Modeling Hybrid Domains Using Process Description Language

Sandeep Chintabathina, Michael Gelfond, and Richard Watson

Texas Tech University
Department of Computer Science
Lubbock, TX, USA
chintaba,mgelfond,rwatson@cs.ttu.edu

Abstract. In previous work, action languages have predominantly been concerned with domains in which values are static unless changed by an action. Real domains, however, often contain values that are in constant change. In this paper we introduce an action language for modeling such hybrid domains called the *process description language*. We discuss the syntax and semantics of the language, model an example using this language, and give a provenly correct translation into answer set programming.

1 Introduction

Designing an intelligent agent capable of reasoning, planning and acting in a changing environment is one of the important research areas in the field of AI. Such an agent should have knowledge about the domain in which it is intended to act and its capabilities and goals.

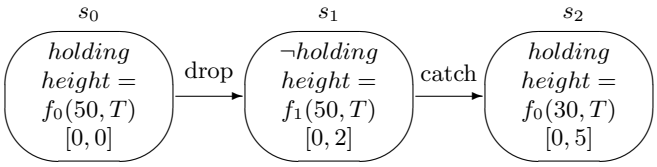
In this paper we are interested in agents which view the world as a dynamical system represented by a transition diagram whose nodes correspond to possible physical states of the world and whose arcs are labeled by actions. A link, (s_0, a, s_1) of a diagram indicates that action a is executable in s_0 and that after the execution of a in s_0 the system may move to state s_1. Various approaches to representation of such diagrams [3, 6, 9] can be classified by languages used for their description. In this paper we will adopt the approach in which the diagrams are represented by action theories - collections of statement in so called action languages specifically designed for this purpose. This approach allows for useful classification of dynamical systems and for the methodology of design and implementation of deliberative agents based on answer set programming.

Most previous work deals with discrete dynamical systems. A state of such a system consists of a set of *fluents* - properties of the domain whose values can only be changed by actions. An example of a fluent would be the position of an electrical switch. The position of the switch can be changed only when an external force causes it to change. Once changed, it stays in that position until it is changed yet again.

In this paper we focus on the design of action languages capable of describing dynamical systems which allow *continuous processes* - properties of an object

whose values change continuously with time. This paper is an evolution of work presented in [18]. Major changes to the language resulted in a significantly simpler and less restrictive syntax and a more precise semantics based on the notion of transition diagrams (following the approach of McCain and Turner [10]). Several other formalisms exist which also allow modeling of continuous processes [4, 15–17]. An advantage of our approach is that, by generalizing McCain and Turner's semantics, it gains the associated benefits (such as the ability to easily represent state constraints). Also, in some of the other formalisms actions have duration, This can lead to problems when such actions overlap. Our actions are instantaneous. This allows us to avoid the problems with overlapping action. Following the approach from [13], an action, A with duration can still be represented using instantaneous actions which denote A's start and end. Due to space considerations a more detailed discussion of the differences between approaches will be left for a expanded version of the paper.

An example of a continuous process would be the function, *height*, of a freely falling object. Suppose that a ball, 50 meters above the ground is dropped. The height of the ball at any time is determined by Newton's laws of motion. The height varies continuously with time until someone catches the ball. Suppose that the ball was caught after 2 seconds. The corresponding transition diagram is shown in Figure 1.



where f_0 and f_1 are defined as:

$$f_0(Y, T) = Y.\ f_1(Y, T) = Y - \tfrac{1}{2}gT^2.$$

Fig. 1. Transitions caused by *drop* and *catch*

Notice that states of this diagram are represented by mapping of values to the symbols *holding* and *height* over corresponding intervals of time. For example in state s_1, *holding* is mapped to false and *height* is defined by the function $f_1(50, T)$ where T ranges over the interval $[0, 2]$.

Intuitively, the time interval of a state s denotes the time lapse between occurrences of actions. The lower bound of the interval denotes start time of s which is the time at which an action initiates s. The upper bound denotes the end time of s which is the time at which an action terminates s. We assume that actions are instantaneous that is the actual duration is negligible with respect to the duration of the units of time in our domain. For computability reasons, we assign local time to states, therefore, the start time of every state s is 0 and the end time of s is the time elapsed since the start of s till the occurrence of an action

terminating s. For example, in Figure 1 the action *drop* occurs immediately after the start of state s_0. The end time of s_0 is therefore 0. The action *catch* occurs 2 time units after the start of state s_1. Therefore the end time of s_1 is 2.

The state s_2 in Figure 1 has the interval $[0, 5]$ associated with it. This interval was chosen randomly from an arbitrary collection of intervals of the form $[0, n]$ where $n \geq 0$. Therefore, any of the intervals $[0, 0]$ or $[0, 1]$ or $[0, 2]$ and so on could have been associated with s_2. In other words, performing *catch* leads to an infinite collection of states which differ from each other in their durations. The common feature among all these states is that *height* is defined by $f_0(30, T)$ and *holding* is true. We do not allow the interval $[0, \infty]$ for any state. We assume that every state is associated with two symbols - 0 and *end*. The constant 0 denotes the start time of the state and the symbol *end* denotes the end time of the state. We will give a formal definition of *end* when we discuss the syntax of the language.

We assume that there is a global clock which is a function that maps every local time point into global time. Figure 2 shows this mapping. Notice that

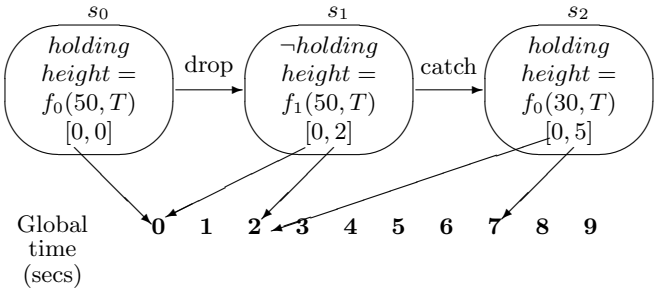


Fig. 2. Mapping between local and global time

this mapping allows one to compute the *height* of the ball at any global time, $t \in [\mathbf{0}, \mathbf{7}]$. This is not necessarily true for the value of *holding*. According to our mapping global time **0** corresponds to two local times: 0 in state s_0 and 0 in state s_1. Since the values of *holding* in s_0 and s_1 are *true* and *false* respectively, the global value of *holding* at global time **0** is not uniquely defined. Similar behavior can be observed at global time **2**. The phenomena is caused by the presence of instantaneous actions in the model. It indicates that **0** and **2** are the points of transition at which the value of *holding* is changed from *true* to *false* and *false* to *true* respectively. Therefore, it is $false$ at **1** and *true* during the interval **[3,7]**. Since the instantaneous actions *drop* and *catch* do not have a direct effect on *height*, its value at global time **0** and **2** is preserved, thereby resulting in unique values for *height* for every $t \in [\mathbf{0}, \mathbf{7}]$.

2 Syntax And Semantics of H

2.1 Syntax

To define our language, H, we first need to fix a collection, Δ, of time points. Ideally Δ will be equal to the set, R^+, of non-negative real numbers, but we can as well use integers, rational numbers, etc. We will use the variable T for the elements of Δ. We will also need a collection, $\mathcal{G}$, of functions defined on Δ, which we will use to define continuous processes. Elements of $\mathcal{G}$ will be denoted by lower case greek letters α, β, etc.

A process description language, $H(\Sigma, \mathcal{G}, \Delta)$, will be parameterized by Δ, $\mathcal{G}$ and a typed signature Σ. Whenever possible the parameters Σ, $\mathcal{G}$, Δ will be omitted. We assume that Σ contains regular mathematical symbols including $0, 1, +, <, \leq, \geq, \neq, *, etc.$ In addition, it contains two special classes, $\mathcal{A}$ and $\mathcal{P} = \mathcal{F} \cup \mathcal{C}$ of symbols called *actions* and *processes*.

Elements of $\mathcal{A}$ are elementary actions. A set $\{a_1, \ldots, a_n\}$ of elementary actions performed simultaneously is called a *compound* action. By actions we mean both elementary and compound actions. Actions will be denoted by a's. Two types of actions - *agent* and *exogenous* are allowed. *agent* actions are performed by an agent and *exogenous* actions are performed by nature. Processes from $\mathcal{F}$ are called *fluents* while those from $\mathcal{C}$ are referred to as *continuous processes*. Elements of $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{C}$ will be denoted by (possibly indexed) letters p's, k's and c's respectively. $\mathcal{F}$ contains a special functional fluent *end* that maps to Δ. *end* will be used to denote the end time of a state. We assume that for every continuous process, $c \in \mathcal{C}$, $\mathcal{F}$ contains two special fluents, $c(0)$ and $c(end)$. For example, the fluents $height(0)$ and $height(end)$ corresponding to *height*. Each process $p \in \mathcal{P}$ will be associated with a set $range(p)$ of objects referred to as the *range* of p. E.g. $range(height) = R^+$.

Atoms of $H(\Sigma, \mathcal{G}, \Delta)$ are divided into *regular* atoms, *c-atoms* and *f-atoms*.

- *regular* atoms are defined as usual from symbols belonging to neither $\mathcal{A}$ nor $\mathcal{P}$.
  E.g. mother(X,Y), sqrt(X)=Y.
- *c-atoms* are of the form $c = \alpha$ where $range(c) = range(\alpha)$.
  E.g. $height = 0$, $height = f_0(Y, T)$, $height = f_0(50, T)$.
  Note that α is strictly a function of time. Therefore, any variable occurring in a *c-atom* other than T is grounded.
  E.g. $height = f_0(Y, T)$ is a schema for $height = \lambda T.f_0(y, T)$ where y is a constant. $height = 0$ is a schema for $height = \lambda T.0$ where $\lambda T.0$ denotes the constant function 0.
- *f-atoms* are of the form $k = y$ where $y \in range(k)$. If k is boolean, i.e. $range(k) = \{\top, \bot\}$ then $k = \top$ and $k = \bot$ will be written simply as k and $\neg k$ respectively. E.g. *holding*, height(0)=Y, height(end)=0. Note that $height(0) = Y$ is a schema for $height(0) = y$.

The atom $p = v$ where v denotes the value of process p will be used to refer to either a *c-atom* or an *f-atom*. An atom u or its negation $\neg u$ are referred to as *literals*. Negation of $=$ will be often written as $\neq$. E.g. $\neg holding$, $height(0) \neq 20$.

Definition 1. An *action description* of H is a collection of statements of the form:

$$l_0 \ \text{ if } \ l_1, \ldots, l_n. \tag{1}$$

$$a_e \ \text{ causes } \ l_0 \ \text{ if } \ l_1, \ldots, l_n. \tag{2}$$

$$\text{impossible } \ a \ \text{ if } \ l_1, \ldots, l_n. \tag{3}$$

where a_e and a are elementary and arbitrary actions respectively and l's are literals of $H(\Sigma, \mathcal{G}, \Delta)$. The l_0's are called the *heads* of the statements (1) and (2). The set $\{l_1, \ldots, l_n\}$ of literals is referred to as the *body* of the statements (1), (2) and, (3). Please note that literals constructed from *f-atoms* of the form $end = y$ will not be allowed in the heads of statements of H.

A statement of the form (1) is called a *state constraint*. It guarantees that any state satisfying $l_1, \ldots, l_n$ also satisfies l_0. A *dynamic causal law* (2) says if an action, a_e, were executed in a state s_0 satisfying literals $l_1, \ldots, l_n$ then any successor state s_1 would satisfy l_0. An *executability condition* (3) states that action a cannot be executed in a state satisfying $l_1, \ldots, l_n$. If $n = 0$ then if is dropped from (1), (2), (3).

Example 1. Let us now construct an action description AD_0 describing the transition diagram from fig (1). Let $\mathcal{G}_0$ contain functions

$$f_0(Y, T) = Y.$$

$$f_1(Y, T) = Y - \frac{1}{2} g T^2.$$

where $Y \in range(height)$, g is acceleration due to gravity, and T is a variable for *time* points.

The description is given in language H whose signature Σ_0 contains actions *drop* and *catch*, a continuous process *height*, and fluents *holding*, $height(0)$ and $height(end)$. *holding* is a boolean fluent; $range(height)$ is the set of non-negative real numbers.

$$drop \ \ causes \ \ \neg holding. \tag{4}$$

$$impossible \ \ drop \ \ if \ \ \neg holding. \tag{5}$$

$$impossible \ \ drop \ \ if \ \ height(end) \ = \ 0. \tag{6}$$

$$catch \ \ causes \ \ holding. \tag{7}$$

$$impossible \ \ catch \ \ if \ \ holding. \tag{8}$$

$$height = f_0(Y, T) \ \ if \ \ height(0) = Y, \ \ holding. \tag{9}$$

$$height = f_1(Y, T) \ \ if \ \ height(0) = Y, \ \ \neg holding. \tag{10}$$

It is easy to see that statements (4) and (7) are dynamic causal laws while statements (5), (6) and (8) are executability conditions and statements (9) and (10) are state constraints.

2.2 Semantics

The semantics of *process description language*, H, is similar to the semantics of action language B given by McCain and Turner [10, 11]. An action description AD of H, describes a transition diagram, *TD(AD)*, whose nodes represent possible states of the world and whose arcs are labeled by actions. Whenever possible the parameter AD will be omitted.

Definition 2. An *interpretation*, I, of H is a mapping that assigns (properly typed) values to the processes of H such that for every continuous process, c, $I(c(end)) = I(c)(I(end))$ and $I(c(0)) = I(c)(0)$.

A mapping I_0 below is an example of an interpretation of action language of Example 1.

$$\begin{array}{l} I_0(end) = 0, \\ I_0(holding) = \top, \\ I_0(height(0)) = 50, \\ I_0(height(end)) = 50, \\ I_0(height) = f_0(50, T). \end{array}$$

Definition 3. An atom $p = v$ is *true in interpretation* I (symbolically $I \models p = v$) if $I(p) = v$. Similarly, $I \models p \neq v$ if $I(p) \neq v$.

An interpretation I is closed under the state constraints of AD if for any state constraint (1) of AD, $I \models l_i$ for every i, $1 \leq i \leq n$ then $I \models l_0$.

Definition 4. A *state*, s, of *TD(AD)* is an interpretation closed under the state constraints of AD.

It is easy to see that interpretation I_0 corresponds to the state s_0 in fig (1). By definition, the states of $TD(AD)$ are *complete*.

Whenever convenient, a state, s, will be represented by a *complete* set $\{l : s \models l\}$ of literals. For example, in Figure 1, the state s_0 will be the set

$$\begin{array}{l} s_0 = \{\, end = 0,\ holding,\ height(0) = 50, \\ \qquad height(end) = 50,\ height = f_0(50, T)\ \} \end{array}$$

Please note that only atoms are shown here. s_0 also contains the literals $holding \neq \bot$, $height(0) \neq 10$, $height(0) \neq 20$ and so on.

Definition 5. Action a is *executable* in a state, s, if for every non-empty subset a' of a, there is no executability condition

$$\text{impossible}\ \ a'\ \ \text{if}\ \ l_1, \ldots, l_n.$$

of AD such that $s \models l_i$ for every i, $1 \leq i \leq n$.

Let a_e be an elementary action that is executable in a state s. $E_s(a_e)$ denotes the set of all direct effects of a_e, i.e. the set of all literals l_0 for which there is a dynamic causal law

$$a_e \ causes \ l_0 \ if \ l_1, \ldots, l_n$$

in AD such that $s \models l_i$ for every i, $1 \leq i \leq n$. If a is a compound action then $E_s(a) = \bigcup_{a_e \in a} E_s(a_e)$.

A set L of literals of H is closed under a set, Z, of state constraints of AD if L includes the head, l_0, of every state constraint

$$l_0 \ \text{ if } \ l_1, \ldots, l_n$$

of AD such that $\{l_1, \ldots, l_n\} \subseteq L$.

The set $Cn_Z(L_1)$ of consequences of L_1 under Z is the smallest set of literals that contains L_1 and is closed under Z.

A transition diagram TD is a tuple$\langle \Phi, \Psi \rangle$ where
1. Φ is a set of states.
2. Ψ is a set of all triples $\langle s, a, s' \rangle$ such that a is executable in s and s' is a state which satisfies the condition

$$s' = Cn_Z(\ E_s(a) \ \cup \ (s \cap s' \ )\ ) \cup \{end = t'\} \tag{11}$$

where Z is the set of state constraints of AD and t' is the *end* time of s' that is $s'(end) = t'$. The argument to Cn_Z in (11) is the union of the set $E_s(a)$ of the "direct effects" of a with the set $s \cap s'$ of facts that are "preserved by inertia". The application of Cn_Z adds the "indirect effects" to this union. Since s' is the successor state of s with $end = t'$, the union of the set resulting after application of Cn_Z with the set $\{end = t'\}$ gives s'.

In the example from figure 1, the set $E_{s_0}(drop)$ of direct effects of *drop* will be defined as

$$E_{s_0}(drop) = \{\neg holding\}$$

The instantaneous action *drop* occurs at global time **0** and has no direct effect on the value of *height* at **0**. This means that the value of *height* at the *end* of s_0 will be preserved at time 0 of s_1. Therefore,

$$s_0 \cap s_1 = \{height(0) = 50\}$$

The application of Cn_Z to $E_{s_0}(drop) \cup (s_0 \cap s_1)$ gives the set

$$Q = \{\neg holding, height(0) = 50, height = f_1(50, T)\}$$

where Z contains the state constraints (9) and (10). The set Q will not represent the state s_1 unless *end* is defined. In the example, $s_1(end) = 2$, therefore, we get

$$\begin{aligned} s_1 = \{ & end = 2, \ \neg holding, \ height(0) = 50, \\ & height(end) = 30, \ height = f_1(50, T) \ \} \end{aligned}$$

Please note that, again, only atoms are shown here.

3 Specifying history

In addition to the action description, the agent's knowledge base may contain the domain's *recorded history* - observations made by the agent together with a record of its own actions.

The recorded history defines a collection of paths in the diagram which, from the standpoint of the agent, can be interpreted as the system's possible pasts. If the agent's knowledge is complete (e.g., it has complete information about the initial state and the occurrences of actions, and the system's actions are deterministic) then there is only one such path.

The *Recorded history*, Γ_n, of a system up to a current moment n is a collection of *observations*, that is statements of the form:

$$obs(v, p, t, i).$$
$$hpd(a, t, i).$$

where i is an integer from the interval $[0, n)$ and time point, $t \in \Delta$. i is an index of the trajectory. For example, $i = 5$ denotes the step 5 of the trajectory reached after performing a sequence of 5 actions. The statement $obs(v, p, t, i)$ means that process p was observed to have value v at time t of step i. Note that p is any process other than *end*. The statement $hpd(a, t, i)$ means that action a was observed to have happened at time t of step i. Observations of the form $obs(y, p, 0, 0)$ will define the initial values of processes.

Definition 6. A pair $\langle AD, \Gamma \rangle$ where AD is an action description of H and Γ is a set of observations, is called a *domain description*.

Definition 7. Given an action description AD of H that describes a transition diagram TD(AD), and recorded history, Γ_n, up to moment n, a path

$$\langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$$

in the TD(AD) is a *model* of Γ_n with respect to TD(AD), if for every i, $0 \leq i \leq n$ and $t \in \Delta$

1. $a_i = \{a : hpd(a, t, i) \in \Gamma_n\}$;
2. if $obs(v, p, t, i) \in \Gamma_n$ then $p = v \in s_i$.

4 Translation into Logic Program

In this section we will discuss the translation of a domain description written in language H into rules of an *A-Prolog* program. *A-Prolog* is a language of logic programs under the answer set semantics [5]. For this paper our translation will comply with the syntax of the SMODELS [12] inference engine.

We know that the statements of H contain continuous functions. Translating these statements into rules of A-Prolog is straight forward, however, due to issues involved with grounding, to run the resulting program under SMODELS,

the functions should be discretized. We will now look at how to discretize these functions.

Let $f : A \rightarrow B$ be a function of H. A discretized set, A_{h_1} corresponding to A is obtained as follows. First, a unit h_1 is selected. Next, A_{h_1} is constructed by selecting all those elements of A that are multiples of h_1. Since, in H, the domain of each function is time, we only consider positive multiples. Therefore,

$$A_{h_1} = \{0, h_1, 2h_1, 3h_1, \ldots\ldots\}$$

After A_{h_1} is defined, the discretized set B_d corresponding to B is then defined as $B_d = \{f(x) | x \in A_{h_1}\}$.

Let $g : A_{h_1} \rightarrow B_d$. The function $g : A_{h_1} \rightarrow B_d$ is called the discretized $\epsilon - approximation$ of f if $\forall x \in A_{h_1}$

$$\mid f(x) - g(x) \mid < \epsilon$$

where $\epsilon > 0$.

Definition 8. Given an action description AD of $H(\Sigma, \delta, \mathcal{G})$, the *discretized action description* $AD^{'}$ with respect to AD is obtained by replacing the occurrence of every function $f \in \mathcal{G}$ in the statements of AD by the function g where g is the discretized $\epsilon - approximation$ of f.

From now on, we will deal with discretized action descriptions. We assume that the agent makes observations at discrete time points and observes only the discretized values of processes.

Definition 9. Given a domain description $\mathcal{D} = \langle AD, \Gamma_n \rangle$, the *discretized domain description* $\mathcal{D}^{'}$ with respect to $\mathcal{D}$ is the pair $\langle AD^{'}, \Gamma_n \rangle$ where $AD^{'}$ is the discretized action description with respect to AD and Γ_n is the recorded history up to moment n.

Next we will show how to translate discretized domain descriptions. Note that, from now on, when we say domain description (or action description) we refer to the discretized one. First, let us look at the general way of declaring actions and processes.

4.1 Declarations

Let us look at a general way of declaring actions and processes:

$$action(action_name, action_type).$$
$$process(process_name, process_type).$$

action_name and *action_type* are non-numeric constants denoting the name of an action and its type respectively. Similarly, *process_name* and *process_type* are

non-numeric constants denoting the name of a process and its type respectively. For instance in example 1 the actions and processes are declared as follows:

$$action(drop, agent).$$
$$action(catch, agent).$$

$$process(height, continuous).$$
$$process(holding, fluent).$$

Now let us see how the range of a process is declared. There are a couple of ways of doing this. The range of *height* from Example 1 is the set of non-negative real numbers. In logic programming this would lead to an infinite grounding. Therefore, we made a compromise and chose integers ranging from 0 to 50.

$$values(0..50).$$
$$range(height, Y) :- values(Y).$$

holding is a boolean fluent. Therefore, we write

$$range(holding, true).$$
$$range(holding, false).$$

Suppose we have a switch that can be set in three different positions, the range of the process *switch_position* is declared as:

$$range(switch_position, low).$$
$$range(switch_position, medium).$$
$$range(switch_position, high).$$

In order to talk about the values of processes and occurrences of actions we have to consider the *time* and *step* parameters. Integers from some interval $[0, n]$ will be used to denote the *step* of a trajectory. I's will be used as variables for *step*. Every *step* has a duration associated with it. Integers from some interval $[0, m]$ will be used to denote the *time* points of every *step*. In this case, m will be the maximum allowed duration for any *step*. T's will be used as variables for *time*. Therefore, we write

$$step(0..n).$$
$$time(0..m).$$

Assume that n and m are sufficiently large for our applications. Then we add the rules

$$\#domain\ step(I; I1).$$
$$\#domain\ time(T; T1; T2).$$

for declaring the variables $I,\ I1,\ T,\ T1$ and, $T2$ in the language of SMODELS. The first domain declaration asserts that the variables I and $I1$ should get their domain from the literal $step(I)$.

4.2 General translations

We will now discuss a general translation of statements of H into rules of A-prolog. If a is an elementary action occurring in a statement that is being translated, it is translated as

$$o(a, T, I)$$

which is read as "*action a occurs at time T of step I*". If a is a compound action then each elementary action $a_e \in a$ will be translated in the same manner.

If l is a literal occurring in any part of a statement, other than the head of a dynamic causal law, then it will be written as

$$\alpha_0(l, T, I)$$

where $\alpha_0(l, T, I)$ is a function, described below, that denotes a case-specific translation of literal l. A literal, l, occurring in the head of a dynamic causal law will be written as

$$\alpha_0(l, 0, I + 1)$$

In this paper, due to difficulties with generalizing inertia axioms, we limit ourselves to action descriptions of H in which the heads of dynamic causal laws are either *f-atoms* or their negations. This can be done without loss of generality as all other dynamic causal laws can be replaced using a dynamic causal law/state constraint pair. From now on we will only consider such action descriptions.

Definition 10. Let AD be an action description of H, n and m be positive integers, and $\Sigma(AD)$ be the signature of AD. We will use n and m as the maximum values for steps and time points respectively. $\Sigma_m^n(AD)$ denotes the signature obtained as follows:
$const(\Sigma_m^n(AD)) = \langle const(\Sigma(AD)) \cup \{0, \ldots, n\} \cup \{0, \ldots, m\}\rangle$;
$pred(\Sigma_m^n(AD)) = \{val, -val, o, process, action, range, step, time, values\}$

Let

$$\alpha_0^n(AD) = \langle \alpha_0(AD), \Sigma_m^n(AD)\rangle, \tag{12}$$

where

$$\alpha_0(AD) = \bigcup_{r \in AD} \alpha_0(r), \tag{13}$$

and $\alpha_0(r)$ is defined as follows:

- $\alpha_0(l_0$ if $l_1, \ldots, l_n.)$ is

$$\alpha_0(l_0, T, I) :- \alpha_0(l_1, T, I), \quad \ldots, \quad \alpha_0(l_n, T, I). \tag{14}$$

- $\alpha_0(a_e$ causes l_0 if $l_1, \ldots, l_n.)$ is

$$\alpha_0(l_0, 0, I + 1) :- o(a_e, T, I), \quad \alpha_0(l_1, T, I), \quad \ldots, \quad \alpha_0(l_n, T, I). \tag{15}$$

- α_0(impossible a if $l_1, \ldots, l_n.)$ is

$$:- o(a, T, I), \quad \alpha_0(l_1, T, I), \quad \ldots, \quad \alpha_0(l_n, T, I). \tag{16}$$

In statement (3), if a is the non-empty compound action $\{a_1, \ldots, a_m\}$ then $o(a, T, I)$ in rule (16) will be replaced by $o(a_1, T, I), \ldots, o(a_m, T, I)$. The construction of $\alpha_0^n(AD)$ in equation (12) is such that the declarations from section (4.1) are added to $\alpha_0(AD)$.

$\alpha_0(l, T, I)$ will be replaced by

- $val(V, c, 0, I)$ if l is an atom of the form $c(0) = v$. It is read as "*V is the value of process c at time 0 of step I*".
  E.g. $height(0) = Y$ will be translated as $val(Y, height, 0, I)$.
- $-val(V, c, 0, I)$ if l is of the form $c(0) \neq v$. It is read as "*V is not the value of process c at time 0 of step I*".
- $val(V, p, T, I)$ if l is an atom of the form $p = v$ other than $c(0) = v$. It is read as "*V is the value of process p at time T of step I*".
  E.g. $height(end) = 0$ will be translated as $val(0, height, T, I)$.
- $-val(V, p, T, I)$ if l is of the form $p \neq v$ other than $c(0) \neq v$. It is read as "*V is not the value of process p at time T of step I*".

$\alpha_0(l, 0, I + 1)$ will be replaced by

- $val(V, p, 0, I + 1)$ if l is of the form $p = v$.
- $-val(V, p, 0, I + 1)$ if l is of the form $p \neq v$.

Note that when translating the *f-atom*, $end = y$ we will not follow the above conventions. Instead we translate it as $end(T, I)$ where T denotes the *end* of step I. Before we look at some examples we will discuss domain independent axioms.

4.3 Domain independent axioms

Domain independent axioms define properties that are common to every domain. We will denote such a collection of axioms by Π_d. Given a action description AD of H, let

$$\alpha^n(AD) = \alpha_0^n(AD) \cup \Pi_d. \tag{17}$$

Π_d is the following set of rules:

1. End of state axioms. These axioms will define the *end* of every state s. The end of a state is the local time at which an action terminates s. When it comes to implementation we talk about the *end* of a *step* instead of state. Therefore, we write

   $$end(T, I) :- o(A, T, I). \tag{18}$$

   If no action occurs during a *step* then *end* will be the maximum time point allowed for that *step*. This is accomplished by using the choice rule

   $$\{end(m, I)\}1. \tag{19}$$

   The consequence of the rule (19) is that the number of end(m,I) that will be true is either 0 or 1. A *step* cannot have more than one *end*. This is expressed by (20).

   $$:- end(T1, I), \quad end(T2, I), \quad neq(T1, T2). \tag{20}$$

Every *step* must end. Therefore, we write

$$ends(I) \ :- \ end(T, I). \tag{21}$$
$$:- \ not \ \ ends(I). \tag{22}$$

Every *step*, i, is associated with an interval $[0, e]$ where 0 denotes the start time and e denotes the end time of i. We will use the relation *out* to define the time points, $t \notin [0, e]$ and *in* to define the time points, $t \in [0, e]$.

$$out(T, I) :- \ end(T1, I), \ \ T > T1. \tag{23}$$

$$in(T, I) :- \ not \ out(T, I). \tag{24}$$

By using these relations in our rules we can avoid computing process values at time points, $t \notin [0, e]$.

2. Inertia axiom. The inertia axiom states that *things normally stay as they are*. It has the following form:

$$val(Y, P, 0, I + 1) \ :- \ val(Y, P, T, I), \ \ end(T, I), \ \ not \ -val(Y, P, 0, I + 1). \tag{25}$$

Intuitively, rule (25) says that actions are instantaneous. In the example from figure 1, the value of *height* at global time **0** remains 50 when the action *drop* occurs at **0**.

3. Other axioms. A fluent remains constant throughout the duration of a *step*. This is expressed by the axiom (26).

$$val(Y, P, T, I) :- \ val(Y, P, 0, I), \ \ process(P, fluent), \ \ in(T, I). \tag{26}$$

Axiom (27) says that no process can have more than one value at the same time.

$$-val(Y1, P, T, I) \ :- \ val(Y2, P, T, I), \ \ neq(Y1, Y2). \tag{27}$$

Adding history Given an action description AD of H and recorded history Γ_n up to moment n, we will construct a logic program that contains translations of the statements of AD and Γ_n.

Γ_n contains observations of the form $obs(v, p, t, i)$ and $hpd(a, t, i)$ which are translated as facts of A-Prolog programs. Let $\Sigma^n_{m,\Gamma}(AD)$ denote the signature obtained as follows:

- $const(\Sigma^n_{m,\Gamma}(AD)) = const(\Sigma^n_m(AD))$;
- $pred(\Sigma^n_{m,\Gamma}(AD)) = pred(\Sigma^n_m(AD)) \cup \{hpd, obs\}$.

Let

$$\alpha^n(AD, \Gamma_n) = \langle \Pi^\Gamma, \Sigma^n_{m,\Gamma}(AD) \rangle. \tag{28}$$

where

$$\Pi^\Gamma = \alpha^n(AD) \cup \hat{\Pi} \cup \Gamma_n. \tag{29}$$

and $\hat{\Pi}$ is the set of rules:

1. Reality check axiom that guarantees that the agent's predictions match with his observations.

$$:- \ obs(Y, P, T, I), \quad -val(Y, P, T, I). \tag{30}$$

2. The following rule says that if action A was observed to have happened at time T of step I then it must have occurred at time T of step I.

$$o(A, T, I) :- \ hpd(A, T, I). \tag{31}$$

3. The following rule is for defining the initial values of processes.

$$val(Y, P, 0, 0) \ :- \ obs(Y, P, 0, 0). \tag{32}$$

Hence $\alpha^n(AD, \Gamma_n)$ is the resulting logic program containing translations for the statements of AD and Γ_n.

4.4 Correctness

The following definitions will be useful for describing the relationship between answer sets of $\alpha^n(AD, \Gamma_n)$ and models of Γ_n.

Definition 11. Let AD be an action description of H and A be a set of literals over $\alpha^n(AD, \Gamma_n)$. We say that A *defines the sequence* $\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ if

$$\sigma_i = \{l \mid \alpha_0(l, t, i) \in A\} \cup \{end = t \mid end(t, i) \in A\}$$

$$\text{for} \ \ 0 \leq i \leq n, \text{ and}$$

$$a_i = \{a \mid o(a, t, i) \in A\}$$

$$\text{for} \ \ 0 \leq i < n.$$

Definition 12. The initial situation of Γ_n is *complete* if and only if for any process p of Σ, Γ_n contains $obs(v, p, 0, 0)$.

The following theorem establishes the relationship between the theory of actions in H and logic programming.

Theorem 1. Given a discretized domain description $\mathcal{D} = \langle AD, \Gamma_n \rangle$; if the initial situation of Γ_n is *complete* then M is a model of Γ_n with respect to $TD(AD)$ iff M is defined by some answer set of $\alpha^n(AD, \Gamma_n)$.

The proof is omitted due to space considerations.

5 Conclusions and Future Work

In this paper we presented a new type of action language, the *process description language*. Our language, H, is capable of representing domains containing continuous processes in a simple and concise manner. In sample runs, computation of small, discrete domains (using the translated action description and SMODELS) is reasonable, but, in general, efficient processing will require a non-ground solver.

The authors would like to thank ARDA, United Space Alliance, and NASA who's grants helped fund this research.

References

1. [BG03] M. Balduccini and M. Gelfond. Diagnostic reasoning with A-Prolog. In *Journal of Theory and Practice of Logic Programming (TPLP)*, 3(4-5):425-461, Jul 2003.
2. [BG03a] M. Balduccini and M. Gelfond. Logic Programs with Consistency-Restoring Rules. In *AAAI Spring 2003 Symposium*, 2003.
3. [BG00] C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In Minker, J,. ed., *Logic-Based AI*, Kluwer Academic publishers,(2000),257-279.
4. [BST02] C. Baral, T. Son and L. Tuan. A transition function based characterization of actions with delayed and continuous effects. In *Proc. of KR'02*, pages 291-302.
5. [GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming, In *Logic Programming: Proc. of the Fifth International Conference and Symposium*, 1988, pp. 1070-1080.
6. [GL98] M. Gelfond and V. Lifschitz. Action Languages. In *Electronic Transactions on Artificial Intelligence*, 3(6),1998.
7. [GW98] M. Gelfond and R. Watson. On Methodology of Representing Knowledge in Dynamic Domains. In *Proc. of the 1998 ARO/ONR/NSF/DARPA Monterey Workshop on Engineering Automation for Computer Based Systems*, pp. 57-66, 1999.
8. [Lif97] V. Lifschitz, Two components of an action language, In *Annals of Mathematics and Artificial Intelligence*, Vol. 21, 1997, pp. 305-320.
9. [Lif99] V. Lifschitz. Action languages, Answer Sets and planning. In *The Logic Programming Paradigm:a 25 year perspective*.357-373, Springer Verlag,1999.
10. [MT95] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI-95*, pages 1978-1984, 1995.
11. [MT97] N. McCain and H. Turner. Causal theories of action and change. In *Proc. of AAAI-97*, pages 460-465, 1997.
12. [NS97] I. Niemela and P. Simons. Smodels - an implementation of the stable model and well founded semantics for normal logic programs. In *Proc. of LPNMR'97*, pages 420-429,1997.
13. [Pin94] J.A. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD Thesis, Department of Computer Science, University of Toronto, 1994.
14. [Rei96] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Principles of Knowledge Representation and Reasoning: Proc. of the Fifth International Conference (KR'96)*, pages 2-13, Cambridge, Massachusetts, U.S.A., November 1996.
15. [Rei01] R. Reiter. Time, concurrency and processes. In *Knowledge in action: Logical Foundations for specifying and implementing dynamical systems*, pages 149-183, ISBN 0-262-18218-1, MIT, 2001.
16. [San89]E. Sandewall. Filter Preferential entailment for the logic of action in almost continuous worlds. In *Proc. of IJCAI'89*, pages 894-899, 1989.
17. [Sha89]M. Shanahan. Representing continuous change in the Event Calculus. In *Proc. of the European Conference on Artificial Intelligence*, pages 598-603, 1990.
18. [WC03] R. Watson and S. Chintabathina. Modeling hybrid systems in action languages. In *Proc. of the 2nd International ASP'03 workshop*, pages 356-370, Messina, Sicily, Italy, September 2003.