Sensing and NonDeterministic Actions in Domains with Multi-Valued Fluents

Tran Cao Son and **Phan Huy Tu** [1]

Abstract. We develop a high-level action description language, called $\mathcal{AM}_K$, with a transition function based semantics for representing and reasoning with (*i*) non-deterministic actions; (*ii*) sensing actions; and (*iii*) multi-valued fluents. We demonstrate the use of $\mathcal{AM}_K$ through examples. To use the language in a logic-programming based planner, we develop an approximation of the entailment relationship between action theories and queries in $\mathcal{AM}_K$ for a subclass of $\mathcal{AM}_K$ action theories.

1 Introduction

Among various approaches to reasoning about actions and changes, approaches using high-level action description languages are appealing for its simple and easy to understand semantics. In these approaches, the semantics of an action theory is defined by a transition function that specifies the state(s) resulting from execution of an action in a state. This approach was first introduced in [4], in which the action language $\mathcal{A}$ was introduced.

Since $\mathcal{A}$ was introduced, several dialects of $\mathcal{A}$ have been proposed to address various limitations of this language. For instance, [5] considers domain constraints, nonpropositional (hereafter referred to as multi-valued fluents) fluents, nondeterministic actions and actions with indirect effects, [9, 10] allows sensing actions, [1] deals with concurrent actions, [2] addresses observations and hypotheses, etc. However, none of these dialects considers nondeterministic actions, domain constraints, sensing actions, and multi-valued fluents.

In this paper, we develop a language, called $\mathcal{AM}_K$, for representing and reasoning about sensing and nondeterministic actions in domains with multi-valued fluents. We will show, by examples only, that the language allows a more compact and concise representation of action theories in domains with multi-valued fluents. The semantics of $\mathcal{AM}_K$ will be defined by a transition function, similar to its predecessor $\mathcal{A}_K$ of [10]. This function is used to define the entailment relationship $\models_{\mathcal{AM}_K}$ between action theories and queries in $\mathcal{AM}_K$. To address the high complexity of computing $\models_{\mathcal{AM}_K}$, we propose an approximation of $\models_{\mathcal{AM}_K}$, called the 0-approximation.

Related Work: Among the high-level action description languages that were developed after the introduction of $\mathcal{A}$, $\mathcal{AM}_K$ subsumes $\mathcal{A}_K$ [10] (sensing actions) and is closely related to $\mathcal{AR}$ [5] (nonpropositional fluents and indirect effects of actions), and $\mathcal{C}^+$ [6] (nonpropositional fluents and nondeterministic actions). Neither $\mathcal{AR}$ nor $\mathcal{C}^+$ consider sensing actions. Furthermore, both of these languages require that domains of nonpropositional fluents are finite.

[1] New Mexico State University, Las Cruces, NM88003, USA

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 $dom(f)$ prescribing what value f can take. $dom(f)$ needs not be finite.

In $\mathcal{AM}_K$, an *m-atom* is of the form $f \in V$ where f is a fluent in **F** and $V \subseteq dom(f)$. f and V will be called the *fluent* and the *value set* of the m-atom $f \in V$, respectively. An m-atom $f \in V$ is *non-empty* if $V \neq \emptyset$. Essentially, an m-atom states that the value of f belongs to V, i.e., the disjunction $\bigvee_{v \in V}(f = v)$ holds. For convenience, we use $f \notin V$, $f = v$ and $f \neq v$, where $V \subseteq dom(f)$ and $v \in dom(f)$, to denote the m-atoms $f \in dom(f) \setminus V$, $f \in \{v\}$ and $f \in dom(f) \setminus \{v\}$ correspondingly. In addition, for a boolean fluent f, we sometimes use f and $\neg f$ as shorthands for $f{=}true$ and $f{=}false$ respectively. An *m-literal* is of the form $f \in V$ or $\neg(f \in V)$ (written as $f \notin V$). A *fluent formula* is composed from m-literals using propositional operators in the usual way. Since $f \notin V$ is equivalent to $f \in V'{=}dom(f) \setminus V$ and $a \vee b \equiv \neg(\neg a \wedge \neg b)$ is a valid propositional formula, a formula defined by m-literals and the propositional operators can be converted into a conjunction of m-atoms. For this reason, we will assume hereafter that fluent formulae in our propositions are conjunctions of m-atoms. A domain description is a collection of propositions of the following forms:

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

$$a \textbf{ causes } f \in V \textbf{ if } \varphi \tag{2}$$

$$a \textbf{ partitions } f \textbf{ into } \{V_j\}_{j \in S} \tag{3}$$

$$f \in V \textbf{ if } \varphi \tag{4}$$

where a is an action name; φ is a fluent formula; f is a fluent and V is a subset of $dom(f)$; and S is a set of indices 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 subsets of $dom(f)$ such that $\cup_{j \in S} V_j = dom(f)$.

Intuitively, the above propositions describe the effect of actions, their executability conditions, and state constraints. Propositions of form (1) state the conditions under which a is executable. It says that for a to be executable, the fluent formula φ must hold (the precise meaning of *hold* is given in the next section). Propositions of form (2), called *dynamic laws*, describe the conditional effects of a non-sensing action on the value of a fluent. It says that the execution of a causes the fluent f to take some value in V if φ holds. Apparently, by the introduction of these dynamic laws, nondeterministic actions now can be represented in the language. Propositions of form (3) describe the effect of sensing actions. It 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 form (3) 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. Propositions of form (4), called *static causal laws*, describe the domain constraints, i.e., the relationship between fluents. (4) states that if φ holds then f must take some value in V.

A *domain description* D is a set of propositions of forms (1)-(4). For simplicity, we assume that for every domain description D, each action a occurs in at most one proposition of form (1) and by default "**executable** a **if** $true$" belongs to D unless otherwise is specified. We will also assume that each action a occurs in at most one proposition of form (3). D is *definite* if every proposition (3) is of the form "a **determines** f".

With the introduction of multi-valued fluents, the representation of domains with multi-valued fluent becomes easier which often results in a more compact and more natural representation of action theories. The following examples demonstrate the use of $\mathcal{AM}_K$ in representing action theories.

Example 1 Consider a traffic light and the action *look*. Looking at the light will tell us whether the traffic light is either *red, yellow*, or *green*.

In $\mathcal{AM}_K$ notation, we will need a fluent, say *color*, whose domain is $\{$*red, yellow, green*$\}$ and the proposition "$look$ **determines** $color$".

Without multi-valued fluents, we would have to use three Boolean fluents, say *is_red, is_yellow*, and *is_green*, to record that the traffic light is either red, yellow, or green and a set of state constraints stating that it has only one color at a time such as[2] "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 representation, we need to specify the effect of the action *look*. Using the language $\mathcal{A}_K$ in [10, 9], we need to introduce at least two of the following three propositions[3], "$look$ **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. Obviously, this representation is not natural and compact when compared with the $\mathcal{AM}_K$ representation.

The next example is more complicated and is taken from the domains of the system SGP [12].

Example 2 (Illness) 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.

Performing a throat culture will return either red, $blue$, or $white$. This color determines the group of illness that the patient has. Specifically, if the color is $white$ then the patient has the illness i_5; if the color is $blue$ then the patient has either i_3 or i_4; and if the color is red then the illness the patient has is either i_1 or i_2. 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 $dom(i)=\{i_1, \ldots, i_5, 0\}$; where 0 denotes that the patient is healthy;
- $color$ with $dom(color) = \{red,\ blue,\ white\}$;
- hc ($high\_blood\_count$), tcd ($throat\_culture\_test\_done$), $dead$, and bsd ($blood\_sample\_done$), each has the boolean domain $\{true, false\}$.

The domain consists of the following actions:

- $stain$: performing a throat culture, making tcd true;
- $inspect$: determining the color of the test, executable only when the throat culture is done;
- $sample$: sampling blood, making bsd true;
- $analyze$: determining hc, executable only when the blood sample is count; and
- $med(X)$: taking the cure X where $X \in \{c_1, \ldots, c_5\}$;

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

$stain$ **causes** tcd	$i \in \{i_3, i_4\}$ **if** $color{=}blue \wedge tcd \wedge i \neq 0$
$inspect$ **determines** $color$	$i \in \{i_1, i_2\}$ **if** $color{=}red \wedge tcd \wedge i \neq 0$
$sample$ **causes** bsd	$i = i_5$ **if** $color{=}white \wedge tcd \wedge i \neq 0$
$analyze$ **determines** hc	$i \in \{i_1, i_3, i_5\}$ **if** $hc \wedge bsd \wedge i \neq 0$
$med(c_j)$ **causes** $i{=}0$ **if** $i{=}i_j$	$i \in \{i_2, i_4\}$ **if** $\neg hc \wedge bsd \wedge i \neq 0$
$med(c_j)$ **causes** $dead$ **if** $i{\neq}i_j$	
executable $inspect$ **if** tcd	**executable** $analyze$ **if** bsd

where $j = 1, \ldots, 5$. In addition to these propositions, the domain also contains a proposition "**executable** a **if** $true$" for every action $a \notin \{inspect, analyze\}$.

The next example shows that the language does not require the finiteness of domains of fluents.

Example 3 (Gas) Consider a simple domain that consists of one fluent gas_in_tank and only one action $look\_at\_indicator$. Looking at the gas meter 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. Notice there is a difference between the sensing action $look\_at\_indicator$ in this example and the sensing actions *inspect* and *analyze_blood* in the previous example. While $look\_at\_indicator$ partitions the domain of gas_in_tank into an infinite sequence of sets of values, the partitions created by *inspect* and *analyze_blood* are finite.

The next example shows how nondeterministic actions can be represented in $\mathcal{AM}_K$.

Example 4 (Coin) The action of tossing a coin can be represented in $\mathcal{AM}_K$ by the proposition: $toss$ **causes** $coin \in \{head, tail\}$.

2.2 Observations

In $\mathcal{AM}_K$, an *observation* is of the form **initially** l (5) where l is an m-atom.

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. Apparently this simplicity does not reduce the expressiveness of the language.

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

$$O_s = \left\{ \begin{array}{ll} \textbf{initially}\ i \neq 0 & \textbf{initially}\ \neg dead \\ \textbf{initially}\ \neg tcd & \textbf{initially}\ \neg bsd \end{array} \right.$$

For the gas domain, we initially know nothing about the amount of gas available in the tank. The set of observations is therefore empty,

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

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

denoted by $O_g = \emptyset$. In what follows, we will use (D_s, O_s) and (D_g, O_g) to denote the action theories for the illness domain and the gas domain respectively.

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 $\mathcal{AM}_K$, a *plan* is defined to be either

- an empty sequence of action, denoted by [ ]; or
- a single action a; or
- an *if-then-else* plan of the form **if** φ **then** p_1 **else** p_2; or
- a *sequential plan* of the form $p_1; p_2$

where φ is a fluent formula, p_1 and p_2 are plans.

Intuitively, the if-then-else plan is a branching statement in which the agent has to determine what it should do next after evaluating the condition φ w.r.t its current knowledge. If it knows that φ is true (resp. false), it will execute p_1 (resp. p_2). Otherwise (φ is unknown), the if-then-else plan will fail and the execution of the whole plan which contains this if-then-else subplan will also fail.

For example, p_s=[$stain; inspect; sample;\ analyze$] is a sequential plan in the illness domain. Let p_6=[] and for j=$1, \ldots, 5$ and let p_j=[**if** i=i_j **then** $med(c_j)$ **else** p_{j+1}]. Then the p_j's, where j=$1, \ldots, 5$, are if-then-else plans.

In $\mathcal{AM}_K$, there are two kinds of queries that we can ask our action theories:

$\mathbf{knows}\ \varphi$ **after** p and/or $\mathbf{kwhether}\ \varphi$ **after** p

where p is a plan and φ is a formula. The first one 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 one 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 intuitive to assert that (D_s, O_s) entails $\mathbf{knows}\ \neg dead$ **after** [] because initially we know that the patient is not dead. The theory, however, should not entail $\mathbf{kwhether}\ i = i_1$ **after** [] because we do not know whether the patient has the sickness i_1 or not, i.e., we cannot confirm or deny this conclusion.

2.4 Semantics

For a domain description D, a *fluent assignment* (or *assignment*) for $f \in \mathbf{F}$ is of the form $f = v$ where $v \in dom(f)$. A set of assignments s is *consistent* if it does not contain two assignments $f = v$ and $f = v'$ with $v \neq v'$ for some $f \in \mathbf{F}$. For a consistent set of assignments s and a fluent f, s_f denotes the value of f in s, and $s_\mathbf{F}$ denotes the set of fluents occurring in s. More precisely, s_f=v if f=$v \in s$ and s_f is undefined otherwise; $s_\mathbf{F}$=$\{f \mid \exists v.[f = v \in s]\}$. s satisfies an m-atom $l = f \in V$, denoted by $s \models l$, if s_f is defined and $s_f \in V$. s satisfies a conjunction of m-atom φ, denoted by $s \models \varphi$, if s satisfies every conjunct of φ. We also say that $f \in V$ (resp. φ) holds in s if s satisfies an m-atom $f \in V$ (resp. φ).

A consistent set of assignments s with $s_\mathbf{F} = \mathbf{F}$ is called a *state* if for every proposition of the form (4), whenever $s \models \varphi$ holds, so does $s \models f \in V$. Two states s_1 and s_2 *agree* on a fluent f w.r.t 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 thinks he might be in when he does not know exactly the value of fluent f.

An action a with the set of dynamic laws

$\{a$ **causes** $f \in V_1$ **if** $\varphi_1, \ldots, a$ **causes** $f \in V_n$ **if** $\varphi_n\}$

is *contradictory* if there exists a state s and a set of indices $\{j_1, \ldots, j_k\} \subseteq \{1, \ldots, n\}$ such that $\varphi_{j_1}, \ldots, \varphi_{j_k}$ hold in s and $\bigcap_{t=1}^{k} V_{j_t} = \emptyset$. A domain description is *non-contradictory* if it does not contain contradictory actions. In this paper, we assume that domain descriptions in consideration are non-contradictory.

A *knowledge state* (or *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. $\sigma = \langle s, \Sigma\rangle$ is *grounded* if $s \in \Sigma$. 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 $s' \in \Sigma$. Otherwise, φ is *unknown* in $\sigma = \langle s, \Sigma\rangle$.

A set of assignments u is closed under a set of domain constraints (form (4)) K if for every constraint "$f \in V$ **if** φ" in K, whenever $u \models \varphi$, so does $u \models f \in V$. By $Cn(u \cup K)$ we denote a minimal set of assignments that contains u and is also closed under K.

An action a is said to be 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, let $V^f_{a,s} = \{V \mid \exists [a\ \mathbf{causes}\ f \in V\ \mathbf{if}\ \varphi] \in D$ and $s \models \varphi\}$, and $F_{a,s} = \{f \mid \exists [a$ **causes** $f \in V$ **if** $\varphi] \in D$ and $s \models \varphi\}$. Intuitively, $\bigcap_{V \in V^f_{a,s}} V$ is the set of possible values of f after the execution of a in s, i.e., if s' is a state resulting from the execution of a in s then $s'_f \in \bigcap_{V \in V^f_{a,s}} V$. $F_{a,s}$ is the set of fluents whose values are changed by a. An *effect* of the execution of a in s is a consistent set of assignments r such that $r_f \in \bigcap_{V \in V^f_{a,s}} V$ for each $f \in F_{a,s}$ and r_f is undefined for $f \in \mathbf{F} \setminus F_{a,s}$. The set of states resulting from the execution of a non-sensing action a in a state s is defined by

$$Res(a, s) = \{s' \mid s' \text{ is a state s.t. } Cn(s') = Cn((s \cap s') \cup e_{a,s} \cup R_D) \text{ and } e_{a,s} \text{ is an effect of } a \text{ in } s\}$$

where R_D is the set of proposition of the form (4) in D. We will now define the transition function Φ of D that maps pairs of actions and c-states into sets of c-states.

Definition 1 (Transition Function) *Let D be a domain description D, a be an action, and $\sigma = \langle s, \Sigma\rangle$ be a c-state. $\Phi(a, \sigma)$ is defined as follows:*

- *if $\sigma = \bot$ or a is not executable in s then $\Phi(a, \sigma) = \{\bot\}$, where $\bot$ denotes an undefined c-state; otherwise,*
- *if a is a non-sensing action, 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 with 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. $s' \overset{f, \{V_j\}_{j\in S}}{\sim} s$ and a is executable in $s'\}$,*

The above definition corresponds to three possible cases when an action is executed: (1) the state is undefined or the action is not executable, (2) the action is a non-sensing action, and (3) the action is a sensing action. We have the following proposition about the transition function.

Proposition 1 *Let σ be a c-state and a be an action. If σ is grounded and a is executable in σ then every c-state in $\Phi(a, \sigma)$ is also grounded.*

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* **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 Σ_0 is the set of all initial states of (D, O).*

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

Definition 3 (Extended Transition Function) *Given an action theory (D, O), a plan p and a c-state $\sigma = \langle s, \Sigma \rangle$, the extended transition function $\hat{\Phi}$ that maps pairs of plans and c-states into sets of c-states is defined as follows:*
- *if $\sigma = \bot$ then $\hat{\Phi}(p, \sigma) = \{\bot\}$; otherwise,*
- *if $p = []$ then $\hat{\Phi}(p, \sigma) = \{\sigma\}$,*
- *if $p = [a]$, $\hat{\Phi}(p, \sigma) = \Phi(a, \sigma)$,*
- *if $p = [$ **if** φ **then** p_1 **else** $p_2]$, where φ is a fluent formula, p_1 and p_2 are plans, then*

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

- *if $p = [p_1; p_2]$, where p_1 and p_2 are plans, $\hat{\Phi}(p, \sigma) = \bigcup_{\sigma' \in \hat{\Phi}(p_1, \sigma)} \hat{\Phi}(p_2, \sigma')$.*

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

Definition 4 (Entailment) *An action theory (D, O) entails a query* **knows** *φ* **after** *p, denoted by $(D, O) \models_{\mathcal{AM}_K}$* **knows** *$\varphi$* **after** *$p$, if for every initial c-state $\sigma_0 = \langle s_0, \Sigma_0 \rangle$, $\bot \notin \hat{\Phi}(p, \sigma_0)$ and φ is known to be true in every c-state in $\hat{\Phi}(p, \sigma_0)$.*

Similarly, (D, O) entails a query **kwhether** *φ* **after** *p, denoted by $(D, O) \models_{\mathcal{AM}_K}$* **kwhether** *$\varphi$* **after** *$p$, if for every initial c-state $\sigma_0 = \langle s_0, \Sigma_0 \rangle$, $\bot \notin \hat{\Phi}(p, \sigma_0)$ and, φ is known to be true in every state in $\hat{\Phi}(p, \sigma_0)$ or φ is known to be false in every state in $\hat{\Phi}(p, \sigma_0)$.*

We can prove that the following propositions hold for the action theories (D_s, O_s) and (D_g, O_g).

Proposition 2 *For the illness domain,*
- *$(D_s, O_s) \models_{\mathcal{AM}_K}$* **kwhether** *$i = i_j$* **after** *$p_s$ for every $j \in \{1, \ldots, 5\}$ and*
- *$(D_s, O_s) \models_{\mathcal{AM}_K}$* **knows** *$i = 0$* **after** *$[p_s; p_1]$.*

(ii) For the gas domain,
- *$(D_g, O_g) \models_{\mathcal{AM}_K}$* **kwhether** *$gas\_in\_tank = v$* **after** *$[look\_at\_indicator]$ for any value v in $[0, 20]$.*

Intuitively, this proposition says that the plan $p_s; p_1$ discussed earlier can help us figure out the illness of the patient and find the appropriate cure for her. The second item says that executing the action $look\_at\_indicator$ will tell us the current amount of gas in the tank.

3 Approximation of $\models_{\mathcal{AM}_K}$

It is easy to see that to compute $\models_{\mathcal{AM}_K}$, we will need to deal with more than 2^{2^n} possible c-states where n is the number of fluents and for every fluent f, $|dom(f)| \geq 2$. Furthermore, when the domain of a fluent is infinite, it will be impossible to enumerate all possible c-states. For this reason, we devise an approximation of $\models_{\mathcal{AM}_K}$ for a subclass of $\mathcal{AM}_K$ action theories. Our motivation behind this task is to use this approximation in a logic programming based conditional planner similar to the planner in [11] to deal with multi-valued fluents. We call this newly defined semantics the 0-approximation of $\models_{\mathcal{AM}_K}$ and denote it by $\models^0_{\mathcal{AM}_K}$. The action theories considered in this section satisfy the following conditions:

- For every fluent f, $dom(f)$ is discrete and finite;
- Every m-atom$f \in V$ occurs in propositions (2) and (5) is of the form $f = v$ or $f \neq v$;
- For every proposition of the form (3), S is finite and $|V_j| = 1$ for every $j \in S$.

Observe that under this restriction, an action – when executed – will provide an agent with (i) the exact information about the value of a fluent; or (ii) additional information that helps the agent narrowing down the set of possible value of a fluent. This leads us to define the notion of an *a-state* to represent the knowledge of an agent which consists of two sets of assignments. The first one records what is known to be true to the agent and the second one records what is known to be false to the agent. We begin with some notations that will be used in formalizing the 0-approximation.

For an assignment s and a fluent f, let $s(f) = \{v \mid (f = v) \in s\}$. An *a-assignment* δ is a pair $\langle T, F \rangle$ where T and F are two sets of assignments satisfying four conditions: (i) for each fluent f, there exists at most one assignment of the form $f = v$ in T, i.e., $\mid T(f) \mid \leq 1$; (ii) for each fluent f, $F(f)$ is strictly a subset of $dom(f)$, i.e., $\mid F(f) \mid < \mid dom(f) \mid$; (iii) if $f = v \in T$ then F contains the set $\{f = v' \mid v' \in dom(f) \setminus \{v\}\}$; and (iv) if F contains the set $\{f = v' \mid v' \in dom(f) \setminus \{v\}\}$ for some $v \in dom(f)$ then T contains the assignment $f = v$. Intuitively, T represents the knowledge that the agent knows to be true and F represents the knowledge that the agent knows to be false.

According to this definition, an a-assignment $\delta = \langle T, F \rangle$ satisfies the following condition $T(f) \cap F(f) = \emptyset$. An m-atom $f \in V$ is said to *hold* in an a-assignment $\delta = \langle T, F \rangle$ if $dom(f) \setminus F(f) \subseteq V$. $f \in V$ *does not hold* in δ if $V \subseteq F(f)$. An m-atom$f \in V$ is said to *possibly hold* in δ if $V \setminus F(f) \neq \emptyset$. Notice that if an m-atom holds in δ then it also possibly holds in δ. Fluent formulas[4] are said to hold (resp. possibly hold) in δ if all of its conjuncts hold (resp. possibly hold) in δ. An action a is executable in δ if there exists a proposition of the form (1) whose action is a and φ holds in δ.

An a-assignment $\delta_1 = \langle T_1, F_1 \rangle$ extends an a-assignment $\delta = \langle T_2, F_2 \rangle$, denoted by $\delta_2 \preceq \delta_1$, if $T_2 \subseteq T_1$ and $F_2 \subseteq F_1$. In addition, if $T_2 \subset T_1$ or $F_2 \subset F_1$, we say that δ_1 is a *proper extension* of δ_2 and denote it by $\delta_2 \prec \delta_1$. δ_1 minimally extends δ_2 if there exists no proper extension δ_3 of δ_2 such that $\delta_3 \prec \delta_1$.

An a-assignment δ *satisfies* a static causal law "$f \in V$ **if** φ" if it is the case that φ holds in δ implies that $f \in V$ holds in δ. An a-assignment δ is an *a-state* of a domain description D if it satisfies all the static causal laws in D. For an a-assignment γ and a set of static causal laws R, by $Cn(\gamma \cup R)$ we denote the set of all a-states satisfying R that minimally extends γ and contains γ. Notice that $Cn(\gamma \cup R)$ may contain more than one element or may be empty.

Before providing the formal definition for the 0-transition function Φ_0, let us introduce some notations below.

- The set of positive effects of a, denoted by $e^+_a(\delta)$, where
$$\begin{aligned} e^+_a(\delta) = \{f = v \mid \exists [a \textbf{ causes } f = v \textbf{ if } \varphi] \in D,\ \varphi \text{ holds in } \delta\} \cup \\ \{f = v \mid \forall v' \in dom(f) \setminus \{v\} \\ \exists [a \textbf{ causes } f \neq v \textbf{ if } \varphi] \in D,\ \varphi \text{ holds in } \delta\}; \end{aligned}$$
- The set of negative effects of a, denoted by $e^-_a(\delta)$, where
$$\begin{aligned} e^-_a(\delta) = \{f = v \mid \exists [a \textbf{ causes } f \neq v \textbf{ if } \varphi] \in D,\ \varphi \text{ holds in } \delta\} \cup \\ \{f = v' \mid \exists [a \textbf{ causes } f = v \textbf{ if } \varphi] \in D,\ \varphi \text{ holds in } \delta, \\ v' \in dom(f) \setminus \{v\}\}; \end{aligned}$$
- The set of possible positive effects of a, denoted by $F^+_a(\delta)$, where

[4] Recall that every fluent formula can be represented as a conjunction of m-atoms.

$F_a^+(\delta) = \{f = v \mid \exists[a$ **causes** $f = v$ **if** $\varphi] \in D,\ \varphi$ possibly holds in $\delta\} \cup \{f = v \mid \forall v' \in dom(f) \setminus \{v\}$ $\exists[a$ **causes** $f \neq v$ **if** $\varphi] \in D,\ \varphi$ possibly holds in $\delta\}$;

- The set of possible negative effects of a, denoted by $F_a^-(\delta)$, where $F_a^-(\delta) = \{f = v \mid \exists[a$ **causes** $f \neq v$ **if** $\varphi] \in D,\ \varphi$ possibly holds in $\delta\} \cup \{f = v' \mid \exists[a$ **causes** $f = v$ **if** $\varphi] \in D$ and φ possibly holds in $\delta, v' \in dom(f) \setminus \{v\}\}$.

Let $T_a = (T \cup e_a^+(\delta)) \setminus F_a^-(\delta)$ and $F_a = (F \cup e_a^-(\delta)) \setminus F_a^+(\delta)$. Intuitively, T_a and F_a is the set of assignments that should be true and false, respectively, in any a-state resulting from the execution of a in δ. The result function $Res_0(a, \delta)$ that maps pairs of actions and a-states into sets of a-states is defined as follows.

- For a non-sensing action a[5],

$$Res_0(a, \delta) = Cn(\langle T_a, F_a\rangle \cup R_D).$$

- For a sensing action a with "a **determines** f" in D, $Res_0(a, \delta) = \bigcup_{v \in dom(f)} Cn(X(f, v, \delta), R_D)$ where $X(f, v, \delta) = \langle T \cup \{f{=}v\}, F \cup \{f{=}v' \mid v' \in dom(f) \setminus \{v\}\}\rangle$.

We are now ready to define the transition function.

Definition 5 (0-Transition Function) *Given a domain description D, an action a and a state δ, the 0-transition function $\Phi_0(a, \delta)$ that maps pairs of actions and a-states into sets of a-states is defined as follows:*

- *if a is not executable in δ then $\Phi_0(a, \delta) = \{\perp\}$, where $\perp$ denotes an undefined state,*
- *if a is executable in δ then $\Phi_0(a, \delta) = Res_0(a, \delta)$ if $Res_0(a, \delta) \neq \emptyset$ and $\Phi_0(a, \delta) = \{\perp\}$ if $Res_0(a, \delta) = \emptyset$.*

The 0-extended transition function $\hat{\Phi}_0$ that maps pairs of conditional plans and states into sets of states is defined similarly to the extended transition function $\hat{\Phi}$. To save space, we omit the precise definition of $\hat{\Phi}_0$ in this abstract.

For an action theory (D, O), let $\delta_0 = \{\langle T_0, F_0\rangle$ where $T_0 {=} \{f{=}v \mid \exists[$ **initially** $f{=}v] \in O\} \cup \{f{=}v \mid \forall v'.[v' \in dom(f) \setminus \{v\},$ **initially** $f \neq v' \in O]\}$ and $F_0 = \{f = v \mid \exists[$ **initially** $f \neq v] \in O\} \cup (\bigcup_{f=v \in T_0} \{f = v' \mid v' \in dom(f) \setminus \{v\}\}$. An a-state δ is called an *initial a-state* of (D, O) if $\delta \in Cn(\delta_0 \cup R_D)$. Apparently, $\delta_s = \langle\{\neg bsd, \neg tcd, \neg dead\}, \{i{=}n, bsd, tcd, dead\}\rangle$ is an initial a-state of (D_s, O_s). The 0-entailment relation is defined next.

Definition 6 (0-Entailment) *An action theory (D, O) entails a query* $\mathbf{knows}\ \varphi$ **after** *p w.r.t the 0-approximation, denoted by $(D, O) \models^0_{\mathcal{AM}_K}$* $\mathbf{knows}\ \varphi$ **after** *p, if for every initial a-state δ_0 of (D, O), $\perp \notin \hat{\Phi}(p, \delta_0)$ and φ holds in every a-state $\delta \in \hat{\Phi}(p, \delta_0)$.*

Likewise, (D, O) entails a query $\mathbf{kwhether}\ \varphi$ **after** *p w.r.t the 0-approximation, denoted by $(D, O) \models_{\mathcal{AM}_K}$* $\mathbf{kwhether}\ \varphi$ **after** *p, if for every initial a-state δ_0 of (D, O), $\perp \notin \hat{\Phi}(p, \delta_0)$ and φ is known every a-state $\delta \in \hat{\Phi}(p, \delta_0)$.*

It is easy to check that the following proposition holds.

Proposition 3 *For the illness domain (D_s, O_s), we have*

- $(D_s, O_s) \models^0_{\mathcal{AM}_K}$ $\mathbf{kwhether}\ i = i_j$ **after** p_s *for every* $j \in \{1, \ldots, 5\}$.
- $(D_s, O_s) \models^0_{\mathcal{AM}_K}$ **knows** $i = 0$ **after** $p_s; p_1$

In the next proposition, we prove that the soundness of the 0-approximation w.r.t the semantics of $\mathcal{AM}_K$.

Proposition 4 *For an action theory (D, O), a conditional plan p and a fluent formula φ, $(D, O) \models^0_{\mathcal{AM}_K}$* $\mathbf{knows}\ \varphi$ **after** *p implies $(D, O) \models_{\mathcal{AM}_K}$* **knows** φ **after** *p.*

[5] R_D is the set of static causal laws in D.

4 Discussion and Future Work

In this paper, we concentrate on the development of a high-level action description language with sensing actions, nondeterministic actions, and multi-valued fluents. The main contribution of this paper is the language $\mathcal{AM}_K$ with a transition function based semantics, that naturally extends the language for sensing actions $\mathcal{A}_K$ in [3]. We have taken the first step towards using $\mathcal{AM}_K$ in the development of a conditional planner for domains with these features. We achieve that by developing an approximation of $\models_{\mathcal{AM}_K}$, the 0-approximation, and use this approximation in building a conditional planner[6]. We also prove the soundness of this approximation.

In term of related works, we are not aware of any action description languages that allow sensing actions, nondeterministic actions, and multi-valued fluents. We were able to show that every definite action theory with discrete and finite multi-valued fluents, where every atom appearing in the theory is of the form $f = v$ or $f \neq v$, can be translated into an equivalent theory in the language $\mathcal{A}_K$ [3]. One of the most closely related works in this direction is that of [6, 5] where nonpropositional fluents and nondeterministic actions are considered. We are currently working on a comprehensive comparison between $\mathcal{C}^+$ and $\mathcal{AM}$ ($\mathcal{AM}_K$ with complete information and without sensing actions).

Much work need to be done in using $\mathcal{AM}_K$ in its full generality to develop 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 levels of nested **if-then-else**. The first problem arises since listing all possible values of a fluent might be time-consuming (or impossible) task (for example, the domain is huge, say an integer less than 10 million?). Thus, a more compact representation of fluents's domains would be more appropriate. The second problem arises because a sensing action may create several branches at the same time (e.g., list all the files (action `ls`, [7]) and do something with each file depending on its name.)

REFERENCES

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

[2] C. Baral, M. Gelfond, and A. Provetti, 'Representing Actions: Laws, Observations and Hypothesis', *JLP*, **31(1-3)**, 201–243, (May 1997).

[3] C. Baral et al., 'Formulating diagnostic problem solving using an action language with narratives and sensing', in *KR'2000*, pp. 311–322.

[4] M. Gelfond and V. Lifschitz, 'Representing actions and change by logic programs', *JLP*, **17**(2,3,4), 301–323, (1993).

[5] E. Giunchiglia, G. Kartha, and V. Lifschitz, 'Representing action: indeterminacy and ramifications', *AIJ*, **95**, 409–443, (1997).

[6] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, 'Nonmonotonic causal theories', *AIJ*. to appear.

[7] K. Golden and D. Weld, 'Representing sensing actions: the middle ground revisited', in *KR 96*, pp. 174–185, (1996).

[8] H. Levesque, 'What is planning in the presence of sensing?', in *AAAI-96*, 1139–1146.

[9] J. Lobo, S. Taylor, and G. Mendez, 'Adding knowledge to the action description language $\mathcal{A}$', in *AAAI 97*, pp. 454–459, (1997).

[10] T.C. Son and C. Baral, 'Formalizing sensing actions - a transition function based approach', *AIJ*, **125**(1-2), 19–91, (January 2001).

[11] T.C. Son, P.H. Tu, C. Baral, 'Planning with Sensing Actions and Incomplete Information using Logic Programming', LPNMR-04, 261–274.

[12] D. Weld, C. Anderson, and D. Smith, 'Extending graphplan to handle uncertainty and sensing actions', in *Proceedings of AAAI 98*, (1998).

[6] The conditional planner that can work with finite $\mathcal{AM}_K$-domains is available from our website. To save space, we omit the discussion on the planner in this extended abstract. The website also contains testing results of our planner against some other conditional planners that are capable of dealing with sensing actions and multi-valued fluents.