ASET: a Multi-Agent Planning Language with Nondeterministic Durative Tasks for BDD-Based Fault Tolerant Planning*

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

Abstract

In this paper, we introduce a multi-agent planning language called ASynchronous Evolving Tasks (ASET). The main contribution of ASET is a novel explicit representation of temporally extended tasks that may be nondeterministic both with respect to duration and effects. Moreover, ASET explicitly models the environment as a set of uncontrollable agents. We formally define ASET descriptions and their transformation to a nondeterministic planning domain. Using a Boolean encoding, fault tolerant planning problems specified in ASET can be solved efficiently with state-of-the-art BDD-based planning systems. Our preliminary experimental results show that the transformation of ASET domains to nondeterministic planning domains is computationally efficient even for ASET descriptions with a high level of temporal detail.

Introduction

The most important obstacle for widespread application of automated planning is lack of scalability. Since the complexity of planning grows with the representational power of the planning language, a good strategy for solving a planning problem efficiently is to use a planning language that is sufficient for representing the problem at hand but among such languages has least representational power.

For this reason, the goal for planning language developers is to expose the representational power of the language by providing intuitive and explicit ways to state abstract real-world phenomena. In addition, well designed high-level languages makes it possible to write short and elegant descriptions of a domain. They further improve the ability of planning systems to exploit structure in domains.

Today powerful planners exist for the STRIPS planning language e.g., (Hoffmann & Nebel 2001). But STRIPS assumes a single agent executing instantaneous and deterministic actions, while most real domains involve multiple asynchronous agents executing temporally extended stochastic actions. There is no simple way of modeling stochastic behavior of actions and multi-agent domains in STRIPS. Its representational power is too low.

A wide range of planning languages have been developed to address the deficiencies of STRIPS including temporal languages e.g., (Fox & Long 2003; Bacchus & Ady 2001; Laborie & Ghallab 1995), nondeterministic languages e.g., (Piergiorgio *et al.* 2002; Giunchiglia, Kartha, & Lifschitz 1997; Jensen & Veloso 2000) and probabilistic languages e.g., (Younes 2003). None of them, however, have simple explicit ways of describing domains that combine all the aspects of real-world domains mentioned above. In particular, we are not aware of any planning language with a single unified construct to define actions that are nondeterministic both with respect to effect and duration. Temporal planning languages have deterministic actions and nondeterministic planning languages do not consider durative actions.

The representational power of some of these languages e.g., (Younes 2003; Musliner, Durfee, & Shin 1993) and classical representations like discrete event systems, timed automata, and Markov Decision Processes (MDPs) is strong enough to model such domains. But it is often tedious and error prone to define domains in these formalisms due to the implicit representation of abstract phenomena. Furthermore, the representational power may be so high that the planning problems become unnecessarily hard to solve.

The research reported in this paper investigates how low we can go in representational power and still be able to define a language in which stochastic durative actions and multi-agent domains can be stated in a unified, intuitive, and explicit way. More specifically, we consider a language with the representation power of a nondeterministic planning domain (i.e., an MDP with no transition probabilities). Our motivation is that stationary policies for nondeterministic planning problems can be synthesized efficiently (Cimatti *et al.* 2003; Jensen, Veloso, & Bryant 2003) using techniques developed in formal verification based on Binary Decision Diagrams (BDDs) (Bryant 1986; Burch, Clarke, & McMillan 1990).

Continuous time and probabilistic models are attractive, but come with a high computational fee. It is well-known that continuous time verification of asynchronous circuits is much harder than discrete time verification of synchronized circuits, and even though efficient symbolic techniques ex-

*This research is sponsored by BBNT Solutions LLC under its prime contract number FA8760-04-C-0002 with the U.S. Air Force and DARPA. 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.

ist for solving MDPs (Hansen & Zilberstein 2001), it is our experience that nonprobabilistic versions of these problems have orders of magnitude lower complexity.

We are interested in high-level planing problems where the goal is to coordinate low-level activities and manage shared resources. Such domains are often combinatorial and discrete in nature. Imagine an automated job shop floor with robots moving objects between machines and storage buffers. Commands to machines and robots are high-level, but fairly accurate models exist of the behavior they trigger. The main problem is to deliver and remove objects from machines in a temporally coordinated manner and share resources such as space.

Our language is based on an action representation called Evolving Tasks (ETs). ETs are Directed Acyclic Graphs (DAGs) of guarded unit time transitions that define the temporal behavior of the task. They can represent temporally extended activities which are nondeterministic both with respect to duration and effect. We consider multi-agent planning domains where each agent is defined by the set of ETs it can execute. The resulting language is called ASynchronous Evolving Tasks (ASET). Like NADL (Jensen & Veloso 2000), ASET explicitly model the environment as a set of uncontrollable agents.

The low-level semantics of an ASET domain is a *unit time transition graph*. The domain, however, is not controllable at this level since tasks are uninterruptible. A *decision graph* is derived from the unit time transition graph by adding transitions between all states where some task is idle and removing all other states from the unit time transition graph. This can be done efficiently using a technique called *iterative squaring* (Burch, Clarke, & McMillan 1990). The decision graph is a nondeterministic planning domain that 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 nondeterministic planning systems (Cimatti *et al.* 2003; Jensen, Veloso, & Bryant 2003).

Using this bottom-up approach, it is easy to define low-level temporal properties of the activities, but plan in a more abstract space. PDDL2.1 and other temporal languages extending STRIPS are based on a top-down approach where many features are used to define the temporal properties of actions. The result is less general languages with temporal semantics of actions that can be hard to understand. The unit time semantics of ETs further solve a general problem of augmenting first order logic with time for temporal planning (Bacchus & Ady 2001; Fox & Long 2003). This often leads to information "holes" caused by concurrent actions hiding the state of domain knowledge they are currently changing. This makes it hard to write domains with mutually dependent asynchronous activities.

The main limitation of ASET is the lack of transition probabilities. Often, however, stochastic behavior is caused by infrequent system failures. This allows us to avoid full-blown probabilistic planning and instead consider nondeterministic plans robust to a limited number of system failures (Jensen, Veloso, & Bryant 2004). Another limitation is that ASET assumes full observability. But this is a reasonable assumption for systems that are engineered to be highly controllable.

We have implemented a BDD-based planning system for ASET. Preliminary experimental results show that a unit time transition graph can be efficiently transformed into decision graph even when the duration of tasks is in the order of 500 time units. This level of temporal granularity is more than sufficient for most applications.

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. The following section briefly reminds about the definition of strong nondeterministic plans and shows how to represent a fault tolerant planning domain in ASET. We then present our experimental results and finally draw conclusions and discuss plans for future work.

ASET Descriptions

An ASET description consists of 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 for 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*. Each transition of a task 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 DAG 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 tasks of other agents. 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 nondeterministically chosen as the effect of the transition. In this way, tasks are nondeterministic 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.

As an example consider the simple job shop domain shown in Figure 1. The domain has four locations a, b, c, and d connected with corridors ab, bd, cd, and ac. The goal is to paint the object (O). It can be carried by the robot (R) to the painting machine (P). The robot spends time navigating between corridors and may have to backtrack to its source location. The robot and painting machine are controllable, but there is also an uncontrollable human operator (H). For security reasons, the robot is not allowed to load and unload the object when the human is at the same location.

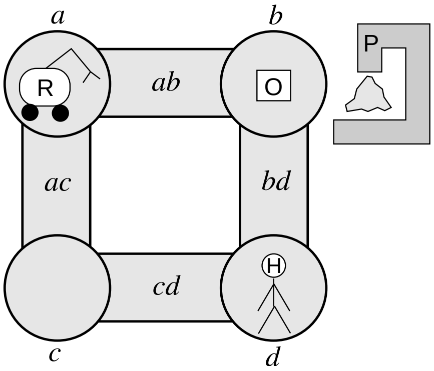

Figure 1: The job shop domain.

Figure 2 shows an ASET description of the job shop domain. Each task is a DAG where vertices are execution states and edges are unit time transitions. The idle execution state is marked by a double circle. Execution states are labeled by numbers (by convention idle states are labeled by zero, but these labels have been omitted to enhance readability). The guard expression or precondition of a unit time transition is shown above the associated edge. The effect of the transition is shown below the edge. The ASET description has two controllable system agents the robot (R) and the painting machine (P). It also has a single uncontrollable environment agent which is the human operator (H). The tasks of the robot are $move(x, y)$[1], $take$, put, and $wait$. During the move task, the robot navigates between the locations via the

[1]The figure shows $move(a, b)$, but there are 7 other move tasks.

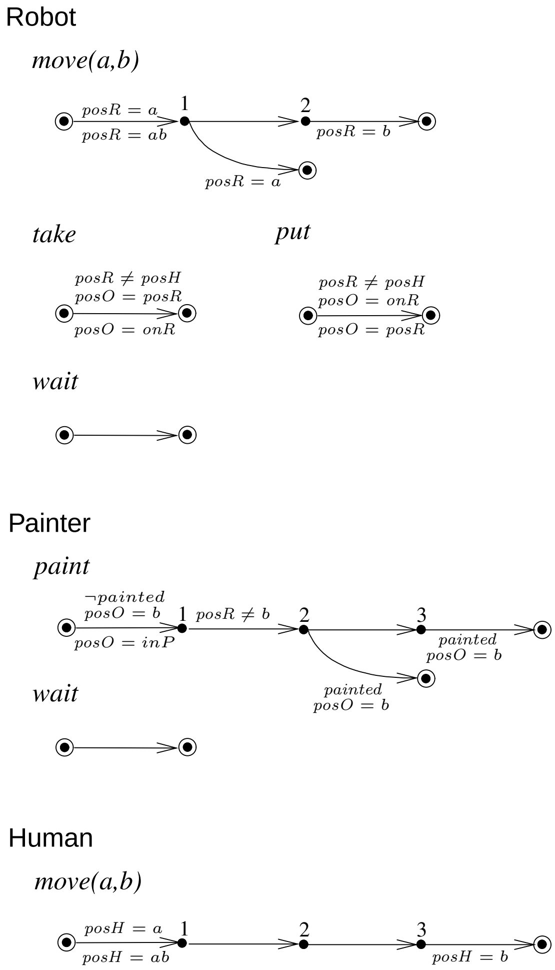

Figure 2: An ASET description of the job shop domain.

corridors. It may succeed after 2 time units and reach its destination, or fail after 3 time units in which case, it returns to the source location. The take and put tasks loads and unloads the object on the robot. They take one time unit and are conditioned by the human being at another location. The wait task also takes one time unit. It does not change any state variables, but only advances time to coordinate the robots activities with other agents. The tasks of the painter are $paint$ and $wait$. The paint task takes either 3 or 4 time units and requires that the object is at location b and is unpainted. Moreover, the robot must avoid location b when the actual painting happens. The wait task of the painter is identical to the wait task of the robot. The human has move tasks similar to the robot except that they are deterministic and have a duration of 4 time units. Since there is no wait task, the human must continuously move between locations. This guarantees that the robot eventually can load and unload the object.

Formally, an ASET description is defined as follows.

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

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 $i = 1 \,..\, m^x$, 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 $R_t^x \subseteq V \times E \times T \times \prod_{i \in M_t^x} V_i^x \times E_k^x$.

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* nondeterministic. None of the above domain descriptions consider nondeterministic actions. Actually, we are not aware of any planning language with temporally extended and nondeterministic 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 job shop 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 eat the same ice cream. If only the first condition defined interfering tasks, both agents could simultaneously eat the ice cream, as the effect $iceCreamEaten$ 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 at a time. 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 nondeterministic 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.

Consider again the job shop domain shown in Figure 2. Assume that all agents are in an idle execution state in the situation depicted at the top of the figure. Suppose that the tasks $move(a, c)$ and $paint$ are chosen for the robot and painter, and that the human happens to choose $move(d, c)$.[2] Figure 3 shows the reachable states in the unit time transition graph from this state until some agents are idle again. Each state is labeled with a vector showing the execution state of the robot, painter, and human respectively.

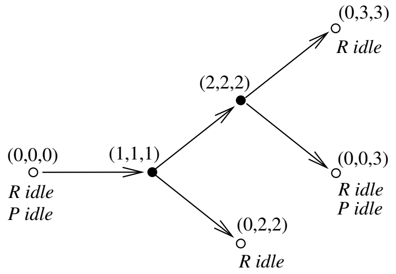

Figure 3: A subset of the unit time transition graph of the job shop domain.

For an ASET description $\mathcal{M} = \langle V, E, T\rangle$, let NC denote nonconflicting tasks of system and environment agents. We have, $\text{NC} = \text{NC}^s \times \text{NC}^e$, where $\text{NC}^x = \{\langle t_1, \ldots, t_{m_x}\rangle \in T^x : M^x_{t_i} \cap M^x_{t_j} = \emptyset$ 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 $\mathcal{M} = \langle V, E, T\rangle$ is a transition system $\mathcal{T} = \langle S_\mathcal{T}, R_\mathcal{T}\rangle$, where*

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

For

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

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 $M = \bigcup_{j=1}^{m^x} M^x_{t'^x_j}$.

In order to use symbolic nondeterministic 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

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

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 $\mathcal{M} = \langle V, E, T\rangle$, let r_i represent requirement i of Definition 2

$$r^x_1 = \bigwedge_{i=1}^{m^x} \bigwedge_{t \in T^x_i} \Big[(\vec{t}'^x_i = t) \Rightarrow R^x_t(\vec{s}, \vec{v}'^x_{p(1)..p(n^x_t)}, \vec{e}'^x_i)\Big]$$

where $M^x_t = \{p(1), \ldots, p(n^x_t)\}$ and $R^x_t(\vec{s}, \vec{v}'^x_{p(1)..p(n^x_t)}, \vec{e}'^x_i)$ is the characteristic function of the set of tuples in R^x_t,

$$r^x_2 = \bigwedge_{i=1}^{m^x} \Big[(\vec{e}^x_i \neq \text{idle}) \Rightarrow (\vec{t}'^x_i = \vec{t}^x_i)\Big],$$

$$r^x_3 = \bigwedge_{i=1}^{n^x} \big[(\bigwedge_{j=1}^{m^x} \bigwedge_{t \in T^x_j(i)} \vec{t}'^x_j \neq t) \Rightarrow (\vec{v}'^x_i = \vec{v}^x_i)\big]$$

where $T^x_j(i) = \{t \in T^x_j : i \in M^x_t\}$.

Further, let NC denote the non-conflicting tasks

$$NC^x = \bigwedge_{\substack{i \in D_1 \\ j \in D_2}} \bigwedge_{\substack{t_1 \in T^x_i \\ t_2 \in T^x_j}} \left[\begin{array}{l} D^x_{t_1} \cap D^x_{t_2} = \emptyset \Rightarrow \\ \neg(\vec{t}^x_i = t_1 \wedge \vec{t}^x_j = t_2) \\ \wedge\neg(\vec{t}'^x_i = t_1 \wedge \vec{t}'^x_j = t_2) \end{array}\right].$$

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

We then have

$$R_\mathcal{T}(\vec{s}, \vec{s}') = \bigwedge_{x \in \{s,e\}} r^x_1 \wedge r^x_2 \wedge r^x_3 \wedge NC^x.^3$$

ASET Decision Graphs

We now consider how to transform the unit time transition graph of an ASET description into a nondeterministic planning domain that we can solve efficiently with a state-of-the-art BDD-based nondeterministic planning system. The nondeterministic 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 nondeterministic choice from a set of possible next states.

[2] The result would have been the same for any of its move tasks.

[3] Since the finite domains of ASET variables are embedded in a binary encoding, there may exist assignments to the Boolean variables that do not correspond to valid domain values. Conjoining an expression that removes these assignments from the Boolean transition relation has been omitted in the definition to simplify the presentation.

Definition 3 (Nondeterministic Planning Domain) *A nondeterministic 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 nondeterministic transition relation of action effects.*

A unit time transition graph is transformed into a nondeterministic planning domain by removing states where no planning decision can be made. A planning decision can be made in states where the task of one or more controllable agents is idle. We call such states *decision states*. For unit time transition graph of the job shop domain shown in Figure 3, all unfilled circles (the end states) are decision states. 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 nondeterministic 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). In the job shop domain, any state where the robot is at location b and the painter is in execution state 1 of its paint task is a blocking state since the paint task is unable to transition due to the guard $posR \neq b$. 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 nondeterministic 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 $\mathcal{M} = \langle V, E, T\rangle$ 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 nondeterministic 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

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

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.

It is nontrivial to compute the decision graph, since it is defined in terms of paths in the unit time transition graph. For symbolic nondeterministic planning, though, the decision graph can be efficiently computed using *iterative squaring* (Burch, Clarke, & McMillan 1990). Iterative squaring of a transition relation introduces transitions between all states connected by a path. The operation is defined recursively. R^0 is the original transition relation. R^1 includes all the transition in R^0, but in addition has transitions between all states in R^0 connected by a path of length 2. R^2 includes all transitions in R^1, but in addition has transitions between all states in R^1 connected by a path of length 2. Since R^1 includes R^0 this means that R^2 includes all the transitions in R^0, but in addition has transitions between all states in R^0 connected by a path of length 2,3, or 4. Thus, for each squaring of the transition relation the length of the paths for which transitions are added doubles.

Consider squaring the unit time transition graph of the job shop domain shown in Figure 3. Figure 4 shows the transitions in R^2.

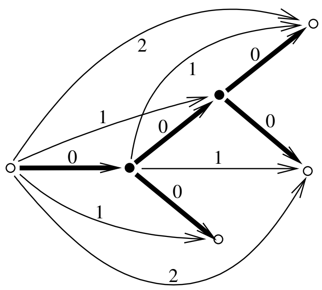

Figure 4: Squaring the unit time transition graph shown in Figure 3. Transitions in R^j for $j \leq i$ are labeled i.

We use a special version of the algorithm that ensures that all intermediate states on paths for which transitions are introduced are neither decision states nor blocking states. Let

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

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_T is the transition relation of the unit time transition graph. R^1_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 up to 3 such states, and R^3 adds transitions that may bypass 7 etc.. In this way, we can define a Boolean encoding of the decision graph as

$$R(\vec{s}, \vec{s}') = R_T^{\lceil \log d \rceil}(\vec{s}, \vec{s}') \wedge (D(\vec{s}) \vee B(\vec{s})) \wedge (D(\vec{s}') \vee B(\vec{s}'))$$

where d is the maximal duration of any task.

Figure 5 shows the decision graph of the unit time transition graph of the job shop domain shown in Figure 3.

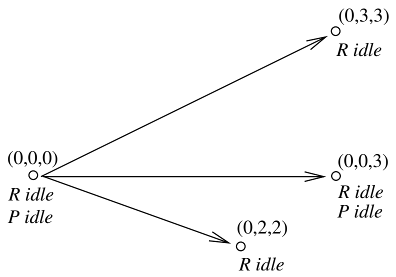

Figure 5: The decision graph of the unit time transition graph of the job shop domain shown in Figure 3.

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 digital systems dominated by clock counting (Gabodi *et al.* 1997). We may expect ASET domains where tasks have long duration to be structurally similar to this kind of circuits.

Solving ASET Planning Problems

The transformation of an ASET description to a nondeterministic planning domain and the Boolean encoding of the decision graph, allows us to use efficient symbolic nondeterministic 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 nondeterministic symbolic planning to define ASET planning problems and solutions.

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

For a nondeterministic 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 nondeterministic plan is a set of *state-action pairs* (SAs).

Definition 6 (Nondeterministic Plan) *Let $\mathcal{D}$ be a nondeterministic planning domain. A nondeterministic 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 nondeterministic 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 nondeterministic 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 $\textsc{States}(\pi) \equiv \{s : \exists a \,.\, \langle s, a \rangle \in \pi\}$. The set of actions in a plan π associated with a state s is $\textsc{Act}(\pi, s) \equiv \{a : \langle s, a \rangle \in \pi\}$. The *closure* of a plan π is the set of possible end states $\textsc{Closure}(\pi) \equiv \{s' \notin \textsc{States}(\pi) : \exists \langle s, a \rangle \in \pi \,.\, s' \in \textsc{Next}(s, a)\}$.

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

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

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 state 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 nondeterministic 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 lead to a goal state in a finite number of steps.

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 via nondeterminism, 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. We would like to be able to state a weaker kind of plans that do not have to cover the most unlikely execution paths. As mentioned in the introduction, going all the way to probabilistic planning is not a solution due to the high computational cost. But we can rephrase a plan with high probability of success as a plan with high tolerance for failures encountered during execution. Such plans can be defined fully within the framework of nondeterministic planning. Plans that guarantee goal achievement if no more than n actions fail during execution 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.[4] 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.

Experimental Evaluation

We have implemented a planning system in C++/STL using the BuDDy BDD package (Lind-Nielsen 1999). Given a textual ASET description, it computes two BDDs representing the transition relation of the unit time transition graph and its associated decision graph.

The experiment reported in this section investigates how fast the computational complexity of synthesizing the decision graph grows with the temporal granularity of the unit time decision graph. We consider a parameterized version of the job shop domain where each task is extended with extra unit time transitions such that the overall structure of the task is maintained. For instance, unit time transitions are added on both the left and right side of the early termination of the painter's paint task and the robot's move task. Since the number of possible ways that tasks can be temporally aligned grows polynomially with their duration, computing the decision graph could potentially be hard.

We conducted the experiments on a 3GHz Pentium 4 with 1024KB L2 cache and 2GB RAM [5] running Linux kernel 2.4.25. Figure 6 shows the computation time of the unit time transition graph and the decision graph. As depicted, the CPU time for computing the unit time transition graph is very low for all versions of the domain. Despite the much longer time needed to compute the decision graph, the asymptotic complexity of this operation is low. Notice the jumps in computation time when the iterative squaring involves computing a new intermediate transition relation.

[4]While this approach is conceptually easy to understand, much better performance can be achieved in real-world domains by distinguishing semantically between failure effects and successful effects and use specialized planning algorithms (Jensen, Veloso, & Bryant 2004).

[5]The experiments, however, were limited to 500MB RAM.

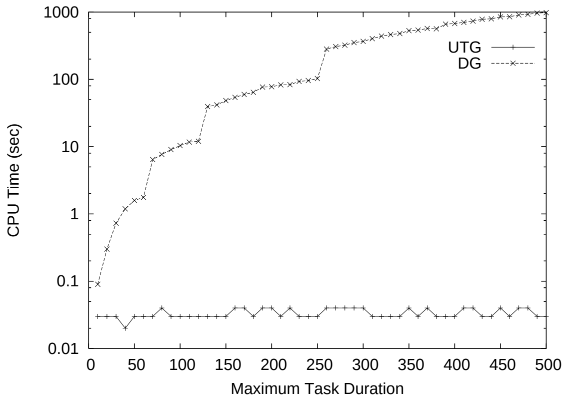

Figure 6: CPU time for synthesizing the unit time transition graph (UTG) and the decision graph (DG) as a function of maximum task duration.

Fortunately the distance between these jumps grows exponentially with the maximum task duration.

Figure 7 shows how the BDD size of the unit time transition graph and the decision graph grows as a function of the maximum task duration. As depicted, the BDD size of the decision graph grows approximately linearly with the computation time of the decision graph. It may be surprising that the BDD of the decision graph is larger than the BDD of the unit time transition graph. Since BDDs represent transitions implicitly, there is no simple relation between the size of a BDD and the number of transitions it represent. That the BDD representing the decision graph is large merely indicates that the subspace of transitions in the decision graph is less structured than that of the unit time decision graph. The question is to what extend this will impair BDD based planning based on the decision graph. Future experiments will address this issue.

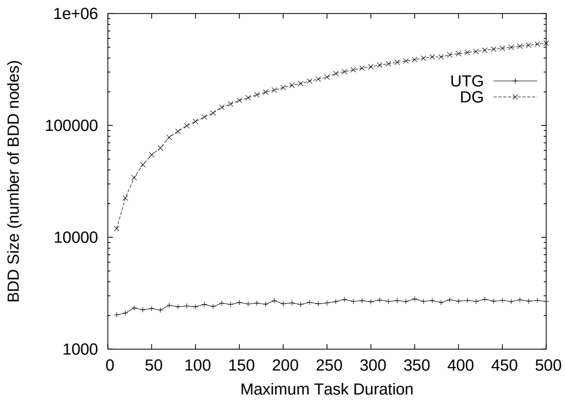

Figure 7: BDD size of the unit time transition graph (UTG) and the decision graph (DG) as a function of maximum task duration.

Conclusion

In this paper, we have introduced a new multi-agent planning language called ASET. The main contribution of ASET is Evolving Tasks (ETs). ETs are, as far as we know, the first action description that in an explicit and intuitive way can represent temporally extended activities which are nondeterministic both with respect to duration and effect. ETs are represented as directed acyclic graphs that in a natural way solves the problem of representing conditional effects and intermediate effects of durative actions.

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

ASET shows that it is possible to model essential aspects of time and stochastic behavior in a language with a representational power as low as a nondeterministic finite automata. This is encouraging since the main challenge of automated planning is to scale to the size of real-world domains, and since dense time and probabilistic models come with a high computational fee.

Preliminary results show that the decision graph of ASET domains can be generated efficiently even for domains with a high level of temporal detail. Future work includes further experiments investigating BDD-based planning based on ASET decision graphs and developing more efficient ways of generating and representing decision graphs (e.g., by using transition relation partitioning (Burch, Clarke, & Long 1991)).

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.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 8:677–691.

Burch, J.; Clarke, E.; and Long, D. 1991. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, 49–58. North-Holland.

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.

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.

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

Hansen, E., and Zilberstein, S. 2001. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

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.

Lind-Nielsen, J. 1999. BuDDy - A Binary Decision Diagram Package. Technical Report IT-TR: 1999-028, Institute of Information Technology, Technical University of Denmark. `http://cs.it.dtu.dk/buddy`.

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

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: A cooperative intelligent real time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6):1561–1574.

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`.

Younes, H. L. S. 2003. Extending PDDL to model stochastic decision processes. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03) Workshop on PDDL*, 95–103.