Aspects of Plan Operators in a Tree Automata Framework

Johanna Björklund
Dept. Comp. Sci., Umeå Univ.
SE–901 87 Umeå, Sweden
Email: johanna@cs.umu.se

Eric Jönsson
Dept. Comp. Sci., Umeå Univ.
SE–901 87 Umeå, Sweden
Email: ericj@cs.umu.se

Lisa Kaati
FOI
SE–16490 Stockholm, Sweden
Email: lisa.kaati@foi.se

***Abstract*—Plan recognition addresses the problem of inferring an agent's goals from its actions. Applications range from anticipating caretakers' needs to predicting volatile situations. In this contribution, we describe a prototype plan recognition system that is based on the well-researched theory of (weighted) finite tree automata. To illustrate the system's capabilities, we use data gathered from matches in the real-time strategy game StarCraft II. Finally, we discuss how more advanced plan operators can be accommodated for in this framework while retaining computational efficiency by taking after the field of formal model checking and over-approximating the target language.**

I. Introduction

The field of plan recognition is devoted to algorithms for inferring an agent's goals, starting from observations of the agent's actions, or the effects of those actions [1], [2]. The agent can (for instance) be a caretaker at a nursing home, an opponent in a game of chess, or an enemy in an armed conflict. The aim is not only to determine what situation the agent is working towards, but also by what means he or she means to bring it about, leading to the related question of what can be done to help or hinder its efforts.

Plan recognition systems typically build on the notion of a plan library, a collection of known plans that can be effectuated by the agent. These plans have as a rule a hierarchical structure, with the agent's long-term goals at the highest level and intermediate goals at lower levels. Furthest down in the hierarchy we find concrete actions and observable effects, which may come from a variety of different sources such as sensors, cameras, text, reports or news media.

When the system is executed, a plan recognizer is used to match observations to specific plan steps in the library and eventually also to top-level plans. Plan recognition is therefore a suitable technique for fusing information from several different information sources. However, the formalisms used to specify plans can only be moderately complex, because they must admit efficient parsing to be of any practical value. Plan recognition is thus of greatest use for closed world situations where the agent's number of choices is limited and can, at least to some degree, be anticipated.

In previous work [3], we modeled the unobstructed keyhole plan recognition problem within the framework of weighted unranked tree automata (WUTA). In this framework, a node in a plan tree is allowed to have an unbounded number of children. By using weighted devices, the plans in the library were prioritized by an impact factor, or by the likelihood of activation with respect to a given sequence of observations. A downside of weighted devices is however that they require prior knowledge, or at the very least informed guesses about the likelihood and impact of a certain event.

In this paper, we describe a prototype of the system in [3] and demonstrate its capabilities using events in the real-time strategy game StarCraft II for illustration. Our second contribution is a discussion of how the complexity introduced by more expressive plan operators can be manged by allowing the system to over-approximate the language of potentially threatening patterns. In particular, we consider what we call *concurrent* and *restricted* logical 'or'. Concurrent 'or' lets us express several events which happen simultaneously (or within the same time frame). Restricted 'or' capture situations in which a certain number of events must be present to realize a plan or subplan. To make the presentation accessible to a larger community, we again restrict ourselves to unweighted devices, but the argumentation can be lifted to weighted automata in a straightforward fashion.

The choice to represent plans as trees and sets of realisations of plans by tree automata stems from the inherent hierarchical structure in plans. Tree automata were originally motivated by applications in computational linguistics, but have since been re-purposed for tasks in natural language processing, formal verification, and model checking. Their theory is now well-studied [4], [5], [6], [7], [8], [9], [10], [11], and in part implemented in toolkits such as Treebag [12] and Tiburon [13].

II. Preliminaries

Automata that deal with tree structures are called *finite tree automata*. The theory of finite tree automata arises as an extension of the theory of finite (string) automata [14]. Even though the two devices are used in different settings, they are closely related because finite automata can be seen as a special case of finite tree automata.

Tree automata can be classified as bottom-up or top-down, depending on the order in which they traverse their input trees. A top-down tree automaton starts its computation at the root of the tree and then processes in parallel the paths spreading from the root of the tree, level by level. A bottom-up tree automaton,

This work was supported by Vinnova through the Vinnmer programme.

moves in the reverse order and starts its computation at the leaves of the input tree, and then works its way up towards the root. In this work we will consider bottom-up tree automata, because they better reflect the deduction from sequences of observations to long-term goals.

A *ranked alphabet* Σ is a finite set of symbols together with a function $\# : \Sigma \to \mathbb{N}$ where $\mathbb{N}$ is the set of all natural numbers. For $f \in \Sigma$, the value $\#(f)$ is called the rank of f. For any $n \geq 0$, we denote by Σ_n the set of all symbols of rank n from Σ.

Definition II.1 (Tree)**.** *A* tree t *over an alphabet* Σ *is a partial mapping* $t : \mathbb{N}^* \to \Sigma$ *that satisfies the following conditions:*

- $dom(t)$ *is a finite, prefix-closed subset of* $\mathbb{N}^*$*, and*
- *for each* $p \in dom(t)$*, if* $\#(t(p)) = n > 0$*, then*

$$\{i \mid pi \in dom(t)\} = \{1, \ldots, n\}\} \ .$$

Each sequence $p \in dom(t)$ *is called a node of* t*. For a node* p*, we define the* i^{th} *child of* p *to be the node* pi*, and we define the* i^{th} *subtree of* p *to be the tree* t' *such that* $t'(p') = t(pip')$ *for all* $p' \in dom(t')$*. The root of* t *is the unique node* $\varepsilon \in dom(t)$*. A leaf of* t *is a node* p *which does not have any children, i.e. there is no* $i \in \mathbb{N}$ *with* $pi \in dom(t)$*.*

We denote by $T(\Sigma)$ the set of all trees over the alphabet Σ, and write a tree t as $f[t_1, \ldots, t_k]$, where $f = t(\varepsilon)$, $k = \#(f)$, and t_i is the i^{th} subtree of ε in t. If $k = 0$, then we simplify this expression as f.

Definition II.2 (Tree Automata)**.** *A finite bottom-up tree automaton (TA) is a quadruple* $A = (Q, \Sigma, \Delta, F)$ *where*

- Q *is a finite set of states,*
- Σ *is a ranked input alphabet,*
- Δ *is a finite set of transition rules, and*
- $F \subseteq Q$ *is a set of accepting (final) states.*

Each transition rule is a triple of the form $((q_1, \ldots, q_n), f, q)$ where $q_1, \ldots, q_n, q \in Q$, $f \in \Sigma$, and $\#(f) = n$. We use $(q_1, \ldots, q_n) \xrightarrow{f} q$ to denote that $((q_1, \ldots, q_n), f, q) \in \Delta$.

In the special case where $n = 0$, we speak about the so-called *leaf rules*, which we abbreviate as $\xrightarrow{f} q$.

If a tree automaton can choose more than one transition rule for an input symbol, it is called a *non-deterministic finite tree automaton* (NFTA). Otherwise it is called a *deterministic finite tree automaton* (DFTA).

Definition II.3 (Tree Automata Semantics)**.** *A* run *of a FTA* $A = (Q, \Sigma, \Delta, F)$ *on a tree* $t \in T_\Sigma$ *is a mapping* $r : dom(t) \to Q$ *such that if* $v \in dom(t)$*, and* $t(v) = f \in \Sigma_k$*, then*

$$(r(v1), \ldots, r(vk)) \xrightarrow{f} r(v) \in \Delta \ ,$$

where $f = t(v)$*. The run is* accepting *if* $r(\varepsilon) \in F$*. A tree* t *is accepted by* A *if there is an accepting run of* A *on* t*. The* language accepted by A *is the set of all trees that it recognises.*

III. Threat models and Plans

Threat models are used to model and foresee possible threats. A common way to represent a threat is to use an hierarchical intelligence model where the root node represents the actual threat and the leaves are indicators. If several indicators are present, then there is a risk that the modeled threat is about to happen. There is an array of intelligence models with different properties that can be used to represent and reason about various kinds of threats.

In our setting, plan recognition is very similar to threat recognition and plans to threat models. The difference is that a plan contains information about dependencies. A plan may for example contain information about in what order actions need to be performed to complete the plan. Another aspect of plan recognition is that one of the objectives is to recognize the actual plan being executed, not just the goal of the plan.

A. Plans and Operators

A plan is a tree where the leaves are labeled with indicators and the internal nodes are labeled with *plan operator symbols*. The plan operator symbols (or simply plan operators) provide the ability to model dependencies among nodes in the plan tree. In a plan tree, the internal nodes represent intermediate goals (or equivalently; subplans) and the indicators correspond to (abstract classes of) observable activities. We say that the subplan represented by the node t is completed if either:

- the indicator representing t is present, or
- t's children are completed according to the plan operator that labels t.

Similarly, we say that a plan is realized if its top-most node is completed.

In this framework we use a set of plan operators

$$\Theta = \{\exists, \exists_n, \forall, \overline{\forall}, \approx\} \ ,$$

the constituents of which are defined as follows:

Definition III.1 (Or)**.** *If a plan* t *is of the form* $\exists[t_1, \ldots, t_n]$*, then* t *is realized if at least one of the subplans* $t_1, \ldots, t_n$ *is realized.*

Definition III.2 (Restricted Or)**.** *If a plan* t *is of the form* $\exists_m[t_1, \ldots, t_n]$*, then* t *is realized if at least* m *of the subplans* $t_1, \ldots, t_n$ *is realized.*

Definition III.3 (Unordered And)**.** *If* $t = \forall[t_1, \ldots, t_n]$*, then* t *is realized if each of the subplans* $t_1, \ldots, t_n$ *are realized, in any order.*

Definition III.4 (Ordered And)**.** *If* $t = \overline{\forall}[t_1, \ldots, t_n]$*, then* t *is realized if each of the subplans* $t_1, \ldots, t_n$ *are realized, in that particular order.*

Definition III.5 (Concurrent)**.** *If* $t = \approx [t_1, \ldots, t_n]$*, then* t *is realized if all of* $t_1, \ldots, t_n$ *are realized, within the same time frame.*

Using the set of plan operator symbols as described above, the definition of a plan tree reads accordingly:

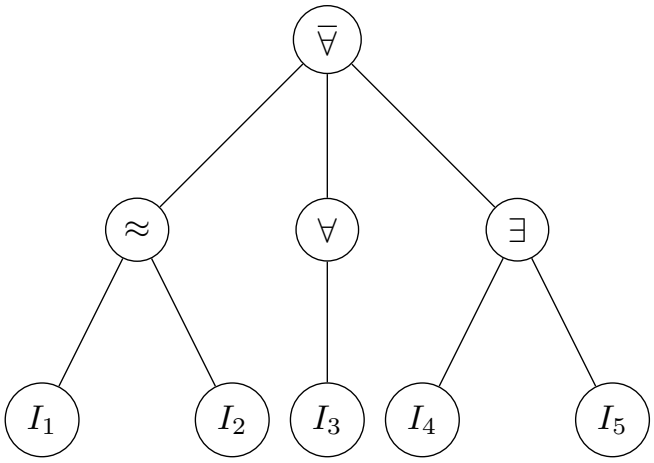


Figure 1: An example of a plan tree.

Definition III.6 (Plan tree)**.** *Let $\Theta = \{\exists, \exists_n, \forall, \overline{\forall}, \approx\}$ be the set of plan operator symbols and Λ a set of indicators. A tree $t \in \mathrm{T}_{\Theta \cup \Lambda}$ is a* plan *(over Θ and Λ) if p is an internal node then $t(p) \in \Theta$, but otherwise $t(p) \in \Lambda$.*

Figure 1 shows an example of a plan tree containing the different operators. For the plan that it represents to be realised, we first have to make the observations I_1 and I_2 within the same time frame, thereafter we must make the observation I_3, and finally at least one of the observations I_4 and I_5.

The operators allows a plan t to be realized in several different ways, and to reason about them we need to think about how information about the observations that are made is propagated upwards through the tree. This will then allow us to identify a particular realisation of t with a selection of subplans (i.e., nodes of the plan tree).

Definition III.7 (Completed set)**.** *Let t be a plan tree and I be a subset of the leaves of t. We denote by $completed_I(t)$ the subset of $dom(t)$ that is completed (as described in Definitions III.1 through III.5) when the set of observations $\{t(i) \mid i \in I\}$ are made and the information is propagated upward through the tree. If the root of t is in $completed_I$, then we say that the completion of the nodes in I leads to the* realisation *$completed_I$ of t.*

It is sometimes informative to view a realisation S of a plan tree t as the tree obtained by (i) first re-labelling every node t in t with the label $\hat{p}$, and (ii) then deleting every node in t that is not in S. What remains is tree representation of a particular way of realising the plan t.

Figure 2 shows the realisation $complete_I(t)$ of the plan tree t in Figure 1 that is happens when we make the observations in $I = \{I_1, I_2, I_3, I_4\}$. In practice, one would rather use unique but informative names for the subplans than the integer strings that index them in the plan tree.

IV. Tree Automata and Plan Recognition

Given a set of plan trees, there is a tree automaton that accepts precisely the set of all possible realisations of the plans that they encode. However, the translation of the operators Unordered And and concurrent would with n arguments would be mapped into 2^n transitions, so a naive translation between the two formalisms is not practical. Instead, we shall adopt the technique of over-approximating the target language from the field of model-checking.

The idea is thus to generate a relatively small tree automaton that recognises a superset of the target plans. This will lead to false positives when the automaton is used to filter large datasets in the search for evidence of potential threats, but a number of unwarranted alerts may be acceptable if we can catch all cases that we are interested in, without sacrificing efficiency. If the number of false positives become too large, then one solution is to use a cascade of increasingly more restrictive automata. This means that most input sequence will be disregarded immediately, and the invocations of the larger and more precise automata can be postponed until they are actually needed.

Let us now turn to the actual construction, and assume that we have a plan tree of the form $t = X[t_1, \ldots, t_n]$, where X is a plan operator symbol from the set Θ. We shall use the nodes of t both as alphabets and as states in our automaton, so as to recognise the tree representations of different realisations of t. To distinguish these separate uses, we write $\hat{p}$ if $p \in dom(t)$ is to be seen as an alphabet symbol, and reserve the plain use of p for when we are viewing it as a state. We also allow the automaton to change states without reading an input symbol, and label the transitions it which this happens with the auxiliary symbol λ. The encoding of a plan tree t into a tree automaton, is the union of the encoding of each of its subplans. The translation of the subplan $X[t_1, \ldots, t_n]$, rooted at the vertex $p \in dom(t)$, depends in turn on the plan operator X as follows:

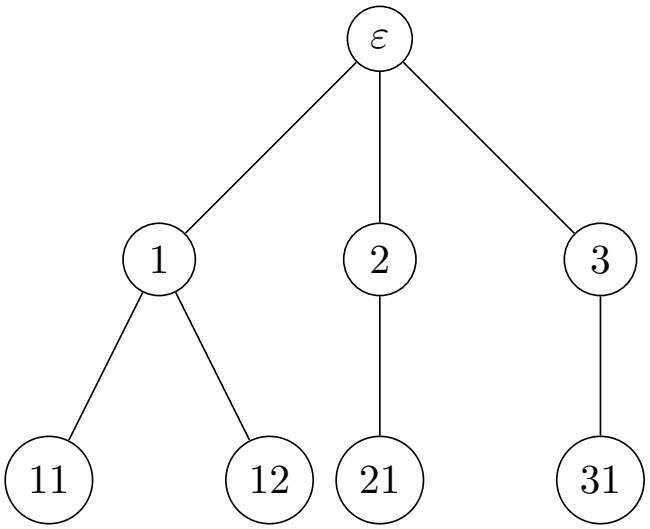


Figure 2: A realisation of the plan tree in Figure 1.

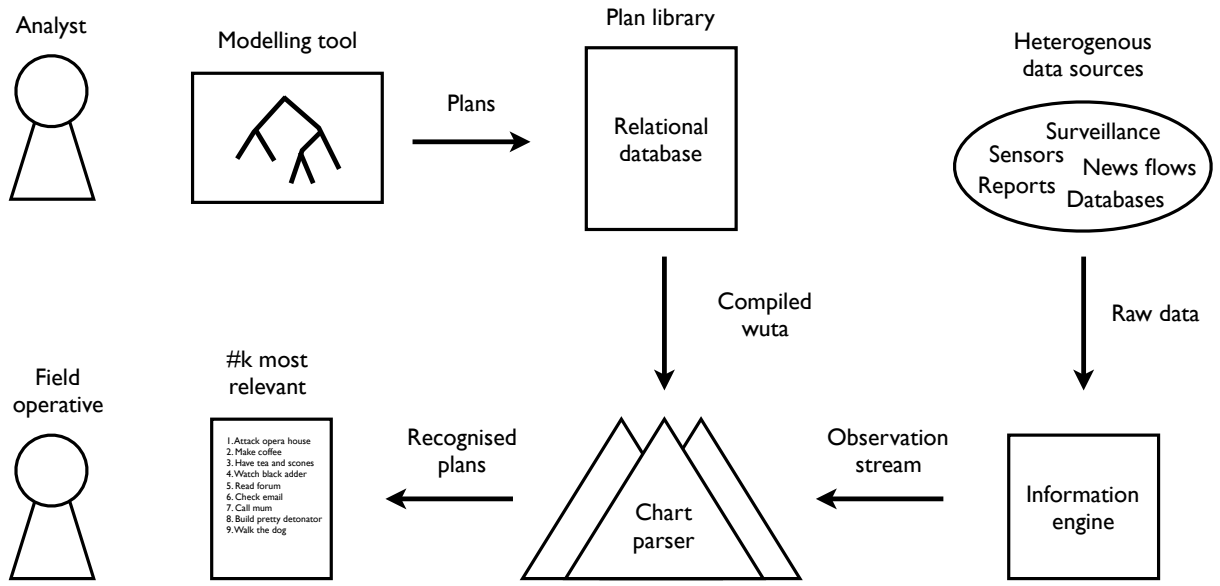


Figure 3: The user works with a modeling tool to create and edit plans. The plans are stored in a plan library and compiled upon demand into a wuta. The wuta can the be used to parse observations streams and produce a list of the k plans that are most likely to be active in a given moment.

A. OR

If $X = \exists$, then the subplan is represented as a sequence of transition rules

$$(p1) \xrightarrow{\hat{p}} p,\ (p2) \xrightarrow{\hat{p}} p,\ \ldots,\ (pn) \xrightarrow{\hat{p}} p\ .$$

The set thus contains n transition rules, each with a left-hand side of length 1.

B. RESTRICTIVE-OR

The operator RESTRICTIVE-OR generalises the operator OR, in that we obtain the latter operator when $m = 1$. However, whereas an operator OR with n arguments only results in n transitions in the output automaton, the operator RESTRICTIVE-OR requires $\binom{n}{m}$ transitions. For this reason, we now introduce the first construction that leads to an over-approximation.

If $X = \exists_m$, then the subplan is represented as a sequence of transition rules

$$(p1) \xrightarrow{\lambda} q,\ (p2) \xrightarrow{\lambda} q,\ \ldots,\ (pn) \xrightarrow{\lambda} q\ ,$$

where q is an auxiliary state not in $dom(t)$. To enforce the constraint that at least m of subplans must be realized, we also add the transition rule

$$\underbrace{(q, q, ..., q)}_{m} \xrightarrow{\hat{p}} p$$

to the automaton.

C. UNORDERED-AND

The encoding of the AND operator also leads to an over-approximation: If $X = \exists_m$, then the subplan is represented as a sequence of transition rules It consists of the set of transition rules of length one:

$$(p1) \xrightarrow{\lambda} q, (p2) \xrightarrow{\lambda} q, ..., (pn) \xrightarrow{\lambda} q$$

To obtain the constraint that at all n of the transition rules needs to be completed, we need to add an extra transition rule to the tree automaton. On the left hand side we add a rule of length n that contains n number of qs

$$\underbrace{(q, q, ..., q)}_{n} \xrightarrow{\hat{p}} p$$

1) ORDERED-AND*:* If $X = \overline{\forall}$, then we need only the single transition

$$(p1, \ldots, pn) \xrightarrow{\hat{p}} p$$

2) CONCURRENT*:* If $X = \approx$, then we add the transitions rules

$$(p1) \xrightarrow{\lambda} q_t, (p2) \xrightarrow{\lambda} q_t, ..., (pn) \xrightarrow{\lambda} q_t$$

where q_t is an auxiliary state depending on the time frame t.

To capture the fact that all subplans are executed within the same time frame, an abstract transition of the form

$$\underbrace{(q_x, q_x, ..., q_x)}_{n} \xrightarrow{\hat{p}} p$$

is added to the tree automaton. This transition fires whenever there are n states q_t where t is one-and-the-same time frame to activate it.

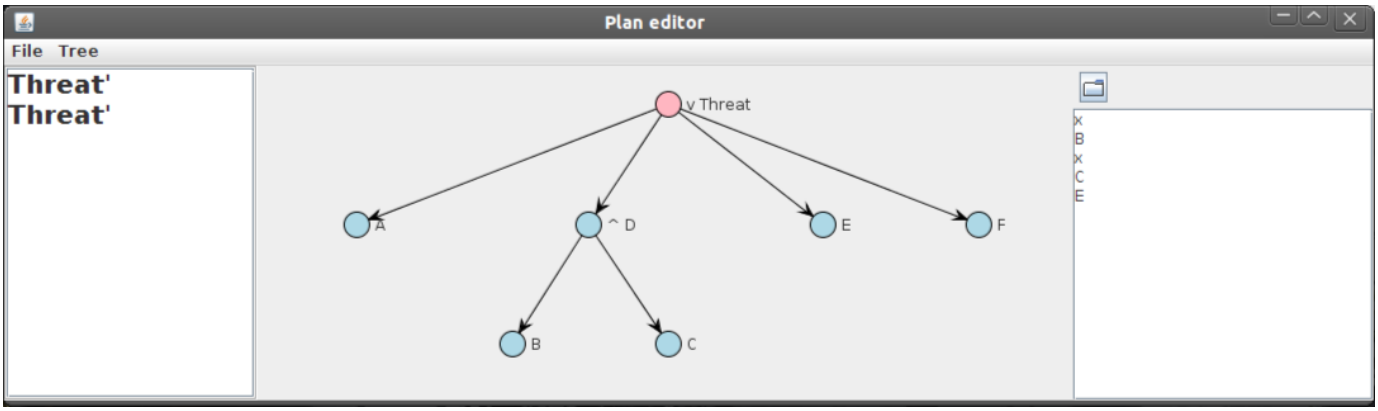


Figure 4: The proof of concept software, displaying a plan tree, warning notifications and an observation log.

The loss of precision that follows from these approximations is as substantial as the computational efficiency gained by them, so whether or not they are appropriate must be decided by the analyst who operates the system. It remains an interesting open question whether there is a satisfactory middle ground between expressiveness and ease of computation.

V. Implementation and demonstration

In [3], we sketched an implementation of this framework which has now been realised as a software prototype [15]. Let us recall the overall outline of this system, which is shown in Figure 3. The core components of the system are an XML-based plan editor, a plan library, a chart parser, and an information engine. In the prototype, the information engine is not fully realised. Instead, information is ingested from a time-stamped text file provided by the user.

The idea is that plans are created and edited by a human analyst using the plan editor, and stored in the plan library, which is typically a relational database. To perform threat detection, the plan library is compiled into a tree automaton, against which a stream of time-stamped observations are parsed by the chart parser by means of an adapted version of the Cocke-Younger-Kasami algorithm. For efficiency reasons, the system maintains a table of partial parses that can be reused throughout the computation [16].

The plan editor is a simple, graphical tool that visualizes plans as trees and provides easy, intuitive ways to modify both their structure and their properties. A host of vertex-related commands are readily available by right-clicking on a vertex, for example labelling them, changing plan operator, disconnecting them from the tree and so forth. The editor lets the user scale and move the tree using the mouse, so even complex trees can be examined easily.

Apart from the main workbench, the plan editor has two additional widgets – an *observation log* and a *notification area*. In the observation log, the incoming observations are tracked in order to provide an overview of incoming data. Whenever a new observation arrives, it is appended to the input string which is again parsed as described in [3]. If the new string of observations parses correctly, a warning is displayed in the notification area, as is evident from Figure 4.

To demonstrate how the system functions, we perform a number of test runs on data from the real-time computer strategy game *StarCraft II*. in this context, two agents compete using game mechanics similar to the rock-paper-scissors hand game. With players not having access to perfect information, they must observe what actions their opponent takes and discern what they are planning in order to win.

Figure 5a shows an in-game situation where a player is sending a scouting unit (circled) to gather intelligence. The software (Figure 5b) has been configured to recognize a single specific threat, of which no relevant observations has been made yet.

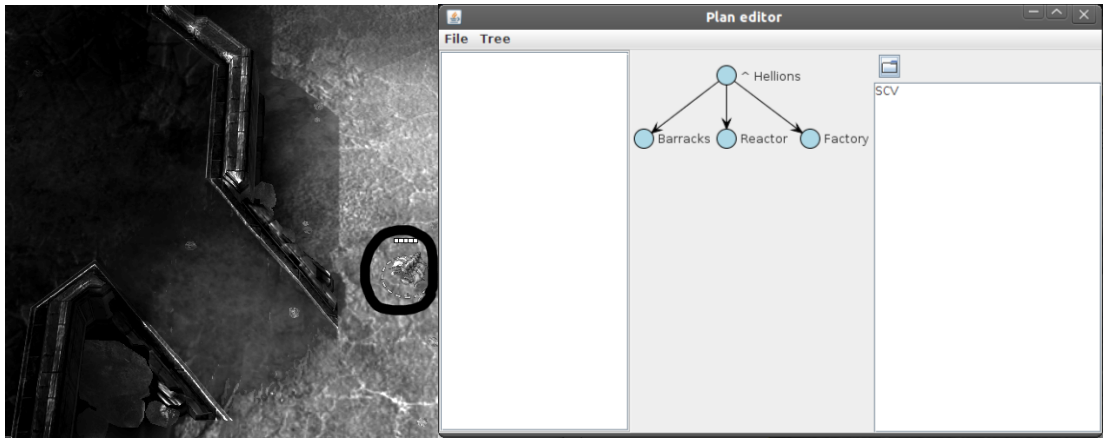
In Figure 6a, the scout has done several observations, namely, the three units needed to produce the threat the software is configured to recognize. The series of observations now parses correctly as a possible threat, so the user is alerted with a warning in the notification area of the software.

VI. Future work

Future work should include a comparison of the descriptive power of plan trees using the proposed plan operators, with that of various kinds of temporal logics. We would also like to investigate how to find the right level of approximation, so as balance computational efficiency with precision, and to further investigate how a single filtering automaton can be converted into an equivalent cascade of simpler automata. Finally, the work on our prototype should continue and reflect the improvements made in the theoretical framework.

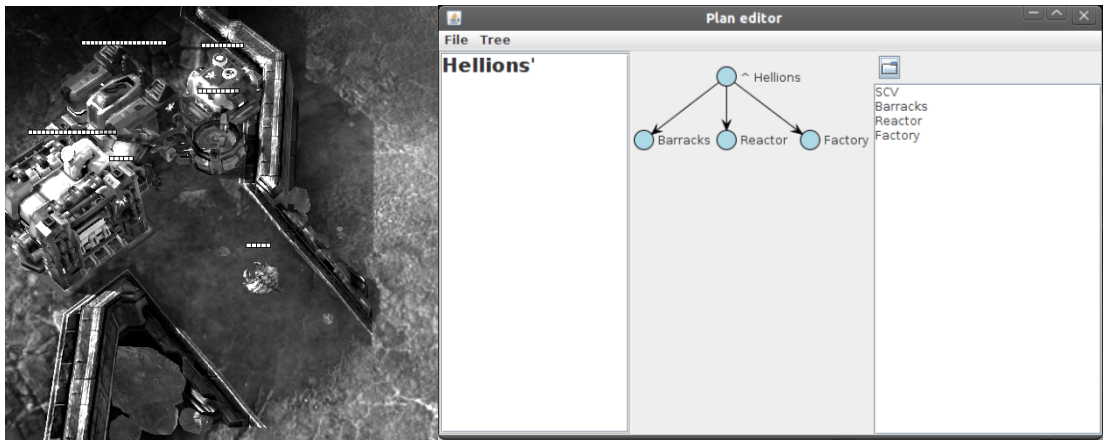
Acknowledgements.

We thank Mohamed Faouzi Atig at Uppsala University for his suggestions regarding the translation form plan trees to tree automata, and Pontus Svenson and Christian Mårtensson for their helpful comments that did much to improve the manuscript.



(a) (b)

Figure 5: A scouting unit. No relevant observations have been made at this point, so warnings are yet to be emitted.



(a) (b)

Figure 6: Three more observations, now parsing correctly as a possible threat.

REFERENCES

[1] H. Kautz, *A Formal Theory of Plan Recognition*. PhD thesis, Dept. of Comp. Sci. University of Rochester, 1987.

[2] S. Carberry, "Techniques for plan recognition," *User Modeling and User-Adapted Interaction*, vol. 11, no. 1-2, pp. 31–48, 2001.

[3] J. Högberg and L. Kaati, "Weighted unranked tree automata as a framework for plan recognition," in *Proc. FUSION 2010, Edinburgh, Scotland*, 2010.

[4] W. S. Brainerd, "The minimalization of tree automata," *Inform. and Control*, vol. 13, no. 5, pp. 484–491, 1968.

[5] J. E. Doner, "Decidability of the weak second-order theory of two successors," *Notices of the American Math. Society*, vol. 12, pp. 365–468, 1965.

[6] J. E. Doner, "Tree acceptors and some of their applications," *J. Comput. Syst. Sci.*, vol. 4, pp. 406–451, 1970.

[7] J. Mezei and J. B. Wright, "Algebraic automata and context-free sets," *Inform. and Control*, vol. 11, no. 1, pp. 3–29, 1967.

[8] J. W. Thatcher, "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory," *J. Comput. Syst. Sci.*, vol. 1, pp. 317–322, 1967.

[9] J. W. Thatcher and J. B. Wright, "Generalized finite automata with an application to a decision problem of second-order logic.," *Math. Sys. Theory*, vol. 2, pp. 57–82, 1968.

[10] W. C. Rounds, *Trees, transducers, and transformations*. PhD thesis, Stanford University, 1968.

[11] W. C. Rounds, "Mappings and grammars on trees," *Math. Sys. Theory*, vol. 4, no. 3, pp. 257–287, 1970.

[12] F. Drewes, "TREEBAG—a tree-based generator for objects of various types," Report 1/98, Univ. Bremen, 1998.

[13] J. May and K. Knight, "Tiburon: A weighted tree automata toolkit," in *Proceedings of the 11th International Conference of Implementation and Application of Automata, CIAA 2006* (O. H. Ibarra and H.-C. Yen, eds.), vol. 4094 of *Lecture Notes in Computer Science*, (Taipei, Taiwan), pp. 102–113, Springer, August 2006.

[14] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, "Tree automata techniques and applications." Available on: http://www.grappa.univ-lille3.fr/tata, 1997. Release October, 1rst 2002.

[15] E. Jönsson, "Evaluating a plan-recognition framework using tree automata," Tech. Rep. UMNAD 895, Umeå University, 2012.

[16] M. Sipser, *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.