ASET: Representing Indeterminate Durative Tasks for Symbolic Fault Tolerant Planning in Multi-Agent Domains*

Rune M. Jensen, Manuela M. Veloso, and **Sergey Shchukin**
Computer Science Department, Carnegie Mellon University,
Pittsburgh, PA 15213-3891, USA

Abstract

In this paper, we introduce a novel planning domain representation called ASynchronous Evolving Tasks (ASET). The main contribution of ASET is a representation of temporally extended tasks that may be non-deterministic both with respect to duration and effects. Like NADL, ASET represents metric domain knowledge as state variables and explicitly models the environment as a set of uncontrollable agents. We formally define ASET descriptions and their transformation to a non-deterministic planning domain. Using a Boolean encoding of this transformation, ASET planning problems can be solved efficiently using state-of-the-art symbolic non-deterministic planning systems. In particular, we show how to augment an ASET description to generate fault tolerant plans via the strong planning algorithm.

Introduction

Real-world planning is often considered a two dimensional challenge: we need powerful planning algorithms to scale to the massive problems frequently encountered in practice, and we need rich domain representations to capture the multifaceted nature of real scenarios. There is also, however, a third dimension which is to *integrate planning and execution*. This problem has been addressed by execution languages for autonomous agents like robots with limited world state knowledge and short perceptual horizons (Georgeff & Lansky 1986; Firby 1989; Simmons 1994). The focus in this work has been on reacting to unexpected events while overall acting deliberatively. Another aspect is to use the planning and execution cycle to *learn* domain knowledge and *adapt* the planning domain to the execution environment. This is relevant for autonomous agents, but seems particularly important for application areas in logistics, manufacturing, and scheduling that often are characterized by a large amount of domain knowledge that is costly to maintain (Sabin & Weigel 1998). Previous work in this direction is sparse, but includes the prodigy project (Wang 1994; Yolanda 1992) and Haigh's work on the Xavier robot (Haigh & Veloso 1998).

The goal of our research is to investigate *domain model adaptation* in a closed-loop planning and execution framework. The main difference between our approach and the previous work mentioned above is that we depart from classical deterministic planning and instead base our work on a novel planning domain representation that combines temporal and non-deterministic planning. This domain representation is the focus of the present paper.

The planning problems, we are interested in, can be illustrated by the following scenario. A manager is in charge of a set of workers, but also observes a set of customers that the manager is unable to control. The manager sits behind a screen showing the current state of the world and is connected to the workers by a telephone. Each customer and worker is always in a state of activity executing some task. When a worker finishes a task, the worker calls the manager. It is then the manager's responsibility to give the worker a new task. The manager does this by taking a careful look at the screen and observing the activities of both workers and customers. Since the manager has a good idea about the nature of the tasks of both customers and workers, the manager assumes that it only is necessary to take action when the phone rings.

We believe that this scenario captures a wide range of real-world planning problems where some planning unit (the manager) has gathered all world information necessary to plan successfully (or at least assumes it has done so) and relies heavily on models of the activities, it must deal with to avoid getting overloaded by micro managing.

It may be claimed that the manager is facing a planning problem with a natural representation in one of the current planning languages. To our best knowledge, this is not the case. There are several difficulties.

1. The manager only has a "good idea" about how tasks may evolve. Exactly how they evolve is uncertain and leads to non-determinism both with respect to duration and effects. Further, there are uncontrollable activities carried out by customers.
2. Any time the phone rings, the manager takes a look at the screen and observes the state of the world. No information is hidden by workers or customers even if they are in the

*This research is sponsored by BBNT Solutions, LLC under contract no. FA8760-04-C-0002 with the US Air Force. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsoring institutions, the U.S. Government or any other entity.

process of changing this information.

The first point requires a domain representation with durative non-deterministic actions. However, temporal planning languages have deterministic actions e.g. (Fox & Long 2003; Bacchus & Ady 2001; Laborie & Ghallab 1995) and non-deterministic planning languages do not consider durative actions e.g. (Piergiorgio *et al.* 2002; Giunchiglia, Kartha, & Lifschitz 1997; Jensen & Veloso 2000). The second point addresses a general problem of augmenting first order logic with time for temporal planning. This often leads to information "holes" (Bacchus & Ady 2001; Fox & Long 2003) caused by concurrent actions hiding the state of domain knowledge they are currently changing.

It may also be claimed that the manager has misunderstood the situation and should decompose the tasks into start and stop event actions. This, however, conflicts with the model of control. The manager is not able to stop tasks in the middle of their execution. A task is uninteruptable.

To model these domains, we have developed an action representation called Evolving Tasks (ETs). ETs are, as far as we know, the first action representation that can represent temporally extended activities which are non-deterministic both with respect to duration and effect. ETs are represented by unit time transition systems. They further define the value of the state variables they effect at every time point. This solves the problem mentioned under point two above. The explicit representation of time and intermediate states of ETs also address another requirement to our task representation: it must be easy to access for humans participating in the model adaptation process. To this end ETs can be represented visually as graphs.

We consider multi-agent planning domains where each agent is defined by a set of ETs it can apply. We call these ASynchronous Evolving Tasks (ASET). Like NADL (Jensen & Veloso 2000), ASET explicitly model the environment as a set of uncontrollable agents (the customers).

In this paper, we formally define an ASET description and its low-level semantics (*unit time transition graph*) and high-level semantics (*decision graph*). The decision graph is a non-deterministic planning domain which allows us to define solutions to ASET planning problems as strong, strong cyclic, and weak plans (Cimatti *et al.* 2003). These plans can be efficiently generated by state-of-the-art symbolic non-deterministic planning systems (Cimatti *et al.* 2003; Jensen, Veloso, & Bryant 2003). To this end, we define a Boolean encoding of the unit time transition graph and a transformation of this representation into a Boolean encoding of the decision graph based on iterative squaring (Burch, Clarke, & McMillan 1990).

A major motivation for transforming an ASET description into a non-deterministic planning domain is to generate fault tolerant plans for ASET planning problems (Jensen, Veloso, & Bryant 2004). Fault tolerant planning is an alternative approach to probabilistic planning, where a plan with high probability of success is expressed as a plan with high level of tolerance for failures occurring during execution. This notion of fault tolerance is well-known from engineering and control theory and in contrast to probabilistic plans, fault tolerant plans bypass a complex manipulation of transition probabilities. We show how to augment an ASET description to obtain a fault tolerant planning domain such that fault tolerant plans can be generated via the strong planning algorithm.

The remainder of the paper is organized as follows. We first define ASET descriptions and discuss how they relate to other planning domain representations. We then present the unit time transition graph of an ASET description and its Boolean encoding and show how to transform the unit time transition graph into a decision graph which is a non-deterministic planning domain. The following section briefly reminds about the definition of strong non-deterministic plans and shows how to represent fault tolerant planning domains in ASET. Finally, we conclude and discuss plans for future work.

ASET Descriptions

An ASET description consists of a *global clock* with a finite domain of discrete integer time points, a disjoint set of system and environment *state variables* with finite domains, and a description of *system* and *environment agents*.

The state variables can be *metric* with finite integer domains, *Boolean*, or *enumerations* with finite domains. The usual arithmetic and relational operations can be carried out on metric variables. The set of state variable assignments defines the state space of the world.

An agent's description is a set of *tasks*. The agents change the state of the world by executing tasks. Each agent is always in a state of activity executing some task. The agents are asynchronous, they may start and stop tasks at different time-points. The system agents model the behavior of the agents controllable by the planner, while the environment agents model the uncontrollable world. To ensure independence of the system and environment agents, they affect a disjoint set of state variables. Their tasks, however, may depend on the complete state of the world.

A task has two parts: a set of *state variables* that the task modifies and a set of *unit time transitions* that defines how the task evolves. Intuitively, the task is responsible of assigning new values to the variables it modifies. It further has exclusive access to the modified variables, no other concurrent task can modify these variables as long as it is active. Each agent is associated with a finite set of execution states. These states are shared between the tasks of the agent and define the transition states of the tasks. Each set of execution states has a special *idle state*. A task is a transition system where each transition has unit time duration. The outgoing transitions from the idle state are taken when a task starts. The incoming transitions to the idle state are taken when the task stops. The remaining transitions of a task form a directed acyclic graph on the execution states causing all execution paths of the task to be finite. Each transition is *guarded*. The guard is an expression on the complete state. This may include the current task and execution state of any agent as well as the current value of any state variable. The transition is only enabled if the guard expression is satisfied. This allows rich behavior models including strong synchronization schemes with other tasks. The effect of the transition is given as an expression on the state variables it

modifies and its execution state. If this expression holds for several assignments, one of these is non-deterministically chosen as the effect of the transition. In this way, tasks are non-deterministic both with respect to duration and effect on modified variables. Notice that there is no need for an explicit precondition. The precondition of a task is the disjunction of the guards of outgoing transitions from the idle state.

Time advances in discrete integer time points. In each unit time step, the currently active tasks perform a unit time transition. Variables not modified by any task maintain their value. The resulting unit time transition graph will *block* if no transition is enabled for some task.

Example 1 Chris and Kim are cave divers. Chris is initially at the cave bottom at a depth of 20 yards with three units of oxygen and must *surface* before running out of oxygen. Kim is initially at the surface and can either *wait* one minute at the surface or *dive* for two minutes. The cave has a difficult passage at a depth of 10 yards. Chris' surface swim is non-deterministic, both with respect to time and amount of oxygen consumed. It may either take one or two minutes. If no problems are encountered, it takes one minute and two gallons of oxygen is consumed. Otherwise, it takes two minutes, and two gallons of oxygen is consumed in the first minute and either one or two gallons is consumed in the second minute. The Initial situation is shown in Figure 1(a).

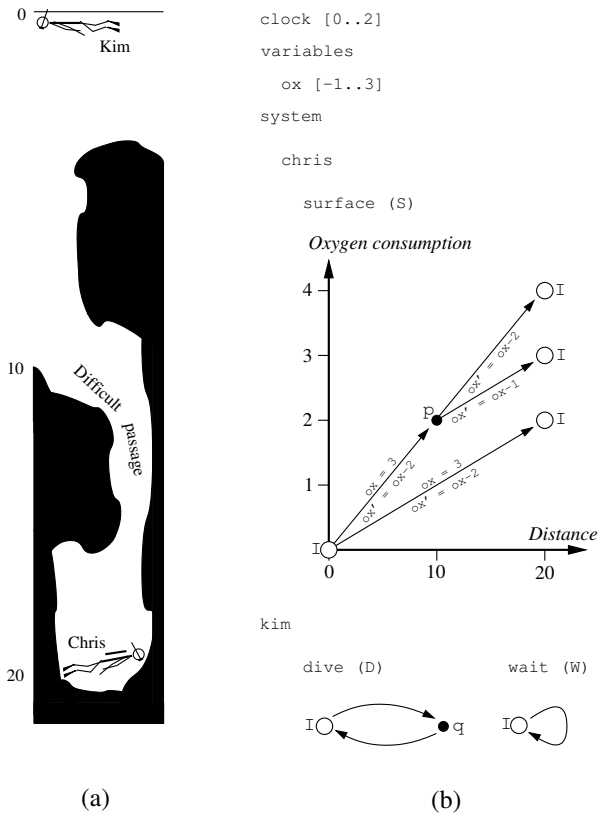

Figure 1: Initial situation (a) and ASET description (b) of the diver domain.

An ASET description of this scenario is shown in Figure 1(b). The domain of the clock is $\{0, \ldots, 2\}$ and there is a single state variable ox with domain $\{-1, 0, \ldots, 3\}$ representing the amount of oxygen Chris has left. To simplify the presentation, we do not include state variables representing the position of Chris and Kim, but it is straight forward to extend the example to include these. Chris and Kim are assumed to be controllable system agents. The execution states associated with Chris and Kim are $\{I, p\}$ and $\{I, q\}$, respectively. The idle state is marked `I`. The guard and effect expression of a transition is written above and below the transition arrow, respectively. To represent conditional effects, unprimed variables refer to the current state of the transition while primed variables refer to the next state. As for the surface task, it may be helpful to use several copies of the idle state, but each of these copies represents the same idle state.

Notice that different formats of the task descriptions are used to emphasize their meaning. For Chris' surface task, we choose to show the relation between traveled distance and oxygen consumption, while automata models are more appropriate for Kim's simple wait and dive tasks.

Figure 2 shows the eight different ways the scenario can evolve from the initial state. It can be seen, however, that no matter how tasks are chosen for Chris and Kim, it cannot be guaranteed that Chris has oxygen left when reaching the surface. In non-deterministic planning terms, we say that there is no *strong* plan for reaching the goal. This will be clarified in the following sections. ◇

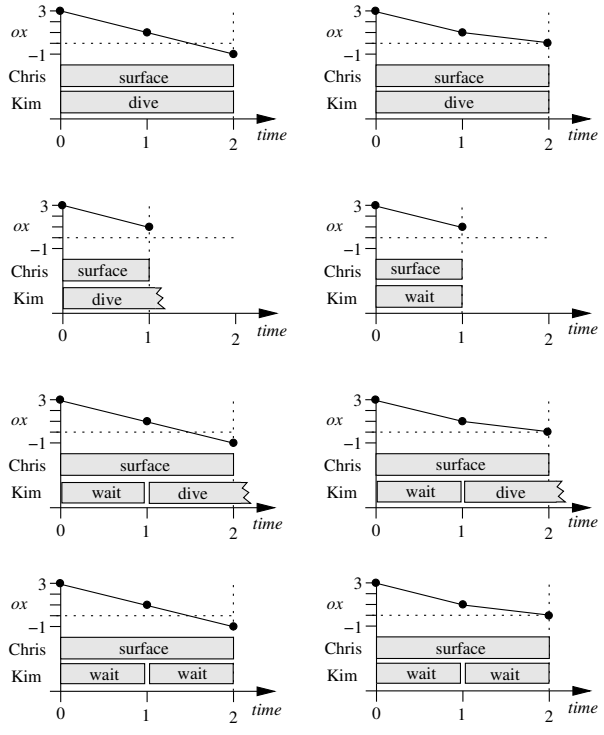

Figure 2: Eight possible evolutions of the diver scenario.

Formally, a ASET description is defined as follows.

Definition 1 (ASET Description) *An ASET description is a tuple $\mathcal{M} = \langle C, V, E, T \rangle$, where*

- C is a finite domain of a *global clock* $C = \{0, 1, \ldots, c_{max}\}$,
- V is a finite domain of n^s *system state variables* and n^e *environment state variables* $V = V^s \times V^e$ where $V^x = \prod_{i=1}^{n^x} V_i^x$ for $x \in \{s, e\}$,
- E is a finite *execution space* of $m^s > 0$ *system agents* and $m^e \geq 0$ *environment agents* each associated with a set of execution states $E = E^s \times E^e$ where $E^x = \prod_{i=1}^{m^x} E_i^x$ for $x \in \{s, e\}$. Each set of execution states includes a special idle state $E_i^x \supseteq \{\text{idle}\}$ for $x \in \{s, e\}$ and $i = 1 \ldots m^x$, and
- T is a finite *task space* of a non-empty set of tasks associated with each agent $T = \prod_{i=1}^{m^s} T_i^s \times \prod_{i=1}^{m^e} T_i^e$. Each task $t \in T_k^x$ is a pair $\langle M_t^x, R_t^x \rangle$, where
  - M_t^x is a set of indices of state variables modified by the task $M_t^x \subseteq \{1, \ldots, n^x\}$, and
  - R_t^x is a set of guarded unit time execution transitions of the task defining how modified variables are changed while the task is active $R_t^x \subseteq C \times V \times E \times T \times \prod_{i \in M_t^x} V_i^x \times E_k^x$.

Compared with the durative action descriptions of PDDL2.1, TLplan, and IxTeT (Fox & Long 2003; Bacchus & Ady 2001; Laborie & Ghallab 1995), the most significant difference of ASET descriptions is that tasks are durative *and* non-deterministic. None of the above domain descriptions consider non-deterministic actions. Actually, we are not aware of any planning language with temporally extended and non-deterministic actions. Another important difference between ASET and the domain descriptions above is the use of state variables. This provides metric values, but so has PDDL2.1. What is probably more important is that our state variables are defined at every time point like state variables in physics and control theory (Cassandras & Lafortune 1999). When augmenting first order logic with time and preserving the precondition and effect notions from classical planning, domain knowledge may only exist at certain time points. An important exception from this, however, are the continuous durative actions of PDDL2.1. For these actions, update functions are provided to define the change of metric information. This approach, however, is not as general as ETs.

Another challenge for durative actions in the classical precondition-effect format is how to handle conditional effects. The problem is that conditional effects require information to be transfered from the state the action is being applied in, to the state the action is completed in. These states, however, may not be adjacent in the planning domain. The problem can be solved by introducing memory propositions (Fox & Long 2003) or instantaneous effects of actions (Bacchus & Ady 2001). For ETs the problem is solved explicitly, since conditional effects can be defined for each unit time transition as shown in the diver example.

An important issue to address when introducing concurrent tasks is synergetic effects between simultaneously executing tasks (Lingard & Richards 1998). A common example of destructive synergetic effects is when two or more tasks require exclusive use of a single resource or when two tasks have inconsistent effects like $pos' = 3$ and $pos' = 2$.

Like actions in NADL, ASET tasks cannot be performed concurrently in the following two conditions: 1) they have inconsistent effects, or 2) they modify an overlapping set of state variables. The first condition is due to the fact that state knowledge is expressed in a monotonic logic that cannot represent inconsistent knowledge. The second condition addresses the problem of sharing resources. Consider for example two agents trying to drink the same glass of water. If only the first condition defined interfering tasks, both agents could simultaneously empty the glass, as the effect $glass\_empty$ of the two tasks would be consistent. With the second condition added, these tasks are interfering and cannot be performed concurrently.

We have chosen this definition of task interference due to our positive experience with it in NADL. There are, however, several issues to address. First, we need to show how to encode synergetic activity strong enough to solve Gelfond's soup problem (Gelfond, Lifschitz, & Rabinov 1991). The problem is to lift a soup bowl without spilling the soup. Two actions, lift left and lift right, can be applied to the bowl. If either is applied on its own the soup will spill, but if they are applied simultaneously then the bowl is raised from the table and no soup spills. The problem is that we cannot model the state of the soup bowl in ASET using just one state variable, since two concurrent lift tasks then would be unable to access that state variable. We can, however, represent such synergetic activity by letting the state of the bowl being expressed by several state variables. If we introduce two Boolean variables $force\_left$ and $force\_right$ the different states of the bowl can be represented by

$$\begin{aligned} onGround &= \neg force\_left \wedge \neg force\_right, \\ spill &= force\_left \text{ XOR } force\_right, \\ lift &= force\_left \wedge force\_right. \end{aligned}$$

Second, we need to address how to handle state variables that represent shared resources. In (Bacchus & Ady 2001) an example of a gas station with 6 refueling bays is given. If this resource is represented by a single state variable in ASET, we once more face the problem of at most one task accessing the resource. Again, we can solve the problem by using several state variables (e.g., a Boolean variable for each refueling bay).

ASET Unit Time Transition Graphs

In order to transform an ASET description into a non-deterministic planning domain, we first compute its *unit time transition graph*. The unit time transition graph is a transition system that represents the combined effect of active tasks. As the name suggests, each transition in the unit time transition graph advances the clock one time unit. There is not, however, necessarily a planning decision associated with each transition. It is only when one or more tasks of controllable system agents terminate that there is a choice

for the planner of which new tasks to start. We call such states *decision states*.

Example 2 Consider again the diver domain described in Example 1. The initial state and goal states is not a part of a ASET description, but we assume that $ox = 3$ in the initial state and the goal is to reach a state where Chris reaches the surface before running out of oxygen. Figure 3 shows the part of the unit time transition graph that can be reached from the initial state. Each state is represented by a triple and a pair $(ox, e_{Chris}, t_{Chris})$ and (e_{Kim}, t_{Kim}), where ox is the value of the ox state variable, and e and t are the execution state and current task of Chris and Kim. In the initial state, we choose *surface* and *wait* as dummy idle tasks. There are boxes around the goal states (i.e. states where Chris has surfaced successfully). Each of these are decision states. Additional decision states are marked by ellipses. Each path in the graph corresponds to one of the eight scenarios shown in Figure 2. ◇

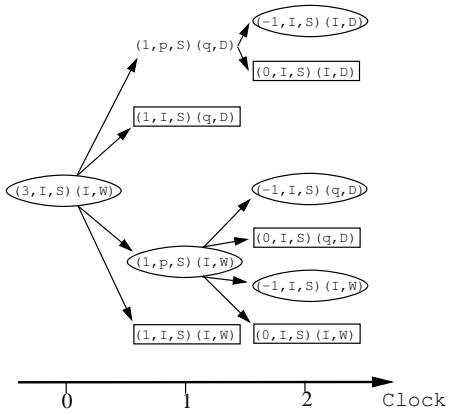

Figure 3: Unit time transition graph of the diver domain.

For an ASET description $\mathcal{M} = \langle C, V, E, T\rangle$, let NC denote non-conflicting tasks of system and environment agents. We have, $\text{NC} = \text{NC}^s \times \text{NC}^e$, where $\text{NC}^x = \{\langle t_1, \ldots, t_{m_x}\rangle \in T^x : M^x_{t_i} \cap M^x_{t_j} = \emptyset \text{ for } i \neq j\}$. We can now define the unit time transition graph of an ASET description as follows.

Definition 2 (Unit Time Transition Graph) *A unit time transition graph of an ASET description $\mathcal{M} = \langle C, V, E, T\rangle$ is a transition system $\mathcal{T} = \langle S_\mathcal{T}, R_\mathcal{T}\rangle$, where*

$S_\mathcal{T}$ is a finite set of states $S_\mathcal{T} = C \times V \times E \times NC$, and
$R_\mathcal{T}$ is a transition relation $R_\mathcal{T} \subseteq S_\mathcal{T} \times S_\mathcal{T}$.

For

$$\begin{aligned} s &= \langle c, v^s_{1..n^s}, v^e_{1..n^e}, e^s_{1..m^s}, e^e_{1..m^e}, t^s_{1..m^s}, t^e_{1..m^e}\rangle \\ s' &= \langle c', v'^s_{1..n^s}, v'^e_{1..n^e}, e'^s_{1..m^s}, e'^e_{1..m^e}, t'^s_{1..m^s}, t'^e_{1..m^e}\rangle \end{aligned}$$

We have $\langle s, s'\rangle \in R_\mathcal{T}$ iff

1. Running tasks transition,
   $\langle s, v'^x_{p(1)..p(n^x_{t'^x_i})}, e'^x_i\rangle \in R^x_{t'^x_i}$ for $x \in \{s, e\}, i = 1..m^x$,
   where $M^x_t = \{p(1), \ldots, p(n^x_t)\}$.
2. Non-idle tasks continue,
   $(e^x_i \neq \text{idle}) \Rightarrow (t'^x_i = t^x_i)$ for $x \in \{s, e\}$ and $i = 1..m^x$.
3. Unmodified variables maintain their value, and
   $v'^x_i = v^x_i$ for $x \in \{s, e\}$ and $i \in \{1, \ldots, m^x\} \setminus M$,
   where $M = \bigcup_{j=1}^{m^x} M^x_{t'^x_j}$.
4. Time advances.
   $$c' = c + 1.$$

In order to use symbolic non-deterministic planners to solve ASET planning problems, we need a Boolean encoding of unit time transition graphs. This is achieved by defining the *characteristic* function of the set of state pairs in $R_\mathcal{T}$ of the unit time transition graph. Let $\vec{s}$ and $\vec{s}'$ be two vectors of Boolean variables representing the current and next state of a unit time transition graph, where

$$\begin{aligned} \vec{s} &= \langle \vec{c}, \vec{v}^s_{1..n^s}, \vec{v}^e_{1..n^e}, \vec{e}^s_{1..m^s}, \vec{e}^e_{1..m^e}, \vec{t}^s_{1..m^s}, \vec{t}^e_{1..m^e}\rangle, \\ \vec{s}' &= \langle \vec{c}', \vec{v}'^s_{1..n^s}, \vec{v}'^e_{1..n^e}, \vec{e}'^s_{1..m^s}, \vec{e}'^e_{1..m^e}, \vec{t}'^s_{1..m^s}, \vec{t}'^e_{1..m^e}\rangle. \end{aligned}$$

Our goal is to define a Boolean function $R_\mathcal{T}(\vec{s}, \vec{s}')$ that is true iff the variables of $\vec{s}$ and $\vec{s}'$ are assigned values corresponding to a transition in $R_\mathcal{T}$. For an ASET description $\mathcal{M} = \langle C, V, E, T\rangle$, let r_i represent requirement i of Definition 2

$$\begin{aligned} r^x_1 &= \bigwedge_{i=1}^{m^x} \bigwedge_{t \in T^x_i} \Big[(\vec{t}'^x_i = t) \Rightarrow R^x_t(\vec{s}, \vec{v}'^x_{p(1)..p(n^x_t)}, \vec{e}'^x_i)\Big], \\ r^x_2 &= \bigwedge_{i=1}^{m^x} \Big[(\vec{e}^x_i \neq \text{idle}) \Rightarrow (\vec{t}'^x_i = \vec{t}^x_i)\Big], \\ r^x_3 &= \bigwedge_{i=1}^{n^x} [(\bigwedge_{j=1}^{m^x} \bigwedge_{t \in T^x_j(i)} \vec{t}'^x_j \neq t) \Rightarrow (\vec{v}'^x_i = \vec{v}^x_i)], \\ r_4 &= \vec{t}' = \vec{t} + 1, \end{aligned}$$

where $M^x_t = \{p(1), \ldots, p(n^x_t)\}$, $R^x_t(\vec{s}, \vec{v}'^x_{p(1)..p(n^x_t)}, \vec{e}'^x_i)$ is the characteristic function of the set of tuples in R^x_t, and $T^x_j(i) = \{t \in T^x_j : i \in M^x_t\}$.

Further, let NCdenote the non-conflicting tasks

$$NC^x = \bigwedge_{\substack{i \in D_1 \\ j \in D_2}} \bigwedge_{\substack{t_1 \in T^x_i \\ t_2 \in T^x_j}} \left[\begin{array}{l} D^x_{t_1} \cap D^x_{t_2} = \emptyset \Rightarrow \\ \neg(\vec{t}^x_i = t_1 \wedge \vec{t}^x_j = t_2) \end{array}\right].$$

where $D_1 = \{1, \ldots, m^x\}$ and $D_2 = \{1, \ldots, m^x\} \setminus \{i\}$.

We then have

$$R_\mathcal{T}(\vec{s}, \vec{s}') = r_4 \wedge \bigwedge_{x \in \{s,e\}} r^x_1 \wedge r^x_2 \wedge r^x_3 \wedge NC^x.$$

ASET Decision Graphs

We now consider how to transform the unit time transition graph of a ASET description into a non-deterministic planning domain that we can solve efficiently with a state-of-the-art symbolic non-deterministic planning system. The non-deterministic planning domains used by these systems are a generalization of classical deterministic planning domains where the effect of an action applied in some state is modeled by a non-deterministic choice from a set of possible next states.

Definition 3 (Non-Deterministic Planning Domain) *A non-deterministic planning domain is a tuple $\langle S, A, R\rangle$ where S is a finite set of states, A is a finite set of actions, and $R \subseteq S \times A \times S$ is a non-deterministic transition relation of action effects.*

A unit time transition graph is transformed into a non-deterministic planning domain by removing states where no planning decision can be made. As mentioned in previous section, a planning decision can be made in states where the task of one or more controllable agents is idle. Let $D_{\mathcal{T}}$ denote these *decision states* of a unit time transition graph $\mathcal{T} = \langle S_{\mathcal{T}}, R_{\mathcal{T}}\rangle$. We have $D_{\mathcal{T}} = \{\langle \ldots, e^s_{1..m^s}, \ldots\rangle \in S_{\mathcal{T}} : e^s_i = \text{idle for some } 1 \leq i \leq m^s\}$.

The non-deterministic planning domain of an ASET description, however, also needs to include *blocking states* where some task is unable to transition. Without including these states we may get an incorrect model that hides the fact that some decision may lead to a dead end (e.g., causing two tasks to "wait" on each other). Let $B_{\mathcal{T}}$ denote the blocking states of a unit time transition graph $\mathcal{T} = \langle S_{\mathcal{T}}, R_{\mathcal{T}}\rangle$. We have $B_{\mathcal{T}} = \{s \in S_{\mathcal{T}} : \langle s, s'\rangle \notin R_{\mathcal{T}} \text{ for all } s' \in S_{\mathcal{T}}\}$.

The non-deterministic planing domain associated with an ASET description is called a *decision graph*. Each transition in the decision graph corresponds to a path between decision states and blocking states in the unit time transition graph. For a set of states Q and a transition relation $U \subseteq Q \times Q$ a *path* of *length* k from v to w is a sequence of states $q_0 q_1 \cdots q_k$ such that $(q_i, q_{i+1}) \in U$ for $i = 0, \ldots, k-1$ and $v = s_0$ and $w = s_k$. We can now define the decision graph as follows.

Definition 4 (ASET Decision Graph) *Given an ASET description $\mathcal{M} = \langle C, V, E, T\rangle$ and a unit time transition graph $\mathcal{T} = \langle S_{\mathcal{T}}, R_{\mathcal{T}}\rangle$ of $\mathcal{M}$, an ASET decision graph of $\mathcal{M}$ is a non-deterministic planning domain $\mathcal{D} = \langle S, A, R\rangle$, where*

- S is the union of the decision and blocking states $S = D_{\mathcal{T}} \cup B_{\mathcal{T}}$,
- A is a finite set of actions $A = 2^{T^s}$, and
- R is a transition relation $R \subseteq S \times A \times S$.

For

$$\begin{aligned} s &= \langle c, v^s_{1..n^s}, v^e_{1..n^e}, e^s_{1..m^s}, e^e_{1..m^e}, t^s_{1..m^s}, t^e_{1..m^e}\rangle \\ s' &= \langle c', v'^s_{1..n^s}, v'^e_{1..n^e}, e'^s_{1..m^s}, e'^e_{1..m^e}, t'^s_{1..m^s}, t'^e_{1..m^e}\rangle \end{aligned}$$

We have $\langle s, a, s'\rangle \in R$ iff

- there exists a path $s_0 \cdots s_k$ in $R_{\mathcal{T}}$ between $s = s_0$ and $s' = s_k$ not visiting other states in S ($s_i \notin S$ for $i = 1, \ldots, k-1$), and
- the action a is the set of system tasks started in s ($a = \bigcup_{e^s_i = \text{idle}} \{t'^s_i\}$).

If $\langle s, a, s'\rangle$ is a transition in a decision graph, the current state s is a decision state and the next state s' is the first decision state or blocking state reached by some path from s when starting the tasks defined by a in the current state s.

Example 3 Figure 4 shows the decision graph of the unit time transition graph of the diver domain shown in Figure 3. Again, we only consider the subset of transitions reachable from the initial state. The decision graph has only one state less than the unit time transition graph. The reason is that the diver example for sake of presentation only has tasks with short duration. For more realistic domains, where most tasks will have longer duration, a much larger fraction of states will be abstracted away.

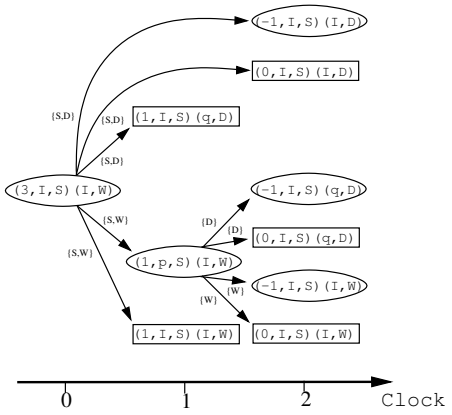

Figure 4: Decision graph of the diver domain.

The diver domain does not have any blocking states. We can introduce one by changing the *surface* task to the one depicted in Figure 5. The difference is that the two top right transitions are guarded by Kim executing the *dive* task and being in execution state q. This results in a blocking state if Kim decides to *wait*. Figure 6 shows the unit time transition graph of the domain. The blocking state is indicated by a sharp edged box. Figure 7 shows the corresponding decision graph. ◇

It is not obvious how to compute the decision graph, since it is defined in terms of paths in the unit time transition graph. For symbolic non-deterministic planning, though, the decision graph can be efficiently computed using *iterative squaring* (Burch, Clarke, & McMillan 1990).
Let

$$\begin{aligned} B(\vec{s}) &= \neg\big[\exists \vec{s}' \,.\, R_{\mathcal{T}}(\vec{s}, \vec{s}')\big], \text{ and} \\ D(\vec{s}) &= \bigvee_{i=1}^{m^s} \vec{t}^{\,s}_i = \text{idle} \end{aligned}$$

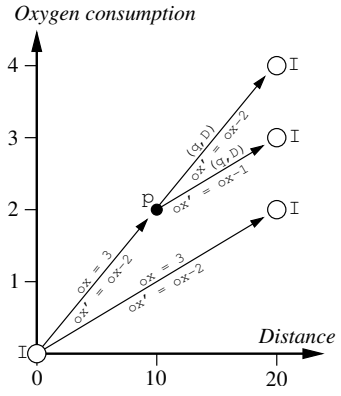

Figure 5: The blocking *surface* task.

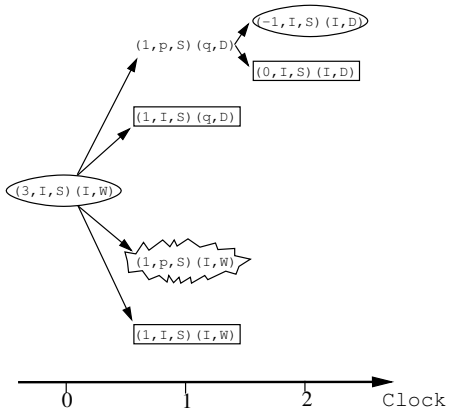

Figure 6: The unit time transition graph of the diver example with the *surface* task shown in Figure 5.

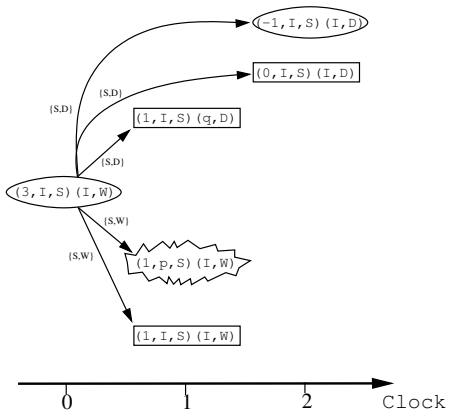

Figure 7: The decision graph of the unit time transition graph shown in Figure 6.

denote the characteristic functions for the set of blocking states and decision states of a unit time transition graph with Boolean encoding $R_{\mathcal{T}}(\vec{s},\vec{s}')$. Further, let $R^i(\vec{s},\vec{s}')$ be defined recursively by

$$\begin{aligned} R^0_{\mathcal{T}}(\vec{s},\vec{s}') &= R_{\mathcal{T}}(\vec{s},\vec{s}'), \\ R^i_{\mathcal{T}}(\vec{s},\vec{s}') &= R^{i-1}_{\mathcal{T}}(\vec{s},\vec{s}') \vee \Big(\exists \vec{s}' \,.\, R^{i-1}_{\mathcal{T}}(\vec{s},\vec{s}') \wedge \\ &\quad \neg(D(\vec{s}') \vee B(\vec{s}')) \wedge R^{i-1}_{\mathcal{T}}(\vec{s}',\vec{s}'')\Big)[\vec{s}''/\vec{s}'], \\ &\quad \text{for } i > 0. \end{aligned}$$

The operator $e[\vec{s}''/\vec{s}']$ renames double primed variables to single primed variables in the expression e. $R^0_{\mathcal{T}}$ is the transition relation of the unit time transition graph. $R^1_{\mathcal{T}}$ includes the transitions of R^0, but adds a transition $\langle s, s''\rangle$ for every path $ss's''$ where s' neither is a blocking state or decision state. Similarly R^2 adds transitions that may bypass two such states, and R^3 adds transitions that may bypass four etc.. In this way, we can define a Boolean encoding of the decision graph as

$$\begin{aligned} R(\vec{s},\vec{s}') &= R^{\lceil \log d \rceil}_{\mathcal{T}}(\vec{s},\vec{s}') \wedge (D(\vec{s}) \vee B(\vec{s})) \wedge \\ &\quad (D(\vec{s}') \vee B(\vec{s}')) \end{aligned}$$

where d is the maximal duration of any task. Iterative squaring is known to be computationally complex. In our case, though, we only need to iterate to "compress" paths of length d, which often will be much less than the diameter of the transition graph. In addition, iterative squaring has been shown to be fairly efficient for transition systems dominated by clock counting (Gabodi *et al.* 1997).

Solving ASET Planning Problems

The transformation of an ASET description to a non-deterministic planning domain and the Boolean encoding of the decision graph, allows us to use efficient symbolic non-deterministic planning algorithms (Cimatti *et al.* 2003; Jensen & Veloso 2000) including heuristic symbolic search algorithms (Jensen, Veloso, & Bryant 2003) to solve ASET planning problems. In the remainder of this section, we apply the machinery developed for non-deterministic symbolic planning to define ASET planning problems and solutions.

Definition 5 (Non-Deterministic Planning Problem) *A non-deterministic planning problem is a tuple $\langle \mathcal{D}, s_0, G\rangle$ where $\mathcal{D}$ is a non-deterministic planning domain, s_0 is an initial state, and $G \subseteq S$ is a set of goal states.*

For a non-deterministic planning domain $\mathcal{D} = \langle S, A, R\rangle$, the set of possible next states of an action a applied in state s is given by $\textsc{Next}(s,a) \equiv \{s' : \langle s, a, s'\rangle \in R\}$. An action a is called *applicable* in state s iff $\textsc{Next}(s,a) \neq \emptyset$. The set of applicable actions in a state s is given by $\textsc{App}(s) \equiv \{a : \textsc{Next}(s,a) \neq \emptyset\}$. A non-deterministic plan is a set of *state-action pairs* (SAs).

Definition 6 (Non-Deterministic Plan) *Let $\mathcal{D}$ be a non-deterministic planning domain. A non-deterministic plan for $\mathcal{D}$ is set of state-action pairs $\{\langle s, a\rangle : a \in \textsc{App}(s)\}$.*

The set of SAs define a function from states to sets of actions relevant to apply in order to reach a goal state. States

are assumed to be fully observable. An execution of a non-deterministic plan is an alternation between observing the current state and choosing an action to apply from the set of actions associated with the state. Notice that the definition of a non-deterministic plan does not give any guarantees about goal achievement. The reason is that, in contrast to deterministic plans, it is natural to define a range of solutions classes. We are particularly interested in strong plans that guarantee goal achievement in a finite number of steps. Following (Cimatti *et al.* 2003), we define strong plans formally by as a CTL formula that must hold on a Kripke structure representing the execution behavior of the plan.

A set of states *covered* by a plan π is $\text{STATES}(\pi) \equiv \{s : \exists a \,.\, \langle s, a\rangle \in \pi\}$. The set of actions in a plan π associated with a state s is $\text{ACT}(\pi, s) \equiv \{a : \langle s, a\rangle \in \pi\}$. The *closure* of a plan π is the set of possible end states $\text{CLOSURE}(\pi) \equiv \{s' \notin \text{STATES}(\pi) : \exists \langle s, a\rangle \in \pi \,.\, s' \in \text{NEXT}(s, a)\}$.

Definition 7 (Execution Model) *An execution model with respect to a non-deterministic plan π for the domain $\mathcal{D} = \langle S, A, R\rangle$ is a Kripke structure $\mathcal{M}(\pi) = \langle Q, U\rangle$ where*

- $Q = \text{CLOSURE}(\pi) \cup \text{STATES}(\pi) \cup G$,
- $\langle s, s'\rangle \in U$ *iff* $s \notin G,\ \exists a \,.\, \langle s, a\rangle \in \pi$ *and* $\langle s, a, s'\rangle \in R$, *or* $s = s'$ *and* $s \in \text{CLOSURE}(\pi) \cup G$.

Notice that all execution paths are infinite which is required in order to define solutions in CTL. If a state is reached that is not covered by the plan (e.g., a goal state or a dead end), the postfix of the execution path from this states is an infinite repetition of it. Given a Kripke structure defining the execution of a plan, strong plans are defined by the CTL formula below.

Definition 8 (Strong Plans) *Given a non-deterministic planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, G\rangle$ and a plan π for $\mathcal{D}$, π is a strong plan iff $\mathcal{M}(\pi), s_0 \models \texttt{AF}\, G$.*

The expression $\mathcal{M}(\pi), s_0 \models \texttt{AF}\, G$ is true if all execution paths in lead to a goal state in a finite number of steps.

Example 4 There exists three non-deterministic plans for the decision graph of the diver problem shown in Figure 4.

$$
\begin{aligned}
p_1 &= \{\langle (3, I, S)(I, W), \{S, D\}\rangle\} \\
p_2 &= \{\langle (3, I, S)(I, W), \{S, W\}\rangle, \langle (1, p, S)(I, W), \{D\}\rangle\} \\
p_3 &= \{\langle (3, I, S)(I, W), \{S, W\}\rangle, \langle (1, p, S)(I, W), \{W\}\rangle\}
\end{aligned}
$$

None of these, however, are guaranteed to reach a goal state. Thus, as mentioned earlier, there does not exist a strong plan for the diver problem. ◇

Fault Tolerance

A weakness of strong plans is that they can be very conservative. In real-world domains most actions may fail. If fault behavior is modeled by non-determinism, a strong plan only exists if the worst case behavior of the plan, where all actions fail, still leads to a goal state. This is seldom the case. Instead it is natural to suggest a weaker requirement, which is to guarantee goal achievement only if no more than n actions fail during execution. Such plans are called n-fault tolerant plans (Jensen, Veloso, & Bryant 2004). Fault tolerant plans can be computed via strong plans by adding fault counters to the domain. This is also possible for ASET domains.

We define a failure of a task as a unit time transition leading to the idle state. In order to generate n-fault tolerant plans, we add a special fault counter state variable f_i for each controllable agent i. For each task of agent i that can fail, we extend the guard and effect of each unit time transition denoting failure with the expression $n > \sum_{i=1}^{m_s} f_i$ and $f_i' = f_i + 1$, respectively. For the remaining transitions of the task, we maintain the value of f_i by extending the effect with $f_i' = f_i$. Finally, the initial state is extended with $f_i = 0$ for $i = 1 \ldots m^s$ and the goal states are extended with $n \geq \sum_{i=1}^{m_s} f_i$.

In this way failures can only happen in the fault extended problem if less than n failures have occurred so far. This is precisely the assumption of n-fault tolerant plans and ensures that a strong plan of the fault extended problem is a valid n-fault tolerant plan.

Example 5 Consider a fault extended version of the diver problem for generating 1-fault tolerant plans via strong planning. In this domain, we assume that the top-most unit transition of the *surface* task is due to an unlikely equipment failure. The new task is shown in Figure 8. Since Kim's *wait* and *dive* tasks are successful, it is sufficient just to add a single fault counter f for Chris. The initial state is extended with $f = 0$ and the goals states are extended with $1 \geq f$.

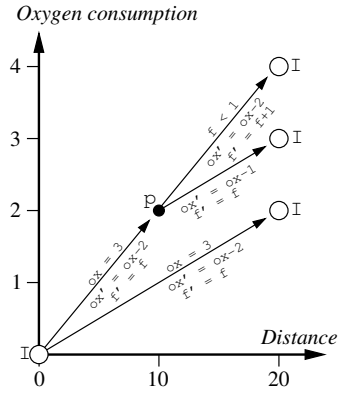

Figure 8: The surface task of the fault extended diver problem.

The decision graph of the fault extend problem is shown in Figure 9. The value of f has been added to the front of the first state tuple. Since the structure of the fault extended decision graph is identical to the decision graph of the original problem shown in Figure 4, there does not exist a strong plan and thus no 1-fault tolerant plan for this problem. Consider, however, that the initial state is a state on an execution path where one failure has already happened. Since the fault transition of the *surface* task now can be assumed not to happen, we get the new decision graph shown in Figure 10. The non-deterministic plan $\{\langle (1, 3, I, S)(I, W), \{S, D\}\rangle$ is a strong plan for this problem. Thus, under these assumptions, there exists a 1-fault tolerant plan. ◇

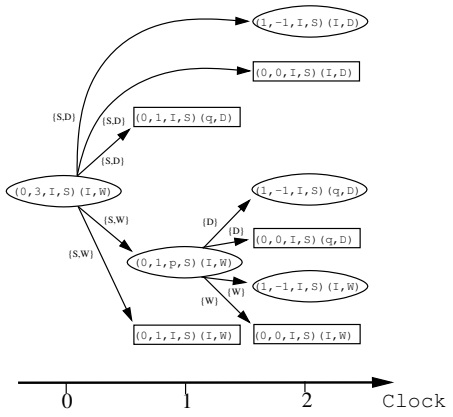

Figure 9: Decision graph of the fault extended diver problem.

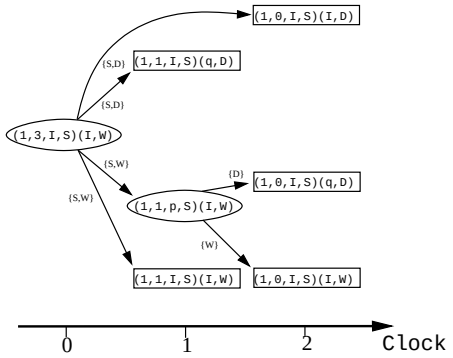

Figure 10: Decision graph of the fault extended diver problem assuming that one fault already has occurred.

Model Adaptation

ASET descriptions support model adaptation learning algorithms by providing a low-level definition of task behavior. Adapting the behavior of tasks to a real-world domain is a structural learning problem, where transitions are added or removed from tasks. In addition, guard and effect expressions can be modified and task descriptions split. The idea is to make a system more controllable by getting rid of non-determinism.

Example 6 Assume that it has been learned that Kim always helps Chris when choosing to dive. In this case, at most one unit of oxygen is consumed in the second time unit of the *surface* task. The learned surface task is shown in Figure 11 and the corresponding decision graph is shown in Figure 12. It is now possible always to save Chris if Kim decides to *dive* (e.g., $\{\langle(1,3,I,S)(I,W),\{S,D\}\rangle$ is a strong plan for the problem). $\diamond$

Conclusion

In this paper, we have introduced a new multi-agent planning domain representation called ASET. The main contribution of ASET is Evolving Tasks (ETs). ETs are, as far as we know, the first action representation that can represent temporally extended activities which are non-deterministic both with respect to duration and effect. ETs are represented as unit time transition systems that in a natural way solves the problem of representing conditional effects and intermediate effects of durative actions. In addition, ETs are explicit and structural representations that are easy to access for humans and suitable for domain knowledge learning algorithms.

We have formally defined ASET descriptions and shown how they can be transformed into non-deterministic planning domains. Using a Boolean encoding of these domains, efficient symbolic non-deterministic planning algorithms can be applied to ASET planning problems.

The ASET representation is currently being integrated as the basic building block in a closed loop planning, execution, and learning framework. Our goal is to develop algorithms that adapt ASET descriptions to real-world domains by learning new domain knowledge from execution failures.

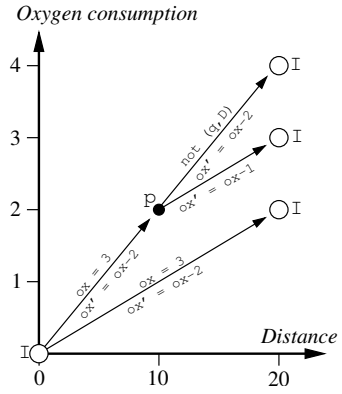

Figure 11: The learned surface task.

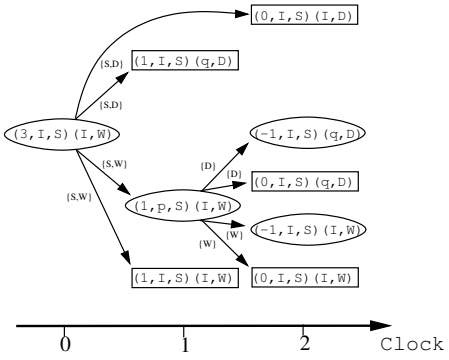

Figure 12: The decision graph of the diver problem with the learned surface task.

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *International Joint Conference on Artificial Intelligence (IJCAI-01)*, 417–424.

Burch, J. R.; Clarke, E. M.; and McMillan, K. 1990. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, 428–439.

Cassandras, C. G., and Lafortune, S. 1999. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence* 147(1-2). Elsevier Science publishers.

Firby, R. J. 1989. Adaptive execution in complex dynamic worlds. Technical Report YALEU/CSD/RR 672, Yale University.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Gabodi, G.; Camurati, P.; Lavagno, L.; and Quer, S. 1997. Disjunctive partitioning and partial iterative squaring. In *Proceedings of the 34th Design Automation Conference DAC-97*.

Gelfond, M.; Lifschitz, V.; and Rabinov, A. 1991. What are the limitations of teh situation calculus. In *Essays in Honor of Woody Bledsoe*. Kluwer Academic. 167–179.

Georgeff, M., and Lansky, A. L. 1986. Procedural knowledge. *Proceedings of IEEE* 74(10):1383–1398.

Giunchiglia, E.; Kartha, G. N.; and Lifschitz, Y. 1997. Representing action: Indeterminacy and ramifications. *Artificial Intelligence* 95:409–438.

Haigh, K. Z., and Veloso, M. M. 1998. Planning, execution and learning in a robotic agent. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, 120–127. AAAI Press.

Jensen, R. M., and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research* 13:189–226.

Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2003. Guided symbolic universal planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling ICAPS-03*, 123–132.

Jensen, R. M.; Veloso, M. M.; and Bryant, R. E. 2004. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling ICAPS-04*.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of (IJCAI-95)*, 1643–1649.

Lingard, A. R., and Richards, E. B. 1998. Planning parallel actions. *Artificial Intelligence* 99:261–324.

Piergiorgio, B.; Bonet, B.; Cimatti, A.; Giunchiglia, E.; Golden, K.; Rintanen, J.; and Smith., D. E. 2002. The NuPDDL home page. `http://sra.itc.it/tools/mbp/#nupddl`.

Sabin, D., and Weigel, R. 1998. Product configuration frameworks-a survey. *IEEE Intelligent Systems* 13(4):42–49.

Simmons, R. 1994. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation* 10(1):34–42.

Wang, X. 1994. Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on AI Planning Systems*, 335–340.

Yolanda, G. 1992. *Acquiring Domain Knowledge for Planning by Experimentation*. Ph.D. Dissertation, Carnegie Mellon University. CMU-CS-92-175.