Approximate reasoning about actions in presence of sensing and incomplete information

Chitta Baral and **Tran Cao Son**
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968
{chitta,tson}@cs.utep.edu

Abstract

Sensing actions are important for planning with incomplete information. A solution for the frame problem for sensing actions was proposed by Scherl and Levesque. They adapt the possible world model of knowledge to situation calculus. In this paper we propose a high level language in the spirit of the language $\mathcal{A}$, that allows sensing actions. We then present two approximation semantics of this language and their translation to logic programs. Unlike, $\mathcal{A}$, where states are two valued interpretations, and unlike the approach in Scherl and Levesque where states are Kripke models, in our approach states are three valued interpretations.

1 Introduction and Motivation

Sensing actions are important for planning with incomplete information [2, 5, 8, 7]. Consider a robot which is asked to get milk. The robot does not know if there is milk in the fridge or not, and of course would prefer to get the milk from the fridge rather than from the neighborhood 24 hr store. A conditional plan of the robot to get milk would be to go to the fridge, open it and sense (or look) if there is milk in it and then bring the milk if it is there; otherwise it has to go to the store and get the milk from there. This conditional plan involves sensing and to generate such a plan the robot has to be able to reason about its sensing actions.

Scherl and Levesque [12] propose an elegant solution to the frame problem in presence of sensing actions. (Sensing actions are also referred to as knowledge producing actions.) To formalize knowledge they adapt the standard possible world model of knowledge to situation calculus. Although elegant, their formulation would require a planner based on it to keep track of the accessibility relations and thus increase the complexity of the planner. (Note that if there are n fluents originally then the number of possible worlds may be 2^n – the number of fluent interpretations – and then the accessibility relation would be a subset of 2^{n+1} elements. So keeping track of the change in the accessibility relation would be expensive.) Moreover, in presence of temporally extended goals [1], their formulation would result in having two kind of modal operators, a knowledge operator, and an operator for temporal goals.

We take a simpler approach using Lukasiewicz's three valued logic [10]. In Lukasiewicz's logic a proposition may have three truth values: $true$ (T), $false$ (F) or $unknown(U)$. For any proposition p, if p is $true$, then $K(p)$ will be $true$, if p is $false$, then $K(p)$ will be $false$ (but $K(\neg p)$ will be true), and if p is unknown, then both $K(p)$ and $K(\neg p)$ will be false. The truth table for the other connectives and operators are defined as follows:

$\wedge$	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

$\vee$	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

$\neg$	
T	F
F	T
U	U

Although, at first glance, this simple formulation seems an intuitive representation of knowledge, there is one problem lurking underneath. In this formulation $K(p \vee \neg p)$ has the truth value $false$, when p is $unknown$, which seems unintuitive. Although, this was pointed out to Lukasiewicz, he stuck to his convictions in his lifetime, and argued that for some modal operators his formulation makes sense.

In our approach a state of the world is represented as a 3-valued interpretation of the fluents in the world. We denote it by a pair $\langle T, F \rangle$, where T is the set of fluents that have truth value $true$, F is the set of fluents that have truth value $false$, and T and F are disjoint. The rest of the fluents in the world have truth value $unknown$. This is more general than states in $\mathcal{A}$, which are 2-valued interpretation of the fluents in the world. But, states in the formulation of Scherl and Levesque [12] are Kripke models and in the formulation of Lobo, Taylor and Mendez [9] are sets of 3-valued interpretations of fluents in the world. Our states being simpler than the last two approaches, result in loss of expressibility. *Nevertheless, we present two approximate reasoning methods to reason about actions in presence of sensing and incomplete information.*

Our most approximate approach, which we refer to as the, *0-semantics*, directly corresponds to the logic of Lukasiewicz. We justify the apparent fallacy of reasoning $K(p \vee \neg p)$ to be $false$ when p is $unknown$, by saying that reasoner (or the robot) does not have the resources (time) to adequately reason (by cases) and makes a hasty decision.

Our next approximate approach, which we refer to as the, *1-semantics*, avoids the above mentioned fallacy by treating the truth value U in a different way. A proposition with truth value U, can either be $true$ or $false$. I.e., with additional knowledge it may turn out to be $true$ or it may turn out to be $false$. Consider a formula $F(P, P_u)$ composed of two sets of propositions P, and P_u. Suppose the truth values of the propositions in P are either $true$ and $false$ and the truth value of the propositions in P_u are $unknown$. Let

$I(P_u)$ be the set of all possible two valued interpretation of P_u. We define the truth value of the formula F as the truth value of

$$\wedge_{I \in I(P_u)} F(P, I)$$

Based on this definition, $K(p \vee \neg p)$ has the truth value *true*, when p is *unknown*.

In the next section we present the syntax of a high level language, in the spirit of $\mathcal{A}$ [4], that allows specification of effects of sensing actions. In later sections we discuss the 1-semantics and the 0-semantics of this language and present translation to disjunctive logic programming. We also discuss when our approximate semantics fail to reason intuitively.

2 A high level language with sensing actions: Syntax

Our language $\mathcal{A}_K$ has two distinct parts. One that is used to define descriptions in the language and another that is used to define queries.

A *description* of an action domain in the language $\mathcal{A}_K$ consists of "propositions" of three kinds. A "v-proposition" specifies the value of a fluent in the initial situation. An "ef-proposition" describes the effect of an action on the truth value of a fluent. A "k-proposition" describes the effect of an action on the knowledge about a fluent's truth value.

We begin with two disjoint nonempty sets of symbols, called *fluent names* and *action names*. A *fluent literal* is either a fluent name or a fluent name preceded by $\neg$. For a fluent literal l, by $\bar{l}$ we denote the fluent literal $\neg l$, and we shorten $\neg\neg l$ to l.

A *v-proposition* is an expression of the form

$$\textbf{initially} \quad f \tag{1}$$

An *ef-proposition* is an expression of the form

$$a \ \textbf{causes} \quad f \ \textbf{if} \quad p_1, \ldots, p_n \tag{2}$$

where a is an action name, and each of $f, p_1, \ldots, p_n$ $(n \geq 0)$ is a fluent literal. The set of fluent literals $\{p_1, \ldots, p_n\}$ is referred to as the *precondition* of the ef-proposition and f is referred to as the *effect* of this ef-proposition. About this proposition we say that it *describes the effect of a on f*, and that $p_1, \ldots, p_n$ are its *preconditions.* If $n = 0$, we will drop **if** and write simply a **causes** f.

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

A *k-proposition* is an expression of the form

$$a \ \textbf{determines} \ f \ \textbf{if} \ q_1, \ldots q_n \tag{3}$$

where a is an action name, and each of f and $q_1, \ldots, q_n$ ($n \geq 0$) is a fluent literal. The set of fluent literals $\{p_1, \ldots, p_n\}$ is referred to as the *precondition* of the k-proposition and f is referred to as the *k-effect* of this k-proposition. About this proposition we say that it stipulates that if a is executed in a situation where $q_1, \ldots, q_n$ are true, then in the resultant state the truth value of f becomes known.

A *proposition* is a v-proposition, ex-proposition, or an k-proposition. A *domain description*, or simply *domain*, is a set of propositions which does not contain contradictory ef-propositions.

Example 1 We have a robot which can perform the actions: *go_to_fridge, look_at_fridge, open_fridge, look_into_fridge, get_milk_from_fridge*, and $get\_milk\_from\_store$. Intuitively, if the robot performs the action go_to_fridge then it will be near the fridge. If it is near the fridge, it can look at the fridge to find out if the fridge is open or not. By performing the action $open\_fridge$, the fluent $fridge\_open$ becomes true. We can formally represent the above (and some additional knowledge) by the following domain description D_1:

go_to_fridge **causes** $near\_fridge$
$look\_at\_fridge$ **determines** $fridge\_open$ **if** $near\_fridge$
$open\_fridge$ **causes** $fridge\_open$ **if** $near\_fridge$
$look\_into\_fridge$ **determines** $milk\_in\_fridge$
  if $fridge\_open, near\_fridge$
$get\_milk\_from\_fridge$ **causes** has_milk
  if $milke\_in\_fridge, fridge\_open, near\_fridge$
$get\_milk\_from\_store$ **causes** has_milk
$get\_milk\_from\_store$ **causes** $\neg near\_fridge$ □

In the presence of incomplete information and knowledge producing actions, we need to extend the normal definition of a plan as a sequence of actions. If we would like to make a plan to have milk w.r.t. the above domain, our plan will not be a sequence of actions, rather it would contain conditional statements. In the following definition we formalize this notion.

Definition 1 Conditional Plan

(i) An empty sequence of action, denoted by [ ], is a conditional plan.

(ii) If a is an action then a is a conditional plan.

(iii) If c_1 and c_2 are conditional plans, then $c_1; c_2$ is a conditional plan.

(iv) If $c_1, \ldots, c_n$ are conditional plans and $p_{i,j}$'s are fluents then the following is a conditional plan. (Such a plan is referred to as a *case plan*).

Case

$$p_{1,1}, \ldots, p_{1,m_1} \rightarrow c_1$$
$$\vdots$$
$$p_{n,1}, \ldots, p_{n,m_n} \rightarrow c_n$$

Endcase

where $\{p_{1,1}, \ldots, p_{1,m_1}\}, \ldots, \{p_{n,1}, \ldots, p_{n,m_n}\}$ are mutually exclusive (but not necessary exhaustive).

(v) Nothing else is a conditional plan.

Example 2 Following is a conditional plan[1] c_1 which will achieve the goal of the robot having milk, w.r.t. the description in D_1.
go_to_fridge;
$look\_at\_fridge$;
Case
$\neg fridge\_open \rightarrow open\_fridge$;
$look\_into\_fridge$;
Case
$milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
$\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
Endcase
$fridge\_open \rightarrow look\_into\_fridge$;
Case
$milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
$\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
Endcase
Endcase □

A query is an expression of the form $\quad f$ **after** $c \qquad (4)$
where f is a fluent and c is a conditional plan.

Example 3 To find out if c_1 indeed achieves the goal of having milk w.r.t. D_1, we need to pose the query $\quad has\_milk$ **after** $c_1 \quad$ to D_1 □

In the following sections we define two approximate semantics of domain descriptions in $\mathcal{A}_k$. In the process we define two entailment relations between domain descriptions and queries of $\mathcal{A}_k$.

3 1-Semantics and its properties

In $\mathcal{A}$ [4] a state is defined as a set of fluents, and corresponds to a state of the world at a particular moment. Here, a state corresponds to the state of the robots knowledge at a particular moment. Hence, it may be incomplete. Formally, a *state* is a pair of disjoint sets of fluent names. Given a fluent name f and a state $\sigma = \langle T, F \rangle$, we say that f *holds* in σ (we sometime denote it by $\sigma \models f$) if $f \in T$; $\neg f$ *holds* in σ if $f \in F$; otherwise both f and $\neg f$ are said to be *unknown* in σ. A *transition function* is a mapping Φ from the set of pairs (a, σ), where a is an action name and σ is a state, into the power set of states. Intuitively, $\Phi(a, \sigma)$ encodes the set of states the robot may reach after executing an actions a in a state σ. A *state* is said to be *complete* if $T \cup F$ is the set of all the fluents in the language. A *structure* is a pair (σ_0, Φ), where σ_0 is a state (the *initial state* of the structure), and Φ is a transition function.

[1] This plan can be made simpler and compact. The reason we present this particular plan without simplifying it is because it was automatically generated by our planner.

Before we define when a structure is a model of a domain description, we have the following notations:

- $E_a^+(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name and there exists an ef-proposition in our domain description whose action is a, effect is f, and whose preconditions are satisfied in $\langle T, F\rangle\}$.

- $E_a^-(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name and there exists an ef-proposition in our domain description whose action is a, effect is $\neg f$, and whose preconditions are satisfied in $\langle T, F\rangle\}$.

- For complete states $\langle T, F\rangle$, if for any action a, $E_a^+(\langle T, F\rangle) \cap E_a^-(\langle T, F\rangle) = \emptyset$, we say $Res(a, \langle T, F\rangle) = \langle T \cup E_a^+(\langle T, F\rangle) \setminus E_a^-(\langle T, F\rangle), F \cup E_a^-(\langle T, F\rangle) \setminus E_a^+(\langle T, F\rangle)$, otherwise we say $Res(a, \langle T, F\rangle)$ is undefined. (Note that, since we restrict our domain description to only contain non-contradicting ef-propositions, the function Res is always defined.)

- Let $\sigma_1 = \langle T_1, F_1\rangle$, and $\sigma_2 = \langle T_2, F_2\rangle$ be two states. We say the state $\langle T_1, F_1\rangle$ extends the state $\langle T_2, F_2\rangle$ if $T_2 \subseteq T_1$ and $F_2 \subseteq F_1$. $\sigma_1 \cap \sigma_2$ is defined as $\langle T_1 \cap T_2, F_1 \cap F_2\rangle$ and $\sigma_1 \setminus \sigma_2$ denotes the set $(T_1 \setminus T_2) \cup (F_1 \setminus F_2)$. For a set of fluent names X we write $X \setminus \langle T, F\rangle = X \setminus (T \cup F)$.

- For an incomplete state $\langle T, F\rangle$, $\mathcal{R}es(a, \langle T, F\rangle)$ denotes the state $\bigcap_{\sigma \in S(\langle T, F\rangle)} Res(a, \sigma)$ where $S(\langle T, F\rangle)$ is the collection of all complete states that extend $\langle T, F\rangle$.

- For a complete state $\langle T, F\rangle$, $K(a, \langle T, F\rangle)$ denotes the set $\{f : f$ is a fluent name and there exists a k-proposition whose action is a, whose k-effect is either f or $\neg f$, and whose preconditions are satisfied in $\langle T, F\rangle\}$.

- For an incomplete state $\langle T, F\rangle$, $\mathcal{K}(a, \langle T, F\rangle)$ denotes the set $\bigcap_{\sigma \in S(\langle T, F\rangle)} K(a, \sigma)$

Example 4 Consider the domain description D_1 from Example 1. The effects of actions, the result functions, and the extended result functions for some actions in D_1 are given below.

$E^+_{go\_to\_fridge}(\langle T, F\rangle) = \{near\_fridge\}$ and $E^-_{go\_to\_fridge}(\langle T, F\rangle) = \emptyset$;
$E^+_{look\_at\_fridge}(\langle T, F\rangle) = E^-_{look\_at\_fridge}(\langle T, F\rangle) = \emptyset$.
$Res(go\_to\_fridge, \langle T, F\rangle) = \langle T \cup \{near\_fridge\}, F \setminus \{near\_fridge\}\rangle$
$Res(look\_at\_fridge, \langle T, F\rangle) = \langle T, F\rangle$
$\mathcal{R}es(go\_to\_fridge, \langle T, F\rangle) = \langle T \cup \{near\_fridge\}, F \setminus \{near\_fridge\}\rangle$
$\mathcal{R}es(look\_at\_fridge, \langle T, F\rangle) = \langle T, F\rangle$ □

Definition 2 A transition function Φ of a domain description D is a mapping of a pair of action a and state σ into a set of states, denoted by $\Phi(a, \sigma)$, where

(i) for a complete state σ, $\Phi(a, \sigma) = \{Res(a, \sigma)\}$, and

(ii) for an incomplete state σ, $\Phi(a, \sigma) = \{\sigma' \mid \sigma'$ extends $\mathcal{R}es(a, \sigma)$ and $\sigma' \setminus \mathcal{R}es(a, \sigma) = \mathcal{K}(a, \sigma) \setminus \mathcal{R}es(a, \sigma)\}$. $\Box$

Note the difference between the definition of transition function above and the definition of transition function in $\mathcal{A}$, where we do not have any sensing actions. In $\mathcal{A}$, the transition function is defined in terms of the Res function. Here, $\mathcal{R}es(a, \sigma)$, only specifies what is known to be true and what is known to be false in all states that may be reached by executing a in the state σ. The transition function Φ takes into account $\mathcal{R}es$ and $\mathcal{K}$ to define the set of states that may be reached by executing a in σ.

Example 5 The domain description D_1 has a unique model (σ_0, Φ_1), where $\sigma_0 = \langle \emptyset, \emptyset \rangle$, and Φ_1 is specified for every possible pair of states $\langle T, F \rangle$ and actions a. For example,
$\Phi_1(go\_to\_fridge, \langle T, F \rangle) = \mathcal{R}es(go\_to\_fridge, \langle T, F \rangle)$ (see Example 4) and

$$\Phi_1(look\_at\_fridge, \langle T, F \rangle) = \begin{cases} \{\langle T \cup \{fridge\_open\}, F \rangle, \\ \ \langle T, F \cup \{fridge\_open\} \rangle\} \\ \qquad \text{if } fridge\_open \notin T \cup F \\ \qquad \text{and } near\_fridge \in T \\ \{\langle T, F \rangle\} \qquad \text{otherwise} \end{cases}$$

In general, a domain description D might have many transition functions but if the actions in D are deterministic then D has a unique transition function as the function Res is also unique.

Definition 3 Given a domain description D, we say a structure (σ_0, Φ) is a model of D if

(i) for any fluent literal f, f is true w.r.t. σ_0 iff **initially** $f \in D$,

(ii) Φ is a transition function of D. $\Box$

Note that if a is a non-sensing action (i.e., there is no k-proposition whose action is a) then $\mathcal{K}(a, \langle T, F \rangle) = \emptyset$ for every state $\langle T, F \rangle$. Hence, $\Phi(a, \langle T, F \rangle) = \{\mathcal{R}es(a, \langle T, F \rangle)\}$ for every state $\langle T, F \rangle$.

Definition 4 For any transition function Φ, we define an extended transition function $\hat{\Phi}$ from conditional plans, and sets of states to power set of states, in the following way:

$$\hat{\Phi}(a, S) = \bigcup_{\sigma \in S} \Phi(a, \sigma)$$

(ii) $\hat{\Phi}(c_1; c_2, S) = \hat{\Phi}(c_2, \hat{\Phi}(c_1, S))$

(iii) Let c be a conditional case plan; then $\hat{\Phi}(c, S)$ is defined as the set

$$\bigcup_{i \leq n} (\bigcup_{\sigma \in S, \sigma \models p_{i1}, \ldots p_{im_i}} \hat{\Phi}(c_i, \sigma))$$

Example 6 Consider the conditional plan c_1 in Example 2 and the transition function Φ_1 in Example 5. We can easily compute that
$\hat{\Phi}_1(c_1, \langle\emptyset, \emptyset\rangle) = \{\langle\{fridge\_open, near\_fridge, milk\_in\_fridge, has\_mik\}, \emptyset\rangle,$
$\langle\{fridge\_open, has\_milk\}, \{near\_fridge, milk\_in\_fridge\}\rangle\}.$

Definition 5 Let $M = (\sigma_0, \Phi)$ be a model of a domain description D. We say the query f **after** c is true w.r.t. M, if f is true in all the states in $\hat{\Phi}(c, \sigma_0)$. We say $D \models f$ **after** c, if the query f **after** c is true w.r.t. all models of D. □

Proposition 1 $D_1 \models has\_milk$ **after** c_1

Proof : It is easy to see that D_1 has a unique model of the form $M = (\langle\emptyset, \emptyset\rangle, \Phi_1)$ where Φ_1 is computed in Example 5. Since
$\hat{\Phi}_1(c_1, \langle\emptyset, \emptyset\rangle) = \{\langle\{near\_fridge, fridge\_open, milk\_in\_fridge, has\_milk\}, \emptyset\rangle,$
$\langle\{fridge\_open, has\_milk\}, \{near\_fridge, milk\_in\_fridge\}\rangle\},$
has_milk is true in every state of $\hat{\Phi}_1(c_1, \langle\emptyset, \emptyset\rangle)$.
Hence, $D_1 \models has\_milk$ **after** c_1. □

3.1 Translation to Disjunctive Logic Programming

In this section we present a translation from a domain description D in $\mathcal{A}_K$ into a disjunctive logic program πD and show that the translation (using answer semantics [3]) is sound w.r.t the entailment relationship in D.

The translation πD of a domain description D, uses variables of three sorts: *situation* variables $S, S', \ldots$, *fluent* variables $F, F', \ldots$, and *action* variables $A, A', \ldots$. Lower case letters are used to denoted constants of the same sort as its upper case counterpart.

In the following we write $R\_holds(L, S)$ $(Holds(L, S))$ to represent $r\_holds(L, S)$ $(holds(L, S))$ if L is a fluent name and $\neg r\_holds(\bar{L}, S)$ $(\neg holds(\bar{L}, S))$ if L is a negative fluent literal where $\bar{L}$ denotes the complementary literal of L. Basically, $holds(F, S)$ $(\neg holds(F, S))$ means that the fluent F is true (false) in the state S and $r\_holds(F, S)$ $(\neg r\_holds(F, S))$ means that the robot knows that the fluent F is true (false) in the state S.

Given a domain description D in $\mathcal{A}_K$ the disjunctive logic program πD corresponding to D will contain following rules.
(i) In the world fluents are either true or false. Hence, for every fluent f in D

$$holds(f, s_0) \vee \neg holds(f, s_0) \leftarrow \tag{5}$$

is a rule in πD.
(ii) The robot knows only what is correct:

$$Holds(F, S) \leftarrow R\_holds(F, S) \tag{6}$$

(iii) For every v-proposition **initially** f

$$R\_holds(f, s_0). \tag{7}$$

is a rule in πD.
(iv) For every ef-proposition a **causes** f **if** $p_1, \ldots, p_n$, the following rules are added to πD.

$$\begin{array}{lcl} R\_holds(f, Res(a, S)) & \leftarrow & Holds(p_1, S), \ldots, Holds(p_n, S) \\ ab(\bar{f}, a, S) & \leftarrow & Holds(p_1, S), \ldots, Holds(p_n, S) \end{array} \tag{8}$$

Intuitively, the above rules encode what the robot thinks is true in the state $Res(a, S)$. But, it tries to reason by cases about the preconditions. For that reason, we have $Holds$, instead of R_holds in the body of the rules.
(v) The inertia axioms

$$\begin{array}{lcl} R\_holds(F, Res(A, S)) & \leftarrow & R\_holds(F, S),\ not\ ab(F, A, S) \\ Holds(F, Res(A, S)) & \leftarrow & Holds(F, S),\ not\ ab(F, A, S) \end{array} \tag{9}$$

(vi) For every k-proposition a **determines** f **if** $p_1, \ldots, p_n$, the program πD will contain the following rule.

$$\begin{array}{lcl} r\_holds(f, Res(a, S)) \vee \neg r\_holds(f, Res(a, S)) & \leftarrow & Holds(p_1, S), \ldots, \\ & & \quad Holds(p_n, S) \end{array} \tag{10}$$

We prove the consistency and soundness of πD in the next propositions.

Proposition 2 (Consistency of πD) Let D be a consistent domain description. Then, the program πD is consistent.

Proposition 3 (Soundness of πD) Let D be a consistent domain description and s_0 be its initial state. Then, for every fluent name f and action a in D, if $\pi D \models r\_holds(f, res(a, s_0))$ (resp. $\pi D \models \neg r\_holds(f, res(a, s_0))$) then $D \models f$ **after** a (resp. $D \models \neg f$ **after** a).

The following example shows that the logic programming translation does not capture the 1-semantics of $\mathcal{A}_K$ completely.

Example 7 Consider the domain description D_2 consisting of the following ef-proposition: a **causes** f **if** $\neg f$.

Suppose that the initial state of D_2 is $s_0 = \langle \emptyset, \emptyset \rangle$. There are two complete extensions of s_0: $s_1 = \langle \{f\}, \emptyset \rangle$ and $s_2 = \langle \emptyset, \{f\} \rangle$. Since $Res(a, s_1) = Res(a, s_2) = s_1$, $\mathcal{R}es(a, s_0) = \langle \{f\}, \emptyset \rangle$. It is easy to see that D has only one model Φ in which $\Phi(a, s_0) = \langle \{f\}, \emptyset \rangle$. Therefore, $D_2 \models f$ **after** a.

On the other hand, the unique answer set of πD_2 is $\emptyset$. Hence, $\pi D_2 \not\models r\_holds(f, res(a, s_0))$. $\square$

We now present an even weaker semantics of $\mathcal{A}_K$, which we call the 0-semantics, that more closely follows Lukasiewicz's logic. We show that the translation π is stronger than this weaker semantics. *One advantage of the 0-semantics is that we have a sound and complete translation of it to disjunctive logic programming.*

4 0-semantics of $\mathcal{A}_K$

Let D be a domain description, $\langle T, F\rangle$ be a state, and f be a fluent name in D. f (resp. $\neg f$) is said to be *possibly correct* in $\langle T, F\rangle$ iff $f \notin F$ (resp. $f \notin T$). A set of fluents $\{f_1, \ldots, f_n\}$ is *possibly correct* in $\langle T, F\rangle$ iff for all i, f_i is possibly correct in $\langle T, F\rangle$. We define,

- $e_a^+(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name and there exists an ef-proposition in our domain description whose action is a, effect is f, and whose preconditions are satisfied in $\langle T, F\rangle\}$.
- $e_a^-(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name and there exists an ef-proposition in our domain description whose action is a, effect is $\neg f$, and whose preconditions are satisfied in $\langle T, F\rangle\}$.
- $F_a^+(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name and there exists an ef-proposition in our domain description whose action is a, effect is f, and whose preconditions are possibly correct in $\langle T, F\rangle\}$.
- $F_a^-(\langle T, F\rangle)$ is defined similarly.
- $k_a(\langle T, F\rangle)$ denotes the set $\{f \mid f$ is a fluent name there exists an k-proposition a **determines** f $(or\ \neg f)$ **if** $p_1, \ldots, p_n$ in our domain description whose preconditions are satisfied in $\langle T, F\rangle\}$.

It is easy to see that if D does not contain contradictory ef-propositions then $e_a^+(\langle T, F\rangle) \cap F_a^-(\langle T, F\rangle) = \emptyset$ and $e_a^-(\langle T, F\rangle) \cap F_a^+(\langle T, F\rangle) = \emptyset$. The result function is then defined as $Res_W(a, \langle T, F\rangle) = \langle T \cup e_a^+ \setminus F_a^-, F \cup e_a^- \setminus F_a^+\rangle$. The transition function is then defined as follows.

Definition 6 Given a domain description D, the transition function Φ_W of D is defined by $\Phi_W(a, \langle T, F\rangle) = \{\sigma \mid \sigma$ extends $Res_W(a, \langle T, F\rangle)$ and $\sigma \setminus Res_W(a, \langle T, F\rangle) = k_a(\langle T, F\rangle) \setminus Res_W(a, \langle T, F\rangle)\}$. □

Definition 7 Given a domain description D, we say a structure (σ_0, Φ_W) is a weak-model of D if

(i) for any fluent literal f, f is true w.r.t. σ_0 iff **initially** $f \in D$,

(ii) Φ_W is a transition function of D.

Entailment w.r.t. weak models is referred to as weak entailment and is denoted by $\models_w$. □

The next example shows that the 0-semantics and 1-semantics agrees on the domain D_1.

Proposition 4 $D_1 \models_w$ *has_milk* **after** c_1. □

We now present a logic programming translation π' which exactly captures the weak entailment w.r.t. domain descriptions.

4.1 The translation $\pi' D$

The program $\pi' D$ differs from the program πD in that it only represents and reasons about what the robot knows. The notation $R\_holds(F, S)$ has the same meaning as in πD. $\overline{R\_holds(L, S)}$ stands for $\neg r\_holds(L, S)$ if L is a fluent name and $r\_holds(\bar{L}, S)$ if L is a negative fluent literal. Given a domain description D in $\mathcal{A}_K$ the disjunctive logic program $\pi' D$ corresponding to D contains the following rules:
(i) Initially, the robot knows what is given. Hence, for every v-proposition **initially** f

$$R\_holds(f, s_0). \tag{11}$$

is a rule in $\pi' D$.
(ii) For every ef-proposition a **causes** f **if** $p_1, \ldots, p_n$, the following rules are added to $\pi' D$.

$$\begin{array}{l} R\_holds(f, res(a, S)) \leftarrow R\_holds(p_1, S), \ldots, R\_holds(p_n, S) \\ ab(\bar{f}, a, S) \leftarrow not\ \overline{R\_Holds(p_1, S)}, \ldots, not\ \overline{R\_Holds(p_n, S)} \end{array} \tag{12}$$

(iii) The inertia axioms

$$R\_holds(F, res(A, S)) \leftarrow R\_holds(F, S),\ not\ ab(F, A, S) \tag{13}$$

(iv) For every k-proposition a **determines** f **if** $p_1, \ldots, p_n$, the program $\pi' D$ will contain the following rule.

$$\begin{array}{l} r\_holds(f, res(a, S)) \vee \neg r\_holds(f, res(a, S)) \quad \leftarrow \\ R\_holds(p_1, S), \ldots, R\_holds(p_n, S) \end{array} \tag{14}$$

In the next propositions we state the consistency, and the soundness and completeness of $\pi' D$ with respect to the weak semantics $\models_w$. (We will present the proof in the full version of the paper.)

Proposition 5 (Consistency of $\pi' D$) Let D be a consistent domain description. Then, the program $\pi' D$ is consistent.

Proposition 6 (Soundness and Completeness of $\pi' D$ w.r.t. $\models_w$) Let D be a consistent domain description and s_0 be its initial state s_0. Then, for every fluent name f and action a in D, $\pi' D \models r\_holds(f, res(a, s_0))$ (resp. $\pi' D \models \neg r\_holds(f, res(a, s_0))$) if and only if $D \models_w f$ **after** a
(resp. $D \models_w \neg f$ **after** a).

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

Proposition 7 (Soundness of 0-semantics w.r.t. 1-semantics) For every state $\sigma' \in \Phi_w(a, \sigma)$, there exists a state $\sigma^* \in \Phi(a, \sigma)$ such that $\sigma' \subseteq \sigma^*$.

The next example shows that 1-semantics is more powerful than the 0-semantics.

Example 8 Consider the domain description D_3 with two ef-propositions:

a **causes** f **if** p and a **causes** f **if** $\neg p$.

Suppose that the initial state is $s_0 = \langle \emptyset, \emptyset \rangle$. It is easy to see that in every complete extension of s_0, p is either true or false. Hence, in the 1-semantics, we have $\mathcal{R}es(a, s_0) = \langle \{f\}, \emptyset \rangle$. Furthermore, we can prove that D_3 has only one model Φ (w.r.t. the 1-semantics) where $\Phi(a, s_0) = \langle \{f\}, \emptyset \rangle$. Therefore, $D_3 \models f$ **after** a.

However, in the 0-semantics, we have $e_a^+(s_0) = e_a^-(s_0) = \emptyset$. Hence, $Res_W(a, s_0) = s_0$. It is easy to see that D_3 has a unique model Φ_W (w.r.t. the 0-semantics) where $\Phi_W(a, s_0) = \{s_0\}$. So, $D_3 \not\models_w f$ **after** a.

5 Weakness of the 1-semantics

In the previous example, we showed that the 1-semantics is stronger than the 0-semantics. This is because the 1-semantics reasons by cases, while the 0-semantics does not. On the other hand we have a sound and complete translation of the 0-semantics to disjunctive logic programming, while we only[2] have a sound translation for the 1-semantics. In the next section we discuss why even the 1-semantics is not strong enough to completely capture our intuition.

5.1 Reasoning about preconditions

Suppose we have the k-proposition a **determines** f **if** p and in the initial state p is unknown. In our formulation after executing a our state remains the same. A stronger semantics may reason that either p is true or p is false. If p is true then there would be two sates $\langle \{p, f\}, \emptyset \rangle$, and $\langle \{p\}, \{f\} \rangle$; and if p is false then there would be the state $\langle \emptyset, \{p\} \rangle$. This is what the semantics of [9] does.

We believe this leads to additional complexity and prefer the relative simplicity of our approach. Nevertheless, we can simulate the above reasoning by thinking that the action a actually is sequentially composed of two actions a_1 and a_2, where a_1 **determines** p and a_2 **determines** f **if** p. Then our approach captures the suggested meaning.

5.2 Reasoning about sequences of actions

A more serious weakness of 1-semantics[3] (and consequently also of the 0-semantics) manifests when reasoning about a sequence of actions in presence of completeness.

[2] We have not been able to find a complete translation to disjunctive logic programming. We suspect that no such translation exists, and we need logic programming with epistemic operators for a complete translation.

[3] We thank the anonymous AAAI97 reviewer who pointed this out. It should be also noted that this paper and [9] were independently done around the same time.

Example 9 Consider the following domain description D_4:

a **causes** p **if** r ; a **causes** q **if** $\neg r$

b **causes** f **if** p ; b **causes** f **if** q

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

Note that, the above example does not have any sensing actions, and even the semantics of $\mathcal{A}$ [4] is able to capture the intuitive meaning. But in $\mathcal{A}$, there is no separation between what is true in the world and what is known to the reasoner. *Since we incorporate sensing, our semantics captures what is known to the reasoner.*

For the reasoner to more accurately reason about his beliefs, he needs either carry more information from one state to another, or he needs to reason by cases for multiple levels. The first approach is taken in [9], and [12], where states are sets of 3-valued interpretations, and Kripke models, resp.

We prefer the second approach, and can define n-semantics (for any number n), where the reasoner does reasoning by cases, for n-levels, guaranteeing intuitive reasoning skills (equivalent to [9]) upto sequences of n actions. The reason we prefer this approach is that the reasoner depending on the time it has can choose the appropriate n. This corresponds to the notion of anytime-reasoning in AI. We will further discuss this issue and present the general formulation of n-semantics in the full paper.

6 Properties of 1-semantics and 0-semantics

In this section we prove some general results about 0-semantics and 1-semantics that show their intuitiveness in capturing the meaning of sensing actions. Sherl and Levesque prove similar results for their formalization in [12].

We first show that a knowledge producing action a does not normally (formalized precisely in Proposition 8) change the truth value of a fluent f whose truth value is either $true$ or $false$ in the state before the execution of a. We then show that our formalization ensures that there will be no unwanted knowledge change when actions are executed.

Let a be an action and s be a state in the domain description D. We say that a does not affect a fluent f in a state s if the following condition holds:

(i) For every ef-proposition a **causes** f **if** $p_1, \ldots, p_n$
$\{p_1, \ldots, p_n\}$ *is not possibly correct in* s *and*

(ii) For every k-proposition a **determines** f **if** $p_1, \ldots, p_m$

$\{p_1, \ldots, p_m\}$ *is not possibly correct in* s.

Furthermore, for a fluent f and a state $\langle T, F\rangle$ in D, we define

$$\mathbf{Knows}(f, \langle T, F\rangle) \stackrel{def}{\equiv} f \in T \cup F$$

Intuitively, $\mathbf{Knows}(f, \langle T, F\rangle)$ holds when we know the truth value of the fluent f in the situation s.

Proposition 8 (Knowledge Producing Effects of 1-semantics and 0-semantics) Consider a consistent domain description D with a model (σ_0, Φ) (resp. (σ_0, Φ_W)). Let a be a knowledge producing action and s be a state in D such that there exists no ef-proposition in D whose action is a and whose preconditions are possibly correct in s. Then, if f is true (or false) w.r.t. s then f is true (or false) w.r.t. every state s' in $\Phi(a, s)$ (resp. $\Phi_W(a, s)$).

The above proposition points out the difference between non-deterministic actions (see e.g. [6]) and knowledge producing actions. A knowledge producing action a, that is only supposed to determine the truth value of a fluent f, does not change the truth value of fluent f if its truth value is either $true$ or $false$ before a is executed. On the other hand the truth value of f may change when executing a non-deterministic action that non-deterministically changes the truth value of f.

Proposition 9 (Knowledge Inertial Effects of 1-semantics and 0-semantics) Consider a consistent domain description D with a model (σ_0, Φ) (resp. (σ_0, Φ_W)). Let a be an action and a state s in D. Then, for every fluent f, if $\mathbf{Knows}(f, s)$ $(\neg\mathbf{Knows}(f, s))$ and a does not affect f in s then $\mathbf{Knows}(f, s')$ $(\neg\mathbf{Knows}(f, s))$ for $s' \in \Phi(a, s)$ (resp. $s' \in \Phi_W(a, s)$).

7 Conclusion

In this paper we presented two approximate formulations of sensing actions in terms of 3-valued logic. Our formulation was first at the level of a high level language and later we presented translations to disjunctive logic programs.

Although, translation of action theories to disjunctive logic programs were done for non-sensing action theories, the current translation has the novelty that we have disjunctions about non-initial states. Earlier translations only required disjunctions about the initial state.

Our language $\mathcal{A}_K$ differs from the language GOLOG (see e.g. [11] and the references therein) in several respects. GOLOG is a language for describing and executing complex plans, whereas $\mathcal{A}_K$ is a high-level language for reasoning about sensing and non-sensing actions, and conditional plans. Second, our conditional plan does not allow loop, whereas complex plans in GOLOG are more general and allow loops.

Our use of three valued logic to capture knowledge was driven by the goal of simplifying the process of planning with incomplete information. As it is,

planning with complete information is expensive, and having Kripke-models or set of 3-valued interpretations as states is going to make it much more expensive to plan in presence of incomplete information. Hence we choose approximations that have states as 3-valued interpretations. The only widely available planner that plans with sensing actions seem to agree with us. In [5] the authors say:
"In UWL (and in SADL) individual literals have truth values expressed in a three-valued logic: T, F, U." Moreover, our approximation semantics can be extended to be equivalent to [9] upto any levels of reasoning. In the full paper we will formally state this result.

References

[1] F. Bacchus, F. Kabanza. Planning for Temporally Extended Goals. In *Proc. of AAAI '96*, 1215–1222, 1996.

[2] O. Etzioni et al. An approach to planning with incomplete information. In *Proc. of KR'92*, 115–125, 1992.

[3] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. In *Proceeding of the 7th ICLP, MIT Press*, 579 –597, 1990.

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

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

[6] G.N. Kartha. A mathematical investigation of reasoning about actions. *Ph.D dissertation. University of Texas at Austin*, 1995.

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

[8] H.J. Levesque. What is planning in the presence of sensing. In *Proc. of AAAI'96*, 1139–1146, 1996.

[9] J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language $\mathcal{A}$. To appear in Proc. of AAAI'97.

[10] J. Lukasiewicz. A system of modal logic. *Journal of Computing Systems* 1, 111-149.

[11] Levesque, H., Reiter, R., Lesperance, Y., Lin, F. and Scherl, R. Golog: A Logic programming language for dynamic domains. *Journal of Logic Progamming* 31, 59–84, 1997.

[12] R. Scherl and H.J. Levesque. Knowledge Producing Actions. In *Proc. of KR*, 1139-1146, 1994.

[13] D. Weld. An introduction to least commitment planning. *AI magazine*, 27–61, Winter 94.