# A transition function based characterization of actions with delayed and continuous effects

**Chitta Baral**
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
*chitta@asu.edu*

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

**Le-Chi Tuan**
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
*lctuan@asu.edu*

## Abstract

In this paper we present a transition function based characterization of actions in a realistic environment. Our language allows for the specification of actions with duration, continuous effects, delayed effects, dependency on non-sharable resources, and accounts for parallel and overlapping execution of actions. One of the main contribution of our paper is a new definition of state in such an environment. Our notion of state encodes not only the fluent values but also obligations due to delayed effect of actions that were executed recently. This allows us to define a Markovian transition function. Although there have been earlier attempts at developing action languages with similar features, none of them present a transition function based characterization.

## 1 INTRODUCTION AND MOTIVATION

In this paper we follow the approach of high level action description languages with a transition function based semantics (as first done in [11]) to characterize more realistic actions. In particular, we consider: (i) actions that have continuous effects such as the action of driving that changes the fluent recording the distance traveled, (ii) actions with fixed and variable durations, (iii) actions that use non-sharable resources such as machines, (iv) overlapping execution of actions that contribute to the change of a resource, and (v) actions with delayed effects. Although some of these features have been earlier considered in some action description languages, none of them present a transition function based semantics.

By a transition function based semantics we mean *defining what a state is and its transition between states due to actions.* If we consider a state to be a snapshot of the world at a particular time, then how do we express this snapshot when actions have durations? It is not enough to use the traditional notion of a state that encodes the value of fluents and resources as it misses out the actions under execution. For example, if we go to a factory and try to express the snapshot of the factory at 12:35 PM, then it is not enough to just say the fluent values, but also important to express which actions are under execution and for how long. Only then we can correctly predict the value of fluents at a future time by only considering the current 'state'. In other words a richer notion of state is necessary to define a Markovian transition function.

The traditional transition functions define a function $\Phi(s, a)$, which gives the state that would be reached if action $a$ was executed in state $s$. Since we now consider actions with durations, and allow overlapping of actions our goal is to define a function $\Phi(s, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$, where $t_1 \leq t_2 \ldots \leq t_n \leq t$, which expresses the state of the world after $t$ units of time (from the time $t_0$ corresponding to $s$) assuming that the actions $a_1, \ldots, a_n$ were started at relative (to $t_0$) times $t_1, \ldots, t_n$ respectively.

Such a transition function is important in planning and other kinds of reasoning about actions and is absent in earlier attempts [7, 15, 20, 21] of characterizing actions with such realistic features. Such a transition function based characterization is also important in light of the upcoming planning contest at AIPS'02 where planners that allow such actions are expected to compete. In Section 3 we will give a slightly more detailed comparison of our approach with earlier reasoning about action attempts [7, 15, 20, 21], and the semantics of the languages PDDL2.1 [9] and PDDL++ [10] that have been recently proposed for use in the AIPS'02 planning contests and beyond.

## 2 THE LANGUAGE $\mathcal{ADC}$

We will call our language $\mathcal{ADC}$ as our language is for $\mathcal{A}$ctions with $\mathcal{D}$elayed and $\mathcal{C}$ontinuous effects.

### 2.1 SYNTAX OF THE DOMAIN DESCRIPTION PART

The alphabet of a domain description in $\mathcal{ADC}$ consists of a set of action names $\mathbf{A}$, a set of fluent names $\mathbf{F}$, and a set of process names $\mathbf{P}$. Each fluent $f \in \mathbf{F}$ has a domain $dom(f)$ associated with it that prescribes what value $f$ can take. Each domain $dom(f)$ is also associated with a set of binary relations over it. For example, if $dom(f)$ is the set of integers then the set of binary relations over $dom(f)$ could contain the standard comparison relations $<, >, \leq, \geq, =, \neq$. Each process name is associated with a fluent definition or a update expression which will be defined precisely later on.

An *evaluable expression* is constructed from functions and fluents in the same way an arithmetic expression is constructed in C, C++ etc. Variables in an evaluable expression are fluents in $\mathbf{F}$. Given the values of fluents, the value of an evaluable expression will be computed in the standard way. For example, if $f$ is a real-valued fluent then $3 * (f + 2)^2$ is an evaluable expression and if the value of $f$ is 2 then the value of $3 * (f + 2)^2$ is 48. The domain of the value of an evaluable expression is called the type of that expression.

An *atom* is of the form $\phi_1$ **op** $\phi_2$, where $\phi_1$ and $\phi_2$ are evaluable expressions of the same type and **op** is a binary relation associated to the type of $\phi_1$ (or $\phi_2$). Atoms of the form $\phi = True$ and $\phi = False$ can be simply written as $\phi$ and $\neg\phi$ respectively.

In $\mathcal{ADC}$, actions can have delayed and continuous effects. This means that effects of actions might not happen immediately or can last over a period of time and therefore are associated with time intervals of the form $[t_1, t_2]$ where $t_1, t_2$ are real numbers, $0 \leq t_1 \leq t_2$. Actions can also initiate (or terminate) a process whose duration is unknown at the time it starts. For example, the action of turning on the water tap of the bath tub will initiate the process of water flowing into the bath tub. The process will be terminated when the tap is turned off. On the other hand, flipping the switch of a lamp changes its position (on to off and off to on) and takes a constant amount of time. To express this, the various propositions in $\mathcal{ADC}$ are of the following form

$$\textbf{executable } a \textbf{ if } c_1, \ldots, c_k \tag{2.1}$$

$$a \textbf{ needs } r_1, \ldots, r_m \tag{2.2}$$

$$a \textbf{ causes } f = valf(f, f_1, \ldots, f_n, t) \textbf{ from } t_1 \textbf{ to } t_2 \tag{2.3}$$

$$a \textbf{ contributes } valf(f, f_1, \ldots, f_n, t) \textbf{ to } f \textbf{ from } t_1 \textbf{ to } t_2 \tag{2.4}$$

$$a \textbf{ initiates } p \textbf{ from } t_s \tag{2.5}$$

$$a \textbf{ terminates } p \textbf{ at } t_s \tag{2.6}$$

$$p \textbf{ is\_associated\_with } f = valf(f, f_1, \ldots, f_n, t) \tag{2.7}$$

$$p \textbf{ is\_associated\_with } f \leftarrow valf(f, f_1, \ldots, f_n, t) \tag{2.8}$$

where

- $a$ is an action name, the $f$'s are fluents, the $c$'s are atoms,
- the $r$'s are atoms of the specific form $f = \phi$ where $f$ is a numeric-type fluent and $\phi$ is an evaluable expression,
- $t_s, t_1, t_2$ $(t_1 \leq t_2)$ are non-negative real numbers, representing time units relative to the time point where $a$'s execution is started,
- $valf(f, f_1, \ldots, f_n, t)$ is a function that takes the value of the fluents $f, f_1, \ldots, f_n$ when $a$ started its execution and the elapsed time $t$ in $[0, t_2 - t_1]$ and returns a value from $dom(f) \cup \{undefined\}$, and
- $p$ is a process name and (2.7) states that $p$ is associated with the *fluent definition* "$f = valf(f, f_1, \ldots, f_n, t)$" whereas (2.8) says that it is associated with the *update expression* "$f \leftarrow valf(f, f_1, \ldots, f_n, t)$", where $f$ and $valf(f, f_1, \ldots, f_n, t)$ have the same meaning as in the above item.

Intuitively, the above propositions describe the effects of actions and their executability conditions. Propositions of the form (2.1)-(2.2) state the conditions under which $a$ is executable. Here, (2.1), called an *executability condition*, represents the requirements on certain fluents (or resources) that $a$ does not need exclusively to start its execution and (2.2), called a *resource condition*, characterizes the conditions on fluents that $a$ needs exclusively to start its execution. For example, a postal worker needs his car for mail delivery s but he must be in the car for the action to be executable.

Propositions of the form (2.3) and (2.4) describe the different ways in which an action can affect the value of a fluent. In (2.3), $a$ causes the value of $f$ to change according to the function $valf(f, f_1, \ldots, f_n, t)$ during the interval $[t_1, t_2]$. On the other hand, $a$ in (2.4) affects the value of $f$ at $t_1$ by contributing an increase specified by $valf(f, f_1, \ldots, f_n, t)$ during the interval $[t_1, t_2]$. We require that in (2.3) if $t_1 = 0$ then $valf(f, f_1, \ldots, f_n, 0) = f$ and in (2.4) if $t_1 = 0$ then $valf(f, f_1, \ldots, f_n, 0) = 0$.

The purpose of the propositions (2.5) and (2.6) is to encode continuous effect of actions that do not have a predefined duration. In that case a proposition of the form (2.5) is used to indicate which actions can initiate the effect (a process) and a proposition of the form (2.6) is used to indicate which actions can terminate the process. Note that unlike in some earlier proposal [20, 21, 10] we do not designate these actions as start and end actions. In our framework the same process can be initiated and terminated by different actions. For example, the termination of the filling of a bathtub can be caused by turning off the bath faucet and also by turning off the main water switch.

We now give some examples to illustrate the kind of domain description that can be expressed in $\mathcal{ADC}$.

**Example 1** Consider the action of driving a car for 10 units of time with a velocity $v$. We represent this action *of fixed duration* by $drive_{0,10}(v)$. Now, let $gas\_in\_tank$ denote the amount of gasoline available in the tank.

The executability condition saying that the action $drive_{0,10}(v)$ is executable if $gas\_in\_tank \geq 20$ can be expressed in $\mathcal{ADC}$ as:

**executable** $drive_{0,10}(v)$ **if** $gas\_in\_tank \geq 20$.

The effect of the action $drive_{0,10}(v)$ on the fluent $loc$ (encoding how far from the initial position the car is) is expressed as:

$drive_{0,10}(v)$ **causes** $locn = locn + v * t$ **from** 0 **to** 10.

We can record the information that the action $drive_{0,10}(v)$ is under execution by the following propositions:

$drive_{0,10}(v)$ **causes** $driving$ **from** 0 **to** 10

and

$drive_{0,10}(v)$ **causes** $\neg driving$ **from** 10 **to** 10

where $driving$ is a Boolean-fluent.

We can express that driving at a velocity $v$ consumes $c(v)$ unit of gasoline per unit time as

$drive_{0,10}(v)$ **contributes** $-c(v) * t$ **to** $gas\_in\_tank$ **from** 0 **to** 10.

Intuitively, if $gas\_in\_tank = 20$ and $c(3) = 1.5$ at time 0, then the level of gasoline in the tank at the time $0, 1, \ldots, 10$ will be as follows, assuming that $drive_{0,10}(3)$ is executed at the time 0:

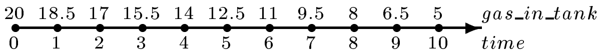


Figure 1: Gasoline in tank and time – $drive_{0,10}(3)$ at 0

The proposition

$fill\_gas_{0,10}$ **contributes** $2 * t$ **to** $gas\_in\_tank$ **from** 0 **to** 10

expresses the contribution of the action $fill\_gas_{0,10}$ to the value of the $gas\_in\_tank$. Intuitively, it says that the action of filling gas for 10 units of time will increase the amount of gasoline, by the amount specified by the function $2 * t$. Let us assume that while driving we can refill. The following diagram shows the value of $gas\_in\_tank$ at different times, assuming that the action $drive_{0,10}(3)$ starts its execution at the time moment 0 and the action $fill\_gas_{0,10}$ starts its execution at the time moment 1:

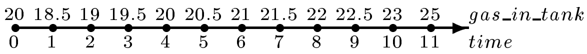


Figure 2: Gasoline in tank and time – $drive_{0,10}(3)$ at 0 and $fill\_gas_{0,10}$ at 1

**Example 2** Let us now consider the action of driving when the duration is not fixed beforehand. In that case we have the following propositions:

$start\_drive(3)$ **initiates** $p_1$ **from** 0
$start\_drive(3)$ **initiates** $p_2$ **from** 0
$stop\_drive(3)$ **terminates** $p_1$ **at** 0
$stop\_drive(3)$ **terminates** $p_2$ **at** 0
$p_1$ **is_associated_with** $locn = locn + 3 * t$
$p_2$ **is_associated_with** $gas\_in\_tank \leftarrow -1.5 * t$

This says that the process of increasing the fluent $loc$ and the process of reducing the amount of gasoline in the tank will start at the time $start\_drive(3)$ is executed. These processes will be stopped when the action *stop_drive*(3) is executed. □

A *domain description* $D$ is a set of propositions of the form (2.1)-(2.8). For simplicity, we require that for each action $a$ in $D$, there is at most one proposition of the form (2.2) whose action is $a$. An *action theory* is a pair $(D, O)$ where $D$ is a domain description and $O$ is a set of *observations* of the form

$$\textbf{initially } f = c \tag{2.9}$$

where $f$ is a fluent and $c$ is a value belonging to $dom(f)$.

### 2.2 QUERIES

In the presence of delayed and continuous effects, we are interested not only about the values of fluents at certain time point but also about the intervals of them and whether the value satisfy some conditions or not. For example, we might be interested in knowing whether the amount of gasoline in the tank is more than 5 units or not; or whether the car is at certain location after driving for some unit of time; or a combination of both questions etc. This leads us to consider *queries* of the following form:

$$\begin{array}{c} c_1[\delta_1^-, \delta_1^+], \ldots, c_k[\delta_k^-, \delta_k^+] \\ \textbf{after } A_1 : t_1, \ldots, A_n : t_n \end{array} \tag{2.10}$$

where $c_1, \ldots, c_k$ are atoms, the $\delta$'s are time units with $0 \le \delta_i^- \le \delta_i^+$, $A_1, \ldots, A_n$ are set of actions, and $0 \le t_1 < \ldots < t_n$ are time units. The intuitive reading of query (2.10) is that whether the atoms $c_i$ are satisfied during the interval $\delta_i^-, \delta_i^+$ for $1 \le j \le k$ given that the $A_j$ are executed at the moment $t_j$ $(1 \le j \le n)$. All the time variables $t$'s are relative to the current situation and the $\delta$'s are relative to $t_n$.

For the action theory in Example 1, let us assume that initially $gas\_in\_tank = 25$, $loc = 0$, and driving with the speed of 3 units of distance per unit of time will consume 1.5 units of gasoline per unit of time. The query

$$(gas\_in\_tank \ge 5)[5,6], (loc = 30)[10,10] \textbf{ after } \{drive_{0,10}(3)\} : 0$$

asks whether the amount of gasoline is greater than or equal to 5 (units of gasoline) during the interval $[5, 6]$ after driving with the speed of 3 for 5 units of time and whether the location of the car is 30 at the time moment 10.

A more complicated query involving the driving action and the *fill_gas* action is

$$(gas\_in\_tank = 20)[15,15] \textbf{ after } \{drive_{0,10}(3)\} : 0, \{fill\_gas_{0,10}\} : 10$$

which asks whether the amount of gasoline in the tank is 20 at the time moment 15 after driving and then refilling it at the time moment 10.

### 2.3 SEMANTICS

We will now present a transition function based semantics for action theories in $\mathcal{ADC}$ that facilitates the answering of queries of the form (2.10) given an action theory $(D, O)$. As it is customary in approaches to reasoning about action and change using high-level action description languages we will define the notion of a state of the world and characterize the transitions between states.

#### 2.3.1 States: Interpretation Of Fluents And Obligations

Given a domain description $D$, an *interpretation* $I$ of $D$ assigns each fluent $f \in \mathbf{F}$ a value $v \in dom(f) \cup \{undefined\}$, denoted by $f = v$. Value of an evaluable expression $x$ with respect to $I$, denoted by $I(x)$, is defined inductively over their structure in the usual way[1].

An interpretation $I$ satisfies an atom $x_1$ $\mathbf{op}$ $\mathbf{x_2}$ if $x_1$ and $x_2$ are defined with respect to $I$ and belong to the same domain, say $dom(f)$ for some fluent $f$, $\mathbf{op}$ is a binary relation associated to $dom(f)$ and $I(x_1)$ $\mathbf{op}$ $\mathbf{I(x_2)}$ holds. When $I$ satisfies an atom $a$, we write $I \models a$. $I$ satisfies a set of atoms $\{a_1, \ldots, a_n\}$, written as $I \models \{a_1, \ldots, a_n\}$, if $I \models a_i$ for every $i$, $1 \le i \le n$.

In transition function based approaches to reasoning about action and change (for example, in [2, 3, 4, 11, 13, 12, 14, 17, 18, 23, 25]), a state of the world is represented by a set of fluents. The semantics of an action theory is specified by a transition function that maps pairs of states and actions into states. In the presence of time and delayed effects, the effects of an action might not happen immediately or might last over a time interval. This stipulates us to represent the current state of the world by a *snapshot of the world recording the fluent values and the future obligations due to*

[1] Because a function can return *undefined* as an answer, *undefined* could be a possible value of $I(x)$. In this case, we say that $x$ is undefined with respect to $I$.

*delayed effects of recent actions and actions that are under execution.* We elaborate more on this point below.

Consider the action $drive_{0,10}(3)$ from Example 1. Suppose that we start $drive_{0,10}(3)$ at the time moment 0. We expect that the fluent $loc$ will change its value from $loc_0 = 0$ to 30 during the interval $[0, 10]$ (according to the equation $3 * t$) and the fluent $gas\_in\_tank$ will decrease its value by 1.5 unit every unit of time. That is, at the time moment 0, when $drive_{0,10}(3)$ starts its execution, we would express the obligation due to this action on $loc$ and $gas\_in\_tank$ by equations of the form

$$loc = loc_0 + 3 * t \text{ and } gas\_in\_tank{-}{=}1.5 * t$$

where $t \in [0, 10]$. At a later moment, say 4 (i.e, after 4 units of time), although these equations do not change the constraint on the time variable does, and they would become

$$loc = loc_0 + 3 * t \text{ and } gas\_in\_tank{-}{=}1.5 * t$$

where $t \in [4, 10]$. Equivalently, we can write

$$loc = loc_0{+}3{*}(t{+}4) \text{ and } gas\_in\_tank{-}{=}1.5{*}(t{+}4)$$

where $t \in [0, 6]$. These equations differ from the previous equations in that $t$, the time variable, refers to the time elapsed from *now* instead of the elapsed time from the beginning of the interval. However, to represent the correct value of the fluent, the time between the beginning of the interval and *now* – which is 4 in the last two equations – must be also recorded. We represent this by the interval $[-4, 6]$ where the negative part of the interval represents the elapsed time since the beginning of the obligation while the positive part says how long more the obligation lasts. We note that $[-4, 6]$ can be obtained from $[0, 10]$ by shifting it to the left by 4 units.

The above discussion suggests that we represent a state by a pair $\langle I, Q\rangle$, hereafter called a *snapshot*, where $I$ is an interpretation and $Q$ is a set of future effects (or obligations, for short). To distinguish an obligation caused by actions in (2.3)-(2.4) from an obligation initiated by actions in (2.5)-(2.6), we attach to each obligation a name. We use $\top$ for the former and the process name for the latter. Further, we will use $[t_s, \infty)$ to represent the interval of an obligation generated by a process to indicate that we do not know when such an obligation will terminate[2]. Obligations caused by an action in (2.3)-(2.5) do change the world but obligations caused by (2.6) do not. Instead, they terminate obligations caused by (2.5). For this reason, we only add to $Q$ obligations caused by actions in (2.3)-(2.5). Whenever an action in (2.6) is executed we will update $Q$ by removing a matching obligation, if any. An obligation is one of the following form

$$(\top, f = valf(v_f, v_{f_1}, \dots, v_{f_n}, t), t_1, t_2) \quad (2.11)$$
$$(\top, f{\leftarrow}\ valf(v_f, v_{f_1}, \dots, v_{f_n}, t), t_1, t_2) \quad (2.12)$$
$$(p, f = valf(v_f, v_{f_1}, \dots, v_{f_n}, t), t_1, \infty) \quad (2.13)$$
$$(p, f{\leftarrow}\ valf(v_f, v_{f_1}, \dots, v_{f_n}, t), t_1, \infty) \quad (2.14)$$

where

- $f$ is a fluent,
- $t_1, t_2$ are two real numbers with $t_1 \leq t_2$, and
- $valf(v_f, v_{f_1}, \dots, v_{f_n}, t)$ is a function with $v_f, v_{f_1}, \dots, v_{f_n}$ are the values of the fluents $f, f_1, \dots, f_n$ at the time the obligation is added to $Q$.

In the following, we will refer to obligations of the form (2.11) or (2.13) as *type-1* obligations. The other obligations are called *type-2* obligations.

For convenience, we refer to the elements of an obligation $e$ by $e.name$, $e.f$, $e.t_1$, and $e.t_2$ and call $f$, $t_1$, and $t_2$ as the fluent, lower, and upper bound of $e$, respectively. We sometimes use $e.[t_1, t_2]$ or $e.[t_1, \infty)$ to represent the interval specified by the lower and upper bound of $e$. A fluent $f$ *occurs* in an obligation $e$ if $f$ is the left hand side of $e.f$.

Intuitively, an obligation $e = (e.name, e.f, e.t_1, e.t_2)$ states that during the time interval $e.[t_1, t_2]$ (or $e.[t_1, \infty)$), the value of a fluent $f$ (the left hand side of $e.f$) will be determined (or increased by an amount) as specified by the function on the right hand side of $e.f$. As time progresses, $e.t_1$ and $e.t_2$ will be updated accordingly. Furthermore, since an action might have delayed effects, we distinguish obligations whose interval contains 0 ($0 \in e.[t_1, t_2]$) from others and call them *active obligations*. Obviously, to determine what is the current value of a fluent, we only need to consider the active obligations.

**Example 3** Let us consider the case where the action $drive_{0,10}(3)$ in Example 1 is executed at the time 0 with the initial value of $loc$ equal to 0. Then,

- The type-1 obligation $(\top, loc = 3 * t, 0, 10)$ is added to $Q$. After one unit of time, this

[2]Adding the name to the obligation or $\infty$ to the interval gives us a uniform representation of obligations.

obligation will be changed to $(\top, loc = 3 * t, -1, 9)$.
   - The type-2 obligation $(\top, gas\_in\_tank \leftarrow -1.5t, 0, 10)$ is added to $Q$. After one unit of time, it will be changed to $(\top, gas\_in\_tank \leftarrow -1.5t, -1, 9)$.

On the other hand, if $start\_drive(3)$ (Example 2) is executed at the time 0, then

- The type-1 obligation $(p_1, loc = 3 * t, 0, \infty)$ is added to $Q$. After one unit of time, the obligation will be changed to $(p_1, loc = 3 * t, -1, \infty)$.
- The type-2 obligation $(p_2, gas\_in\_tank \leftarrow -1.5t, 0, \infty)$ is added to $Q$. After one unit of time, it will be changed to $(p_2, gas\_in\_tank \leftarrow -1.5t, -1, \infty)$.

If $stop\_drive(3)$ is executed at the moment 1, then its effect is to remove the obligations named $p_1$ and $p_2$ from $Q$. Therefore, $Q$ will no longer contain the obligations $(p_1, loc = 3 * t, -1, \infty)$ and $(p_2, gas\_in\_tank \leftarrow -1.5t, -1, \infty)$

Now that we have defined what a state is, we need to discuss when a snapshot $\langle I, Q \rangle$ represents a valid state of the world. It is easy to see that the obligations of a snapshot dictate the values of fluents in the future. Further, to determine the value of a fluent, say $f$, at a time moment $\delta$, we need to consider all the obligations that will assign a value for $f$ (or contribute some increase for $f$) at $\delta$. We call these obligations *active* at $\delta$ and characterize them in the next definition.

**Definition 1** Let $\langle I, Q \rangle$ be a snapshot and $\delta \geq 0$. An obligation $e = (e.name, e.f, e.t_1, e.t_2)$ is *active at* $\delta$ if $e.t_1 \leq \delta \leq e.t_2$. □

Because actions in $\mathcal{ADC}$ can be executed in parallel and different actions might assign different value to a fluent $f$ at some future moment of time, there might be situations in which two type-1 obligations assign different value to a fluent $f$ simultaneously. The following picture depicts such a situation:

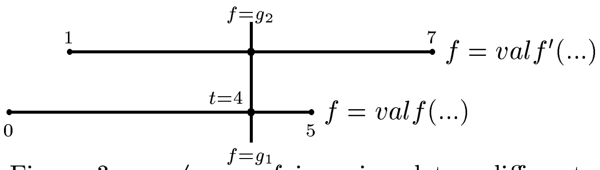


Figure 3: $g_1 \neq g_2$ – $f$ is assigned two different value at time 4

We consider this a *conflict situation* and call a snapshot that causes a conflict situation an *inconsistent snapshot*. We formulate precisely this notion in the next definition.

**Definition 2** A snapshot $\langle I, Q \rangle$ is said to be *inconsistent* if one of the following conditions is satisfied:

1. there exists a time moment $\delta \geq 0$ and two type-1 obligations $e$ and $e'$, both are active at $\delta$, with $e.f$ is "$f = valf(v_f, v_{f_1}, \ldots, v_{f_n}, t)$" and $e'.f$ is "$f = valf'(v_f, v_{g_1}, \ldots, v_{g_m}, t)$" and there exists a $\delta'$ such that $0 \leq \delta' < \delta$ and
   - both $e$ and $e'$ are active at $\delta'$, and
   - $valf(v_f, v_{f_1}, \ldots, v_{f_n}, \delta - e.t_1) \neq valf'(v_f, v_{g_1}, \ldots, v_{g_m}, \delta - e'.t_1)$.
2. there exists a time moment $\delta \geq 0$, a fluent $f$, a type-1 obligation $e$, and a type-2 obligation $e'$ such that (i) $f$ occurs in $e$ and $e'$, and (ii) both $e$ and $e'$ are active at $\delta$.

We say that $\langle I, Q \rangle$ is *consistent* if it is not inconsistent. □

In the above definition, in condition 1 we require the existence of $\delta'$ that is smaller than $\delta$ and both $e$ and $e'$ are active at $\delta'$. The rationality behind this decision is similar to that of allowing obligations of actions to override the current values of fluents. More precisely, consider two type-1 obligations $e$ and $e'$, both assign value to $f$. Suppose that $e'$ happens later than $e$ ($e.t_1 \leq e'.t_1$). Thus, at the time moment $e'.t_1$, value of $f$ dictated by $e$ should be considered as the value by *inertia* whereas the *new* value of $f$ dictated by $e'$ should be considered the value of $f$. This said, if $e.[t_1, t_2]$ and $e'.[t_1, t_2]$ share some points then the value of $f$ dictated by $e$ and $e'$ should be the same on every point common to these intervals except the lower bound of $e'$[3]. Condition 2 requires that a fluent is not assigned a value by an action and updated by another at the same time.

### 2.3.2 Transition Function

In this section we present several definitions leading to the definition of a transition function. Since we have to go through several definitions, we start with a road map. As mentioned in the intro-

[3] Another possible way of dealing with this problem is to ignore the type-1 obligation that is currently active when a new type-1 obligation on the same fluent becomes active. This seems not natural because under this view, every possible snapshot is valid which is unlikely in most real-life domains.

duction, our objective here is to define the function $\Phi(s, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$. We do this by defining a function $\tau(s, \delta)$ (Definition 6) that gives us the state after the passage of $\delta$ units of time when starting from the state $s$. (Note that this is not $s$. Even though no new action has started some of the obligations in $s$ may become due because of passage of time.) The definition of $\tau(s, \delta)$ involves defining the new interpretation denoted by $I^{Q,\delta}$ (Definitions 3-4) and the new obligations denoted by $Shift(Q, \delta)$ (Definition 5).

Our next goal is to define a function $\Phi(s, A, \delta)$ (Definition 10) that defines the state reached after executing the set of actions $A$ immediately in the state $s$ and then waiting for $\delta$ time units. $\Phi(s, \{(a_1, t_1), \ldots, (a_n, t_n)\}, t)$ is then defined using $\Phi(s, A, \delta)$ by sorting $\{(a_1, t_1), \ldots, (a_n, t_n)\}$ into $\{(A'_1, t'_1), \ldots, (A'_k, t'_k)\}$ such that each set $A'_i \subseteq \{a_1, \ldots, a_n\}$, and for any $i$, if $a \in A'_i$ then $(a, t'_i) \in \{(a_1, t_1), \ldots, (a_n, t_n)\}$; and progressing through them one by one.

To define $\Phi(s, A, \delta)$ we (i) use $\tau$, (ii) define executability of $A$ in $s$ (Definition 7) (iii) define the new obligations due to $A$ denoted by $E(A, s)$ (Definition 8), and (iv) define updating of obligations because of possible removals caused by $A$ denoted by $Update(Q, T)$ (Definition 9).

We now define $\tau$ that maps a pair of a snapshot $s$ and a non-negative number $\delta$ into a snapshot $\tau(s, \delta)$. Let us assume that $\tau(s, \delta) = \langle I', Q' \rangle$. We will define $I'$ by showing how to compute the value of a fluent $f$ at $\delta$ given a consistent snapshot $\langle I, Q \rangle$. We need the following definition.

**Definition 3** Let $\langle I, Q \rangle$ be a consistent snapshot, $f$ be a fluent, and $\delta$ be a non-negative number. A type-1 obligation $e_0$ is called the *most recent type-1 obligation of $f$ relative to $\delta$* if $f$ occurs in $e_0$ and $e_0.t_2 = \max\{e'.t_2 \mid f \text{ occurs in } e' \in Q, e'.t_2 < \delta\}$. □

The above definition identifies the most recent type-1 obligation relative to $\delta$ that affects the value of $f$. Let us call this obligation $e$. Intuitively, $e$ dictates the value of $f$ starting from $e.t_2$ to $\delta$ if no type-1 obligation that changes $f$ is present. And, if there are contributions to $f$ from $e.t_2$ to $\delta$, then they should be added to its value at $e.t_2$. We will use this to define the value of $f$ at $\delta$ in the next definition.

**Definition 4** Let $\langle I, Q \rangle$ be a consistent snapshot and $\delta$ be a non-negative number. The *value of a fluent $f$ with respect to $\langle I, Q \rangle$ at $\delta$*, denoted by $I^{Q,\delta}(f)$, is defined as follows. First, let $e_0$ be a most recent type-1 obligation of $f$ relative to $\delta$. Let $t_i = e_0.t_2$ and $I^*(f) = valf'(v_f, v_{h_1}, \ldots, v_{h_k}, e_0.t_2 - e_0.t_1)$ if $e_0$ exists and $t_i = 0$ and $I^*(f) = I(f)$, otherwise. Let

$\Gamma(f) = \{\delta v \mid \delta v = valf(v_f, v_{f_1}, \ldots, v_{f_n}, \delta - e.t_1)$ - $valf(v_f, v_{f_1}, \ldots, v_{f_n}, 0)$ where:
there exists a type-2 obligation $e \in Q$ with
(i) $e$ is active at some $\delta'$, $t_i \leq \delta' \leq \delta$; and
(ii) $e.f =$ is "$f \leftarrow valf(v_f, v_{f_1}, \ldots, v_{f_n}, t)$"$\}$.

- For $f$, that does not occur in any type-1 obligation $e$ in $Q$, that is active at $\delta$, if $\Gamma(f) \neq \emptyset$ then $I^{Q,\delta}(f) = I^*(f) + \Sigma_{v \in \Gamma(f)} v$,; otherwise, $I^{Q,\delta}(f) = I^*(f)$,
- For $f$, that occurs in an type-1 obligation $e$ in $Q$, that is active at $\delta$, with $e.f$ is "$f = valf(v_f, v_{g_1}, \ldots, v_{g_m}, t)$" and $e.t_1 = \max\{e'.t_1 \mid f \text{ occurs in } e' \in Q, e' \text{ is active at } \delta\}$, $I^{Q,\delta}(f) = valf(v_f, v_{g_1}, \ldots, v_{g_n}, \delta - e.t_1)$.[4]

□

In the above definition, the set $\Gamma(f)$ contains all the contributions to the fluent $f$ at $\delta$ which are specified by type-2 obligations. The value of $f$ is then defined using the sum of all the contributions made to $f$ (first case) or the value dictated by a type-1 obligation (second case).

**Example 4** Consider the domain description of the action $drive_{0,10}(3)$ from Example 1, the interpretation $I = \{loc = 0, gas\_in\_tank = 20\}$, and the set of obligations $Q = \{(\top, loc = 3 * t, 0, 10), (\top, gas\_in\_tank \leftarrow -1.5 * t, 0, 10)\}$. For $0 \leq \delta \leq 10$, we have that $\Gamma(gas\_in\_tank) = \{-1.5\delta\}$, which gives

- $I^{Q,\delta}(loc) = 3 * t = 3\delta$ (because of the type-1 obligation in $Q$), and
- $I^{Q,\delta}(gas\_in\_tank) = I(gas\_in\_tank) - 1.5\delta = 20 - 1.5\delta$ (because of the type-2 obligation in $Q$).

[4] The consistency requirement on $\langle T, Q \rangle$ suggests that $f$ does not occur in any type-1 obligations that are active at $\delta$. Hence, we do not need to worry about $\Gamma(f)$ in this case. However, $f$ might occur in more than one type-1 obligations. As we have discussed after Definition 2, the value of $f$ at $\delta$ should be obtained from the type-1 obligation whose upper bound is greatest among them. For instance, if $drive_{0,10}(v)$ (Example 1) starts at the time 0, then the intuitive value of $driving$ at the moment 10 is false which is also the value specified by the type-1 effect $(\top, driving = false, 10, 10)$ instead of the type-1 effect $(\top, driving = true, 0, 10)$.

Definition 4 tells us how to compute the interpretation of $\tau(s, \delta)$. To complete the definition of $\tau(s, \delta)$, we need to specify how its set of obligations should be computed. Obviously, we have to eliminate from $Q$ all the obligations that are expired, i.e., their upper bound is smaller than $\delta$. Furthermore, we have to update the obligations's interval. We call this a shifting of $Q$ and define it as follows.

**Definition 5** For a set of obligations $Q$ and a time moment $\delta \geq 0$, a shifting of $Q$ by $\delta$, denoted by $Shift(Q, \delta)$, is obtained from $Q$ by :

- Removing from $Q$ every obligation $e$ with $e.t_2 < \delta$,
- For every obligation $e$, that remains after the first step, replacing $e.t_1$ and $e.t_2$ with $e.t_1 - \delta$ and $e.t_2 - \delta$, respectively. □

We now define the *step transition function* $\tau$, that maps pairs of snapshots and time moments into snapshots.

**Definition 6 (Step Transition Function)** Let $s = \langle I, Q \rangle$ be a snapshot and $\delta \geq 0$. $\tau(s, \delta)$ is defined as follows.

1. if $\langle I, Q \rangle$ is inconsistent then $\tau(s, \delta)$ is undefined;
2. otherwise, $\tau(s, \delta) = \langle I', Q' \rangle$ where $I'(f) = I^{Q,\delta}(f)$ for every $f \in \mathbf{F}$ and $Q' = Shift(Q, \delta)$ □

As $\mathcal{ADC}$ allows actions to occur in parallel, we need to characterize when a set of actions can be executed in parallel. In [4], concurrent actions have been discussed and different ways in dealing with concurrency have been proposed. The approach in [4] assumes that actions are instantaneous. Reiter modified the situation calculus to deal with time and concurrency [21]. His approach, however, requires that the action theory contains preconditions for concurrent actions which are represented by a set of simple actions. In the next definition, we characterize when a set of actions can be executed at the same time.

**Definition 7** Let $s = \langle I, Q \rangle$ be a snapshot and $A$ be a set of actions. We say that $A$ is executable in $s$ if,

1. every action $a \in A$ is executable in $s$, i.e., for every $a \in A$ there exists a executable condition **executable** $a$ **if** $c_1, \ldots, c_k$ and $I \models c_i$ for every $c_i$; and
2. $I \models f \geq \Sigma_{(f,\phi) \in S} I(\phi)$, where $S = \{(f, \phi) \mid$ there exists an action $a \in A$ and $f = \phi$ is an atom belonging to the resource condition whose action is $a\}$. □

The intuition behind the above definition is that two actions can be executed in parallel only when the exclusive resources need for two actions must be sufficient. For example, consider a simple job-shop scheduling problem with two tasks $t_1$ and $t_2$. Both $t_1$ and $t_2$ require a machine $m$ to complete. This requirement can be expressed in $\mathcal{ADC}$ by the two resource condition "$t_1$ **needs** $m = 1$" and "$t_2$ **needs** $m = 1$". Clearly, $t_1$ and $t_2$ can be executed in parallel only if we have at least 2 machines ($m$). This is expressed by the second condition of Definition 7. We note that these two resource conditions cannot be replaced by the two executable conditions "**executable** $t_1$ **if** $m = 1$" and "**executable** $t_2$ **if** $m = 1$" because this would imply that the two tasks can be executed in parallel. Next, we define $E(a, s)$, the obligations due to execution of $a$ in $s$.

**Definition 8** The obligations of an action $a$ in a snapshot $s = \langle I, Q \rangle$ denoted by $E(a, s)$ contains

- a type-1 obligation $(\top, f = valf(v_f, v_{f_1}, \ldots, v_{f_m}, t),\ t_1, t_2)$ for each proposition "$a$ **causes** $f = valf(f, f_1, \ldots, f_m, t)$ **from** $t_1$ **to** $t_2$",
- a type-2 obligation $(\top, f \leftarrow valf(v_f, v_{f_1}, \ldots, v_{f_m}, t), t_1, t_2)$ for each proposition "$a$ **contributes** $valf(f, f_1, \ldots, f_m, t)$ **to** $f$ **from** $t_1$ **to** $t_2$",
- a type-1 obligation $(p, f = valf(v_f, v_{f_1}, \ldots, v_{f_m}, t), t_s, \infty)$ for each proposition "$a$ **initiates** $p$ **from** $t_s$" and "$p$ **is_associated_with** $f = valf(f, f_1, \ldots, f_m, t)$",
- a type-2 obligation $(p, f \leftarrow valf(v_f, v_{f_1}, \ldots, v_{f_m}, t), t_s, \infty)$ for each proposition "$a$ **initiates** $p$ **from** $t_s$" and "$p$ **is_associated_with** $valf(f, f_1, \ldots, f_m, t)$ **to** $f$",

where $valf(v_f, v_{f_1}, \ldots, v_{f_m}, t)$ is the function over time obtained from $valf(f, f_1, \ldots, f_m, t)$ by simultaneously replacing every occurrence of the fluents $f, f_1, \ldots, f_m$ with their values with respect to $I$, $v_f, v_{f_1}, \ldots, f_{f_m}$, respectively.

For a set of actions $A$, let $E(A, s) = \cup_{a \in A} E(a, s)$. □

Since an action $a$ may also terminate some obligations, we define the set $T(a, s) = \{(p, t_s) \mid$ "$a$ **terminates** $p$ **from** $t_s$" belongs to $D\}$ that lists the obligations that are terminated by $a$. Let $T(A,s) = \cup_{a \in A} T(a,s)$. The next definition shows how to update a set of obligations given a set of terminations of the form $(p,t)$.

**Definition 9** Let $Q$ be a set of obligations and $T$ be a set of pairs $(p,t)$ where $p$ is a process name and $t$ is a time variable. The *update* of $Q$ with respect to $T$, denoted by $Update(Q,T)$, is defined as the set of obligations obtained from $Q$ by

- Removing from $Q$ all effects $(e.name, e.f, e.t_1, e.t_2)$ for which $T$ contains a pair $(e.name, t)$ such that $t \leq e.t_1$,
- For each remaining $e$ in $Q$ with $e = (e.name, e.f, e.t_1, e.t_2)$ if $T$ contains a pair $(e.name, t)$ such that $e.t_1 \leq t \leq e.t_2$, then replacing $e.t_2$ with $t$. $\square$

We now define $\Phi(s,A,\delta)$ and $\Phi(s, \{(A_1,t_1), \ldots, (A_n,t_n)\}, \delta)$.

**Definition 10 (Transition Function)** Let $D$ be a domain description, $A$ be a set of actions, $s = \langle I, Q \rangle$ be a snapshot, and $\delta \geq 0$. We define $\Phi(s,A,\delta)$ as follows.

1. if $A$ is not executable in $s$ or $s$ is inconsistent, then $\Phi(s,A,\delta)$ is undefined;
2. if $A$ is executable in $s$, then $\Phi(s,A,\delta) = \tau(\langle I, Update(E(A,s) \cup Q, T(A,s)) \rangle, \delta)$. $\square$

**Definition 11** For a snapshot $s$, a sequence of action occurrences $[A_1 : t_1], \ldots, [A_n : t_n]$ such that $0 \leq t_1 < t_2 < \ldots < t_n$ and a time moment $\delta \geq t_n$, we define the sequence of snapshots $s_0, \ldots, s_n$ as follows.

- $s_0 = \Phi(s, \emptyset, t_1)$,
- for $i = 1, \ldots, n-1$, $s_i = \Phi(s_{i-1}, A_i, t_{i+1} - t_i)$, and
- $s_n = \Phi(s_{n-1}, A_n, \delta - t_n)$.

We denote $s_n$ by $\Phi(s, \{(A_1,t_1), \ldots, (A_n,t_n)\}, \delta)$. $\square$

The following proposition follows directly from the above definition.

**Proposition 1** For $A$'s, $t$'s, $\delta$, and $s$ that satisfy the conditions given in Definition 11, we have that $\Phi(s, \{(A_1,t_1), \ldots, (A_n,t_n)\}, \delta) = \Phi(\Phi(s, \{(A_1,t_1), \ldots, (A_{n-1},t_{n-1})\}, t_n - t_{n-1}), A_n, \delta - t_n)$. $\square$

The significance of this proposition is that it allows us to compute the state resulting from the execution of a sequence of action occurrences step by step; thus allowing the development of a forward planner that can exploit domain specific knowledge to improve its efficiency like those planners for action theories with actions of fixed durationlike TLPlan [1], TALplan [8] and SHOP [19].

#### 2.3.3 Query Entailment

We now define when a query is entailed by a domain description. We first state when a temporal atom of the form $c[t_1,t_2]$ is true in a snapshot $s$.

**Definition 12** A snapshot $s$ entails a temporal atom of the form $c[t_1,t_2]$ if (i) $s$ is consistent; and (ii) for every $\delta \in [t_1,t_2]$, $\tau(s,\delta) \models c$. $\square$

In the next definition, we define what is the initial state of a domain description.

**Definition 13 (Initial State)** Let $(D,O)$ be an action theory. An *initial state* $s_0$ of $(D,O)$ is a snapshot $\langle I_0, \emptyset \rangle$ of $D$ that satisfies every observation in $O$, i.e. $I_0 \models f = v$ for every observation "**initially** $f = v$" in $O$. $\square$

**Definition 14 (Entailment)** Let $(D,O)$ be an action theory. A query
$c_1[\delta_1^-, \delta_1^+], \ldots, c_k[\delta_k^-, \delta_k^+]$
**after** $A_1{:}t_1, \ldots, A_n{:}t_n$
is entailed by $(D,O)$, denoted by
$(D,O) \models c_1[\delta_1^-, \delta_1^+], \ldots, c_k[\delta_k^-, \delta_k^+]$ **after** $[A_1 : t_1], \ldots, [A_n : t_n]$,
if for every initial state $s_0$ of $(D,O)$, $s$ is defined and $s \models c_i[\delta_i^-, \delta_i^+]$ for every $i$, $1 \leq i \leq k$, where $s = \Phi(s_0, \{(A_1,t_1), \ldots, (A_n,t_n)\}, 0)$, and $\Phi$ is the transition function of $D$. $\square$

We illustrate the last definition through the following proposition and its proof.

**Proposition 2** Consider the action theory $(D,O)$ where $O$ consists of two observations

**initially** $loc = 0$, and
**initially** $gas\_in\_tank = 20$,

and $D$ is the domain description consisting of the actions $drive_{0,10}(3)$, $fill\_gas_{10}$, $start\_drive(3)$, and $stop\_drive(3)$ and the fluents $loc$ and $gas\_in\_tank$ from Example 1. Then, $(D,O) \models (gas\_in\_tank \geq 5)[5,6]$ **after** $\{drive_{0,10}(3)\} : 0$.

**Proof:** (sketch) First, it is easy to see that the only initial state of $(D,O)$ is the snapshot $s_0 = \langle I_0, \emptyset \rangle$ with $I_0(loc) = 0$ and $I_0(gas\_in\_tank) = 20$. Furthermore, $\tau(s_0, 0) = s_0$.

Since $I_0 \models gas\_in\_tank > 0$ and $D$ does not contain a proposition of the form (2.2) whose action is $drive_{0,10}(3)$, we conclude that $drive_{0,10}(3)$ is executable in $s_0$. $E(drive_{0,10}(3), s_0)$ consists of a type-1 obligation $(\top, loc = 3t, 0, 10)$ and a type-2 obligation $(\top, gas\_in\_tank \leftarrow -1.5t, 0, 10)$ (see Example 3). By definition, we have that

$$\begin{aligned}
&\Phi(s_0, \{(drive_{0,10}(3), 0)\}, 0) \\
&\quad = \tau(\langle I_0, E(drive_{0,10}(3) \cup \emptyset)\rangle, 0) \\
&\quad = \tau(\langle I_0, \{(\top, loc = 3t, 0, 10), \\
&\qquad\qquad (\top, gas_in_tank \leftarrow -1.5t, 0, 10)\}\rangle, 0) \\
&\quad = \langle I_0, \{(\top, loc = 3t, 0, 10), \\
&\qquad\qquad (\top, gas_in_tank \leftarrow -1.5t, 0, 10)\}\rangle \\
&\quad = s_1.
\end{aligned}$$

It follows from the computation in Example 4 that $\tau(s_1, \delta) \models (gas\_in\_tank \geq 5)$ for $\delta \in [5, 6]$. Thus, $s_1 \models (gas\_in\_tank \geq 5)[5, 6]$. In other words, we have proved that $(D, O) \models (gas\_in\_tank \geq 5)[5, 6]$ **after** $\{drive_{0,10}(3)\} : 0$. □

# 3 RELATED WORK, CONCLUSION AND FUTURE WORK

In this paper we have presented a transition function based characterization of actions in presence of unsharable resources, with durations, and delayed and continuous effects. To make the presentation in this extended abstract simpler and to stay within page limits we have on purpose simplified the language a bit by not allowing conditional effects, and instead added a lot of explanations and examples. In the full paper (with more space) we will have conditional effects with temporal conditions. We would like to point out that by using delayed effects we can express actions with fixed durations as we have shown through many examples. The main contribution of our paper is the notion of state that not only has fluent interpretations but also *obligations*, and a transition function.

Although actions with durations and continuous effects were earlier considered, none of them presented a transition function based characterization. Our characterization allows us to easily map it to a model generation based planner. We describe such a planner based on logic programming in [6] and example codes can be found in http://www.public.asu.edu/~cbaral/resources/.

In terms of related work, the three main works with similar goals as ours are [20, 21], [7, 15], and [10]. In [20, 21], Reiter adds an explicit notion of time to situation calculus and uses start and stop actions to characterize processes. Among the main differences between his work and ours is that (i) his time is "actual time" while all our notions of time are relative times, (ii) he needs triggers to characterize actions with fixed durations, (iii) he does not have delayed effects and resource needs, and (iv) his characterization is in logic (Situation calculus) while ours requires a simpler background of sets and functions. The thesis [15] is perhaps one of the latest report on the work at Linkoping that is related to this paper. Among the main differences between the work there and this paper are: (i) they do not consider continuous effects (an early paper by Sandewall [22] does), and (ii) theirs is a logic based characterization with no notion of states and transition functions. The recent paper [10] gives a language for planning with actions similar to the ones in this paper. The language is divided into 4 levels and continuous effects are in level 4. The semantic characterization in [9] is only up to level 3 with a pointer that a semantics of level 4 is defined based on hybrid automata. Besides this, the main difference between the characterization there and ours is that their characterization is only for planning while ours (as we will elaborate in the full paper) is for planning and other reasoning about action tasks. Their notion of states is much simpler than ours and does not record obligations. Hence, it surely won't lead to a Markovian transition function.

In terms of future work, we plan to have a more thorough comparison with related work. In particular we plan to investigate particular subclasses of domain descriptions in our language for which our transition function based semantics matches with the semantics of the above mentioned works and those addressing continuous changes [24] or numeric fluents [16]. We plan to allow for more general observations, as in [5], and conclusion about missing action occurrences from those observations. We also plan to further extend our planner in [6] based on the characterization in this paper and use domain dependent knowledge to expedite planning in such an environment. We would also like to enhance $\mathcal{ADC}$ by adding to it new features such as those for expressing temporal conditions, conditional effects, or state constraints. This will allow the use of $\mathcal{ADC}$ in representing and reasoning about domains in which the effects of actions depend on the value of some fluents temporally and conditionally.

**Acknowledgments**

The first two authors would like to acknowledge the support of the NASA grant NCC2-1232. The first author is partially supported by a NSF grant 0070463. The second author is also partially supported by a NSF grant EIA-9810732.

## References

[1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1,2):123–191, 2000.

[2] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 2017–2023. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[3] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 866–871. Morgan Kaufmann Publishers, San Francisco, CA, 1993.

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

[5] C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming*, 31(1-3):201–243, May 1997.

[6] C. Baral, T.C. Son, and L.C. Tuan. Reasoning about actions in presence of resources: applications to planning and scheduling. In *Proceedings of International conference on information technology (to appear)*, 2001.

[7] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnstrom. Tal: Temporal action logics – language specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 3(15), 1998. http://www.ep.liu.se/ej/etai/1998/015.

[8] P. Doherty and J. Kvarnstom. TALplanner: An Empirical Investigation of a Temporal Logic-based Forward Chaining Planner. In *Proceedings of the 6th Int'l Workshop on the Temporal Representation and Reasoning, Orlando, Fl. (TIME'99)*, 1999.

[9] M. Fox and D. Long. PDDL+: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, 2001.

[10] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, 2001.

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

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

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

[14] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: preliminary report. In *Proceedings of AAAI 98*, pages 623–630, 98.

[15] J. Gustafsson. Extending Temporal Action Logic. PhD thesis, Linkoping University, 2001.

[16] J. Lee and V. Lifschitz. Additive fluents. In *AAAI Spring Symposium: "Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning"*, pages 116–123. AAAI Press, 2001.

[17] N. McCain and M. Turner. Causal theories of action and change. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 460–467. AAAI Press, 1997.

[18] N. McCain and M. Turner. A causal theory of ramifications and qualifications. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1978–1984. Morgan Kaufmann Publishers, San Mateo, CA, 95.

[19] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Conference on Artificial Intelligence*, pages 968–973. AAAI Press, 1999.

[20] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L. Aiello, J. Doyle, and S. Shapiro, editors, *KR 96*, pages 2–13, 1996.

[21] R. Reiter. *KNOWLEDGE IN ACTION: Logical Foundations for Describing and Implementing Dynamical Systems.* MIT Press, 2001.

[22] E. Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Artificial Intelligence*, pages 894–899, 1989.

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

[24] M. Thielscher. Modelling actions with ramifications in nondeterministic, concurrent, and continuous domains – and a case study. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'2000)*, pages 497–502, 2000.

[25] H. Turner. Representing actions in logic programs and default theories. *Journal of Logic Programming*, 31(1-3):245–298, May 1997.