Theory and Practice of Logic Programming **1** (2): 129–184, March 2001.

Knowledge and the action description language $\mathscr{A}$

JORGE LOBO*

Network Computing Research Department, Bell Laboratories, Murray Hill, NJ 07974, USA
e-mail: jlobo@research.bell-labs.com

GISELA MENDEZ†

Departamento de Matemáticas, Universidad Central de Venezuela, Caracas, Venezuela
e-mail: gmendez@ciens.ucv.ve

STUART R. TAYLOR

Raytheon Systems Company, Expeditionary Warfare & Industrial Automotive,
13532 N. Central Exp., MS 37, Dallas, TX, 75243, USA
e-mail: s-taylor5@ti.com

Abstract

We introduce $\mathscr{A}_k$, an extension of the action description language $\mathscr{A}$ (Gelfond and Lifschitz, 1993) to handle actions which affect knowledge. We use sensing actions to increase an agent's knowledge of the world and non-deterministic actions to remove knowledge. We include complex plans involving conditionals and loops in our query language for hypothetical reasoning. We also present a translation of $\mathscr{A}_k$ domain descriptions into epistemic logic programs.[1]

KEYWORDS: action theories, epistemic logic programs

1 Introduction

Since its introduction, the action description language $\mathscr{A}$ has served as a platform to study several aspects that arise when we try to formalize theories of actions in logic (Gelfond and Lifschitz, 1993). $\mathscr{A}$ was designed as a minimal core of a high level language to represent and reason about actions and their effects. Domain descriptions written in this language have direct translations into extended logic programs. Extensions to $\mathscr{A}$ have been developed to study and reason about the concurrent execution of actions (Baral and Gelfond, 1997), the non-deterministic effects of some actions (Thielscher, 1994) and to study many instances of the qualification and ramification problems (Kartha and Lifschitz, 1994; Kartha and Lifschitz, 1997; McCain and Turner, 1997).

* Partially funded by Argonne National Laboratory under Contract No. 963042401. The research was partially conducted at the EECS department of the University of Illinois at Chicago.

† Work done while visiting the University of Illinois at Chicago and Bell Labs.

[1] This paper extends the results of the work first presented in Lobo *et al.* (1997).

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

In this paper we propose a new action description language called $\mathscr{A}_k$. $\mathscr{A}_k$ is a minimal extension of $\mathscr{A}$ to handle *sensing* actions. A sensing action is an action that does not have any effect in the world. The effect is only in the perception of the reasoning agent about the world. The execution of a sensing action will increase the agent's knowledge about the current state of the world. Take for example a deactivated agent placed inside a room. The agent has duties to carry out and will be activated by a timer. Let us assume the agent is always placed facing the door. The agent, once activated, may become damaged if it attempts to leave the room and the door is closed. Before the agent tries to leave the room it needs to perform some act of sensing in order to determine whether the door is opened or not. The agent has incomplete knowledge with respect to the door. A sensing action such as *looking at the door* would provide information to the agent concerning the status of the door.

In our simple model there will be two sources of knowledge available to an agent: initial knowledge, i.e. knowledge provided to the agent at initialization time, and knowledge gained from sensing actions. We will assume that the agent is acting in isolation. Thus, once an agent has gained knowledge about its world, only its actions or limitations of its reasoning mechanism (such as limited memory) could make the agent lose knowledge. We will assume an ideal agent and expect that only actions can remove knowledge. An action can cause the loss of knowledge if its effect is non-deterministic. Take for example the action of tossing a coin. We know it will land with either heads showing or with tails showing, but exactly which cannot be predicted. Non-deterministic actions and sensing actions have opposite effects on an agent's knowledge.

The main contributions of this paper are:

- The language $\mathscr{A}_k$, which incorporates sensing and non-deterministic actions.
- A query sub-language with complex plans that allow hypothetical reasoning in the presence of incomplete information. These complex plans include conditionals (if-then-else) and routines (while-do).
- A sound and complete translation of domain descriptions written in a subset of $\mathscr{A}_k$ into epistemic logic programs.

The paper is organized as follows. In Section 2, we start with the syntax and semantics of domains with deterministic and sensing actions only. Section 3 presents the query sub-language of $\mathscr{A}_k$ with conditional plans. In Section 4, the language is extended to include non-deterministic actions and Section 5 adds loops to the query language. Section 6 gives an outline of epistemic logic programs as they pertain to $\mathscr{A}_k$. In Section 7, we present the translation of domains in $\mathscr{A}_k$ into epistemic logic programs. In Section 8, we discuss how our work relates to other work in the field. Section 9 presents a few directions for future work and concluding remarks.

2 $\mathscr{A}_k$: Domain Language

2.1 Syntax of $\mathscr{A}_k$

The language of $\mathscr{A}_k$ consists of two non-empty disjoint sets of symbols F, A. They are called *fluents* and *actions*. As in $\mathscr{A}$, fluents are statements or observations about

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

the world. The set A consists of two disjoint sets of actions, *sensing* actions and *non-sensing* actions. Actions will be generically denoted by a, possibly indexed. A *fluent literal* is a fluent or a fluent preceded by a $\neg$ sign. A fluent literal is negative when preceded by $\neg$ and is positive otherwise. Fluent literals will be denoted by f, p and q possibly indexed.

There are three kinds of propositions in $\mathscr{A}_k$: *object effect* propositions, *value* propositions and *non-deterministic effect* propositions. We discuss non-deterministic effect propositions in Section 4. Object effect propositions are expressions of the form

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n \tag{1}$$

where a is a non-sensing action, and f and $p_1, \ldots, p_n$, with $n \leqslant 0$, are fluent literals. This expression intuitively means that in a situation where $p_1, \ldots, p_n$ are true, the execution of a causes f to become true.

When $n = 0$ in the preconditions of (1) we will write the proposition as

$$a \textbf{ causes } f \tag{2}$$

A *value proposition* is an expression of the form

$$\textbf{initially } f \tag{3}$$

where f denotes a fluent literal. Value propositions describe the initial knowledge the agent has about the world.

There are also *knowledge* laws. Knowledge laws are expressions of the form

$$a_s \textbf{ causes to know } f \textbf{ if } p_1, \ldots, p_n \tag{4}$$

where a_s is a sensing action, f is a fluent and $p_1, \ldots, p_n$ are preconditions as in (1). Intuitively this expression says that in a situation where $p_1, \ldots, p_n$ are true the execution of a_s causes the agent to realize the current value of f in the world. We do not allow sensing actions to occur in any effect proposition.

If $n = 0$ in (4), we will write the knowledge law as

$$a_s \textbf{ causes to know } f \tag{5}$$

At this point we should remark that we are assuming the agent may have incomplete but always correct knowledge about the world. Propositions and laws in $\mathscr{A}_k$ describe how the knowledge of the agent changes, but if these changes are the result of propositions like (1) we assume that the effects in the world would be the same as if the world were in a state where $p_1, \ldots, p_n$ are true, that is, there are not external entities that modify the world and the specification of the laws are correct and deterministic.

Definition 2.1
A collection of the above propositions and laws is called a *domain description*. A domain description D is *simple* if for any sensing action a_s and any fluent f there exists at most one knowledge law in D of type (4).

The following example illustrates how knowledge laws can be used to reason about actions.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Example 2.2

A robot is instructed to replace the bulb of a halogen lamp. If the lamp is on when the bulb is screwed in, the robot's circuits will get burned out from the heat of the halogen bulb, and it will not be able to complete the task. The robot will have to find a sequence of actions that will allow it to complete the task without burning out. We assume that the robot is already at the lamp. This is represented by the following domain description,

$$D_1 \left\{ \begin{array}{l} r_1 \ : \ \textbf{initially} \ \neg burnOut \\ r_2 \ : \ \textbf{initially} \ \neg bulbFixed \\ r_3 : changeBulb \ \textbf{causes} \ burnOut \ \textbf{if} \ switchOn \\ r_4 : changeBulb \ \textbf{causes} \ bulbFixed \ \textbf{if} \ \neg switchOn \\ r_5 : turnSwitch \ \textbf{causes} \ switchOn \ \textbf{if} \ \neg switchOn \\ r_6 : turnSwitch \ \textbf{causes} \ \neg switchOn \ \textbf{if} \ switchOn \end{array} \right.$$

It follows from D_1 that in the initial state the robot does not know the state of the switch in the lamp. Hence, there does not exist a way to determine before hand what will be the result of the action *changeBulb*. When the robot goes to carry out the action *changeBulb*, it could end up in a resulting state in which either *bulbFixed* is true or in a state where it will be burned out and unable to complete the task. Without knowing whether the switch is on or off, the robot will not be able to find a plan to accomplish its task. The robot must first check the state of the switch. After realizing whether the switch is on or off, it will take the appropriate actions to complete the task. The robot will need a knowledge law such as:

$$r_7 : checkSwitch \ \ \textbf{causes to know} \ switchOn \ \ \textbf{if} \ \neg burnOut$$

After checking the switch the robot will *know* whether the switch is on or off. Sensing gives the robot that extra knowledge it would need to accomplish the task without burning out and provides a branching point in its hypothetical reasoning. If the switch is on it will turn the switch and replace the bulb. If the switch is off it will directly replace the bulb. This conditional reasoning will enable the robot to show that there is a sequence of actions to accomplish the task.

2.2 *Semantics of $\mathscr{A}_k$*

The semantics of $\mathscr{A}_k$ must describe how an agent's knowledge changes according to the effects of actions defined by a domain description. We begin by presenting the structure of an agent's knowledge. We will represent the knowledge of an agent by a set of possibly incomplete worlds in which the agent believes it can be. We call these worlds *situations* and a collection of worlds an *epistemic state*. A situation, since it could be an incomplete description of the world, will be represented by a collection of sets of fluents. A set of fluents will be called a *state*. If a formula is true in an epistemic state of an agent (to be defined later), by our assumption it means that the agent knows that the formula is true in the real world. Epistemic states will also

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

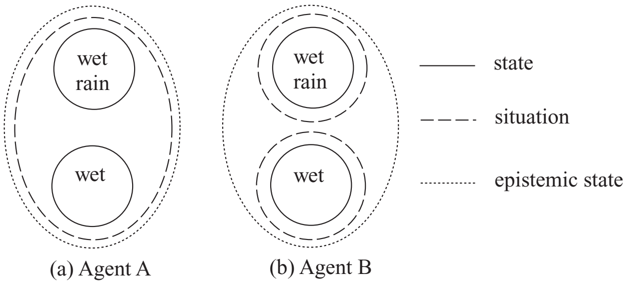

Fig. 1. Epistemic states for Agent A and Agent B.

allow us to distinguish when the agent knows that the disjunction $f_1 \vee f_2$ is true from when it either knows f_1 or knows f_2.[2]

We will say that a fluent f is true or holds in a state σ (denoted by $\sigma \models f$) iff $f \in \sigma$. A fluent f does not hold in a state σ (denoted by $\sigma \not\models f$) iff $f \notin \sigma$. $\sigma \models \neg f$ iff $\sigma \not\models f$. For more complex formulas, their truth value can be recursively defined as usual. A formula φ made of fluents is true in (or modeled by) a situation Σ (denoted by $\Sigma \models \varphi$) if the formula is true in every state in Σ; it is false if $\neg\varphi$ is true in every state Σ. A formula is true in an epistemic state if is true in every situation in the epistemic state; it is false if its negation is true.

A situation is *consistent* if it is non-empty; otherwise it is *inconsistent*. A situation is *complete* if it contains a single state; otherwise it is *incomplete*. An epistemic state is inconsistent if it is empty or contains an inconsistent situation; otherwise it is consistent. An epistemic state is complete if it contains only one complete situation. Figure 1 shows two consistent epistemic states in which the fact "*Ollie is wet*" (represented by *wet*) is known by Agent A and Agent B. In the epistemic state (a), containing an incomplete situation, Agent A does not have knowledge about the weather. In the other epistemic state (b), containing two complete situations, Agent B either knows it is raining or knows that it is not raining outside. Recall that epistemic states will be used in the context of plans for hypothetical reasoning. That is, predicting properties if the plan were executed. Thus, if an agent plans to execute a series of actions that takes it to the epistemic state (a), it will not know how to dress if it needs to go outside and does not want to get wet. In the epistemic state (b), the agent will know how to proceed.

Interpretations for $\mathscr{A}_k$ are transition functions that map pairs of actions and situations into situations. To define when an interpretation models a domain description, we will define an auxiliary function that interprets the effect of actions at the state level. We call this function a *0-interpretation*. *0-interpretations* are functions that map actions and states into states[3]. A 0-interpretation Φ_0 is a *0-model* of a domain description D iff for every state σ:

[2] Note the similarity with a collection of belief sets in Gelfond and Przymusinska (1991).

[3] 0-interpretations and 0-models are similar to interpretations and models for domains in $\mathscr{A}$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

1. For a fluent f of any effect proposition of the form "a **causes** f **if** $p_1, \dots, p_n$" in D, the fluent f holds in $\Phi_0(a, \sigma)$ if its preconditions $p_1, \dots, p_n$ holds in σ,
2. For a fluent literal $\neg f$ of any effect proposition of the form 'a **causes** $\neg f$ **if** $p_1, \dots, p_n$' in D, the fluent f does not hold in $\Phi_0(a, \sigma)$ if its preconditions $p_1, \dots, p_n$ holds in σ,
3. For a fluent f, if there are no effect propositions of the above types, then $f \in \Phi_0(a, \sigma)$ if and only if $f \in \sigma$.

Before we define when an interpretation Φ is a *model* of a domain description D, we need the following definition that will let us interpret knowledge laws. The interest of the defintion will become clear after we explore the scenario in Example 2.4.

Definition 2.3
Let Σ be a consistent situation, f a fluent and φ a disjunction of conjunctions of fluent literals (preconditions). A consistent situation Σ' is 'f, φ-*compatible*' with Σ iff $\Sigma' = \Sigma$ whenever f is either true or false in Σ. Otherwise Σ' must satisfy one of the following conditions:

1. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is not true in } \sigma\}$
2. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is true in } \sigma,\ f \notin \sigma\}$
3. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is true in } \sigma,\ f \in \sigma\}$

Example 2.4
Let us return to the agent scenario from the introduction. Imagine that currently the agent is deactivated in the room. The agent will be automatically activated by an internal clock. Then it needs to find the door, leave the room, and perform some duties. When the agent is initially activated it will know nothing about its surroundings and will remain ignorant of its surroundings until it performs a sensing action. We will show how the conditions presented in Definition 2.3 are enough to represent the result of sensing. Its only action is to *look*. We assume the action consist of opening its 'eyes' and looking. This domain is represented below with only one knowledge law,

$$D_2 \left\{ \ r_1 : look \ \textbf{causes to know} \ \ doorOpened \ \textbf{if} \ facingDoor \right.$$

This initial situation of complete ignorance is represented by the situation

$$\{\{\}, \{doorOpened\}, \{facingDoor\}, \{doorOpened, facingDoor\}\}.$$

If the action *look* is executed in the real world the agent may find that it is not facing the door and will not know whether the door is opened or not, this is represented by the situation

$$\{\ \{\ \}, \{doorOpened\}\ \}.$$

Another possibility could be that the agent was facing the door and after it is activated, it will know that it is facing the door and will also know that the door is not opened

$$\{\ \{facingDoor\}\ \}.$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Still another possibility could be that the agent was facing the door and after being activated, it will learn that it is facing the door and that the door is opened

$$\{\ \{doorOpened,\ facingDoor\}\ \}.$$

Since the agent will be doing hypothetical reasoning (i.e. planning) it will have no way of knowing which situation it will be in until the action is actually executed. Thus, the agent can only assume that it will be in one of the three situations, so when the agent analyses what would be the consequences of executing *look* it concludes that the result will take it to the epistemic state that consists of the following situations:

1. $\{\ \{\ \},\ \{doorOpened\}\ \}$
2. $\{\ \{facingDoor\}\ \}$
3. $\{\ \{doorOpened,\ facingDoor\}\ \}$

Each situation is *doorOpened, facingDoor − compatible*. The first situation corresponds to the first case of Definition 2.3. The agent knows it is not facing the door since *facingDoor* is false in all states contained in the situation. The same cannot be said for *doorOpened* since in one state it is false and the other state it is true. This is to be expected since in this situation the agent is not facing the door, and it cannot know if the door is opened or closed.

The second situation corresponds to the second case of Definition 2.3. This situation contains all the states in which the precondition *facingDoor* is true and the fluent *doorOpen* is false. The agent not only knows it is facing the door but also knows the door is not opened.

The last situation is from the last case of Definition 2.3. In this situation the agent knows it is facing the door and also knows that the door is opened.

Observe that a result of sensing is that the preconditions of the sensing action will become known to the agent if the value of the fluent being sensed is initially unknown. This occurs even if the effect of the action remains unknown after executing the action, which hapens in the situation coming from the states where the preconditions of the execution of the sensing action in a knowledge laws are not true.

Definition 2.5

A state σ is called an *initial state* of a domain description D iff for every value proposition of the form '**initially** φ' in D, φ is true in σ. The *initial situation* Σ_0 of D is the set of all the initial states of D.

Definition 2.6

A fluent f is a *potential sensing effect* of a sensing action a_s in a domain D if there is a knowledge law of the form

$$a_s\ \textbf{causes to know}\ f\ \textbf{if}\ \varphi$$

in D. We will also say that f is the *potential sensing effect* of the knowledge law.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

The *knowledge precondition* of a fluent f with respect to a sensing action a_s in a domain D is the disjunction $\varphi_1 \vee \ldots \vee \varphi_n$ if and only if

$$\begin{aligned} &a_s \textbf{ causes to know } f \textbf{ if } \varphi_1 \\ &\quad\vdots \\ &a_s \textbf{ causes to know } f \textbf{ if } \varphi_n \end{aligned}$$

are all the knowledge laws in which a_s occurs and f is a potential sensing effect.

Note that if the domain is simple (Definition 2.1) then the knowledge precondition of any fluent in the domain with respect to any sensing actions is either empty or it has only one disjoint.

Definition 2.7
Given an interpretation Φ of $\mathscr{A}_k$, Φ is a *model* of a domain description D, if and only if for any consistent situation Σ:

1. There exists a 0-model Φ_0 of D, such that for any non-sensing action a,
$$\Phi(a, \Sigma) = \bigcup_{\sigma \in \Sigma} \{\Phi_0(a, \sigma)\}.$$
2. For each sensing action a_s, let $f_1, \ldots, f_n$ be the potential sensing effects of a_s and φ_i the knowledge precondition of f_i with respect to a_s. Then, $\Phi(a_s, \Sigma)$ must be consistent and if $n = 0$, $\Phi(a_s, \Sigma) = \Sigma$ otherwise $\Phi(a_s, \Sigma) = \bigcap_{i \in [1..n]} \Sigma_i$, such that each Σ_i is a situation $f_i, \varphi_i - \textit{compatible}$ with Σ.

$\Phi(a, \Sigma) = \emptyset$ for any action a if $\Sigma = \emptyset$.

Example 2.8
The third floor agent of a building has the job of making sure the white-board in a room on that floor is clean. The agent will approach the room, look into the room, clean the white-board if it is not clean, and then leave the room. We focus here on ‘looking into the room’. When the agent looks into the room it will know whether the white-board in that room is clean. Also if the curtains are open the agent will learn whether it is raining outside. Sensing actions can not appear in object effect propositions, but there is no restriction on the number of knowledge laws associated with a sensing action. Thus, the action could affect the truth value of several fluents simultaneously. In this example the sensing action *lookInRoom* will appear in two knowledge laws. We will see how the resulting situations are f, φ-compatible with the initial situation and briefly discuss the models of this domain description. The following simple domain description illustrates the scenario:

$$D_3 \left\{ \begin{array}{l} r_1 : \textbf{initially } \mathit{curtainOpen} \\ r_2 : \textbf{initially } \mathit{lightOn} \\ r_3 : \mathit{lookInRoom} \textbf{ causes to know } \mathit{rainOutside} \textbf{ if } \mathit{curtainOpen} \\ r_4 : \mathit{lookInRoom} \textbf{ causes to know } \mathit{boardClean} \textbf{ if } \mathit{lightOn} \end{array} \right.$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

The initial situation Σ_0 of D_3 has four states[4]

$$\begin{aligned}\Sigma_0 = \{ \quad & \{curtainOpen, lightOn\},\\ & \{rainOutside, curtainOpen, lightOn\},\\ & \{boardClean, curtainOpen, lightOn\},\\ & \{rainOutside, boardClean, curtainOpen, lightOn\}\}\end{aligned}$$

There is only one action in D_3, and any model of D_3 applied to the initial situation Σ_0 may behave in one of the following forms:

$$\begin{aligned}\Phi_1(lookInRoom, \Sigma_0) &= \{\{curtainOpen, lightOn\}\}\\ \Phi_2(lookInRoom, \Sigma_0) &= \{\{rainOutside, curtainOpen, lightOn\}\}\\ \Phi_3(lookInRoom, \Sigma_0) &= \{\{boardClean, curtainOpen, lightOn\}\}\\ \Phi_4(lookInRoom, \Sigma_0) &= \{\{rainOutside, boardClean, curtainOpen, lightOn\}\}\end{aligned}$$

Models may differ in how they behave when they are applied to other situations different to Σ_0, but for Σ_0 they must be equal to one of the Φ_i above. Unlike domains in $\mathscr{A}$ in which given an initial situation there is only one model for the domain, our language allows for several models.

Observe, too, that since *lookInRoom* is a sensing action, its occurrence does not change any fluent's value. If we start from Σ_0, and then reach one of the four situations, any new execution of *lookInRoom* will result in the same situation.

To verify that each of the Φ_i can be a partial description of a model of r_3 and r_4, let

$$\begin{aligned}\Sigma_1 = \{ \quad & \{curtainOpen, lightOn\}, \{boardClean, curtainOpen, lightOn\}\}\\ \Sigma_2 = \{ \quad & \{rainOutside, curtainOpen, lightOn\},\\ & \{rainOutside, boardClean, curtainOpen, lightOn\}\}\\ \Sigma_3 = \{ \quad & \{curtainOpen, lightOn\}, \{rainOutside, curtainOpen, lightOn\}\}\\ \Sigma_4 = \{ \quad & \{boardClean, curtainOpen, lightOn\},\\ & \{rainOutside, boardClean, curtainOpen, lightOn\}\}\end{aligned}$$

Note that Σ_1 and Σ_2 are *rainOutside, curtainOpen*-compatible with Σ_0, and that Σ_3 and Σ_4 are *boardClean, lightOn*-compatible with Σ_0, and

$$\begin{aligned}\Phi_1(lookInRoom, \Sigma_0) &= \Sigma_1 \cap \Sigma_3\\ \Phi_2(lookInRoom, \Sigma_0) &= \Sigma_2 \cap \Sigma_3\\ \Phi_3(lookInRoom, \Sigma_0) &= \Sigma_1 \cap \Sigma_4\\ \Phi_4(lookInRoom, \Sigma_0) &= \Sigma_2 \cap \Sigma_4\end{aligned}$$

Note also that none of the situations are f, φ-compatible with Σ_0 by part (1) of Definition 2.3 because there is no knowledge precondition φ of either *rainOutside* or *boardClean* with respect to *lookInRoom* in the domain description D_3 that is false in any of the states in the initial situation Σ_0.

[4] Observe that the initial epistemic state of the robot has always a single situation. To be able to specify more complex initial epistemic states the language must be changed.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

3 $\mathscr{A}_k$: Query language – part I

Given a domain description, an agent would like to ask how the world would be after the execution of a sequence of actions starting from the initial situation. Using actions as in $\mathscr{A}$, queries in $\mathscr{A}_k$ can be of the form

$$\varphi \textbf{ after } [a_1, \ldots, a_n] \tag{6}$$

where φ is a conjunction of fluent literals. The answer to this query will be *yes* (or true) in a domain D if for every model Φ of D the test condition φ is true in the situation

$$\Phi(a_n, \Phi(a_{n-1}, \ldots \Phi(a_1, \Sigma_0) \cdots))$$

i.e. the situation that results after the execution of $a_1, \ldots, a_n$ from the initial situation Σ_0 of D. The answer will be *no* (or false) if for every model Φ of D φ is false in $\Phi(a_n, \Phi(a_{n-1}, \ldots \Phi(a_1, \Sigma_0) \cdots))$. Otherwise the answer will be *unknown*. With this notion we can define an entailment relation between domain descriptions and queries. We say that a domain D entails a query Q, denoted by $D \models Q$, if the answer for Q in D is yes. For example, if we add **initially** *switchOn* to D_1, it can be easily shown that

$$D_1 \models \mathit{bulbFixed} \textbf{ after } [\mathit{turnSwitch}, \mathit{changeBulb}]$$

However, from the original D_1 (even including r_7) there does not exist a sequence of actions α such that $D_1 \models \mathit{bulbFixed}$ **after** α. The inferences from D_1 are conditioned to the output of the sensing action: if the switch is on then the sequence $[\mathit{turnSwitch}, \mathit{changeBulb}]$ will cause the light to be fixed, else the single action $[\mathit{changeBulb}]$ will fix it. Reasoning in the presence of sensing actions requires the projections to be over plans more complex than a simple sequence of actions.

We recursively define a *plan* as follows:[5]

1. an empty sequence denoted by $[]$ is a plan.
2. If a is an action and α is a plan then the concatenation of a with α denoted by $[a|\alpha]$ is also a plan.
3. If φ is a conjunction of fluent literals and α, α_1 and α_2 are plans then $[\textbf{ if } \varphi \textbf{then } \alpha_1|\alpha]$ and $[\textbf{ if } \varphi \textbf{then } \alpha_1 \textbf{else } \alpha_2|\alpha]$ are (conditional) plans.
4. Nothing else is a plan.

Now we redefine a query to be a sentence of the form

$$\varphi \textbf{ after } \alpha \tag{7}$$

Where φ is a test condition (a conjunction of fluent literals) and α is a plan.

[5] We will use the list notation of Prolog to denote sequences.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Example 3.1
(Conditionals) Here we add the knowledge law to D_1 and rename it $D_{1'}$.

$$D_{1'}\left\{\begin{array}{l} r_1\ :\ \textbf{initially}\ \neg burnOut \\ r_2\ :\ \textbf{initially}\ \neg bulbFixed \\ r_3 : changeBulb\ \textbf{causes}\ burnOut\ \textbf{if}\ switchOn \\ r_4 : changeBulb\ \textbf{causes}\ bulbFixed\ \textbf{if}\ \neg switchOn \\ r_5 : turnSwitch\ \textbf{causes}\ switchOn\ \textbf{if}\ \neg switchOn \\ r_6 : turnSwitch\ \textbf{causes}\ \neg switchOn\ \textbf{if}\ switchOn \\ r_7 : checkSwitch\ \textbf{causes to know}\ switchOn\ \textbf{if}\ \neg burnOut \end{array}\right.$$

We can define a conditional plan to fix the bulb:

$$\begin{array}{lll} bulbFixed\ \textbf{after} & [checkSwitch, & \\ & \ \textbf{if}\ \neg switchOn & \textbf{then}\ [changeBulb] \\ & & \textbf{else}\ [turnSwitch, changeBulb]]. \end{array}$$

The above query provides two alternatives for reasoning. The **else** clause is followed if the test condition is false. A conditional can be expanded to a **case** statement in general when reasoning needs to be done along several different sequences of plans. Note that if the conditional plan was attempted before or without the sensing action *checkSwitch*, the query may not succeed because the test condition could evaluate to neither true nor false, but rather unknown. Sensing actions need to be executed before the conditionals to ensure the test conditions will evaluate to either true or false.

3.1 Plan evaluation function and query entailment

To formally define entailment we need to define first the evaluation of a plan in terms of interpretations. In other words, we define how the plan will change an initial situation based on an interpretation.

Definition 3.2
The plan evaluation function Γ_Φ of an interpretation Φ is a function such that for any situation Σ:

1. $\Gamma_\Phi([], \Sigma) = \Sigma$.
2. $\Gamma_\Phi([a|\alpha], \Sigma) = \Gamma_\Phi(\alpha, \Phi(a, \Sigma))$ for any action a.
3. $\Gamma_\Phi([\ \textbf{if}\ \varphi \textbf{then}\ \alpha_1|\alpha], \Sigma) = \Gamma_\Phi(\alpha, \Sigma')$, where

$$\Sigma' = \begin{cases} \Gamma_\Phi(\alpha_1, \Sigma) & \text{if } \varphi \text{ is true in } \Sigma \\ \Sigma & \text{if } \varphi \text{ is false in } \Sigma \\ \emptyset & \text{otherwise} \end{cases}$$

4. $\Gamma_\Phi([\ \textbf{if}\ \varphi \textbf{then}\ \alpha_1 \textbf{else}\ \alpha_2|\alpha], \Sigma) = \Gamma_\Phi(\alpha, \Sigma')$, where

$$\Sigma' = \begin{cases} \Gamma_\Phi(\alpha_1, \Sigma) & \text{if } \varphi \text{ is true in } \Sigma \\ \Gamma_\Phi(\alpha_2, \Sigma) & \text{if } \varphi \text{ is false in } \Sigma \\ \emptyset & \text{otherwise} \end{cases}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Definition 3.3

A query φ **after** α is entailed by a domain description D ($D \models \varphi$ **after** α) iff for every model Φ of D, φ is true in $\Gamma_\Phi(\alpha, \Sigma)$.

It is easy to check that

$$\begin{array}{lll} D_{1'} \models bulbFixed \textbf{ after} & [checkSwitch, & \\ & \textbf{if } \neg switchOn & \textbf{then } [changeBulb] \\ & & \textbf{else } [turnSwitch, changeBulb]]. \end{array}$$

It is easy to see the task will be completed regardless of what model we are in. This is due in part to the combination of the sensing action and the conditional plan.

4 Actions with non-deterministic effects

There are several different reasons why knowledge may be removed from the set of facts known by the agent. There may be decay of the knowledge, difficulty accessing the knowledge, or it may execute an action that makes a particular knowledge no longer valid. In our description, we assume an ideal agent; an agent whose knowledge persists and is not subject to any type of failure or obstacles preventing the quick access of its knowledge. Given this assumption, the first two possibilities for the removal of knowledge are impossible. However, *non-deterministic actions* may remove knowledge. A non-deterministic action is an action in which the outcome cannot be predicted beforehand. An example of such an action with an unpredictable outcome is the toss of a coin. A coin on a table will show either heads or tails. Looking at the coin, one can gain knowledge of which side of the coin shows. Once the action of tossing the coin takes place we are no longer certain of which side will show. The coin will land and will show either heads or tails. We will not know which side shows until we do the sensing action of looking. We describe the removal of knowledge as no longer knowing the truth value of a fluent.

A non-deterministic effect proposition is an expression of the form

$$a \textbf{ may affect } f \textbf{ if } p_1, \ldots, p_n \tag{8}$$

where a is a non-sensing action and f is a fluent. The preconditions $p_1, \ldots, p_n$ are defined as in equation (1). Intuitively the proposition states that the truth value of f may change if a is executed in a situation where $p_1, \ldots, p_n$ is true.

When $n = 0$, equation (10) becomes

$$a \textbf{ may affect } f \tag{9}$$

We now re-define 0-interpretations to take into account non-deterministic actions. A 0-interpretation Φ_0 is a *0-model* of a domain description D iff for every state σ, $\Phi_0(a, \sigma)$ is such that

1. For a fluent f of any effect proposition of the form 'a **causes** f **if** $p_1, \ldots, p_n$' in D, $f \in \Phi_0(a, \sigma)$ if $p_1, \ldots, p_n$ holds in σ,
2. For a fluent literal $\neg f$ of any effect proposition of the form 'a **causes** $\neg f$ **if** $p_1, \ldots, p_n$' in D, the $f \notin \Phi_0(a, \sigma)$ if $p_1, \ldots, p_n$ holds in σ,

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

3. For a fluent f such that there are no effect propositions of the above types, $f \in \Phi_0(a, \sigma)$ if and only if $f \in \sigma$ unless there is a non-deterministic effect proposition of the form 'a **may affect** f **if** $p_1, \ldots, p_n$' for which $p_1, \ldots, p_n$ holds in σ.

Example 4.1
Our agent is ordered at this time to put ice from a bag into cups. The ice in the bag is solid. The agent needs to break the ice into pieces that are able to fit in the cups. The agent decides to drop the bag of ice as a means to complete the task.

$$D_5 \begin{cases} t1 : \textbf{initially}\ \mathit{inHandIceBag} \\ t2 : \textbf{initially}\ \mathit{solidIce} \\ t3 : \textbf{initially}\ \mathit{noDrops} \\ t4 : \mathit{pickUp}\ \textbf{causes}\ \mathit{inHandIceBag}\ \textbf{if}\ \neg \mathit{inHandIceBag} \\ t5 : \mathit{drop}\ \textbf{causes}\ \neg \mathit{inHandIceBag}\ \textbf{if}\ \mathit{inHandIceBag} \\ t6 : \mathit{drop}\ \textbf{may affect}\ \mathit{solidIce}\ \textbf{if}\ \mathit{noDrops} \\ t7 : \mathit{drop}\ \textbf{may affect}\ \mathit{solidIce}\ \textbf{if}\ \mathit{fewDrops} \\ t8 : \mathit{drop}\ \textbf{causes}\ \mathit{fewDrops}\ \textbf{if}\ \mathit{noDrops} \\ t9 : \mathit{drop}\ \textbf{causes}\ \mathit{enoughDrops}\ \textbf{if}\ \mathit{fewDrops} \\ t10 : \mathit{drop}\ \textbf{causes}\ \neg \mathit{solidIce}\ \textbf{if}\ \mathit{enoughDrops} \\ t11 : \mathit{checkIce}\ \textbf{causes to know}\ \mathit{solidIce} \\ t12 : \mathit{putIceInCups}\ \textbf{causes}\ \mathit{iceInCups}\ \textbf{if}\ \neg \mathit{solidIce} \end{cases}$$

This example combines many of the ideas previously presented. Let us examine this domain description to see how this all fits together.

- Rules t1–t3 establish what is initially known in the world. The values of all other fluents are unknown at this time.
- Rules t4 and t5 describe the effect that theactions *Drop* and *pickUp* have on *inHandIceBag*.
- Rules t6 and t7 describe the non-deterministic effect of the action *drop* on the ice.
- Rules t8–t10 are object effect propositions which ensure that the ice will break after no more than three *drops* (i.e. the execution of the action *drop* three times). In the example *noDrops* is equated with 0 drops, *fewDrops* with one drop, and *enoughDrops* with two drops.
- Rule t11 is the sensing action which allows the agent to know whether the ice is broken or not after the execution of the non-deterministic action *drop*. Rule t12 is the goal of the task the agent is to perform.

The non-determinism appears in the action of dropping the bag of ice. Before the action is carried out, the agent knows that the ice is solid. After the non-deterministic action, the agent is no longer certain if the ice is still solid or in pieces. The knowledge of knowing the ice is solid has been removed. The agent can only regain that knowledge by performing a sensing action.

If the robot wants to fill the cup with ice it will iterate the process of dropping the ice until it breaks. A plan to accomplish this goal will look like:

$$\textbf{while}\ \neg \mathit{solidIce}\ \textbf{do}\ [\mathit{drop}, \mathit{pickup}, \mathit{checkIce}], \mathit{putIceInCups}]$$

Adding loops to plans is the topic of the next section.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

5 $\mathscr{A}_k$: Query language – part II

If we allow while-loops in our plan we could verify that D_5 entails the following query:

$$iceInCups \textbf{ after } [\textbf{while } \neg solidIce \textbf{ do } [drop, pickUp, checkIce],\ putIceInCups]$$

Similar to conditional plans, a sensing action is placed before checking the exit condition of the loop. We extend the definition of plans to include loops as follows:

1. An empty sequence denoted by [] is a plan.
2. If a is an action and α is a plan then the concatenation of a with α denoted by $[a|\alpha]$ is also a plan.
3. If φ is a conjunction of fluent literals and α, α_1 and α_2 are plans then $[\textbf{ if } \varphi \textbf{ then } \alpha_1|\alpha]$ and $[\textbf{ if } \varphi \textbf{ then } \alpha_1 \textbf{else } \alpha_2|\alpha]$ are (conditional) plans.
4. If φ is a conjunction of fluent literals and α and α_1 are plans then $[\textbf{ while } \varphi \textbf{ do } \alpha_1|\alpha]$ is also a (routine) plan.
5. Nothing else is a plan.

5.1 Plan evaluation function and query entailment

To extend the definition of entailment to plans with while loops we need to extend the definition of the plan evaluation function Γ_Φ. We will define this function using very elementary tools from denotational semantics for programming languages (as in Chapter 4 of (Davey and Priestley, 1990)). The intuitive idea of the denotational semantics is to associate the execution of a plan (or a program) of the form '**while** φ **do** α' with one of the while-free plans:[6]

$$\begin{array}{l}
\textbf{if } \varphi \textbf{ then } \emptyset \\
\textbf{if } \varphi \textbf{ then } [\alpha,\ \textbf{if } \varphi \textbf{ then } \emptyset] \\
\textbf{if } \varphi \textbf{ then } [\alpha,\ \textbf{if } \varphi \textbf{ then } [\alpha,\ \textbf{if } \varphi \textbf{ then } \emptyset]] \\
\qquad\qquad\vdots
\end{array}$$

If the while-plan terminates then there exists an n such that the nth plan in this infinite sequence computes exactly the same function that the while-plan computes. Moreover, for each $m < n$, the mth plan is an approximation of the computation of the while-plan. If the while-plan does not terminate, any plan in the sequence is an approximation of the while-plan, but none is equivalent since the while-plan computation is infinite. Thus, to define this sequence we start by defining a partial order over the set of functions that map situations into situations. The order will arrange the functions as in the sequence of plans above.

[6] Recall that the situation $\emptyset$ represents inconsistency.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Definition 5.1

Let $\mathscr{E}$ be the set of all situations and $\mathscr{P}$ the set of all total functions f mapping situations into situations, $\mathscr{P} = \{f \mid f : \mathscr{E} \to \mathscr{E}\}$. We say that for any pair of functions $f_1, f_2 \in \mathscr{P}, f_1 \leqslant f_2$ if and only if for any $\Sigma \in \mathscr{E}$ if $f_1(\Sigma) \neq \emptyset$, then $f_1(\Sigma) = f_2(\Sigma)$.

Then, we associate a (continuous) transformation inside this order to each plan α. Informally speaking, the transformation starts with the first plan in the sequence and in each application returns the next element in the sequence. Finally, we will define the meaning of the plan based on the least fix-points of these transformations.

Let $f_\emptyset$ denote the function that maps any situation into the empty situation $\emptyset$.

Definition 5.2

Let α be a plan and Γ a function that maps plans and situations into situations. Let φ be a conjunction of fluent literals. Then, we define the function $\mathscr{F}^{\Gamma}_{\alpha,\varphi} : \mathscr{P} \to \mathscr{P}$ such that for any function $f \in \mathscr{P}$,

$$\mathscr{F}^{\Gamma}_{\alpha,\varphi}(f)(\Sigma) = \begin{cases} \Sigma & \text{if } \varphi \text{ is false in } \Sigma \\ f(\Gamma(\alpha, \Sigma)) & \text{if } \varphi \text{ is true in } \Sigma \\ \emptyset & \text{otherwise} \end{cases}$$

We can define the powers of $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ as follows:

1. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow 0 = f_\emptyset$.
2. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow n+1 = \mathscr{F}^{\Gamma}_{\alpha,\varphi}(\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow n)$.
3. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow \omega = \ldots \mathscr{F}^{\Gamma}_{\alpha,\varphi} \ldots (\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow 0))\ldots)\ldots$, i.e. the infinite composition of $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ applied to $f_\emptyset$.

It can be shown that this power is correctly defined. Proof and a formal definition of powers can be found in Appendix A.

We now extend the definition of the evaluation function Γ_Φ to apply to plans with routines by adding item

5. $\Gamma_\Phi([\,\textbf{while}\ \varphi\ \textbf{do}\ \alpha_1|\alpha], \Sigma) = \Gamma_\Phi(\alpha, \Sigma')$, where $\Sigma' = \mathscr{F}^{\Gamma_\Phi}_{\textbf{if}\,\varphi\,\textbf{then}\,\alpha_1,\varphi} \uparrow \omega$

to Definition 3.2. The definition of entailment remains unchanged. That is, $D \models \varphi$ **after** α iff for every model Φ of D, φ is true in $\Gamma_\Phi(\alpha, \Sigma_0)$.

Example 5.3

$$D_5 \models iceInCups\ \textbf{after}\ [\ \ \textbf{while}\ \neg solidIce\ \ \textbf{do}\ [drop, pickup, checkIce], \\ putIceInCups]$$

5.2 *Plan termination*

Notice that the query above with the while loop could have been written using three nested conditionals. A more natural example will replace rules $t6-10$ with the single rule

drop **may affect** *solidIce*

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

However, in this domain we are not be able to prove termination. The verification of termination is a difficult task, especially for planning. How do we really know that the ice will eventually break? Or how do we know that the cup is filling up? With time the ice will either melt or break, and if we do not place infinitesimally small amounts of ice in the cup the cup will eventually fill up or we will run out of ice. We have simplified the problem in our example by adding $t6$–$t10$. These propositions state that the ice will break with no more than three 'drops'.

We are faced with a similar situation in the following example.

Example 5.4
Consider the following situation. On the floor of a room there are cans. An agent is given an empty bag and instructed to fill the bag with cans. We assume that there are more than enough cans on the floor to fill the bag. The domain description for this task is

$$D_4 \begin{cases} r_1 : \ \textbf{initially}\ \neg bagFull \\ r_2 : drop\ \textbf{causes}\ \neg canInHand \\ r_3 : drop\ \textbf{causes}\ canInBag\ \textbf{if}\ canInHand \\ r_4 : lookInBag\ \ \textbf{causes to know}\ \ bagFull \\ r_5 : pickUp\ \textbf{causes}\ canInHand\ \textbf{if}\ \neg canInHand \end{cases}$$

This task of picking up cans and dropping them into the bag involves the repetition of a small sequence of actions. There is a degree of uncertainty inherent in this task because it is unknown how many cans are needed to fill the bag. Therefore a loop that executes the sequence of actions repeatedly until the task is completed is needed. If the number of cans needed to fill the bag is known beforehand, then the set of actions would be repeated sequentially for those number of times.

A query that we would like to prove is

$$\begin{aligned} bagFull\ \textbf{after}\ [lookInBag,\ \textbf{while}\ \neg bagFull\ \textbf{do}\ [\ \ & pickUpCan, \\ & dropCanInBag, \\ & lookInBag]] \end{aligned}$$

Ideally, we would like to use a routine which could solve any type of task that involves uncertainty of its end. However, each task has its own conditions for termination. For example, filling the volume of a bag differs from finding an unfamiliar store in an unfamiliar area based on the vague directions of a stranger. Do we really know the bag will become full? Or how useful are vague directions such as, 'Just walk down Lincoln Avenue, you can't miss it' when generating a plan. A hole may tear in the bag, or suppose that the stranger who had all the best intentions was mistaken about the location of the store. To ensure termination (either with success or failure) we need to add to our domain descriptions general axioms or constraints. We do not have constraints in $\mathscr{A}_k$ but we may be able to add them by using other extensions of $\mathscr{A}$ such as the one in Baral *et al.* (1997). To address this problem we should first look at the standard techniques of problem verifications such as those founded in Aho and Ullman (1995) or Cousot (1990). These classical ideas have been used by Manna and Waldinger (1987) to prove termination of plans with loops

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

but without sensing actions. For sensing, it might also be useful to consider the techniques described in Gefner and Bonet (1998) to detect loop-termination using probability approaches. However, analysis of the termination of plans is outside the scope of this paper. We will discuss in Section 9 how some of the problems of termination may be addressed in simple situations.

6 Epistemic logic programs

In the past, domain descriptions of dialects of $\mathscr{A}$ have been translated into extended logic programs (Baral and Gelfond, 1997; Gelfond and Lifschitz, 1993). Extended logic programs use two types of negation to represent incomplete information. There is strong or classical negation $\neg$ and negation as failure *not*. The semantics of extended logic programs is defined by a collection of sets of literals called answer sets (Gelfond and Lifschitz, 1991). However, we are required to represent incomplete information that crosses over multiple sets of answer sets. This will be the case in our translation of domain descriptions into logic programs where situations will be closely related to sets of answer sets, and domain descriptions act over epistemic states which are sets of situations. In this case, extended logic programs will no longer be sufficient to codify domain descriptions.

Gelfond has extended disjunctive logic programs to work with sets of sets of answer sets (Gelfond and Lifschitz, 1991). He calls his new programs *epistemic logic programs*. In epistemic logic programs, the language of extended logic programs is expanded with two modal operators K and M. KF is read as 'F is known to be true' and MF is read as 'F may be believed to be true'.

Universal and existential quantifiers are also allowed as well as the epistemic disjunctive '*or*' which the semantics is based on the minimal model semantics associated with disjunctive logic programs (Lobo *et al*., 1992). As an example, when F *or* G is defined as a logic program, its models are exactly F and G. Note that the classical $F \vee G$ cannot be defined as a logic program, because it has models which are not minimal.

In the rest of this section, we will review the syntax and the semantics of the subclass of epistemic logic programs that will be required to represent our domain descriptions. Readers interested in more details about epistemic logic programs are referred to Gelfond (1994).

The semantics of an epistemic logic program is defined by pairs $\langle A, W\rangle$. A is a collection of sets of ground literals called the set of *possible beliefs*. Each set in A can be indexed as $A = \{A_1 \dots A_n\}$. W is a set in A called the *working set of beliefs*. To define the semantics, we restrict our formulas to be: ground literals, a ground literal preceded by a modal operator, a ground literal preceded by a modal operator and $\neg$, or a conjunction of such formulas. The truth of a formula F in $\langle A, W\rangle$ is denoted by $\langle A, W\rangle \models F$ and the falsity by $\langle A, W\rangle =\!\mid F$, and are defined as follows:

$$\langle A, W\rangle \models F \text{ iff } F \in W, \text{ when } F \text{ is a ground atom.}$$

$$\langle A, W\rangle \models KF \text{ iff } \langle A, A_i\rangle \models F,\ \forall A_i \in A.$$

$$\langle A, W\rangle \models F \wedge G \text{ iff } \langle A, W\rangle \models F \text{ and } \langle A, W\rangle \models G.$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

$\langle A, W \rangle \models \neg F$ iff $\langle A, W \rangle \;{=}\!| \; F$.

$\langle A, W \rangle \;{=}\!|\; F$ iff $\neg F \in W$, when F is a ground atom.

$\langle A, W \rangle \;{=}\!|\; KF$ iff $\langle A, W \rangle \not\models KF$

$\langle A, W \rangle \;{=}\!|\; F \wedge G$ iff $\langle A, W \rangle \;{=}\!| F$ or $\langle A, W \rangle \;{=}\!| G$.

$\langle A, W \rangle \;{=}\!|\; \neg F$ iff $\langle A, W \rangle \models F$.

$\langle A, W \rangle \models F$ *or* G iff $\langle A, W \rangle \models \neg(\neg F \wedge \neg G)$

Note that when a formula G is of the form KF, or $\neg KF$, its evaluation in $\langle A, W \rangle$ does not depend on W. Thus, we will write $A \models G$ or $A \;{=}\!|\; G$. Moreover, the evaluation of object formulas does not depends on A. If G is objective we sometimes write $W \models G$ or $W \;{=}\!|\; G$.

An epistemic logic program is a collection of rules of the form

$$F_1 \; or \ldots or \; F_k \leftarrow G_1, \ldots, G_m, \textbf{not } F_{m+1}, \ldots, \textbf{not } F_n \tag{10}$$

where $F_1 \ldots F_k$ and $F_{m+1} \ldots F_n$ are (not necessarily ground) objective literals (without K or M) and $G_1 \ldots G_m$ are (not necessarily ground) subjective (with K or M) or objective literals.

Let Π be an epistemic logic program without variables, **not**, or modal operators. A set W of ground literals is a belief set of Π if it is a minimal set of ground literals, satisfying the following properties:

1. $W \models F$ for every rule $F \leftarrow G_1 \ldots G_m$ in Π which $W \models G_1 \wedge \ldots \wedge G_m$.
2. If there is a pair of complementary literals, i.e F *and* $\neg F$, in W then W is the set of all literals.

Let Π be an epistemic logic program with **not** and variables but does not contain any modal operator. Let $Ground(\Pi)$ be the epistemic logic program that is obtained from Π by replacing each rule in Π with all its ground instances. Let W be a set of ground literals (literals in W and Π are from the same language). Π^W is obtained from Π by removing from $Ground(\Pi)$

1. All the rules which contain formulas of the form **not** G such that $W \models G$.
2. All occurrences of formulas of the form **not** G from the remaining rules.

W is a belief set of Π if and only if W is a belief set of Π^W.

Let Π be any epistemic logic program, and $\mathbf{A}$ a collection of sets of literals. $[\Pi]_{\mathbf{A}}$ is the epistemic logic program obtained by removing from $Ground(\Pi)$

1. All rules with formulas of the form G such that G contains M or K, and $\mathbf{A} \not\models G$,
2. All occurrences of formulas containing M or K from the remaining rules.

A set $\mathbf{A}$ is a *world view* of Π if $\mathbf{A}$ is the collection of all belief sets of $[\Pi]_{\mathbf{A}}$. A world view of Π is consistent if it does not contain the belief set of all literals. An epistemic logic program is consistent if it has at least one consistent non-empty world view. In epistemic logic programs the only *working sets of beliefs* that are considered are

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

world views and the *possible belief* is always a member of the working set under consideration (i.e. a belief set).

Let Π be an epistemic logic program and $\mathbf{A}$ be a world view of Π. A literal L is true in $\mathbf{A}$ iff for every ground instance F of L, $\langle \mathbf{A}, A_i \rangle \models F$ for all A_i in $\mathbf{A}$. F is true in Π, denoted by $\Pi \models F$, iff $\mathbf{A} \models F$ for every world view $\mathbf{A}$ of Π.

Example 6.1
The epistemic program

1. $q(a) \leftarrow \neg K p(a),$
2. $p(a) \leftarrow \neg K q(a),$

has two world views

$$\{\{p(a)\}\} \qquad \{\{q(a)\}\}$$

In the first world view $Kp(a)$ is true and $Kq(a)$ is true in the second.

The epistemic program

1. $q(a)$ *or* $q(b)$.
2. $p(a) \leftarrow \neg K q(a).$

has one world view

$$\{\{p(a), q(a)\}, \{p(a), q(b)\}\}$$

Note that $Kq(a)$ is not true in this world view because $q(a)$ is not member of the second belief set. The main intuition to have when reading a formula of the form KF is that it will be true iff F is true in every belief set of the program.

7 Translation to epistemic logic programs

In this section we start with a sound and complete translation of *simple* domain descriptions into epistemic logic programs. This will let us explain the logic program rules under the simple scenario and will make clear the rules for the general case.

Our epistemic logic programs will use variables of three sorts: *situation* variables denoted by S or S' possibly indexed, *fluent* variables denoted by F or F' possibly indexed, *action* variables denoted by A or A' possibly indexed, and the special situation constant s_0 that represents the initial situation. We will also have a constant symbol for each fluent symbol f in the language, and we add the constant symbol $\bar{f}$ to represent $\neg f$. For simplicity we will denote the fluent literal constants by the fluent literal they represent. We will also add the special constant symbol *true* to the set of fluent literal constants.

7.1 The domain independent translation

We start by first giving the rules for inertia. These rules encode that a fluent remains unchanged if no actions that affect the fluent is executed. Whenever a fluent literal appears as an argument in a predicate, it is representing a corresponding constant in the program. For any fluent literal l, if $l = \neg f$, $\bar{l}$ will denote f in the program.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

For every fluent literal f there is an inertia rule of the form:

$$holds(f, res(A, S)) \leftarrow holds(f, S), \textbf{not } ab(\bar{f}, A, S).$$

For every fluent symbol f there is an or-classicalization rule of the form

$$holds(f, s_0) \ or \ holds(\bar{f}, s_0).$$

The above rule states that our belief sets are complete in the sense that either $holds(f, s_0)$ or $holds(\bar{f}, s_0)$ must be true since $f \vee \neg f$ is a tautology in every state. Note that because of the minimal model semantics interpretation of the '*or*' we will not have both $holds(f, s_0)$ or $holds(\bar{f}, s_0)$ holding simultaneously.

We will also have two more domain independent rules that we will call rules of *suppression*:

$$holds(true, res(A, S)) \leftarrow holds(true, S)$$
$$holds(F, S) \leftarrow holds(true, S)$$

These rules will be used to implement compatibility. For example, if a situation $\Sigma = \{\sigma_1, \sigma_2\}$ with two states is split into two situations $\Sigma_1 = \{\sigma_1\}$ and $\Sigma_2 = \{\sigma_2\}$, for compatibility after the execution of a sensing action, Σ_1 will be generated by suppressing σ_2 from Σ using these rules. How this is accomplished will become apparent when we introduce the domain dependent rules produced by the knowledge laws.

7.2 The domain dependent translation

Value propositions of the form '**initially** f' are translated into

$$holds(f, s_0)$$

The translation of effect propositions of the form 'a **causes** f **if** $p_1, \ldots, p_n$' is the standard translation for effect propositions introduced by Gelfond and Lifschitz in Gelfond and Lifschitz (1993) for $\mathscr{A}$. The translation produces two rules. The first one is

$$holds(f, res(a, S)) \leftarrow holds(p_1, S), \ldots, holds(p_n, S)$$

It allows us to prove that f will hold after the result of the execution of a if preconditions are satisfied. The second rule is

$$ab(f, a, S) \leftarrow holds(p_1, S), \ldots, holds(p_n, S), \textbf{not } holds(true, res(a, S))$$

where the predicate $ab(f, a, S)$ disables the inertia rule in the cases where f can be affected by a.

We will introduce the domain dependent translation of knowledge laws using the following domain description.

Example 7.1

$$D_1^0 \left\{ \begin{array}{l} r_1 \ : \ \textbf{initially} \ \neg bulbFixed \\ r_2 \ : checkSwitch \ \textbf{causes to know} \ switchOn \ \textbf{if} \ \neg burnOut \end{array} \right.$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

In this example the initial situation is:

$$\begin{aligned}\Sigma = \; & \{\{burnOut, \neg bulbFixed, switchOn\},\\ & \{burnOut, \neg bulbFixed, \neg switchOn\},\\ & \{\neg burnOut, \neg bulbFixed, switchOn\},\\ & \{\neg burnOut, \neg bulbFixed, \neg switchOn\}\}\end{aligned}$$

after the robot executes the action *checkSwitch* we will have the following resulting situations:

$$\begin{aligned}\Phi_1(checkSwitch, \Sigma) = \{ \quad & \{\neg burnOut, \neg bulbFixed, switchOn\}\}\\ \Phi_2(checkSwitch, \Sigma) = \{ \quad & \{\neg burnOut, \neg bulbFixed, \neg switchOn\}\}\\ \Phi_3(checkSwitch, \Sigma) = \{ \quad & \{burnOut, \neg bulbFixed, switchOn\},\\ & \{burnOut, \neg bulbFixed, \neg switchOn\}\}\end{aligned}$$

These correspond to the three $(switchOn, \neg burnOut)$-compatible sub-sets of Σ (see Definition 2.3). Note also that

$$\begin{aligned}&\Phi_1(checkSwitch, \Phi_1(checkSwitch, \Sigma)) = \Phi_1(checkSwitch, \Sigma)\\ &\Phi_2(checkSwitch, \Phi_2(checkSwitch, \Sigma)) = \Phi_2(checkSwitch, \Sigma)\\ &\Phi_3(checkSwitch, \Phi_2(checkSwitch, \Sigma)) = \Phi_3(checkSwitch, \Sigma)\end{aligned}$$

Our logic program translation of this domain will have three world views, one corresponding to each of the transition functions Φ_1 Φ_2, and Φ_3. Φ_1 is depicted on the left-hand side of Figure 2, Φ_2 on the right hand side and Φ_3 in the middle.

The world view associated with Φ_1 on the left-hand side of the figure will have four belief sets. One will contain the union of the two sets

$$W^1_{s_0} = \{holds(\overline{burnOut}, s_0), holds(\overline{bulbFixed}, s_0), holds(switchOn, s_0)\}$$

and

$$\begin{aligned}W^1_{res(cs,s_0)} = \{ \quad & holds(\overline{burnOut}, res(checkSwitch, s_0)),\\ & holds(\overline{bulbFixed}, res(checkSwitch, s_0),\\ & holds(switchOn, res(checkSwitch, s_0))\}\end{aligned}$$

This union represents the fact that $\{\neg burnOut, \neg bulbFixed, switchOn\}$ is an initial state (encoded $W^1_{s_0}$) and that the same set is also a state in $\Phi_1(checkSwitch, \Sigma)$ (encoded in $W^1_{res(cs,s_0)}$). The rest of the literals in W^1 are the same as in $W^1_{res(cs,s_0)}$ except that the situation constant in each literal is replaced by situation constants of the form $res(checkSwitch, res(\ldots, res(checkSwitch, s_0)\ldots))$ representing that the state remains the same after any number of applications of the action *checkSwitch* to the state (the loop arc on the left of the figure).

The second belief set will contain

$$W^2_{s_0} = \{holds(\overline{burnOut}, s_0),\ holds(\overline{bulbFixed}, s_0),\ holds(\overline{switchOn}, s_0)\}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

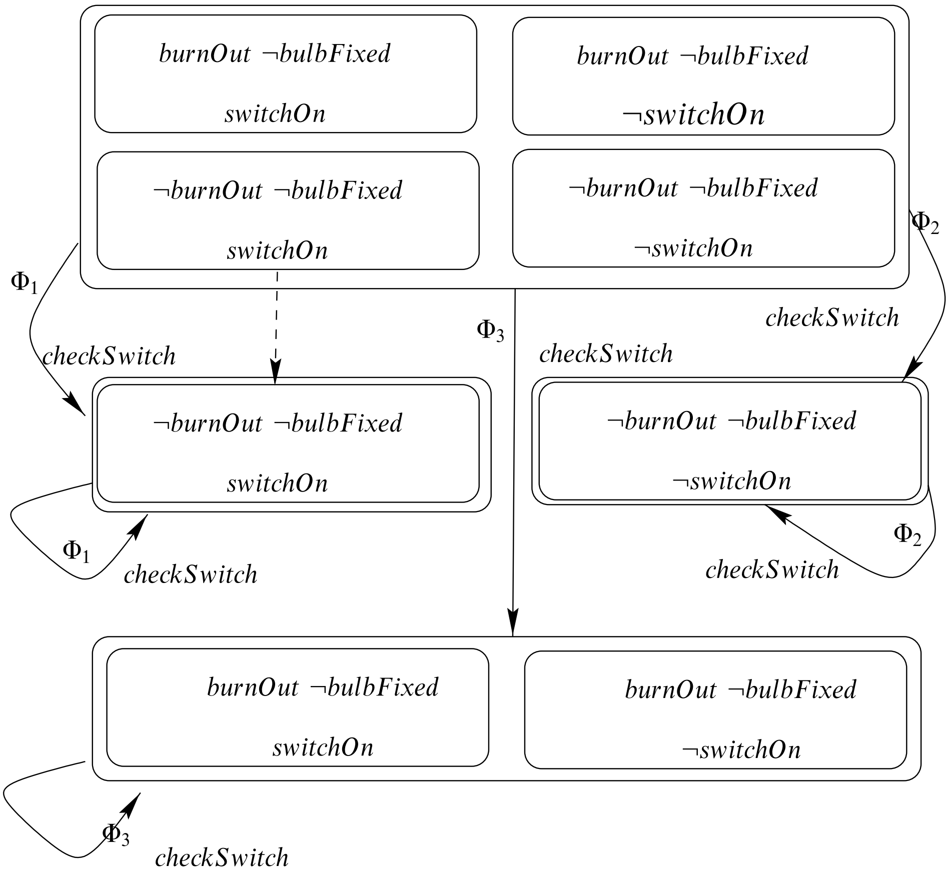
burnOut ¬bulbFixed switchOn
burnOut ¬bulbFixed ¬switchOn
¬burnOut ¬bulbFixed switchOn
¬burnOut ¬bulbFixed ¬switchOn
Φ1
checkSwitch
Φ2
checkSwitch
Φ3
checkSwitch
¬burnOut ¬bulbFixed switchOn
¬burnOut ¬bulbFixed ¬switchOn
Φ1
checkSwitch
Φ2
checkSwitch
burnOut ¬bulbFixed switchOn
burnOut ¬bulbFixed ¬switchOn
Φ3
checkSwitch

Fig. 2

representing that $\{\neg burnOut, \neg bulbFixed, \neg switchOn\}$ is also an initial state. However, this state is not part of $\Phi_1(checkSwitch, \Sigma)$. Then, we need to suppress this state from the world view. We will do that by adding the set

$$
\begin{aligned}
W^2_{res(cs,s_0)} \quad = \{ \quad & holds(\overline{burnOut}, res(checkSwitch, s_0)), \\
& holds(\overline{bulbFixed}, res(checkSwitch, s_0), \\
& holds(\overline{switchOn}, res(checkSwitch, s_0)) \\
\} \quad & \bigcup \\
\{ \quad & holds(burnOut, res(checkSwitch, s_0)), \\
& holds(bulbFixed, res(checkSwitch, s_0), \\
& holds(switchOn, res(checkSwitch, s_0)), \\
& holds(true, res(checkSwitch, s_0))\}
\end{aligned}
$$

to the belief set. Actually, we will have in the domain dependent translation a rule that adds $holds(true, res(checkSwithc, s_0))$, and the second domain independent suppression rule will add the rest. The rest of the literals in W^2 are the same as in $W^2_{res(cs,s_0)}$ except that the situation constant is replaced by situation constants of the form $res(checkSwitch, res(\ldots, res(checkSwitch, s_0)\ldots))$ representing that the state

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

remains suppressed in the result of applying the action *checkSwitch* to the state. This is the effect of the first domain independent suppression rule.

The other two belief sets W^3 and W^4 are similar to W^2:

$$\begin{aligned} W^3_{s_0} &= \{holds(burnOut, s_0),\ holds(\overline{bulbFixed}, s_0),\ holds(switchOn, s_0)\} \\ W^4_{s_0} &= \{holds(burnOut, s_0),\ holds(\overline{bulbFixed}, s_0),\ holds(\overline{switchOn}, s_0)\} \end{aligned}$$

The rest of W^3 and W^4 is exactly as in W^2 since the states they represent are also suppressed from the result.

Note that both $holds(\overline{f}, res(checkSwitch, s_0))$ and $holds(f, res(checkSwitch, s_0))$ are members of the belief sets W^2, W^3 and W^4, for any fluent f. Therefore, for any fluent literal g, the proof of $holds(g, res(checkSwitch, s_0)$ in the world view is not affected by these belief sets. The consequence is that we are ignoring three states after the execution of *checkSwitch* under the model Φ_1.

There are two more world views that correspond to the transitions in the middle and on the right-hand side of the figure. The definition is very similar to the first world view. There are four belief sets in the middle, two of them suppressing initial states, and four belief sets in the last world view, three of them suppressing initial states.

Thus, the domain dependent translation of D^0_1 will be:

$$holds(\overline{bulbFixed}, s_0) \leftarrow$$

Rule x_1 is the translation of rule r_1. The rest of the rules correspond to the different suppression cases since states that are not suppressed by the transition will be moved to the next situation by the domain independent rule of inertia. Take for example, Φ_1:

$$\Phi_1(checkSwitch, \Sigma) = \{\sigma \in \Sigma | \sigma \models \neg burnOut, switchOn \in \sigma\}$$

Hence, we would like to suppress two kinds of states. 1) States where $\neg switchOn$ is true, and 2) States where *burnOut* is true. The rule for the first case is:

$$\begin{aligned} holds(true, res(checkSwitch, S)) \quad \leftarrow \quad & K holds(switchOn, res(checkSwitch, S)) \\ & K holds(\overline{burnOut}, res(checkSwitch, S)), \\ & holds(\overline{switchOn}, S) \end{aligned} \tag{11}$$

The first two literals in the body of the rule verify that we are in the case of Φ_1, that is, both *switchOn* and $\neg burnOut$ are true in every state of the resulting situation (i.e. the two literals $K holds(switchOn, res(checkSwitch, S))$ and $K holds(\overline{burnOut}, res(checkSwitch, S))$ are true). The last predicate checks that we are suppressing the state where $\neg switchOn$ is true in the current situation (i.e. $holds(\overline{switchOn}, S)$).

The rule for the second case is very similar. We only need to change the last literal to indicate that we are suppressing the state where *burnOut* is true (i.e. $holds(burnOut, S)$):

$$\begin{aligned} holds(true, res(checkSwitch, S)) \quad \leftarrow \quad & K holds(switchOn, res(checkSwitch, S)) \\ & K holds(\overline{burnOut}, res(checkSwitch, S)), \\ & holds(burnOut, S) \end{aligned} \tag{12}$$

Let us look now at Φ_2:

$$\Phi_2(checkSwitch, \Sigma) = \{\sigma \in \Sigma | \sigma \models \neg burnOut, switchOn \notin \sigma\}$$

We also suppress two kinds of states. 1) States where *switchOn* is true, and 2) States where *burnOut* is true. We need to check that $\neg switchOn$ and $\neg burnOut$ are true in every state of the resulting situation (i.e. $Kholds(\overline{switchOn}, res(checkSwitch, S))$ and $Kholds(\overline{burnOut}, res(checkSwitch, S))$ are true) to verify that we are in the case of Φ_2. The rules for the cases are:

$$\begin{array}{lll} holds(true, res(checkSwitch, S)) & \leftarrow & Kholds(\overline{switchOn}, res(checkSwitch, S)) \\ & & Kholds(\overline{burnOut}, res(checkSwitch, S)), \\ & & holds(switchOn, S) \\ holds(true, res(checkSwitch, S)) & \leftarrow & Kholds(\overline{switchOn}, res(checkSwitch, S)) \\ & & Kholds(\overline{burnOut}, res(checkSwitch, S)), \\ & & holds(burnOut, S) \end{array} \qquad (13)$$

For Φ_3 all the states where $\neg burnOut$ holds (i.e. $holds(\overline{burnOut}, S)$) should be suppressed since

$$\Phi_3(checkSwitch, \Sigma) = \{\sigma \in \Sigma | \sigma \not\models \neg burnOut\}$$

To verify that we are in the case of Φ_3 we need to check there is at least one state in the result where *burnOut* holds (i.e. $\neg Kholds(\overline{burnOut}, res(checkSwithc, S))$). The rule for this case is:

$$\begin{array}{lll} holds(true, res(checkSwitch, S)) & \leftarrow & \neg Kholds(\overline{burnOut}, res(checkSwitch, S)) \\ & & holds(\overline{burnOut}, S) \end{array} \qquad (14)$$

There is a condition that must be added to all the rules. The condition is that if the fluent *switchOn* is already known in the original situation (for example if we have **initially** $\neg switchOn$) then none of the states is suppressed from the situations. In other words, the rules above applied only if *switchOn* is unknown. To check that this is the case we must add to the body of each rule the literals $\neg Kholds(switchOn, S)$ and $\neg Kholds(\overline{switchOn}, S)$. These literals are not required in this particular example but it must be part of the general case.

In general, knowledge laws of the form 'a **causes to know** f **if** $p_1, \ldots, p_n$' are translated into the rules:

$$\begin{array}{lll} holds(true, res(a, S)) & \leftarrow & \neg Kholds(f, S), \neg Kholds(\bar{f}, S), \\ & & \neg Kholds(p_1, res(a, S)), \\ & & holds(p_1, S), \ldots, holds(p_n, S) \\ & \vdots & \\ holds(true, res(a, S)) & \leftarrow & \neg Kholds(f, S), \neg Kholds(\bar{f}, S), \\ & & \neg Kholds(p_n, res(a, S)), \\ & & holds(p_1, S), \ldots, holds(p_n, S) \end{array} \qquad (15)$$

$$\begin{array}{lll} holds(true, res(a, S)) & \leftarrow & \neg Kholds(f, S), \neg Kholds(\bar{f}, S), Kholds(\bar{f}, res(a, S)), \\ & & Kholds(p_1, res(a, S)), \ldots, Kholds(p_n, res(a, S)), \\ & & holds(f, S) \end{array} \qquad (16)$$

$$\begin{aligned}
holds(true, res(a,S)) &\leftarrow \neg K holds(f,S), \neg K holds(\bar{f},S), K holds(\bar{f}, res(a,S)),\\
&\quad K holds(p_1, res(a,S)), \ldots, K holds(p_n, res(a,S)),\\
&\quad holds(\bar{p}_1, S)\\
&\vdots\\
holds(true, res(a,S)) &\leftarrow \neg K holds(f,S), \neg K holds(\bar{f},S), K holds(\bar{f}, res(a,S)),\\
&\quad K holds(p_1, res(a,S)), \ldots, K holds(p_n, res(a,S)),\\
&\quad holds(\bar{p}_n, S) \qquad (17)\\
holds(true, res(a,S)) &\leftarrow \neg K holds(f,S), \neg K holds(\bar{f},S), K holds(f, res(a,S)),\\
&\quad K holds(p_1, res(a,S)), \ldots, K holds(p_n, res(a,S)),\\
&\quad holds(\bar{f}, S) \qquad (18)\\
holds(true, res(a,S)) &\leftarrow \neg K holds(f,S), \neg K holds(\bar{f},S), K holds(f, res(a,S)),\\
&\quad K holds(p_1, res(a,S)), \ldots, K holds(p_n, res(a,S)),\\
&\quad holds(\bar{p}_1, S)\\
&\vdots\\
holds(true, res(a,S)) &\leftarrow \neg K holds(f,S), \neg K holds(\bar{f},S), K holds(f, res(a,S)),\\
&\quad K holds(p_1, res(a,S)), \ldots, K holds(p_n, res(a,S)),\\
&\quad holds(\bar{p}_n, S) \qquad (19)
\end{aligned}$$

We have added to every rule the condition $\neg K holds(f,S), \neg K holds(\bar{f},S)$. None of these rules apply if f is currently known. In this case, by inertia everything stays the same after the execution of the sensing action a. Assume now that neither f nor $\bar{f}$ holds in the 'situation' S. Thus, according to the definition of compatibility (Definition 2.3), we would have three types of world views. (1) One type for which we can find a p_i for each of the belief sets such $holds(p_i, res(a,S))$ does not hold; (2) World views in which $K holds(\bar{f}, res(a,S))$ and every $K holds(p_i, res(a,S))$ hold; (3) World views in which $K holds(f, res(a,S))$ and every $K holds(p_i, res(a,S))$ hold.

Intuitively, to capture these three cases the logic programming rules will suppress the belief set that breaks the rules. To suppress a belief set in the situation $res(a,S)$ the rules will add $holds(true, res(a,S))$ to the belief set, and this atom together with the second suppression rule will add $holds(l, res(a,S))$, for every fluent literal l. Recall that the effect of having every literal hold for a particular situation in a belief set is that the belief set can be ignored when checking if the literal holds in the world view. Case (1) is captured by the first set of rules (15). Case (2) is captured by rule (16) and the set of rules (17). Case (3) is captured by rule (18) and the set of rules (19).

A non-deterministic effect proposition of the form 'a **may affect** f **if** $p_1, \ldots, p_n$' is translated into

$$\begin{aligned}
holds(f, res(a,S)) &\leftarrow \textbf{not } holds(\bar{f}, res(a,S)),\\
&\quad holds(p_1, S), \ldots, holds(p_n, S)\\
holds(\bar{f}, res(a,S)) &\leftarrow \textbf{not } holds(f, res(a,S)),\\
&\quad holds(p_1, S), \ldots, holds(p_n, S)
\end{aligned}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

$$\begin{aligned}
ab(f,a,S) \leftarrow\ & \textbf{not}\ holds(\bar{f},res(a,S)), holds(p_1,S),\ldots,holds(p_n,S),\\
& \textbf{not}\ holds(true,S)\\
ab(\bar{f},a,S) \leftarrow\ & \textbf{not}\ holds(f,res(a,S)), holds(p_1,S),\ldots,holds(p_n,S),\\
& \textbf{not}\ holds(true,S)
\end{aligned}$$

To illustrate this translation, we use Rule t_6 from domain description D_5, and show how it will work from the initial situation s_0. We also include the translation for t_8, $holds(noDrops,s_0)$ and $holds(solidIce,s_0)$ along with the inertia rule to get the following program. Notice that the suppression rules do not apply here since we are not considering any knowledge laws:

$$\begin{aligned}
holds(solidIce,res(drop,s_0)) \leftarrow\ & \textbf{not}\ holds(\overline{solidIce},res(drop,s_0)),\\
& holds(noDrops,s_0).\\
holds(\overline{solidIce},res(drop,s_0)) \leftarrow\ & \textbf{not}\ holds(solidIce,res(drop,s_0)),\\
& holds(noDrops,s_0).
\end{aligned}$$

$$\begin{aligned}
ab(solidIce,drop,s_0) \leftarrow\ & \textbf{not}\ holds(\overline{solidIce},res(drop,s_0)),\\
& holds(noDrops,s_0), \textbf{not}\ holds(true,s_0).\\
ab(\overline{solidIce},drop,s_0) \leftarrow\ & \textbf{not}\ holds(solidIce,res(drop,s_0)),\\
& holds(noDrops,s_0), \textbf{not}\ holds(true,s_0).
\end{aligned}$$

$$\begin{aligned}
& holds(\overline{noDrops},res(drop,s_0)) \leftarrow holds(noDrops,s_0).\\
& ab(\overline{noDrops},drop,s_0) \leftarrow holds(noDrops,s_0).
\end{aligned}$$

$$\begin{aligned}
& holds(noDrops,s_0) \leftarrow\\
& holds(solidIce,s_0) \leftarrow
\end{aligned}$$

$$\begin{aligned}
holds(noDrops,res(drop,s_0)) \leftarrow\ & holds(noDrops,s_0),\\
& \textbf{not}\ ab(\overline{noDrops},drop,s_0).\\
holds(\overline{noDrops},res(drop,s_0)) \leftarrow\ & holds(\overline{noDrops},s_0),\\
& \textbf{not}\ ab(noDrops,drop,s_0).\\
holds(solidIce,res(drop,s_0)) \leftarrow\ & holds(solidIce,s_0),\\
& \textbf{not}\ ab(\overline{solidIce},drop,s_0).\\
holds(\overline{solidIce},res(drop,s_0)) \leftarrow\ & holds(\overline{solidIce},s_0),\\
& \textbf{not}\ ab(solidIce,drop,s_0).
\end{aligned}$$

The program only has objective formulas. Thus, its semantics is given by its world view which consists of *belief sets* (belief sets are the same as answer sets in extended logic programs). The world view W of the above program is $W = \{B_1,\ B_2\}$

$$\begin{aligned}
B_1 = \{\ & holds(solidIce,res(drop,s_0)),\ ab(solidIce,drop,s_0),\\
& holds(solidIce,s_0),\ holds(\overline{noDrops},res(drop,s_0)),\\
& ab(\overline{noDrops},drop,s_0),\ holds(noDrops,s_0)\\
\}\ &\\
B_2 = \{\ & holds(\overline{solidIce},res(drop,s_0)),\ ab(\overline{solidIce},drop,s_0),\\
& holds(solidIce,s_0),\ holds(\overline{noDrops},res(drop,s_0)),\\
& ab(\overline{noDrops},drop,s_0), holds(noDrops,s_0)\\
\}\ &
\end{aligned}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Notice that the query $holds(noDrops, res(drop, s_0))$ evaluates to true for the above belief sets. If we were to ask the queries $holds(solidIce, res(drop, s_0))$ or $holds(\overline{solidIce}, res(drop, s_0))$, we see neither would be able to produce an answer of *yes* or *no*. Both queries' answer is *unknown*.

The recursion through negation provides the desired effect of two possible interpretations for the effect of a in f (note that the two first rules of the example have the form $c \leftarrow$ **not** b and $b \leftarrow$ **not** c and this program has two answer sets, one is $\{c\}$ and the other is $\{b\}$).

The *translation* of a domain D is defined as the union of the domain dependent and domain independent rules.

7.3 General domains

The assumption that we made for simple domains was that for any sensing action a_s and fluent f there is at most one knowledge law of the form

$$a_s \textbf{ causes to know } f \textbf{ if } p_1, \ldots, p_n \tag{20}$$

Suppose now we have the following domain

$$D \begin{cases} r_1 : lookInRoom \textbf{ causes to know } boardClean \textbf{ if } curtainOpen \\ r_2 : lookInRoom \textbf{ causes to know } boardClean \textbf{ if } lightOn \end{cases}$$

and assume we start with the following situation

$$\Sigma = \{\{boardClean, curtainOpen\}, \{boardClean, lightOn\}, \{boardClean\}, \{\}\}$$

There is one model Φ_1 that will result in the states where the fluent *boarClean* is true and the knowledge precondition $curtainOpen \vee lightOn$ of *boardClean* with respect to *lookInRoom* is also true (this corresponds to the third case of compatibility).

$$\Phi_1(lookInRoom, \Sigma) = \{\{boardClean, curtainOpen\}, \{boardClean, lightOn\}\}$$

We will need a suppression rule similar to the rules in group (19) of the translation of simple domains. The rule will be something like

$$\begin{aligned} holds(true,\quad & res(lookInRoom, S)) \\ \leftarrow \quad & \neg K holds(boardClean, S), \neg K holds(\overline{boardClean}, S), \\ & K holds(boardClean, res(lookInRoom, S)), \\ & \text{“}K holds(curtainOpen \vee lightOn, res(lookInRoom, S))\text{”}, \\ & holds(\overline{curtainOpen}, S), holds(\overline{lightOn}, S), \end{aligned}$$

The question is how to encode “$K holds(curtainOpen \vee lightOn, \ldots)$”? We will do it by adding two rules to the program

$$\begin{aligned} & holds(p^{lookInRoom}_{boardClean}, S) \leftarrow holds(curtainOpen, S) \\ & holds(p^{lookInRoom}_{boardClean}, S) \leftarrow holds(lightOn, S) \end{aligned}$$

Now the disjunction can be replaced by “$K holds(p^{lookInRoom}_{boardClean}, res(lookInRoom, S))$”. The symbol $p^{lookInRoom}_{boardClean}$ is a new constant symbol not appearing anywhere else in the program. We complete the program with the rule

$$holds(\overline{p^{lookInRoom}_{boardClean}}, S) \leftarrow \textbf{not } holds(p^{lookInRoom}_{boardClean}, S)$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

and the translation becomes

$$\begin{array}{lll} holds(true, & res(lookInRoom, S)) & \\ & \leftarrow & \neg K holds(boardClean, S), \neg K holds(\overline{boardClean}, S), \\ & & K holds(boardClean, res(lookInRoom, S)), \\ & & K holds(p^{lookInRoom}_{boardClean}, res(lookInRoom, S)), \\ & & holds(\overline{p^{lookInRoom}_{boardClean}}, S) \end{array}$$

In general, if D is a domain description (not necessarily simple) then, for any sensing action a and any fluent f, if $\varphi_1 \vee \ldots \vee \varphi_m$ with $\varphi_i = p^i_1 \wedge \ldots \wedge p^i_{k_i}, i = 1, \ldots, m$, is the knowledge precondition of f with respect to a in the domain D, we will have a new constant symbol p^a_f in the language of the logic program. Then for the knowledge laws:

$$\begin{array}{l} a \textbf{ causes to know } f \textbf{ if } \varphi_1 \\ \vdots \\ a \textbf{ causes to know } f \textbf{ if } \varphi_m \end{array}$$

the domain dependent translation will have the rules:

$$\begin{array}{rcll} holds(p^a_f, S) & \leftarrow & holds(p^1_1, S), \ldots, holds(p^1_{k_1}, S) & \\ & \vdots & & \\ holds(p^a_f, S) & \leftarrow & holds(p^m_1, S), \ldots, holds(p^m_{k_m}, S) & (21) \\ holds(\overline{p^a_f}, S) & \leftarrow & \textbf{not } holds(p^a_f, S) & (22) \\ holds(true, res(a, S)) & \leftarrow & \neg K holds(f, S), \neg K holds(\bar{f}, S), & \\ & & \neg K holds(p^a_f, res(a, S)), & \\ & & holds(p^a_f, S) & (23) \\ holds(true, res(a, S)) & \leftarrow & \neg K holds(f, S), \neg K holds(\bar{f}, S), K holds(\bar{f}, res(a, S)), & \\ & & K holds(p^a_f, res(a, S)), holds(f, S) & (24) \\ holds(true, res(a, S)) & \leftarrow & \neg K holds(f, S), \neg K holds(\bar{f}, S), K holds(\bar{f}, res(a, S)), & \\ & & K holds(p^a_f, res(a, S)), & \\ & & holds(\overline{p^a_f}, S) & (25) \\ holds(true, res(a, S)) & \leftarrow & \neg K holds(f, S), \neg K holds(\bar{f}, S), K holds(f, res(a, S)), & \\ & & K holds(p^a_f, res(a, S)), holds(\bar{f}, S) & (26) \\ holds(true, res(a, S)) & \leftarrow & \neg K holds(f, S), \neg K holds(\bar{f}, S), K holds(f, res(a, S)), & \\ & & K holds(p^a_f, res(a, S)), & \\ & & holds(\overline{p^a_f}, S) & (27) \end{array}$$

The set of rules (15) corresponds to rule (23). Rule (16) corresponds to rule (24). The set of rules (17) correspond to rule (25). Rule (18) corresponds to rule (26) and rule (19) corresponds to rule (27).

7.4 Query translation

To answer queries in the epistemic logic program we need to include rules to implement the evaluation functions Γ. The query 'f **after** α' will be true in a

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

consistent domain D if and only if $\mathit{holds\_after\_plan}(f, \alpha)$ is true in the epistemic logic program obtained from D plus the rules:

$$
\begin{aligned}
&\mathit{holds\_after\_plan}(F, P) \\
&\qquad \leftarrow \ \mathit{find\_situation}(P, s_0, S), \mathit{holds}(F, S) \\
&\mathit{find\_situation}([], S, S) \\
&\qquad \leftarrow \\
&\mathit{find\_situation}([a|\alpha], S, S_1) \\
&\qquad \leftarrow \ \mathit{find\_situation}(\alpha, \mathit{res}(a, S), S_1) \\
&\mathit{find\_situation}([\textbf{ if } \varphi \textbf{ then } \alpha_1|\alpha_2], S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\overline{\varphi}, S), \mathit{find\_situation}(\alpha_2, S, S_1) \\
&\mathit{find\_situation}([\textbf{ if } \varphi \textbf{ then } \alpha_1|\alpha_2], S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\varphi, S), \mathit{find\_situation}(\alpha_1, S, S'), \\
&\qquad\quad\ \ \mathit{find\_situation}(\alpha_2, S', S_1) \\
&\mathit{find\_situation}([\textbf{ if } \varphi \textbf{ then } \alpha_1 \textbf{ else } \ \alpha_1'|\alpha_2], \\
&\qquad\qquad S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\overline{\varphi}, S), \mathit{find\_situation}(\alpha_1', S, S'), \\
&\qquad\quad\ \ \mathit{find\_situation}(\alpha_2, S', S_1) \\
&\mathit{find\_situation}([\textbf{ if } \varphi \textbf{ then } \alpha_1 \textbf{ else } \ \alpha_1'|\alpha_2], \\
&\qquad\qquad S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\varphi, S), \mathit{find\_situation}(\alpha_1, S, S'), \\
&\qquad\quad\ \ \mathit{find\_situation}(\alpha_2, S', S_1) \\
&\mathit{find\_situation}([\textbf{ while } \varphi \ \textbf{ do } \alpha_1|\alpha_2], S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\overline{\varphi}, S), \\
&\qquad\quad\ \ \mathit{find\_situation}(\alpha_2, S, S_1) \\
\\
&\mathit{find\_situation}([\textbf{while } \varphi \textbf{ do } \alpha_1|\alpha_2], S, S_1) \\
&\qquad \leftarrow \ \mathit{Kholds}(\varphi, S), \\
&\qquad\quad\ \ \mathit{find\_situation}(\alpha_1, S, S'), \\
&\qquad\quad\ \ \mathit{find\_situation}([\textbf{while } \varphi \textbf{ do } \alpha_1|\alpha_2], S', S_1)
\end{aligned}
$$

As you may note from the rules, $\mathit{holds\_after\_plan}(F, P)$ works in two steps. First, it finds the situation s that results from applying P to the initial situation (using $\mathit{find\_situation}(P, s_0, S)$) and then shows that F holds in that situation. Since the translation of the domain may have several world views the program needs to find a situation for each world view. The following example illustrates how the process works.

Example 7.2

$$
D_1^1 \left\{
\begin{array}{l}
r_1 \ : \ \textbf{initially } \neg \mathit{burnOut} \\
r_2 \ : \ \textbf{initially } \neg \mathit{bulbFixed} \\
r_3 \ : \mathit{changeBulb} \textbf{ causes } \mathit{burnOut} \textbf{ if } \mathit{switchOn} \\
r_4 \ : \mathit{changeBulb} \textbf{ causes } \mathit{bulbFixed} \textbf{ if } \neg \mathit{switchOn} \\
r_5 \ : \mathit{turnSwitch} \textbf{ causes } \mathit{switchOn} \textbf{ if } \neg \mathit{switchOn} \\
r_6 \ : \mathit{turnSwitch} \textbf{ causes } \neg \mathit{switchOn} \textbf{ if } \mathit{switchOn} \\
r_7 \ : \mathit{checkSwitch} \textbf{ causes to know } \mathit{switchOn} \textbf{ if } \neg \mathit{burnOut}
\end{array}
\right.
$$

Assume we would like to show that

$$
\begin{aligned}
D_1^1 \models \mathit{bulbFixed} \textbf{ after } [\quad & \mathit{checkSwitch}, \\
& \ \textbf{if } \mathit{switchOn}\textbf{then } [\mathit{turnSwitch}], \\
& \mathit{changeBulb}]
\end{aligned}
$$

The states in the initial situation of this example are:

$$\begin{aligned}\Sigma = \{ \quad & \{\neg burnOut, \neg bulbFixed, switchOn\},\\ & \{\neg burnOut, \neg bulbFixed, \neg switchOn\}\}\end{aligned}$$

It has two models Φ_1 and Φ_2 that for the sensing action *checkSwitch* behave very much like in Example 7.1. Then, the logic program translation of this domain has two world views. The world view $\mathscr{W}$ corresponding to Φ_1 has two belief sets W^1 and W^2, such that $W^1_{s_0} \cup W^1_{res(cs,s_0)}$ from Example 7.1 is a subset of W^1 and $W^2_{s_0} \cup W^2_{res(cs,s_0)}$ is a subset of W^2. W^1 also contains the sets $W^1_{res(ts,res(cs,s_0))}$ =

$$\begin{aligned}&\{holds(\overline{burnOut}, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(\overline{bulbFixed}, res(turnSwitch, res(checkSwitch, s_0)),\\ &holds(\overline{switchOn}, res(turnSwitch, res(checkSwitch, s_0)))\}\end{aligned}$$

and

$$\begin{aligned}&W^1_{res(cb,res(ts,res(cs,s_0)))} =\\ &\{holds(\overline{burnOut}, res(changeBulb, res(turnSwitch, res(checkSwitch, s_0)))),\\ &holds(bulbFixed, res(changeBulb, res(turnSwitch, res(checkSwitch, s_0)))),\\ &holds(\overline{switchOn}, res(changeBulb, res(turnSwitch, res(checkSwitch, s_0))))\}\end{aligned}$$

and W^2 the set

$$\begin{aligned}W^2_{res(ts,(res(cs,s_0)))} = \quad &\{holds(\overline{burnOut}, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(\overline{bulbFixed}, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(\overline{switchOn}, res(turnSwitch, res(checkSwitch, s_0)))\}\\ &\bigcup\\ &\{holds(burnOut, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(bulbFixed, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(switchOn, res(turnSwitch, res(checkSwitch, s_0))),\\ &holds(true, res(turnSwitch, res(checkSwitch, s_0)))\}\end{aligned}$$

There is also a similar set $W^2_{res(cb,res(ts,(res(cs,s_0))))}$, with the same elements of $W^2_{res(ts,(res(cs,s_0)))}$, replacing the situation argument with

$$res(changeBulb, res(turnSwitch, (res(checkSwitch, s_0))))$$

This corresponds to the sequence

$$\begin{aligned}&\Sigma,\\ &\Phi_1(checkSwitch, \Sigma),\\ &\Phi_1(turnSwitch, \Phi_1(checkSwitch, \Sigma)),\\ &\Phi_1(changeBulb, \Phi_1(turnSwitch, \Phi_1(checkSwitch, \Sigma)))\end{aligned}$$

Thus, in this world view the predicate

$$\begin{aligned}find\_situation([\quad & checkSwitch,\\ & \ \textbf{if } switchOn \textbf{ then } [turnSwitch],\\ & changeBulb], s_0, S)\end{aligned}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

will hold in $\mathscr{W}$ iff $S = res(changeBulb, res(turnSwitch, res(checkSwitch, s_0)))$. The second step will check if

$$holds(bulbFixed, res(changeBulb, res(turnSwitch, res(checkSwitch, s_0))))$$

is in $\mathscr{W}$. The answer is yes since the atom belongs to both W^1 and W^2.

The world view associated with Φ_2 is defined in a similar manner, but in this world view $S = res(changeBulb, res(checkSwitch, s_0))$.

Let Π_D be the epistemic logic program corresponding to the translation of a domain description D, and denote by Π_D^Q the union of Π_D and the rules to interpret queries given above. Then we can show:

Theorem 7.3
Given a consistent domain description D and a plan β. $D \models F$ **after** β iff $\Pi_D^Q \models holds\_after\_plan(F, \beta)$.

Proof See the appendix.

8 Relation to other work

In Levesque (1996) there is also a programming language based on the situation calculus which uses sensing actions. This work is based on previous work from Scherl and Levesque (1993), in which knowledge is represented using two levels. There is a representation of the actual situation (called s) in which the agent is in, and there are situations accessible from s (called s') which the agent thinks it might be in. Something is known to the agent as being true (false) if it is true (false) in all situations s' which are accessible from the actual situation s and is unknown otherwise. In other words Scherl and Levesque (1993) distinguish between what is known by the agent and what is true in world. We only represent what is know by the agent, and assume that this knowledge might be incomplete, but always correct. Something is known in our representation if its value is the same throughout the states in a situation and unknown otherwise.

Levesque (1996) and Scherl and Levesque (1993) use preconditions which are executability conditions for an action's execution. For example, a precondition to clean a white-board is one must be in front of the white-board. Our preconditions differ in that they are conditions on the effects. We can always execute an action, but its effect varies according to its precondition in the effect propositions. Extending $\mathscr{A}_k$ to include executability conditions can be done as for extensions of $\mathscr{A}$. The use of conditions on effects, however, allows us to represent a phenomenon of sensing in which the value of previously unknown preconditions are learned along with the fluent we are trying to gain knowledge about. This is shown in examples 2.4 and 2.8 where the robot will know whether or not it is facing the door after executing the action *look*.

In Levesque (1996), once knowledge is gained it is never lost. We, on the other hand, explore the use of non-deterministic actions as a mechanism to remove knowledge. Our use of non-deterministic actions is similar to Thielscher (1994), where the effect of a non-deterministic action is to make a fluent true or false, but

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

exactly which is indeterminate. As might be expected, there are cases where the possible outcome is not intuitive. Take for example a deterministic action *Shoot* that causes Ollie to be dead. Any observation, which depends on Ollie being alive, such as 'Ollie is walking' can be made false using the same action Shoot. Shoot can be used as a restriction which causes Ollie not to walk. In the resulting situation, Ollie will not be alive and therefore will not be walking around. This is not the case when Shoot has the non-deterministic effect of making Ollie dead or leaving Ollie alive (suppose that the gun is not working well). With the same restriction, Ollie may be dead and not walking in one situation and alive and not walking in the other. Assuming one can walk as long as one is alive, then the later situation makes no sense. The same holds true without the restriction but this time Ollie will be dead and walking in one situation. If shoot also has a non-deterministic effect on walking, we are no better off.

These cases are prevented with integrity constraints as in Kartha and Lifschitz (1994). Our language could be extended to include constraints as in Baral *et al.* (1997) but our interest in non-determinism is its effect on knowledge. We discuss the topic of integrity constraints in Section 9.

Most translations for dialects of $\mathscr{A}$ are to extended logic programs. Our translation is to epistemic logic programs because of its ability to represent knowledge and incomplete information. To the best of our knowledge this is the first use of epistemic logic programs in a translation from action languages. The closest work related to our results is presented in Baral and Son (1997). In that paper $\mathscr{A}$ is also extended to handle sensing actions but the semantics is some what limited because they work with a three value semantics and only approximate knowledge. Furthermore, in their language sensing actions have no conditional effects. These restrictions allow Baral and Son to write translations into extended logic programs. Showing whether is possible to find a translation into extended logic programs or first order logic of our domains is an open question.

9 Future work

We already mentioned the need to clarify the complexity of adding sensing actions to domain descriptions. Our translation suggests that it might be computationally more complex to deal with conditional sensing actions than sensing with no conditions. Two other possible directions of research are: first, the ability of an agent to query itself about what it knows (i.e introspection). This is useful when the cost of executing a series of plans is expensive maybe in terms of time. Allowing an agent to query whether it knows that it knows something may be a cheaper alternative and cost effective. The use of a modal operator as shown below may be sufficient to accomplish this.

$$\textbf{if } \neg\textbf{knows}(\varphi) \textbf{ then } [\alpha] \tag{28}$$

where φ is a test condition (as defined in Section 5), α a plan, and **knows**(φ) would be an introspective operator on the test condition.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Take for example Agent A in Fig. 1 from Section 2. Agent A knows that Ollie is wet (denoted by *wet*), but does not know if it is raining outside (denoted by *rain*). Agent A would have to find a window and then look out that window to see if it is raining outside. Suppose the program or control module for finding a window in a building is long and very costly as far as battery power, Agent A would have to find a window and then check for rain. Agent B would benefit from the conditional

$$\textbf{if} \ \ \neg\textbf{knows}(\varphi) \ \textbf{then} \ [findWindow, lookOutside]$$

Without an introspective operator, both Agent A and Agent B would have to find a window and then look outside. Agent B can save on battery power if it has the ability to query itself on what it knows.

Secondly, we could investigate expanding the initial epistemic state. At present, domains only may start from a situation with only one initial epistemic state. For more states or to represent multiple initial situations in a domain, the language to describe domains must be extended with modal operators.

As mentioned earlier in this paper, integrity constraints could be added. Integrity constraints define dependency relationships between fluents. Taking the example from the previous section, walking depends on Ollie being alive. This could be represented following the approach outlined in Kartha and Lifschitz (1994) and Lifschitz (1996):

$$\textbf{never} \ \varphi \ \textbf{if} \ \psi \tag{29}$$

where φ and ψ are conjunctions of fluent literals. It states that φ can not be true when ψ is true. Our example with Ollie would look like this

$$\textbf{never} \ walking \ \textbf{if} \ \neg alive$$

Conditions of effect are used throughout this paper. An example of a executability condition is the fact that one has to be at a light bulb to change the light bulb. Executability conditions found in Lifschitz (1996) could be implemented using the methods found in Kartha and Lifschitz (1994) and Lifschitz (1996):

$$\textbf{impossible} \ A \ \textbf{if} \ \psi \tag{30}$$

where A is an action and ψ is a conjunction of fluent literals. The execution of action A cannot take place as long as ψ is true.

Using (30) we could express the constraint that in order to change the bulb, one has to be at the lamp as

$$\textbf{impossible} \ changeBulb \ \textbf{if} \ \neg atLamp$$

This paper will conclude with three thoughts. One is relaxing the assumption that an agent has incomplete but always correct knowledge of its world. One could imagine the agent not only reasoning on information that it knows is true, but also reasoning on what it believes is true. At present we have not explored this topic. The other idea is given that $\mathscr{A}_k$ is a high level action description language that deals with incomplete information across multiple possible worlds, it stands to reason that $\mathscr{A}_k$ could be translated to formalisms, such as Levesque's (Levesque, 1996; Scherl

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

and Levesque, 1993) or autoepistemic logic (Moore, 1985; Marek and Truszczynski, 1991), which also hold this property. The third refers to the termination of routines; there are certain tasks for which routines can be limited by a sensing action that determines the ‘size’ of the problem. Take for example the number of pages in a book or the number of doors on the second floor of an office building. The number of pages contained in a book will ensure the termination of a search for a word through that book. The same applies to the number of doors on the second floor with respect to a security routine which checks that all the doors on the second floor are locked. For this type of task, a counter is sufficient. To include counters, we do not require constraints but variables in $\mathscr{A}_k$. These loops correspond to for-loops in regular programming languages. Consider the situation described in Section 5.4. On the floor of a room there are cans. An agent is given an empty bag and instructed to fill the bag with cans. We assume that there are more than enough cans on the floor to fill the bag. We can model the space left in the bag by having initially true one (and only one) of the following fluents:

$$\begin{array}{l} spaceLeft(0) \\ spaceLeft(s(0)) \\ \vdots \\ spaceLeft(s^n(0)) \\ \vdots \end{array}$$

The effect of *drop* can be now described by the effect proposition:

$$r_1 : drop\ \mathbf{causes}\ spaceLeft(x)\ \mathbf{if}\ spaceLeft(s(x))$$

However, we need to restrict the world to only allow one *spaceLeft* fluent to be true at any moment. This can be described with a constraint of the form

$$\mathbf{never}\ spaceLeft(x) \wedge spaceLeft(y)\ \mathbf{if}\ x \neq y$$

Note that the constraint encodes a ramification of *drop* since not only the execution of the action *drop* makes $spaceLeft(x)$ true, but also indirectly causes $spaceLeft(s(x))$ to become false.

An orthogonal problem to the issue of constraints, is that we still need in our domain a value proposition that tells us how much space we initially have in the bag. Adding the initial value proposition is not a completely satisfactory solution since the plan

$$\mathbf{while}\ nospaceLeft(0)\ \mathbf{do}\ [dropCanInBag]$$

fills the bag irrespectively of the initial situation and (in normal circumstances) the plan will always terminate. Furthermore, in a realistic setting, plans need to consider limitation of resources. Plans may need to limit the amount of time devoted to any task or limit the amount of energy that can be used. These bounds can be applied to all tasks, but still a counter is required. Further research in termination, specially in a common-sense approach to proof of termination is necessary to deal with loops in plans.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Acknowledgements

We would like to thank the anonymous referees for their invaluable suggestions. Section 7.3 is a direct result of one of their comments.

A While-evaluation function

Definition A.1
Let $\mathscr{E}$ be the set of all situations and $\mathscr{P}$ the set of all total functions f mapping situations into situations, $\mathscr{P} = \{f \mid f : \mathscr{E} \to \mathscr{E}\}$. We say that for any pair of functions $f_1, f_2 \in \mathscr{P}, f_1 \leqslant f_2$ if and only if for any $\Sigma \in \mathscr{E}$ if $f_1(\Sigma) \neq \emptyset$, then $f_1(\Sigma) = f_2(\Sigma)$.

The next proposition follows from the above definition.

Proposition A.2
The above relation $\leqslant$ defines a partial order in $\mathscr{P}$.

Moreover, this partial order is a complete semi-lattice with the bottom element equal to the function that maps every situation to $\emptyset$. We will denote the bottom element by $f_\emptyset$.

Definition A.3
Let α be a plan and Γ a function that maps plans and situations into situations. Let φ be a conjunction of fluent literals. Then, we define the function $\mathscr{F}^{\Gamma}_{\alpha,\varphi} : \mathscr{P} \to \mathscr{P}$ such that for any function $f \in \mathscr{P}$,

$$\mathscr{F}^{\Gamma}_{\alpha,\varphi}(f)(\Sigma) = \begin{cases} \Sigma & \text{if } \varphi \text{ is false in } \Sigma \\ f(\Gamma(\alpha, \Sigma)) & \text{if } \varphi \text{ is true in } \Sigma \\ \emptyset & \text{otherwise} \end{cases}$$

Our goal is to show that $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ is continuous. For this, we will need to show that for any directed set $D \subseteq \mathscr{P}$, the least upper bound of D, denoted by $\bigsqcup D$ exists, and that $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D) = \bigsqcup\{\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d) \mid d \in D\}$. A directed set is a set such that for any finite subset of it, the least upper bound of that set exists, and belongs to the directed set. The existence of $\bigsqcup D$ follows from the following proposition.

Proposition A.4
Let D be a directed subset of $\mathscr{P}$, and let $d \in D$. If $d(\Sigma) = \Sigma' \neq \emptyset$, for a situation Σ, then for any $d' \in D$ either $d'(\Sigma) = \emptyset$ or $d'(\Sigma) = \Sigma'$.

It follows from this proposition that,

$$\bigsqcup D(\Sigma) = \begin{cases} \emptyset & \text{if } \forall d \in D, d(\Sigma) = \emptyset \\ \Sigma' & \text{if } \exists d \in D \text{ such that } d(\Sigma) = \Sigma' \text{ and } \Sigma \neq \emptyset \end{cases}$$

A similar function is defined by $\bigsqcup\{\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d) \mid d \in D\}$. This function will be used in the proof of the following theorem.

Theorem A.5
For any plan α and any conjunction of fluent literals φ, the function $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ is continuous with respect to the order $\leqslant$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Proof

Let $\bigsqcup \mathscr{F}^{\Gamma}_{\alpha,\varphi}[D]$ denote the function $\bigsqcup\{\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d) \mid d \in D\}$. To prove the theorem, it suffices to show that, for any directed set $D \subseteq \mathscr{P}$, $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D) = \bigsqcup \mathscr{F}^{\Gamma}_{\alpha,\varphi}[D]$. Let Σ be a situation.

(a) If φ is false in Σ then for any $f \in \mathscr{P}$, $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(f)(\Sigma) = \Sigma$. Hence,

$$\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D)(\Sigma) = \Sigma = \bigsqcup \mathscr{F}^{\Gamma}_{\alpha,\varphi}[D](\Sigma).$$

(b) If φ is true in Σ, then $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D)(\Sigma) = \bigsqcup D(\Gamma(\alpha,\Sigma))$. Let $\bigsqcup \mathscr{F}^{\Gamma}_{\alpha,\varphi}[D](\Sigma) = \Sigma'$. By Proposition A.4, $\Sigma' = \emptyset$ iff $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d)(\Sigma) = \emptyset$, for any $d \in D$ since $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d)(\Sigma) = d(\Gamma(\alpha,\Sigma))$ and D is directed. Therefore, $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D)(\Sigma)$ must be $\emptyset$. If $\Sigma' \neq \emptyset$, then for every $d \in D$ such that $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d)(\Sigma) \neq \emptyset$, it must be the case that $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d)(\Sigma) = \Sigma'$ since $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(d)(\Sigma) = d(\Gamma(\alpha,\Sigma))$ and D is directed. Then $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(\bigsqcup D)(\Sigma) = \Sigma'$.

(c) When φ is neither true nor false in Σ, the proof is similar to part (a) since $\mathscr{F}^{\Gamma}_{\alpha,\varphi}(f)(\Sigma) = \emptyset$ for any $f \in \mathscr{P}$. □

We define the powers of $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ as follows:

1. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow 0 = f_{\emptyset}$.
2. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow n+1 = \mathscr{F}^{\Gamma}_{\alpha,\varphi}(\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow n)$.
3. $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow \omega = \bigsqcup\{\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow n \mid n \leqslant \omega\}$.

From the continuity of $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ the corollary below follows.

Corollary A.6

The least fix-point of $\mathscr{F}^{\Gamma}_{\alpha,\varphi}$ is $\mathscr{F}^{\Gamma}_{\alpha,\varphi} \uparrow \omega$.

B Proofs

In this section we present the proof of Theorem 7.3 by givingn a detailed proof of the correctness of the translation for simple domains. The proof for the general case is a direct extension. In our proofs we will use the splitting lemma of extending logic programs (Lifschitz and Turner, 1994). For completeness we will include some definitions and the statement of the lemma below.

Consider a nonempty set of symbols called atoms. A literal is an atom possibly preceded by the classical negation symbol ¬. A rule is determined by three finite set of literals – the set of head literals , the set of positive subgoals and the set of negated subgoals. The rule with the head literals $L_1,\ldots,L_q$, the positive subgoals $L_{i+1},\ldots,L_m$ and the negated subgoals $L_{m+1},\ldots,L_n$ is written as

$$L_1 \textit{ or } \ldots \textit{ or } L_q \leftarrow L_{q+1},\ldots,L_m, \textbf{not } L_{m+1},\ldots,L_n$$

The three parts of a rule r are denoted by $head(r)$, $pos(r)$ and $neg(r)$; $lit(r)$ stands for $head(r) \cup pos(r) \cup neg(r)$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Definition B.1
(Splitting set) (Lifschitz and Turner, 1994) A splitting set for a logic program Π is any set U of literals such that, for every rule $r \in \Pi$, if $head(r) \cap U \neq \emptyset$ then $lit(r) \subseteq U$. If U is a splitting set for Π, we also say that U splits Π. The set of rules $r \in \Pi$ such that $lit(r) \subseteq U$ is called the *bottom* of Π relative to the splitting set U and is denoted by $b_U(\Pi)$. The subprogram $\Pi - b_U(\Pi)$ is called the *top* of Π relative to U.

Definition B.2
(Partial evaluation) (Lifschitz and Turner, 1994) The partial evaluation of a program Π with splitting set U w.r.t. a set of literals X is the program $e_U(\Pi, X)$ defined as follows. For each rule $r \in \Pi$ such that:

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

put in $e_u(\Pi, X)$ the rule r' which satisfies the following property:

$$head(r') = head(r), pos(r') = pos(r) - U, neg(r') = neg(r) - U.$$

Definition B.3
(Solution) (Lifschitz and Turner, 1994) Let U be a splitting set for a program Π. A solution to Π w.r.t. U is a pair (X, Y) of sets of literals such that:

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

Lemma B.4
(Splitting lemma) (Lifschitz and Turner, 1994) Let U be a splitting set for a program Π. A set A of literals is a consistent answer set of Π if and only if $A = X \cup Y$ for some solution (X, Y) to Π w.r.t. U.

From now on we will refer to the *simple domain description* 20, as *domain description* to simplify the statements.

The proof of Theorem 7.3 (*Theorem:* Given a consistent domain description D and a plan β. $D \models F$ **after** β iff $\Pi_D^O \models holds\_after\_plan(F, \beta)$.) is organized as follows:

1. First, we will prove that the epistemic logic program translation models correctly the execution of a single non-sensing action. Intuitively this can be done by looking at all the predicates of the form $holds(f, res(a, s_0))$, for any non-sensing action a. Furthermore, we should be able to replace the initial constant s_0 with any fixed situation constant s of the form $res(a_1, \ldots, res(a_k, s_0) \ldots)$. In the proof we will show that given any situation constant s, state σ, and 0-model of the domain Φ_0, we can find a sub-set of the program $ground(\Pi_D)$ in which assuming s to be the initial situation constant one of its belief sets corresponds to Φ_0. We will also prove the other direction. That is, for any belief set of the mentioned sub-set of $ground(\Pi_D)$, there exists a corresponding function Φ_0, 0-model of D. This covers the general case of a single non-sensing action applied to a situation since, by the definition of 2.7, this reduces to the application of 0-interpretations to each of the states in the situation.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

2. The second part of the proof extends the first part to cover the execution of sensing actions. In this case the sub-set of $ground(\Pi_D)$ includes rules with the modal operator K. We show that each world view of the sub-program corresponds to an interpretation Φ, model of D. We also show that for any model Φ of D there is an associated world view of the sub-program.
3. The next step extends step 2 from the application of a single action to the application of any sequence of actions by induction.
4. The final step extends the proof from sequence a of actions to complex plans. The proof shows by structural induction on the complexity of the plans that given a fixed world view any plan (that terminates) can be reduced to the execution of a sequence of actions.

Given a situation constant s, denote by $\Pi^1_{(D,s)}$ the subprogram of $Ground(\Pi_D)$ that is restricted to those rules in $ground(\Pi_D)$, such that either the only situation constant appearing in the heads is of the form $res(a,s)$ for an action symbol a, or is of the form $ab(f,a,s)$ for a fluent literal f and action symbol a.

For any possible action a, we will denote by $\Pi^1_{(D,a,s)}$ the subprogram of $\Pi^1_{(D,s)}$ that is restricted to those rules in Π_D that only involve the action a in its predicates, besides other action symbols occurring in s. We call a domain description a *universal* domain if there are no value propositions in the domain. Given a universal domain description D, and state σ, we denote by D_σ the domain consisting of $D \cup \{\textbf{initially } f : f \in \sigma\} \cup \{\textbf{initially } \neg f : f \notin \sigma\}$.

Definition B.5
For a domain description D, let σ be a state, s a situation constant, and Φ_0 a 0-interpretation, We define the set of literals $A_{(\Phi_0,\sigma,s)}$ as follows:

For any action a and any fluent f,

1. $holds(f,s) \in A_{(\Phi_0,\sigma,s)} \iff f \in \sigma$,
2. $holds(\bar{f},s) \in A_{(\Phi_0,\sigma,s)} \iff f \notin \sigma$,
3. $holds(f,res(a,s)) \in A_{(\Phi_0,\sigma,s)} \iff f \in \Phi_0(a,\sigma)$,
4. $holds(\bar{f},res(a,s)) \in A_{(\Phi_0,\sigma,s)} \iff f \notin \Phi_0(a,\sigma)$.
5. $ab(f,a,\sigma) \in A_{(\Phi_0,\sigma,s)}$ if and only if there exists an object effect proposition of the form
$$a \textbf{ causes } f \textbf{ if } p_1 \dots p_m$$
such that $p_1 \dots p_m$ holds in σ or a non deterministic effect proposition of the form
$$a \textbf{ may affect} f \textbf{ if } p_1, \dots, p_m$$
in D such that $p_1, \dots, p_m$ holds in σ and f holds $\Phi_0(a,\sigma)$.
6. $ab(\bar{f},a,\sigma) \in A_{(\Phi_0,\sigma,s)}$ if and only if there exists an object effect proposition of the form
$$a \textbf{ causes } \neg f \textbf{ if } p_1 \dots p_m$$
such that $p_1 \dots p_m$ or a non deterministic effect proposition of the form
$$a \textbf{ may affect} f \textbf{ if } p_1, \dots, p_m$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

in D such that $p_1, \ldots, p_m$ and f does not hold in $\Phi_0(a, \sigma)$.

Nothing else belongs to $A_{(\Phi_0,\sigma,s)}$.

Definition B.6

Let D be a domain description, Φ_0 a *0-interpretation*, and σ a state. We will say that the pair (Φ_0, σ) is a *0-specific model* of the domain description D if σ is an initial state of D and Φ_0 one of its 0-models.

In the next two theorems we will prove that the logic program models correctly the execution of a single non-sensing action.

Theorem B.7

Let D be a consistent universal domain description with no knowledge laws, and σ a state. If (Φ_0, σ) is a 0-specific model of D_σ, then $A_{(\Phi_0,\sigma,s)}$ satisfies every rule in $\Pi^1_{D_{(\sigma,s)}}$, for any situation constant s.

Proof

Assume that (Φ_0, σ) is a *0-specific model* of D_σ. Then any fact $holds(f, s)$ or $holds(\bar{f}, s)$ in $\Pi^1_{(D_\sigma,s)}$ is such that either $holds(f, s)$ is in $A_{(\Phi_0,\sigma,s)}$ or $holds(\bar{f}, s)$ is in $A_{(\Phi_0,\sigma,s)}$, which is obvious. Furthermore, for any literal of the form $holds(f, s)$ (resp. $holds(\bar{f}, s)$) in $\Pi^1_{(D_\sigma,s)}$ obtained from the translation of a proposition of the form **initially** f (resp. **initially** $\neg f$), we will have by construction that $holds(f, s) \in A_{(\Phi_0,\sigma,s)}$ (resp. $holds(\bar{f}, s) \in A_{(\Phi_0,\sigma,s)}$). Now, let us take a pair of rules of the form

$$holds(f, res(a, s)) \leftarrow holds(p_1, s) \ldots, holds(p_m, s)$$
$$ab(f, a, s) \leftarrow holds(p_1, s), \ldots, holds(p_m, s), \textbf{not}\ holds(true, s)$$

obtained from the translation of a proposition of the form

$$a\ \textbf{causes}\ f\ \textbf{if}\ p_1, \ldots, p_m,$$

and assume that $holds(p_1, s), \ldots, holds(p_m, s) \in A_{(\Phi_0,\sigma,s)}$. Then by construction, $p_1, \ldots, p_m$ holds in σ and $holds(true, s) \notin A_{(\Phi_0,\sigma,s)}$. Therefore $ab(f, a, s) \in A_{(\Phi_0,\sigma,s)}$ and f is in $\Phi_0(a, \sigma)$. Consequently, $holds(f, res(a, s))$ holds in $A_{(\Phi_0,\sigma,s)}$.

The rules:

$$holds(true, res(A, S)) \leftarrow holds(true, S)$$
$$holds(F, S) \leftarrow holds(true, S)$$

are trivially satisfied since there are no atoms of the form $holds(true, s)$ in $A_{(\Phi_0,\sigma,s)}$.

Now we will make several considerations on $A_{(\Phi_0,\sigma,s)}$ to evaluate the other rules (ground instances of the inertia rule and rules obtained from the translation of non-deterministic effect propositions):

1. $ab(f, a, s)$ holds in $A_{(\Phi_0,\sigma,s)}$
   Any rule of the form $holds(\bar{f}, res(a, s)) \leftarrow holds(f, s), \textbf{not}\ ab(f, a, s)$ (instance of the inertia rule) is removed from $\Pi^1_{(D_\sigma,s)}$ to verify that $A_{(\Phi_0,\sigma,s)}$ is a belief

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

set of $\Pi^1_{(D_\sigma,s)}$. Moreover, by the definition of $A_{(\Phi_0,\sigma,s)}$, there must be an effect proposition with one of the following forms

$$a\ \textbf{causes}\ f\ \textbf{if}\ p_1,\ldots,p_m$$

with $p_1,\ldots,p_m$ true in σ or

$$a\ \textbf{may affect}f\ \textbf{if}\ p_1,\ldots,p_m$$

in D_σ with $p_1,\ldots,p_m$ true in σ and $holds(f,res(a,s))$ member of $A_{(\Phi_0,\sigma,s)}$ by case (3) above. So any pair of rules of the form

$$\begin{array}{ll}
ab(f,a,s) \leftarrow & \textbf{not}\ holds(\bar{f},res(a,s)),\\
 & holds(p_1,s)),\ldots holds(p_n,s)),\textbf{not}\ holds(true,s)\\
holds(f,res(a,s)) \leftarrow & \textbf{not}\ holds(\bar{f},res(a,s)),\\
 & holds(p_1,s),\ldots holds(p_n,s)
\end{array}$$

coming from the translation of a non-deterministic effect proposition of the form

$$a\ \textbf{may affect}f\ \textbf{if}\ p_1,\ldots,p_m$$

will be trivially satisfied in $A_{(\Phi_0,\sigma,s)}$. The other two rules obtained from the non-deterministic effect propositions are of the form

$$\begin{array}{ll}
ab(\bar{f},a,s) \leftarrow & \textbf{not}\ holds(f,res(a,s)),\\
 & holds(p_1,s)),\ldots,holds(p_n,s),\textbf{not}\ holds(true,s)\\
holds(\bar{f},res(a,s)) \leftarrow & \textbf{not}\ holds(f,res(a,s)),\\
 & holds(p_1,s),\ldots holds(p_n,s)
\end{array}$$

and they will also be removed from $\Pi^1_{(D_\sigma,s)}$ to verify that $A_{(\Phi_0,\sigma,s)}$ is a belief set of $\Pi^1_{(D_\sigma,s)}$, since $holds(\bar{f},res(a,s)) \in A_{(\Phi_0,\sigma,s)}$, and this concludes the proof for this case.

2. $ab(\bar{f},a,s)$ holds in $A_{(\Phi_0,\sigma,s)}$
   Similar to previous case.
3. Neither $ab(f,a,s)$ nor $ab(\bar{f},a,s)$ are in $A_{(\Phi_0,\sigma,s)}$
   In this case we will have no effect propositions of the form:

   - a **causes** f **if** $p_1,\ldots,p_n$
   - a **causes** $\bar{f}$ **if** $p_1,\ldots,p_n$
   - a **may affect**f **if** $p_1,\ldots,p_n$

   in D_σ with $p_1,\ldots,p_n$ true in σ. Therefore any rule r in $\Pi^1_{(D_\sigma,s)}$ with predicates involving **not** , a and f, will be such that the body of r does not hold in $A_{(\Phi_0,\sigma,s)}$, unless, possibly for those rules of the form

$$\begin{array}{l}
holds(f,res(a,s)) \leftarrow holds(f,s),\textbf{not}\ ab(\bar{f},a,s)\\
holds(\bar{f},res(a,s)) \leftarrow holds(\bar{f},s),\textbf{not}\ ab(f,a,s)
\end{array}$$

   instances of the inertia rule. So we will verify these by cases,

   - $holds(f,s)$ holds in $A_{(\Phi_0,\sigma,s)}$, then $holds(\bar{f},s)$ does not belong to $A_{(\Phi_0,\sigma,s)}$ and there is nothing to verify for the second rule. The first rule is transformed into

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

$holds(f, res(a, s)) \leftarrow holds(f, s)$ and it is satisfied by $A_{(\Phi_0,\sigma,s)}$ because f is in σ and since there are no effect propositions of the above types, and f is true in $\Phi_0(a, s)$, by definition of $A_{(\Phi_0,\sigma,s)}$, we have $holds(f, res(a, s)) \in A_{(\Phi_0,\sigma,s)}$.

- $holds(\bar{f}, s)$ holds in $A_{(\Phi_0,\sigma,s)}$
  The proof is similar to the previous case.

Theorem B.8
Let D be a consistent universal domain description with no knowledge laws, and σ be a state. If (Φ_0, σ) is a 0-specific model of D_σ, then $A_{(\Phi_0,\sigma,s)}$ is a belief set of $\Pi^1_{D_{(\sigma,s)}}$, for any situation constant s.

By the above theorem we just need to prove that $A_{(\Phi_0,\sigma,s)}$ is minimal in the family of models of $\Pi^1_{D_{(\sigma,s)}}$. Let B be a proper subset of $A_{(\Phi_0,\sigma,s)}$ and Q some predicate in $A_{(\Phi_0,\sigma,s)} \setminus B$. Then Q could be a literal of one of the following five types:

(i) $Q = holds(f, s)$, in this case there will be a fact in $\Pi^1_{(D_\sigma,s)}$ not covered by B, so it would not be a belief set of $\Pi^1_{(D_\sigma,s)}$.

(ii) $Q = holds(f, res(a, s))$. $f \in \Phi_0(a, \sigma)$ since Q is in $A_{(\Phi_0,\sigma,s)}$,[7] therefore,

— If there is a rule 'a **causes** f **if** $p_1, \ldots, p_m$' in D_σ with $p_1, \ldots, p_m$ holding in σ, there is a rule $holds(f, res(a, s)) \leftarrow holds(p_1, s), \ldots, holds(p_m, s)$ in $\Pi^1_{(D_\sigma,s)}$ with $holds(p_1, s) \ldots, holds(p_m, s)$ members of $A_{(\Phi_0,\sigma,s)}$ and by (i), $holds(p_1, s) \ldots, holds(p_m, s)$ hold in B, therefore this rule will not be satisfied in B.

— If there is a rule a **may affect**f **if** $p_1, \ldots, p_m$ with $p_1, \ldots, p_m$ in σ, since $holds(\bar{f}, res(a, s))$ can not be in B (otherwise $A_{(\Phi_0,\sigma,s)}$ would be inconsistent) and $holds(f, res(a, s))$ is not in B, we will have that B does not satisfies the rule $holds(f, res(a, s)) \leftarrow$ **not** $holds(\bar{f}, res(a, s)), holds(p_1, s), \ldots, holds(p_m, s)$.

— If there are no effect propositions in D_σ involving a and f, then, we have that f is in σ, because in this case $f \in \Phi_0(a, \sigma)$ if and only if $f \in \sigma$, and the rule that will not be satisfied by B is the (ground instance of the) inertia rule $holds(f, res(a, s)) \leftarrow holds(f, s),$ **not** $ab(\bar{f}, a, s)$.

(iii) $Q = holds(\bar{f}, res(a, s))$
The proof of this case is similar to the previous case.

(iv) $Q = ab(f, a, s)$
In this case we have that there is either an effect proposition of the form

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_m$$

with $p_1, \ldots, p_m$ true in σ or

$$a \textbf{ may affect} f \textbf{ if } p_1, \ldots, p_m$$

in D_σ with $p_1, \ldots, p_m$ true in σ, and $holds(f, res(a, s)) \in A_{(\Phi_0,\sigma,s)}$.

[7] Note that by consistence of D_σ (we are assuming that (Φ_0, σ) is a 0-specific model) there is no rule of the form 'a **causes** $\neg f$ **if** $p_1, \ldots, p_m$' in D_σ with $p_1, \ldots p_m$ holding in σ.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Hence, one of the following two rules are not satisfied in B

$$\begin{array}{rl} ab(f,a,s) \leftarrow & holds(p_1,s)\ldots holds(p_m,s), \mathbf{not}\ holds(true,s) \\ ab(f,a,s) \leftarrow & \mathbf{not}\ holds(\bar{f},res(a,s)), \\ & holds(p_1,s),\ldots,holds(p_m,s), \mathbf{not}\ holds(true,s). \end{array}$$

(v) $Q = ab(\bar{f},a,s)$.
Similar to previous case.

We prove completeness in two steps. First, we show that if a belief set of $\Pi^1_{(D_\sigma,s)}$ is defined as in Defintion B.5 then (Φ_0,σ) is a 0-specific model of D_σ. Then we show that every belief set of $\Pi^1_{(D_\sigma,s)}$ must be of this form.

Theorem B.9
Let D be a consistent universal domain description with no knowledge laws, Φ_0 a 0-interpretation, and σ a state. If $A_{(\Phi_0,\sigma,s)}$ is a *belief set* of $\Pi^1_{(D_\sigma,s)}$, then (Φ_0,σ) is a *0-specific model* of D_σ.

Proof

Let $A_{(\Phi_0,\sigma,s)}$ be a belief set of $\Pi^1_{(D_\sigma,s)}$. Clearly, by construction, σ is an initial state of D_σ. Now let f be a fluent such that there is an effect proposition of the form 'a **causes** f **if** $p_1,\ldots,p_m$' in D_σ, and assume that $p_1,\ldots,p_m$ hold in σ. Then, by construction, there is a rule of the form

$$holds(f,res(a,s)) \leftarrow holds(p_1,s),\ldots,holds(p_m,s)$$

in $\Pi^1_{(D_\sigma,s)}$ such that $holds(p_1,s),\ldots,holds(p_m,s) \in A_{(\Phi_0,\sigma,s)}$.

Therefore $holds(f,res(a,s)) \in A_{(\Phi_0,\sigma,s)}$ and hence $f \in \Phi_0(a,\sigma)$. The proof is analogous for effect propositions of the form a **causes** $\neg f$ **if** $p_1,\ldots,p_m$. If f is a fluent such that there are no effect propositions of the above two types we have two possible situations:

(i) If there are no non-deterministic effect propositions of the form

$$a\ \mathbf{may\ affect} f\ \mathbf{if}\ p_1,\ldots,p_m$$

with $p_1,\ldots,p_m$ holding in σ, we will have that there are no rules in $\Pi^1_{(D_\sigma,s)}$ such that $ab(f,a,s)$ appears in the head of the rule and whose body holds in $A_{(\Phi_0,\sigma,s)}$. Therefore by the rules

$holds(f,res(a,s)) \leftarrow holds(f,s), \mathbf{not}\ ab(f,a,s)$ and
$holds(\bar{f},res(a,s)) \leftarrow holds(\bar{f},s), \mathbf{not}\ ab(f,a,s)$,

(ground instances of the inertia rule), we will have that either $holds(f,res(a,s))$ is in $A_{(\Phi_0,\sigma,s)}$ or $holds(\bar{f},res(a,s))$ is in $A_{(\Phi_0,\sigma,s)}$, since either $f \in \sigma$ or $f \notin \sigma$, forcing either $holds(f,s)$ or $holds(\bar{f},s)$ to be in $A_{(\Phi_0,\sigma,s)}$. Thus, $holds(f,res(a,s))$ (resp. $holds(\bar{f},res(a,s))$) is in $A_{(\Phi_0,\sigma,s)}$ if and only if $holds(f,s)$ (resp. $holds(\bar{f},s)$) is in $A_{(\Phi_0,\sigma,s)}$. Therefore $f \in \Phi_0(a,\sigma)$ if and only if f is in σ.

(ii) On the other hand, if there is a proposition of the form

$$a\ \mathbf{may\ affect} f\ \mathbf{if}\ p_1,\ldots,p_m$$

in D_σ with $p_1, \ldots, p_m$ holding in σ, by construction, we will have in $\Pi^1_{(D_\sigma,s)}$ the following rules

$$
\begin{array}{lcl}
 & holds(f, res(a,s)) & \leftarrow & \textbf{not } holds(\bar{f}, res(a,s)), \\
 & & & holds(p_1, s), \ldots, holds(p_m, s) \\
 & holds(\bar{f}, res(a,s)) & \leftarrow & \textbf{not } holds(f, res(a,s)), \\
 & & & holds(p_1, s), \ldots, holds(p_m, s) \\
 : & ab(f, res(a,s)) & \leftarrow & \textbf{not } holds(\bar{f}, res(a,s)), \textbf{not } holds(true, s), \\
 & & & holds(p_1, s), \ldots, holds(p_m, s) \\
 & ab(\bar{f}, a_n, res(a,s)) & \leftarrow & \textbf{not } holds(f, res(a,s)), \textbf{not } holds(true, s), \\
 & & & holds(p_1, s), \ldots, holds(p_m, s)
\end{array}
$$

Thus, since $A_{(\Phi_0,\sigma,s)}$ is a belief set of $\Pi^1_{(D_\sigma,s)}$ and the $holds(p_i, s)$ are assumed to belong to $A_{(\Phi_0,\sigma,s)}$ for every i, then either $holds(f, res(a,s))$ or $holds(\bar{f}, res(a,s))$ must be in $A_{(\Phi_0,\sigma,s)}$, but not both. Therefore it does not matter if f is or is not part of $\Phi_0(a, \sigma)$. Hence (Φ_0, σ) is a 0-specific model of D_σ.

Observe that for any domain description D, any state σ and any initial situation constant s, a set of predicates A will be a belief set of $\Pi^1_{(D,s)}$ if and only if A is the union of belief sets of $\Pi_{(D,s,a)}$, for each possible action a. $A = \bigcup\{A_a : a \text{ is a possible action}\}$ with each A_a a belief set of $\Pi_{(D,s,a)}$. This is because if a_1 and a_2 are two different actions then none of the predicates in rules in $\Pi_{(D,s,a_1)}$ appear in any predicate of any rule in $\Pi_{(D,s,a_2)}$, so the computation of the belief sets for one of the programs does not affect the computation for the other one.

Theorem B.10

Given a consistent domain description D, and a situation constant s. If A is a belief set for $\Pi^1_{(D,s)}$, then there exists a state σ, and a 0-specific model Φ_0 of D_σ such that $A = A_{(\Phi_0,\sigma,s)}$.

Proof

By definition of $\Pi^1_{D,s}$, A must be complete. That is, for any fluent f we have that either $holds(f, s)$ is in A or $holds(\bar{f}, s)$ is in A. Thus, if we let $\sigma = \{f : holds(f, s) \in A\}$ and Φ_0 be such that for any possible action a, $f \in \Phi_0(a, \sigma)$ if and only if $holds(f, res(a,s)) \in A$, we will have by completeness that $f \notin \sigma$ if and only if $holds(\bar{f}, s)$ is in A and $f \notin \Phi_0(a, s)$ if and only if $holds(\bar{f}, res(a,s)) \in A$. Moreover, if some predicate $ab(f, a, s)$ is in A then one of the following facts holds:

- There is a rule in $\Pi^1_{(D,s)}$ whose body is

  $$holds(p_1, s), \ldots, holds(p_m, s), \textbf{not } holds(true, s)$$

  and whose head is $ab(f, a, s)$ such that $holds(p_i, s) \in A$ for any $i = 1, \ldots, m$, and $holds(true, s) \notin A$. Thus, there must be an effect proposition of the form 'a **causes** f **if** $p_1, \ldots, p_m$' in D with $p_1, \ldots, p_m$ true in σ.
- There is a rule in $\Pi^1_{(D,s)}$ whose body is

  not $holds(\bar{f}, res(a,s))$, $holds(p_1, s), \ldots, holds(p_m, s)$, **not** $holds(true, s)$ whose head

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

is $ab(f, a, s)$ such that $holds(f, res(a, s))$ and each $holds(p_i, s)$ are in A, for each $i = 1, \ldots, m$, and $holds(true, s)$ is not in A. In this case there exists a non-deterministic effect proposition of the form 'a **may affect**f **if** $p_1, \ldots, p_m$' in D with $f, p_1, \ldots p_m$ true in σ.

If for some fluent f, $ab(f, a, s)$ is in A, we will have by similar reasons that there exists a proposition of the form

$$a \textbf{ causes } \neg f \textbf{ if } p_1, \ldots, p_m \quad \text{or } a \textbf{ may affect} f \textbf{ if } p_1, \ldots, p_m$$

in D such that $p_1, \ldots, p_m$ hold in σ and $f \notin \Phi_0(a, \sigma)$.

So we have proved that $A = A_{(\Phi_0, \sigma, s)}$, and by Theorem B.10, (Φ_0, σ) is a 0-specific model.

We now extend the proof to handle sensing actions. Let $Mod_0(D, \Sigma)$ denote the set $\{(\Phi_0, \sigma) : \text{ 0-specific model of } D, \ \sigma \in \Sigma\}$, where Σ is the set of initial states of D. Note that the set can be empty if there is no state in Σ that is an initial state of D.

Definition B.11

Given a consistent situation Σ, and an interpretation Φ. (Φ, Σ) will be a 1-*specific model* of a consistent domain description D if:

1. Σ is an initial situation of D.
2. For any non-sensing action a, $\Phi(a, \Sigma) = \bigcup_{\Phi_0 \in Mod_0(D,\Sigma)} \bigcup_{\sigma \in \Sigma} \{\Phi_0(a, \sigma)\}$.
3. For each sensing action a, if

$$\begin{array}{l} a \ \textbf{ causes to know } f_1 \ \textbf{ if } \ \varphi_1 \\ \vdots \\ a \ \textbf{ causes to know } f_s \ \textbf{ if } \ \varphi_{s_a} \end{array}$$

are all the knowledge laws in D where a occurs. Then, $\Phi(a, \Sigma)$ must be consistent and if $s_a = 0$, $\Phi(a, \Sigma) = \Sigma$; otherwise $\Phi(a, \Sigma) = \bigcap_{l=1,\ldots,s} \Sigma_l$ such that each Σ_l is a situation (f_l, φ_l)-compatible with Σ. (Recall that since D is simple, all the f_i are different.)

Definition B.12

Let (Φ, Σ) be a 1-specific model of a domain description D. Denote by $Asso_D(\Phi, \Sigma)$ the set of 0-specific models of D such that for any non-sensing action a, $\Phi(a, \Sigma) = \bigcup_{\sigma \in \Sigma} \{\Phi_0(a, \sigma)\}$.

Let a be a sensing action. Define:

$A^a_{(\sigma,s)} = \{holds(f, res(a, s)) : \sigma \models f$ with f a fluent literal $\}$ if $\sigma \in \Phi(a, \Sigma)$. Otherwise, $A^a_{(\sigma,s)} = \{holds(true, res(a, s))\} \cup \{holds(f, res(a, s)) : f$ fluent literal $\}$.
Let $A'_{(\Phi_0,\sigma,s)} = A_{(\Phi_0,\sigma,s)} \cup \bigcup_{a \in Sensing} A^a_{(\sigma,s)}$.
Let $\mathbf{A}_{(\Phi,\Sigma,s)} = \{A'_{(\Phi_0,\sigma,s)} : (\Phi_0, \sigma) \in Asso_D(\Phi, \Sigma)\}$.

As a straightforward consequence of this definition we have that for any fluent f and any action a, $\Phi(a, \Sigma) \models f$ iff $\mathbf{A}_{(\Phi,\Sigma,s)} \models holds(f, res(a, s))$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

For any set A of literals we will denote by σ_A the state $\sigma_A = \{f : holds(f, s) \in A\}$.

We will denote by $D^{n.s}$ the set of value and effect propositions in D, and by D^{sen} the set of knowledge laws in D.

As a corollary of the theorems [B.8, B.10] we will have the soundness and completeness of the logic program translation for the execution of a single action (sensing or not). The next corollary shows soundness and Corollary B.14 shows completeness.

Corollary B.13

Let D be a consistent domain description. If (Φ, Σ) is a 1-specific model of D then $\mathbf{A}_{(\Phi,\Sigma,s)}$ is a world view of $\Pi^1_{(D,s)}$.

Proof

Let us suppose that (Φ, Σ) is a 1-specific model of D. We will prove that $\mathbf{A}_{(\Phi,\Sigma,s)}$ is a world view of $\Pi^1_{(D,s)}$. In other words, we will show that $\mathbf{A}_{(\Phi,\Sigma,s)}$ is the collection of belief sets of $[\Pi^1_{(D,s)}]_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ (see Section 6 for the definition of $[\Pi]_{\mathbf{A}}$ and $[\Pi]^A_{\mathbf{A}}$). Given an $A \in \mathbf{A}_{(\Phi,\Sigma,s)}$, let $A = A'_{(\Phi_0,\sigma,s)}$, and denote by Π the program $[\Pi^1_{(D,s)} \setminus \Pi^1_{(D^{n.s},s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ which is equal to $[\Pi^1_{(D^{sen},s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ union all the rules of the form $holds(true, res(a, s)) \leftarrow holds(true, s)$ and $holds(f, res(a, s)) \leftarrow holds(true, res(a, s))$ where f is a fluent literal and a is a sensing action.

The set $U = lit(\Pi^1_{(D^{n.s},s)})$ split $[\Pi^1_{(D,s)}]_{\mathbf{A}_{(\Phi,\Sigma,s)}}$, and by theorem B.8 $A_{(\Phi_0,\sigma,s)}$ is a belief set of $\Pi^1_{(D^{n.s},s)}$, moreover $b_U(\Pi^1_{(D,s)}) = \Pi^1_{(D^{n.s},s)}$ and any answer set of $e_U([\Pi^1_{(D,s)}]_{A_{(\Phi_0,\sigma,s)}} \setminus \Pi^1_{(D^{n.s},s)}, A_{\Phi_0,\sigma,s)})$ is a belief set of $A_{\Phi_0,\sigma,s)})$.

Hence by *Splitting Lemma* we only need to prove that A is a belief set of $A_{(\Phi_0,\sigma,s)} \cup \Pi$. We first prove that all the rules in the program hold in A and then we show that A is minimal.

Obviously, any fact in $A_{(\Phi_0,\sigma,s)} \cup \Pi$ is in A, thus we will prove that any rule R in Π holds in A. Let, for a given sensing action a,

$$
\begin{array}{l}
a \ \textbf{causes to know} \ f_1 \ \textbf{if} \ p^1_1, \ldots, p^1_{n_1} \\
\vdots \\
a \ \textbf{causes to know} \ f_{s_a} \ \textbf{if} \ p^{s_a}_1, \ldots, p^{s_a}_{n_{s_a}}
\end{array}
$$

be all the knowledge laws in D, involving a, and $\Phi(a, \Sigma) = \bigcap^{s_a}_{l=1} \Sigma_l$ where each Σ_l is $(f_l, p^l_1, \ldots, p^l_{n_l})$-compatible with Σ. The rules R in Π that mention a either in its body or in its head will be evaluated as follows:

1. If $\Sigma \models f_l$ or $\Sigma \models \neg f_l$, there are no rules in the program with the predicate $holds(true, res(a, s))$ in the head that are not ground instances of domain independent rules (because any rule in $\Pi^1_{(D^{sen},s)}$ will be removed, to get $[\Pi^1_{(D^{sen},s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ after checking

   $\neg K holds(f_1, res(a, s)), \neg K holds(\bar{f}_1, res(a, s)))$.
2. If $\Sigma \not\models f_l$ and $\Sigma \not\models \neg f_l$, then either (a) $\Phi(a, \Sigma) \models p^l_1, \ldots, p^l_{n_l}$, and $\Phi(a, \Sigma) \models f_l$.

   In this case, R must be one of the following:

$$
\begin{aligned}
&holds(true, res(a,s)) \leftarrow holds(\bar{p}^l_1, s)\\
&\vdots\\
&holds(true, res(a,s)) \leftarrow holds(\bar{p}^l_{n_l}, s)\\
&holds(true, res(a,s)) \leftarrow holds(\bar{f}_l, s)
\end{aligned}
$$

and each of these rules are verified in A, because; if $holds(\bar{f}_l, s)$ is in A or for some i $holds(\bar{p}^l_i, s) \in A$ then $\sigma_A \models \bar{p}^l_i$ or $\sigma_A \models \bar{f}_l$ and in both cases $\sigma_A \notin \Phi(a, \Sigma)$, and hence, $holds(true, res(a,s)) \in A$,by definition of $A^a_{(\sigma,s)}$.

(b) $\Phi(a,\Sigma) \models p^l_1, \ldots, p^l_{n_l}$, and $\Phi(a,\Sigma) \models \bar{f}_l$ This case is similar to (a) changing the last rule for

$$holds(true, res(a,s)) \leftarrow holds(f_l, s).$$

3. $\Sigma \not\models f_l$ and $\Sigma \not\models \neg f_l$, and there exists $i = 1, \ldots, n_l$, such that $\Phi(a,\Sigma) \not\models p^l_i$. In this case, there is only one rule that remains in the program;

   $$holds(true, res(a,s)) \leftarrow holds(p^l_1, s), \ldots, holds(p^l_{n_l}, s)$$

   Now, if for any $i = 1, \ldots, n_l$, $holds(p^l_i, s) \in A$, then $\sigma_A \models p^l_i$, for every $i = 1, \ldots, n_l$, and $\sigma_A \notin \Phi(a,\Sigma)$. Thus, $holds(true, res(a,s)) \in A$.
4. From the domain independent rules, R could also be of the form

   $$holds(true, res(a,s)) \leftarrow holds(true, s)$$

   But, by construction of A, $holds(true, s) \notin A$, and thus, R is satisfied by A.
5. The last rules to consider, also coming from the domain independent rules, are all the rules of the form $holds(f, res(a,s)) \leftarrow holds(true, res(a,s))$, where f is a fluent literal. If $holds(true, res(a,s))$ belongs to A then, by construction, $\sigma_A \notin \Phi(a,\Sigma)$. Therefore, by construction too, $holds(f, res(a,s))$ also belongs to A, for every f, fluent literal.

To prove the minimality of A, let C be a proper subset of A and h a predicate in $A \setminus C$.[8] We will find a rule R in $[\Pi^1_{(D,s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ such that R does not hold in C.

If h is of the form $ab(f,a,s)$, $holds(f,s)$ or $holds(f, res(a,s))$, with a a non-sensing action, then R can be found in $[\Pi^1_{(D^{n.s},s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ by Theorem [B.8]. Thus, all these h must be in C. Therefore, it suffices to consider the case when $h = holds(f, res(a,s))$, with a a sensing action. If $h = holds(f, res(a,s))$, with f a fluent literal, and the rule $holds(f, res(a,s)) \leftarrow holds(true, res(a,s))$ is satisfied in C, then $holds(true, res(a,s)) \notin C$, in which case, if $holds(true, res(a,s))$ is in $A'_{(\Phi_0,\sigma,s)} = A$ then $\sigma \notin \Phi(a,\Sigma)$, hence we will have that there exists an $l = 1, \ldots, s_a$, such that $\sigma \notin \Sigma_l$,then since $\Phi(a,\Sigma)$ is $f_l, p^l_1, \ldots, p_{n_l}$-compatible with Σ and the remark at the end of [B.12] we will have

[8] Recall that $A = A'_{(\Phi_0,\Sigma,s)}$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

that one rule R of the following:

$$\begin{array}{ll}
holds(true, res(a,s)) & \leftarrow holds(\bar{f}_l, s) \\
holds(true, res(a,s)) & \leftarrow holds(f_l, s) \\
holds(true, res(a,s)) & \leftarrow holds(\bar{p}^l_1, s) \\
 & \vdots \\
holds(true, res(a,s)) & \leftarrow holds(\bar{p}^l_{n_l}, s) \\
holds(true, res(a,s)) & \leftarrow holds(p^l_1, s), \ldots, holds(p^l_{n_l}, s)
\end{array}$$

has to be such that both (i) $R \in [\Pi^1_{(D^{sens},s)}]^A_{\mathbf{A}(\Phi,\Sigma,s)}$ and (ii) the fluent literals appearing on the body of R will be in σ_A. Hence the body of R will be true in A and therefore in C, thus we can conclude that R is not satisfied by C. If $holds(true, res(a,s))$ is not in A, then $\sigma \in \Phi(a,\Sigma)$ and $holds(f, res(a,s))$ is in A, but this happens if and only if $f \in \sigma$, which is true iff $holds(f,s) \in A$, and the inertia rule will not be true in C. To complete the proof we need to show that any belief set of $[\Pi^1_{(D,s)}]_{\mathbf{A}(\Phi,\Sigma,s)}$ is of the form $A'_{(\Phi_0,\sigma,s)}$, for some $(\Phi_0,\sigma) \in Asso_D(\Phi,\Sigma)$. Take now A, a belief set of $[\Pi^1_{(D,s)}]_{\mathbf{A}(\Phi,\Sigma,s)}$. By the splitting lemma, the set $A_0 = A \setminus \{holds(f, res(a,s)) : a$ sensing action $\}$, is a belief set of $\Pi^1_{(D^{n.s},s)}$. Then by (B.10) there exists (Φ_0,σ) in $Asso_D(\Phi,\Sigma)$ such that $A_0 = A_{(\Phi_0,\sigma,s)} = A_{(\Phi_0,\sigma_A,s)}$ taking $\Phi(a,\Sigma) = \bigcup_{\sigma\in\Sigma}\{\Phi_0(a,\sigma)\}$, it only remains to be shown that for any sensing action a, both of the following are satisfied: (i) If $\sigma_A \in \Phi(a,\Sigma)$ then $f \in \sigma_A \Leftrightarrow holds(f, res(a,s)) \in A$, and (ii) $\sigma_A \notin \Phi(a,\Sigma) \Leftrightarrow holds(true, res(a,s)) \in A$.

For case (i), let $\sigma_A \in \Phi(a,\Sigma)$. Then, for any $l = 1, \ldots, s_a$, $\sigma_A \in \Sigma_l$. The rules with heads of the form $holds(f, res(a,s))$ and f a fluent literal are: $holds(f, res(a,s)) \leftarrow holds(true, res(a,s))$ and the one of the ground instances of the inertia rule. The body of the first rule is false in A because any rule with $holds(true, res(a,s))$ in its head must have its body false in A. Then, $holds(f, res(a,s)) \in A \Leftrightarrow holds(f,s) \in A \Leftrightarrow f \in \sigma_A$.

For (ii), $holds(true, res(a,s)) \in A$ if and only if there exists a rule R which body is true in A and its head $holds(true, res(a,s))$. Hence, R must be one of the following rules:

$$\begin{array}{ll}
holds(true, res(a,s)) & \leftarrow holds(\bar{f}_l, s) \\
holds(true, res(a,s)) & \leftarrow holds(f_l, s) \\
holds(true, res(a,s)) & \leftarrow holds(\bar{p}^l_1, s) \\
 & \vdots \\
holds(true, res(a,s)) & \leftarrow holds(\bar{p}^l_{n_l}, s) \\
holds(true, res(a,s)) & \leftarrow holds(p^l_1, s), \ldots, holds(p^l_{n_l}, s)
\end{array}$$

for some $l = 1, \ldots, s_a$, and in any case, the rule R belongs to $[\Pi^1_{(D,s)}]_{\mathbf{A}(\Phi,\Sigma,s)}$ and its body is true in A, if and only if $\sigma_A \notin \Sigma_l$. Hence, $\sigma_A \notin \Phi(a,\Sigma)$.

Corollary B.14

(Completeness) Let D be a consistent domain description. If $\mathbf{A}$ is a world view of $\Pi^1_{(D,s)}$ then there exists a 1-specific model (Φ,Σ) of D such that $\mathbf{A} = \mathbf{A}_{(\Phi,\Sigma,s)}$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Proof

Let $\mathbf{A}$ be a world view of $\Pi^1_{(D,s)}$. Let $\Sigma = \{\sigma_A : A \in \mathbf{A}\}$. If $A \in \mathbf{A}$, Φ_0^A will be a 0-interpretation such that $A_{(\Phi_0^A,\sigma_A,s)} = A \setminus \{holds(f,res(a,s)) : a$ is a sensing action and f is *true* or a fluent literal$\}$, which can be found making use of Theorem [B.10]. We define an interpretation Φ such that $Asso_D(\Phi,\Sigma) = \{(\Phi_0^A,\sigma_A) : A \in \mathbf{A}\}$, and $\Phi(a,\Sigma) = \{\sigma_A : holds(true,res(a,s)) \notin A\}$ for any sensing action a. Note that if (Φ,Σ) is a 1-specific model of D then $A = A'_{(\Phi_0^A,\sigma_A,s)}$ for any $A \in \mathbf{A}$. Thus, we will show that (Φ,Σ) is a 1-specific model of D and we will have that $\mathbf{A} = \mathbf{A}_{(\Phi,\Sigma,s)}$. It is clear that Σ is the initial situation of D. Then, if a is a non-sensing action, by definition, $\Phi(a,\Sigma) = \bigcup\{\{\Phi_0^A(a,\sigma_A)\} : A \in \mathbf{A}\} = \bigcup\{\{\Phi_0(a,\sigma)\} : (\Phi_0,\sigma) \in Asso_D(\Phi,\Sigma)\}$.

If a is a sensing action and, a **causes to know** f_l **if** $p_1^l,\dots,p_{n_l}^l, l = 1,\dots,s_a$ are exactly the knowledge laws where a appears, we need to show that for each $l = 1,\dots,s_a$, there exists a Σ_l, $(f_l,p_1^l,\dots,p_{n_l}^l)$-compatible with Σ such that $\Phi(a,\Sigma) = \bigcap_{l=1,\dots,s_a} \Sigma_l$:

1. If $\mathbf{A} \models holds(f_l,s)$ or $\mathbf{A} \models holds(\bar{f}_l,s)$ then let $\Sigma_l = \Sigma$.
2. If $\mathbf{A} \not\models holds(f_l,s)$ and $\mathbf{A} \not\models holds(\bar{f}_l,s)$ and $\mathbf{A} \models holds(p_i^l,res(a,s))$, for $i = 1,\dots,n_l$, then:

   (a) if $\mathbf{A} \models holds(f_l,res(a,s))$, then let $\Sigma_l = \{\sigma \in \Sigma : \sigma \models p_1^l,\dots,p_{n_l}^l,f_l\}$.
   (b) if $\mathbf{A} \models holds(\bar{f}_l,res(a,s))$, then let $\Sigma_l = \{\sigma \in \Sigma : \sigma \models p_1^l,\dots,p_{n_l}^l,\bar{f}_l\}$.
3. If $\mathbf{A} \not\models holds(f_l,s)$ and $\mathbf{A} \not\models holds(\bar{f}_l,s)$ and $\mathbf{A} \not\models holds(p_i^l,res(a,s))$, for some $i = 1,\dots,n_l$, we let $\Sigma_l = \{\sigma \in \Sigma : \exists k = 1,\dots,n_l, \sigma \not\models p_k^l\}$.

We need to show next that $\Phi(a,\Sigma) = \bigcap_{l=1}^{s_a} \Sigma_l$. For that, we will prove that for any $\sigma_A \in \Sigma$, $\sigma_A \notin \bigcap_{l=1,\dots,s_a} \Sigma_l$ if and only if $holds(true,res(a,s)) \in A$.

First, we will show that if $\sigma_A \notin \bigcap_{l=1,\dots,s_l} \Sigma_l$, then $holds(true,res(a,s)) \in A$. Let k be an l such that $\sigma_A \notin \Sigma_l$. Thus, since $\sigma_A \in \Sigma$ then $\Sigma \not\models f_k$ and $\Sigma \not\models \neg f_k$ and case (1) above does not occur, and one of the following cases must hold:

2.a. $\mathbf{A} \models holds(p_i^k,res(a,s))$, for every $i = 1,\dots,n_k$, and $\mathbf{A} \models holds(f_k,res(a,s))$. Therefore, either $\sigma_A \not\models p_j^k$ for some $j = 1,\dots,n_k$ or $\sigma_A \not\models f_k$. Hence, $holds(\bar{p}_j^k,s) \in A$ or $holds(\bar{f}_k,s) \in A$ and the following rules will be part of $[\Pi^1_{(D,s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$:

$$holds(true,res(a,s)) \leftarrow holds(\bar{p}_j^k,s)$$
$$holds(true,res(a,s)) \leftarrow holds(\bar{f}_k,s)$$

Thus, $holds(true,res(a,s)) \in A$.

2.b. $\mathbf{A} \models holds(p_i^k,res(a,s))$, for $i = 1,\dots,n_k$, and $\mathbf{A} \models holds(\bar{f}_k,res(a,s))$, is similar to (2.a).

3. $\mathbf{A} \not\models holds(p_i^k,res(a,s))$, for some $i = 1,\dots,n_k$. Therefore, for every $j = 1,\dots,n_k$, $\sigma_A \models p_j^k$. Hence $A \models holds(p_j^k,s)$ and the following rule will be part of $[\Pi^1_{(D,s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$:

$$holds(true,res(a,s)) \leftarrow holds(p_1^k,s),\dots,holds(p_{n_k}^k,s)$$

Thus, $holds(true,res(a,s)) \in A$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

For the other direction, assume $holds(true, res(a, s)) \in A$. Then, it must be the case that there exists a rule in $[\Pi^1_{(D,s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ with $holds(true, res(a, s))$ in the head and its body true in A. Note that this rule cannot be $holds(true, res(a, s)) \leftarrow holds(true, s)$ by the construction of A. Thus, $\mathbf{A} \not\models holds(f, s)$ and $\mathbf{A} \not\models holds(\bar{f}, s)$; otherwise, there will be no rule in $[\Pi^1_{(D,s)}]^A_{\mathbf{A}_{(\Phi,\Sigma,s)}}$ with $holds(true, res(a, S))$ in its head (these are ground instances of rules derived from knowledge laws). We will inspect the remaining rules with $holds(true, res(a, s))$ in the head and we will show that there exists Σ_k such that $\sigma_A \notin \Sigma_k$.

1. If the rules are of the form:

   $holds(true, res(a, s)) \leftarrow holds(\bar{p}^k_j, s)$
   $holds(true, res(a, s)) \leftarrow holds(\bar{f}_k, s)$

   then $\mathbf{A} \models holds(p^k_i, res(a, s))$, for $i = 1, \ldots, n_k$, and $\mathbf{A} \models holds(f_k, res(a, s))$. Therefore, since $holds(\bar{p}^k_j, s)$ or $holds(\bar{f}_k, s)$ has to belong to A, $\sigma_A \notin \Sigma_k$.
2. If the rules are of the form:

   $holds(true, res(a, s)) \leftarrow holds(\bar{p}^k_j, s)$
   $holds(true, res(a, s)) \leftarrow holds(f_k, s)$

   then, similar to 1, $\mathbf{A} \models holds(p^k_i, res(a, s))$, for $i = 1, \ldots, n_k$, and $\mathbf{A} \models holds(\bar{f}_k, res(a, s))$. Therefore, since $holds(\bar{p}^k_j, s)$ or $holds(f_k, s)$ has to belong to A, $\sigma_A \notin \Sigma_k$.
3. If the rule is of the form:

   $holds(true, res(a, s)) \leftarrow holds(p^k_1, s), \ldots, holds(p^k_{n_k}, s)$

   then $\mathbf{A} \not\models holds(p^k_i, res(a, s))$, for some $i = 1, \ldots, n_k$. Therefore, since for every $j = 1, \ldots, n_k$ $holds(p^k_j, s)$ has to be in A, $\sigma_A \notin \Sigma_k$.

The next step is to show soundness and completeness for sequences of actions. Sequences of actions are the most simple plans. We then extend the proof to plans of all classes. The general proof will be by induction on the complexity of the plans. Thus, we start by formally defining complexity and other definitions required for the inductions.

Definition B.15
We will define the *complexity* of a plan β ($comp(\beta)$) by: if the empty plan is [], $comp([]) = 0$. For an action a $comp(a) = 1$. For complex plans, $comp(\textbf{ if } \varphi \textbf{ then } \alpha)$ and $comp(\textbf{ while } \varphi \textbf{ do } \alpha)$ is $comp(\alpha) + 1$ and $comp(\textbf{ if } \varphi \textbf{ then } \alpha_1 \textbf{ else } \alpha_2)$ and $comp([\alpha_1, \alpha_2]$ is $comp(\alpha_1) + comp(\alpha_2)$.

We will say that a plan α is an *n-plan* if it has complexity n, it will be an $\leqslant$ *n-plan* if it has complexity less or equal than n. P_n will denote the set of n-plans, and $P_{\leqslant n}$ the set of $\leqslant n$-plans.

We define the complexity of a situation constant s inductively as 0 if $s = s_0$; or 1 plus the complexity of s' if $s = res(a, s')$, for any action a. A situation s will be called an *n-situation* if its complexity is n. The complexity of a predicate of the form $holds(f, s)$ with f a fluent literal, or $holds(true, s)$, will be the complexity of s, the complexity of predicates of the form $ab(f, a, s)$, with f a fluent literal and a an

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

action will be equal to the complexity of s plus one, the complexity of predicates of the form $find\_situation(\beta, s_1, s)$ will be the complexity of β plus the complexity of s_1, and the complexity of a predicate of the form $holds\_after\_plan(F, \beta)$ will be the complexity of the plan β. We will say that a predicate h is an $\leqslant n$*-predicate*, if h has complexity m and $m \leqslant n$. Given a plan α, $[\alpha^1]$ will denote the plan α and $[\alpha^{n+1}]$ will denote the plan $[\alpha|[\alpha^n]]$. Denote by Π_D^n the subprogram of Π_D restricted to those rules in Π_D with $\leqslant n$-predicates. Note that in any k-predicate in Π_D^n, the constant situation is a sequence of k actions.

Given a domain description D denote by D^r the sub-domain of D obtained when we remove from D any value proposition.

Definition B.16
For any $n > 0$, we will say that a pair (Φ, Σ) where Φ is an interpretation and Σ a situation, is an $n+1$*-specific model* of D if and only if it is an n-specific model of D and for any sequence of actions $seq_n = a_1, \ldots, a_n$, $(\Phi, \Gamma_\Phi([seq_n], \Sigma))$ is a 1-specific model of D^r (i.e. D minus the value propositions). (Φ, Σ) will be a specific model of D if it is an n-specific model of D for any $n \geqslant 1$.

Definition B.17
Given a sequence of actions $seq = a_1, \ldots, a_n$ and a situation constant s, $res((seq), s)$ denotes the situation constant $res(a_n, \ldots, res(a_1, s))$, $seq_\emptyset$ denotes the empty sequence and $res((seq_\emptyset), s)$ will be equal to s. Let Act_n be the set of all the sequences of n actions. For any set of literals A we take $\sigma_{(A,(seq))}$ as the state such that $f \in \sigma_{(A,(seq))} \Leftrightarrow holds(f, res((seq), s_0)) \in A$.

Definition B.18
Given a pair (Φ, Σ) of interpretation and situation, $n \geqslant 0$ and the situation constant s_0, we will denote by $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ the following family of sets:

- if $n = 1$, $\mathbf{A}^1_{(\Phi,\Sigma,s_0)} = \mathbf{A}_{(\Phi,\Sigma,s_0)}$
- If $n \geqslant 1$, let for any set A of $(\leqslant n+1)$-predicates, A_n and A_1 denote the sets of $(\leqslant n)$-predicates in A and $(n+1)$-predicates in A (resp.). Then, $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ is defined as a family of sets A of $(\leqslant n+1)$-predicates, such that the following is satisfied:
  1. $A_n \in \mathbf{A}^n_{(\Phi,\Sigma,s_0)}$.
  2. If $holds(true, (seq_n, s_0))$ is in A_n then $A_1 = \bigcup_{seq_n \in Act_n} A_{(seq_n)}$ with $A_{(seq_n)}$ the set of all the predicates of the form $holds(true, res((seq_n, a), s_0))$ or $holds(f, res((seq_n, a), s_0))$
  3. If $holds(true, (seq_n, s_0))$ is not in A_n then $A_1 = A_n \cup \bigcup_{seq_n \in Act_n} A_{(seq_n)}$ where $A_{(seq_n)}$ is $A'_{(\Phi_0, \sigma_{(A_n, seq_n)}, res((seq_n), s_0))}$ for some $(\Phi_0, \sigma_{(A_n,(seq_n))})$ in $Asso_{D^r}(\Phi, \Gamma_\Phi((seq_n), \Sigma)$.

Lemma B.19
Given $n > 0$, a consistent domain description D, a pair (Φ, Σ), n-specific model of D, the initial situation constant s_0. We will have that for any fluent literal f and any sequence of actions $seq = a_1, \ldots, a_n$, $holds(f, res((seq), s_0)$ holds in $\mathbf{A}^n_{(\Phi,\Sigma,s_0)}$ if and only if $f \in \Gamma_\Phi([seq], \Sigma)$.

Proof

The proof is by induction on n and it is straightforward from the definition of $\mathbf{A}^{n+1}_{(\Phi,\Sigma,S)}$.

The next corollary proves by induction on the length of the sequence of actions that the logic program translation is sound and complete for the execution of a sequence of actions.

Corollary B.20
Given a consistent domain description D and $n > 0$. $\mathbf{A}$ is a world view of Π^n_D if and only if there exists a pair (Φ, Σ), n-specific model of D such that

$$\mathbf{A} = \mathbf{A}^n_{(\Phi,\Sigma,s_0)}$$

Proof

- The base case ($n = 0$) follows from B.13 and B.14.
- Suppose the result is valid for any $m \leqslant n$.
- ($\Rightarrow$) Let $\mathbf{A}$ be a world view of Π^{n+1}_D. As in Definition B.18, define for any $A \in \mathbf{A}$, A_n to be the subset of A restricted to those predicates in A involving just $\leqslant n$-predicates, so $\mathbf{A}_n = \{A_n : A \in \mathbf{A}\}$ will be a world view of Π^n_D, and by inductive hypothesis there is a pair (Φ_1, Σ), n-specific model of D such that $\mathbf{A}_n = \mathbf{A}^n_{(\Phi_1,\Sigma,s_0)}$.

We first define an interpretation Φ such that (Φ, Σ) is $n + 1$-specific model of D. Given an action a, let A'' be the set $\{holds(f, res((seq_n), s_0)) \in A_n : holds(true, res((seq_n), s_0)) \notin A_n \wedge seq_n \in Act_n\}$ and A_1 be A'' union

$$\begin{aligned}&[\{holds(f, res((seq_n, a), s_0)) : seq_n \text{ is any sequence of } n \text{ actions}\}\\ &\cup\{ab(f, a, res((seq_n), s_0)) : seq_n \text{ is any sequence of } n \text{ actions}\}\\ &\cup\{holds(true, res((seq_n, a), s_0)) : seq_n \text{ is any sequence of } n \text{ actions}\}] \cap A\end{aligned}$$

Note that $A = A_n \cup A_1$. By the splitting lemma, A_1 is a belief set of $A'' \cup \bigcup_{seq_n \in Act_n} [\Pi^1_{(D^r, res((seq_n), s_0))}]^A_{\mathbf{A}}$.

Hence by B.14 there exists an interpretation Φ_2 and a 0-interpretation Φ_0 such that for any sequence of n actions seq_n with $holds(true, res((seq_n), s_0)) \notin A_n$, the following properties are satisfied:

1. $(\Phi_2, \Gamma_{\Phi_1}([seq_n], \Sigma))$ is a 1-specific model of D^r.
2. $(\Phi_0, \sigma_{(A_n,(seq_n))})$ is in $Asso_{D^r}(\Phi_2, \Gamma_{\Phi_1}([seq_n], \Sigma))$.
3. $A_1 =$
   $\bigcup\{A'_{(\Phi_0,\sigma_{(A_n,(seq_n))}, res((seq_n),s_0))} : seq_n \in Act_n \wedge holds(true, res((seq_n), s_0)) \notin A_n\} \cup A''$.

Defining Φ such that for any sequence of actions $seq_n \in Act_n$ if the atomic formula $holds(true, res((seq_n), s_0))$ is not in A_n, then $\Gamma_\Phi([seq_n], \Sigma)$ is equal to $\Gamma_{\Phi_1}([seq_n], \Sigma)$, we will have that (Φ, Σ) is an n-specific model of D. Moreover if we take Φ such that for any action a and any sequence of actions $seq_n \in Act_n$, $\Phi(a, \Gamma_\Phi([seq_n], \Sigma)) = \Phi_2(a, \Gamma_\Phi([seq_n], \Sigma))$, we will have that for any sequence of actions $seq_n \in Act_n$, $(\Phi, \Gamma_\Phi([seq_n], \Sigma))$ is a 1-specific model of D^r. Therefore, (Φ, Σ) is an $n + 1$-specific model of D.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

It is clear by construction that either the predicate $holds(true, res((seq_n), s_0))$ is a member of A_n, and by the inertia rules, A_1 is the set of predicates of the form $holds(true, res((seq_n, a), s_0))$ and $holds(f, res((seq_n, a), s_0))$ with a an action, (seq_n) a sequence of n actions and f a fluent literal, or $holds(true, res((seq_n), s_0)) \notin A_n$ and for any sequence of n actions seq_n the following propositions hold:

1. $(\Phi_0, \sigma_{(A_n,(seq_n))}) \in Asso_{D^r}(\Phi, \Gamma_\Phi([seq_n], \Sigma))$,
2. $A_1 = A'_{(\Phi_0,\sigma_{(A_n,(seqn))}, res((seq_n),s_0))}$.

Therefore by definition of $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$, we have proved that $A \in \mathbf{A}$ if and only if it is in $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$.

($\Leftarrow$) Reciprocally, let (Φ, Σ) be an $(n+1)$-specific model of D. Then, by B.13, induction and by the splitting lemma, to prove that $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ is a world view of Π^{n+1}_D, it suffices to show that for any A, $A \in \mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ if and only if $A = A_n \cup B_A$ where A_n is the set of predicates in A of $\leqslant n$-complexity and B_A is a belief set of the program

$$\Pi = A_n \cup [\Pi^{n+1}_D - \Pi^n_D]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}} = A_n \cup [\bigcup_{seq_n \in Act_n} \Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$$

To prove this, we will show that the beliefs sets of the program Π are exactly the sets of the form $A_n \cup A_1$ defined in B.18.

First note that the belief sets of $[\bigcup_{seq_n \in Act_n} \Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ are the unions of belief sets of $[\Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ where seq_n is varying over the set of sequences of n actions. This is so because these programs are independent of each other.

Hence, we will calculate the belief sets of each program $\Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ where $seq_n \in Act_n$ and we will prove that the set of these union is $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$.

To show this, let seq_n be a sequence of n actions. Then there are two possible cases:

1. $holds(true, res((seq), s_0)) \in A_n$, and $A_n \cup [\Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ has only the belief set $B_A = A_n \cup A_1$ with A_1 the set of predicates of the form $holds(true, res((seq, a), s_0))$ or $holds(f, res((seq, a), s_0))$ where a is any action and f is any fluent literal.
2. $holds(true, res((seq), s_0)) \notin A_n$ and by theorem [B.14], any belief set of $A_n \cup [\Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ is equal to $A_n \cup A'_{(\Phi_0,\sigma_{(A_n,seqn)}, res((seq_n),s_0))}$ for some 0-interpretation such that $(\Phi_0, \sigma_{(A_n,seq_n)}) \in Asso_{D^r}(\Phi, \Gamma_\Phi(([seq_n, a], \Sigma))$.

Therefore the belief sets of $A_n \cup [\Pi^1_{(D^r, res((seq_n),s_0))}]_{\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}}$ are precisely the elements on $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$, and this implies that $\mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ is a world view of Π^{n+1}_D.

Definition B.21

Given a domain description D, an initial situation constant s_0 and a specific-model (Φ, Σ) of D, we will denote by $\mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}$ the following family of sets, $A \in \mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}$ if and only if for any $n \geqslant 1$ there exist $A_n \in \mathbf{A}^{n}_{(\Phi,\Sigma,s_0)}$ and $A_{n+1} \in \mathbf{A}^{n+1}_{(\Phi,\Sigma,s_0)}$ with $A_n \subseteq A_{n+1}$, and $A = \bigcup_{n \geqslant 1} A_n$ We will denote by $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ the family of sets A, such that A is the union of sets A_Π and A_Q where A_Π is an element of $\mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}$ and A_Q is a belief set of $A_\Pi \cup [Q]^{A_\Pi}_{\mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}}$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Lemma B.22
Given a consistent domain description D and the initial situation constant s_0, $\mathbf{A}$ is a world view of Π_D, if and only if there exists a specific model (Φ, Σ) of D such that $\mathbf{A} = \mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$.

Proof

Given a world view $\mathbf{A}$ of Π^{ω}_{D}, and $A \in \mathbf{A}$, we will denote by A_Π the subset of A restricted to those predicates in A of the form $holds(f, s)$ or $ab(f, a, s)$ where s is a situation constant and a is an action, then the set $\mathbf{A}_\Pi = \{A_\Pi : A \in \mathbf{A}\}$ is a world view of Π_D, and by (B.20) there exists a specific model of Π_D such that $\mathbf{A}_\Pi$ is equal to $\mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}$, since $[Q]^{A}_{\mathbf{A}}$ is equal to $[Q]^{A}_{\mathbf{A}^{\omega}_{(\Phi,\Sigma,s_0)}}$, applying the splitting lemma to $[\Pi^{Q}_{D}]^{A}_{\mathbf{A}} = [\Pi_D]^{A}_{\mathbf{A}} \cup [Q]^{A}_{\mathbf{A}}$ we have the result.

Theorem B.23
Let D be a domain description, s_0 be an initial constant situation and (Φ, Σ) be a specific model of D. Then, given a sequence of actions $seq = a_1, \ldots, a_k$, and the situation constant $s = res((seq), s_0)$, we will have that, for any situation constant s_1 and any plan β, there exists a sequence of actions $seq_{(\beta,s)}$ such that $\Gamma_\Phi([seq, seq_{(\beta,s)}], \Sigma)$ is equal to $\Gamma_\Phi(\beta, \Gamma_\Phi([seq], \Sigma)$ and for any $A \in \mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$, $find\_situation(\beta, s, s_1)$ belongs to A if and only if $s_1 = res((seq, seq_{(\beta,s)}), s_0)$.

Proof

The proof will be a double induction, using the loop nesting in the plan and the complexity of the plans. Thus, the while-complexity of a plan β, denoted by $wcomp(\beta)$, is 0 if β is either the empty plan [] or an action a, $max(wcomp(\alpha_1), wcopm(\alpha_2)$ If $\beta =$ **if** φ**then** α_1**else** α_2 or $\beta = [\alpha_1|\alpha_2]$, $1 + wcomp(\alpha)$ if $\beta =$ **while** φ **do** α.

Taking $wcomp(\beta) = 0$ we will do induction on $comp(\beta)$.

- If $\beta = []$, the result is immediate, because
$$\Gamma_\Phi(\beta, \Gamma_\Phi([seq], \Sigma)) = \Gamma_\Phi([], \Gamma_\Phi([seq], \Sigma)),$$
and for any $A \in \mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ $find\_situation([\beta], s, s_1)$ is in A if and only if $s_1 = s$. Therefore taking $seq_{(\beta,s)} = seq_\emptyset$ the claim follows.
- Suppose the theorem is valid for $\leqslant n$-plans.
- Let β be an $(n + 1)$-plan. Then we have the following possibilities:
(I) $\beta = [a|\alpha]$ where α is an n-plan and a is an action. Then, by inductive hypothesis $\Gamma_\Phi(\alpha, \Gamma_\Phi([seq, a], \Sigma))$ is equal to $\Gamma_\Phi([seq, a, seq_{(\alpha,res(a,s))}], \Sigma)$, therefore we will have that
$$\begin{aligned}&\Gamma_\Phi(\beta, \Gamma_\Phi([seq], \Sigma)) =\\ &\Gamma_\Phi([a, \alpha], \Gamma_\Phi([seq], \Sigma)) =\\ &\Gamma_\Phi(\alpha, \Gamma_\Phi([seq, a], \Sigma)) =\\ &\Gamma_\Phi([seq_{(\alpha,res(a,s))}], \Gamma_\Phi([seq, a], \Sigma)) =\\ &\Gamma_\Phi([a, seq_{(\alpha,res(a,s))}], \Gamma_\Phi([seq], \Sigma)) =\\ &\Gamma_\Phi([seq_{(\beta,s)}], \Gamma_\Phi([seq], \Sigma)) =\\ &\Gamma_\Phi([seq, seq_{(\beta,s)}], \Sigma).\end{aligned}$$

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Moreover for any A in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$, $find\_situation(\beta, s, s_1)$ is in A if and only if $find\_situation(\alpha, res(a, s), s_1)$ is in A.

Thus, if we take $seq_{(\beta,s)} = (seq_{(\alpha,res(a,s))}, a)$, since by inductive hypothesis $find\_situation([\alpha], res(a, s), s_1)$ is in A if and only if s_1 is equal to $res(((seq, a), seq_{(\alpha,res(a,s))}), s_0)$, we have that $find\_situation([\beta], s, s_1)$ is in A if and only if $s_1 = res((seq, (a, seq_{(\alpha,s)})), s_0)$ which is equal to the situation $res((seq, seq_{(\beta,s)}), s_0)$, and

$$\begin{gathered}\Gamma_\Phi([\beta], \Gamma_\Phi([seq], \Sigma)) = \\ \Gamma_\Phi([seq, seq_{(\beta,s)}], \Gamma_\Phi([seq], \Sigma)) = \\ \Gamma_\Phi([seq_{(\beta,s)}], \Gamma_\Phi([seq], \Sigma))\end{gathered}$$

(II) $\beta = [$ **if** φ **then** $\alpha_1, \alpha_2]$ where α_1, α_2 are n_1 and n_2-plans respectively, with $n_1 + n_2 = n$ then, we have that for any A in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$, $find\_situation(\beta, s, s_1)$ is in A if and only if either:

(i) $holds(\bar{\varphi}, s)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ and $find\_situation([\alpha_2], s, s_1)$ is in A, or
(ii) $holds(\varphi, s)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ and $find\_situation([\alpha_1, \alpha_2], s, s_1)$ is in A.

By inductive hypothesis $\Gamma_\Phi(\alpha_2, \Gamma_\Phi([seq], \Sigma))$ is equal to $\Gamma_\Phi([seq, seq_{(\alpha_2,s)}], \Sigma)$ and $\Gamma_\Phi([\alpha_1, \alpha_2], \Gamma_\Phi([seq], \Sigma))$ is equal to $\Gamma_\Phi([seq, seq_{([\alpha_1,\alpha_2],s)}], \Sigma)$ for the sequences of actions $seq_{(\alpha_2,s)}$ and $seq_{([\alpha_1,\alpha_2],s)}$

Hence by [B.19] and inductive hypothesis, $find\_situation(\beta, s, s_1)$ is in A if and only if either

(i) $\bar{\varphi}$ holds in $\Gamma_\Phi([seq], \Sigma)$, and $s_1 = res((seq, seq_{(\alpha_2,s)}), s_0)$. or
(ii) φ holds in $\Gamma_\Phi([seq], \Sigma)$, and $s_1 = res((seq, seq_{([\alpha_1,\alpha_2],s)}), s_0)$.

Thus, if we take $seq_{(\beta,s)} = seq_{(\alpha_2,s)}$ in case (i), and in case (ii) $seq_{(\beta,s)}$ equal to $seq_{([\alpha_1,\alpha_2],s)}$ we will have that $\Gamma_\Phi(\beta, \Gamma_\Phi([seq], \Sigma)) = \Gamma_\Phi([seq, seq_{(\beta,s)}], \Sigma))$ and $find\_situation(\beta, s, s_1)$, belongs to A if and only if s_1 is equal to the situation $res((seq, seq_{(\beta,s)}), s_0)$.

(III) $\beta = [$ **if** φ **then** α_1 **else** $\alpha'_1, \alpha_2]$, where α_1, α'_1 and α_2 are n_1, n'_1 and n_2-plans (resp.) with $max(n_1, n'_1) + n_2 = n$. This case is similar to the previous one.

(IV) $\beta = [$ **while** φ **do** $\alpha_1, \alpha_2]$, where α_1 and α_2 are n_1 and n_2-plans (resp.), with $n_1 + n_2 = n$. Here we may suppose by inductive hypothesis that for any $k \geqslant 0$ the plans $[\alpha_1^k]$ and $[\alpha_1^k, \alpha_2]$ verify the theorem, and we will denote by $seq_{(k,\alpha_1)}$ the sequence $seq_{(\alpha_1,s)}$ and by $seq_{(k,\alpha_2)}$ the sequence $seq_{([\alpha_2],res((seq_{([\alpha_1^k],s)}),s_0))}$.

Using the fix-point operator $\mathbf{T}_\Pi$ defined by $\mathbf{T}_\Pi(I) = \{p : \exists$ a rule $p \leftarrow q_1, \ldots, q_n$ in Π with each q_i a fact in $I\}$, for any positive logic program Π, we know that if $\mathbf{T}_\Pi{\uparrow}^1$ is defined to be equal to $\mathbf{T}_\Pi(\emptyset)$ and $\mathbf{T}_\Pi{\uparrow}^{k+1} = \mathbf{T}_\Pi(\mathbf{T}_\Pi{\uparrow}^k)$, then $\mathbf{T}_\Pi{\uparrow}^\omega$, which is the set $\bigcup_{k \geqslant 1} \mathbf{T}_\Pi{\uparrow}^k$, is a fix-point for $\mathbf{T}_\Pi$. Moreover A is a belief set for $[\Pi^\omega_{(D,s_0)}]^A_{\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}}$ if and only if

$$A = \mathbf{T}_{[\Pi^\omega_{(D,s_0)}]^A_{\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}}}{\uparrow}^\omega .$$

Therefore, $h = find\_situation(\beta, s, s_1)$ is in A if and only if there exists $k \geqslant 0$ such that h is in $\mathbf{T}_{[\Pi^\omega_{(D,s_0)}]^A_{\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}}}{\uparrow}^{k+1}$.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Let k_0 be the minimum k such that h belongs to $\mathbf{T}_{[\Pi^{\omega}_{(D,s_0)}]^{A}_{\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}}} \uparrow^{k+1}$. We have that $h \in \mathbf{T}_{[\Pi^{\omega}_{(D,s_0)}]^{A}_{\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}}} \uparrow^{k_0+1}$, if and only if there exists m such that the following properties are satisfied:

(1) For any $j < m$ $holds(\varphi, res((seq, seq_{(j,\alpha_1)}), s_0)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$
(2) $holds(\bar{\varphi}, s')$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$
(3) $find\_situation([\alpha_1^m], s, s') \in A$
(4) $find\_situation([\alpha_2], s', s_1) \in A$

If we fix m with properties (1), (2), (3) and (4), we have by inductive hypothesis and [B.19] that, $h \in A$ if and only if φ holds in $\Gamma_\Phi([seq, seq_{(m,\alpha_1)}], s_0)$, $s' = res((seq_{(m,\alpha_1)}), s_0)$ and $s_1 = res((seq_{(m,\alpha_2)}), s_0)$. Hence taking $seq_{(\beta,s)} = seq_{(m,\alpha_2)}$, we will have that $\Gamma_\Phi(\beta, \Gamma_\Phi([seq], \Sigma)$ is equal to $\Gamma_\Phi([seq, seq_{(\beta,s)}], \Sigma)$ and $h \in A$ if and only if $s_1 = res((seq_{(\beta,s)}), s_0)$.

The inductive step on $wcomp(\beta)$ follows the same reasoning as in the base case.

Corollary B.24
Given a consistent domain description D, the initial situation constant s_0 and a specific model of D, (Φ, Σ), we will have that for any fluent f and any plan β, $hold\_after\_plan(f, \beta)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ if and only if $f \in \Gamma_\Phi(\beta, \Sigma)$.

Proof

Let $seq_{(\beta,s_0)}$ be the sequence of actions described in [B.23], such that $\Gamma_\Phi(\beta, \Sigma) = \Gamma_\Phi([seq_{(\beta,s_0)}], \Sigma)$ and $find\_situation(\beta, s_0, s_1)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ if and only if $s_1 = res((seq_{(\beta,s_0)}), s_0)$. Then since $hold\_after\_plan(f, \beta)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ if and only if $find\_situation(\beta, s_0, s_1)$ and $holds(f, s_1)$ hold in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$, we have by [B.19] $hold\_after\_plan(f, \beta)$ holds in $\mathbf{A}^{Q}_{(\Phi,\Sigma,s_0)}$ if and only if $f \in \Gamma_\Phi([seq_{(\beta,s_0)}], \Sigma) = \Gamma_\Phi(\beta, \Sigma)$.

Hence by [B.22] we have the following:

Corollary B.25
Given a simple and consistent domain description D and a plan β. $D \models F$ **after**β if and only if $\Pi^{Q}_{D} \models hold\_after\_plan(F, \beta)$.

Theorem 7.3 Given a simple consistent domain description D and a plan β. $D \models F$ **after**β if and only if $\Pi^{Q}_{D} \models hold\_after\_plan(F, \beta)$.

Proof Direct from Corollary B.25.

References

Aho, A. and Ullman, J. (1995) *Foundations of Computer Science, C edition.* Computer Science Press.

Baral, C. and Gelfond, M. (1997) Reasoning about effects of concurrent actions. *J. Logic Programming*, **31**, 85–118.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press

Baral, C. and Son, T. (1997) Approximate reasoning about actions in presence of sensing and incomplete information. In: Maluszynsky, J. (ed.), *Proceedings of International Logic Programming Symposium*, pp. 387–401.

Baral, C., Gelfond, M. and Provetti, A. (1997) Representing actions: Laws, observations and hypotheses. *J. Logic Programming*, **31**(1–3), 201–244.

Cousot, P. (1990) Methods and logics for proving programs. In: van Leeuwen, J. (ed.), *Handbook of theoretical computer science*, vol. B, pp. 841–993. MIT Press/Elsevier.

Davey, B. A. and Priestley, H. A. (1990) *Introduction to Lattice Theory*. Cambridge Mathematical Text Books.

Geffner, H. and Bonet, B. (1998) High-level planning and control with incomplete information using POMDP's. *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics.*

Gelfond, M. (1994) Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 98–116.

Gelfond, M. and Lifschitz, V. (1991) Classical negation in logic programs and disjunctive databases. *New Generation Comput.* **9**(3–4), 365–387.

Gelfond, M. and Lifschitz, V. (1993) Representing actions and change by logic programs. *J. Logic Programming*, **17**(2–4), 301–323.

Gelfond, M. and Przymusinska, H. (1991) Definitions of epistemic specification. In: Nerode, A., Marek, V. W. and Subrahmanian, V. S. (eds.), *Proceedings of 1st International Workshop on Logic Programming and Non-monotonic Reasoning*, p. 249–263.

Kartha, G. N. and Lifschitz, V. (1994) Actions with indirect effects. In: Doyle, J., Sandewall, E. and Torasso, P. (eds.), *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 341–350.

Kartha, G. N. and Lifschitz, V. (1997) Representing actions: Indeterminacy and ramification. *Artificial Intelligence*, **95**, 409–443.

Levesque, H. (1996) What is planning in the presence of sensing? *Proceedings of 13th National Conference on Artificial Intelligence*, vol. 2, pp. 1139–1146.

Lifschitz, V. (1996) Two components of an action language. *Working Papers of the 3rd Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense '96).*

Lifschitz, V. and Turner, H. (1994) Splitting a logic program. In: van Hentenryck, P. (ed.), *Proceedings of the 11th International Conference of Logic Programming*, pp. 23–37.

Lobo, J., Minker, J. and Rajasekar, A. (1992) *Foundations of Disjunctive Logic Programming*. MIT Press.

Lobo, J., Mendez, G. and Taylor, S. (1997) Adding knowledge to the action description language $\mathcal{A}$. *Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 454–459. AAAI Press.

Manna, Z. and Waldinger, R. (1987) How to clear a block: A theory of plans. *J. Automated Reasoning*, **3**, 343–377.

Marek, W. and Truszczynski, M. (1991) Autoepistemic logic. *J. ACM*, **38**(3), 588–619.

McCain, N. and Turner, H. (1997) Causal theories of action and change. *Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 460–465.

Moore, R. C. (1985) Semantical considerations on non-monotonic reasoning. *Artificial Intelligence*, **28**, 75–94.

Scherl, R. and Levesque, H. (1993) The frame problem and knowledge-producing actions. *Proceedings of the 11th National Conference on Artificial Intelligence*, pp. 689–695.

Thielscher, M. (1994) Representing actions in equational logic programming. In: van Hentenryck, P. (ed.), *Proceedings of the International Conference on Logic Programming*, pp. 207–224.

https://doi.org/10.1017/S1471068400001022 Published online by Cambridge University Press