Symbolic Behavior-Recognition

Gal A. Kaminka and Dorit Avrahami
MAVERICK Group
Bar Ilan University, Israel
{avrahad1,galk}@cs.biu.ac.il

Abstract

It is important for robots to model other robots' unobserved actions, plans, goals and behaviors. However, classic plan recognition is ill-suited to modeling robotic systems, as (i) it assumes that actions are discrete, instantaneous and cannot take place in parallel; and (ii) it uses a planning operator-based representation, which differs significantly from the behavior-based controllers often used with robots—thus making it difficult to represent the reactive components of robots interactions with their environment. We present a behavior-based approach to plan-recognition, in which hierarchical behaviors are used to model the observed robots. We show that our new model allows efficient, practical inference based on observations of parallel and continuous actions, and knowledge of the observed robots. We present highly efficient symbolic algorithms for answering key queries about observed robots, under conditions of lossy and lossless observations.

1. Introduction

It is important for robots to be able to reason about other robots' internal state, such as their selected behaviors, plans, intentions, and goals. A model of other robots is important, for instance, in assisting other robots [9], or countering their adversarial actions [15]. Since it is often impractical for a robot to rely on its peers to continuously transmit their internal unobservable state to it, a robot modeling its peers must often rely on inferring its peers' unobservable state based on their observable actions. This inference process is called *plan recognition*.

However, most plan recognition methods are ill-suited to modeling modern robots. First, plan recognition typically assumes that observed actions are discrete, instantaneous, and come one at a time. However, robots often use take continuous actions that have duration, and affect several actuators at once (e.g., maintain velocity and direction over a period of time) [2, 12]. Second, the representation underlying plan-recognition is based on STRIPS-like operators, a representation not commonly used in generating reactive behavior in robots. An operator-based model of a robot's interaction with the environment would not capture the reactive components of its behavior.

There have been attempts at addressing these challenges (see Section 2 for details). RESC [15] and RESL [7] use a behavior-based representation to infer the current behaviors selected by observed agents. However, they do not take a history of observations into account, and assume that agents do not change states (behaviors) unobservably. Other methods are often able to take a history of observations into account, but assume all relevant features (e.g., actions of the robot) will always be observable (e.g., [5, 13, 4]). Moreover, these methods require a translation of the observed robots behavior-based control structure into a form suitable for the probabilistic recognition algorithms used in these approaches. Also, none of the approaches discussed above can utilize negative evidence, i.e., inference from a lack of an observation [3].

This paper focuses on comprehensive mechanisms for *behavior recognition*, the task of recognizing the unobservable behavior-based state of a robot, given observations of its interaction with its environment. We examine the key behavior recognition queries that may be asked of a behavior recognition system, and provide algorithms to infer the answers to these queries, building on a a representation of hierarchical behavior that is general and compatible with many existing behavior-based control methodologies. We analyze the complexity of the algorithms, and show that they are efficient, even when handling loss of observations, unobservable behaviors, and negative evidence. The algorithms are all symbolic, in that they produce all possible hypotheses that are consistent with the observations.

2. Background and Related Work

Classic plan recognition work has focused on efficiently matching observed atomic actions against STRIPS-like operators, to infer a sequence of operators that (best) accounts for the actions (e.g., [8]). This approach faces inherent difficulties when applied to the complex, dynamic settings in which robots are typically deployed.

Indeed, robots that operate in complex, dynamic settings, may interrupt planned action sequences in order to react to unexpected situations. To do this, many robots utilize a behavior-based control approach, which executes multiple control modules (called *behaviors*) so as to produce the desired behavior, while still maintaining the ability to react to unexpected situations [10, 12, 1]. Such behaviors often control several actuators at once, and with varying duration.

An ideal behavior recognition system would be able to address the deficiencies above, while taking into account a history of observations to infer answers for recognition queries such as: (i) What is the current selected behavior of the observed robot? (ii) What sequence of behaviors did the observed robot go through?

Ideally, the recognition system would be able to provide answers to the recognition queries despite challenges often associated with real-world robotic applications, such as lossy observations (e.g., occasionally not being able observe a robot's heading), and varying behavior execution durations. The response to queries can be a single answer, or a set of hypothesized answers (in case of uncertainty). We focus in this paper on pure symbolic approaches, and do not consider the issue of ranking hypotheses according to some criteria (e.g., likelihood). We follow the plan recognition literature in assuming that the underlying model used for recognition completely covers the behavioral repertoire of the observed robot.

There have been several relevant previous investigations. RESC [15] uses a hierarchical behavior-based representation to infer the current behaviors selected by observed robots. In each run-time cycle, RESC maintains only a single hypothesis as to the current state of the observed robot. RESL [7] is similar, but maintains multiple hypotheses as to the current state. Both algorithms essentially reset with every new observation, and do not take a history of observations into account. Thus they cannot provide hypotheses as to the sequence of unobservable states that the observed agents has gone through, nor can they provide predictions as to the next possible state.

Other alternatives to classic plan recognition are also relevant. Many of these are probabilistic in nature, and also are able to take a history of observations into account. [5] explores an approach in which a Bayesian network for plan-recognition is automatically constructed out of reaction-plan specifications (that are similar in nature to behavior-based controllers). [13] explores an efficient probabilistic grammar representation for plan-recognition. However, it assumes discrete actions come one at a time. [4] explores a recognition approach based on hidden Markov models, and allows for behaviors to be interrupted. All of these approaches assume all relevant features (e.g., actions of the robot) are always observable. Also, none of the approaches discussed above (including RESC and RESL) can utilize negative evidence, i.e., inference from a lack of an observation [3].

3. Representation

Many state-of-the-art robotic controllers employ hierarchical behavior-based control methodologies (e.g., [2, 12, 10, 1]). Given this emphasis in the literature, we utilize a behavior-based recognition representation which serves as the basis for representing the modeled behaviors of the observed robot. In choosing a representation for recognition, we are fortunately not constrained by a specific methodology—since the representation does not express executable controllers—but instead can focus on common features to these methodologies. As a result, the representation can be generic, and be used to represent controllers of various types.

We follow previous work in representations for monitoring [6], and represent the behavior-based controllers of an observed robot as a directed acyclic connected graph, where vertices denote behaviors, and edges can be of two types: vertical edges that decompose top behaviors into sub-behaviors, and sequential edges that specify the expected temporal order of execution. Each behavior has associated with it a set of conditions on observable features of the robot or world that specify the settings under which observations are said to match the behavior.

At any given time, the observed robot is assumed to be controlled by a *behavior-path*, a root-to-leaf path of behaviors that follows decomposition edges. Figure 1 shows an example portion of a behavior graph, inspired by the behavior hierarchies of existing robotic soccer teams [14]. The figure shows decomposition edges (solid arrows) and sequential edges (dashed arrows). For presentation clarity, we show the decomposition edges only to the first (in temporal order) child behaviors. Thus in the figure, the behavior path $root \rightarrow defend \rightarrow turn \rightarrow with\ ball$ can be an hypothesis as to the current internal state of an observed robot. A set of such behavior paths would constitute a set of hypotheses. An observed robot may change its internal state in two ways. First, it may follow the sequential edges, such that when no further sequential links are available, control goes back to the parent (which then continues using its own sequential edges, if they exist). Second, control may be interrupted at any time to respond reactively to the environment, and a new (first) behavior may be selected. For instance, suppose a robot was executing $root \rightarrow defend \rightarrow turn \rightarrow with\ ball$, and then interrupted this behavior. It may now choose $root \rightarrow attack \rightarrow pass$, but not $root \rightarrow attack \rightarrow turn$. The figure does not show the observation conditions associated with behaviors. For instance, suppose there is a feature $have\_ball$ whose value is true whenever

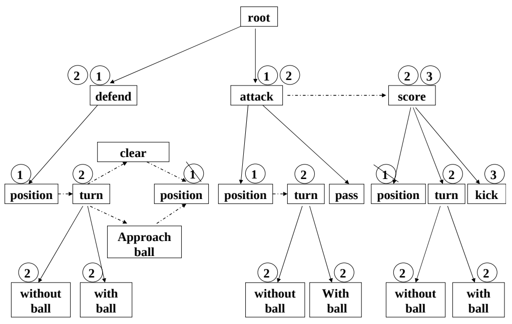

Figure 1. Example behavior graph. Circled numbers denote timestamps.

the ball is observed to be in close proximity to the observed robotic soccer player. The behavior $kick$ may have a condition that specifies $have\_ball = true$, while the behavior $approach\_ball$ would test for $have\_ball = false$.

4. Behavior Recognition Algorithms

We now turn to presenting basic behavior recognition algorithms that utilize the representation above. The algorithms work in three incremental phases: (i) *matching* observations to behaviors, tagging those that match using observation timestamps (Section 4.1); (ii) *propagating* timestamp tags up and down the graph, to tag complete behavior paths (Section 4.2); and (iii) extracting hypotheses according to the query (Section 4.3).

4.1. Matching Observations to Behaviors

We consider complex observations, that consist of a tuple of observed features, including states of the world (e.g.,the observed robot's uniform number), actions taken by the robots (e.g., kick ball), and execution conditions maintained by the robot (e.g., speed=200). It is likely, in realistic settings, that more than one behavior will match a set of observations, and this may result in multiple hypotheses as to the internal state of the observed robot.

Matching observations to behaviors can be expensive, if we go over all behaviors and for each behavior check all observed features. Since not all behaviors utilize all observed features in their associated observation conditions (see previous section), much of this effort may be wasted.

To overcome this problem, we augment the behavior graph with a novel data-structure, a *Feature Decision Tree* (FDT), which allows efficient mapping from observations to behaviors that may match them. An FDT works similarly to a machine-learning decision tree [11], and is constructed similarly, but with important differences.

Each node in an FDT corresponds to an observation feature (e.g., velocity, heading, etc.). Each branch descending from a node, represents one of the possible values of this feature. Unlike traditional decision trees, each leaf also has pointers that point to the behaviors that test for the feature represented by the leaf. In this way, each node in the FDT divides a set of behaviors to subsets according to values of one feature. Thus determining the behaviors that match a set of observations features is efficiently achieved by traversing the FDT top-down, taking branches that correspond to the observed values of features, until a leaf node is reached. The leaf points to behaviors that match the conjunctive set of observations.

Similarly to a decision tree, the construction of an FDT can use information gain to determine the most important features to test first, thus guaranteeing optimal testing of features. We briefly review this well-known process here (see [11] for details). First, we create a root node (corresponding to all behaviors), and associates it with the feature that provides the greatest information gain (intuitively, that divides behaviors that test it as uniformly as possible). We then create children FDT nodes for each of its values, and recursively repeat the process of selecting a feature that best divides the behaviors associated with the node. The process continues until a node points at only a single behavior, or there are no more features that differentiate behaviors.

Unlike machine-learning decision trees, that are built based on examples of the target data, here we base the construction of the decision tree on the behavior graph which is given to us by the designer of the behavior recognition system. This behavior graph contains all the behaviors executable in principle by an observed robot. There is no uncertainty in determining which behaviors match a set of observations, and no need to prune nodes to prevent over-fitting ([11]). In case some behaviors do not test a feature, they are simply passed in the construction phase to all children FDT nodes, as they are consistent with all values of the features they do not test.

For example, the FDT in Figure 2, shows a portion of an FDT using features associated with behaviors in Figure 1, such as distance from other players, $have\_ball$, opponent goal visibility, and uniform number. The FDT separates the behaviors according to the values of these features. To determine matching behaviors, the matching algorithm first checks the $have\_ball$ feature. Based on its value, it continues the appropriate branch to test in sequence other features, until it finally reaches a leaf node. This leaf node will have pointers to all instances of the behaviors associated with it in the behavior graph. For instance the leaf-node for $position$ will have four separate pointers into the behavior graph in Figure 1. Note that since the behavior $turn$ is applicable regardless of whether $have\_ball$ is true or false, a node associated with it will appear in both left and right subtrees of the $have\_ball$ root node.

Matching behaviors to observations is efficiently done using an FDT, by following along a root-to-leaf path. The

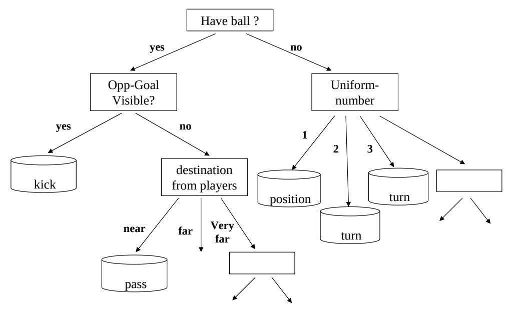

Figure 2. Example FDT.

height of an FDT is (in the worst case) $O(F)$ where F is the number of the features. This would be true only if a behavior test all available features, an unrealistic case. We believe that in realistic settings, the height of the tree would be closer to the best case of $O(logF)$. This result should be contrasted with previous algorithms (e.g., [15]). In these algorithms, the matching step goes over all behaviors, and for each behavior tests all features, therefore the time complexity of these two algorithm for the matching step is $O(FL)$, where L is the number of vertices in the behavior graph.

4.2. Tagging and Propagating

Once matching behaviors are found, they are tagged by the observation time-stamp. These tags are then propagated up and down the behavior graph, to determine complete behavior-paths that constitute hypotheses as to the internal state of the observed robot, at the time the observations were made. However, the propagation is not a simple matter of following child and parent edges.

One complication in tagging is that a behavior may match the observations (and is therefore tagged), and yet it cannot be a part of a valid hypothesis, when a history of observations is considered, i.e., it is *temporally inconsistent*. For instance, suppose that the first set of observations match the $pass$ behavior (Figure 1), and the second set of observations match the $turn$ behavior. The FDT would point the propagation algorithm to the three instances of $turn$, under $defend$, $attack$, and $score$. Assuming no observations were lost, only the behavior instance under $score$ is valid, since it is the only instance in which $turn$ could have been selected without first going through $position$. This reasoning about hypothesis consistency over time is a key novelty compared to previous symbolic behavior recognition algorithms (e.g., [15]).

The process is shown in Algorithm 1—Propagate—which is called for each of the behaviors that match the observations. It takes a pointer to a matching behavior, and tags behaviors using uses timestamps to keep track of the order in which the hypotheses are formed. It exploits the sequential edges and the timestamps to disqualify

Algorithm 1 Propagate(Node w, Behavior Graph g, Timestamp t)

```
1:  $T \leftarrow \emptyset$
2:  $propagateUpSuccess \leftarrow true$
3:  $v \leftarrow w$
4:  while $v \neq root(g) \wedge propagateUpSuccess \wedge \neg tagged(v,t)$ do
5:    if $\exists X$, s.t. $seq\_edge(X,v) \in g \wedge tagged(X,t-1)$ then
6:      $tag(v,t)$
7:    else if $\neg\exists X$, s.t. $seq\_edge(X,v) \in g$ then
8:      $tag(v,t)$
9:    if $tagged(v,t)$ then
10:     $T \leftarrow tagged \cap \{v\}$
11:     $v \leftarrow parent(v)$
12:   else
13:     $propagateUpSuccess \leftarrow false$
14: if $propagateUpSuccess$ then
15:   $PropagateDown(w,g,t)$
16: else
17:   for all $a \in T$ do
18:     $delete\_tag(a,t)$
```

hypotheses that are inconsistent given *a history* of observations.

The algorithm operates as follow: Lines 4–13 climb up the graph, tagging the behavior-path towards the root. If successful (line 14), it also propagates the time-stamp tag down (Algorithm 2). When propagating up or down, successful propagation is determined using the conditions (lines 5–8, Algorithm 1, and lines 2–5, Algorithm 2). These conditions are the key to the temporal validity of the hypothesis. There are two cases: (a) the node in question follows a sequential edge from a behavior that was successfully tagged at time $t-1$; or (b) the node is a first child (there is no sequential edge leading into it). A first child may be selected at any time (for instance, if another behavior was interrupted). If neither of these cases is applicable, then the node is not part of a temporally-consistent hypothesis, and its tag should be deleted, along with all tags that it has caused in climbing up the graph (lines 16–17).

For each behavior instance that matches the observations, the entire propagation traverses the height of the behavior graph, and may thus take $O(L^2)$ in a theoretical worst case in which we check previous tag for each behavior. Realistically, we believe that run-time will often be closer to $O(log^2 L)$.

Figure 1 shows the process in action (the circled numbers in the figure denote the timestamps). Assume that the FDT is used at time $t = 1$ to find multiple matching instances of the $position$ behavior. Propagate (Algorithm 1) begins with these four instances. It immediately fails to tag the instance that follows $clear$ and $approach\ ball$, since

Algorithm 2 PropagateDown(Node w, behavior graph g, time-stamp t)

```
1: for all $c \in children(w)$ do
2:   if $\exists X$, s.t. $seq\_edge(X, c) \in g \wedge tagged(X, t-1)$ then
3:     $tag(c, t)$
4:   else if $\neg\exists X$, s.t., $(X, c) \in g$ then
5:     $tag(c, t)$
6:   if $tagged(c, t)$ then
7:     for all $b \in children(c)$ do
8:       $tag(b, t)$
9:       $propagateDown(b, g, t)$
```

these were not tagged (at $t = 0$). The $position$ instance under $score$ is initially tagged, but in propagating the tag up, the parent $score$ fails, because it follows $attack$, and $attack$ is not tagged. Therefore, all tags $t = 1$ will be removed from $score$ and its child $position$. The two remaining instances successfully tag up and down, and result in possible hypotheses $root \rightarrow defend \rightarrow position$ and $root \rightarrow attack \rightarrow position$.

At time $t = 2$, suppose the observations match the $turn$ behavior (three instances). The tag $t = 2$ propagates successfully up and down in the behavior tree, for all instances. Now, there are six possible hypotheses (we omit the common $root$ prefix): $defend \rightarrow turn \rightarrow without\ ball$, $defend \rightarrow turn \rightarrow with\ ball$, $attack \rightarrow turn \rightarrow without\ ball$, $attack \rightarrow turn \rightarrow with\ ball$, $score \rightarrow turn \rightarrow without\ ball$, $score \rightarrow turn \rightarrow without\ ball$. Now, we can not decide which of the three main behaviors took place: $defend$, $attack$ or $score$. However, getting the next observation can disambiguate the hypotheses. If we will next observe a $clear$ or $approach\ ball$, then it would be clear that the observed robot is executing the $defend$ hypothesis. Otherwise, we can eliminate this hypothesis. In other words, we can exploit negative evidence to disambiguate the hypotheses space. This process is tightly coupled to the hypothesis generation phase, described next.

4.3. Generating Hypotheses

After having the behavior tree tagged with labels according to time-stamps, we can generate hypotheses to answer recognition queries. As briefly mentioned above, the same process also exploits negative evidence to further eliminate hypotheses.

Generating hypotheses about the current selected state (behavior path), is trivial: Given the latest time-stamp t_0, we traverse the behavior-graph, identifying complete behavior paths that are tagged $t = t_0$. The set of these behavior paths constitutes the response to this type of query.

However, generating hypotheses as to the sequence of states that was selected over time is not a simple matter of enumerating combinations of the above queries for times $t = 0, t = 1 \ldots, t = t_0$. The reason for this is that new hypotheses, generated at time t_0, may serve to rule out hypotheses that successfully matched at time $t < t_0$, by exploiting failures to observe expected behaviors.

Before discussing the hypotheses generation method, it would be useful to see an example of how reasoning about a sequence of behavior paths can lead to using negative evidence.

Suppose that after having made the observations at times $t = 1$ and $t = 2$ in the example of the previous section, we now make observations at time $t = 3$ that match $kick$. The $score$ behavior is the only behavior consistent with the tag $t = 3$, though both $defend$ and $attack$ are tagged for times $t = [1, 2]$. However, after having made the observation at $t = 3$, we can safely rule out the possibility that $defend$ was ever selected by the robot, because $score$ can only follow $attack$, and the lack of evidence for either $clear$ or $approach\ ball$ at time $t = 3$ (which would have made $defend$ a possibility at this time) can be used to rule it out. Thus we infer that the sequence of behavior paths that was selected by the robot is $attack \rightarrow position$ (at $t = 1$), $attack \rightarrow turn$ at $t = 2$ (though we cannot be sure which one of $turn$'s children was selected), and finally $score \rightarrow kick$.

We now turn back to the hypothesis generation method. Extracting all paths is not trivial (since as we saw, some successful tags at time $t < t_0$ are invalid at $t = t_0$. Here we present an incrementally-maintained structure that holds hypotheses according to time stamps. The advantage is that with every time stamp t, we can use the structure to eliminate hypotheses that were tagged at time $t - 1$, that have become invalid.

We use a connected graph G', whose vertices correspond to successfully-tagged behavior paths in the behavior graph (i.e., hypotheses). Edges in G' connect hypothesis vertices tagged with time stamp t to hypothesis vertices tagged with time stamp $t + 1$. G' is therefore built in levels, where each level represents hypotheses that hold in each time stamp. For each set of observations made at time t_0, we add to G' a level t_0 all possible hypotheses that were tagged $t = t_0$ and propagated successfully in the behavior graph. We then created edges between vertices $y_1, \ldots, y_n$ in level $t - 1$ to each new hypothesis X in level t in the following manner: If X is not part of a sequence (i.e., it is a first child), then we connect each y_i to X; otherwise, if X is part of a sequence, we connect y_i to X if any of the behaviors in y_i has a sequential edge to any behavior in X.

Now, to generate all sequences of behavior paths that are consistent with the observations, we traverse G' from level to level, keeping track of all G' paths that take us from the first level to the last level (the most recent observation).

For example, based on the behavior tree in Figure 1, we construct G' (Figure 4.3). In the first level, we put all paths in the behavior tree that were tagged with time-stamp 1,

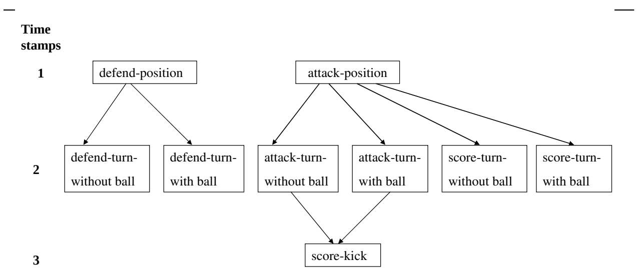

Figure 3. An example extracting graph G'.

in the next level we put all paths that were tagged with time-stamp 2. Now, from each node in time-stamp 2, we check which nodes can be appropriate in time-stamp 1. The $t = 1$ node $defend \rightarrow position$ can be connected to the $t = 2$ nodes $defend \rightarrow turn \rightarrow without\ ball$ and $defend \rightarrow turn \rightarrow with\ ball$, because there exist sequential edges in the behavior graphs that connect $position$ to $turn$ under $defend$. Similarly, $attack \rightarrow position$ has edges to $score \rightarrow turn \rightarrow without\ ball$ and $score \rightarrow turn \rightarrow with\ ball$. Once we add the observations for $t = 3$, the various $defend$ hypotheses have no link to time $t = 3$. If we now go back to asking what hypotheses exist for the current behavior paths at time $t = 2$, we will get $attack \rightarrow turn \rightarrow without\ ball$ and $attack \rightarrow turn \rightarrow with\ ball$. Here we can see the advantage of using G': Out of six hypotheses that matched the observations until time-stamp 2, four hypotheses are eliminated once we incorporate the evidence in time-stamp 3.

The worst-case runtime complexity of constructing G' over N observation time steps is $O(NL^2)$, where and L is the worst-case number of behaviors that the matching algorithm had returned given a single observation. For each node in G' with time stamp t (of which there could be at most $O(L)$), we check all nodes in time stamp $t - 1$ (again, $O(L)$), thus a factor of L^2 for each step of adding another level. However, Note that the L component is a purely theoretical worst-case, as it corresponds to a recognition system that simply returns all behaviors in the behavior graph.

5. Algorithms for Realistic Settings

Real-world applications sometimes violate assumptions that are made in recognition systems. Two common violations of assumptions involve intermittent observation failures, and varying behavior execution duration. This section addresses these challenges.

5.1. Lossy Observations

In section 4.1, we showed how efficiently determine which behaviors match a set of observations. An implicit assumption was made (present also in most related work) that all relevant features were in fact observables. However, in realistic settings, some features may be intermittently unobservable, e.g., due to hardware failures, communication errors, etc. Observations that are lost would fail the conditions associated with behavior, and thus the matching phase will fail.

We propose to use an augmented FDT, called LFDT (Lossy Feature Decision Tree), which has all the properties of FDT, but deals with lossy observations. The LFDT representation is the same as FDT, except that for each node, we add an extra branch that represents a *missing value*. During construction of the LFDT, all behaviors that are consistent with the node (and which are divided based on the value of the feature associated with the node) would be passed as-is to the missing value branch. When the LFDT is traversed, if a feature is temporarily unobservable, we will follow the missing value branch instead of one of the normal branches.

The runtime complexity of LFDT is the same as FDT (Section 4.1), though the size of the LFDT would be greater: (a) it will have more branches than FDT (extra branch for each feature); (b) its height may be deeper than FDT (because of the need to handle missing features at the leaves).

5.2. Behaviors with Duration

In state-of-the-art applications, behaviors can vary greatly in the duration of their execution. Even the same behavior can vary in its duration. Therefore, the observation time and behavior time are not synchronized. For example, the $approach\ ball$ behavior in Figure 1 can sometimes take much time, and sometimes be done almost immediately (depending on the distance to the ball). As a result, we may have multiple observation timestamps ($t, t+1, \ldots t+k$) that are all consistent with a single behavior, and only reflect the duration that its execution requires.

Fortunately, it is easy to allow for unknown durations in the approach we presented. Instead of insisting that we time-stamp t a behavior v only if it follows a sequential edge from a behavior timestamped $t - 1$, we must allow for the possibility that v itself has previously matched, and that the observed robot is simply continuing its execution of v. We therefore add a condition underwhich we allow $tag(v, t)$. To the tests in lines 5–8 of Algorithm 1, and lines 2–5 of Algorithm 2 we add **if** $tagged(v, t - 1))$ **then** $tag(v, t)$.

6. Summary and Future Work

It is important for a robot to monitor other robots in order to carry out its tasks. To do this, robots must often rely on their observations of others, to infer their unobservable internal state, such as goals, plans, or selected behaviors.

However, plan-recognition approaches to this task are insufficient for modern robotic applications.

This paper addresses this challenge by defining a behavior-based recognition representation and a comprehensive set of algorithms that can answer a variety of recognition queries. The algorithms we propose are efficient, and can handle important real-world challenges to existing techniques, such as intermittent failures in observations, behaviors with duration, etc. In the future, we home to extend our approach to cover interleaved and resumable behaviors. A promising direction is to exploit the high efficiency of these algorithms to pre-process hypotheses for probabilistic ranking.

References

[1] T. Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, 1998.

[2] R. J. Firby. An investigation into reactive planning in complex domains. In *AAAI-87*, 1987.

[3] R. P. Goldman, C. W. Geib, and C. A. Miller. A new model of plan recognition. In *UAI-1999*, Stockholm, Sweden, July 1999.

[4] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the IJCAI-99 Workshop on Team Behavior and Plan-Recognition*, 1999. Also appears in Proceedings of the 9th International Symposium of Robotics Research (ISSR-99).

[5] M. J. Huber, E. H. Durfee, and M. P. Wellman. The automated mapping of plans for plan recognition. In *Proceedings of UAI-94*, 1994.

[6] G. A. Kaminka and M. Bowling. Robust teams with many agents. In *AAMAS-02*, 2002.

[7] G. A. Kaminka and M. Tambe. Robust multi-agent teams via socially-attentive monitoring. *JAIR*, 12:105–147, 2000.

[8] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *AAAI-86*, pages 32–37. AAAI press, 1986.

[9] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation—the framework and the basic task patterns. In *the IEEE International Conference on Robotics and Automation*, pages 767–773, San-Diego, CA, May 1994. IEEE Computer Society Press.

[10] M. J. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.

[11] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[12] M. Nicolescu and M. J. Mataric. A hierarchical architecture for behavior-based robots. In *AAMAS-02*, pages 227–233, Bologna, Italy, July 15–19 2002.

[13] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *UAI-2000*, pages 507–514, 2000.

[14] M. Tambe, J. Adibi, Y. Al-Onaizan, A. Erdem, G. A. Kaminka, S. C. Marsella, and I. Muslea. Building agent teams using an explicit teamwork model and learning. *AIJ*, 111(1):215–239, 1999.

[15] M. Tambe and P. S. Rosenbloom. RESC: An approach to agent tracking in a real-time, dynamic environment. In *IJCAI-95*, August 1995.