T. C. Son
P. H. Tu
X. Zhang

# Reasoning about Sensing Actions in Domains with Multi-Valued Fluents

**Abstract.** In this paper, we discuss the weakness of current action languages for sensing actions with respect to modeling domains with multi-valued fluents. To address this problem, we propose a language with sensing actions and multi-valued fluents, called $\mathcal{AM}_K$, provide a transition function based semantics for the language, and demonstrate its use through several examples from the literature. We then define the entailment relationship between action theories and queries in $\mathcal{AM}_K$, denoted by $\models_{\mathcal{AM}_K}$, and discuss some properties about $\mathcal{AM}_K$.

*Keywords*: Actions Languages, Sensing Actions, Multi-valued Fluents, Incomplete Information.

## 1. Introduction and Motivation

Sensing (or knowledge producing) actions have been the topic of intensive research in reasoning about action and change and planning (see e.g. [7, 9, 11–14]). Situation calculus based approaches such as [7, 11, 12] can be used in domains with *multi-valued fluents* since functional fluents are allowed in situation calculus. Additionally, in [7], we can find several examples in the UNIX domain articulating the need for dealing with *sensing actions* in domains with multi-valued fluents.

On the other hand, approaches to reasoning about sensing actions using high-level description language by extending the language $\mathcal{A}$ in [4, 5] such as [9, 13] concentrate on providing solution to the frame problem for domains with Boolean fluents and sensing actions. In [2], the approach of [13] is extended to deal with domains with state constraints (or static causal laws). As such, the simplification to domains with Boolean fluents does not necessarily limit the expressiveness of the languages developed along this line with respect to domains with multi-valued fluents since these domains can be easily represented using Boolean fluents with state constraints. Although adequate for certain representational and reasoning purposes, this solution is too cumbersome and not intuitive. We will illustrate this problem in the next example.

Consider the traffic light domain that consists of a traffic light and the only action *look*. Looking at the light will tell us whether the traffic light is either *red, yellow*, or *green*. This domain can be best described by the action *look* and a fluent denoting the color of a traffic light whose value

*Studia Logica* **68**: 1–26, 2001.

is either *red, yellow*, or *green*. In the case of Boolean fluents, we may need to use three Boolean fluents *is_red, is_yellow*, and *is_green* to represent the fact that the traffic light is either red, yellow, or green and in addition, a set of state constraints stating that it has only one color at a time such as[1] "$is\_red$ **if** $\neg is\_yellow, \neg is\_green$", "$\neg is\_red$ **if** $is\_yellow$", "$\neg is\_red$ **if** $is\_green$", etc. To complete the description, we need to specify the effect of the action *look*. Using the language $\mathcal{A}_K$ in [13], we may need to introduce at least two out of the following three k-propositions[2], "*look* **determines** $is\_red$", "*look* **determines** $is\_green$", and "*look* **determines** $is\_yellow$". This is because the third fluent can be uniquely determined from the other two. Although satisfactory for predicting and reasoning about the effects of the action *look*, this solution is clearly not intuitive and cumbersome.

It is easy to see that a rather better and more intuitive description for the the traffic light domain would be one consisting of a single fluent, say *color*, whose domain is {*red, green, yellow*}, and an k-proposition "*look* **determines** *color*". From the representational perspective, this is a clear call for a direct treatment of multi-valued fluents. Furthermore, as static causal laws are a source of non-determinism and thus could potentially make the reasoning process (and therefore, the planning process) harder, it is advantageous to eliminate unnecessary static causal laws like those that express the fact that the value of *color* is unique.

*Our main goal in this paper is to develop a language for sensing actions with multi-valued fluents.* We call this language $\mathcal{AM}_K$ as it is an extension of the language $\mathcal{A}_K$ to multi-valued fluents. To the best of our knowledge, $\mathcal{AM}_K$ is the first high-level, transition function based semantics language that allows both sensing actions and multi-valued fluents. We note that among the variants of the language $\mathcal{A}$, the action language $\mathcal{AR}$ in [6] introduces nonpropositional fluents that are similar to our multi-valued fluents but does not consider sensing actions. Multi-valued fluents could also be encoded in the language $\mathcal{K}$ [3] but this language does not allow sensing actions either.

In the next sections we will define $\mathcal{AM}_K$ and illustrate its use through several examples taken from the literature. We will then discuss the relationship between the two languages $\mathcal{AM}_K$ and $\mathcal{A}_K$ and conclude the paper with a short discussion on future work.

---

[1] Though seemingly redundant, such constraints are needed in the current transition function based approaches to reasoning about sensing actions.

[2] In [9], **causes_to_knows** is used instead of **determines**.

## 2. Language $\mathcal{AM}_K$

This section begins with the $\mathcal{AM}_K$ syntax, and then illustrates its use through some well-known planning problems from the literature. Afterward, the notions of observations, queries and plans are introduced, followed by the semantics of $\mathcal{AM}_K$.

### 2.1. Syntax

The alphabet of a domain description in $\mathcal{AM}_K$ consists of a set of action names $\mathbf{A}$, and a set of fluent names $\mathbf{F}$. Each fluent $f$ in $\mathbf{F}$ has a domain $dom(f)$ associated with it that prescribes what value $f$ can take.

An *atom* is of the form $f = v$ where $f$ is a fluent and $v \in dom(f)$. A *fluent literal* (or literal for short) is an atom or its negation and a *(fluent) formula* is a propositional formula constructed from fluent literals. For convenience, we use $f \neq v$ to denote the negation of $f = v$ (or $\neg(f = v)$). The two constants *true* and *false* are used to represent *tautology* and *falsity*, respectively. To describe a domain description, we use propositions of the following forms

$$\textbf{executable } a \textbf{ if } \varphi \tag{i}$$

$$a \textbf{ causes } f = v \textbf{ if } \varphi \tag{ii}$$

$$a \textbf{ partitions } \ f \textbf{ into } \ \{V_j\}_{j \in S} \tag{iii}$$

$$f = v \textbf{ if } \varphi \tag{iv}$$

where

- $a$ is an action name in $\mathbf{A}$;
- $f$ is a fluent in $\mathbf{F}$ and $v$ is some value belonging to $dom(f)$;
- $\varphi$ is a formula; and
- $S$ is a set of indices and $\{V_j\}_{j \in S}$ is a partition of the domain of the fluent $f$, i.e., a (possibly infinite) sequence of pairwise disjoint sets of values belonging to $dom(f)$ such that $\cup_{j \in S} V_j = dom(f)$.

Intuitively, the above propositions describe the executability conditions, effects of an actions, and state constraints. Propositions of the form (i) state the conditions for which $a$ is executable. It says that for $a$ to be executable, the formula $\varphi$ must hold (precise meaning of *hold* is given in Section 2.4). Propositions of the form (ii) describe the conditional effects of a *non-sensing action* on the value of a fluent. It says that the execution of $a$ in any situation where the condition $\varphi$ holds will cause the fluent $f$ to take

the value $v$. Propositions of the form (iii) describe the effect of a *sensing action*. It says that executing an action $a$ will help an agent to partially determine the value of the fluent $f$ in that it knows the possible values of $f$. In what follows, we will use $a$ **determines** $f$ as a shorthand for a proposition of the form (iii) where each of the $V_j$'s contains exactly one element, i.e., executing $a$ helps the agent to precisely determine $f$'s value. Propositions of the form (iv) describe the state constraints, i.e., the relationship between fluents. It states that the value of $f$ is $v$ in any situation where $\varphi$ holds.

It is worth noting here that we do not allow negative literals of the form $f \neq v$ to appear in the place of $f = v$ in propositions (ii) and (iv). Allowing this would mean considering domains with non-deterministic actions which is not the main topic of this paper and deserves a throughout discussion. We leave it as one of the topics of our future research.

Two effect propositions $a$ **causes** $f = v$ **if** $\varphi$ and $a$ **causes** $f = v'$ **if** $\varphi'$, where $v, v' \in dom(f)$ and $v \neq v'$, are called *contradictory* if there exists a state in which both $\varphi$ and $\varphi'$ hold simultaneously.

A *domain description* $D$ is a set of propositions of the forms (i)-(iii) without contradictory effect propositions. For simplicity, we assume that for every domain description $D$, each action $a$ occurs in at most one proposition of the form (i) and by default **executable** $a$ **if** $true$ belongs to $D$ unless otherwise specified. We also assume that non-sensing actions and sensing actions do not overlap each other, and each sensing action $a$ occurs in at most one proposition of the form (iii).

With the introduction of multi-valued fluents, the traffic light domain representation now becomes trivial. It can be represented by one proposition

$$look \textbf{ partitions } color \textbf{ into } \{red\}, \{green\}, \{yellow\}$$

with the obvious meanings associated to the action *look* and the fluent *color*. We now demonstrate the use of $\mathcal{AM}_K$ through some examples in the literature. The first one, which is taken from [7], is about the UNIX domain.

Example 1 (UNIX domain). In this domain, actions are UNIX commands such as $ls, cd, ping$, etc. The domain, denoted by $D_u$, can be represented in $\mathcal{AM}_K$ by the following propositions

$$cd(D) \textbf{ causes } curdir = D \textbf{ if } exists(D)$$
$$ls \textbf{ determines } files(curdir)$$
$$ls(X) \textbf{ determines } exists(X)$$
$$ping(M) \textbf{ determines } alive(M)$$
$$exists(D) \textbf{ if } exists(D/X)$$

where $curdir$, $files(D)$, $exists(X)$, and $alive(M)$ are fluents denoting the current directory, the files in a directory $D$, the existence of a directory or a file $X$, and the aliveness of a machine $M$ respectively. The domains of the first two fluents are the the set of valid directory names and the set of valid file names correspondingly. The latter two are Boolean fluents.

The first proposition says that executing action $cd(D)$ will change the current directory ($curdir$) to $D$ if $D$ exists. The second proposition describes the effect of the (sensing) action $ls$: executing $ls$ allows us to determine the files in the current directory. The third proposition is similar to the previous one, except that it allows us to determine the existence of a particular file or directory. The next proposition describes the fact that executing *ping* tells us whether or not a particular machine is alive. Finally, the last proposition is a static causal law stating that a directory exists if one of its sub-directories or files exists. □

The next example, taken from [15], is about a patient whose sickness can be cured if the doctor is able to give her the right medication.

Example 2 (Illness Domain). In this domain, there are 5 different kinds of illnesses, $i_1, \ldots, i_5$, each of which needs a particular medication. A patient is sick and we need to find an appropriate cure for her. Taking right medication can make the patient get cured but using wrong medication is fatal.

Performing a throat culture will return either *red*, *blue*, or *white* which determine the group of illnesses that the patient has. Inspecting the color allows us to observe the color returned by a throat culture depending on the sickness of the patient. Taking a blood sample tells us whether the patient has a high white cell count that we can know after analyzing the blood. This domain can be represented in $\mathcal{AM}_K$ as follows. First, the set of fluents consists of the following fluents:

- $i$ (stands for *illness*) with $dom(i) = \{i_1, \ldots, i_5, none\}$; where *none* denotes that the patient is healthy;
- *color* with $dom(color) = \{red,\ blue,\ white\}$;
- *hc* (stands for *high blood count*), *tcd* (*throat culture test done*), *dead*, and *bsd* (*blood sample done*). All of them are Boolean fluents.

The set of actions is described next.

- *stain*: indicates that a throat culture is done, i.e., this action makes *tcd* becomes true;

- *inspect*: this action can be executed only when the throat culture is done and it tells us the color of the test, depending on the illness;
- *blood_sample*: this action makes *bsd* true;
- *analyze_blood*: is executable only when the blood sample is count and determines whether *hc* is true;
- $medicate(X)$: takes the medication $X$ where $X \in \{c_1, \ldots, c_5\}$;

The domain, denoted by $D_s$, consists of the following propositions

| | |
|---|---|
| **executable** *inspect* **if** *tcd* | *color=blue* **if** $(i{=}i_3 \vee i{=}i_4) \wedge tcd$ |
| **executable** *analyze_blood* **if** *bsd* | *color=red* **if** $(i{=}i_1 \vee i{=}i_2) \wedge tcd$ |
| *stain* **causes** *tcd* | *color=white* **if** $i{=}i_5 \wedge tcd$ |
| *blood_sample* **causes** *bsd* | $\neg hc$ **if** $(i{=}i_2 \vee i{=}i_4) \wedge bsd$ |
| $medicate(c_j)$ **causes** $i{=}none$ **if** $i{=}i_j$ | $hc$ **if** $(i{=}i_1 \vee i{=}i_3 \vee i{=}i_5) \wedge bsd$ |
| $medicate(c_j)$ **causes** *dead* **if** $i{\neq}i_j$ | |
| *analyze_blood* **determines** *hc* | |
| *inspect* **determines** *color* | |

where the last two propositions are for $j = 1, \ldots, 5$. □

In the next example, we consider a blocks world domain with two blocks.

Example 3 (Blocks World domain). Consider a blocks world with two blocks – $a$ and $b$, and a robot. The robot has an arm that can *pickup, putdown, stack,* or *unstack* a block; it can also *sense* whether it is holding block $a$ (likewise, block $b$) or not.

To represent this domain, we use fluents $loc(X)$ where $X \in \{a, b\}$ to denote the location of the block $X$. The domains of these fluents are $dom(loc(a)) = \{onTable, inHand, on(b)\}$ and $dom(loc(b)) = \{onTable, inHand, on(a)\}$.

The domain, denoted by $D_b$, consists of the following propositions

$$\textbf{executable } pickup(X) \textbf{ if } loc(X) = onTable$$
$$\textbf{executable } putdown(X) \textbf{ if } loc(X) = inHand$$
$$\textbf{executable } stack(X) \textbf{ if } loc(X) = inHand$$
$$\textbf{executable } unstack(X) \textbf{ if } loc(X) = on(Y)$$
$$pickup(X) \textbf{ causes } loc(X) = inHand \textbf{ if } loc(X) = onTable$$
$$putdown(X) \textbf{ causes } loc(X) = onTable \textbf{ if } loc(X) = inHand$$
$$stack(X) \textbf{ causes } loc(X) = on(Y) \textbf{ if } loc(X) = inHand$$
$$unstack(X) \textbf{ causes } loc(X) = inHand \textbf{ if } loc(X) = on(Y)$$
$$loc(X) = onTable \textbf{ if } loc(Y) = inHand \vee loc(Y) = on(X)$$
$$sense(X) \textbf{ partitions } \ loc(X) \textbf{ into } \ \{onTable, on(Y)\}, \{inHand\}$$

where $X$ and $Y$ are in $\{a, b\}$ and $X \neq Y$. □

The last example demonstrates that the partition created by a sensing action can be infinite.

EXAMPLE 4 (Gas Domain). Consider a simple domain that consists of only one action *look_at_indicator* and one fluent *gas_in_tank*. Looking at the gas indicator will tell us the amount of gasoline available in the tank. Furthermore, we know in advance the capacity of the tank is 20 gallons, i.e., the domain of the fluent *gas_in_tank* is [0,20].

This domain, denoted by $D_g$, can be described by a single proposition

*look_at_indicator* **determines** *gas_in_tank*

□

Note that there is a subtle difference between *look_at_indicator* (Example 4) and sensing actions in the previous examples. While the partition created by *gas_in_tank* is infinite, the partitions created by sensing actions in the previous examples are finite.

### 2.2. Observations

An *observation* in $\mathcal{AM}_K$ is of the form

**initially** $l$ (v)

where $l$ is a literal. When $l$ is of the form $f = v$ (resp. $f \neq v$), we say that (v) is a *positive* (resp. *negative*) observation. It is worth noting that there exists a difference between observations in the Boolean and multi-valued fluent settings. In the Boolean case, an arbitrary observation would allow us to know the precise value of the fluent appearing in it. A negative observation in multi-valued domains, say **initially** $f \neq v$ where $|dom(f)| > 2$, does not however allow us to know the precise value of $f$. It only limits the set of possible values of $f$ rather.

An *action theory* is a pair $(D, O)$ where $D$ is a domain description and $O$ is a set of observations. Again, for simplicity, we assume that each fluent in $D$ occurs in at most one observation in $O$.

For the UNIX domain, some of the observations may be the fact that the current directory is `/mydoc/papers`, the file name `paper.tex` is in the directory `/mydoc`, the machine named `church.domain` is alive, etc. This information can be represented by the following set of observations

$$O_u = \left\{ \begin{array}{l} \textbf{initially } curdir = /\texttt{mydoc/papers} \\ \textbf{initially } alive(\texttt{church.domain}) \\ \textbf{initially } in(\texttt{paper.tex}, files(/\texttt{mydoc})) \end{array} \right.$$

For the illness domain, we know that the patient is not dead but has some illness. We also know that none of the tests has been done. This information can be represented by the following set of observations

$$O_s = \begin{cases} \textbf{initially } i \neq none \\ \textbf{initially } \neg dead \\ \textbf{initially } \neg tcd \\ \textbf{initially } \neg bsd \end{cases}$$

For the block worlds domain, we assume that the robot knows that $a$ is initially on the table, i.e., $O_b = \{$ **initially** $loc(a) = onTable\}$.

For the gas domain, we initially know nothing about the gas available in the tank, i.e., the set of observations is empty and denoted by $O_g = \emptyset$.

### 2.3. Plans and Queries

As discussed in [8], in the presence of incomplete information and knowledge producing actions, we need to extend the notion of a plan from a sequence of actions so as to allow conditional statements. In this paper, we consider plans that are defined as follows. A *plan* is either

- an empty sequence of action, denoted by [ ];
- or a *sequential* structure of the form $[a; p]$ where $a$ is an action and $p$ is a plan.
- or a *if-then-else* structure of the form [ **if** $\varphi$ **then** $p_1$ **else** $p_2$] where $\varphi$ is a formula and $p_1$ and $p_2$ are plans;

Intuitively, the if-then-else plan is a branching statement where the agent evaluates the condition $\varphi$ with respect to its knowledge and decides on which plan to execute. If it knows that $\varphi$ is true (resp. false), it executes $p_1$ (resp. $p_2$). Otherwise ($\varphi$ is unknown), the if-then-else plan fails and the execution of the plan which contains this statement also fails.

There are two kinds of *queries* that we can ask our action theories. They are of the form

$$\textbf{knows } \varphi \textbf{ after } p \tag{vi}$$
$$\textbf{kwhether } \varphi \textbf{ after } p \tag{vii}$$

where $p$ is a plan and $\varphi$ is a formula. Intuitively, the first query is about asking if an action theory entails that $\varphi$ will be known to be true after executing the conditional plan $p$ in the initial situation, and the second

query is about asking if an action theory entails that $\varphi$ will be known after executing the plan $p$ in the initial situation.

As an example, intuitively $(D_s, O_s)$ entails **knows** $\neg dead$ **after** [] because we know that the patient is not dead initially. The theory, however, does not entail **kwhether** $(i = i_1)$ **after** [] because we do not know whether the patient has the sickness $i_1$ or not.

### 2.4. Semantics

Given a domain description $D$, an *interpretation* $s$ of $D$ assigns each fluent $f \in \mathbf{F}$ a value $v \in dom(f)$, denoted by $s(f) = v$. An interpretation $s$ satisfies an atom $f = v$ (resp. $f \neq v$) if $s(f) = v$ (resp. $s(f) \neq v$). When $s$ satisfies a literal $l$, we say that $l$ holds in $s$ and write $s \models l$. The truth value of a formula $\varphi$, denoted by $s(\varphi)$, with respect to an interpretation $s$ is defined as usual. When $s(\varphi)$ is true we say that $s$ satisfies $\varphi$ and write $s \models \varphi$.

An interpretation $s$ is a *state* if for every proposition of the form (iv), whenever $s \models \varphi$ holds, so does $s \models l$.

Two states $s_1$ and $s_2$ *agree* on a fluent $f$ with respect to a partition $\{V_j\}_{j \in S}$ of $f$, denoted by $s_1 \overset{f,\{V_j\}_{j\in S}}{\sim} s_2$, if $s_1(f) \in V_j$ iff $s_2(f) \in V_j$. In this case, $s_1$ and $s_2$ represent two possible worlds that an agent think he might be in when he does not know the exact value of the fluent $f$.

A *k-state* is a set of states. A *combined state* (or *c-state* for short) is a pair $\langle s, \Sigma \rangle$ where $s$ is a state and $\Sigma$ is a k-state. Intuitively, the first component (sometimes called top part) in a c-state $\langle s, \Sigma \rangle$ is the real state of the world and the second component (called bottom part) is the set of possible states which an agent believes it might be in. We say a c-state $\sigma = \langle s, \Sigma \rangle$ is *grounded* if $s \in \Sigma$. Intuitively, grounded c-states correspond to the assumption that the world state belongs to the set of states that the agent believes it might be in.

A formula $\varphi$ is known to be true in a c-state $\sigma = \langle s, \Sigma \rangle$ if it is known to be true in every state in $\Sigma$. $\varphi$ is known to be false in $\sigma = \langle s, \Sigma \rangle$ if it is known to be false in every state in $\Sigma$. Otherwise, $\varphi$ is unknown.

Before formally defining the transition function of a domain $D$, we introduce some more notations. A *(fluent) assignment* is of the form $f = v$ where $f$ is a fluent and $v$ is some value in $dom(f)$. A set of (fluent) assignments is said to be *domain-consistent* if it does not contain more than one assignment for every fluent. In what follows, whenever we refer to a set of assignments, we mean it is domain-consistent.

Given a set of assignments $S$ and a fluent $f$, the value of $f$ with respect to $S$, denoted by $S(f)$, is defined as follows. $S(f) = v$ if $f = v$ belongs to

$S$; otherwise, we say that $S(f)$ is unknown and write $S(f) = unknown$. $S$ satisfies an atom $f = v$, denoted by $S \models (f = v)$, if $S(f) = v$. $S$ satisfies $f \neq v$ if $S(f) = v'$ for some $v'$ in $dom(f) \setminus \{v\}$. The truth value of a formula $\varphi$ with respect to $S$ is defined as usual (note that it may be true, false, or unknown). We say that $S$ satisfies $\varphi$, denoted by $S \models \varphi$, if the truth value of $\varphi$ with respect to $S$ is true. $S$ is said to be closed under a set of domain constraints (form (iv)) $K$ if for every constraint $f = v$ **if** $\varphi$ in $K$, whenever $S \models \varphi$, so does $S \models (f = v)$. By $Cn(S \cup K)$ we denote the smallest set of assignments from $D$ that contains $S$ and is closed under $K$ and domain-consistent. Notice that $Cn(S \cup K)$ might not exist but it is unique if it does. The following proposition shows this.

Proposition 1. *For a set of assignments $S$ and a set of constraints of the form (iv) $K$, $Cn(S \cup K)$ is unique if it exists.*

Proof. For a set of assignments $X$, let

$$\Delta(X, K) = \{f = v \mid (f = v \textbf{ if } \varphi) \in K, X \models \varphi\}$$

and

$$T(X) = \begin{cases} X \cup \Delta(X, K) & \text{if } X \cup \Delta(X, K) \text{ is domain-consistent} \\ undefined & \text{otherwise} \end{cases}$$

Let $T^0(S) = S$ and, for $i \geq 0$, let $T^{i+1}(S) = T(T^i(S))$. It is easy to show that (by induction) if $Cn(S \cup K)$ exists then $T^i(S)$ is defined, $T^i(S) \subseteq T^{i+1}(S)$, and $T^i(S) \subseteq Cn(S \cup K)$ for all $i \geq 0$. Therefore $lim_{i \to \infty} T^i(S)$ exists. Let $S'$ denote such limit. Clearly, $S'$ is closed under $K$. From the minimality of $Cn(S \cup K)$, we have that $Cn(S \cup K) = S'$, which implies that $Cn(S \cup K)$ is unique. □

We now determine the possible next states after the execution of an action in a given state. An action $a$ is executable in a state $s$ if there exists a proposition (i) in $D$ such that $s \models \varphi$. For a non-sensing action $a$ and a state $s$, if $a$ is executable in $s$ then the direct effects of $a$ on $s$, $e(a, s)$, is defined by the following set of assignments

$$e(a, s) = \{f = v \mid D \text{ contains an effect proposition (ii) such that } s \models \varphi\}$$

However, in the presence of static causal laws, in addition to $e(a, s)$, there may be some other indirect effects (or *ramifications*). We follow [10] to define

the possible next states after the execution of $a$ in $s$ as follows

$$Res(a, s) = \{s' \mid s' \text{ is a state such that } s' = Cn((s \cap s') \cup e(a, s) \cup R)\}^3$$

where $R$ denotes the set of proposition of the form (iv) in $D$. When $Res(a, s)$ is empty, we say that the execution of $a$ in $s$ fails and results in an *undefined state* $\perp$. The transition function $\Phi$ between c-states is defined as follows.

DEFINITION 1 (Transition Function). *Let $D$ be a domain description. For an action $a$ and a c-state $\sigma = \langle s, \Sigma\rangle$,*

- *if $\sigma = \perp$ or $a$ is not executable in $s$ then $\Phi(a, \sigma) = \{\perp\}$;*
- *if $a$ is a non-sensing action and executable in $s$*
  - *$Res(a, s) = \emptyset$ then $\Phi(a, \sigma) = \{\perp\}$;*
  - *$Res(a, s) \neq \emptyset$ then $\Phi(a, \sigma) = \{\langle s_1, \Sigma'\rangle \mid s_1 \in Res(a, s)\}$ where $\Sigma' = \{s' \mid s' \in Res(a, s'')$ for some $s'' \in \Sigma$ s.t. $a$ is executable in $s''\}$;*
- *if $a$ is a sensing action and executable in $s$ with the proposition $a$ **partitions** $f$ **into** $\{V_j\}_{j\in S}$ in $D$ then $\Phi(a, \sigma) = \{\langle s, \Sigma'\rangle\}$, where $\Sigma' = \{s' \mid s' \in \Sigma$ s.t. $a$ is executable in $s'$ and $s' \overset{f,\{V_j\}_{j\in S}}{\sim} s\}$.*

It is easy to see that if $\langle s, \Sigma\rangle$ is a grounded c-state and $\perp \notin \Phi(a, \langle s, \Sigma\rangle)$ then $\Phi(a, \langle s, \Sigma\rangle)$ is a set of grounded c-states. In the next definition, we define initial states and initial c-states of an action theory.

DEFINITION 2 (Initial states and c-states). *For an action theory $(D, O)$,*

- *a state $s_0$ is called an* initial state *of $(D, O)$ if $s_0 \models l$ for every proposition* **initially** *$l$ in $O$;*
- *a c-state $\langle s_0, \Sigma_0\rangle$ is called an* initial c-state *of $(D, O)$ if $s_0$ is an initial state and $\Sigma_0$ is the set of all initial states of $(D, O)$.*

It is worth noticing that each initial c-state is grounded. To determine the c-states resulting from the execution of a plan in a c-state, we extend the transition function to the extended transition function $\hat{\Phi}$ as follows.

[3]Note that by the definition of assignments, a state can also be viewed as a set of assignments.

Definition 3 (Extended Transition Function). *Let $(D, O)$ be an action theory and $\Phi$ be the transition function of $D$. For a plan $p$ and a c-state $\sigma = \langle s, \Sigma \rangle$,*

- *if $\sigma = \bot$ then $\hat{\Phi}(p, \sigma) = \{\bot\}$; otherwise,*
- *if $p = []$ then $\hat{\Phi}(p, \sigma) = \{\sigma\}$;*
- *if $p = [a; p']$, where $a$ is an action and $p'$ is a plan, then*

$$\hat{\Phi}(p, \sigma) = \bigcup_{\sigma' \in \Phi(a,\sigma)} \hat{\Phi}(p', \sigma');$$

- *if $p = [$ **if** $\varphi$ **then** $p_1$ **else** $p_2]$, where $\varphi$ is a formula, $p_1$ and $p_2$ are plans, then*

$$\hat{\Phi}(p, \sigma) = \begin{cases} \hat{\Phi}(p_1, \sigma) & \text{if } \varphi \text{ is known to be true in } \sigma; \\ \hat{\Phi}(p_2, \sigma) & \text{if } \varphi \text{ is known to be false in } \sigma; \\ \{\bot\} & \text{if } \varphi \text{ is unknown in } \sigma. \end{cases}$$

We will now define the entailment relation $\models_{\mathcal{AM}_K}$.

Definition 4 (Entailment). *For an action theory $(D, O)$,*

- *$(D, O)$ entails a query* **knows** $\varphi$ **after** $p$, *denoted by*

$$(D, O) \models_{\mathcal{AM}_K} \textbf{knows } \varphi \textbf{ after } p,$$

*if for every initial c-state $\sigma_0 = \langle s_0, \Sigma_0 \rangle$, $\bot \notin \hat{\Phi}(p, \sigma_0)$ and $\varphi$ is known to be true in every c-state in $\hat{\Phi}(p, \sigma_0)$.*

- *$(D, O)$ entails a query* **kwhether** $\varphi$ **after** $p$, *denoted by*

$$(D, O) \models_{\mathcal{AM}_K} \textbf{kwhether } \varphi \textbf{ after } p,$$

*if for every initial c-state $\sigma_0 = \langle s_0, \Sigma_0 \rangle$, $\bot \notin \hat{\Phi}(p, \sigma_0)$ and $\varphi$ is known (to be true or false) in every c-state in $\hat{\Phi}(p, \sigma_0)$.*

We now illustrate the intuition of the above definition through the action theories $(D_u, O_u)$, $(D_s, O_s)$, $(D_b, O_b)$, and $(D_g, O_g)$. It is easy to see that the following proposition holds for the UNIX domain.

Proposition 2. *For the UNIX domain,*

- $(D_u, O_u) \models_{\mathcal{AM}_K}$ **knows** $(curdir = /\texttt{mydoc})$ **after** $[cd(/\texttt{mydoc})]$ *and,*
- $(D_u, O_u) \models_{\mathcal{AM}_K}$ **kwhether** $exists(f)$ **after** $[ls(f)]$.

The first item tells us that the current directory will change to `/mydoc` after the action $cd(/\texttt{mydoc})$ is executed (this directory exists since one of its sub-directories `/mydoc/paper` exists). The second item says that after executing the action $ls(f)$ we will know whether or not the file $f$ exists.

We will now show that the plan $p_s = [stain; inspect; blood\_sample; analyze\_blood]$ will help us to determine the illness of the patient.

PROPOSITION 3. *For the illness domain,*

$$(D_s, O_s) \models_{\mathcal{AM}_K} \textbf{kwhether } (i = i_j) \textbf{ after } p_s \;\; (j \in \{1, \ldots, 5\}).$$

PROOF. Let us denote a state of this domain by a tuple $\langle i, dead, tcd, bsd, color, hc\rangle$. The table on the next page shows the computation related to the proof of this proposition. The second to seventh column contains the possible values of a fluent. The column $\#s$ and $\#c{-}states$ contains the number of states and c-states, respectively. The first subtable represents initial states and c-states. The subsequent subtables denote states and c-states after the execution of $p_s$ up to the action whose name appears in the first column. For example, in the subtable with *stain* in the first column and five rows, the c-states resulting from the execution of $[stain]$ from one of the initial c-state are of the form $\langle s, \Sigma\rangle$ where $s$ is a state specified by the corresponding row and $\Sigma$ is the set of states specified by the rows which have the value of $i$ belonging to the set specified in the right side of the equation in the last column (ALL indicates all possible states).

It is easy to see that there are 30 initial states since $i$ can take the values $\{i_1, \ldots, i_5\}$, *color* can be either *red*, *blue*, or *white*, and *hc* can be either *true* or $false$ (the first row of the table). Executing *stain* will reduce the number of states and the number of c-states to 10 possible c-states. Executing *inspect* will not reduce the number of c-states but significantly reduce the size of the k-state. Indeed, the k-state of a c-state now contains either 4 or 2 states. Execution of *blood_sample* will further reduce the number of possible c-states to 5 and executing *analyze_blood* will result in 5 c-states, whose bottom part contains exactly one state which is the top part. That means that after the above sequence of action, one can know exactly the illness of the patient, and hence, find an appropriate cure for her.

| | i | dead | tcd | bsd | color | hc | # s | # c-states |
|---|---|---|---|---|---|---|---|---|
| initial | – | $false$ | $false$ | $false$ | – | – | 30 | 30 ($\Sigma$=ALL) |
| stain | $i_1$ | $false$ | $true$ | $false$ | $red$ | – | 2 | 2 ($\Sigma$=ALL) |
| | $i_2$ | $false$ | $true$ | $false$ | $red$ | – | 2 | 2 ($\Sigma$=ALL) |
| | $i_3$ | $false$ | $true$ | $false$ | $blue$ | – | 2 | 2 ($\Sigma$=ALL) |
| | $i_4$ | $false$ | $true$ | $false$ | $blue$ | – | 2 | 2 ($\Sigma$=ALL) |
| | $i_5$ | $false$ | $true$ | $false$ | $white$ | – | 2 | 2 ($\Sigma$=ALL) |
| inspect | $i_1$ | $false$ | $true$ | $false$ | $red$ | – | 2 | 2 ($\Sigma$=$\{i_1, i_2\}$) |
| | $i_2$ | $false$ | $true$ | $false$ | $red$ | – | 2 | 2 ($\Sigma$=$\{i_1, i_2\}$) |
| | $i_3$ | $false$ | $true$ | $false$ | $blue$ | – | 2 | 2 ($\Sigma$=$\{i_3, i_4\}$) |
| | $i_4$ | $false$ | $true$ | $false$ | $blue$ | – | 2 | 2 ($\Sigma$=$\{i_3, i_4\}$) |
| | $i_5$ | $false$ | $true$ | $false$ | $white$ | – | 2 | 2 ($\Sigma$=$\{i_5\}$) |
| blood- | $i_1$ | $false$ | $true$ | $true$ | $red$ | $true$ | 1 | 1 ($\Sigma$=$\{i_1, i_2\}$) |
| _sample | $i_2$ | $false$ | $true$ | $true$ | $red$ | $false$ | 1 | 1 ($\Sigma$=$\{i_1, i_2\}$) |
| | $i_3$ | $false$ | $true$ | $true$ | $blue$ | $true$ | 1 | 1 ($\Sigma$=$\{i_3, i_4\}$) |
| | $i_4$ | $false$ | $true$ | $true$ | $blue$ | $false$ | 1 | 1 ($\Sigma$=$\{i_3, i_4\}$) |
| | $i_5$ | $false$ | $true$ | $true$ | $white$ | $true$ | 1 | 1 ($\Sigma$=$\{i_5\}$) |
| analyze- | $i_1$ | $false$ | $true$ | $true$ | $red$ | $true$ | 1 | 1 ($\Sigma$=$\{i_1\}$) |
| _blood | $i_2$ | $false$ | $true$ | $true$ | $red$ | $false$ | 1 | 1 ($\Sigma$=$\{i_2\}$) |
| | $i_3$ | $false$ | $true$ | $true$ | $blue$ | $true$ | 1 | 1 ($\Sigma$=$\{i_3\}$) |
| | $i_4$ | $false$ | $true$ | $true$ | $blue$ | $false$ | 1 | 1 ($\Sigma$=$\{i_4\}$) |
| | $i_5$ | $false$ | $true$ | $true$ | $white$ | $true$ | 1 | 1 ($\Sigma$=$\{i_5\}$) |

□

Let $p_6 = []$ and $p_j = [\ \textbf{if}\ i = i_j\ \textbf{then}\ \ medicate(c_j)\ \ \textbf{else}\ \ p_{j+1}]$ for $j = 1, \ldots, 5$. The below proposition follows directly from Proposition 3.

Proposition 4. *For the illness domain,*

$$(D_s, O_s) \models_{\mathcal{AM}_K} \textbf{knows}\ (i = none)\ \textbf{after}\ [p_s; p_1].$$

In the next proposition we show that executing the action *look* will tell us whether or not the tank is empty in $(D_g, O_g)$.

Proposition 5. *For the gas domain and a number $v \in [0, 20]$,*

$$(D_g, O_g) \models_{\mathcal{AM}_K} \textbf{kwhether}\ (gas\_in\_tank = v)\ \textbf{after}\ [look].$$

Proof. Observe that any state in this domain consists of a single atom $gas\_in\_tank = v$ for some $v \in [0, 20]$. Let $s_v = \{gas\_in\_tank = v\}$ and $\Sigma_0 = \{s_v \mid v \in [0, 20]\}$. Clearly, for every $v \in [0, 20]$, $\sigma_v = \langle s_v, \Sigma_0 \rangle$ is an initial c-state. On the other hand, since $\Phi(look, \sigma_v) = \{\langle s_v, \{s_v\}\rangle\}$, we can conclude that $(D_g, O_g) \models \textbf{kwhether}\ (gas\_in\_tank = v)\ \textbf{after}\ look$. □

The next proposition shows that it is not always the case that the value of a fluent is known after the execution of an action which senses that fluent.

PROPOSITION 6. *For the block world domain,*

$$(D_b, O_b) \not\models_{\mathcal{AM}_K} \textbf{kwhether } (loc(b) = on(a)) \textbf{ after } [sense(b)].$$

PROOF. It is easy to see that there are only five states in this domain
$s_1 = \{loc(a) = onTable, loc(b) = onTable\}$,
$s_2 = \{loc(a) = onTable, loc(b) = inHand\}$,
$s_3 = \{loc(a) = onTable, loc(b) = on(a)\}$,
$s_4 = \{loc(a) = inHand, loc(b) = onTable\}$, and
$s_5 = \{loc(a) = on(b), loc(b) = onTable\}$
and the set of possible initial states is $\Sigma_0 = \{s_1, s_2, s_3\}$. This gives us three initial c-states $\langle s_i, \Sigma_0\rangle$ $(i = 1, 2, 3)$. Executing $sense(b)$ results in three c-states $\langle s_1, \{s_1, s_3\}\rangle$, $\langle s_2, \{s_2\}\rangle$, and $\langle s_3, \{s_1, s_3\}\rangle$. Because $s_1 \models \neg(loc(b) = on(a))$ and $s_3 \models (loc(b) = on(a))$ we have that $loc(b) = on(a)$ is unknown in $\langle s_1, \{s_1, s_3\}\rangle$. According to the definition of entailment, it follows that

$$(D_b, O_b) \not\models_{\mathcal{AM}_K} \textbf{kwhether } (loc(b) = on(a)) \textbf{ after } [sense(b)].$$

□

## 3. Relationship Between $\mathcal{AM}_K$ Domains and Boolean Domains

As we have informally discussed in the beginning, the introduction of discrete and finite multi-valued fluents in $\mathcal{AM}_K$ does not increase its expressiveness in comparison to its predecessor $\mathcal{A}_K$ (by $\mathcal{A}_K$ we mean the language $\mathcal{A}_K$ in [2]) extended with state constraints. We will now show that each $\mathcal{AM}_K$ action theory, $(D, O)$, with discrete, finite multi-valued fluents whose sensing actions are of the simple form $a$ **determines** $f$ can be translated into a semantically equivalent $\mathcal{A}_K$ action theory $(D^b, O^b)$. In what follows, we attach a superscript $b$ to elements of the theory $(D^b, O^b)$ to distinguish them with their counterparts in $(D, O)$. The translation is as follows.

- $D^b$ contains the set of Boolean fluents $\{f_v \mid f \in \mathbf{F} \text{ and } v \in dom(f)\}$.
- $D^b$ has the same set of actions as $D$ does.
- A literal $l \equiv f = v$ (resp. $f \neq v$) in $D$ is translated into a corresponding literal $l^b \equiv f_v$ (resp. $\neg f_v$) in $D^b$.
- A formula $\varphi$ in $D$ is translated into a corresponding formula $\varphi^b$ in $D^b$ by replacing each literal $l$ that occurs in $\varphi$ with $l^b$. For instance, $(f = 1) \wedge (g \neq 2)$ becomes $f_1 \wedge \neg g_2$.

- For every proposition of the form (i), $D^b$ contains the proposition

$$\textbf{executable } a \textbf{ if } \varphi^b. \tag{viii}$$

- For every proposition of the form (ii), $D^b$ contains

$$a \textbf{ causes } f_v \textbf{ if } \varphi^b \tag{ix}$$

and the set of propositions

$$\{a \textbf{ causes } \neg f_{v'} \textbf{ if } \varphi^b \mid v' \in dom(f) \setminus \{v\}\}. \tag{x}$$

- For every sensing action $a$ with $a$ **determines** $f \in D$, $D^b$ contains the following propositions

$$\{a \textbf{ determines } f_v \mid v \in dom(f)\}. \tag{xi}$$

- For every proposition of the form (iv), $D^b$ contains

$$f_v \textbf{ if } \varphi^b \tag{xii}$$

and the set of propositions

$$\{\neg f_{v'} \textbf{ if } \varphi^b \mid v' \in dom(f) \setminus \{v\}\}. \tag{xiii}$$

- For every fluent $f \in \mathbf{F}$, $D^b$ contains the following sets of constraints

$$\{f_v \textbf{ if } \bigwedge_{v' \in dom(f)\setminus\{v\}} \neg f_{v'} \mid v \in dom(f)\} \tag{xiv}$$

$$\{\neg f_v \textbf{ if } f_{v'} \mid f \in \mathbf{F}, v, v' \in dom(f), v \neq v'\}. \tag{xv}$$

- The set of observations in $D^b$ is

$$O^b = \{ \textbf{initially } l^b \mid \textbf{ initially } l \in O\}. \tag{xvi}$$

- Nothing else in $D^b$ and $O^b$.

Similarly, for a plan $p$ in $\mathcal{AM}_K$, we construct a corresponding plan $p^b$ in $\mathcal{A}_K$ by replacing each formula $\varphi$ occurring in $p$ with $\varphi^b$ and leaving actions unchanged.

We will now discuss some properties of the translation from $(D, O)$ into $(D^b, O^b)$. First, note that the size of an action theory $(D, O)$ depends on

- the number of actions, i.e., the size of $\mathbf{A}$;

- the number of fluents, i.e., the size of $\mathbf{F}$;
- the size of fluent domains, i.e., $\Sigma_{f \in \mathbf{F}} |dom(f)|$;
- the number of propositions in $D$; and
- the length of formulae, i.e., the number of literals, occurring in the propositions in $D$.

Assume that we only need a constant amount of bytes to encode fluents and actions then we can define the size of a proposition in $(D, O)$ by the length of the formula occurring in it, e.g., the size of $a$ **causes** $f$ **if** $\varphi$ is the length of $\varphi$. Then the size of an action theory $(D, O)$ is defined as the sum of

1. the sum over the size of all propositions in $(D, O)$;
2. the number of actions;
3. the number of fluents;
4. the sum over $|dom(f)|$ for $f \in \mathbf{F}$.

Under the assumption that $(D, O)$ is a discrete and finite multi-valued fluent action theory, we have the following observations.

- The translation of each fluent $f$ in $(D, O)$ into the set of fluents and constraints of the forms (xiv) and (xv) in $(D^b, O^b)$ is bounded by a polynomial function of the size of $dom(f)$;
- The translation of a formula $\varphi$ into $\varphi^b$ is bounded by a polynomial function of the length of $\varphi$;
- The translation of a proposition of the forms (i)– (iv) in $(D, O)$ into propositions of the forms (viii) – (xiii) in $(D^b, O^b)$ is bounded by a polynomial function of the length of $\varphi$;
- The translation of a proposition of the form (iii) in $(D, O)$ into propositions of the form (xi) in $(D^b, O^b)$ is bounded by a polynomial function of $|dom(f)|$; and
- The translation of an observation of the form (v) in $(D, O)$ into propositions of the form (xvi) in $(D^b, O^b)$ can be done in constant time.

This implies that the following proposition holds.

Proposition 7. *For a discrete and finite multi-valued fluents action theory* $(D, O)$*,*

- *the translation from* $(D, O)$ *into* $(D^b, O^b)$ *can be performed in polynomial time in the size of* $(D, O)$*; and*
- *the size of* $(D^b, O^b)$ *is polynomially bounded in the size of* $(D, O)$*.*

It is easy to see that a formula $\varphi^b$ in $(D^b, O^b)$ can be translated into a unique formula $\varphi$ in $(D, O)$ whose binary version is exactly $\varphi^b$. We can prove the following

Proposition 8. *For a discrete and finite multi-valued fluents action theory* $(D, O)$*, a plan* $p$*, and formula* $\varphi$

$$(D, O) \models_{\mathcal{AM}_K} \textbf{knows } \varphi \textbf{ after } p \quad \textit{iff} \quad (D^b, O^b) \models_{\mathcal{A}_K} \textbf{knows } \varphi^b \textbf{ after } p^b$$

*and,*

$$(D, O) \models_{\mathcal{AM}_K} \textbf{kwhether } \varphi \textbf{ after } p \quad \textit{iff} \quad (D^b, O^b) \models_{\mathcal{A}_K} \textbf{kwhether } \varphi^b \textbf{ after } p^b.$$

Proof. See Section 6. □

The above proposition has two interesting consequences. The 'negative' but expected one is the fact that introducing multi-valued fluents does not increase the expressiveness of the language $\mathcal{A}_K$ as long as domains of fluents are discrete and finite. Nevertheless, the examples show that $\mathcal{AM}_K$ is a much better knowledge representation language for representing and reasoning with sensing actions in term of compactness of the representation. The 'positive' one, however, is about the complexity of planning with sensing actions and incomplete information. It shows that for $\mathcal{AM}_K$ action theories with a deterministic transition function[4] and discrete, finite domains, the computational complexity results in [1] will hold, i.e., planning in $\mathcal{AM}_K$ is not harder than that in $\mathcal{A}_K$. We list one of them below.

Corollary 1. *The planning problem with* $\mathcal{AM}_K$ *finite, discrete, and deterministic domains (incomplete information about the initial state and sensing actions) is* **PSPACE***-complete.*

[4] $(D, O)$ is deterministic if for every c-state $\sigma$ and non-sensing action $a$, $|\Phi(a, \sigma)| \leq 1$.

## 4. Discussion and Future Work

In this paper, we concentrate on the development of a high-level action description language with sensing actions and multi-valued fluents. The main contribution of this paper is the language $\mathcal{AM}_K$ with a transition function based semantics. We illustrate the use of $\mathcal{AM}_K$ in some examples taken from the literature and show that adding multi-valued fluents does not result in an increase in both expressiveness and complexity of planning with sensing actions in discrete, finite domains. This opens the applicability of results in approximating $\models_{\mathcal{A}_K}$ (e.g. in [13]) for multi-valued fluents to deal with the problem of huge search space. We leave it as one of the topics of our future research. We will also investigate the allowance of negative literals of the form $f \neq v$ in propositions of the form (ii) or (iv) in the place of $f = v$. At present, we are using $\mathcal{AM}_K$ in developing a planner for reasoning with sensing actions in the presence of incomplete information.

## 5. Acknowledgment

The authors wish to thank Chitta Baral for his comments and suggestions on an earlier version of this paper. Tran Cao Son and Phan Huy Tu are partially supported by the grant NSF 0220590.

## 6. Appendix - Proof of Proposition 8

First, let us briefly discuss the essential features of $\mathcal{A}_K$. The main difference between $\mathcal{A}_K$ and $\mathcal{AM}_K$ is that fluents in $\mathcal{A}_K$ are Boolean fluents, whereas those in $\mathcal{AM}_K$ are not. Domains in $\mathcal{A}_K$ consist of propositions similar to (i)–(iv) except that only Boolean literals are allowed in formulae $\varphi$ and in the place of $f = v$. Because fluents are Boolean, propositions of the form (iii) in $\mathcal{A}_K$ are of the simple form $a$ **determines** $f$ only, where $f$ is a fluent. The notions of states, c-states, and agreements on fluents between states in $\mathcal{A}_K$ are defined similarly as in $\mathcal{AM}_K$. A set of assignments in $\mathcal{AM}_K$ corresponds to a set of literals in $\mathcal{A}_K$. The notion of satisfiability of a formula with respect to a set of literals or a state is also defined accordingly. The semantics of $\mathcal{A}_K$ theories is given by an entailment relation whose definition is based on the transition function that maps from pairs of actions and c-states into sets of c-states. For completeness of the paper, we include the definition of the transition function of $\mathcal{A}_K$ herewith.

Definition 5 (Transition Function of $\mathcal{A}_K$). *Let $D$ be a binary domain description. For an action $a$ and a c-state $\sigma = \langle s, \Sigma \rangle$,*

- *if $\sigma = \bot$ or $a$ is not executable in $s$ then $\Phi(a, \sigma) = \{\bot\}$;*
- *if $a$ is a non-sensing action and executable in $s$ and*
  - *$Res(a, s) = \emptyset$ then $\Phi(a, \sigma) = \{\bot\}$;*
  - *$Res(a, s) \neq \emptyset$ then $\Phi(a, \sigma) = \{\langle s_1, \Sigma' \rangle \mid s_1 \in Res(a, s)\}$ where $\Sigma' = \{s' \mid s' \in Res(a, s'')$ for some $s'' \in \Sigma$ s.t. $a$ is executable in $s''\}$;*
- *if $a$ is a sensing action and executable in $s$ with the proposition $a$ **determines** $f_1, \ldots, a$ **determines** $f_n$ in $D$ then $\Phi(a, \sigma) = \{\langle s, \Sigma' \rangle\}$ where $\Sigma' = \{s' \mid s' \in \Sigma$ s.t. $a$ is executable in $s'$ and $s'(f_i) = s(f_i)$ for every $1 \leq i \leq n\}$,*

*where $Res(a, s)$ and $e(a, s)$ are defined similarly as in $\mathcal{AM}_K$.*

The extended transition function $\hat{\Phi}$ and the entailment relation for binary domains are defined as in Definitions 3 and 4.

To prove the equivalence between the entailment relationships of $\mathcal{A}_K$ and $\mathcal{AM}_K$ let us introduce some more notations. Recall that for a theory $(D, O)$ in $\mathcal{AM}_K$, $(D^b, O^b)$ by convention denotes the corresponding theory in $\mathcal{A}_K$ obtained from $(D, O)$ by the translation in Section 3.

Observe that a set of literals in $D^b$ can be viewed as a set of assignments. For this reason, we will often refer to a set of fluent literals in $D^b$ as a set of assignments. The value of a fluent (resp. a formula) with respect to a set of assignments in $\mathcal{A}_K$ is defined in the same way as before. We say that a set of assignments $\theta^b$ in $D^b$ is *domain-consistent* if for every fluent $f$ then either $\theta^b(f_v)$ is undefined for all $v \in dom(f)$ or there exists $v \in dom(f)$ such that $\theta^b(f_v) = true$ and $\theta^b(f_{v'}) = false$ for all $v' \in dom(f) \setminus \{v\}$. Clearly, all states in $D^b$ are domain-consistent since they satisfy the static causal laws of the form (xv). As with sets of assignments in $D$, whenever we say $\theta$ is a set of assignments in $D^b$, we mean that it is domain-consistent.

We first define the equivalence relationship between fluent assignments in $D$ and $D^b$ and then extend this notion for states, sets of states and c-states. We then prove several lemmas that are needed for proving Proposition 8.

For a set of assignments $\theta$ in $D$, let

$$\begin{aligned} \theta^b = \quad & \{f_v \mid \theta(f) = v, v \in dom(f)\} \cup \\ & \{\neg f_{v'} \mid \theta(f) = v, v \in dom(f), v' \in dom(f) \setminus \{v\}\} \end{aligned}$$

and for a set of assignments $\eta^b$ in $D^b$, let

$$\eta = \{f = v \mid v \in dom(f), \eta^b(f_v) = true\}$$

We say that $\theta^b$ (resp. $\eta$) is the binary (resp. multi-valued) version of $\theta$ (resp. $\eta^b$). It is easy to see that the following lemma hold.

LEMMA 1. *(1) Let $\theta$, $l$, and $\varphi$ (resp. $\theta^b$, $l^b$, and $\varphi^b$) be a set of assignments, a literal, and a formula in $D$ (resp. in $D^b$), respectively. Then (i) $\theta \models l$ iff $\theta^b \models l^b$ and (ii) $\theta \models \varphi$ iff $\theta^b \models \varphi^b$.*

*(2) Let $\theta_1$ and $\theta_2$ (resp. $\theta_1^b$ and $\theta_2^b$) be two sets of assignments in $D$ (resp. in $D^b$) such that $\theta_1 \cup \theta_2$ (resp. $\theta_1^b \cup \theta_2^b$) is domain-consistent then $\theta_1^b \cup \theta_2^b$ (resp. $\theta_1 \cup \theta_2$) is domain-consistent and (i) $(\theta_1 \cup \theta_2)^b = \theta_1^b \cup \theta_2^b$; and (ii) $(\theta_1 \cap \theta_2)^b = \theta_1^b \cap \theta_2^b$.*

For a set of states $S$ and for a c-state $\sigma = \langle s, \Sigma \rangle$ in $D$, let $S^b = \{s^b \mid s \in S\}$ and $\sigma^b = \langle s^b, \Sigma^b \rangle$. Likewise, for a set of states $S^b$ and a c-state $\sigma^b = \langle s^b, \Sigma^b \rangle$, in $D^b$, let $S = \{s \mid s^b \in S^b\}$ and $\sigma = \langle s, \Sigma \rangle$. Since a state is essentially a set of assignments, the following lemma follows directly from the previous lemma.

LEMMA 2. *Let $\sigma$ and $\varphi$ (resp. $\sigma^b$ and $\varphi^b$) be a c-state and a formula in $D$ (resp. $D^b$), respectively. Then $\varphi$ holds in $\sigma$ iff $\varphi^b$ holds in $\sigma^b$.*

The following lemma shows the equivalence of executability condition of actions in $D$ and $D^b$.

LEMMA 3. *For a state $s$ in $D$ (resp. $s^b$ in $D^b$) and an action $a$, $a$ is executable in $s$ iff $a$ is executable in $s^b$.*

PROOF. Assume that $a$ is executable in $s$. That means there exists a proposition of form (i) such that $s \models \varphi$. According to the translation, there is a corresponding proposition of the form (viii) in $D^b$. Since $\varphi$ holds in $s$, we have $\varphi^b$ also holds in $s^b$ (Lemma 2). Thus $a$ is also executable in $s^b$.

On the other hand, if $a$ is executable in $s^b$ then there exists a proposition of the form (viii) in $D^b$ such that $s^b \models \varphi^b$. This implies that there exists a proposition of the form (i) in $D$. From Lemma 2, we have $s \models \varphi$. Hence, $a$ is also executable in $s$. □

LEMMA 4. *Let $s$ be a state in $D$ (resp. $s^b$ be a state in $D^b$) and $a$ be a non-sensing action executable in $s$. Then $(Res_{\mathcal{AM}_K}(a, s))^b = Res_{\mathcal{A}_K}(a, s^b)$*[5].

[5]From now on, we use subscripts $\mathcal{AM}_K$ and $\mathcal{A}_K$ to distinguish between the result functions, the transition functions and the extended transition functions in $D$ and $D^b$.

Proof. The lemma is trivial if $Res_{\mathcal{AM}_K}(a,s)$ or $Res_{\mathcal{A}_K}(a,s^b)$ is empty. We will prove this lemma by showing that

i. if $u$ is a state in $Res_{\mathcal{AM}_K}(a,s)$ then $u^b$ is in $Res_{\mathcal{A}_K}(a,s^b)$; and

ii. if $u^b$ is a state in $Res_{\mathcal{A}_K}(a,s^b)$ then $u$ is in $Res_{\mathcal{AM}_K}(a,s)$.

Let us denote the sets of constraints of the forms (xii), (xiii), (xiv), and (xv) by $R_1$, $R_2$, $R_3$ and $R_4$ respectively. Clearly the set of constraints in $D^b$ is $R^b = R_1 \cup R_2 \cup R_3 \cup R_4$. It is easy to see that the following observation holds: *if* $\theta_s = \{f = v \mid f = v \textbf{ if } \varphi \in R \text{ and } s \models \varphi\}$ *then*

$$\begin{array}{rcl} \theta_s^b & = & \{f_v \mid (f_v \textbf{ if } \varphi^b) \in R_1 \cup R_3 \text{ and } s^b \models \varphi^b\} \cup \\ & & \{\neg f_v \mid (\neg f_v \textbf{ if } \varphi^b) \in R_2 \cup R_4 \text{ and } s^b \models \varphi^b\} \end{array}$$

*and vice versa*. Thus $(s \cup \theta_s)^b = s^b \cup \theta_s^b$ (Item (2), Lemma 1) and both are domain-inconsistent.

**Proof of (i)**. Let $u$ be a state in $Res_{\mathcal{AM}_K}(a,s)$. We will show that $u^b \in Res_{\mathcal{A}_K}(a,s)$. By the definition of the *Res* function, we have

$$u = Cn((s \cap u) \cup e_{\mathcal{AM}_K}(a,s) \cup R)$$

where $R$ denotes the set of propositions of the form (iv) in $D$. That is, there exists a monotonic sequence of domain-consistent sets of assignments $s_1, s_2, \ldots, s_n$ $(n \geq 1)$ such that $s_1 \subset s_2 \subset \cdots \subset s_{n-1} = s_n = u$ where $s_1 = (s \cap u) \cup e_{\mathcal{AM}_K}(a,s)$ and

$$s_{i+1} = s_i \cup \{f = v \mid (f = v \textbf{ if } \varphi) \in R \text{ s.t. } s_i \models \varphi\} \ \ (1 \leq i \leq n-1).$$

By Lemmas 1 and 2, we have that $(e_{\mathcal{AM}_K}(a,s))^b = e_{\mathcal{A}_K}(a,s^b)$. By Item (2), Lemma 1, and from the above observation we can show that

$$s_1^b = (s^b \cap u^b) \cup e_{\mathcal{A}_K}(a,s^b) \text{ and } s_{i+1}^b = s_i^b \cup \theta_{s_i}^b.$$

This implies that $u^b = s_n^b$ and $s_1^b, \ldots, s_n^b$ is a monotonic sequence. By the definition of the closure and by Proposition 1, $s_n^b$ is $Cn(u^b \cup R^b)$. As a result, we have $u^b \in Res_{\mathcal{A}_K}(a,s^b)$.

**Proof of (ii)**. Similar to (i). □

Lemma 5. *For a c-state* $\sigma$ *in* $D$ *(resp.* $\sigma^b$ *in* $D^b$*) and an action* $a$*,*

$$(\Phi_{\mathcal{AM}_K}(a,\sigma))^b = \Phi_{\mathcal{A}_K}(a,\sigma^b).$$

PROOF. Let $\sigma = \langle s, \Sigma \rangle$ be a c-state in $D$. Consider the following cases

**Case 1.** $\sigma = \bot$. So we have $\sigma^b = \bot$. Thus, according to the definition of the transition functions, we have $\Phi_{\mathcal{AM}_K}(a, \sigma) = \Phi_{\mathcal{A}_K}(a, \sigma^b) = \{\bot\}$. and thus, the lemma holds for this case.

**Case 2.** $\sigma \neq \bot$, $a$ is not executable in $s$. By Lemma 3, $a$ is also not executable in $s^b$. Again by the definitions of $\Phi_{\mathcal{A}_K}$ and $\Phi_{\mathcal{AM}_K}$, both $\Phi_{\mathcal{AM}_K}(a, \sigma)$ and $\Phi_{\mathcal{A}_K}(a, \sigma^b)$ are $\{\bot\}$ and hence $(\Phi_{\mathcal{AM}_K}(a, \sigma))^b = \Phi_{\mathcal{A}_K}(a, \sigma^b)$.

**Case 3.** $\sigma \neq \bot$, $a$ is a non-sensing action executable in $\sigma$.

If $Res_{\mathcal{AM}_K}(a, s) = \emptyset$, by Lemma 4, we have that $Res_{\mathcal{A}_K}(a, s^b) = \emptyset$ and hence, $(\Phi_{\mathcal{AM}_K}(a, \sigma))^b = \Phi_{\mathcal{A}_K}(a, \sigma^b) = \{\bot\}$.

Assume that $Res_{\mathcal{AM}_K}(a, s) \neq \emptyset$ and $\sigma_1$ is a c-state in $\Phi_{\mathcal{AM}_K}(a, \sigma)$. According to the definition of $\Phi_{\mathcal{AM}_K}$ then $\sigma_1 = \langle s_1, \Sigma' \rangle$ for some $s_1 \in Res_{\mathcal{AM}_K}(a, s)$ and,

$$\Sigma' = \{s' \mid s' \in Res_{\mathcal{AM}_K}(a, s''), s'' \in \Sigma, a \text{ is executable in } s''\}.$$

By Lemma 4, we have $s_1^b \in Res_{\mathcal{A}_K}(a, s^b)$. Thus, $\langle s_1^b, \Sigma'' \rangle \in \Phi_{\mathcal{A}_K}(a, \sigma^b)$ where $\Sigma'' = \{s' \mid s' \in Res_{\mathcal{A}_K}(a, s''), s'' \in \Sigma^b, a \text{ is executable in } s''\}$.

Furthermore, because $(Res_{\mathcal{AM}_K}(a, s))^b = Res_{\mathcal{A}_K}(a, s^b)$ and by Lemma 3, we have that $(\Sigma')^b = \Sigma''$, i.e., $\sigma_1^b = \langle s_1^b, \Sigma'^b \rangle \in \Phi_{\mathcal{A}_K}(a, \sigma^b)$.

Similarly, we can show that for every c-state $\sigma_1^b$ in $\Phi_{\mathcal{A}_K}(a, \sigma^b)$, $\sigma_1$ belongs to $\Phi_{\mathcal{AM}_K}(a, \sigma)$. As a result, $(\Phi_{\mathcal{AM}_K}(a, \sigma))^b = \Phi_{\mathcal{A}_K}(a, \sigma^b)$.

**Case 4.** $\sigma \neq \bot$, $a$ is a sensing action with $a$ **determines** $f$ in $D$ and $a$ is executable in $\sigma$. According to the translation, $D^b$ contains the corresponding propositions $\{a \textbf{ determines } f_v \mid v \in dom(f)\}$. It is easy to see that by the definition of agreement, $s \overset{f,\{V_j\}_{j\in S}}{\sim} s'$ when each $V_j$, $j \in S$, has exactly one element, if and only if $s(v_j) = s'(v_j)$ for all $j \in S$. Let

$$\Sigma' = \{s' \mid s' \in \Sigma \text{ s.t. } a \text{ is executable in } s' \text{ and } s'(f) = s(f)\}.$$

Then, we have that

$$\begin{aligned} \Sigma'^b \quad = \quad & \{s'^b \mid s'^b \in \Sigma^b \text{ s.t. } a \text{ is executable in } s'^b \text{ and} \\ & s'^b(f_v) = s^b(f_v)\ \forall v \in dom(f)\}. \end{aligned}$$

By the definition of the transition functions, we have $\Phi_{\mathcal{AM}_K}(a, \sigma) = \{\langle s, \Sigma' \rangle\}$ and $\Phi_{\mathcal{A}_K}(a, \sigma^b) = \{\langle s^b, \Sigma'^b \rangle\}$. Hence, $(\Phi_{\mathcal{AM}_K}(a, \sigma))^b = \Phi_{\mathcal{A}_K}(a, \sigma^b)$.

Similarly, we can show that if $\sigma^b = \langle s^b, \Sigma^b \rangle$ is a c-state in $D^b$ then $(\Phi_{\mathcal{AM}_K}(a, \sigma))^b = \Phi_{\mathcal{A}_K}(a, \sigma^b)$. □

The next lemma generalizes the previous lemma to arbitrary plans.

Lemma 6. *Let $\sigma$ be a state in $D$ (resp. $\sigma^b$ in $D^b$). For every plan $p$ in $\mathcal{AM}_K$, $(\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma))^b = \hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b)$.*

Proof. Consider the following cases

**Case 1.** $\sigma = \bot$. So, $\sigma^b$ is also undefined, $\bot$. Therefore,

$$(\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma))^b = \hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b) = \{\bot\}.$$

**Case 2.** $\sigma \neq \bot$. We will prove the lemma by induction on the length of the plan $p$ but first of all let us define the length of a plan (for both $\mathcal{AM}_K$ and $\mathcal{A}_K$), denoted by *len*, as follows

i. $len([]) = 0$.

ii. $len([a;p']) = len(p') + 1$.

iii. $len([\ \textbf{if}\ \varphi\ \textbf{then}\ \ p_1\ \textbf{else}\ \ p_2]) = max\{len(p_1), len(p_2)\} + 1$.

*Base case:* $len(p) = 0$, that is, $p = []$ and $p^b = []$. By the definition of the transition function, we have $\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma) = \{\sigma\}$ and $\hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b) = \{\sigma^b\}$. The conclusion of the lemma for this case follows from Lemma 5.

*Hypothesis:* Suppose that the lemma is true for all plans p whose length is less than some integer $k \geq 1$.

*Inductive Step:* Let $p$ be a plan of length $k$. There are two possibilities

a) $p = [a;p']$ where $a$ is an action and $p'$ is a plan of the length less $k$. The corresponding plan in $\mathcal{A}_K$ is $p^b = [a;p'^b]$. By Lemma 5 and by the inductive hypothesis, we easily show that $(\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma))^b = \hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b)$.

b) $p = [\ \textbf{if}\ \varphi\ \textbf{then}\ \ p_1\ \textbf{else}\ \ p_2]$, where $p_1$ and $p_2$ are sub-plans whose lengths are less than $k$. We have that $p^b = [\ \textbf{if}\ \varphi^b\ \textbf{then}\ \ p_1^b\ \textbf{else}\ \ p_2^b]$.

• If $\varphi$ is known to be true in $\sigma$ then $\varphi^b$ is also known to be true in $\sigma^b$ (Lemma 2). Hence, $\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma) = \hat{\Phi}_{\mathcal{AM}_K}(p_1,\sigma)$ and $\hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b) = \hat{\Phi}_{\mathcal{A}_K}(p_1^b,\sigma^b)$. In addition, by the hypothesis, we have $(\hat{\Phi}_{\mathcal{AM}_K}(p_1,\sigma))^b = \hat{\Phi}_{\mathcal{A}_K}(p_1^b,\sigma^b)$. Hence, we can conclude the inductive step for this case.

• If $\varphi$ is known to be false in $\sigma$, the proof is similar.

• If $\varphi$ is unknown in $\sigma$ then both $\hat{\Phi}_{\mathcal{AM}_K}(p,\sigma)$ and $\hat{\Phi}_{\mathcal{A}_K}(p^b,\sigma^b)$ are undefined, i.e., the inductive step still holds for this case as well. □

The final lemma is about the equivalence between initial states and between initial c-states.

Lemma 7. *(i) If $s_0$ is an initial state of $(D,O)$ then $s_0^b$ is an initial state of $(D^b,O^b)$ and vice versa.*

*(ii) If $\sigma_0$ be an initial c-state of $(D, O)$ then $\sigma_0^b$ is an initial c-state of $(D^b, O^b)$ and vice versa.*

PROOF. **Proof of (i).** Assume that $s_0$ is an initial state of $(D, O)$. We need to prove that $s_0^b$ is an initial state of $(D^b, O^b)$.

Since $s_0$ is an initial state of $D$, $s_0 \models l$ for all literal $l$ such that **initially** $l \in O$. Hence $s_0^b \models l^b$ for all $l^b$ such that **initially** $l^b \in O^b$. In addition, $s_0$ satisfies all static causal laws in $R$ and thus $s_0^b$ satisfies all static causal laws in $R^b$. As a result, $s_0^b$ is an initial state of $D^b$.

Likewise, we can prove that if $s_0^b$ is an initial state of $(D^b, O^b)$ then $s_0$ is an initial state of $(D, O)$.

**Proof of (ii).** Immediately follows from (i). □

From Lemmas 6 and 7 we can infer Proposition 8.

## References

[1] BARAL, C., KREINOVICH, V., AND TREJO, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence 122*, Elsevier, 241–267.

[2] BARAL, C., MCILRAITH, S., AND SON, T. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proceedings of the Seventh International Conference on Principles of Knowledge and Representation and Reasoning (KR'2000)*. Morgan Kaufmann, 311–322.

[3] EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2000. Planning under incomplete information. In *Proceedings of the First International Conference on Computational Logic (CL'00)*. Springer Verlag, LNAI 1861, 807–821.

[4] GELFOND, M. AND LIFSCHITZ, V. 1993. Representing actions and change by logic programs. *Journal of Logic Programming 17,* 2,3,4, Elsevier, 301–323.

[5] GELFOND, M. AND LIFSCHITZ, V. 1998. Action languages. *ETAI 3,* 6.

[6] GIUNCHIGLIA, E., KARTHA, G., AND LIFSCHITZ, V. 1997. Representing action: indeterminacy and ramifications. *Artificial Intelligence 95*, Elsevier, 409–443.

[7] GOLDEN, K. AND WELD, D. 1996. Representing sensing actions: the middle ground revisited. In *Proceedings of the Fifth International Conference on Principles of Knowledge and Representation and Reasoning (KR 1996)*. Morgan Kaufmann, 174–185.

[8] LEVESQUE, H. 1996. What is planning in the presence of sensing? In *Proceedings of the Thirdteenth Conference on Artificial Intelligence*. AAAI Press, 1139–1146.

[9] LOBO, J., TAYLOR, S., AND MENDEZ, G. 1997. Adding knowledge to the action description language $\mathcal{A}$. In *Proceedings of the Fourteenth Conference on Artificial Intelligence*. AAAI Press, 454–459.

[10] McCain, N. and Turner, H. 1995. A causal theory of ramifications and qualifications. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1978–1984.

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

[12] Scherl, R. and Levesque, H. 1993. The frame problem and knowledge producing actions. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press, 689–695.

[13] Son, T. and Baral, C. 2001. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence 125,* 1-2 (January), Elsevier, 19–91.

[14] Thielscher, M. 2000. Representating the knowledge of a robot. In *Proceedings of the Seventh International Conference on Principles of Knowledge and Representation and Reasoning (KR'2000)*. Morgan Kaufmann, 109–120.

[15] Weld, D., Anderson, C., and Smith, D. 1998. Extending graphplan to handle uncertainity and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 897–904.

Tran Cao Son
Computer Science Department
New Mexico State University
MSC CS, PO Box 30001
Las Cruces, NM 88003
USA
tson@cs.nmsu.edu

Phan Huy Tu
Computer Science Department
New Mexico State University
MSC CS, PO Box 30001
Las Cruces, NM 88003
USA
tphan@cs.nmsu.edu

Xin Zhang
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287
USA
Xin.Zhang@asu.edu