Reasoning about Sensing Actions in Domains with Multi-Valued Fluents

Tran Cao Son
Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Phan Huy Tu
Computer Science
New Mexico State University
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

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 $\mathcal{AM}_K$, and provide a transition function based semantics for $\mathcal{AM}_K$. We define the entailment relationship between action theories and queries in $\mathcal{AM}_K$, denoted by $\models_{\mathcal{AM}_K}$. We discuss some complexity result about the planning problem in $\mathcal{AM}_K$. We demonstrate the use of $\mathcal{AM}_K$ through examples from the literature.

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. [Moore, 1985; Scherl and Levesque, 1993; Golden and Weld, 1996; Lobo *et al.*, 1997; Son and Baral, 2001; Thielscher, 2000]). Situation calculus based approaches such as that of [Moore, 1985; Scherl and Levesque, 1993; Golden and Weld, 1996] can be used in domains with *multi-valued fluents* since functional fluents are allowed in situation calculus. In [Golden and Weld, 1996], we can find several examples in the UNIX domain articulating the need for dealing with sensing actions in domains with multi-valued fluents. One of them is the action of listing the files in a directory, `ls`, that informs us all the file names in the current directory.

On the other hand, approaches to reasoning about sensing actions using high-level description language by extending the language $\mathcal{A}$ in [Gelfond and Lifschitz, 1993; 1998] such as [Lobo *et al.*, 1997; Son and Baral, 2001] concentrate on providing solution to the frame problem for domains with Boolean fluents and sensing actions. In [Baral *et al.*, 2000b], the approach of [Baral and Son, 1998] 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 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 is either *red*, *yellow*, or *green*. This fluent can be represented by three Boolean fluents *is_red*, *is_yellow*, and *is_green* denoting that the traffic light is either red, yellow, or green and the 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 [Son and Baral, 2001], we 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, because 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 expressing 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 $\mathcal{A}_K$ with multi-valued fluents. To the best of our knowledge, $\mathcal{AM}_K$ is the first high-level language with a transition function based

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

[2]In [Lobo *et al.*, 1997], **causes_to_knows** is used instead of **determines**.

semantics 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 [Giunchiglia *et al.*, 1997] 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 $\mathcal{K}$ [Eiter *et al.*, 2000] but this language does not allow sensing actions either.

In the next section we define $\mathcal{AM}_K$ and illustrate its use in examples taken from the literature. We 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.

2 Language $\mathcal{AM}_K$

2.1 Syntax

The alphabet of a domain description in $\mathcal{AM}_K$ consists of a set of action names **A**, and a set of fluent names **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 $v \in dom(f)$. An atom or its negation will be called as a *fluent literal*. For convenience, we often use $f \neq v$ to denote the negation of $f = v$ (or $\neg(f = v)$). *Fluent formulas* are composed from fluent literals using Boolean operations in the usual way. To describe a domain description, we use propositions of the following form:

$$\textbf{executable } a \textbf{ if } \varphi \tag{1}$$

$$a \textbf{ causes } l \textbf{ if } \varphi \tag{2}$$

$$l \textbf{ if } \varphi \tag{3}$$

$$a \textbf{ partitions } f \textbf{ into } \{V_j\}_{j \in S} \tag{4}$$

where

- a is an action name;
- l is fluent literal;
- φ is fluent formula; and
- S is a set of indexes and $\{V_j\}_{j \in S}$ is a (possibly infinite) partition of the domain of the fluent f, i.e., it is a 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 effects of actions, their executability conditions, and state constraints. Propositions of the form (1) state the conditions under which a is executable. It says that for a to be executable, the fluent formula φ must hold (precise meaning follows). Propositions of the form (2) describe the conditional effects of an action on the value of a fluent. It says that execution of a causes the literal l to be true if φ holds. Propositions of the form (3) describe the domain constraints, i.e., the relationship between fluents and propositions of the form (4) describe the effect of sensing actions. (3) states that if φ holds then l must be true. (4) says that executing an action a will help an agent to partially determine a fluent f in that it knows the possible values of f. In what follows, we will use "a **determines** f" as the shorthand for a proposition of the form (4) where each of the V_j contains exactly one element, i.e., executing a helps the agent to precisely determine what the value of f is.

Two effect propositions

$$a \textbf{ causes } l \textbf{ if } \varphi \quad \text{and} \quad a \textbf{ causes } l' \textbf{ if } \varphi'$$

are contradictory if there exists a state such that the φ and φ' hold simultaneously and l and l' are in contradiction in D.

A *domain description* D is a set of propositions of the forms (1)-(4) 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 (1) and by default **executable** a **if** $true$ belongs to D unless otherwise specified. We will also assume that each action a occurs in at most one proposition of the form (4). In the latter case, we will omit the proposition from the description.

We note that with the introducing of multi-valued fluents, the traffic light domain in the introduction becomes trivial. It can be represented by a single proposition $look$ **determines** $color$ with the obvious meaning associated to the action $look$ and the fluent $color$. We now demonstrate the use of $\mathcal{AM}_K$ through examples, taking from the literature. The first example is taken from the paper [Golden and Weld, 1996].

Example 1 (UNIX domain) In this domain, denoted by D_u, actions are UNIX commands such as `ls, cd, ping,` etc. Fluents in this domain are often multi-valued fluents. Below, we will demonstrate how some of the common UNIX commands can be represented in $\mathcal{AM}_K$.

The fluent $curdir$, denoting the current directory, has the domain as the set of valid directories. This fluent can be changed by the action `cd(X)`, where X is an existing directory, whose effect is described by the proposition

$$\texttt{cd}(X) \textbf{ causes } curdir = X \textbf{ if } exist(X).$$

The effect of the action `ls` can be described by the proposition

$$\texttt{ls} \textbf{ determines } files(curdir)$$

where $files(curdir)$ is a fluent whose domain consists of sets of valid file names.

The action `ls(X)`, on the other hand, determines whether or not the file X exists. This will be represented by the proposition

$$\texttt{ls}(X) \textbf{ determines } exist(X)$$

where $exist(X)$ is a Boolean fluent.

The command `ping(machine)` tells us whether the machine $machine$ is alive or not. This command can be represented by the proposition

$$\texttt{ping}(X) \textbf{ determines } alive(X).$$

One of the trivial causal relations of this domain says that if a directory exists then so does its parent directory. This information is represented by the following proposition

$$exist('dir') \textbf{ if } exist('dir/X').$$

The next example, taken from the list of examples of the system SGP [Weld *et al.*, 1998], 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 needs a particular medication. A patient is sick and we need to find an appropriate cure for her. Taking medication for the correct illness can cure the patient but using the wrong cure is fatal.

Performing a throat culture will return red, $blue$, or $white$. This color determines the group of illness that the patient has. For example, if the patient has i_5 then the color is $white$. 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.

The fluents in this domain are

- 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 (stands for $throat\_culture\_test\_done$), $dead$, and bsd (stands for $blood\_sample\_done$), each has the domain $\{true, false\}$.

The actions:

- $stain$: 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;
- $analyze\_blood$: executed only when the blood sample is count and determines whether hc is true;
- $medicate(X)$ takes the cure X where $X \in \{c_1, \ldots, c_5\}$;

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

$$\begin{array}{c}
\textbf{executable } inspect \textbf{ if } tcd \\
\textbf{executable } analyze\_blood \textbf{ if } bcd \\
stain \textbf{ causes } tcd \\
inspect \textbf{ determines } color \\
color{=}blue \textbf{ if } (i{=}i_3 \vee i{=}i_4) \wedge tcd \\
color{=}red \textbf{ if } (i{=}i_1 \vee i{=}i_2) \wedge tcd \\
color{=}white \textbf{ if } i{=}i_5 \wedge tcd \\
blood\_sample \textbf{ causes } bsd \\
analyze\_blood \textbf{ determines } hc \\
hc \textbf{ if } (i{=}i_1 \vee i{=}i_3 \vee i{=}i_5) \wedge bsd \\
\neg hc \textbf{ if } (i{=}i_2 \vee i{=}i_4) \wedge bsd \\
medicate(c_j) \textbf{ causes } i{=}none \textbf{ if } i{=}i_j \\
medicate(c_j) \textbf{ causes } dead \textbf{ if } i{\neq}i_j
\end{array}$$

where the last two propositions are for $j = 1, \ldots, 5$. For each action a, $a \neq inspect$ and $a \neq analyze\_blood$, the domain will contain a proposition of the form **executable** a **if** $true$.

In the next example, we consider a blocks world domain with two blocks and actions that tell the location of the block.

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; the robot can also *sense* whether it is holding block a (likewise, block b) or not. We will use $loc(X)$ where $X \in \{a, b\}$ to denote the location of the block X. We have that $dom(loc(a)) = \{onTable, inHand, on(b)\}$ and $dom(loc(b)) = \{onTable, inHand, on(a)\}$.

The domain, denoted by D_b, consists of the following propositions (X, Y stand for either a or b and $X \neq Y$):

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

The next example demonstrates that the partition created by sensing actions 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:

$$look\_at\_indicator \textbf{ determines } gas\_in\_tank$$

Notice the difference between the action $look\_at\_indicator$ and sensing actions in the previous examples. While $look\_at\_indicator$ partitions the domain of gas_in_tank into an infinite sequence of set of values, all partitions created by other actions are finite.

2.2 Observations

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

$$\textbf{initially } l \tag{5}$$

where l is a fluent literal. When l is of the form $f = v$ (resp. $f \neq v$), we say that (5) is a *positive* (resp. *negative*) observation. Notice the difference between the binary and multi-valued fluents cases. In the Boolean case, an arbitrary observation would allow us to know the value of the fluent occurred in it. On the other hand, a negative observation in multi-valued domains, say " **initially** $\neg f = v$" where $|dom(f)| > 2$, does not allow us to know the value of f. It only limits the domain of possible values of f.

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 will assume that each fluent in D occurs in at most one observation in O.

For the UNIX domain, some of the observations could be 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:

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

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

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

For the block worlds domain, we assume that the robot knows initially that a is 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, denoted by $O_g = \emptyset$.

In what follows, we will use (D_u, O_u), (D_s, O_s), (D_b, O_b), (D_g, O_g) to denote the UNIX domain, the illness domain, the block worlds domain, and the gas domain with the domain description in Examples 1, 2, 3, and 4 respectively, and the corresponding set of observations.

2.3 Queries

As discussed in [Levesque, 1996], 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 single action a;
- or a *if-then-else* plan of the form
  if φ **then** p_1 **else** p_2
  where φ is a fluent formula;
- or a *sequential plan* of the form
  $p_1; p_2$
  where p_1 and p_2 are plans.

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

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

$$\textbf{knows } \varphi \textbf{ after } p \quad (6)$$

$$\textbf{kwhether } \varphi \textbf{ after } p \quad (7)$$

where p is a plan and φ is a formula. Intuitively, the first query is about asking if an action theory entails that φ 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 φ will be known after executing the plan p in the initial situation.

As an example, it is obvious that (D_s, O_s) entails **knows** $\neg dead$ **after** [] because we know that the patient is not dead initially. However, the theory 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 $f = v$ if $s(f) = v$. s satisfies $\neg f = v$ if $s(f) \neq v$. When s satisfies a literal l, we write $s \models l$. The truth value of a fluent formula φ, denoted by $s(\varphi)$, with respect to an interpretation s is defined as usual. When $s(\varphi)$ is true we say that s satisfies φ and write $s \models \varphi$.

An interpretation s is a *state* if for every proposition of the form (3), 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 $s_2(f) \in V_j$ holds. 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 value of fluent f.

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

A fluent formula φ is known to be true in a c-state $\sigma = \langle s, \Sigma \rangle$ if $s \models \varphi$ for every state $s' \in \Sigma$. φ is known to be false in $\sigma = \langle s, \Sigma \rangle$ if $s \models \neg\varphi$ for every $s' \in \Sigma$.

Before we define the transition function of a domain D, we introduce some more notation. Given a set of fluent formulae S and a formula φ, we say that S satisfies φ, denoted by $S \models \varphi$, if φ logically follows from S. A set S of fluent formulae is said to be closed under a set of domain constraints (form (3)) K if for every constraint "l **if** φ" in K, whenever $S \models \varphi$, so does $S \models l$. By $Cn(S \cup K)$ we denote the least logically closed set of formulae from D that contains S and is also closed under K.

An action a is executable in a state s, if there exists a proposition (1) in D such that $s \models \varphi$. For an action a and a state s, if a is executable in s, the effect of a on s, denoted by $e(a, s)$, is defined by the set

$$e(a, s) = \{l \mid D \text{ contains an effect proposition (2) such that } s \models \varphi\}$$

The set of states resulting from the execution of a non-sensing action a in a state s is defined by

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

where R denotes the set of proposition of the form (3) in D. Intuitively, $Res(a, s)$ is the set of states resulting from executing a in s. We are now ready to define Φ, the transition function between c-states.

Definition 1 (Transition Function) *A function Φ from actions and c-states into c-states is called a* transition function *of D if for all c-state $\sigma = \langle s, \Sigma \rangle$ and action a,*

- *if a is not executable in s then $\Phi(a, \sigma)$ is undefined, denoted by $\Phi(a, \sigma) = \bot$;*

- *if a is executable in s and a is a non-sensing action, then $\Phi(a,\sigma) = \langle s_1, \Sigma_1\rangle$ where $s_1 \in Res(a,s)$ and $\Sigma_1 = \{s' \mid s' \in Res(a,s'')$ for some $s'' \in \Sigma$ such that a is executable in $s''\}$; and*
- *if a is executable in s and a is a sensing action with*

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

  in D then $\Phi(a,\sigma) = \langle s, \Sigma'\rangle$ where $\Sigma' = \{s' \mid s' \in \Sigma$ such that $s \overset{f,\{V_j\}_{j\in S}}{\sim} s'$ and a is executable in $s'\}$.

The above definition corresponds to the three possible cases when an action is executed: the action is non-executable, non-sensing action, or a sensing action, respectively. In the next definition, we define the initial state and initial c-state of an action theory.

Definition 2 (Initial State) *For an action theory (D,O),*

- *a state s is called an* initial state *of (D,O) if $s \models l$ for every proposition*

  $$\textbf{initially } l$$

  in O;
- *a c-state $\langle s_0, \Sigma_0\rangle$ is an* initial c-state *of (D,O) if s_0 is an initial state and Σ_0 is a set of initial states of (D,O).*

In order to determine the states resulting from the execution of a plan from a state, we define a function $\hat{\Phi}$ that extends the transition function Φ as follows.

Definition 3 (Extended Transition Function) *Let (D,O) be an action theory, Φ be a transition function, p be a plan and $\sigma = \langle s, \Sigma\rangle$ be a c-state. The extended transition function of (D,O), denoted by $\hat{\Phi}$, which maps a pair of plans and c-states into c-states, is defined as follows:*

- *$\hat{\Phi}([], \sigma) = \sigma$.*
- *For an action a, $\hat{\Phi}(a,\sigma) = \Phi(a,\sigma)$.*
- *For $p =$ **if** φ **then** p_1 **else** p_2, where φ is a fluent formula, p_1 and p_2 are plans,*

  $$\hat{\Phi}(p,\sigma) = \begin{cases} \hat{\Phi}(p_1,\sigma) & \textit{if } \varphi \text{ is known to be true in } \sigma; \\ \hat{\Phi}(p_2,\sigma) & \textit{if } \varphi \text{ is known to be false in } \sigma; \\ \perp & \textit{if } \varphi \text{ is unknown in } \sigma. \end{cases}$$
- *For $p = p_1; p_2$, where p_1, p_2 are plans, $\hat{\Phi}(p,\sigma) = \hat{\Phi}(p_2, \hat{\Phi}(p_1,\sigma))$.*
- *$\hat{\Phi}(p,\perp) = \perp$ for every plan p.*

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

Definition 4 (Entailment) *An action theory (D,O) entails a query*

$$\textbf{knows } \varphi \textbf{ after } p$$

*(denoted by $(D,O) \models_{\mathcal{AM}_K}$ **knows** φ **after** p) if for every transition function Φ of (D,O) and each initial c-state $\sigma_0 = \langle s_0, \Sigma_0\rangle$ the following conditions are satisfied:*

- *$\hat{\Phi}(p,\sigma_0) \neq \perp$ and*
- *φ is known to be true in $\hat{\Phi}(p,\sigma_0)$.*

Similarly, (D,O) entails the query

$$\textbf{kwhether } \varphi \textbf{ after } p$$

*(denoted by $(D,O) \models_{\mathcal{AM}_K}$ **kwhether** φ **after** p) if for every transition function Φ of (D,O) and each initial c-state $\sigma_0 = \langle s_0, \Sigma_0\rangle$ the following conditions are satisfied:*

- *$\hat{\Phi}(p,\sigma_0) \neq \perp$ and*
- *φ is known to be true in $\hat{\Phi}(p,\sigma_0)$ or φ is known to be false in $\hat{\Phi}(p,\sigma_0)$.*

We now illustrate the above definitions using the domains (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 hold for the UNIX domain.

Proposition 1 *For the UNIX domain,*

- *$(D_u,O_u) \models_{\mathcal{AM}_K}$ **knows** $curdir = '/mydoc'$ **after** $cd('/mydoc')$.*
- *$(D_u,O_u) \models_{\mathcal{AM}_K}$ **kwhether** $exist(a)$ **after** $ls(a)$.*

The first item tells us that the current directory will change to `'/mydoc'` if we execute the action `cd('/mydoc')` (the directory exists since the directory `'/mydoc/paper'` exists. The second one indicates that we will know whether the file `a` exists if we execute the action `ls(a)`. 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 2 *For the domain (D_s,O_s), $(D_s,O_s) \models_{\mathcal{AM}_K}$ **kwhether** $i = i_j$ **after** p_s for every $j \in \{1,\ldots,5\}$.*

Proof. The concrete computation of the states and c-states is given in Appendix A. □

Let $p_6 = []$ and for $j = 1,\ldots,5$, let $p_j =$ **if** $i = i_j$ **then** $medicate(c_j)$ **else** p_{j+1}. The next proposition follows from the above proposition.

Proposition 3 *For the sick domain, $(D_s,O_s) \models_{\mathcal{AM}_K}$ **knows** $i = none$ **after** $p_s; p_1$.*

On the other hand, $(D_b,O_b) \not\models_{\mathcal{AM}_K}$ **kwhether** $(loc(b) = on(a))$ **after** $sense(b)$. This is because the sensing action $sense(b)$ does not tell us the exact location of b. The concrete computation is given in Appendix A.

Proposition 4 *For the gas domain and a number $v \in [0,20]$, $(D_g,O_g) \models_{\mathcal{AM}_K}$ **kwhether** $gas\_in\_tank = v$ **after** $look$.*

Intuitively, the proposition says that executing the action $look$ will tell us whether or not the tank is empty.

Proof. Observe that any state in this domain consists of a single atom $gas\_in\_tank = v$ for some $v \in [0,20]$. Let $s_v = \{gas\_in\_tank = v\}$. Clearly, $\Sigma_0 = \{s_v \mid v \in [0,20]\}$ is the set of initial states. Furthermore, for every $v \in [0,20]$, $\langle s_v, \Sigma_0\rangle$ is an initial c-state. Consider an arbitrary initial c-state $\sigma_0 = \langle s_v, \Sigma_0\rangle$ and transition function Φ. We have that $\Phi(look, \sigma_0) = \langle s_v, \{s_v\}\rangle$. This holds for every initial c-state and transition function Φ. Therefore, we can conclude that $(D_g,O_g) \models$ **kwhether** $(gas\_in\_tank = 0)$ **after** $look$.

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

As we have informally discussed in the introduction, the addition of discrete, multi-valued fluents to create $\mathcal{AM}_K$ does not increase its expressiveness in comparing to its predecessor $\mathcal{A}_K$ (by $\mathcal{A}_K$ we mean the language $\mathcal{A}_K$ extended with state constraints in [Baral *et al.*, 2000b]). 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). For simplicity, we will assume that domains of fluents in (D, O) are pairwise disjoint (this could be done by associating each value of a fluent with its name). We sketch below the translation. In our description, we place the subscript b to component of the theory (D, O) to denote its correspondence in (D_b, O_b). For instance, l_b or φ_b denotes the correspondence of a literal l or formula φ, respectively. The translation is as follows.

- For each fluent f, D_b contains a set of Boolean fluents $\{is_f(f, v_f) \mid v_f \in dom(f)\}$ and a set of constraints $\{is_f(f, v_f)$ **if** $\bigwedge_{v' \in dom(f) \setminus v_f} \neg is_f(f, v') \mid v_f \in dom(f)\}$ and a set of constraints $\{\neg is_f(f, v_f)$ **if** $is_f(f, v') \mid v_f, v' \in dom(f)$ and $v_f \neq v'\}$.
- For an atom l, l is of the form $f = v_f$ (resp. $f \neq v_f$), $l_b = is_f(f, v_f)$ (resp. $l_b = \neg is_f(f, v_f)$).
- φ_b is obtained from φ by replacing each literal l that occurs in φ by l_b.
- Each proposition of the form (1) is translated into **executable** a **if** φ_b.
- Each proposition of the form (2) is converted into a **causes** l_b **if** φ_b.
- Each proposition of the form (3) is converted into l_b **if** φ_b.
- Each proposition of the form (4) is converted into a set of propositions $\{a$ **determines** $is_f(f, v_f) \mid v_f \in dom(f) \setminus \{v_0\}\}$ where v_0 is an arbitrary constant belonging to $dom(f)$.
- Each positive observation **initially** $f = v$ is translated into an observation **initially** $is_f(f, v_f)$ and each negative observation **initially** $f \neq v$ is translated into an observation **initially** $\neg is_f(f, v_f)$.

We will now discuss some properties of the translation from (D, O) into (D_b, O_b). We will begin with a discussion on the complexity of the translation. First, note that the size of an action theory (D, O) depends on

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

Let us define the length of a formula φ by the number of literals occurring in it. Assume that we only need constant amount of bytes to encode fluents and actions, we can define the size of a proposition in (D, O) by the length of the formulae occurring in it, e.g., the size of a **causes** f **if** φ is the length of φ. Then, the size of an action theory (D, O) is defined as the sum of (i) the sum over the size of all propositions in (D, O); (ii) the number of actions; (iii) the number of fluents; (iv) the sum over $|dom(f)|$ for $f \in \mathbf{F}$. Under the assumption that (D, O) is a discrete and finite multi-valued fluents action theory, we have that

- The translation of each fluent f into the set of fluents and constraints in (D_b, O_b) is bounded by a polynomial function of the size of $dom(f)$;
- The translation of a fluent formula φ into φ_b is bounded by a polynomial function of the length of φ;
- The translation of a proposition of the form (1)– (3) in (D, O) into a proposition in (D_b, O_b) is bounded by a polynomial function of the length of φ; and
- The translation of a proposition of the form (4) in (D, O) into a proposition in (D_b, O_b) is bounded by a polynomial function of $|dom(f)|$.

This shows that the following proposition holds.

Proposition 5 *For each 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.*

It is easy to see that under the assumption that domains of fluents in (D, O) are pairwise disjoint, a fluent formula φ_b in (D_b, O_b) could be translated into a unique formula φ in (D, O) whose binary version is exactly φ_b. We can prove the following

Proposition 6 *For each discrete and finite multi-valued fluents action theory (D, O), a plan p, and formula φ*

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

$$\text{iff}$$

$$(D_b, O_b) \models_{\mathcal{A}_K} \textbf{knows } \varphi_b \textbf{ after } p_b.$$

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 the 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 and discrete, finite domains[3], the computational complexity results in [Baral *et al.*, 2000a] will hold, i.e., planning in $\mathcal{AM}_K$ is not harder than planning 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 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 examples taken from the literature. We also show that adding multi-valued fluents does not result in a increasing of expressiveness but also in

[3](D, O) is deterministic if for every c-state σ and non-sensing action a, $|\Phi(a, \sigma)| \leq 1$. Examples of deterministic action theories include theories without propositions of the form (3).

complexity of planning with sensing actions in discrete, finite domains. This opens the possibility for the application of results in approximating $\models_{\mathcal{A}_K}$ (e.g. in [Son and Baral, 2001]) in dealing with multi-valued fluents and the problem of the huge number of states, we propose an approximation of $\models_{\mathcal{AM}_K}$. We leave it as one of the topics of our future research. We are using $\mathcal{AM}_K$ in developing a planner for planning with sensing actions in the presence of incomplete information. Two interesting problems arise: (i) specifying the fluent domains; (ii) creating multiple level of nested **if-then-else**. The first problem arises since listing all possible values of a fluent is sometime time-consuming task (what happens if the domain is huge, say an integer less than 10 million?). In several cases, this can be addressed by using a function call to an external function that validates the value of a fluent. The second problem arises because a sensing action cause several branches at the same time (e.g., we would like to list all the files (action `ls`) and do something with each file afterward knowing the file names). Currently, we are investigating the use of another construct such as **for-all**.

References

[Baral and Son, 1998] C. Baral and T.C. Son. Formalizing sensing actions: a transition function based approach. In *AAAI Fall Symposium - Cognitive Robotics*, pages 13–20, 1998.

[Baral *et al.*, 2000a] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122:241–267, 2000.

[Baral *et al.*, 2000b] C. Baral, S. McIlraith, and T.C. Son. 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)*, pages 311–322, 2000.

[Eiter *et al.*, 2000] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete information. In *Proceedings of the First International Conference on Computational Logic (CL'00)*, pages 807–821. Springer Verlag, LNAI 1861, 2000.

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

[Gelfond and Lifschitz, 1998] M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 3(6), 1998.

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

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

[Levesque, 1996] H. Levesque. What is planning in the presence of sensing? In *Proceedings of the 14th Conference on Artificial Intelligence*, pages 1139–1146. AAAI Press, 1996.

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

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

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

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

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

[Weld *et al.*, 1998] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI 98*, 1998.

Appendix A

Computation for the blocks world domain: It is easy to see that the domain has only 5 states:
$s_1 = \{loc(a) = onTable, loc(b) = onTable\}$,
$s_2 = \{loc(a) = onTable, loc(b) = inHand\}$,
$s_3 = \{loc(a) = onTable, loc(b) = on(a)\}$,
$s_4 = \{loc(a) = inHand, loc(b) = onTable\}$, and
$s_5 = \{loc(a) = on(b), loc(b) = onTable\}$

and the set of initial states $\Sigma_0 = \{s_1, s_2, s_3\}$ that gives us three initial c-states $\langle s_i, \Sigma_0\rangle$ $(i = 1, 2, 3)$. Execution the action $sense(b)$ resulting into three c-states $\langle s_1, \{s_1, s_3\}\rangle$, $\langle s_3, \{s_1, s_3\}\rangle$, and $\langle s_2, \{s_2\}\rangle$. Following the definition, we have that $(D_b, O_b) \not\models_{\mathcal{AM}_K}$ **kwhether** $loc(b) = on(a)$ **after** $sense(b)$.

Computation for the proof of Proposition 2: Let us denote a state of this domain by a tuple (i, d, tcd, bsd, c, hc) where d stands for $dead$ and c stands for $color$. It is easy to see that there are 30 initial states since i can take the values $\{i_1, \ldots, i_5\}$, c 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 to 10 and there are only 10 possible c-states after the execution of $stain$ whose k-state contains exactly the 10 possible states. Inspection 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-statesto 5 and analyzing the blood will result in 5 c-states, whose top and bottom are exactly the same state. That means that after the above sequence of action, one can know which illness the patient has, and hence, find the correct cure to use.

	i	d	tcd	bsd	c	hc	# s	# c-states
Ini	i_1	f	f	f	u	u	30	30 (Σ=ALL)
	i_2							
	i_3							
	i_4							
	i_5							
stn	i_1	f	t	f	r	u	2	2 (Σ=ALL)
	i_2	f	t	f	r	u	2	2 (Σ=ALL)
	i_3	f	t	f	b	u	2	2 (Σ=ALL)
	i_4	f	t	f	b	u	2	2 (Σ=ALL)
	i_5	f	t	f	w	u	2	2 (Σ=ALL)
ins	i_1	f	t	f	r	u	2	2 (Σ=$\{i_1, i_2\}$)
	i_2	f	t	f	r	u	2	2 (Σ=$\{i_1, i_2\}$)
	i_3	f	t	f	b	u	2	2 (Σ=$\{i_3, i_4\}$)
	i_4	f	t	f	b	u	2	2 (Σ=$\{i_3, i_4\}$)
	i_5	f	t	f	w	u	2	2 (Σ=$\{i_5\}$)
bls	i_1	f	t	t	r	t	1	1 (Σ=$\{i_1, i_2\}$)
	i_2	f	t	t	r	f	1	1 (Σ=$\{i_1, i_2\}$)
	i_3	f	t	t	b	t	1	1 (Σ=$\{i_3, i_4\}$)
	i_4	f	t	t	b	f	1	1 (Σ=$\{i_3, i_4\}$)
	i_5	f	t	t	w	t	1	1 (Σ=$\{i_5\}$)
anb	i_1	f	t	t	r	t	1	1 (Σ=$\{i_1\}$)
	i_2	f	t	t	r	f	1	1 (Σ=$\{i_2\}$)
	i_3	f	t	t	b	t	1	1 (Σ=$\{i_3\}$)
	i_4	f	t	t	b	f	1	1 (Σ=$\{i_4\}$)
	i_5	f	t	t	w	t	1	1 (Σ=$\{i_5\}$)

In the above table, *r, b, w* denotes *red, blue, white*, respectively. *t,f* stands for *true, false*, respectively. u stands for unknown. Ini, stn, ins, bls, and anb stands for $Initial$, $stain$, $inspect$, $blood\_sample$, and $analyze\_blood$, respectively. 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. In each of the subtable, e.g. the one with *stain* in the first column and the 5 rows, the c-states resulting from executing the action *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 Σ is the set of states specified by the rows which have the value of i belonging to the set specified in the left side of the equation in the last column (ALL indicates all the possible states).