximation that does some case analysis and is able to make the intuitive conclusion in the above example.

3.2 1-Approximation

The 1-Approximation improves on 0-Approximation by defining a new result function which given an incomplete a-state σ and an action a, considers all complete extensions of σ, and applies a to these extensions and then considers what is true and what is false in all the resulting states. Such a transition function does intuitive reasoning w.r.t. the Example 7. We now formally define the new result function. For an a-state σ, let $Comp(\sigma)$ be the set of all the complete a-states that extend σ. The result function, Res_1, which maps a pair of an action a and an a-state σ into an a-state $Res_1(a, \sigma)$ is defined as follows.

$$Res_1(a, \sigma) = \bigcap_{\sigma' \in Comp(\sigma)} Res_0(a, \sigma').$$

The notion of executability changes slightly. Now, an action a is said to be *1-executable* in an a-state σ if it is 0-executable in all a-states in $Comp(\sigma)$. The 1-transition function is defined next.

Definition 11 Given a domain description D, the 1-transition function Φ_1 of D is defined as follows:

- If a is not 1-executable in σ then $\Phi_1(a, \sigma) = \{\bot\}$;
- If a is 1-executable in σ and a is a non-sensing action then $\Phi_1(a, \sigma) = \{Res_1(a, \sigma)\}$; and
- If a is 1-executable in σ and a is a sensing action then $\Phi_1(a, \sigma) = \{\sigma' \mid \sigma \preceq \sigma' \text{ and } K(a, \sigma) \setminus \sigma = \sigma' \setminus \sigma\}$.

□

A 1-model of D is then defined as a pair (σ_0, Φ_1) where σ_0 is the initial a-state of D and Φ_1 is the 1-transition function of D. The notion of 1-extended function and 1-entailment is then defined as in Definitions 8 and 10 using 1-transition function and 1-model, respectively.

In the next example we shows that the 1-Approximation allows us to reason by cases.

Example 8 Let us consider again the domain D_3 from Example 7. The initial a-state of D_3 is $\sigma_0 = \langle \emptyset, \emptyset \rangle$. The set of complete extensions of σ_0, $Comp(\sigma_0)$, is the set of all complete a-states of D_3. More precisely, $Comp(\sigma_0) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ where $\sigma_1 = \langle \{f, g\}, \emptyset \rangle$, $\sigma_2 = \langle \{f\}, \{g\} \rangle$, $\sigma_3 = \langle \emptyset, \{f, g\} \rangle$, and $\sigma_4 = \langle \{g\}, \{f\} \rangle$.

Since $Res_0(a, \sigma_1) = \langle \{f, g\}, \emptyset \rangle$, $Res_0(a, \sigma_2) = \langle \{f\}, \{g\} \rangle$, $Res_0(a, \sigma_3) = \langle \{f\}, \{g\} \rangle$, and $Res_0(a, \sigma_4) = \langle \{f, g\}, \emptyset \rangle$ we have that $Res_1(a, \sigma_0) = \langle \{f\}, \emptyset \rangle$. Thus, for any 1-model (σ_0, Φ_1) of D_3, $\Phi_1(a, \sigma_0) = \{\langle \{f\}, \emptyset \rangle\}$. Hence, we can conclude that $D_3 \models_1$ **Knows** f **after** a. □

We now state the relation between the 0-Approximation and the 1-Approximation of domain descriptions of $\mathcal{A}_K$.

Proposition 4 (Soundness of $\models_0$ w.r.t. $\models_1$) Let D be a domain description, φ be a fluent formula of D, and c be a conditional plan. Then,

if $D \models_0$ **Knows** φ **after** c then $D \models_1$ **Knows** φ **after** c.

Proof. *(Sketch)* Similar to Proposition 1 we can prove that for every domain description D[4], the 0-model and 1-model of D are uniquely determined. Furthermore, the initial a-state in the 0-Approximation is also the initial a-state in the 1-Approximation. Let us denote the 0-model and 1-model of D by (σ_0, Φ_0) and (σ_0, Φ_1) respectively. Let σ and δ be two a-states of D such that $\sigma \preceq \delta$. Then, for every action a of D, we can prove that

(i) for each $\sigma' \in \Phi_0(a, \sigma)$ there exists a $\delta' \in \Phi_1(a, \delta)$ such that $\sigma' \preceq \delta'$;

(ii) for each $\delta' \in \Phi_1(a, \delta)$ there exists a $\sigma' \in \Phi_0(a, \sigma)$ such that $\sigma' \preceq \delta'$.

Using (i) and (ii) we can then prove that for any conditional plan c such that $\bot \not\in \hat{\Phi}_0(c, \sigma)$,

(iii) $\bot \not\in \hat{\Phi}_1(c, \delta)$;

(iv) for each $\sigma' \in \hat{\Phi}_0(c, \sigma)$ there exists a $\delta' \in \hat{\Phi}_1(c, \delta)$ such that $\sigma' \preceq \delta'$; and

(v) for each $\delta' \in \hat{\Phi}_1(c, \delta)$ there exists a $\sigma' \in \hat{\Phi}_0(c, \sigma)$ such that $\sigma' \preceq \delta'$.

(iii) proves that if c is 0-executable in σ_0 then c is 1-executable in σ_0. This, together with (iv) and (v), and the fact that $\sigma_0 \preceq \sigma_0$, proves the proposition. □

The next example shows that the 1-Approximation is also not able to make some intuitive conclusions[5] that can be made using the $\mathcal{A}_K$ semantics.

Example 9 Consider the domain description:

$$\left.\begin{array}{l} a \textbf{ causes } p \textbf{ if } r \\ a \textbf{ causes } q \textbf{ if } \neg r \\ b \textbf{ causes } f \textbf{ if } p \\ b \textbf{ causes } f \textbf{ if } q \\ \textbf{executable } a \\ \textbf{executable } b \end{array}\right\} = D_4$$

The initial a-state is $\langle \emptyset, \emptyset \rangle$, where p, q, r, and f are unknown. Although intuitively and also according to the rational semantics of $\mathcal{A}_K$, after executing a followed by b in the initial a-state, f should be true, our 1-Approximation is not able to capture this. This is because the 1-Approximation reasons by cases only up to 1-level. Since after reasoning by cases for 1-level, it summarizes its reasoning to a pair $\langle T, F \rangle$, it is not able to capture the fact that after executing a in the initial a-state $p \vee q$ is true. □

To overcome the limitation of 1-Approximation as illustrated by the above example, we can define 2-Approximation which will reason by cases up to 2 levels. But it will break down when reasoning by cases up to 3 levels is necessary, and so on. In the next section, we define ω-Approximation which allows reasoning by cases for multiple levels without setting a limit on the number of levels.

---

[4] Recall that we do not allow contradictory v-propositions or contradictory ef-propositions in D

[5] We thank the anonymous AAAI97 reviewer who pointed this out.

3.3 ω-Approximation

Our intention in ω-Approximation is to reason by cases to as many levels as possible. But this number is limited by the structure of the plan. We can only reason by cases through sequences of non-sensing actions. For that reason, given a sequence of actions $\alpha = a_1; \ldots; a_n$, we now define the longest prefix of α consisting of only non-sensing actions or a single sensing action, denoted by $pre(\alpha)$, as follows:

- if a_1 is a sensing action then $pre(\alpha) = a_1$; or
- if α does not contain a sensing action then $pre(\alpha) = \alpha$; or
- if a_j is the first sensing action in α, $1 < j \leq n$, then $pre(\alpha) = a_1, \ldots, a_{j-1}$.

The sequence of actions obtained from α after removing its prefix $pre(\alpha)$ is called the *remainder* of α and is denoted by $rem(\alpha)$.

Given a sequence of non-sensing actions $\alpha = a_1, \ldots, a_n$, we now define $Res_\omega(\alpha, \sigma)$ by considering all complete extensions of σ, applying α to each of them and then determining their intersection. This corresponds to doing case by case reasoning for n-levels. More formally,

$$Res_\omega(\alpha,\sigma) = \bigcap_{\sigma' \in Comp(\sigma)} Res_0(a_n, Res_0(a_{n-1}, \ldots, Res_0(a_1, \sigma'))).$$

An action a is *ω-executable* in σ if a is 0-executable in all complete extensions of σ. And, a sequence of non-sensing actions α is ω-executable in σ if α is *0-executable* in all complete extensions of σ.

The ω-Approximation of D is defined by a function Φ_ω, called ω-transition function, which maps a pair of a sequence of actions α and an a-state σ into a set of a-states, denoted by $\Phi_\omega(\alpha, \sigma)$, as follows.

$$\Phi_\omega(\alpha,\sigma) = \begin{cases} \{\bot\} \text{ if } pre(\alpha) \text{ is not } \omega\text{-executable in } \sigma; \\ \\ \{Res_\omega(\alpha,\sigma)\} \text{ if } \alpha \text{ does not contain a sensing action and is } \omega\text{-executable in } \sigma; \\ \\ \{\sigma' \mid \sigma' \text{ extends } \sigma, \text{ and } \sigma' \setminus \sigma = K(a,\sigma) \setminus \sigma\} \\ \qquad\qquad \text{if } \alpha = a, \ a \text{ is a sensing action, and } a \text{ is } \omega\text{-executable in } \sigma; \text{ and} \\ \\ \bigcup_{\sigma' \in \Phi_\omega(pre(\alpha),\sigma)} \Phi_\omega(rem(\alpha), \sigma'), \quad \text{otherwise.} \end{cases}$$

A sequence of actions α is ω-executable in σ if $\bot \not\in \Phi_\omega(\alpha, \sigma)$.

An ω-model for a domain description D is then defined as the pair (σ_0, Φ_ω), where σ_0 is the initial a-state of D and Φ_ω is an ω-transition function of D.

To extend the function Φ_ω over pairs of conditional plans and a-states we need the following observation.

Observation 3.1 Every conditional plan c can be represented as a sequence of conditional plans $c_1; \ldots; c_n$ where

(a) c_i is either a sequence of actions or a case plan; and

(b) for every $i < n$, if c_i is a sequence of actions then c_{i+1} is a case plan. □

From now on, we will often write a conditional plan c as a sequence $c = c_1; \ldots; c_n$ where c_i's satisfy the conditions (a) and (b) of Observation 3.1.

The extended transition function of Φ_ω, denoted by $\hat{\Phi}_\omega$, is defined next.

For a conditional plan c and an a-state σ, we define

1. For $c = c_1$, where c_1 is a sequence of actions, $\hat{\Phi}_\omega(c, \sigma) = \Phi_\omega(c_1, \sigma)$;
2. For $c =$ Case
   $\varphi_1 \rightarrow c_1$
   ...
   $\varphi_l \rightarrow c_l$
   Endcase,
   $$\hat{\Phi}_\omega(c,\sigma) = \begin{cases} \hat{\Phi}_\omega(c_i,\sigma) \text{ if } \varphi_i \text{ holds in } \sigma, \\ \\ \{\bot\} \quad \text{if none of } \varphi_1, \ldots, \varphi_l \text{ holds in } \sigma; \end{cases}$$
3. For $c = c_1; c_2; \ldots; c_n$, $n > 1$,
   (a) if c_1 is a sequence of actions,
   $$\hat{\Phi}_\omega(c,\sigma) = \bigcup_{\sigma' \in \hat{\Phi}_\omega(c_1,\sigma)} \hat{\Phi}_\omega(c_2; \ldots; c_n, \sigma')$$
   (b) if $c_1 =$ Case
   $\varphi_1 \rightarrow p_1$
   ...
   $\varphi_m \rightarrow p_m$
   Endcase,
   $$\hat{\Phi}_\omega(c,\sigma) = \begin{cases} \bigcup_{\sigma' \in \hat{\Phi}_\omega(p_i,\sigma)} \hat{\Phi}_\omega(c_2; \ldots; c_n, \sigma') \text{ if } \varphi_i \text{ holds in } \sigma, \\ \\ \{\bot\} \quad \text{if none of } \varphi_1, \ldots, \varphi_m \text{ holds in } \sigma; \end{cases}$$
4. $\hat{\Phi}_\omega(c, \bot) = \{\bot\}$ for every conditional plan c.

The notion of ω-entailment is then defined as in Definition 10 using the ω-model.

The next example shows that this generalization indeed overcomes the problem of 1-Approximation in Example 9, through reasoning by cases for multiple levels.

Example 10 Let us consider the domain description D_4 from Example 9. Let σ be a complete extension of σ_0. Since σ is complete, either r or $\neg r$ holds in σ. Thus, either p or q holds in $Res_0(a, \sigma)$. This implies that $e_b^+(Res_0(a,\sigma)) = \{f\}$. Since D_4 does not contain an ef-proposition, whose effect is $\neg f$, we have that $F_b^-(Res_0(a,\sigma)) = \emptyset$. Hence, f holds in $Res_0(b, Res_0(a,\sigma))$ for every complete a-state σ. Thus f holds in $Res_\omega([a;b], \sigma_0)$. By definition of Φ_ω, we have that f holds in $\Phi_\omega([a;b], \sigma_0)$ where σ_0 is the initial a-state of D_4. Since $a;b$ is a sequence of actions, $\hat{\Phi}_\omega([a;b], \sigma_0) = \Phi_\omega([a;b], \sigma_0)$. Thus, $D_4 \models_\omega$ **Knows** f **after** $[a;b]$. □

We prove the soundness of $\models_1$ w.r.t. $\models_\omega$ in the next proposition.

Proposition 5 (Soundness of $\models_1$ w.r.t. $\models_\omega$) Let D be a domain description, φ be a fluent formula, and c be a conditional plan. Then,

$$\text{if } D \models_1 \textbf{Knows } \varphi \textbf{ after } c \text{ then } D \models_\omega \textbf{Knows } \varphi \textbf{ after } c.$$

Proof. The proof is similar to the proof of Proposition 4. □

3.4 Soundness of 0, 1 and ω-Approximations w.r.t. $\mathcal{A}_K$-semantics

In the previous sub-sections we discussed three different approximations of $\mathcal{A}_K$. Our next goal is to show that these approximations are sound w.r.t. $\mathcal{A}_K$. Since we have already shown in Propositions 4 and 5 that $\models_0$ is sound w.r.t. $\models_1$ and $\models_1$ is sound w.r.t. $\models_\omega$, we will now show that the ω-Approximation is sound w.r.t. $\mathcal{A}_K$.

Proposition 6 (Soundness of $\models_\omega$ w.r.t. $\models_{\mathcal{A}_K}$) Let D be a domain description, φ be a fluent formula, and c be a conditional plan. Then,

$$\text{if } D \models_\omega \textbf{Knows } \varphi \textbf{ after } c \text{ then } D \models_{\mathcal{A}_K} \textbf{Knows } \varphi \textbf{ after } c.$$

Proof. In Appendix C. □

Even though ω-Approximation can reason more than the 1-Approximation, it still can not match the $\mathcal{A}_K$ semantics. The following example illustrates this.

Example 11 Let D_5 be the following domain description.

$$\left.\begin{array}{l}
a \textbf{ causes } \neg p \textbf{ if } r \\
b \textbf{ determines } r \\
c \textbf{ causes } p \textbf{ if } r \\
\textbf{initially } p \\
\textbf{executable } a \\
\textbf{executable } b \\
\textbf{executable } c
\end{array}\right\} = D_5$$

We have that $\sigma_0 = \langle \{p\}, \emptyset \rangle$ is the initial a-state of D_5.

Let $\alpha = [a; b; c]$.

There are two complete extensions of σ_0: $\sigma_1 = \langle \{p\}, \{r\} \rangle$ and $\sigma_2 = \langle \{p, r\}, \emptyset \rangle$. This implies that $Res_\omega(a, \sigma_0) = Res_0(a, \sigma_1) \cap Res_0(a, \sigma_2) = \langle \{p\}, \{r\} \rangle \cap \langle \{r\}, \{p\} \rangle = \langle \emptyset, \emptyset \rangle$. Furthermore, $\Phi_\omega(b, \langle \emptyset, \emptyset \rangle) = \{\langle \{r\}, \emptyset \rangle, \langle \emptyset, \{r\} \rangle\}$, and $\Phi_\omega(c, \langle \{r\}, \emptyset \rangle) = \{\langle \{p, r\}, \emptyset \rangle\}$ and $\Phi_\omega(c, \langle \emptyset, \{r\} \rangle) = \{\langle \emptyset, \{r\} \rangle\}$.

Since

$$\begin{aligned}
\hat{\Phi}_\omega(\alpha, \sigma_0) = \Phi_\omega(\alpha, \sigma_0) &= \textstyle\bigcup_{\sigma \in \Phi_\omega(a, \sigma_0)} \Phi_\omega([b; c], \sigma) = \Phi_\omega([b; c], \langle \emptyset, \emptyset \rangle) \\
&= \textstyle\bigcup_{\sigma \in \Phi_\omega(b, \langle \emptyset, \emptyset \rangle)} \Phi_\omega(c, \sigma) = \Phi_\omega(c, \langle \{r\}, \emptyset \rangle) \cup \Phi_\omega(c, \langle \emptyset, \{r\} \rangle) \\
&= \{\langle \{p, r\}, \emptyset \rangle, \langle \emptyset, \{r\} \rangle\},
\end{aligned}$$

we have that $D_5 \not\models_\omega$ **Knows** p **after** $[a; b; c]$ and $D_5 \not\models_\omega$ **Knows** $\neg p$ **after** $[a; b; c]$.

Now, we will show that $D_5 \models^r_{\mathcal{A}_K}$ **Knows** p **after** $[a;b;c]$. Let $s_1 = \{p\}$, $s_2 = \{p,r\}$, and $s_3 = \{r\}$. D_5 has two initial c-states: $\langle s_1, \{s_1, s_2\}\rangle$ and $\langle s_2, \{s_1, s_2\}\rangle$. We have that

$\hat{\Phi}([a;b;c], \langle s_1, \{s_1, s_2\}\rangle) = \hat{\Phi}([b;c], \langle s_1, \{s_1, s_3\}\rangle) = \hat{\Phi}(c, \langle s_1, \{s_1\}\rangle) = \langle s_1, \{s_1\}\rangle$ and

$\hat{\Phi}([a;b;c], \langle s_2, \{s_1, s_2\}\rangle) = \hat{\Phi}([b;c], \langle s_3, \{s_1, s_3\}\rangle) = \hat{\Phi}(c, \langle s_3, \{s_3\}\rangle) = \langle s_2, \{s_2\}\rangle$.

It is easy to check that p is known to be true in $\hat{\Phi}([a;b;c], \langle s_1, \{s_1, s_2\}\rangle)$ and $\hat{\Phi}([a;b;c], \langle s_2, \{s_1, s_2\}\rangle)$. Thus $D_5 \models^r_{\mathcal{A}_K}$ **Knows** p **after** $[a;b;c]$. □

3.5 Complexity of progression

In this sub-section we will compare the complexity of progression in the various approximations. Suppose the number of fluents we have is n, and d is the size of the domain description. Given an a-state $\langle T, F\rangle$, such that the size of $T \cup F$ is m, the complexity of computing Res, Res_0, Res_1, and Res_ω in the different approximations are as follows:

- 0-Approximation: The complexity of computing $Res_0(a, \sigma)$ is $m\times$ number of ef-propositions in the domain description. This is of the order of $m \times d$.
- 1-Approximation: Here we need to compute Res_1. This is of the order of $2^{n-m} \times m \times d$.
- ω-Approximation: Here we also need to compute Res_ω. This is also of the order of $2^{n-m} \times m \times d$.

It is easy to see that if a sensing action a determines p fluents and σ is a-state where none of these fluents are known, then $\Phi_0(a, \sigma)$, $\Phi_1(a, \sigma)$, $\Phi_\omega(a, \sigma)$ will have 2^p a-states.

From the above analysis, it is clear that progression can be done much faster in the 0-Approximation than in the other two. On the other hand there is no significant difference in doing progression between 1-Approximation and ω-Approximation. (A more formal result was recently given in [BKT99], where it was shown that while computing the next state $Res_0(a, \sigma)$ is a polynomial time procedure, computing $Res_1(a, \sigma)$ is coNP-Complete.)

4 Related research

In this section we first discuss the expressiveness and limitations of our formulations in this paper as compared to other formulations in the literature and then do detailed comparisons with works that are closest to ours.

4.1 Expressiveness and limitations of $\mathcal{A}_K$

Since our main goal in this paper has been to formalize sensing actions, to avoid distractions we have on purpose limited the expressiveness of the rest of the language. For example, we do not allow multi-valued fluents [GKL97], static causal laws [Lin95, MT95, Bar95], concurrent actions [LS92, BG93, BG97], narratives [BGP98, MS94], etc. In Section 2.2 we briefly discuss how most of these restrictions can be lifted. Besides these, we also make some additional assumptions that limit the expressiveness of our language. We now briefly discuss these assumptions and why we make them.

- We follow the approach in [GL93] in not having a full first-order language. This allows us to avoid the additional induction axioms described in [Rei91, Rei98]. Although, we do not have full first order language we do allow variables, and propositions with variables such as:

  $move(X, Y)$ **causes** $at(Y)$

  Here, the proposition is viewed as a 'schema' representing a set of propositions where X and Y are bound. Also, we assume our domain to be finite. I.e, we assume that we have a finite set of actions, and fluents.

- We assume that there is a single agent who is planning and acting and our interest is in formalizing his knowledge about the world vis-a-vis the real state of the world. Moreover, unlike in [SL93] we make the assumptions of the modal logic S5 and hard code it into our semantics. This allows us to use the simpler c-states instead of using Kripke models. Moreover, as we show in Section 2.4, this leads to a smaller state space. A similar approach is followed in most of the chapters in [FHMV95]

- We assume the sensing actions (i.e., the operation of the sensors) to be perfect. Bacchus, Halpern, and Levesque [BHL95] extend the situation calculus approach in [SL93] to allow for noisy sensors. In the future, we plan to extend our approach to this case. Also, in the Operations Research literature POMDPs (partially observable Markov decision processes) are used in formulating noisy observations. We plan to formulate sensing actions using POMDPs and compare it with our current formulation.

- We follow the high-level language doctrine in [Lif97] and the approach in databases and use a limited query language. This allows us to have a simpler formulation. Our query language can be easily extended to allow for knowledge and temporal operators as in [GW96], but it is not straightforward and nor we favor the generality of allowing quantifiers (as in [Rei91, Rei98]).

- In most of the paper our interest is in progression and verification of conditional plans. In other words, given the description (possibly partial) of an initial state, a conditional plan and a goal, we would like to verify if the given plan executed in the initial state will take us to the goal. Because of this limited interest, we can use the simpler formulation in (1.5) and (1.6) instead of (1.3) and (1.4). When using the simpler formulation we can not add an observation of the form $\exists \mathsf{S}.Knows(f, \mathsf{S})$ to find out what S is. This is a limitation only when we use the logical formulation, and not at the semantic level.

4.2 Relationship with Scherl and Levesque's formulation

In Section 2.3 we gave a translation of domain descriptions in D to a first-order theory that used Scherl and Levesque's [SL93] successor-state axiom (which is based on Moore's [Moo85] formulation) and showed the equivalence w.r.t. queries in the language of $\mathcal{A}_K$. Since Scherl and Levesque directly formalize in first-order logic, their formulation is more general than ours; (i) in terms of allowing more general descriptions about the domain such as being able to choose which modal logic to use, and observations about non-initial situations; and (ii) in terms of allowing more general queries.

On the other hand our goal in this paper has been to have a simpler formulation, perhaps at the cost of generality. For example, the 'state' of the agent's knowledge in Scherl and Levesque's formulation (and also in Moore's formulation) would be a Kripke model. Since planning in a state space where a 'state' is a Kripke model is more difficult, we have a simpler notion of a 'state' which we call a c-state. (For instance, if we have n fluents then the number of different Kripke models are $2^{2^{2n}+n}$, while the number of different c-states are 2^{2^n+n}.) As mentioned earlier, our c-state has two components, the real state of the world and the set of possible states that the agent thinks it may be in. Our c-state is actually equivalent to a Kripke model when we consider the logic S5. Thus with a goal to make things simpler we sacrifice generality and make an a-priori decision on which logic of knowledge to use.

Also, since we develop a high level language $\mathcal{A}_K$, with an independent semantics – that does not depend on standard logics, it can serve the role of a benchmark for languages with sensing actions, at least for the restricted class of queries in $\mathcal{A}_K$. Moreover, this high level language makes it easier for us to prove the soundness of approximations that have a much less and more manageable state space. By having sound and complete translations of domain descriptions in $\mathcal{A}_K$ to theories in first-order logic that use Scherl and Levesque's axioms, our sound approximations are also in a way sound approximations of Scherl and Levesque's formalism.

Finally, we would like to mention that loop-free robot programs of [Lev96] are special cases of our conditional plans. In particular, the statements $seq(a, r)$ and $branch(a, r_1, r_2)$ of [Lev96] can be recursively translated to conditional plans $a; r$ and

```
$a$
Case
    $f \rightarrow r_1$
    $\neg f \rightarrow r_2$
Endcase
```

respectively. In this paper we do not allow loops in our conditional plans. But the ideas in [Lev96, LTM97] can be used to extend our conditional plans to allow loops.

4.3 Relationship with Lobo et al.'s semantics

Lobo et al. in [LTM97] have a goal similar to ours, in terms of developing a high-level language that allows sensing actions and giving translations of it to theories in a standard logical language. We now list some of the major differences between both approaches:

- They represent the state of an agent's knowledge by a set of states (which they refer to as a situation), and the transition function Φ in their model is defined such that for a sensing action a and a situation Σ, $\Phi(a, \Sigma)$ is a subset of Σ that consists of all states which agree on the fluent values determined by the sensing action a. A drawback of this approach is that domain descriptions have a lot of models. But more importantly, it is possible that when a domain description has two actions a and b that determine the same fluent f, there are models Φ, such that $\Phi(a, \Sigma) \neq \Phi(b, \Sigma)$ for some Σ's. In other words, while f may be true in all states in $\Phi(a, \Sigma)$, it may be false in all states in $\Phi(b, \Sigma)$. We find such models unintuitive.
- The semantics of $\mathcal{A}_K$ is more general than the semantics of Lobo et al. in the sense that in their formulation the assumption about models being rational is hard wired into the semantics.
- On the other hand the high level language used by Lobo et al. is more general than the one we are using. They allow conditional sensing through preconditions in k-propositions. We do not allow preconditions in k-propositions but we allow executability conditions.

- Lobo et al. give translations of their domain descriptions to theories in epistemic logic programs [Gel91]. We have translations to disjunctive logic programs [BS98, S00], which are simpler than epistemic logic programs. We also give translations to first-order theories.
- Finally, we consider sound approximations of our language. In the later part of this section we show our semantics to be equivalent (sometimes) to theirs. Thus our approximations are also sound approximations of their formulation.

We now give a quick overview of the formulation in [LTM97], restricted to the common syntax of $\mathcal{A}_K$ and their language. We then show that our rational semantics is equivalent to the semantics in [LTM97] for this restricted case. The semantics of [LTM97] is defined through transition functions that map pairs of actions and situations into situations where a situation is a set of states. A situation is consistent if it is not empty. Given a domain description D, the situation consisting of all the initial states of D, denoted by Σ_0, is called the *initial situation* of D. A fluent f is said to be *true* in a situation Σ if $f \in s$ for every $s \in \Sigma$. A fluent formula φ is said to be *true* in a situation Σ if φ is true in every state s belonging to Σ. We will need the following definition.

Definition 12 Let Σ be a consistent situation and f a fluent. A consistent situation Σ' is "*f-compatible*" with Σ iff

1. $\Sigma' = \{\sigma \in \Sigma \mid f \notin \sigma\}$; or
2. $\Sigma' = \{\sigma \in \Sigma \mid f \in \sigma\}$

For a domain description D, a function Φ from pairs of actions and situations into situations is called an interpretation of D.

Definition 13 An interpretation Φ of a domain description D is a *model* of D if and only if

1. for any consistent situation Σ.
   (a) for any non-sensing action a,
   $$\Phi(a, \Sigma) = \bigcup_{s \in \Sigma} \{Res(a, s)\}.$$
   (b) for each sensing action a, let
   $$\begin{array}{c} a \textbf{ determines } f_1 \\ \dots \\ a \textbf{ determines } f_n \end{array}$$
   be the k-propositions in which a occurs. Then,
   - $\Phi(a, \Sigma)$ must be a consistent situation; and
   - $\Phi(a, \Sigma) = \Sigma_1 \cap \Sigma_2 \ldots \cap \Sigma_n$ where Σ_i is a f_i-compatible situation with Σ for $i = 1, \ldots, n$
2. for any action a, $\Phi(a, \emptyset) = \emptyset$.

Lobo et al. extend the function Φ to a plan evaluation function $\Gamma_\Phi(c, \Sigma)$ which allows conditional plans. The definition of $\Gamma_\Phi(c, \Sigma)$ given in [LTM97] is very similar to the definition of $\hat{\Phi}$ and we omit it here for brevity. In the following example, we show the difference between our models and the models of Lobo et al.

Example 12 Let us consider the domain description D_1 from Example 1. The states of D_1 are:

$$\begin{array}{ll} s_1 = \emptyset. & s_5 = \{disarmed\} \\ s_2 = \{locked\} & s_6 = \{disarmed, exploded\} \\ s_3 = \{exploded\} & s_7 = \{disarmed, locked\} \\ s_4 = \{locked, exploded\} & s_8 = \{disarmed, locked, exploded\} \end{array}$$

The initial situation of D_1 is $\Sigma_0 = \{s_1, s_2\}$. There are two *locked*-compatible situations with Σ_0: $\Sigma_1 = \{s_1\}$ and $\Sigma_2 = \{s_2\}$. Thus, if Φ is a model of D_1, then either $\Phi(look, \Sigma_0) = \{s_1\}$ or $\Phi(look, \Sigma_0) = \{s_2\}$, i.e., in the approach of Lobo et al. there are (at least) two different models which differ from each other by the transition functions. On the other hand, in our approach we have two rational models which differ only by the initial c-states. □

The entailment relation w.r.t. Lobo et al.'s semantics is defined next.

Definition 14 $D \models_{LTM}$ **Knows** φ **after** c iff for every model Φ of D, φ is true in $\Gamma_\Phi(c, \Sigma)$.

The following proposition relates Lobo et al's semantics with ours.

Proposition 7 (Equivalence between $\models^r_{\mathcal{A}_K}$ and $\models_{LTM}$) Let D be a domain description, φ be a fluent formula in D, and c be a conditional plan in D. Then,

$$D \models^r_{\mathcal{A}_K} \textbf{Knows } \varphi \textbf{ after } c \quad \text{iff} \quad D \models_{LTM} \textbf{Knows } \varphi \textbf{ after } c.$$

Proof. In [BS98, S00]. □

4.4 Past research on planning with sensing

In the past several planners have been developed that can plan (to some extent) in presence of incompleteness, and some of these planners use sensing actions. In this section we briefly describe a few of these planners, the semantics they use and compare it with our semantics.

4.4.1 Golden and Weld's work

Golden, Weld and their colleagues in [GW96, Gol97, EHW+92] have developed planning languages and planners that can plan in presence of incompleteness, use sensing actions, and plan for 'knowledge' goals. Two of these languages are UWL [EHW+92] and SADL [GW96]. We now list some of their main contributions and compare their formulation with that of ours.

- As evident from the title 'Representing sensing actions – the middle ground revisited' of [GW96], their goal is to develop a middle ground in formulating sensing actions. After reading Golden's thesis [Gol97] and communicating with him it seems that their formulation is close to our 0-approximation, and like 0-approximation it does not do the case-by-case reasoning necessary to make the desired conclusion in D_3 of Example 7. But, while they do not have a soundness result, they have implemented and incorporated their planner into Softbot agents.

- One of their main contributions is their notion of LCW (local closed world) and reasoning with (making inferences and updates) LCW. We do not have a similar notion in this paper.

- They introduce a minimal but extremely useful set of knowledge-temporal goal. In UWL, they have the annotations 'satisfy', 'hands-off' and 'findout' and in SADL, they have 'satisfy', 'hands-off' and 'initially'. Intuitively, the annotation $satisfy(p)$ means to reach a state where p is true and the agent knows that p is true; the annotation *hands-off*(p) means that during the execution of the plan, the truth value of p does not change; and the annotation $initially(p)$ is used to specify the goal of sensing the truth value of p at the time the goal is given, the idea being that after the agent has finished executing his plan, he will know the truth value of p when he started. They also formulate regression with respect to goals formulated using such annotations.

  We have one small reservation about their annotation 'initially'. In [Gol97], Golden says that

  > initially(p) is not achievable by an action that changes the fluent p since such an action only obscures the initial value p. However, changing p after determining its initial value is fine.

  We think the above condition is restrictive because sometimes we can determine the initial value of p, even though we change its value. Consider the case where we do not know the value of p, and we have an action a and an action $sense_g$ whose effects can be described as follows:

  a **causes** g **if** p
  a **causes** $\neg g$ **if** $\neg p$
  a **causes** p **if** $\neg p$
  a **causes** $\neg p$ **if** p
  $sense_g$ **determines** g

  Now, even though the action a changes the value of p, we can find the initial value of p by executing the plan $a; sense_g$.

  We believe these annotations are an important contribution, and additional research is necessary in developing a general knowledge-temporal language for representing more expressive queries over trajectories of c-states. For example, we may want to maintain knows-whether(p), i.e., during the execution of the plan, we do not want to be in a state where we do not know the value of p. This is different from hands-off(p), where we are not allowed to change the value of p, but we don't have to know the value of p all through the trajectory.

- An important difference between their approach and ours is that their focus is on combining planning with execution, while our focus is more close to the classical planning paradigm where we would like to generate a complete plan (possibly with conditional statements and sensing actions) before starting execution. This difference in our focus shows up in the difference in our characterization of sensing actions.

4.4.2 Goldman and Boddy's work

In their KR 94 paper [GB94], Goldman and Boddy use a single model of the world representing the planners state of the knowledge. They then first consider actions with executability conditions (but no conditional effects) and with explicit effects that may make fluents unknown. They define progression (the knowledge state reached after executing an action), and regression with respect to such actions.

Next they extend their action definition to include conditional actions which have a set of mutually exclusive and exhaustive possible outcomes (i.e., exactly one of the outcomes will be the result of the action). They suggest that such conditional actions can be used to describe observation operators by requiring that if such an action is supposed to observe the fluent f, then $unknown(f)$ must be in the executability condition of that action.

They argue about the difficulty of adding conditional effects to their model, which does not have representations of both the state of the world and the planner's state of knowledge.

The following points compare and contrast their approach to that of ours:

- Since they use a single model to represent both the world and the planners knowledge about the world, their formulation is perhaps similar to our approximations, where we also have a single model. But, their formulation has not been shown to be sound with respect a full formulation.
- Their formulation of sensing actions (or observation operators as they call it) can wrongly consider the tossing of a coin action to be a sensing action if the state of the coin (whether 'heads' or 'tails') is unknown before the coin was tossed. Because of this we do not believe that their formulation (restricted to a common subset with our language) will be sound with respect to our formulation.
- They allow actions - even in the absence of conditional effects – to explicitly make fluents unknown. We do not have such actions, but because of conditional effects, our actions can also make fluents unknown.

In a later paper [GB96], they extend classical planning to allow conditional plans, context-dependent actions, and non-deterministic outcomes and argue the necessity of separately modeling the planner's information state and the world state. They use propositional dynamic logic to express conditional plans, and reason about information-gathering (sensing) and the agent's information state. We like their idea of using propositional dynamic logic and results about it and appreciate their goal to explore a middle ground between having a full formulation of sensing actions, and not allowing incompleteness at all. That coincides with our motivation for exploring approximation. But, after carefully reading the paper several times, we believe that more details about their formulation are necessary to fairly and more comprehensively compare their approach to ours.

4.5 Regression

Our focus in this paper so far has been on progression and plan verification. Considering the recent success of model-based planning using propositional satisfiability [KS92, KMS96, KS99] our formulation is geared towards such an approach. Nevertheless, we would like to briefly comment on the notion of regression and its role in conditional planning with sensing actions.

Regression with respect to simple actions has been studied in [Ped94, Rei98]. Scherl and Levsque [SL93] study regression with respect to sensing actions. The intuition behind regression of a formula φ with respect to an action a, is to find a formula ψ such that ψ holds in a situation s if and only if φ will hold in the situation $do(a, \mathsf{s})$. Regression can be used to verify the correctness of a plan by regressing the goal to the initial situation and verifying if the regressed formula holds in the initial situation. Regression can be also used in the least commitment approach to planning [BGPW93, Wel94]. We now present the regression rules for regressing knowledge formulas with respect to conditional plans. The first four rules are adapted from [SL93] and further simplified. The simplification is due to the use of S5 modal logic where only one level of knowledge is sufficient. The regression over conditional plans is our original contribution.

1. For a fluent f and an action a with the ef-propositions
   a **causes** f **if** ϱ_1,...,a **causes** f **if** ϱ_n, a **causes** $\neg f$ **if** ϱ'_1,...,a **causes** $\neg f$ **if** ϱ'_m

   $Regression(f, a) = \bigvee_{i=1}^{n} \varrho_i \vee (f \wedge \neg \bigwedge_{j=1}^{m} \varrho'_j)$.

2. For a fluent formula φ and a non-sensing action a,

   $Regression(\mathbf{Knows}\,(\varphi), a) = \mathbf{Knows}\,(Regression(\varphi, a))$.

3. For a fluent f and a sensing action a which senses the fluents $f_1, \ldots, f_n$, let $I(a, f_1, \ldots, f_n)$ be the set of conjunctions of literals representing the interpretations of the set $\{f_1, \ldots, f_n\}$[6]. Let φ be a fluent formula. Then,

   $Regression(\mathbf{Knows}\,(\varphi), a) = \bigwedge_{\gamma \in I(a, f_1, \ldots, f_n)} \gamma \rightarrow \mathbf{Knows}\,(\gamma \rightarrow \varphi)$.

4. Regression over c-formulas[7]

   - $Regression(\varphi_1 \wedge \varphi_2, a) = Regression(\varphi_1, a) \wedge Regression(\varphi_2, a)$;
   - $Regression(\varphi_1 \vee \varphi_2, a) = Regression(\varphi_1, a) \vee Regression(\varphi_2, a)$;
   - $Regression(\neg\varphi, a) = \neg Regression(\varphi, a)$.

5. Regression over conditional plans and c-formulas. (In the equations below, φ's are c-formulas, and φ_i's are fluents formulas.)

   - $Regression(\varphi, []) = \varphi$;
   - $Regression(\varphi, \alpha; a) = Regression(Regression(\varphi, a), \alpha)$ where α is a sequence of actions;
   - $Regression(\varphi, case\ \varphi_1 \rightarrow p_1, \ldots, \varphi_n \rightarrow p_n\ endcase) = \bigvee_{i=1}^{n} (\mathbf{Knows}\,(\varphi_i) \wedge Regression(\varphi, p_i))$
   - $Regression(\varphi, c_1; c_2; \ldots; c_n) = Regression(Regression(\varphi, c_n), c_1; \ldots; c_{n-1})$ where c_i's are conditional plans satisfying the conditions of Observation 3.1.

The next proposition proves the soundness and completeness of the regression formula.

Proposition 8 *Given a domain description D, let φ be a c-formula, and $\sigma_1, \ldots, \sigma_n$ be the set of grounded initial c-states of D, and c be a conditional plan that is executable in all the grounded initial c-states of D.*

- $\forall i, 1 \leq i \leq n\ \sigma_i \models Regression(\varphi, c)$ *iff* $\forall j, 1 \leq j \leq n\ \hat{\Phi}(c, \sigma_j) \models \varphi$

Proof. In Appendix D. □

---

[6] For example, if a senses f and g then $I(a, f, g) = \{\neg f \wedge \neg g, \neg f \wedge g, f \wedge \neg g, f \wedge g\}$.

[7] A knowledge formula (k-formula) is a formula of the form **Knows** (φ), where φ is a fluent formula, and we say **Knows** (φ) holds in a c-state $\sigma = \langle s, \Sigma \rangle$, if φ holds in all states of Σ. A combined formula (c-formula) is a formula constructed using fluent formulas, k-formulas and the propositional connectives, and when a c-formula holds in a c-state is defined in a straightforward way.

5 Conclusion and future work

In this paper we presented a high-level action description language that takes into account sensing actions and distinguishes between the state of the world and the state of the knowledge of an agent about the world. We gave sound and complete translation of domain descriptions in our language to theories in first-order logic and have similar translations [BS98, S00] to disjunctive logic programming. We compared our formulation with others and analyze the state space of our formulation and that of the others. We then gave sound approximations of our formulation with a much smaller state space. We believe the approximations in this paper will be very important in developing practical planners.

Some of our future plans are:

- We would like to analyze existing planners[8] that construct conditional plans and use sensing actions and develop more efficient planners based on the approximations described in this paper. We have made a head start in this direction by implementing a simple generate and test planner in Prolog.
- We would like to further explore the notions of 1-Approximation and ω-Approximation.
- We would like to follow satisfiability planning [KS92, KMS96, KS99] and S-model based planning [DNK97, EL99, Erd] by adapting our classical logic and logic programming formulations to plan with sensing actions.
- We would like to adapt our formulation of sensing to other action description languages – particularly the action description language for narratives [BGP97, BGP98] – to develop notions of diagnosis and diagnostic and repair planning with respect to a narrative. Intuitively, the later means to develop a plan – possibly with sensing actions – that leads to a unique diagnosis of a system.

Acknowledgment

We would like to thank the anonymous reviewer for his/her valuable comments that help us to improve the paper in many ways.

References

[AB91] K. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3,4):335–365, 1991.

[Bar95] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proc. of IJCAI 95*, pages 2017–2023, 1995.

[BG93] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 866–871, 1993.

---

[8] From the following quote in [GW96]:
"In UWL (and in SADL) individual literals have truth values expressed in three valued logic: T, F, U (unknown). " it seems that they are using an approximation. We would like to analyze this planner to figure out what kind of approximation they are using and if it is sound w.r.t. one of the formulations discussed in this paper.

[BG94] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[BG97] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117, May 1997.

[BGP97] C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming*, 31(1-3):201–243, May 1997.

[BGP98] C. Baral, A. Gabaldon, and A. Provetti. Formalizing Narratives using nested circumscription. *Artificial Inteligence*, 104(1-2):107–164, Sept 1998.

[BGPW93] A. Barrett, K. Golden, J. Penberthy, and D. Weld. UCPOP User's manual, version 2.0. Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington, 1993.

[BHL95] F. Bacchus, J. Halpern, and H. Levesque. Reasoning about noisy sensors in the situation calculus. In *IJCAI 95*, pages 1933–1940, 1995.

[BKT99] C. Baral, V. Kreinovich, and R. Trejo. Planning and approximate planning in presence of incompleteness. In *IJCAI 99*, pages 948–953, 1999.

[BS98] C. Baral and T. Son. Formalizing sensing actions: a transition function based approach. Technical report, Dept of Computer Science, University of Texas at El Paso (http://cs.utep.edu/chitta/chitta.html), 1998.

[CSW95] W. Chen, T. Swift, and D. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–201, 1995.

[DMB92] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proc. of ECAI92*, 1992.

[DNK97] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proc. of European conference on Planning*, pages 169–181, 1997.

[Dun93] P. Dung. Representing actions in logic programming and its application in database updates. In D. S. Warren, editor, *Proc. of ICLP-93*, pages 222–238, 1993.

[EFG$^+$99] T. Eiter, W. Faber, G. Gottlob, C. Koch, C. Mateis, N. Leone, G. Pfeifer, and F. Scarcello. The dlv system. In J. Minker, editor, *Pre-prints of Workshop on Logic-Based AI*, 1999.

[EG93] T. Eiter and G Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium (ILPS), Vancouver*, pages 266–178. MIT Press, 1993.

[EHW$^+$92] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach to planning with incomplete information. In *KR 92*, pages 115–125, 1992.

[EL99] E. Erdem and V. Lifschitz. Transformations of logic programs related to causality and planning. In *Proc. Fifth International Conf. on Logic Programming and Non-monotonic reasoning (to appear)*, 1999.

[Erd] E. Erdem. Application of logic programming to planning: computational experiments. draft (http://www.cs.utexas.edu/tag).

[FHMV95] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge.* MIT press, 1995.

[GB94] R. Goldman and M. Boddy. Representing uncertainity in simple planners. In *KR 94*, pages 238–245, 1994.

[GB96] R. Goldman and M. Boddy. Expressive planning and explicit knowledge. In *AIPS 96*, pages 110–117, 1996.

[Gel91] M. Gelfond. Strong introspection. In *Proc. AAAI-91*, pages 386–391, 1991.

[GEW96] K. Golden, O. Etzioni, and D. Weld. Planning with execution and incomplete informations. Technical report, Dept of Computer Science, University of Washington, TR96-01-09, February 1996.

[GKL97] E. Giunchiglia, G. Kartha, and V. Lifschitz. Representing action: indeterminacy and ramifications. *Artificial Intelligence*, 95:409–443, 1997.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In *Joint International Conference and Symposium on Logic Programming.*, pages 559–573, 1992.

[GL93] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[Gol97] K. Golden. *Planning and knowledge representation for softbots.* PhD thesis, University of Washington, November 1997.

[GW96] K. Golden and D. Weld. Representing sensing actions: the middle ground revisited. In *KR 96*, pages 174–185, 1996.

[Haa86] L. Haas. A syntactic theory of belief and action. *Artificial Intelligence*, 28:245–292, May 1986.

[Kar93] G. Kartha. Soundness and completeness theorems for three formalizations of action. In *IJCAI 93*, pages 724–729, 1993.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proc. of KR 96*, pages 374–384, 1996.

[KOG92] K. Krebsbach, D. Olawsky, and M. Gini. An empirical study of sensing and defaulting in planning. In *First Conference of AI Planning Systems*, pages 136–144, 1992.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI-92*, pages 359–363, 1992.

[KS99] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proc. of IJCAI 99*, pages 318–325, 1999.

[Lev96] H. Levesque. What is planning in the presence of sensing? In *AAAI 96*, pages 1139–1146, 1996.

[Lif95] V. Lifschitz. Nested abnormality theories. *Artificial Intelligence*, 74:351–365, 1995.

[Lif97] V. Lifschitz. Two components of an action language. *Annals of Math and AI*, 21(2-4):305–320, 1997.

[Lif99] V. Lifschitz. Answer set planning. In *Proc. International Conf. on Logic Programming (to appear)*, 1999.

[Lin95] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI 95*, pages 1985–1993, 95.

[LR94] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, October 1994.

[LS92] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc. of AAAI–92*, pages 590–595, 1992.

[LT] J. Lobo and G. Trajcevski. Minimal and consistent evolution of knowledge bases. To appear *Journal of Applied Non-classical Logics.*

[LTM97] J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language $\mathcal{A}$. In *AAAI 97*, pages 454–459, 1997.

[Moo79] R. Moore. *Reasoning about knowledge and action.* PhD thesis, MIT, Feb 1979.

[Moo85] R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. Moore, editors, *Formal theories of the commonsense world.* Ablex, Norwood, NJ, 1985.

[MS89] W. Marek and V.S. Subrahmanian. The relationship between logic program semantics and non–monotonic reasoning. In G. Levi and M. Martelli, editors, *Proc. of the Sixth Int'l Conf. on Logic Programming*, pages 600–617, 1989.

[MS94] R. Miller and M. Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513–530, October 1994.

[MT95] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI 95*, pages 1978–1984, 95.

[NS97] I. Niemela and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429, 1997.

[PC96] L. Pryor and G. Collins. Planning for contingencies: a decision based approach. *Journal of AI research*, 1996.

[Ped94] E. Pednault. ADL and the state-transition model of actions. *Journal of Logic and Computation*, 4(5):467–513, October 1994.

[PS92] M. Peot and D. Smith. Conditional non-linear planning. In *First Conference of AI Planning Systems*, pages 189–197, 1992.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[Rei98] R. Reiter. *Knowledge in action: logical foundation for describing and implementing dynamical systems.* MIT press, 1998. manuscript.

[S00] S.C. Tran. *Reasoning about sensing actions and its application to diagnostic problem solving.* PhD thesis, University of Texas at El Paso, February 2000.

[Sha97] M. Shanahan. *Solving the frame problem: A mathematical investigation of the commonsense law of inertia.* MIT press, 1997.

[SL93] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *AAAI 93*, pages 689–695, 1993.

[SW98] D. Smith and D. Weld. Conformant graphplan. In *AAAI 98*, 1998.

[Tur94] H. Turner. Signed logic programs. In *Proc. of the 1994 International Symposium on Logic Programming*, pages 61–75, 1994.

[Tur97] H. Turner. Representing actions in logic programs and default theories. *Journal of Logic Programming*, 31(1-3):245–298, May 1997.

[WAS98] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainity and sensing actions. In *AAAI 98*, pages 897–904, 1998.

[Wel94] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, winter 1994.

Appendix A

Proposition 9 *For every sequence of actions α of D_1,*

$$D_1 \not\models^r_{\mathcal{A}_K} \textbf{Knows } \mathit{disarmed} \wedge \neg \mathit{exploded} \textbf{ after } \alpha.$$

Proof. Let $s_1 = \emptyset$ and $s_2 = \{\mathit{locked}\}$. The two initial c-states of D_1 are $\sigma_1 = \langle s_1, \{s_1, s_2\}\rangle$ and $\sigma_2 = \langle s_2, \{s_1, s_2\}\rangle$. Let α be an arbitrary sequence of actions of D_1 and β be its longest prefix which does not contain the action *disarm*. Since no action in β changes the value of the fluent *exploded*, we can conclude that β is executable in σ_1 and σ_2. Let $\hat{\Phi}(\beta, \sigma_1) = \langle s_{1\beta}, \Sigma_{1\beta}\rangle$ and $\hat{\Phi}(\beta, \sigma_2) = \langle s_{2\beta}, \Sigma_{2\beta}\rangle$. We first prove by induction over the length of β, denoted by $|\beta|$, the following:

$$\{s_1, s_2\} = \{s_{1\beta}, s_{2\beta}\}, s_{1\beta} \in \Sigma_{1\beta} \quad \text{and} \quad s_{2\beta} \in \Sigma_{2\beta}. \tag{2.14}$$

Base case: $|\beta| = 0$, i.e., $\beta = []$. By definition of $\hat{\Phi}$, we have that $\hat{\Phi}(\beta, \sigma_1) = \sigma_1$ and $\hat{\Phi}(\beta, \sigma_2) = \sigma_2$. Thus (2.14) holds.

Inductive step: Assume that we have proved (2.14) for $|\beta| < n$. We need to prove (2.14) for $|\beta| = n$. Let $\beta = \beta'; a$. Since $a \neq disarm$, a is either *turn* or *look*. If $a = turn$, we have that $\{s_{1\beta}, s_{2\beta}\} = \{Res(a, s_{1\beta'}), Res(a, s_{2\beta'})\} = \{Res(a, s_1), Res(a, s_2)\} = \{s_1, s_2\}$. And, if $a = look$, we have that $\{s_{1\beta}, s_{2\beta}\} = \{s_{1\beta'}, s_{2\beta'}\} = \{s_1, s_2\}$. Furthermore, by definition of Φ, from $s_{1\beta'} \in \Sigma_{1\beta'}$ we can conclude that $Res(a, s_{1\beta'}) = s_{1\beta} \in \Sigma_{1\beta}$. Similarly, we have $s_{2\beta} \in \Sigma_{2\beta}$. The inductive hypothesis is proven.

We now use (2.14) to prove the proposition. Recall that D_1 has two models (σ_1, Φ) and (σ_2, Φ).

From the construction of β, there are three cases: (1) $\beta = \alpha$, (2) $\beta; disarm = \alpha$, and (3) $\beta; disarm$ is a proper prefix of α.

Case 1: $\beta = \alpha$. Since *disarmed* is not true in s_1 and s_2, *disarmed* is not known to be true in $\hat{\Phi}(\beta, \sigma_1)$ and $\hat{\Phi}(\beta, \sigma_2)$. Thus, by definition 6, $D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** *disarmed* **after** α;

Case 2: $\beta; disarm = \alpha$. It follows from (2.14) that $Res(disarm, s_1) = s_3$ belongs to $\Sigma_{1\alpha}$ or $\Sigma_{2\alpha}$ where $\hat{\Phi}(\beta; disarm, \sigma_1) = \langle s_{1\alpha}, \Sigma_{1\alpha}\rangle$ and $\hat{\Phi}(\beta; disarm, \sigma_2) = \langle s_{2\alpha}, \Sigma_{2\alpha}\rangle$. Since $\neg exploded$ does not holds in s_3, we conclude that $\neg exploded$ is not known to be true in $\hat{\Phi}(\beta; disarm, \sigma_1)$ or $\hat{\Phi}(\beta; disarm, \sigma_2)$. Again, by Definition 6, $D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** $\neg exploded$ **after** α.

Case 3: $\beta; disarm$ is a proper prefix of α. Since $s_1 \in \{s_{1\beta}, s_{2\beta}\}$, either $s_{1\beta} = s_1$ or $s_{2\beta} = s_1$. Since $Res(disarm, s_1) = s_3$ and none of the actions of D_1 is executable in s_3 we can conclude that $\hat{\Phi}(\alpha, \sigma_1) = \perp$ or $\hat{\Phi}(\alpha, \sigma_2) = \perp$. This means that α is not executable in all c-initial states of D_1. By Definition 6, $D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** $disarmed \wedge \neg exploded$ **after** α

The above three cases show that $D_1 \not\models^r_{\mathcal{A}_K}$ **Knows** $disarmed \wedge \neg exploded$ **after** α. This proves the proposition. □

Appendix B - Soundness and Completeness of the translation D to $R(D)$

We now prove the propositions 2 and 3. Recall that we assume that D is a domain description with $\mathbf{m}$ v-propositions **initially** $G_1, \ldots,$ **initially** $G_{\mathbf{m}}$ and $\mathbf{n}$ sensing actions $K_1, \ldots, K_{\mathbf{n}}$ with the k-propositions K_1 **determines** $F_1, \ldots, K_{\mathbf{n}}$ **determines** $F_{\mathbf{n}}$. And, we also assume that for each action A, D contains at least one executability condition whose action is A and each sensing action occurs only in one k-proposition.

In the following, we write $\sigma.1$ and $\sigma.2$ to denote the first and second component of a c-state σ, respectively. In other words, if $\sigma = \langle s, \Sigma\rangle$, then $\sigma.1$ and $\sigma.2$ denote s and Σ respectively. For a state s and an action sequence $\alpha = [a_1; \ldots; a_k]$, where $[]$ denotes the empty sequence of actions, if α is executable in s then $Res(\alpha, s)$ denotes the state $Res(a_k, Res(a_{k-1}, \ldots, Res(a_1, s)))$; otherwise, $Res(\alpha, s) = \perp$ (or undefined). A *situation interpretation* in D is defined by a sequence of actions α followed by a state s, such that α is executable in s, and is denoted by $[\alpha]s$. For an interpretation I of the theory $R(D)$, we write $I[\![p]\!]$ to denote the set of tuples belonging to the extent of the predicate p in I. $I[\![f]\!](\vec{x})$ denotes the object which function f maps $\vec{x}$ into in I. When f is a 0-ary function symbol, we simplify $I[\![f]\!]()$ to $I[\![f]\!]$.

Definition 15 *Let D be a domain description and $M = (\sigma_0, \Phi)$ be a model of D. The* M-interpretation *of $R(D)$, denoted by M_R, is defined as follows.*

The universes of M_R:

(U.1) *the universe of actions, denoted by* $|M_R|_{action}$*, is the set of actions* $\mathbf{A}$ *of* D*, i.e.,* $|M_R|_{action} = \mathbf{A}$*;*

(U.2) *the universe of fluents, denoted by* $|M_R|_{fluent}$*, is the set of fluents* $\mathbf{F}$ *of* D*, i.e.,* $|M_R|_{fluent} = \mathbf{F}$*; and*

(U.3) *the universe of situations, denoted by* $|M_R|_{situation}$*, is defined by the set of situation interpretations, i.e.,* $|M_R|_{situation} = \{[\alpha]s \mid s \subseteq \mathbf{F}, \alpha \text{ is an action sequence executable in } s\} \cup \{\bot\}$ *where* $\bot$ *denotes the "impossible" situation.*

The interpretations of M_R*:*

(I.1) *Fluent constants and action constants are interpreted as themselves;*

(I.2) *Each situation* S *is interpreted as a situation interpretation. In particular,* $M_R[\![\mathsf{S}_0]\!] = []\sigma_0.1$*;*

(I.3) *The interpretation of the predicate* $Holds$ *is defined by* $\langle F, [\alpha]S\rangle \in M_R[\![Holds]\!]$ *iff* $Res(\alpha, S)$ *is defined and* F *holds in* $Res(\alpha, S)$*;*

(I.4) *The interpretation of the predicate* K *is defined inductively as follows:*

- $\langle []S', []S\rangle \in M_R[\![K]\!]$ *if* $S = \sigma_0.1$ *and* $S' \in \sigma_0.2$*; and*
- $\langle [\alpha; A]S', [\alpha; A]S\rangle \in M_R[\![K]\!]$ *if the following conditions are satisfied*
  - $\langle [\alpha]S', [\alpha]S\rangle \in M_R[\![K]\!]$*;*
  - A *is executable in* $Res(\alpha, S)$ *and* $Res(\alpha, S')$*; and*
  - *either* A *is a non-sensing action or* A *is a sensing action that senses the fluent* F *and* $Res(\alpha, S)$ *and* $Res(\alpha, S')$ *agree on* F*.*
- $\langle [\alpha']S', [\alpha]S\rangle \notin M_R[\![K]\!]$ *otherwise;*

(I.5) *The interpretation of the function* do *is defined by* $M_R[\![do]\!](A, [\alpha]S) = [\alpha; A]S$ *if* A *is executable in* $Res(\alpha, S)$*; otherwise* $M_R[\![do]\!](A, [\alpha]S) = \bot$*.*

□

The interpretation M_R is then extended to the predicates introduced in Subsection 2.3 such as γ_F^+, γ_F^-, $Poss$, etc. For example, for a situation interpretation $[\alpha]S$,

- $\langle \varphi, [\alpha]S\rangle \in M_R[\![Holds]\!]$ iff φ holds in $Res(\alpha, S)$; or
- $\langle A, [\alpha]S\rangle \in M_R[\![\gamma_F^+]\!]$ iff there exists an ef-proposition "A **causes** F **if** ρ" $\in D$ such that $\langle \rho, [\alpha]S\rangle \in M_R[\![Holds]\!]$; or
- $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$ iff there exists an ex-proposition "**executable** A **if** ρ" $\in D$ such that $\langle \rho, [\alpha]S\rangle \in M_R[\![Holds]\!]$;
- etc.

We next prove some lemmas about the relationship between a model M of D and the M-interpretation M_R which will be used in proving the propositions 2 and 3. For convenience, for a formula φ in the language of $R(D)$, if φ is true in M_R we write $M_R \models \varphi$.

Lemma 2.1 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, a fluent F, an action A, and a situation interpretation $[\alpha]S$*

(i) $\langle A, [\alpha]S\rangle \in M_R[\![\gamma_F^+]\!]$ *iff* $F \in E_A^+(Res(\alpha, S))$*; and*

(ii) $\langle A, [\alpha]S\rangle \in M_R[\![\gamma_F^-]\!]$ *iff* $F \in E_A^-(Res(\alpha, S))$.

Proof. $[\alpha]S$ is a situation interpretation implies that α is executable in S. Therefore, $Res(\alpha, S)$ is defined. We have that

$\langle A, [\alpha]S\rangle \in M_R[\![\gamma_F^+]\!]$
iff there exists an ef-proposition "A **causes** F **if** ϱ" $\in D$ such that $\langle \varrho, [\alpha]S\rangle \in M_R[\![Holds]\!]$ (Definition of γ_F^+)
iff there exists an ef-proposition "A **causes** F **if** ϱ" $\in D$ such that ϱ holds in $Res(\alpha, S)$ (by Item (I.3) of Definition 15)
iff $F \in E_A^+(Res(\alpha, S))$. Thus (i) is proved.

Similarly, we can prove (ii). □

Lemma 2.2 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, an action A, and a situation interpretation $[\alpha]S$,*

(i) $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$ *iff A is executable in $Res(\alpha, S)$; and*

(ii) if $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$ *then* $M_R[\![do]\!](A, [\alpha]S) = [\alpha; A]S$.

Proof. Again, since $[\alpha]S$ is a situation interpretation, we have that α is executable in S. Thus $Res(\alpha, S)$ is defined. From the definition of $Poss$, we have that $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$
iff there exists an ex-proposition "**executable** A **if** ρ" $\in D$ such that $\langle \rho, [\alpha]S\rangle \in M_R[\![Holds]\!]$
iff there exists an ex-proposition "**executable** A **if** ρ" $\in D$ such that ρ holds in $Res(\alpha, S)$ (by Item (I.3) of Definition 15)
iff A is executable in $Res(\alpha, S)$. (1)

The second item follows immediately from (1) and Item (I.5) of Definition 15. □

Lemma 2.3 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, M_R satisfies the axiom (2.8).*

Proof. Consider an action A, a situation S, and a positive fluent literal F. Let $M_R[\![\mathsf{S}]\!] = [\alpha]S$. The axiom (2.8) is true in M_R if $\langle A, [\alpha]S\rangle \notin M_R[\![Poss]\!]$. Thus we need to prove that it is also true in M_R when $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$.

From $\langle A, [\alpha]S\rangle \in M_R[\![Poss]\!]$ and Lemma 2.2, we have that A is executable in $Res(\alpha, S)$. Hence, by (I.5) of Definition 15, $M_R[\![do]\!](A, [\alpha]S) = [\alpha; A]S$. (1)

We have that $M_R \models Holds(F, do(A, \mathsf{S}))$
iff $\langle F, M_R[\![do]\!](A, [\alpha]S)\rangle \in M_R[\![Holds]\!]$
iff $\langle F, [\alpha; A]S\rangle \in M_R[\![Holds]\!]$ (because $M_R[\![do]\!](A, [\alpha]S) = [\alpha; A]S$, by (1))
iff F holds in $Res([\alpha; A], S)$ (by Item (I.3) of Definition 15)
iff $F \in Res(A, Res(\alpha, S))$
iff $F \in E_A^+(Res(\alpha, S))$
  or $F \in Res(\alpha, S) \wedge F \notin E_A^-(Res(\alpha, S))$ (by definition of Res)
iff $\langle A, [\alpha]S\rangle \in M_R[\![\gamma_F^+]\!]$ (by Lemma 2.1)
  or $(\langle F, [\alpha]S\rangle \in M_R[\![Holds]\!] \wedge \langle A, [\alpha]S\rangle \notin M_R[\![\gamma_F^-]\!])$ (by Lemma 2.1 and (I.3) of Definition 15)
iff $M_R \models \gamma_F^+(A, \mathsf{S}) \vee (Holds(F, \mathsf{S}) \wedge \neg\gamma_F^-(A, \mathsf{S}))$ (2)

Similarly, we can prove (2) for negative fluent literal. Thus M_R satisfies (2.8). □

Lemma 2.4 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, M_R satisfies the axioms (2.10)-(2.12).*

Proof. Consider a situation S_1 such that $M_R \models K(\mathsf{S}_1, \mathsf{S}_0)$.
This means that $\langle M_R[\![\mathsf{S}_1]\!], M_R[\![\mathsf{S}_0]\!]\rangle \in M_R[\![K]\!]$. Hence, by Item (I.4) of Definition 15 and from the fact that $M_R[\![\mathsf{S}_0]\!] = []\sigma_0.1$, we can conclude that $M_R[\![\mathsf{S}_1]\!] = []S$ for some $S \in \sigma_0.2$. Since M is a model of D, S is an initial state of D. Therefore, G_i holds in S for every $i = 1, \ldots, \mathbf{m}$. Because $[]$ is executable in S, by Item (I.3) of Definition 15, we conclude that $\langle \bigwedge_{i=1}^{\mathbf{m}} G_i, []S\rangle \in M_R[\![\mathit{Holds}]\!]$, i.e., $M_R \models \bigwedge_{i=1}^{\mathbf{m}} \mathit{Holds}(G_i, \mathsf{S}_1)$. Since this holds for every S_1 such that $M_R \models K(\mathsf{S}_1, \mathsf{S}_0)$, we can conclude that M_R satisfies (2.11). (1)

Since M is a model of D, we have that $\sigma_0.1 \in \sigma_0.2$. Thus, by Item (I.4) of Definition 15, we have that $M_R \models K(\mathsf{S}_0, \mathsf{S}_0)$. Hence, M_R satisfies (2.12). (2)

Since $\sigma_0.1$ is also an initial state of D, from (1) and (2), we can conclude that M_R satisfies (2.10). (3)

The lemma follows from (1)-(3). □

Lemma 2.5 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, M_R satisfies the axiom 2.9.*

Proof. Consider an action A and a situation S. Let $M_R[\![\mathsf{S}]\!] = [\alpha]S$. Similar to Lemma 2.3, it suffices to prove that M_R satisfies the axiom 2.9 when $\langle A, [\alpha]S\rangle \in M_R[\![\mathit{Poss}]\!]$. By Lemma 2.2, this implies that $\mathit{Res}(\alpha, S)$ is defined, A is executable in $\mathit{Res}(\alpha, S)$, and $M_R[\![\mathit{do}]\!](A, [\alpha]S) = [\alpha; A]S$. (1)

There are two cases:

- **Case 1:** $M_R \models K(\mathsf{S}_2, \mathit{do}(A, \mathsf{S}))$ for some situation S_2. Let $M_R[\![\mathsf{S}_2]\!] = [\alpha_2]S_2$. We will prove that the following formula is also true in M_R:

$$\begin{array}{c}\exists \mathsf{s}_1.[(K(\mathsf{s}_1, \mathsf{S}) \wedge \mathit{Poss}(A, \mathsf{s}_1) \wedge \mathsf{S}_2 = \mathit{do}(A, \mathsf{s}_1)) \wedge \\ ((\wedge_{i=1}^{\mathbf{n}} A \neq K_i) \vee \bigvee_{i=1}^{\mathbf{n}} (A = K_i \wedge \mathit{Holds}(F_i, \mathsf{s}_1) \equiv \mathit{Holds}(F_i, \mathsf{S}))]\end{array} \tag{2.15}$$

  $M_R \models K(\mathsf{S}_2, \mathit{do}(A, \mathsf{S}))$ implies that $\langle [\alpha_2]S_2, M_R[\![\mathit{do}]\!](A, [\alpha]S)\rangle \in M_R[\![K]\!]$, and hence, $\langle [\alpha_2]S_2, [\alpha; A]S\rangle \in M_R[\![K]\!]$. By Item (I.4) of Definition 15 we have that $\alpha_2 = \alpha; A$, and

  - $\langle [\alpha]S_2, [\alpha]S\rangle \in M_R[\![K]\!]$; (2)
  - A is executable in $\mathit{Res}(\alpha, S_2)$ and $\mathit{Res}(\alpha, S)$; (3)
  - A is a non-sensing action or if A is sensing action, say K_i, then $\mathit{Res}(\alpha, S_2)$ and $\mathit{Res}(\alpha, S)$ agree on F_i; (4)

  Let S_1 be a situation such that $M_R[\![\mathsf{S}_1]\!] = [\alpha]S_2$. It follows from (3) and Lemma 2.2 that $\langle A, [\alpha]S_2\rangle \in M_R[\![\mathit{Poss}]\!]$. Furthermore, from (3) and (4), we can conclude that $M_R \models ((\wedge_{i=1}^{\mathbf{n}} A \neq K_i) \vee \bigvee_{i=1}^{\mathbf{n}} (A = K_i \wedge \mathit{Holds}(F_i, \mathsf{S}_2) \equiv \mathit{Holds}(F_i, \mathsf{S}))$. Together with (2), we have that S_1 satisfies (2.15).

- **Case 2:** Assume that the formula (2.15) is true for some S_1 with $M_R[\![\mathsf{S}_1]\!] = [\alpha_1]S_1$. We want to show that

$$M_R \models K(\mathsf{S}_2, \mathit{do}(A, \mathsf{S})) \tag{2.16}$$

where $\mathsf{S}_2 = do(A, \mathsf{S}_1)$. Similar to the above case, from $M_R \models K(\mathsf{S}_1, \mathsf{S})$ and $\langle A, [\alpha_1]S_1\rangle \in M_R[\![Poss]\!]$, we can conclude that $\alpha_1 = \alpha$, and A is executable in $Res(\alpha, S_1)$. Thus, $M_R[\![do]\!](A, [\alpha]S_1) = [\alpha; A]S_1$. (5)

It follows from (2.15) that $M_R \models ((\wedge_{i=1}^{\mathbf{n}} A \neq K_i) \vee \bigvee_{i=1}^{\mathbf{n}} (A = K_i \wedge Holds(F_i, \mathsf{S}_1) \equiv Holds(F_i, \mathsf{S}))$. This implies that either A is a non-sensing action or A is a sensing action, say K_i, and $Res(\alpha, \mathsf{S}_1)$ and $Res(\alpha, \mathsf{S})$ agree on F_i. (6)

It follows from (1) and (5)-(6) and (I.4) of Definition 15 that $\langle [\alpha; A]S_1, [\alpha; A]S\rangle \in M_R[\![K]\!]$. This, together with (1) and (5), implies that $\langle M_R[\![do]\!](A, [\alpha]S_1), M_R[\![do]\!](A, [\alpha]S)\rangle \in M_R[\![K]\!]$ which proves that (2.16) is true in M_R.

It follows from the above two cases that M_R satisfies (2.9). □

Lemma 2.6 *For each model $M = (\sigma_0, \Phi)$ of a domain description D, the M-interpretation of $R(D)$, M_R, is a model of $R(D)$.*

Proof. It follows from Lemma 2.3-2.5 thatM_R satisfies the axioms (2.8)-(2.12). It is easy to see that the closure assumptions and unique name assumptions for fluents and actions are satisfied by M_R too. Thus, M_R is a model of $R(D)$. □

Lemma 2.7 *For each situation interpretation $[\alpha]S$, the following statements are equivalent:*

(i) $\langle [\alpha]S, [\alpha]\sigma_0.1\rangle \in M_R[\![K]\!]$; and

(ii) α is executable in S and $\sigma_0.1$, and $Res(\gamma, S) \in \hat{\Phi}(\gamma, \sigma_0).2$ for every prefix γ of α.

Proof. Induction over $|\alpha|$.

Base case: $|\alpha| = 0$, i.e., $\alpha = []$. By Item (I.4) of Definition 15, $\langle []S, []\sigma_0.1\rangle \in M_R[\![K]\!]$ iff $S \in \sigma_0.2 = \hat{\Phi}(\alpha, \sigma_0).2$. Together with the fact that $[]$ is executable in S and $\sigma_0.1$, we conclude the base case.

Inductive step: Assume that we have proved the lemma for $|\alpha| < k$. We need to show it for $|\alpha| = k$. Let $\alpha = [\beta; A]$. Then, $|\beta| < k$. We consider two cases:

"(i) ⇒ (ii)": From Item (I.4) of Definition 15 and (i) for $\alpha = [\beta; A]$, we have that

- $\langle [\beta]S, [\beta]\sigma_0.1\rangle \in M_R[\![K]\!]$; (1)
- A is executable in $Res(\beta, S)$ and $Res(\beta, \sigma_0.1)$; and (2)
- if A senses F_j then $Res(\beta, S)$ and $Res(\beta, \sigma_0.1)$ agree on F_j. (3)

By inductive hypothesis, from (1), we conclude that β is executable in S and $\sigma_0.1$, and $Res(\gamma, S) \in \hat{\Phi}(\gamma, \sigma_0).2$ for every prefix γ of β. (4)

From (2) and the fact that β is executable in S and $\sigma_0.1$, we have that $[\beta; A]$ is executable in S and $\sigma_0.1$. (5)

From (3) and the fact that $Res(\beta, S) \in \hat{\Phi}(\beta, \sigma_0).2$, we conclude that $Res([\beta; A], S) \in \hat{\Phi}([\beta; A], \sigma_0).2$. (6)

The inductive step for this direction follows from (4)-(6).

"(ii) $\Rightarrow$ (i)": α is executable in S and $\sigma_0.1$ implies that β is executable in S and $\sigma_0.1$. Furthermore, every prefix of β is a prefix of α. Hence, by inductive hypothesis, we have that $\langle [\beta]S, [\beta]\sigma_0.1\rangle \in M_R[\![K]\!]$. (7)

α is executable in S and $\sigma_0.1$ also implies that A is executable in $Res(\beta, S)$ and $Res(\beta, \sigma_0.1)$. (8)

$Res(\beta, S) \in \hat{\Phi}(\beta, \sigma_0).2$ and $Res([\beta; A], S) \in \hat{\Phi}([\beta; A], \sigma_0).2$ implies that if A is a sensing action, say K_i, then $Res(\beta, S)$ and $Res(\beta, \sigma_0.1)$ must agree on F_i. (9)

It follows from (7)-(9) and Item (I.4) of Definition 15 that $\langle [\beta; A]s, [\beta; A]\sigma_0.1\rangle \in M_R[\![K]\!]$. This concludes the inductive step for this direction.

The inductive step is proved. Hence, by mathematical induction, we conclude the lemma. □

Lemma 2.8 *For every state S and action sequence α, α is executable in S iff $\langle \alpha, []S\rangle \in M_R[\![Poss]\!]$.*

Proof. By induction over $|\alpha|$.

Base case: $\alpha = []$. The lemma is trivial because $[]$ is executable in every state S and $\langle [], []S\rangle \in M_R[\![Poss]\!]$ for every state S.

Inductive step: Assume that we have proved the lemma for $|\alpha| < k$. We need to show it for $|\alpha| = k$. Let $\alpha = [\beta; A]$. We have that $|\beta| < k$ and
$\beta; A$ is executable in S
iff β is executable in S and A is executable in $Res(\beta, S)$
iff $\langle \beta, []S\rangle \in M_R[\![Poss]\!]$ (by inductive hypothesis)
and $\langle A, M_R[\![do]\!](\beta, []S)\rangle \in M_R[\![Poss]\!]$ (by Lemma 2.2)
iff $\langle [\beta; A], []S\rangle \in M_R[\![Poss]\!]$. □

Lemma 2.9 *Let D be a domain description and $M = (\sigma_0, \Phi)$ be a model of D. Then, there exists a model M_R of $R(D)$ such that for any fluent formula φ and sequence of actions α of D, α is executable in σ_0 and φ is known to be true in $\hat{\Phi}(\alpha, \sigma_0)$ iff $M_R \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$.*

Proof. Let M_R be the M-interpretation of $R(D)$. By Lemma 2.6 we have that M_R is a model of $R(D)$. We will prove that M_R satisfies the conclusion of the lemma.

We have that

- α is executable in σ_0
iff α is executable in $\sigma_0.1$
iff $\langle \alpha, []\sigma_0.1\rangle \in M_R[\![Poss]\!]$ (by Lemma 2.8)
iff $M_R \models Poss(\alpha, \mathsf{S}_0)$. (1)

- φ is known to be true in $\hat{\Phi}(\alpha, \sigma_0)$
iff for every $S \in \hat{\Phi}(\alpha, \sigma_0).2$, φ holds in S (by definition)
iff for every $S \in \sigma_0.2$ such that
α is executable in S and $Res(\gamma, S) \in \hat{\Phi}(\gamma, \sigma_0).2$ for every prefix γ of α, φ holds in $Res(\alpha, S)$
iff for every $[\alpha]S$ such that $\langle [\alpha]S, [\alpha]\sigma_0.1\rangle \in M_R[\![K]\!]$ (by Lemma 2.7)
and $\langle \varphi, [\alpha]S\rangle \in M_R[\![Holds]\!]$ (by Item (I.3) of Definition 15)
iff $M_R \models Knows(\varphi, do(\alpha, \mathsf{S}_0))$ (by definition of $Knows$)(2)

The lemma follows from (1) and (2). □

We now prove the counterpart of Lemma 2.9. Let D be a domain description and M_R be a model of $R(D)$. Since $R(D)$ contains the DCA and UNA axioms for actions and fluents we can assume that the domains of actions and fluents are $\mathbf{A}$ and $\mathbf{F}$ respectively, i.e., $|M_R|_{action} = \mathbf{A}$ and $|M_R|_{fluent} = \mathbf{F}$. In what follows, whenever we say a situation S we mean a ground situation term. We define

Definition 16 *Let D be a domain description and M_R be a model of $R(D)$. For each ground situation term s in M_R, let $\mathsf{s}^* = \{F \mid Holds(F, \mathsf{s})$ is true in M_R, F is a positive fluent literal$\}$*[9]*. The M_R-*initial c-state *of D, denoted by σ_0^*, is defined as follows.*

(M.1) $\sigma_0^*.1 = \mathsf{S}_0^*$;

(M.2) $\sigma_0^*.2 = \{\mathsf{s}^* \mid K(\mathsf{s}, \mathsf{S}_0)$ is true in $M_R\}$.

*We call $M = (\sigma_0^*, \Phi)$, where σ_0^* is the M_R-initial c-state and Φ is the transition function of D (defined in Definition 2), the M_R-*based model of D.

Lemma 2.10 *For a domain description D and a model M_R of $R(D)$, the M_R-*based model of D. *is a model of D.*

Proof. Consider an arbitrary v-proposition "**initially** G_i" of D. There are two cases:

- G_i is a positive literal. Since M_R is a model of $R(D)$, from (2.10) we have that $Holds(G_i, \mathsf{S}_0)$ is true in M_R. By (M.1) of Definition 16, we have that $G_i \in \sigma_0^*.1$, i.e., G_i holds in $\sigma_0^*.1$. (1)

- G_i is a negative literal, say $G_i = \neg G$. Again, since M_R is a model of $R(D)$, from (2.10) we have that $Holds(\neg G, \mathsf{S}_0)$ is true in M_R, or $Holds(G, \mathsf{S}_0)$ is false in M_R. Thus, by (M.1) of Definition 16, we have that $G \notin \sigma_0^*.1$, i.e., G_i holds in $\sigma_0^*.1$. (2)

It follows from (1) and (2) that $\sigma_0^*.1$ is an initial state of D. (3)

Consider $S \in \sigma_0^*.2$. By (M.2) of Definition 16, we conclude that there exists S such that $S = \mathsf{S}^*$ and $K(\mathsf{S}, \mathsf{S}_0)$ is true in M_R. Hence, by axiom (2.11), $\bigwedge_{i=1}^{\mathbf{m}} Holds(G_i, \mathsf{S})$ is true in M_R. Similar to (1) and (2) we can prove that S is an initial state of D. (4)

From (3) and (4) we have that σ_0^* is an initial c-state. Furthermore, axiom (2.12) and (M.2) of Definition 16 indicate that $\sigma_0^*.1 \in \sigma_0^*.2$, i.e., σ_0^* is a grounded initial c-state. Since Φ is the transition function of D and σ_0^* is an initial ground c-state, $M = (\sigma_0^*, \Phi)$ is a model of D. □

The next corollary follows immediately from Definition 16.

Corollary 1 *For each model M_R of $R(D)$, a fluent formula φ, and a situation S,*

$Holds(\varphi, \mathsf{S})$ holds in M_R iff φ holds in S^* □

Lemma 2.11 *For each model M_R of $R(D)$, a fluent F, a situation S, and an action A*

(i) $\gamma_F^+(A, \mathsf{S})$ is true in M_R iff $F \in E_A^+(\mathsf{S}^*)$; and

(ii) $\gamma_F^-(A, \mathsf{S})$ is true in M_R iff $F \in E_A^-(\mathsf{S}^*)$.

---

[9] Recall that $Holds(\neg F, \mathsf{s})$ stands for $\neg Holds(F, \mathsf{s})$

Proof. We have that

$\gamma_F^+(A, \mathsf{S})$ is true in M_R
iff $\bigvee_{\text{“}A \textbf{ causes } F \textbf{ if } \varrho\text{''} \in D} Holds(\varrho, \mathsf{S})$ is true in M_R (Definition of $\gamma_F^+(A, \mathsf{S})$)
iff there exists an ef-proposition “A **causes** F **if** ϱ” $\in D$ s.t. ϱ holds in S^* (by Corollary 1)
iff $F \in E_A^+(\mathsf{S}^*)$. Thus (i) is proved.

Similarly, we can prove (ii). □

Lemma 2.12 *For each model M_R of $R(D)$, a situation S, and an action A,*

(i) $Poss(A, \mathsf{S})$ is true in M_R iff A is executable in S^*; and

(ii) if $Poss(A, \mathsf{S})$ is true in M_R then $(\mathsf{S}')^* = Res(A, \mathsf{S}^*)$ where $\mathsf{S}' = do(A, \mathsf{S})$.

Proof. $Poss(A, \mathsf{S})$ is true in M_R
iff there exists an ex-proposition “**executable** A **if** ρ” $\in D$ s.t. $Holds(\rho, \mathsf{S})$ is true in M_R
iff there exists an ex-proposition “**executable** A **if** ρ” $\in D$ s.t. ρ holds in S^*
iff A is executable in S^*. (1)

We have that if $Poss(A, \mathsf{S})$ is true in M_R then, for a fluent F,

$F \in (\mathsf{S}')^*$
iff $Holds(F, do(A, \mathsf{S}))$ is true in M_R
iff $\gamma_F^+(A, \mathsf{S}) \vee (Holds(F, \mathsf{S}) \wedge \neg\gamma_F^-(A, \mathsf{S}))$ is true in M_R (by Axiom (2.8))
iff $F \in E_A^+(\mathsf{S}^*)$ or ($F \in \mathsf{S}^*$ and $F \notin E_A^-(\mathsf{S}^*)$) (by Lemma 2.11)
$F \in Res(A, \mathsf{S}^*)$. (2)

The lemma follows from (1) and (2). □

Lemma 2.13 *For each model M_R of $R(D)$, a situation S, and a sequence of actions α:*

(i) $Poss(\alpha, \mathsf{S})$ is true in M_R iff α is executable in S^*; and

(ii) if $Poss(\alpha, \mathsf{S})$ is true in M_R then $[do(\alpha, \mathsf{S})]^* = Res(\alpha, \mathsf{S}^*)$.

Proof. By induction over $|\alpha|$.

Base case: $|\alpha| = 0$. (i) is trivial because $Poss([], \mathsf{S})$ is true (by definition) and $[]$ is executable in every state. (ii) follows immediately from (M.1) of Definition 16. The case $|\alpha| = 1$ is proven by Lemma 2.12.

Inductive step: Assume that we have proved the lemma for $|\alpha| < k$. We need to show it for $|\alpha| = k$. Let $\alpha = [\beta; A]$. We have that $|\beta| < k$. We have that

$Poss(\alpha, \mathsf{S})$ is true in M_R
iff $Poss(\beta, \mathsf{S})$ is true in M_R and $Poss(A, do(\beta, \mathsf{S}))$ is true in M_R (by Definition)
iff $Poss(\beta, \mathsf{S})$ is true in M_R and
there exists an ex-proposition “ **executable** A **if** ρ” $\in D$ s.t. $Holds(\rho, do(\beta, \mathsf{S}))$ is true in M_R
iff β is executable in S^* (by inductive hypothesis, Item (i))
and there exists an ex-proposition “ **executable** A **if** ρ” $\in D$ such that ρ holds in $Res(\beta, \mathsf{S}^*)$
(by inductive hypothesis, Item (ii))
$[\beta; A]$ is executable in S^*. (1)

Let $\mathsf{S}' = \mathit{do}(\beta, \mathsf{S})$. By Lemma 2.12, we have that $[\mathit{do}(A, \mathsf{S}')]^* = \mathit{Res}(A, (\mathsf{S}')^*)$. By inductive hypothesis, we have that $(\mathsf{S}')^* = \mathit{Res}(\beta, \mathsf{S}^*)$. Hence, $[\mathit{do}([\beta; A], \mathsf{S})]^* = [\mathit{do}(A, \mathit{do}(\beta, \mathsf{S}))]^* = \mathit{Res}(A, (\mathsf{S}')^*) = \mathit{Res}(A, \mathit{Res}(\beta, \mathsf{S}^*)) = \mathit{Res}([\beta; A], \mathsf{S}^*)$. (2)

The inductive step follows from (1) and (2). □

Lemma 2.14 *Let D be a domain description and M_R be a model of $R(D)$. Then, there exists a model (σ_0, Φ) of D such that for any fluent formula φ and sequence of actions α of D, $M_R \models \mathit{Knows}(\varphi, \mathit{do}(\alpha, \mathsf{S}_0)) \wedge \mathit{Poss}(\alpha, \mathsf{S}_0)$ iff α is executable in σ_0 and φ is known to be true in $\hat{\Phi}(\alpha, \sigma_0)$.*

Proof. We will prove that the M_R-based model of D, $M = (\sigma_0^*, \Phi)$, satisfies the conclusion of the lemma. By Lemma 2.10, M is a model of D. By Lemma 2.13, we have that

$\mathit{Poss}(\alpha, \mathsf{S}_0)$ is true in M_R iff α is executable in $(\mathsf{S}_0)^* = \sigma_0^*.1$ iff α is executable in σ_0^*. (1)

We now prove by induction over the length of α that M satisfies the lemma and the following properties.

(i) $K(\mathsf{s}, \mathit{do}(\alpha, \mathsf{S}_0))$ is true in M_R iff $\mathsf{s}^* \in \hat{\Phi}(\alpha, \sigma_0^*).2$.

Base case: $|\alpha| = 0$. The conclusion of the lemma is trivial because of the definition of M. (i) is equivalent to

$$K(\mathsf{s}, \mathsf{S}_0) \text{ is true in } M_R \quad \text{iff} \quad \mathsf{s}^* \in \sigma_0^*.2$$

which follows immediately from Item (M.2) of Definition 16 and the fact that $K(\mathsf{S}_0, \mathsf{S}_0)$ is true in M_R. This proves the base case.

Inductive step: Assume that we have proved the lemma for $|\alpha| < l$. We need to prove it for $|\alpha| = l$. Let $\alpha = [\beta; A]$.

It follows from the construction of $R(D)$ that
$K(\mathsf{s}, \mathit{do}([\beta; A], \mathsf{S}_0))$ is true in M_R
iff $\exists \mathsf{s}_1.[(K(\mathsf{s}_1, \mathit{do}(\beta, \mathsf{S}_0)) \wedge \mathit{Poss}(A, \mathsf{s}_1) \wedge \mathsf{s} = \mathit{do}(A, \mathsf{s}_1)$
and $(\bigwedge_{j=1}^{\mathbf{n}} (A \neq K_j) \wedge \bigvee_{j=1}^{\mathbf{n}} (A = K_j \wedge$
$(\mathit{Holds}(F_j, \mathsf{s}_1) \equiv \mathit{Holds}(F_j, \mathit{do}(\beta, \mathsf{S}_0))))))]$ is true in M_R (by (2.9))
iff $\mathsf{s}_1^* \in \hat{\Phi}(\beta, \sigma_0^*).2$ (by inductive hypothesis)
and $\mathsf{s}^* = \mathit{Res}(A, \mathsf{s}_1^*)$ (by $\mathsf{s} = \mathit{do}(A, \mathsf{s}_1)$ and (M.1) of Definition 16)
and if $A_l = K_j$ then $F_j \in \mathsf{s}_1^*$ iff $F_j \in \hat{\Phi}(\beta, \sigma_0^*).1$
iff $\mathsf{s}^* \in \hat{\Phi}(\beta, \sigma_0^*).2$ (2)

Consider a fluent formula φ, we have that

$M_R \models \mathit{Knows}(\varphi, \mathit{do}(\alpha, \mathsf{S}_0)) \wedge \mathit{Poss}(\alpha, \mathsf{S}_0)$
iff $\mathit{Poss}(\alpha, \mathsf{S}_0)$ is true in M_R and $\forall \mathsf{s}.[K(\mathsf{s}, \mathit{do}(\alpha, \mathsf{S}_0)) \supset \mathit{Holds}(F, \mathsf{s})]$ is true in M_R
iff α is executable in σ_0^* and $\forall \mathsf{s}^* \in \hat{\Phi}(\alpha, \sigma_0^*).2$, φ holds in s^* (by (1) and (i))
iff α is executable in σ_0^* and φ is known to be true in $\hat{\Phi}(\alpha, \sigma_0^*)$. (3)

(2)-(3) prove the inductive step for (i) and the lemma's conclusion. The lemma is proved. □

We now prove Proposition 2.

Proposition 2 *Let D be a domain description, φ be a fluent formula, and α be a sequence of actions of D. Then,*

$$D \models_{\mathcal{A}_K} \textbf{Knows } \varphi \textbf{ after } \alpha \quad \textit{iff} \quad R(D) \models \mathit{Knows}(\varphi, \mathit{do}(\alpha, \mathsf{S}_0)) \wedge \mathit{Poss}(\alpha, \mathsf{S}_0).$$

Proof.

1. Assume that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α. We will prove that $R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. Assume the contrary, $R(D) \not\models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. By definition, there exists a model M_R of $R(D)$ such that $M_R \not\models Knows(\varphi, do(\alpha, \mathsf{S}_0))$ or $M_R \not\models Poss(\alpha, \mathsf{S}_0)$. Then, by Lemma 2.14, there exists a model M of D such that $M \not\models$ **Knows** φ **after** α. This implies that $D \not\models_{\mathcal{A}_K}$ **Knows** φ **after** α which contradicts with the assumption that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α. Hence, our assumption is incorrect, i.e., we have proved that $R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. Therefore, we can conclude that

   if $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α then $R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. (1)

2. Assume that $R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. We will prove that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α. Assume the contrary, $D \not\models_{\mathcal{A}_K}$ **Knows** φ **after** α. This means that there exists a model M of D such that $M \not\models$ **Knows** φ **after** α. Then, by Lemma 2.9, there exists a model M_R of $R(D)$ such that $M_R \not\models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. This implies that $R(D) \not\models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$ which contradicts our assumption. Hence, we have that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α. So,

   if $R(D) \models Knows(\varphi, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$ then $D \models_{\mathcal{A}_K}$ **Knows** φ **after** α. (2)

From (1) and (2), we can conclude that
$D \models_{\mathcal{A}_K}$ **Knows** φ **after** α iff $R(D) \models Knows(F, do(\alpha, \mathsf{S}_0)) \wedge Poss(\alpha, \mathsf{S}_0)$. □

We will now extend the Lemmas 2.9 and 2.14 to conditional plans. We need the following notation and lemmas.

Let c be a conditional plan, we define the number of case plans of c, denoted by $count(c)$, inductively as follows.

1. If $c = []$, then $count(c) = 0$.
2. If $c = a$, a is an action, then $count(c) = 0$
3. If c_1 and c_2 are conditional plans, then $case(c_1; c_2) = count(c_1) + count(c_2)$
4. If c is a case plan of the form

   Case
   $\varphi_1 \rightarrow c_1$
   $\dots$
   $\varphi_n \rightarrow c_n$
   Endcase

   then $count(c) = 1 + \Sigma_{i=1}^{n} count(c_i)$.

It follows directly from the definition of B_{Apply} the following lemma.

Lemma 2.15 *Let* s, s' *be situations and* c *be a conditional plan. The following formulas are entailed by* B_{Apply}*:*

(i) $Apply([], \mathsf{s}, \mathsf{s}') \supset \mathsf{s} = \mathsf{s}'$;

(ii) $Apply([case([])|c], \mathsf{s}, \mathsf{s}') \supset \mathsf{s}' = \perp$*; and*

(iii) $Apply(c, \perp, \mathsf{s}') \supset \mathsf{s}' = \perp$.

Proof. Assume that (i) is not entailed by B_{Apply}. Then, there exists a model M of B_{Apply} such that $Apply([], \mathsf{s}, \mathsf{s}')$ is true in M but $\mathsf{s} \neq \mathsf{s}'$. It is easy to see that $M \setminus Apply([], \mathsf{s}, \mathsf{s}')$ is also model of B_{Apply}. This violates the minimality of M. Thus, (i) is true in every model of B_{Apply}. Similarly, we can prove (ii) and (iii). □

Lemma 2.16 *Let* $\mathsf{s}, \mathsf{s}', \mathsf{s}''$ *be situations,* a *be an action,* φ *be a fluent formula,* α *be a sequence of actions, and* c, c', c'' *be conditional plans. The following formulas are entailed by* B_{Apply}*:*

(i) $Apply([a|\alpha], \mathsf{s}, \mathsf{s}') \wedge \mathsf{s} \neq \perp \supset ((Poss(a, \mathsf{s}) \supset Apply(\alpha, do(a, \mathsf{s}), \mathsf{s}')) \wedge (\neg Poss(a, \mathsf{s}) \supset \mathsf{s}' = \perp))$;

(ii) $Apply([case([(\varphi, c)|r'])|c''], \mathsf{s}, \mathsf{s}') \wedge \mathsf{s} \neq \perp \supset$
$((knows(\varphi, \mathsf{s}) \supset \exists \mathsf{s}''.Apply(c, \mathsf{s}, \mathsf{s}'') \wedge Apply(c'', \mathsf{s}'', \mathsf{s}')) \wedge$
$(\neg knows(\varphi, \mathsf{s}) \supset Apply([case(r')|c''], \mathsf{s}, \mathsf{s}')))$*; and*

(iii) $Apply(c, \mathsf{s}, \mathsf{s}') \wedge Apply(c, \mathsf{s}, \mathsf{s}'') \supset \mathsf{s}' = \mathsf{s}''$.

Proof. Assume that (i) is not entailed by B_{Apply}. It means that there exists a model M of B_{Apply}, an action a, a sequence of actions α, and two situations s and s' such that $Apply([a|\alpha], \mathsf{s}, \mathsf{s}') \wedge \mathsf{s} \neq \perp$ is true in M and $(Poss(a, \mathsf{s}) \supset Apply([a|\alpha], do(a, \mathsf{s}), \mathsf{s}')) \wedge (\neg Poss(a, \mathsf{s}) \supset \mathsf{s}' = \perp)$ is not true in M. By definition of B_{Apply}, the model $M' = M \setminus \{Apply([a|\alpha], \mathsf{s}, \mathsf{s}')\}$ is a model of B_{Apply}. This contradicts the assumption that M is a minimal model of B_{Apply}. Hence, our assumption that (i) is not true in M is incorrect, i.e., we have proved that (i) is a valid sentence of B_{Apply}.

Similarly, we can prove Item (ii). The proof of Item (iii) is based on induction over $count(c)$ and is omitted here. □

Lemma 2.17 *Let* $c_1, \ldots, c_n$ *be* n *arbitrary conditional plans,* $(n \geq 1)$*. Then, the following formula is entailed by* B_{Apply}*:*
$Apply([c_1; \ldots; c_n], \mathsf{s}_1, \mathsf{s}_{n+1}) \equiv \exists(\mathsf{s}_2, \ldots, \mathsf{s}_n).[Apply(c_1, \mathsf{s}_1, \mathsf{s}_2) \wedge \ldots \wedge Apply(c_n, \mathsf{s}_n, \mathsf{s}_{n+1})]$.

Proof. We prove the lemma by induction over n.

Base case: $n = 1$. Then, we have that the right hand side is $\exists \mathsf{s}_2.Apply(c_1, \mathsf{s}_1, \mathsf{s}_2)$ and the left hand side is $Apply(c_1, \mathsf{s}_1, \mathsf{s}_2)$. It follows from Item (iii) of Lemma 2.16 that $Apply(c_1, \mathsf{s}_1, \mathsf{s}_2) \equiv \exists(\mathsf{s}_2).Apply(c_1, \mathsf{s}_1, \mathsf{s}_2)$. This proves the base case.

Inductive step: Assume that we have proved the lemma for n. We need to prove it for $n + 1$. Since c_n and c_{n+1} are conditional plans, by definition, we have that $c = c_n; c_{n+1}$ is a conditional plan. Hence, by inductive hypothesis for n plans $c_1, \ldots, c_{n-1}, c$, we have that

$$Apply([c_1; \ldots; c_{n-1}; c], \mathsf{s}_1, \mathsf{s}_{n+2}) \equiv \exists(\mathsf{s}_2, \ldots, \mathsf{s}_n).[Apply(c_1, \mathsf{s}_1, \mathsf{s}_2) \wedge \ldots \wedge Apply(c, \mathsf{s}_n, \mathsf{s}_{n+2})]. \quad (1)$$

By inductive hypothesis for 2 plans c_n and c_{n+1}, we have that

$$Apply(c, \mathsf{s}_n, \mathsf{s}_{n+2}) \equiv \exists(\mathsf{s}_{n+1}).[Apply(c_n, \mathsf{s}_n, \mathsf{s}_{n+1}) \wedge Apply(c_{n+1}, \mathsf{s}_{n+1}, \mathsf{s}_{n+2})]. \quad (2)$$

The inductive step follows from (1) and (2). I.e., the lemma is proved. □

Lemma 2.18 *Let c be a case plan of the form*

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

and $\mathsf{s} \neq \perp$. Then, the following formula is entailed by B_{Apply}:

$Knows(\varphi_j, \mathsf{s}) \wedge Apply(c, \mathsf{s}, \mathsf{s}') \equiv Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}')$.

Proof. Let M be a model of B_{Apply}. Obviously, if $Knows(\varphi_j, \mathsf{s})$ is false in M for $1 \leq j \leq l$, the formula is true in M. So, we need to prove it for the case there exists some j, $1 \leq j \leq l$, $Knows(\varphi_j, \mathsf{s})$ is true in M. We consider two cases:

(a) **Left to Right:** Assume that $Knows(\varphi_j, \mathsf{s}) \wedge Apply(c, \mathsf{s}, \mathsf{s}')$ is true in M. Then, since φ_j's are mutual exclusive, we can conclude that $\neg Knows(\varphi_i, \mathsf{s})$ is true in M, for $i \neq j$, $1 \leq i \leq l$. Hence, by Item (ii) of Lemma 2.16 (for $c'' = [], \varphi = \varphi_j, c = p_j$) we have that

$$\exists \mathsf{s}''.Apply(p_j, \mathsf{s}, \mathsf{s}'') \wedge Apply([], \mathsf{s}'', \mathsf{s}') \text{ is true in } M. \tag{1}$$

From Item (i) of Lemma 2.15, we have that $\mathsf{s}'' = \mathsf{s}'$. Hence, (1) is equivalent to,

$$Apply(p_j, \mathsf{s}, \mathsf{s}') \text{ is true in } M. \tag{2}$$

It follows from the assumption that $Knows(\varphi_j, \mathsf{s})$ is true in M and (2) that $Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}')$ is true in M, which proves (a).

(b) **Right to Left:** Assume that $Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}')$ holds in M. Similar argument as above concludes that $\neg Knows(\varphi_i, \mathsf{s})$ is true in M, for $i \neq j$, $1 \leq i \leq l$. Hence, by definition of *Apply* (case 6, for $c'' = []$, r' is the sequence $[(\varphi_1, p_1), \ldots, (\varphi_{j-1}, p_{j-1}), (\varphi_{j+1}, p_{j+1}), \ldots, (\varphi_l, p_l)]$), we have that

$Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}') \wedge Apply([], \mathsf{s}', \mathsf{s}'') \supset Apply([case([(\varphi_j, p_j)|r'], \mathsf{s}, \mathsf{s}'')$ holds in M.

Furthermore, from (i) of Lemma 2.15, we have that $\mathsf{s}'' = \mathsf{s}'$. Hence, we conclude that $Apply(c, \mathsf{s}, \mathsf{s}') \wedge Knows(\varphi_j, \mathsf{s})$ holds in M. This proves (b).

The lemma follows from (a) and (b). □

Lemma 2.19 *Assume that $c = c_1, \ldots, c_n$ is a conditional plan where $c_1, \ldots, c_n$ is a sequence of conditional plans satisfying the conditions of Observation 3.1. Let c_1 be a case plan of the form*

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

and s be a situation term. Let M be a model of B_{Apply} such that $M \models Knows(\varphi_j, \mathsf{s})$ for some j, $1 \leq j \leq l$. Then, $M \cup B_{\text{Apply}} \models Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c', \mathsf{s}, \mathsf{s}')$ where $c' = p_j; c_2; \ldots; c_n$.

Proof. By Lemma 2.17, there exists $\mathsf{s}_1, \ldots, \mathsf{s}_{n-1}$ such that
$Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c_1, \mathsf{s}, \mathsf{s}_1) \wedge \ldots \wedge Apply(c_n, \mathsf{s}_{n-1}, \mathsf{s}')$ is true in $M \cup B_{\text{Apply}}$ (1)

Since $M \models Knows(\varphi_j, \mathsf{s})$, by Lemma 2.18 and from (1), we have that

$Knows(\varphi_j, \mathsf{s}) \wedge Apply(c_1, \mathsf{s}, \mathsf{s}_1) \equiv Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}_1)$ is true in $M \cup B_{\text{Apply}}$ (2)

It follows from (1) and (2) that

$M \cup B_{\text{Apply}} \models Knows(\varphi_j, \mathsf{s}) \wedge Apply(c, \mathsf{s}, \mathsf{s}') \equiv Knows(\varphi_j, \mathsf{s}) \wedge Apply(p_j, \mathsf{s}, \mathsf{s}_1) \wedge Apply(c_2, \mathsf{s}_1, \mathsf{s}_3) \wedge \ldots \wedge Apply(c_n, \mathsf{s}_{n-1}, \mathsf{s}')$

which implies that

$M \cup B_{\text{Apply}} \models Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c', \mathsf{s}, \mathsf{s}')$ (By Lemma 2.17) (3)

The lemma follows from (3). □

Lemma 2.20 *Assume that $c = c_1, \ldots, c_n$ is a conditional plan where $c_1, \ldots, c_n$ is a sequence of conditional plans satisfying the conditions of Observation 3.1. Let c_1 be a sequence of actions and c_2 be a case plan of the form*

Case
  $\varphi_1 \to p_1$
  $\vdots$
  $\varphi_l \to p_l$
Endcase

and s be a situation term. Let M be a model of B_{Apply} such that $M \models Knows(\varphi_j, do(c_1, \mathsf{s}))$ for some j, $1 \leq j \leq l$. Then, $M \cup B_{\text{Apply}} \models Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c', \mathsf{s}, \mathsf{s}')$ where $c' = c_1; p_j; \ldots; c_n$.

Proof. Since $M \models Knows(\varphi_j, do(c_1, \mathsf{s}))$, we conclude that $Poss(c_1, \mathsf{s})$ is true in M. By Lemma 2.17, there exists $\mathsf{s}_1, \ldots, \mathsf{s}_{n-1}$ such that $Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c_1, \mathsf{s}, \mathsf{s}_1) \wedge \ldots \wedge Apply(c_n, \mathsf{s}_{n-1}, \mathsf{s}')$ holds in $M \cup B_{\text{Apply}}$ (1)

Since c_1 is a sequence of actions, we have that $\mathsf{s}_1 = do(c_1, \mathsf{s})$. Therefore, from $M \models Knows(\varphi_j, do(c_1, \mathsf{s}))$, $\mathsf{s}_1 = do(c_1, \mathsf{s})$, and by Lemma 2.18, we have that

$M \cup B_{\text{Apply}} \models Apply(c_2, \mathsf{s}_1, \mathsf{s}_2) \equiv Apply(p_j, \mathsf{s}_1, \mathsf{s}_2).$ (2)

It follows from (1) and (2) that

$M \cup B_{\text{Apply}} \models Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c_1, \mathsf{s}, \mathsf{s}_1) \wedge Apply(p_j, \mathsf{s}_1, \mathsf{s}_2) \wedge Apply(c_3, \mathsf{s}_2, \mathsf{s}_3) \wedge \ldots \wedge Apply(c_n, \mathsf{s}_{n-1}, \mathsf{s}')$

which implies that

$M \cup B_{\text{Apply}} \models Apply(c, \mathsf{s}, \mathsf{s}') \equiv Apply(c', \mathsf{s}, \mathsf{s}')$ (By Lemma 2.17) (3)

The lemma follows from (3). □

Lemma 2.21 *Let D be a domain description and $M = (\sigma_0, \Phi)$ be a model of D. Then, there exists a model M_R of $R(D)$ such that for any fluent formula φ and conditional plan c,*

- *c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ iff $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$;*
- *$\hat{\Phi}(c, \sigma_0) = \perp$ iff $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \perp)$.*

Proof. From the observation 3.1, we can assume that $c = c_1; \ldots ; c_n$ where c_i is a sequence of actions or a case plan and for every i, $1 \leq i \leq n-1$, if c_i is a sequence of actions then c_{i+1} is a case plan.

Let M_R be the M-interpretation of D. By Lemma 2.2, M_R is model of $R(D)$. We will prove by induction over $count(c)$ that M_R satisfies the lemma.

Base case: $count(c) = 0$. Using Items (i) and (iii) of Lemma 2.16, we can prove that
$M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \supset (Poss(c, \mathsf{S}_0) \supset \mathsf{s} = do(c, \mathsf{S}_0)) \wedge (\neg Poss(c, \mathsf{S}_0) \supset \mathsf{s} = \perp)$. (1)

By Lemma 2.9, we have that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ iff $M_R \models Knows(\varphi, do(c, \mathsf{S}_0)) \wedge Poss(c, \mathsf{S}_0)$. (2)

It follows from (1) and (2) that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ iff
$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. (3)

(3) proves the first item of the lemma. To complete the base case, we need to prove the second item. Since c is a sequence of actions, we have that $\hat{\Phi}(c, \sigma_0) = \perp$
iff c is not executable in σ_0
iff $M_R \models \neg Poss(c, \mathsf{S}_0)$ (Lemma 2.9)
iff $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \perp)$ (by (1)).
So, the second item of the lemma is proved. The base case is proved.

Inductive step: Assume that we have proved the lemma for $count(c) \leq k$. We need to prove the lemma for $count(c) = k + 1$.

Case 1: c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$. We will show that $Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ is true in $M_R \cup B_{\text{Apply}}$.

We consider two cases:

Case 1.1: c_1 is a case plan. Assume that c_1 is the following case plan

Case
  $\varphi_1 \rightarrow p_1$
  $\vdots$
  $\varphi_l \rightarrow p_l$
Endcase

Since c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ we have that $\perp \neq \hat{\Phi}(c, \sigma_0)$. It implies that there exists j, $1 \leq j \leq l$, such that φ_j is known to be true in σ_0. (4)

Let $c' = p_j; c_2; \ldots ; c_n$. Then, by definition of $\hat{\Phi}$ and from (4) we have that $\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0)$. Hence, φ is known to be true in $\hat{\Phi}(c', \sigma_0)$. Since $count(c') \leq count(c) - 1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis, we can conclude that

$Knows(\varphi, \mathsf{s}) \wedge Apply(c', \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ is true in $M_R \cup B_{\text{Apply}}$. (5)

It follows from Lemma 2.19 that $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$. (6)

From (5) and (6) we have that $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$.

Case 1.2: c_1 is a sequence of actions. Then, c_2 is a case plan. Let us assume that c_2 is the case plan.

Case
  $\varphi_1 \to p_1$
  $\vdots$
  $\varphi_l \to p_l$
Endcase

Since c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ we have that $\perp \neq \hat{\Phi}(c, \sigma_0)$. This implies that there exists j, $1 \leq j \leq l$, such that φ_j is known to be true in $\hat{\Phi}(c_1, \sigma_0)$. (7)

Let $c' = c_1; p_j; c_3; \ldots; c_n$. From (7) and the definition of $\hat{\Phi}$, we have that $\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0)$. This implies that φ is known to be true in $\hat{\Phi}(c', \sigma_0)$. (8)

Since $count(c') \leq count(c) - 1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis and (8), we conclude that

$$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c', \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp. \tag{9}$$

From Lemma 2.20, we have that

$$M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s}). \tag{10}$$

(9) and (10) prove that $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$.

The above two cases prove that if c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ then $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$.

Case 2: $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge s \neq \perp$. We will prove that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$. We consider two cases:

Case 2.1: c_1 is a case plan. Assume that c_1 is the following case plan

Case
  $\varphi_1 \to p_1$
  $\vdots$
  $\varphi_l \to p_l$
Endcase

Since $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$, by Items (ii) of Lemma 2.16, we conclude that there exists j, $1 \leq j \leq l$, such that $M_R \models Knows(\varphi_j, \mathsf{S}_0)$. (11)

Let $c' = p_j; c_2; \ldots; c_n$. By Lemma 2.19 we have that $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$. This implies that

$$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c', \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp. \tag{12}$$

Furthermore, from the definition of M_R and (11), we have that φ_j is known to be true in σ_0. This implies that

$$\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0). \tag{13}$$

Since $count(c') \leq count(c)-1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis and (12), we can conclude that

c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c', \sigma_0)$, and from (13), we have that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

Case 2.2: c_1 is a sequence of actions. Then, c_2 is a case plan. Let us assume that c_2 is the case plan.

Case
  $\varphi_1 \rightarrow p_1$
  $\vdots$
  $\varphi_l \rightarrow p_l$
Endcase

Similar to Case 2.1, we conclude that there exists j, $1 \leq j \leq l$, such that
$M_R \cup B_{\text{Apply}} \models Knows(\varphi_j, \mathsf{s}') \wedge Apply(c_1, \mathsf{S}_0, \mathsf{s}') \wedge \mathsf{s}' \neq \perp$. (14)

Let $c' = c_1; p_j; c_3 \ldots; c_n$. From (14) and Lemma 2.20, we have that $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$. (15)

By inductive hypothesis, we have that c' is executable in σ_0 and φ_j is known to be true in $\hat{\Phi}(c_1, \sigma_0)$. Hence,
$\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0)$. (16)

Since $count(c') \leq count(c) - 1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis, we can conclude that

φ is known to be true in $\hat{\Phi}(c', \sigma_0)$. This, together with (16), proves that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

The two cases 2.1 and 2.2 prove that if $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ then c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

The two cases 1 and 2 prove that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ iff $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. (17)

The proof of the inductive step for the last item of the lemma, $\hat{\Phi}(c, \sigma_0) = \perp$ iff $Apply(c, \mathsf{S}_0, \perp)$ is true in $M_R \cup B_{\text{Apply}}$ has also four cases similar to the cases (1.1)-(1.2) and (2.1)-(2.2). We will show next the first case. The other cases are similar and are omitted here.

Assume that $\hat{\Phi}(c, \sigma_0) = \perp$ where $c = c_1, \ldots, c_n$ and c_1 is the following case plan

Case
  $\varphi_1 \rightarrow p_1$
  $\vdots$
  $\varphi_l \rightarrow p_l$
Endcase

We will show that $Apply(c, \mathsf{S}_0, \perp)$ is true in $M_R \cup B_{\text{Apply}}$. We consider two cases:

1. there exists no j such that φ_j is known to be true in σ_0. By Lemma 2.9, we have that $\neg Knows(\varphi_j, \mathsf{S}_0)$ is true in M_R for $1 \leq j \leq l$. Applies the last definition of *Apply* l times, we have that

   $Apply(c, \mathsf{S}_0, \mathsf{S}') \supset Apply([case([])|c'], \mathsf{S}_0, \mathsf{S}')$ is true in B_{Apply} where $c' = c_2, \ldots, c_n$. By the third item in the definition, we can then conclude that $Apply(c, \mathsf{S}_0, \mathsf{S}') \supset Apply(c, \mathsf{S}_0, \bot)$ is true in $M_R \cup B_{\text{Apply}}$, i.e., the inductive step is proved.

2. there exists some j such that φ_j is known to be true in σ_0. Again, by Lemma 2.9, we have that $Knows(\varphi_j, \mathsf{S}_0)$ is true in M_R and $Knows(\varphi_i, \mathsf{S}_0)$ is false in M_R for $1 \leq i \neq j \leq l$. Then, by Lemma 2.19, we have that

   $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$.

   where $c' = p_j; c_2; \ldots; c_n$. This, together with the fact that $\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0) = \bot$ and the inductive hypothesis implies that $Apply(c, \mathsf{S}_0, \bot)$ is true in $M_R \cup B_{\text{Apply}}$

The above two cases prove the inductive step for the second item of the lemma. (18)

The lemma follows from (17) and (18). □

Lemma 2.22 *Let D be a domain description and M_R be a model of $R(D)$. Then, there exists a model $M = (\sigma_0, \Phi)$ of D such that for any fluent formula φ and any conditional plan c,*

- $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s})) \wedge \mathsf{s} \neq \bot$ *iff c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$;*
- $\hat{\Phi}(c, \sigma_0) = \bot$ *iff* $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \bot)$.

Proof. From the observation 3.1, we can assume that $c = c_1; \ldots; c_n$ where c_i is a sequence of actions or a case plan and for every i, $1 \leq i \leq n-1$, if c_i is a sequence of actions then c_{i+1} is a case plan.

Let $M = (\sigma_0, \Phi)$ be the M_R-based model of D. By Lemma 2.10, M is a model of D. We will prove by induction over the number of case plan in c, *count*(c), that M satisfies the lemma.

Base case: *count*$(c) = 0$. Using Item (i) of the Lemma 2.16, we can prove that
$M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \supset (Poss(c, \mathsf{S}_0) \supset \mathsf{s} = do(c, \mathsf{S}_0)) \wedge (\neg Poss(c, \mathsf{S}_0) \supset \mathsf{s} = \bot)$. (1)

By Lemma 2.14, we have that

$M_R \models Knows(\varphi, do(c, \mathsf{S}_0)) \wedge Poss(c, \mathsf{S}_0)$ iff c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$. This, together with (1), proves that

$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \bot$ iff c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$. (2)

Furthermore, since c is a sequence of actions, from (1), $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \bot)$ iff $\neg Poss(c, \mathsf{S}_0)$ is true in M_R. Again, by Lemma 2.14, this is equivalent to c is not executable in σ_0. The base case for the third item of the lemma is proved. (3)

The base case of the lemma follows from (2)-(3).

Inductive step: Assume that we have proved the lemma for *count*$(c) \leq k$. We need to prove the lemma for *count*$(c) = k + 1$.

Case 1: c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c,\sigma_0)$. We will show that $M_R \cup B_{\text{Apply}} \models \mathit{Knows}(\varphi,\mathsf{s}) \wedge \mathit{Apply}(c,\mathsf{S}_0,\mathsf{s}) \wedge \mathsf{s} \neq \perp$. We consider two cases:

Case 1.1: c_1 is a case plan. Assume that c_1 is the following case plan

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

Since c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c,\sigma_0)$ we have that $\perp \neq \hat{\Phi}(c,\sigma_0)$. This implies that there exists j, $1 \le j \le l$, such that φ_j is known to be true in σ_0. (4)

Let $c' = p_j; c_2; \ldots; c_n$. Then, by definition of $\hat{\Phi}$ and from (4) we have that $\hat{\Phi}(c,\sigma_0) = \hat{\Phi}(c',\sigma_0)$. This implies that

φ is known to be true in $\hat{\Phi}(c',\sigma_0)$. (5)

From(4) and Lemma 2.19, we have that

$$M_R \cup B_{\text{Apply}} \models \mathit{Apply}(c,\mathsf{S}_0,\mathsf{s}) \equiv \mathit{Apply}(c',\mathsf{S}_0,\mathsf{s}). \tag{6}$$

Since $\mathit{count}(c') \le \mathit{count}(c)-1$, we have that $\mathit{count}(c') \le k$. Thus, by inductive hypothesis and (5), we can conclude that

$M_R \cup B_{\text{Apply}} \models \mathit{Knows}(\varphi,\mathsf{s}) \wedge \mathit{Apply}(c',\mathsf{S}_0,\mathsf{s}) \wedge \mathsf{s} \neq \perp$. This, together with (6), proves that $M_R \cup B_{\text{Apply}} \models \mathit{Knows}(\varphi,\mathsf{s}) \wedge \mathit{Apply}(c,\mathsf{S}_0,\mathsf{s}) \wedge \mathsf{s} \neq \perp$.

Case 1.2: c_1 is a sequence of actions. Then, c_2 is a case plan. Again, let us assume that c_2 is the case plan.

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

Since c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c,\sigma_0)$ we have that $\perp \neq \hat{\Phi}(c,\sigma_0)$. It implies that there exists j, $1 \le j \le l$, such that φ_j is known to be true in $\hat{\Phi}(c_1,\sigma_0)$. (7)

Let $c' = c_1; p_j; c_3; \ldots; c_n$. From (7), we have that $\hat{\Phi}(c,\sigma_0) = \hat{\Phi}(c',\sigma_0)$. (8)

It follows from (8) and Lemma 2.20 that

$$M_R \cup B_{\text{Apply}} \models \mathit{Apply}(c,\mathsf{S}_0,\mathsf{s}) \equiv \mathit{Apply}(c',\mathsf{S}_0,\mathsf{s}). \tag{9}$$

Since $\mathit{count}(c') \le \mathit{count}(c)-1$, we have that $\mathit{count}(c') \le k$. Thus, by inductive hypothesis and (8), we can conclude that

$M_R \cup B_{\text{Apply}} \models \mathit{Knows}(\varphi,\mathsf{s}) \wedge \mathit{Apply}(c',\mathsf{S}_0,\mathsf{s}) \wedge \mathsf{s} \neq \perp$. This, together with (9), proves that $M_R \cup B_{\text{Apply}} \models \mathit{Knows}(\varphi,\mathsf{s}) \wedge \mathit{Apply}(c,\mathsf{S}_0,\mathsf{s}) \wedge \mathsf{s} \neq \perp$.

From the two cases 1.1 and 1.2, we can conclude that if c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ then $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$.

Case 2: $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. We will prove that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$. We consider two cases:

Case 2.1: c_1 is a case plan. Assume that c_1 is the following case plan

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

Since $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$, using of Lemma (2.16), we can conclude that there exists j, $1 \leq j \leq l$, such that $M_R \models Knows(\varphi_j, \mathsf{S}_0)$. (10)

Let $c' = p_j; c_2; \ldots; c_n$. Then, by Lemma 2.19 and (10), we conclude that

$M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$. Together with the assumption that $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$, we have that

$$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c', \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp. \tag{11}$$

Furthermore, from the definition of M and (10), we have that φ_j is known to be true in σ_0. This implies that

$$\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0). \tag{12}$$

Since $count(c') \leq count(c) - 1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis and (11), we can conclude that

c' is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c', \sigma_0)$, and from (12), we can conclude that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

Case 2.2: c_1 is a sequence of actions. Then, c_2 is a case plan. Again, let us assume that c_2 is the case plan.

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

Similar to Case 2.1, we conclude that there exists j, $1 \leq j \leq l$, such that $M_R \cup B_{\text{Apply}} \models Knows(\varphi_j, \mathsf{s}') \wedge Apply(c_1, \mathsf{S}_0, \mathsf{s}')$. (13)

Let $c' = c_1; p_j; c_3; \ldots; c_n$. From (13) and Lemmas 2.20, we have that $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \equiv Apply(c', \mathsf{S}_0, \mathsf{s})$. Hence

$$M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c', \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp. \tag{14}$$

Since c_1 is a sequence of actions, we have that φ_j is known to be true in $\hat{\Phi}(c_1, \sigma_0)$. Hence, $\hat{\Phi}(c, \sigma_0) = \hat{\Phi}(c', \sigma_0)$. (15)

Since $count(c') \leq count(c)-1$, we have that $count(c') \leq k$. Thus, by inductive hypothesis and (14), we conclude that

c' is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c', \sigma_0)$. This, together with (15), proves that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

From the two cases 2.1 and 2.2, we conclude that if $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ then c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$.

The two cases 1 and 2 show that c is executable in σ_0 and φ is known to be true in $\hat{\Phi}(c, \sigma_0)$ iff $M_R \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. (16)

The proof of the third item of the lemma, i.e., $\hat{\Phi}(c, \sigma_0) = \perp$ iff $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \perp)$ is similar to the proof of the second item of Lemma 2.21 and is omitted here. This, together with (16), proves the inductive step of the lemma, and hence, proves the lemma. □

Proposition 3 Let D be a domain description and $R(D)$ be the corresponding first order theory. Let c be a conditional plan and φ be a fluent formula. Then,

$$D \models_{\mathcal{A}_K} \textbf{Knows } \varphi \textbf{ after } c \quad \text{iff} \quad R(D) \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge Knows(\varphi, \mathsf{s}) \wedge \mathsf{s} \neq \perp .$$

Proof.

1. Assume that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. We will prove that $R(D) \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. Assume the contrary, $R(D) \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. By definition, there exists a model M_R of $R(D)$ such that $M_R \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. There are two possibilities

   (a) $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \perp)$. Hence, by Lemma 2.22, there exists a model $M = (\sigma_0, \Phi)$ of D such that $\hat{\Phi}(c, \sigma_0) = \perp$. This implies that $D \not\models_{\mathcal{A}_K}$ **Knows** φ **after** c. Hence, this case cannot happen. (1)

   (b) $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ and $M_R \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s})$. Then, by Lemma 2.22, there exists a model M of D such that $M \not\models$ **Knows** φ **after** c. This implies that $D \not\models_{\mathcal{A}_K}$ **Knows** φ **after** c. This contradicts with the assumption that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. Hence, this case cannot happen too. (2)

   From (1) and (2), we conclude that our assumption is incorrect, i.e., we have proved that $R(D) \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. Therefore, we have that

   if $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c then $R(D) \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. (3)

2. Assume that $R(D) \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. We will prove that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. Assume the contrary, $D \not\models_{\mathcal{A}_K}$ **Knows** φ **after** c. This means that there exists a model $M = (\sigma_0, \Phi)$ of D such that $M \not\models$ **Knows** φ **after** c. There are two sub-cases:

   (a) $\hat{\Phi}(c, \sigma_0) = \perp$. Then, by Lemma 2.21, there exists a model M_R of $R(D)$ such that $M_R \cup B_{\text{Apply}} \models Apply(c, \mathsf{S}_0, \perp)$. This implies that $M_R \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$, which contradicts with our assumption. Therefore, this case cannot happen. (4)

(b) $\hat{\Phi}(c, \sigma_0) \neq \perp$. Then, F is not known to be true in $\hat{\Phi}(c, \sigma_0)$. Then, by Lemma 2.21, there exists a model M_R of $R(D)$ such that $M_R \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. This implies that $R(D) \cup B_{\text{Apply}} \not\models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$, which contradicts our assumption. Hence, this case cannot happen too. (5)

From (4) and (5, we have that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. Hence, we have that

if $R(D) \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$ then $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. (6)

From (3) and (6), we can conclude that
$D \models_{\mathcal{A}_K}$ **Knows** φ **after** c iff $R(D) \cup B_{\text{Apply}} \models Knows(\varphi, \mathsf{s}) \wedge Apply(c, \mathsf{S}_0, \mathsf{s}) \wedge \mathsf{s} \neq \perp$. □

Appendix C - Soundness of ω-Approximation

In this section we prove the soundness of the ω-Approximation w.r.t. the semantics of $\mathcal{A}_K$. Throughout the section, by D we denote an arbitrary but fixed domain description. We will need the following notations and lemmas.

Let $\sigma = \langle T, F \rangle$ be an a-state and $\delta = \langle u, \Sigma \rangle$ be a c-state. We say σ *agrees with* δ if for every state $s \in \Sigma$, $T \subseteq s$ and $F \cap s = \emptyset$.

For an a-state $\sigma = \langle T, F \rangle$, by $true(\sigma)$ and $false(\sigma)$ we denote the set T and F respectively.

For a sequence of actions $\alpha = a_1, \ldots, a_n$ $(n \geq 1)$ and a state s, by $Res(\alpha, s)$ we denote the state $Res(a_n, \ldots, Res(a_1, s))$. Similarly $Res_0(\alpha, \sigma)$ denotes the state $Res_0(a_n, \ldots, Res_0(a_1, \sigma))$ where σ is an a-state and $\alpha = a_1, \ldots, a_n$.

The following observations are trivial and will be used in the proofs in this section.

Observation 3.2 *Let $\sigma = \langle T, F \rangle$ be an a-state and $\delta = \langle u, \Sigma \rangle$ be a grounded c-state such that σ agrees with δ, then*

1. *if φ is a fluent formula and φ holds in σ ($\sigma \models \varphi$), then for every $s \in \Sigma$, φ holds in s;*
2. *for every action a, $\Phi(a, \delta)$ is a grounded c-state;*
3. *$\Sigma \subseteq \{true(\sigma') \mid \sigma' \in Comp(\sigma)\}$;*
4. *if α is a sequence of non-sensing actions and σ is complete, $true(Res_0(\alpha, \sigma)) = Res(\alpha, true(\sigma))$;*
5. *If a sequence of non-sensing actions, α, is ω-executable in σ then α is executable in δ.*

The proof of Proposition 6 is based on the following lemmas.

Lemma 3.23 *Let D be a domain description, σ be an a-state, and δ be a grounded c-state of D such that σ agrees with δ. Then, for every sequence of non-sensing actions α of D, $Res_\omega(\alpha, \sigma)$ agrees with $\hat{\Phi}(\alpha, \delta)$.*

Proof. Assume that $\sigma = \langle T, F \rangle$, $\delta = \langle s, \Sigma \rangle$, and $\hat{\Phi}(\alpha, \delta) = \langle \hat{s}, \hat{\Sigma} \rangle$.

Let $f \in true(Res_\omega(\alpha, \sigma))$.
$\Rightarrow f \in \bigcap_{\sigma' \in Comp(\sigma)} true(Res_0(\alpha, \sigma'))$. (by definition of Res_ω)
$\Rightarrow \forall \sigma' \in Comp(\sigma) \quad f \in true(Res_0(\alpha, \sigma'))$.

$\Rightarrow \forall \sigma' \in Comp(\sigma) \quad f \in Res(\alpha, true(\sigma'))$. (by Item 4, Observation 3.2)
$\Rightarrow \forall s' \in \Sigma \quad f \in Res(\alpha, s')$. (by Item 3, Observation 3.2)
$\Rightarrow \forall s^* \in \hat{\Sigma} \quad f \in s^*$. (1)

Let $f \in false(Res_\omega(\alpha, \sigma))$.
$\Rightarrow f \in \bigcap_{\sigma' \in Comp(\sigma)} false(Res_0(\alpha, \sigma'))$. (by definition of Res_ω)
$\Rightarrow \forall \sigma' \in Comp(\sigma) \quad f \in false(Res_0(\alpha, \sigma'))$.
$\Rightarrow \forall \sigma' \in Comp(\sigma) \quad f \notin Res(\alpha, true(\sigma'))$. (by Item 4, Observation 3.2)
$\Rightarrow \forall s \in \Sigma \quad f \notin Res(\alpha, s)$. (by Item 3, Observation 3.2)
$\Rightarrow \forall s^* \in \hat{\Sigma} \quad f \notin s^*$. (2)

The lemma follows from (1) and (2). □

Lemma 3.24 *Let D be a domain description, σ be an a-state, and δ be a grounded c-state of D such that σ agrees with δ. Then, for every sensing action a of D that is ω-executable in σ, there exists $\sigma' \in \Phi_\omega(a, \sigma)$ such that σ' agrees with $\Phi(a, \delta)$.*

Proof. Assume that a occurs in the k-propositions: a **determines** f_1, ..., a **determines** f_n By definition, we have that $K(a, \sigma) = \{f_1, \ldots, f_n\}$.

Assume that $\sigma = \langle T, F \rangle$ and $\delta = \langle s, \Sigma \rangle$. Let $K_1 = s \cap K(a, \sigma)$ and $K_2 = K(a, \sigma) \setminus s$.

Since δ is a grounded c-state, we have that $s \in \Sigma$. From the assumption that σ agrees with δ, we have that $T \subseteq s$ and $F \cap s = \emptyset$. This, together with the definitions of K_1 and K_2, implies that $K_1 \cap F = \emptyset$ and $K_2 \cap T = \emptyset$. Therefore, we have that $\sigma' = \langle T \cup K_1, F \cup K_2 \rangle \in \Phi_\omega(a, \sigma)$. We will prove that σ' agrees with $\Phi(a, \delta)$.

Let $\Phi(a, \delta) = \langle s, \Sigma' \rangle = \delta'$. Consider an arbitrary $s' \in \Sigma'$. By definition of $\Phi(a, \delta)$, we have that $s' \cap \{f_1, \ldots, f_n\} = s \cap \{f_1, \ldots, f_n\} = K_1$ and $\{f_1, \ldots, f_n\} \setminus s' = \{f_1, \ldots, f_n\} \setminus s = K_2$. Thus, $K_1 \subseteq s'$ and $s' \cap K_2 = \emptyset$. Since σ agrees with δ we have that $T \subseteq s'$ and $F \cap s' = \emptyset$. Therefore, $T \cup K_1 \subseteq s'$ and $(F \cup K_2) \cap s' = \emptyset$. (1)

Since (1) holds for every state $s' \in \Sigma'$, we have that $\langle T \cup K_1, F \cup K_2 \rangle$ agrees with $\langle s, \Sigma' \rangle$. This proves the lemma. □

The next lemma is the generalization of the lemmas 3.23 and 3.24 to a sequence actions consisting of both sensing and non-sensing actions.

Lemma 3.25 *Let D be a domain description, σ be an a-state, and δ be a grounded c-state of D such that σ agrees with δ. Then, for every sequence of actions α that is ω-executable in σ,*

(i) α is executable in δ; and

(ii) there exists an a-state $\sigma' \in \hat{\Phi}_\omega(\alpha, \sigma)$ such that σ' agrees with $\hat{\Phi}(\alpha, \delta)$.

Proof. Let $n_s(\alpha)$ be the number of sensing actions occurring in α. We prove the lemma by induction over $n_s(\alpha)$.

Base case: $n_s(\alpha) = 0$, i.e., α is a sequence of non-sensing actions. Item 5 of Observation 3.2 proves that α is executable in δ. Furthermore, by Lemma 3.23, we have that $Res_\omega(\alpha, \sigma)$ agrees with $\hat{\Phi}(\alpha, \delta)$. Since $\alpha = pre(\alpha)$, by definition of $\hat{\Phi}_\omega$, we have that $\hat{\Phi}_\omega(\alpha, \sigma) = \Phi_\omega(\alpha, \sigma) = \{Res_\omega(\alpha, \sigma)\}$. This proves the base case.

Inductive step: Assume that the first sensing action occurring in α is a, i.e., $\alpha = \beta; a; \gamma$ where β does not contain a sensing action. Let $\hat{\Phi}(\beta, \delta) = \delta_1$ and $Res_\omega(\beta, \sigma) = \sigma_1$. Then, by Lemma 3.23, σ_1 agrees with δ_1.

Since δ is a grounded c-state and β is a sequence of non-sensing actions, using Item 2 of Observation 3.2, we can easily prove that δ_1 is a grounded c-state. α is ω-executable in σ implies that $a; \gamma$ is ω-executable in σ_1. Hence, by Lemma 3.24, a is executable in δ_1 and $\exists \sigma_2 \in \Phi_\omega(a, \sigma_1)$ such that σ_2 agrees with $\Phi(a, \delta_1) = \delta_2$.

Again, from the assumption that α is ω-executable in σ we conclude that γ is ω-executable in σ_2. Since $n_s(\gamma) = n_s(\alpha) - 1$, by the induction hypothesis, we conclude that

γ is executable in δ_2 and $\exists \sigma_3 \in \hat{\Phi}_\omega(\gamma, \sigma_2)$ such that σ_3 agrees with $\hat{\Phi}(\gamma, \delta_2) = \delta_3$.

From $Res_\omega(\beta, \sigma) = \sigma_1$, $\sigma_2 \in \Phi_\omega(a, \sigma_1)$, $\sigma_3 \in \hat{\Phi}_\omega(\gamma, \sigma_2)$, and by definition of Φ_ω, we have that $\sigma_3 \in \hat{\Phi}_\omega(\alpha, \sigma)$ (1)

From $\hat{\Phi}(\beta, \delta) = \delta_1$, $\Phi(a, \delta_1) = \delta_2$, $\hat{\Phi}(\gamma, \delta_2) = \delta_3$, and by definition of $\hat{\Phi}$, we have that $\hat{\Phi}(\alpha, \delta) = \delta_3$. (2)

Since σ_3 agrees with δ_3, from (1) and (2), we can conclude the induction step. Hence, the lemma is proved. ☐

In the next lemma we extend the result of Lemma 3.25 to an arbitrary conditional plan.

Lemma 3.26 *Let D be a domain description, σ be an a-state, and δ be a grounded c-state of D such that σ agrees with δ. Then, for every conditional plan c such that c is executable in σ,*

(i) c is executable in δ; and

(ii) there exists an a-state $\sigma' \in \hat{\Phi}_\omega(c, \sigma)$ such that σ' agrees with $\hat{\Phi}(c, \delta)$.

Proof. By Observation 3.1, we know that c can be represented as a sequence of conditional plans $c_1; \ldots; c_n$ where c_i is either a sequence of actions or a case plan and for every $i < n$ if c_i is a sequence of actions then c_{i+1} is a case plan. We prove the lemma by induction over $count(c)$, the number of case plans in c.

The base case, $count(c) = 0$, is proved by Lemma 3.25.

We now prove the inductive step, i.e., assume that the lemma is shown for $count(c) \leq k$, we prove the lemma for $count(c) = k + 1$. We consider two cases:

(a) c_1 is a case plan. Assume that c_1 is the following plan

Case
$\varphi_1 \rightarrow p_1$
$\vdots$
$\varphi_l \rightarrow p_l$
Endcase

From the assumption that $\perp \notin \hat{\Phi}_\omega(c, \sigma)$ we can conclude that there exists j, $1 \leq j \leq l$, such that φ_j holds in σ. Let $c' = p_j; c_2 \ldots; c_n$. Then, by definition of $\hat{\Phi}_\omega$ we have that

$$\hat{\Phi}_\omega(c, \sigma) = \hat{\Phi}_\omega(c', \sigma). \quad (1)$$

Hence, $\bot \not\in \hat{\Phi}_\omega(c', \sigma)$. Since $count(c) > count(c')+1$, we have that $count(c') \leq k$. By inductive hypothesis, we have that

c' is executable in δ and there exists a state $\sigma' \in \hat{\Phi}_\omega(c', \sigma)$ such that σ' agrees with $\hat{\Phi}(c', \delta)$. (2)

Since σ agrees with δ and φ_j holds in σ, we can conclude that φ_j holds in δ, which implies that c is executable in δ and

$\hat{\Phi}(c, \delta) = \hat{\Phi}(c', \delta).$ (3)

From (3), (2), and (1), we have that $\sigma' \in \hat{\Phi}_\omega(c, \sigma)$ and σ' agrees with $\hat{\Phi}(c, \delta)$. This proves the lemma for the case (a).

(b) c_1 is a sequence of actions. Let $c' = c_2; \ldots; c_n$. Then, by definitions of $\hat{\Phi}$ and $\hat{\Phi}_\omega$, we have that $\hat{\Phi}(c, \delta) = \hat{\Phi}(c', \hat{\Phi}(c_1, \delta))$ and

$\hat{\Phi}_\omega(c, \sigma) = \bigcup_{\sigma' \in \hat{\Phi}_\omega(c_1, \sigma)} \hat{\Phi}_\omega(c', \sigma').$ (4)

Since σ agrees with δ and δ is a grounded c-state, by Lemma 3.25, we know that there exists a state $\sigma_1 \in \hat{\Phi}_\omega(c_1, \sigma)$ such that σ_1 agrees with $\hat{\Phi}(c_1, \delta)$. Since c is ω-executable in σ we have that c' is ω-executable in σ_1. Furthermore, since c' starts with a case plan and $count(c') = k + 1$, from the first case, we can conclude that

c_1 is executable in δ and c' is executable in $\hat{\Phi}(c_1, \delta)$ and there exists a state $\sigma' \in \hat{\Phi}_\omega(c', \sigma_1)$ such that σ' agrees with $\hat{\Phi}(c', \hat{\Phi}(c_1, \delta))$ (5)

From (4) and (5), we have that c is executable in δ, $\sigma' \in \hat{\Phi}_\omega(c, \sigma)$ and σ' agrees with $\hat{\Phi}(c', \hat{\Phi}(c_1, \delta)) = \hat{\Phi}(c, \delta)$. Hence the inductive step is proved for case (b).

The inductive step follows from the above two cases. □

We are now ready to prove the Proposition 6.

Proposition 6 (Soundness of ω-Approximation w.r.t. $\models_{\mathcal{A}_K}$) Let D be a domain description, φ be a fluent formula, and c be a conditional plan. Then,

if $D \models_\omega$ **Knows** φ **after** c then $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c.

Proof. Let σ_0 be the initial a-state of D and δ_0 be a grounded initial c-state of D. By definition of σ_0 and δ_0, we have that σ_0 agrees with δ_0. (1)

From $D \models_\omega$ **Knows** φ **after** c, by definition of $\models_\omega$, we have that

$\bot \not\in \hat{\Phi}_\omega(c, \sigma_0)$, and (2)

for every $\sigma' \in \hat{\Phi}_\omega(c, \sigma_0)$, φ holds in σ'. (3)

By Lemma 3.26, (1)-(3), we have that

c is executable in δ_0, and (4)

there exists a state $\sigma' \in \hat{\Phi}_\omega(c, \sigma_0)$, such that σ' agrees with $\hat{\Phi}(c, \delta_0)$. This, together with (4), implies that φ is known to be true in $\hat{\Phi}(c, \delta_0)$. (5)

(4) and (5) hold for every model (δ_0, Φ) of D. This implies that $D \models_{\mathcal{A}_K}$ **Knows** φ **after** c. The proposition is proved. □

Appendix D - Proof of the Regression proposition

In this section, we prove the regression proposition. For shorter notation, we write $\sigma \models \varphi$ (resp. $\sigma \not\models \varphi$) to denote that φ holds in σ (resp. φ does not hold in σ). We first prove several lemmas that we will use in the proof.

Lemma 4.27 *Let f be a fluent literal, a be an action, and s be a state. Assume that a is executable in s. Then, f holds in $Res(a,s)$ iff $Regression(f,a)$ holds in s.*

Proof. Consider the case that a is a non-sensing action and f is a fluent. Assume that a **causes** f **if** ϱ_1, …, a **causes** f **if** ϱ_n, and a **causes** $\neg f$ **if** ϱ'_1,…, a **causes** $\neg f$ **if** ϱ'_m are the ef-propositions in D whose action is a. Then, we have that

$Regression(f,a) = \bigvee_{i=1}^{n} \varrho_i \vee (f \wedge \bigwedge_{i=1}^{m} \neg \varrho'_i)$ holds in s
iff there exists an ef-proposition a **causes** f **if** ϱ in D such that ϱ holds in s or
f holds in s and there exists no ef-proposition a **causes** $\neg f$ **if** ϱ' in D such that ϱ' holds in s
iff $f \in E_a^+(s)$ or ($f \in s$ and $f \notin E_a^-(s)$)
iff $f \in s \cup E_a^+(s) \setminus E_a^-(s)$
iff f holds in $Res(a,s)$. (1)

Similarly, we can prove that (1) also holds when f is a negative fluent literal. (2)

Consider the case that a is a sensing action. Then, we have that $Res(a,s) = s$ (Recall that we assume that the set of sensing actions and non-sensing actions are disjoint.) And, $Regression(f,a) = f$. Thus, the lemma is trivial for this case. (3)

The lemma follows from (1)-(3). □

The next corollary follows immediately from Lemma 4.27 and the fact that $Regression(\bigwedge_i^n f_i, a) = \bigwedge_i^n Regression(f_i, a)$.

Corollary 2 *For a conjunction of fluent literals φ, an action a, a state s such that a is executable in s, φ holds in $Res(a,s)$ iff $Regression(\varphi,a)$ holds in s.* □

Lemma 4.28 *Let φ be a fluent formula, a be an action, and s be a state such that a is executable in s. Then, φ holds in $Res(a,s)$ iff $Regression(\varphi,a)$ holds in s.*

Proof. Since every Boolean expression can be represented by a CNF formula, we assume that $\varphi = \bigvee_i \varphi_i$ where each φ_i is a conjunction of fluent literals. Thus the lemma follows directly from Corollary 2 and the fact that $Regression(\bigvee_i^n \varphi_i, a) = \bigvee_i^n Regression(\varphi_i, a)$. □

Lemma 4.29 *Let φ be a fluent formula, a be an action, and $\sigma = \langle s, \Sigma \rangle$ be a grounded c-state. Assume that a is executable in every state belonging to Σ*[10]*. Then,*

- *if $Regression($**Knows** $(\varphi), a)$ holds in σ then* **Knows** (φ) *holds in $\Phi(a,\sigma)$; and*
- *if $Regression($**Knows** $(\varphi), a)$ does not hold in σ then* **Knows** (φ) *does not hold in $\Phi(a,\sigma)$.*

Proof. Consider the case a is a non-sensing action. Then, we have that $Regression($**Knows** $(\varphi), a) =$ **Knows** $(Regression(\varphi, a))$.

---

[10] This implies that a is executable in s since σ is a grounded c-state.

- $Regression(\mathbf{Knows}\,(\varphi), a)$ holds in σ implies that $Regression(\varphi, a)$ holds in every state $s' \in \Sigma$. This implies that φ holds in $Res(a, s')$ for every state $s' \in \Sigma$ such that a is executable in s' (Lemma 4.27). Therefore, $\mathbf{Knows}\,(\varphi)$ holds in $\Phi(a, \Sigma)$.

- $Regression(\mathbf{Knows}\,(\varphi), a)$ does not hold in σ means that there exists $s' \in \Sigma$ such that $Regression(\varphi, a)$ does not hold in s'. Since a is executable in s', by Lemma 4.27, we conclude that φ does not hold in $Res(a, s')$. This implies that $\mathbf{Knows}\,(\varphi)$ does not hold in $\Phi(a, \sigma)$.

Consider the case a is a sensing action that senses a fluent g[11], we have that $Regression(\varphi, a) = \varphi$ and $\Phi(a, \sigma) = \langle s, \Sigma' \rangle$ where $\Sigma' \subseteq \Sigma$ and each state s' in Σ' agrees with s on g.

- $Regression(\mathbf{Knows}\,(\varphi), a) = (g \rightarrow \mathbf{Knows}\,(g \rightarrow \varphi)) \wedge (\neg g \rightarrow \mathbf{Knows}\,(\neg g \rightarrow \varphi))$ holds in σ implies that $g \in s$ (resp. $g \notin s$) implies that $\mathbf{Knows}\,(g \rightarrow \varphi)$ (resp. $\mathbf{Knows}\,(\neg g \rightarrow \varphi)$) holds in σ. So, if $g \in s$ (resp. $g \notin s$) then $g \in s'$ (resp. $g \notin s'$) implies that φ holds in s' for every $s' \in \Sigma$. In other words, for every $s' \in \Sigma$, if s and s' agree on g then φ holds in s'. Hence, φ is known to be true in $\Phi(a, \sigma)$, i.e., $\mathbf{Knows}\,(\varphi)$ holds in $\Phi(a, \sigma)$.

- $Regression(\mathbf{Knows}\,(\varphi), a) = (g \rightarrow \mathbf{Knows}\,(g \rightarrow \varphi)) \wedge (\neg g \rightarrow \mathbf{Knows}\,(\neg g \rightarrow \varphi))$ does not hold in σ implies that either (i) $(g \rightarrow \mathbf{Knows}\,(g \rightarrow \varphi))$ does not hold in σ or (ii) $(\neg g \rightarrow \mathbf{Knows}\,(\neg g \rightarrow \varphi))$ does not hold in σ. Let us assume that $g \rightarrow \mathbf{Knows}\,(g \rightarrow \varphi)$ does not hold in σ, i.e., (i) holds. This means that g holds in σ but $\mathbf{Knows}\,(g \rightarrow \varphi)$ does not. So, there exists a state s' in Σ such that $g \in s'$ and φ does not hold in s' or for every s' in Σ, $g \notin s'$. The first case implies that $\mathbf{Knows}\,(\varphi)$ does not hold in $\Phi(a, \sigma)$. The second case is impossible because σ is a grounded c-state. Thus if (i) holds then $\mathbf{Knows}\,(\varphi)$ does not hold in $\Phi(a, \sigma)$. Similarly, if (ii) holds, we can show that $\mathbf{Knows}\,(\varphi)$ does not hold in $\Phi(a, \sigma)$.

The lemma follows from the above two cases. $\square$

Lemma 4.30 *For a c-formula φ^*, an action a, and a grounded c-state $\sigma = \langle s, \Sigma \rangle$ such that a is executable in every state belonging to Σ,*

- *if $\sigma \models Regression(\varphi^*, a)$ then $\Phi(a, \sigma) \models \varphi^*$; and*
- *if $\sigma \not\models Regression(\varphi^*, a)$ then $\Phi(a, \sigma) \not\models \varphi^*$.*

Proof. Follows from Lemmas 4.28 and 4.29 and the fact that each c-formula φ^* can be represented by a disjunction $\bigvee_{i=1}^{n} \varphi_i^*$ where φ_i^* is a conjunction of fluent literals and k-formulas of the form $\mathbf{Knows}\,(\varrho)$ for some fluent formula ϱ. $\square$

Lemma 4.31 *Let φ be a c-formula and c be a conditional plan. Then, $Regression(\varphi, c)$ is a c-formula.*

Proof. The proof is done inductively over $count(c)$, the number of case plans in c. The base case, c is a squence of actions, follows immediately from Items (1)-(4) and the first two sub-items of Item (5) of the definition of the regression formulas. The inductive step follows from inductive hypothesis and the last two sub-items of Item (5) of the definition of the regression formulas. $\square$

---

[11] The proof for the case when a senses more than 1 fluents $g_1, \ldots, g_n$ is similar and is omitted here.

Lemma 4.32 *For a c-formula φ, an action sequence α, and a grounded c-state σ such that α is executable in every grounded c-state $\sigma' = \langle s', \Sigma' \rangle$ where $\Sigma' \subseteq \Sigma$,*

- *if $\sigma \models Regression(\varphi, \alpha)$ then $\hat{\Phi}(\alpha, \sigma) \models \varphi$; and*
- *if $\sigma \not\models Regression(\varphi, \alpha)$ then $\hat{\Phi}(\alpha, \sigma) \not\models \varphi$.*

Proof. Induction over $|\alpha|$, the length of α.

Base case: $|\alpha| = 0$, i.e., $\alpha = []$.. Then, we have that $Regression(\varphi, []) = \varphi$ and $\hat{\Phi}([], \sigma) = \sigma$. The lemma is trivial. (Notice that for $|\alpha| = 1$, the lemma follows from Lemma 4.30). The base case is proved.

Inductive step: Assume that we have proved the lemma for $|\alpha| = n$. We need to prove the lemma for $|\alpha| = n + 1$. Let $\alpha = \beta; a$. Then, we have that $|\beta| = n$.

We have that $Regression(\varphi, \beta; a) = Regression(\varphi, \beta; a) = Regression(Regression(\varphi, a), \beta)$. By inductive hypothesis, we have that

if $\sigma \models Regression(\varphi, \alpha)$ then $\hat{\Phi}(\beta, \sigma) \models Regression(\varphi, a)$. Thus, by Lemma 4.30, $\Phi(a, \hat{\Phi}(\beta, \sigma)) \models \varphi$, i.e., $\hat{\Phi}(\alpha, \sigma) \models \varphi$.

if $\sigma \not\models Regression(\varphi, \alpha)$ then $\hat{\Phi}(\beta, \sigma) \not\models Regression(\varphi, a)$. Again, by Lemma 4.30, we have that $\Phi(a, \hat{\Phi}(\beta, \sigma)) \not\models \varphi$, i.e., $\hat{\Phi}(\alpha, \sigma) \not\models \varphi$.

Lemma 4.33 *For a c-formula φ, a grounded c-state $\sigma = \langle s, \Sigma \rangle$, and a conditional plan c such that c is executable in every c-state $\sigma' = \langle s', \Sigma' \rangle$ where $\Sigma' \subseteq \Sigma$,*

- *if $\sigma \models Regression(\varphi, c)$ then $\hat{\Phi}(c, \sigma) \models \varphi$; and*
- *if $\sigma \not\models Regression(\varphi, c)$ then $\hat{\Phi}(c, \sigma) \not\models \varphi$.*

Proof. As in previous proofs related to conditional plans, we assume that c is a sequence of conditional plans $c_1; \ldots; c_n$ where c_i is either a sequence of actions or a case plan and for every $i < n$ if c_i is a sequence of actions then c_{i+1} is a case plan. We prove the lemma by induction over $count(c)$, the number of case plans in c.

Base case: $count(c) = 0$. Then, c is a sequence of actions. The base case follows from Lemma 4.32.

Inductive step: Assume that we have proved the lemma for $count(c) \leq k$. We need to prove the lemma for $count(c) = k + 1$. By construction of c, we have two cases:

Case 1: c_n is a case plan of the form

Case
  $\varphi_1 \rightarrow p_1$
  $\vdots$
  $\varphi_l \rightarrow p_l$
Endcase

Let $c' = c_1; \ldots; c_{n-1}$. We have that

$Regression(\varphi, c'; c_n) = Regression(Regression(\varphi, c_n), c')$. Since $count(c) = count(c') + count(c_n)$ and $count(c_n) \geq 1$, we have that $count(c') \leq k$. Furthermore, by Lemma 4.31, we have that $Regression(\varphi, c_n)$ is a c-formula. Consider two cases:

Case 1.1: $\sigma \models \mathit{Regression}(\varphi, c)$. By inductive hypothesis (for $\mathit{Regression}(\varphi, c_n)$, σ, and c'), we have that $\hat{\Phi}(c', \sigma) \models \mathit{Regression}(\varphi, c_n)$.

Let $\delta = \hat{\Phi}(c', \sigma)$. Since c is executable in σ we conclude that there exists some j $(1 \leq j \leq l)$ such that $\delta \models \mathbf{Knows}\,(\varphi_j)$ and $\delta \not\models \mathbf{Knows}\,(\varphi_i)$ for $i \neq j$. This, together with the fact that $\mathit{Regression}(\varphi, c_n) = \bigvee_{i=1}^{l}(\mathbf{Knows}\,(\varphi_i) \wedge \mathit{Regression}(\varphi, p_i))$, implies that $\delta \models \mathbf{Knows}\,(\varphi_j) \wedge \mathit{Regression}(\varphi, p_j)$. Hence, $\delta \models \mathit{Regression}(\varphi, p_j)$. Applying the inductive hypothesis one more time (for φ, δ, and p_j), we can conclude that $\hat{\Phi}(p_j, \delta) \models \varphi$. Since $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) = \hat{\Phi}(p_j, \delta)$, we have that $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) \models \varphi$, i.e., $\hat{\Phi}(c, \sigma) \models \varphi$. (1)

Case 1.2: $\sigma \not\models \mathit{Regression}(\varphi, c)$. Again, by inductive hypothesis (for $\mathit{Regression}(\varphi, c_n)$, σ, and c'), we have that $\hat{\Phi}(c', \sigma) \not\models \mathit{Regression}(\varphi, c_n)$.

Let $\delta = \hat{\Phi}(c', \sigma)$. Since c is executable in σ we conclude that there exists some j $(1 \leq j \leq l)$ such that $\delta \models \mathbf{Knows}\,(\varphi_j)$ and $\delta \not\models \mathbf{Knows}\,(\varphi_i)$ for $i \neq j$. This, together with the fact that $\mathit{Regression}(\varphi, c_n) = \bigvee_{i=1}^{l}(\mathbf{Knows}\,(\varphi_i) \wedge \mathit{Regression}(\varphi, p_i))$, implies that $\delta \not\models \mathbf{Knows}\,(\varphi_j) \wedge \mathit{Regression}(\varphi, p_j)$. Hence, $\delta \not\models \mathit{Regression}(\varphi, p_j)$. Applying the inductive hypothesis one more time (for φ, δ, and p_j), we have that $\hat{\Phi}(p_j, \delta) \not\models \varphi$. Since $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) = \hat{\Phi}(p_j, \delta)$, we have that $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) \not\models \varphi$, i.e., $\hat{\Phi}(c, \sigma) \not\models \varphi$. (2)

The inductive step for Case 1 follows from (1) and (2).

Case 2: c_n is a sequence of actions.

Let $c' = c_1; \ldots; c_{n-1}$. We have that

$\mathit{Regression}(\varphi, c'; c_n) = \mathit{Regression}(\mathit{Regression}(\varphi, c_n), c')$.

It follows from Observation 3.1 that c_{n-1} is a case plan. By case 1 and the inductive hypothesis, (for $\mathit{Regression}(\varphi, c_n)$, σ, and c'), we have that

if $\sigma \models \mathit{Regression}(\varphi, c)$ then $\hat{\Phi}(c', \sigma) \models \mathit{Regression}(\varphi, c_n)$. Then, by Lemma 4.32 (for φ, $\hat{\Phi}(c', \sigma)$, and c_n), we can conclude that $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) \models \varphi$, i.e., $\hat{\Phi}(c, \sigma) \models \varphi$.

if $\sigma \not\models \mathit{Regression}(\varphi, c)$ then $\hat{\Phi}(c', \sigma) \not\models \mathit{Regression}(\varphi, c_n)$. Again, by Lemma 4.32, we have that $\hat{\Phi}(c_n, \hat{\Phi}(c', \sigma)) \not\models \varphi$, i.e., $\hat{\Phi}(c, \sigma) \not\models \varphi$.

This proves the inductive step and hence, the lemma is proved. □

We now prove Proposition 8.

Proposition 8 *Given a domain description D, let φ be a c-formula, and $\sigma_1, \ldots, \sigma_n$ be the set of grounded initial c-states of D, and c be a conditional plan that is executable in all the grounded initial c-states of D.*

- $\forall i, 1 \leq i \leq n\ \sigma_i \models \mathit{Regression}(\varphi, c)$ *iff* $\forall j, 1 \leq j \leq n\ \hat{\Phi}(c, \sigma_j) \models \varphi$

Proof. Let $\sigma_i = \langle s, \Sigma \rangle$ be a grounded initial c-state of D. It is easy to see that each grounded c-state $\langle s', \Sigma' \rangle$ where $\Sigma' \subseteq \Sigma$ is also a grounded initial c-state of D. Thus, by the first item of Lemma 4.33, we have that

if $\sigma_i \models \mathit{Regression}(\varphi, c)$ then $\hat{\Phi}(c, \sigma_j) \models \varphi$ (1)

Using the second item of Lemma 4.33, we can prove that

if $\hat{\Phi}(c, \sigma_j) \models \varphi$ then $\sigma_i \models \mathit{Regression}(\varphi, c)$. (2)

The conclusion of the lemma follows from the fact that (1) and (2) hold for every i, $1 \leq i \leq n$. □

Appendix E - Overview of Nested Circumscription

Nested Abnormality Theories (NATs) is a novel circumscription [**?**, **?**] technique introduced by Lifschitz [Lif95]. With NATs it is possible to circumscribe several predicates each with respect to only parts of the theory of interest, as opposed to previous techniques such as parallelized and circumscription theories where the circumscription must be done with respect to all of the axioms in the underlying theory. Furthermore, all the complications arising from the interaction of multiple circumscription axioms in a theory are avoided in NATs with the introduction of blocks. A *block* is characterized by a set of axioms $A_1, \ldots, A_n$—possibly containing the abnormality predicate Ab –which 'describe' a set of predicate/function constants $C_1, \ldots, C_m$. The notation for such a theory is

$$\{C_1, \ldots, C_m \ : \ A_1, \ldots, A_n\} \tag{4.17}$$

where each A_i may itself be a block of form (4.17). The 'description' of $C_1, \ldots, C_m$ by a block may depend on other descriptions in embedded blocks.

Interference between circumscription in different blocks is prevented by replacing a predicate Ab with an existentially quantified variable. Lifschitz's idea is to make Ab 'local' to the block where it is used, since abnormality predicates play only an auxiliary role, i.e. the interesting consequences of the theory are those which do not contain Ab. The next section contains the formal definitions of this concepts.

The following definitions are from [Lif95]. Let L be a second order language which does not include Ab. For every natural number k, let L_k be the language obtained by adding the k-ary predicate constant Ab to L. $\{C_1, ..., C_m \ : \ A_1, ..., A_n\}$ is a *block* if each $C_1, \ldots, C_m$ is a predicate or a function constant of L, and each $A_1, \ldots, A_n$ is a formula of L_k or a block.

A *Nested Abnormality Theory* is a set of blocks. The semantics of NATs is characterized by a mapping φ from blocks into sentences of L. If A is a formula of language L_k, φA stands for the universal closure of A, otherwise

$$\varphi\{C_1, ..., C_m \ : \ A_1, ..., A_n\} \ = \ (\exists ab)F(ab)$$

where

$$F(Ab) = \mathrm{CIRC}[\varphi A_1 \wedge ... \wedge \varphi A_n; Ab; C_1, ..., C_m]$$

Recall that $\mathrm{CIRC}[T; P; Q]$, means circumscription of the theory T, by minimizing the predicates in P, and varying the objects in Q.

For any NAT T, φT stands for $\{\varphi A | A \in T\}$. A *model* of T is a model of φT in the sense of classical logic. A *consequence* of T is a sentence ϕ of language L that is true in all models of T. In this paper, as suggested in [Lif95], we use the abbreviation

$$\{C_1, \ldots, C_m, \ \min P : A_1, \ldots, A_n\}$$

to denote blocks of the form

$$\{C_1, \ldots, C_m, P : P(x) \supset Ab(x), A_1, \ldots, A_n\}$$

As the notation suggests, this type of block is used when it is necessary to circumscribe a particular predicate P in a block. In [Lif95] it is shown that

$$\varphi\{C_1, \ldots, C_m,\ \min P : A_1, \ldots, A_n\}$$

is equivalent to the formula

$$\mathrm{CIRC}[A_1 \wedge \ldots \wedge A_n; P; C_1, \ldots, C_m]$$

when each A_i is a sentence.