A TOOL ENVIRONMENT FOR SPECIFYING AND VERIFYING MULTI-AGENT SYSTEMS

Christian Schwarz, Ammar Mohammed
Universität Koblenz-Landau, Computer Science Department, 56070 Koblenz, Germany

Frieder Stolzenburg
Hochschule Harz, Automation and Computer Sciences Department, 38855 Wernigerode, Germany

Keywords: Multi-agents specification, Verification, Hybrid automata.

Abstract: We present a tool environment with a constraint logic programming core, that allows us to specify multi-agent systems graphically and verify them automatically. This combines the advantages of graphical notations from software engineering and formal methods. We demonstrate this on a Robocup rescue scenario.

1 INTRODUCTION

Specifying the behavior of multi-agent systems (MAS) in safety critical environments is a demanding task. Henzinger (1996) introduced Hybrid Automata (HA) which have been used to model and verify MAS, especially because of their capability to catch both, the continuous dynamics of physical systems and the discrete evolution of computational systems. Mohammed and Stolzenburg (2008, 2009) mention several of application areas and benchmarks. With the help of verification tools like HyTech (Henzinger et al., 1995) and PHAVer (Frehse, 2005) one can verify and control multi-agent plans. But the specification and verification of MAS behaviors by means of hybrid automata is still a challenging task.

One of the problems, which is especially difficult for MAS, is that the state space grows exponentially in the number of states in the composed automata. Mohammed and Stolzenburg (2008) have presented an approach based on constraint logic programming (CLP) which tackles this problem. However, so far a graphical user-interface for the definition of MAS has not been available. Therefore, this paper aims at the simplification of the specification process by taking advantage of the graphical notations taken from software engineering more directly. Mohammed and Schwarz (2009) presented a prototype, that allows to enter the model of a hybrid automaton and its requirements graphically and which automates the verification process. Here, we present an extension of this tool and demonstrate it in a scenario from the RoboCup rescue scenario.

2 HYBRID AUTOMATA

We describe MAS by means of several hybrid automata, where a single agent is represented by one automaton. This section briefly into their the syntax and semantics. A *hybrid automaton* is represented graphically as a state transition diagram like statecharts in the unified modeling language (UML) Object Management Group, Inc. (2009), augmented with mathematical annotations on transitions and locations. Formally speaking, a hybrid automaton, which represents an agent in a continuous domain, is defined as follows:

Definition 1: A *hybrid automaton* is a tuple $H = (X, Q, Inv, Flow, E, Jump, Reset, Event, \sigma_0)$ where:

- X is a set of real variables that describe the continuous dynamics of the automaton.
- Q is a finite set of control locations.
- $Inv(q)$ is the invariant predicate, which assigns a constraint on the valuation to each location.
- $Flow(q)$ is the flow predicate on the valuation for each location. In the graphical representation, a flow of a variable x is denoted as $\dot{x}$.

Schwarz C., Mohammed A. and Stolzenburg F. (2010).
A TOOL ENVIRONMENT FOR SPECIFYING AND VERIFYING MULTI-AGENT SYSTEMS.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Agents*, pages 323-326
DOI: 10.5220/0002727703230326

- $E \subseteq Q \times Q$ is the discrete transition relation over the control locations. Each edge may have the following annotations:

  Jump denotes a jump condition (guard), which is a constraint over the variables that must hold to fire transitions. Omitting a jump condition means, that it is true.

  Reset is a constraint, which may reset the variables by executing a specific assignments. In the graphical representation, $x' = v$ denotes that the variable x is reset to the value v. Omitting Reset means, that the variables are not changed.

  Event is a synchronization label, used to synchronize and coordinate concurrent automata.

- $\sigma_0 \in Q \times \mathbb{R}^n$ is the initial state.

The semantics of a hybrid automaton is defined in terms of a labeled transition system between states, where a state is defined as follows:

Definition 2: The *state* σ of a hybrid automaton is given by $\sigma = \langle q, \mathbf{v} \rangle \in Q \times \mathbb{R}^n$ where $\mathbf{v} = (a_1, \ldots, v_n)$ and v_i assigns a value to each variable $x_i \in X$. The state is *admissible* iff $Inv(q)[v]$ holds.

A state transition system of a hybrid automaton H starts with the *initial state* σ_0. It then evolves depending on two kinds of transitions: continuous transitions, capturing the continuous evolution of states, and discrete transitions, capturing the changes of location. More formally, we can define the operational semantics as follows:

Definition 3: For each transition between two admissible states $\sigma_1 = \langle q_1, v_1 \rangle$ and $\sigma_2 = \langle q_2, v_2 \rangle$ one of the following condition holds:

(discrete) There is an edge $e = (q_1, q_2) \in E$, $Jump(e)[v_1]$ holds and the variables are reset according to $Reset(e)$. In addition an event $a \in Event(e)$ occurs. A discrete transition can be written as $q_1 \xrightarrow{a} q_2$.

(continuously) In this case $q_1 = q_2$ holds, Δt is the time spent in q_1 and v_1 and v_2 are valuations of the variables according to $Flow(q_1)$ with respect to Δt. The invariant predicate $Inv(q_1)$ must hold continuously during Δt.

The continuous evolution generates an infinite number of reachable states. Thus the state-space exploration techniques require an appropriate symbolic representation for sets of states. We represent the infinite states symbolically as finite intervals which are called regions.

Definition 4: A *region* $\Gamma = \langle q, V \rangle$ is the set of possible states reached at location q by means of continuous transitions, where V represent an interval of reached valuations of the variables. A region Γ is admissible if $inv(q)[v]$ holds for all $v \in V$.

Now, the run of hybrid automata can be defined as a form of reached regions, where the change from one region to another is induced using a discrete step.

Definition 5: A *run* of hybrid automaton H is $\sum_H = \Gamma_0\Gamma_1, ...$, a (possibly infinite) sequence of admissible regions, where a transition from a region Γ_i to a region Γ_{i+1} is enabled (written as $\Gamma_i \xrightarrow{a} \Gamma_{i+1}$), if $q_i \xrightarrow{a} q_{i+1}$, where $a \in Event$ is the generated event before the control goes to the region Γ_{i+1}. Γ_0 is the initial region reached from a start state σ_0 by means of continuous transitions.

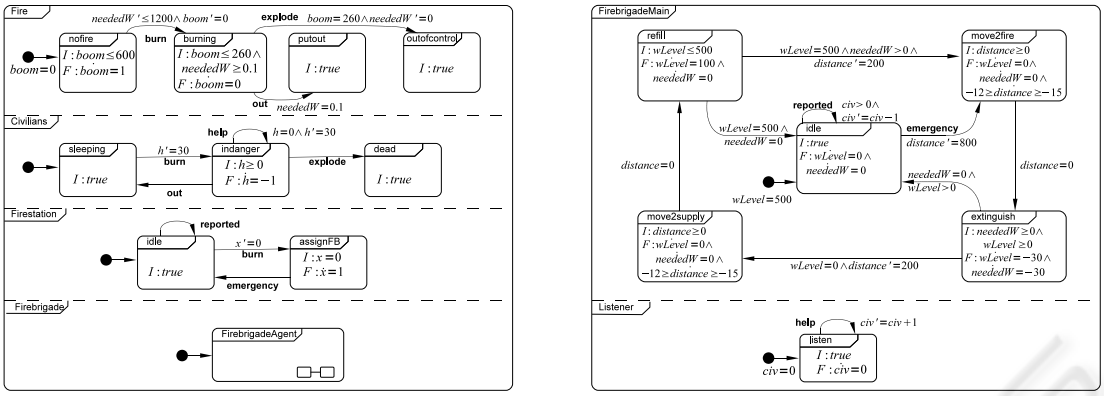
The operational semantics is the basis for verification of a hybrid automaton. In particular, model checking of a hybrid automaton is defined in terms of the reachability analysis of its underlying transition system. The most useful question to ask about hybrid automata is the reachability of a given state. Thus, we define the reachability of states as follows.

Definition 6: A region Γ_i is called *reachable* in $\sum_H$, if $\Gamma_i \subseteq \sum_H$. Consequently, a state σ_j is called reachable, if there is a reached region Γ_i such that $\sigma_j \in \Gamma_i$

To specify MAS, hybrid automata can be composed in parallel, where a hybrid automaton is given for each agent in the MAS, and communication between the different agents may occur via shared variables and synchronization labels. Technically, the parallel composition of hybrid automata is obtained from the different parts using a Cartesian product construction (composition) of the participating automata. The transitions from the different automata are interleaved, unless they share the same synchronization label. In this case, they are synchronized during the execution. Differently to most of hybrid automata techniques, Mohammed and Stolzenburg (2008) show how the composition of hybrid automata can be constructed on the fly. Additionally, they presented a CLP approach to encode the previous semantics.

3 THE RESCUE EXAMPLE

In the RoboCup rescue simulation league (Tadokoro et al., 2000) a team of heterogeneous agents is simulated in a city which is partly destroyed by an earthquake. The agents (e.g. police, fire brigade and am-



(a) The environment (b) The fire brigade agent

Figure 1: A scenario from the RoboCup rescue simulation league modeled as hybrid automata.

bulance) can solve one specific task only. So the need for coordination and synchronization is obvious.

3.1 Specification

Consider the following simple scenario. When a fire breaks out somewhere in the city, a fire brigade agent is ordered by its headquarters to extinguish the fire. If the agent runs out of water it has to refill its tank at a supply station and return to the fire to complete its task. When the fire is put out, the brigade agent will become idle again and wait for its next operation. Additionally the agent has to report any discovered injured civilian. A limiting factor is, that the fire will eventually get out of control, so it must be extinguished within a specified time limit.

The environment is modeled in Fig. 1(a). The fire will start within the first 10 minutes of the simulation. The variable *neededW* represents the amount of water that is needed to put it off. The civilians are modeled to be sleeping initially. When the fire breaks out, they will wake up and call for help periodically. The task of the fire station is to assign a fire brigade to a fire as soon as it is discovered.

The fire brigade agent is modeled in Fig. 1(b). It consists of two parts: the main control structure *FirebrigadeMain* and *Listener* that counts the number of discovered civilians. *FirebrigadeMain* starts in *idle* and jumps to *move2fire*, which models the moving towards the fire, when it is assigned by the fire station. The distance between fire and the the fire brigade is modeled by the variable *distance*.

After it has arrived at the fire, the fire brigade tries to extinguish it. This is modeled by decreasing the value of *wLevel* (the water in the tank) , and *neededW* (the water needed to put out the fire) by the same rate. If the water in the tank runs out, the fire brigade has to move to the next refill station. After the tank is refilled, the fire brigade moves towards the fire again. After the fire is put off or is got out of control, the fire brigade becomes idle again and can then report any found civilians.

3.2 Verification

Using this model, we can now demonstrate some exemplary model checking tasks. We can examine if certain control locations are reachable. So we can ask: “Is it possible to extinguish the fire?” (*putout* is reachable), or “Won’t the fire get out of control?”, (*outofcontrol* is not reachable). It is also possible, to check the reachability of composed locations. This allows questions like “Won’t the fire brigade move to the fire if it is not burning?” Then, no composed locations are reachable, where Firebrigade is in *move2fire* and Fire is in *nofire*, *putout* or *outofcontrol*. Furthermore, we can check the reachability of certain intervals in the continuous valuation of the automaton, e.g.: “Won’t the agent try to extinguish with an empty water tank?” This would hold if *extinguish* is active while $wLevel < 0$.

Finally, we can check properties of runs. This allows more complex questions like “Does the agent report all discovered civilians?” This question contains two properties to be checked: (a) all discovered civilians are reported eventually and (b) the agent does never report a civilian than he did not find. Property (a) corresponds to the fact that from every reachable state there is a state reachable where all discovered civilians have been reported. Property (b) holds if in the history of each reachable state the number of transitions labeled with *help* is always greater or equal than the number of transitions labeled with *reported*.

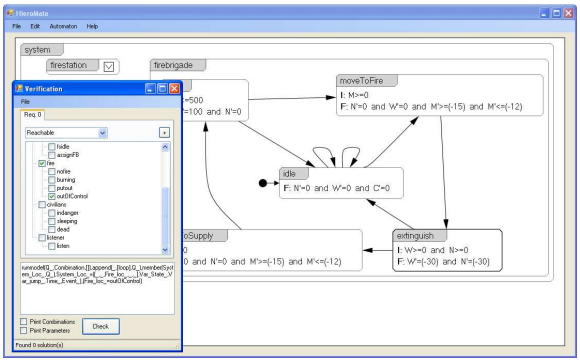


Figure 2: A screenshot of HieroMate while verifying the fire brigade example (some of the automata are hidden).

4 THE HIEROMATE TOOL

Mohammed and Schwarz (2009) initially presented a tool for graphical specification and verification of hierarchical hybrid automata (HHA), that include hierarchical specifications as known from UML statecharts, such that the overall system can be expressed on several levels of abstraction. In HHA, locations are generalized (cf. Mohammed and Stolzenburg, 2008): The set of all locations Q is partitioned simple, composite and concurrent locations. In essence, the locations of plain hybrid automata correspond to simple locations in HHA.

Now, in this paper, the concurrent view based on the outline in Mohammed and Stolzenburg (2008) has also been plugged into the tool, such that a user can select which type of view is needed to be modeled. This tool works as a front end for a model checking engine that is written in the CLP language ECLiPSe Prolog (Apt and Wallace, 2007). The tool assists the user in specifying hybrid automata (and MAS) by supporting graphical specification, on-the-fly syntax checking, and automated CLP code generation.

The user interaction is realized mainly using context sensitive menus that allow only meaningful actions e.g. the user will be able to add another location to an automaton by right clicking onto the automaton and selecting the item “Add location” from the context menu. The specification can then be checked directly in the tool. Therefore the user can either specify queries manually using CLP Prolog, use the tool to generate simple queries automatically, or combine both methods. Fig. 2 shows a screenshot of HieroMate while verifying the rescue example.

5 CONCLUSIONS

In this paper, we presented a tool environment with a constraint logic programming core that is able to graphically specify and formally verify MAS in terms of hybrid automata, where the graphical specification and a requirement can be given to the tool and it will convert them into a specification written in CLP. Then, the resulting CLP specification will be checked using an abstract state machine in terms of reachability analysis automatically. The paper has demonstrated this on a MAS scenario taken from rescue scenario.

REFERENCES

Apt, K. R. and M. Wallace (2007). *Constraint Logic Programming Using ECLiPSe*. Cambridge, UK: Cambridge University Press.

Frehse, G. (2005). PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele (Eds.), *Hybrid Systems: Computation and Control, 8th International Workshop, Proceedings*, LNCS 3414, pp. 258–273. Springer.

Henzinger, T. (1996). The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, New Brunswick, NJ, pp. 278–292. IEEE Computer Society Press.

Henzinger, T., P.-H. Ho, and H. Wong-Toi (1995). HyTech: The Next Generation. In *IEEE Real-Time Systems Symposium*, pp. 56–65.

Mohammed, A. and C. Schwarz (2009). HieroMate: A graphical tool for specification and verification of hierarchical hybrid automata. In B. Mertsching, M. Hund, and Z. Aziz (Eds.), *KI 2009: Advances in Artificial Intelligence, Proceedings of 32nd Annual German Conference on Artificial Intelligence*, LNAI 5803, Paderborn, pp. 695–702. Springer, Berlin, Heidelberg, New York.

Mohammed, A. and F. Stolzenburg (2008). Implementing hierarchical hybrid automata using constraint logic programming. In S. Schwarz (Ed.), *Proceedings of 22nd Workshop on (Constraint) Logic Programming*, Dresden, pp. 60–71. University Halle Wittenberg, Institute of Computer Science. Technical Report 2008/08.

Mohammed, A. and F. Stolzenburg (2009). Using constraint logic programming for modeling and verifying hierarchical hybrid automata. Arbeitsberichte des Fachbereichs Informatik 6/2009, Universität Koblenz-Landau.

Object Management Group, Inc. (2009). *OMG Unified Modeling Language (OMG UML): Infrastructure; Superstructure*. Object Management Group, Inc.

Tadokoro, S. et al. (2000). The RoboCup-Rescue project: A robotic approach to the disaster mitigation problem. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2000)*, pp. 4089–4104.