Global State Checker: Towards SAT-Based Reachability Analysis of Communicating State Machines*

Petra Kaufmann[1], Martin Kronegger[2], Andreas Pfandler[2],
Martina Seidl[1,3], and Magdalena Widl[4]

[1] Business Informatics Group, TU Wien
[2] Database and Artificial Intelligence Group, TU Wien
[3] Institute for Formal Models and Verification, JKU Linz
[4] Knowledge-Based Systems Group, TU Wien
{firstname.lastname@tuwien.ac.at}

Abstract. We present a novel propositional encoding for the reachability problem of communicating state machines. The problem deals with the question whether there is a path to some combination of states in a state machine view starting from a given configuration. Reachability analysis finds its application in many verification scenarios. By using an encoding inspired by approaches to encode planning problems in artificial intelligence, we obtain a compact representation of the reachability problem in propositional logic. We present the formal framework for our encoding and a prototype implementation. A first case study underpins its effectiveness.

1 Introduction

In model-based engineering (MBE), software models take over the role as core development artifact, which textual code has in traditional software engineering. The goal of MBE is to leverage the abstraction power of models in order to deal with the complexity of modern software systems [2]. Executable code is to be directly generated from the models with little or no intervention of a developer [12]. With this valorization of software models, stronger requirements on their correctness come along. As a consequence, formal methods found their way into MBE to verify models, often by reusing techniques successfully applied for the verification of traditional software systems. Among the most successful techniques to verify hardware and software systems are approaches based on *model checking* [4], which exhaustively traverse the states of a system to answer questions related to the reachability of certain states from an initial configuration. In order to deal with large state spaces, symbolic methods have been introduced to compactly encode state spaces. Thereby propositional logic turned out to be particularly useful. The success of model checking is closely connected to the observation that in many cases it is sufficient to show correctness only for a restricted number of execution steps, which resulted in the method of bounded model checking [3].

In the context of MBE, model checking is used for the verification of behavioral models like UML state machines. For this purpose, many encodings of state machines have been proposed which translate the state machines to the input format of the model checkers. Usually,

* This work was partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018 and by the Austrian Science Fund (FWF): P25518-N23.

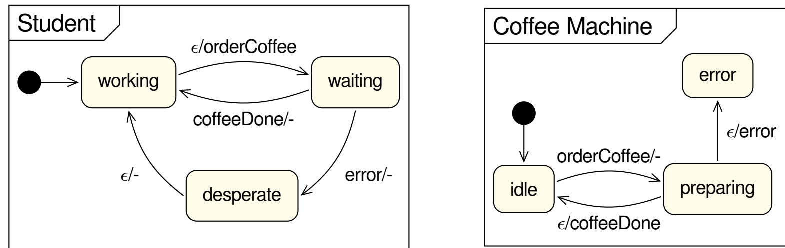

Fig. 1. State machines of a student and a coffee machine.

the model checkers provide languages to describe finite state automata, which are also the conceptual basis of state machines. However, it still can be challenging to find a semantics-preserving translation as even similar concepts may strongly diverge in their semantics. In this paper, we propose to directly encode the reachability problem for composite state machines into the problem of satisfiability of propositional logic (SAT). The motivation behind this approach is an integration into our formal MBE framework [15] where we successfully used SAT technologies in the context of optimistic model versioning. Our current encoding reuses ideas from SAT-based planning [11] to encode the search of paths between global states of a state machine view.

This paper is structured as follows. First, we review related work in Sec. 2. Then we provide a concise problem definition in Sec. 3 which serves as basis for our encoding presented in Sec. 4. In Sec. 5 we introduce our Eclipse-based implementation and report on a first case study. Finally, we conclude with an outlook to future work.

2 Related Work

Several works have been presented which deal with the transformation of UML state machines to input languages of model checkers (see for example in [1,5,6,8,10]). These languages provide high-level constructs to model software systems and in many aspects they provide similar constructs as do modeling languages like UML, in particular UML state machines.

However, one of the major challenges of such approaches is to overcome semantical heterogenities, which raises the question if this translation step is really required. This has already been recognized by Niewiadomski et al., who propose an encoding to propositional logic for bounded reachability analysis of state machines, which they show to be more efficient than translations to standard model checkers [9]. In this paper, we follow the approach of [9] to encode the reachability problem to SAT, but propose an alternative encoding where we formulate the reachability analysis problem of UML state machines inspired by encodings as used for solving planning problems [11]. As a result, we obtain an intuitive encoding which allows to directly extract a path of a length bounded by k from a solution without the need of looping through values lower than the bound.

3 Problem Definition

To motivate our approach consider the following example. Fig. 1 shows the state machines of a student and a coffee machine implementing a simplified workflow of a student's interaction

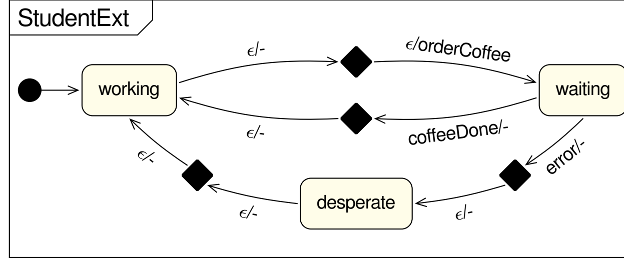

Fig. 2. Extended state machine of the student machine.

with a coffee machine. The initial state of each state machine is indicated by an arrow from a black filled circle. States are visualized as rounded rectangles and are connected to each other by transitions. Each transition carries a label consisting of a symbol called *trigger* to the left, and a set of symbols called *effects* to the right of a "/". The empty trigger ϵ indicates that the transition can be executed without receiving any trigger symbol. This symbol can be used to model an on-completion event. The symbol "–" represents an empty set of effects. The receipt of the trigger symbol causes the state machine to change its current state from the source state to the target state of the transition. The symbols in the set of effects are sent during the execution of the transition. The communication among state machines is synchronous and therefore the execution of a transition is only possible if each of its effects is understood by a different state machine in its current state.

For a given set of state machines, the *Global State Checking* (GSC) problem asks whether a certain combination of states of the state machines can be reached from an initial configuration. In this paper we consider the *k-Global State Checking (k-*GSC*) problem*. Given a set $\mathcal{M}$ of state machines which communicate over an alphabet $\mathcal{A}$, the k-GSC problem asks whether a certain combination of states of the state machines can be reached from an initial configuration by a path with a length of at most k. For example, it can be checked whether the combination of the states working in state machine student and preparing in state machine coffee machine is reachable starting from the initial states through a path of length 10, or whether the combination of waiting and error is not reachable by a path of length 1000. In the following, we present a precise definition of the semantics of a state machine view and of the k-GSC problem. We start by defining a state machine as follows:

Definition 1 (State Machine). *Given an alphabet $\mathcal{A}$, a* state machine *M is a quintuple $(S,\iota,A^{tr},A^{eff},T)$, where S is a set of* states*, $\iota \in S$ is a designated* initial state*, $A^{tr} \subseteq \mathcal{A}$, $A^{eff} \subseteq \mathcal{A}$, and $T \subseteq S \times A^{tr} \cup \{\epsilon\} \times \mathcal{P}(A^{eff}) \times S$.*

A state machine consists of a set of states, two alphabets, and a transition relation between the states. For a transition $t \in T$ with $t = (s, tr, eff, s')$, s is the source state of the transition, s' is the target state, tr is a symbol (trigger) which upon receipt triggers the execution of the transition, and eff is a set of symbols (effects) that are sent when the transition is executed. The trigger symbol can be the special symbol $\epsilon \notin \mathcal{A}$ standing for an empty trigger. A transition containing ϵ can be triggered no matter whether any symbol is received. In order for the execution of the transition to finish, each symbol in eff must be received by a different state machine. In Fig. 1, state machine Student contains states $S = \{$working,desperate,waiting$\}$,

triggers $\mathcal{A}^{tr} = \{\mathsf{coffeeDone}, \mathsf{error}\}$ and effect $\mathcal{A}^{\mathit{eff}} = \{\mathsf{orderCoffee}\}$. An example for a transition is $(\mathsf{working}, \epsilon, \{\mathsf{orderCoffee}\}, \mathsf{waiting})$.

In order to give a precise semantics of the interaction between state machines, we introduce the notion of an *extended* state machine.

Definition 2 (Extended State Machine). *Given a state machine $M = (S, \iota, A^{tr}, A^{\mathit{eff}}, T)$, the* extended state machine *M^* is a quintuple $(S \cup S^*, \iota, A^{tr}, A^{\mathit{eff}}, T^*)$ where $S^* = \{s_t^* \mid t \in T\}$ and $T^* = \{(s, tr, \emptyset, s_t^*), (s_t^*, \epsilon, \mathit{eff}, s') \mid t = (s, tr, \mathit{eff}, s') \in T\}$.*

An extended state machine introduces an *intermediate state* s_t^* for each transition t. This intermediate state has one incoming transition, which is triggered by the trigger of t and has no effects. It also has one outgoing transition, which leads to the target state of t with ϵ as trigger and the effects of t. The states contained in S^* we call *extended states*. Fig. 2 depicts the extended state machine constructed from state machine Student in Fig. 1. The reason for the construction of an extended state machine is to distinguish between the event of having received the trigger and the event of being able to send the effects.

The communication between state machines takes place through a structure called *message set*. A message set contains a sender state machine and a set of pairs, each containing a symbol sent by the sender state machine and a receiving state machine. Each of the symbols is sent by the same state machine but received by a different state machine. This is captured in the following definition.

Definition 3 (Message Set). *Given a set $\mathcal{M} = \{M_1^*, ..., M_l^*\}$ of extended state machines with $M_i^* = (S_i, \iota_i, A_i^{tr}, A_i^{\mathit{eff}}, T_i)$ for $1 \le i \le l$, a* message set *is a pair $(\sigma, \{(a_1, R_1^*), ..., (a_k, R_k^*)\})$ where*

- *$\sigma = M_d^*$ with $1 \le d \le l$ and*
- *$\{(a_1, R_1^*), ..., (a_k, R_k^*)\} \in \mathcal{P}(A_d^{\mathit{eff}} \times \mathcal{M} \setminus \{\sigma\})$*

such that for $1 \le i \le k$ all R_i^* are pairwise distinct.

For a message set $(\sigma, \{(a_1, R_1^*), \ldots, (a_k, R_k^*)\})$, σ is an extended state machine which executes a transition leaving an extended state with the set $\{a_1, ..., a_k\}$ of effects, and for each $1 \le i \le k$, R_i^* is an extended state machine which executes a transition leaving an original (non-extended) state through trigger a_i. Note that $\{(a_1, R_1^*), ..., (a_k, R_k^*)\}$ can be the empty set, which represents an empty set of effects on a transition.

A message set can be *admissible* in some global configuration. Such a configuration is given by a *global state*, a tuple of states containing exactly one state per state machine. By applying a message, a *global successor state* is reached.

Definition 4 (Application of a Message Set). *Given a set of extended state machines $\mathcal{M} = \{M_1^*, ..., M_l^*\}$ with $M_i^* = (S_i, \iota_i, A_i^{tr}, A_i^{\mathit{eff}}, T_i)$ for $1 \le i \le l$, and a global state $\hat{s} = (s_1, ..., s_l) \in S_1 \times \cdots \times S_l$, a message set $m = (M_d^*, \{(a_1, R_1^*), ..., (a_k, R_k^*)\})$, with $1 \le d \le l$ and $1 \le k < l$, is* admissible *in $\hat{s}$ if*

(i) $(s_d, \epsilon, \{a_1, ..., a_k\}, s_d') \in T_d$, and
(ii) there exists a set $\mathcal{R} \subseteq \{1, ..., l\} \setminus \{d\}$ and a bijective function $\mathsf{rec} : \{1, ..., k\} \to \mathcal{R}$ such that $R_j^* = M_{\mathsf{rec}(j)}^*$ and $(s_{\mathsf{rec}(j)}, a_j, \emptyset, s'_{\mathsf{rec}(j)}) \in T_{\mathsf{rec}(j)}$ for each $1 \le j \le k$.

Given a global state $\hat{s}$ and a message set m that is admissible in $\hat{s}$, a global successor state $\hat{s}'$ *of* $\hat{s}$ after applying m *is given by* $\hat{s}' = (\mathsf{next}(s_1),...,\mathsf{next}(s_l))$ *where*

$$\mathsf{next}(s_i) = \begin{cases} s'_d & \text{if } i{=}d \\ s'_i & \text{if } i \in \mathcal{R} \\ s_i & \text{otherwise} \end{cases}$$

There are two requirements for a message set to be admissible in a global state: (1) the sender's state in the global state is an extended state with an outgoing transition containing the set $\{a_1,...,a_k\}$ of effects, and (2) each receiver's state in the global state has an outgoing transition triggered by the respective symbol from the message set. Note that we are dealing with extended state machines, which means that a transition cannot carry a trigger symbol other than ϵ together with a non-empty set of effects. Therefore it can never happen that a receiver state machine R_i^* sends any effects while executing the transition triggered by some symbol a_i. The global successor state $\hat{s}'$ is reached by applying a message set. It differs from $\hat{s}$ in states of the sender and the receiver state machines contained in the applied message set: The sender's state changes from an extended state to its only successor state, and the receivers' states change according to the received symbol into an extended state.

We combine message sets of disjoint sets of state machines in a *transaction* as follows.

Definition 5 (Transaction). *A* transaction *is a nonempty set of message sets $\{m_1,...,m_l\}$ with $m_i = (\sigma_i,\{(a_{i,1},R^*_{i,1}),...,(a_{i,k_i},R^*_{i,k_i})\})$ such that all state machines occurring in the message sets, i. e., all σ_i and $R^*_{i,j}$ (with $1 \le i \le l, 1 \le j \le k_i$), are pairwise distinct.*

A transaction is admissible if all its message sets are admissible. The global state reached by applying a transaction is the global state reached by applying each of its message sets.

We further define a path as a sequence of transactions as follows.

Definition 6 (Path). *A* path μ from a global state $\hat{s}_0$ to a global state $\hat{s}_k$ *is a sequence $\mu = [n_1,...,n_k]$ of transactions such that there exists a sequence $[\hat{s}_0,...,\hat{s}_k]$ of global states where for all $1 \le i \le k$, n_i is admissible in state $\hat{s}_{i-1}$ and $\hat{s}_i$ is the global successor state of $\hat{s}_{i-1}$ after applying n_i.*

A global state $\hat{s}_j$ *is reachable from* $\hat{s}_i$ if there is a path from $\hat{s}_i$ to $\hat{s}_j$. The *length* of a path is the number of its transactions.

The k-GSC problem deals with reaching a combination of states of state machines in a state machine view. This combination contains at most one state of each state machine. Hence such a combination not necessarily specifies a complete global state. We therefore define a partial global state as follows.

Definition 7 (Partial Global State). *Given a set $\mathcal{M} = \{M_1^*,\dots,M_l^*\}$ of extended state machines with $M_i^* = (S_i,\iota_i,A_i^{tr},A_i^{eff},T_i)$ for $1 \le i \le l$, a* partial global state *is an l-tuple $\hat{s}_p \in S_1 \cup \{?\} \times \cdots \times S_l \cup \{?\}$, where ? is a new symbol not contained in any S_i. $\hat{s}_p = (s_1,...,s_l)$* matches *a global state $\hat{s} = (q_1,...,q_l)$ if for all $1 \le i \le l$, $s_i = q_i$ whenever $s_i \ne ?$.*

Finally, we define the k-GSC problem as follows.

k-GSC	
Instance:	A set $\mathcal{M}$ of state machines, a global state $\hat{s}$, and a partial global state $\hat{s}_p$.
Question:	Is there a path of length at most k from $\hat{s}$ to a global state $\hat{s}'$ that matches $\hat{s}_p$?

The global state $\hat{s}$ is also referred to as *initial state* and the partial global state $\hat{s}_p$ is also referred to as *goal*. The initial state usually contains each state machine's initial state.

4 Encoding

In order to find solutions to the k-GSC problem, we propose to encode it to the satisfiability problem of propositional logic (SAT) and hand it to a SAT solver. In the following we present a detailed description of the SAT formula representing the k-GSC problem. In the next section we describe our tool, the Global State Checker, which builds upon this encoding.

Let k be a positive integer, $\hat{s}$ a global state representing an initial state, and $\hat{s}_p$ a partial global state representing a goal. Then the formula φ is satisfiable if and only if there is a global state $\hat{s}'$ that is reachable from $\hat{s}$ by a path of length at most k and that matches $\hat{s}_p$.

Recall that in a k-GSC instance we are given a set $\mathcal{M} = \{M_1, \dots, M_l\}$ of state machines, a global state $\hat{s} = (x_1, \dots, x_l)$, and a partial state $\hat{s}_p = (g_1, \dots, g_l)$. For each $1 \leq i \leq l$ let $M_i = (S_i, \iota_i, A_i^{tr}, A_i^{eff}, T_i)$ and the corresponding extended state machine be $M_i^* = (S_i \cup S_i^*, \iota_i, A_i^{tr}, A_i^{eff}, T_i^*)$.

In order to define the set of variables of φ, we introduce a set $\mathcal{T} \coloneqq \bigcup_{1\leq i\leq l} T_i$ of transitions, a set $\mathcal{A} \coloneqq \bigcup_{1\leq i\leq l} \left(A_i^{tr} \cup A_i^{eff}\right)$ of symbols, a set $\mathcal{S} \coloneqq \bigcup_{1\leq i\leq l} S_i$ of states, and a set $\mathcal{S}^* \coloneqq \bigcup_{1\leq i\leq l} S_i^*$ of extended states. Then the set of variables is given by $\{v^i \mid v \in (\mathcal{T} \cup \mathcal{A} \cup \mathcal{S} \cup \mathcal{S}^*), 0 \leq i \leq k\}$ were i is an index capturing the relative position of the variable in the path. That is, each transition, symbol, state, and extended state together with one index up to k represents a variable.

Let $t = (s, tr, eff, s')$ be a transition of a state machine and $(s, tr, \emptyset, s_t^*)$ and $(s_t^*, \epsilon, eff, s')$ be the corresponding transitions in the respective extended state machine. To simplify the presentation, we use the functions $\mathsf{src}(t) \coloneqq s$, $\mathsf{int}(t) \coloneqq s_t^*$, $\mathsf{trg}(t) \coloneqq tr$, $\mathsf{eff}(t) \coloneqq eff$, and $\mathsf{tgt}(t) \coloneqq s'$.

Further let $\mathcal{T}^* \coloneqq \bigcup_{1\leq i\leq l} T_i^*$. Given a state $s \in \mathcal{S}$, let $\mathsf{environ(s)} \coloneqq \{s^* \mid (s^*, \epsilon, \emptyset, s) \in \mathcal{T}^*\} \cup \{s^* \mid (s, \epsilon, eff, s^*) \in \mathcal{T}^*\}$ be a set of extended states containing a predecessor of s if the transition does not contain any effects and a successor of s if the transition contains ϵ as trigger.

The formula φ is then given by a conjunction of the following subformulas:

$$\varphi_{\text{init}} \coloneqq \bigwedge_{i=1}^{l} \Bigg(\bigwedge_{s\in\mathcal{S}_i, s=x_i} s^0 \wedge \bigwedge_{s\in\mathcal{S}_i\cup\mathcal{S}_i^*, s\neq x_i} \overline{s}^0 \wedge \bigwedge_{a\in\mathcal{A}} \overline{a}^0 \Bigg)$$

$$\varphi_{\text{goal}} \coloneqq \bigwedge_{i=1, g_i\neq ?}^{l} \Bigg(g_i^k \vee \bigvee_{s\in\mathsf{environ(g_i)}} s^k \Bigg)$$

$$\varphi_1 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{t\in\mathcal{T}} \Bigg[t^i \rightarrow \Bigg(\mathsf{src}(t)^i \wedge \mathsf{int}(t)^{i+1} \wedge \mathsf{trg}(t)^i \wedge \overline{\mathsf{trg}(t)}^{i+1} \quad \wedge \bigwedge_{eff\in\mathsf{eff}(t)} \Big(\overline{eff}^i \wedge eff^{i+1} \Big) \Bigg) \Bigg]$$

$$\varphi_2 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{trg\in\mathcal{A}} \Bigg[trg^i \wedge \overline{trg}^{i+1} \rightarrow \bigvee_{t\in\mathcal{T}, trg=\mathsf{trg}(t)} t^i \Bigg]$$

$$\varphi_3 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{\mathit{eff} \in \mathcal{A}} \left[\overline{\mathit{eff}}^{i} \wedge \mathit{eff}^{i+1} \rightarrow \bigvee_{t \in \mathcal{T}, \mathit{eff} \in \mathsf{eff}(t)} t^{i} \right]$$

$$\varphi_4 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{s \in S} \left[s^{i} \wedge \overline{s}^{i+1} \rightarrow \bigvee_{t \in \mathcal{T}, s = \mathsf{src}(t)} t^{i} \right]$$

$$\varphi_5 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{t \in \mathcal{T}} \left[\mathsf{int}(t)^{i} \wedge \overline{\mathsf{int}(t)}^{i+1} \rightarrow \bigwedge_{\mathit{eff} \in \mathsf{eff}(t)} \overline{\mathit{eff}}^{i+1} \right]$$

$$\varphi_6 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{t \in \mathcal{T}} \left[\mathsf{int}(t)^{i} \wedge \mathsf{int}(t)^{i+1} \rightarrow \bigwedge_{\mathit{eff} \in \mathsf{eff}(t)} \mathit{eff}^{i+1} \right]$$

$$\varphi_7 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{t \in \mathcal{T}} \left[\left(\mathsf{int}(t)^{i} \wedge \bigwedge_{\mathit{eff} \in \mathsf{eff}(t)} \overline{\mathit{eff}}^{i+1} \right) \rightarrow \left(\overline{\mathsf{int}(t)}^{i+1} \wedge \mathsf{tgt}(t)^{i+1} \right) \right]$$

$$\varphi_8 \coloneqq \bigwedge_{i=0}^{k-1} \bigwedge_{j=1}^{l} \left[\left(\bigvee_{s \in (S_j \cup S_j^*)} s^{i} \right) \wedge \bigwedge_{s_1, s_2 \in (S_j \cup S_j^*), s_1 \neq s_2} \left(\overline{s_1}^{i} \vee \overline{s_2}^{i} \right) \right].$$

The intuition behind these subformulas is as follows: φ_{init} initialises the path by setting the initial states with index 0 to true, and all other states and all symbols to false. φ_{goal} encodes the goal states and the extended states in their environment for index k. For all other path indices, a symbol variable in its positive polarity means that the respective symbol has been made available as effect through the transaction at the respective index and is waiting to be consumed by some transition as a trigger in a later transaction. When the symbol has been consumed, the respective symbol variable occurs in its negative polarity. φ_1 ensures that whenever a transition is executed, the state machine changes to the respective extended state. Then the trigger symbol is set its negative polarity and the effect symbols are set to their positive polarity. φ_2 and φ_3 express that whenever the polarity of a symbol is changed, there must be a transition causing this change. φ_4 encodes that leaving a state is always caused by some transition. φ_5 and φ_6 ensure that after executing a transaction either all effect symbols of a transition are consumed or none of them. The formulas φ_j with $j \in \{2,...,5\}$ are also called *framing axioms*. These formulas ensure that every change in a global state has a cause. Note that it is not necessary to state all changes explicitly, because they are already covered implicitly by other formulas. φ_7 forces a machine to move to the target state if all effect symbols have been consumed. φ_8 expresses that each machine is in exactly one state after each transaction.

Note that the encoding allows that nothing happens, i. e., no transaction takes place at an index. It is ensured by the framing axioms that in this case, the global state remains the same. This relaxation implicitly encodes the “at most k” formulation of the problem: If at n indices nothing happens and the goal is reached at index k, it means that the length of the transaction sequence is $k-n$. The framing axioms also ensure that state machines not participating in a transaction do not change.

A solution returned by the SAT solver consists of a set of variables set to *true*. By extracting those variables that represent states and transitions (sets $\mathcal{S}$, $\mathcal{S}^*$, and $\mathcal{T}$) we obtain a path to the goal. If the length of the path is less than k, then for some consecutive indices the state variables represent the same states. The shorter path can therefore be easily extracted.

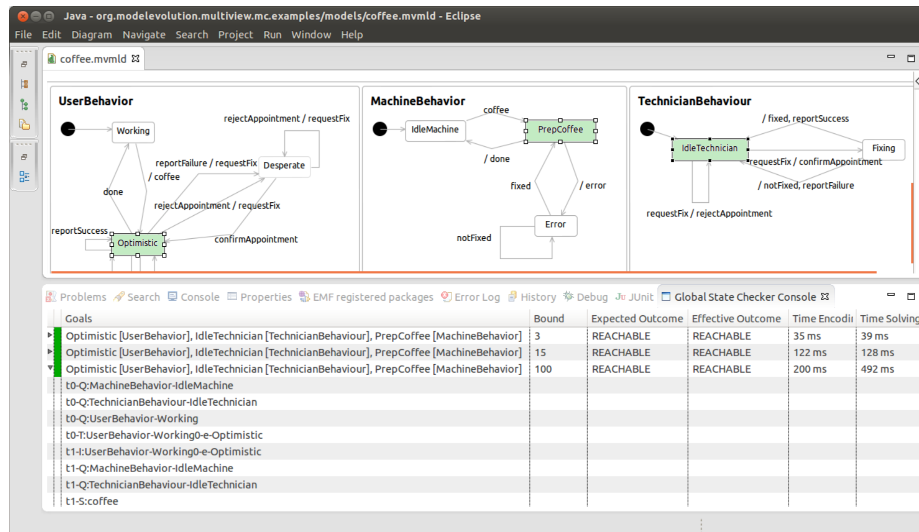

Fig. 3. UI of the Global State Checker.

In order to simplify the encoding we assume that after applying a transaction each symbol can be consumable only once. Allowing a symbol to be consumable multiple times after one transaction requires the integration of counters, which can be realized, e.g., by building upon ideas presented in [13]. We are currently developing such an extension.

5 Implementation and Case Study

We have implemented the encoding described above as Eclipse plugin[5] and designed a set of instances for an initial case study. Our prototype implementation is available on our Eclipse update site[6]. Further resources and instructions can be found on our project website[7]. In the following, we shortly describe our implementation and discuss a first case study.

Implementation. The *Global State Checker* prototype is embedded into the Eclipse modeling framework (EMF). The metamodel for the statemachine view described in Sec. 3 is provided by an Ecore metamodel. Instances of the k-GSC problem are created as models of the defined language. Our *encoder* module takes a state machine view as input and encodes it into a formula of propositional logic according to the encoding described in Sec. 4. Additional measures are taken to convert the formula into conjunctive normal form (CNF), which is a common format used by SAT solvers. The data structure representing the encoding is handed to the SAT4J [7] solver, a SAT solver written in Java, which integrates seamlessly into our tool. The SAT solver either returns `UNSAT` or `SAT`. The former means that the problem has no solution, i.e., that the specified state is not reachable in k steps. The latter case means that there exists a solution, i.e., a path from the initial configuration to a global state matching

[5] `http://www.eclipse.org/`
[6] `http://modelevolution.org/updatesite`
[7] `http://modelevolution.org/prototypes/gsc`

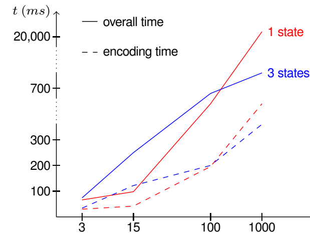

Fig. 4. Runtime measures for the coffee machine test case.

the specified goal. In this case, the SAT solver additionally returns a logical model of the formula representing the problem. A logical model assigns one of the truth values *true* and *false* to each propositional variable. Since each variable represents either a state, a transition, or a symbol with respect to a certain index, such a model can be easily translated into a path leading to the specified state. This way we retrieve a solution to our original problem.

Fig. 3 shows the graphical user interface of our prototype. The user can select a set of states directly in the modeling editor, enter a bound, and start the k-GSC tool. For convenience, it may be specified whether the given global state is expected to be reachable or not. Red or green highlighting indicate how the expected result compares to the effective result. All test cases are listed in the global state checker console. The expanded subitems of the third test case in Fig. 3 show the path found by the SAT solver.

Case Study. We have designed three different benchmark instances with varying number of state machines, states, and transitions in order to test our prototype. The instances have been adapted from our previous work [15]. The first instance is shown in Fig. 3 and represents the communication of a coffee machine, a PhD student, and a technician who repairs the coffee machine in case of an error. The second instance represents a simplified version of the SMTP protocol. The third instance represents the well-known dining philosophers problem with three philosophers. Of each instance we have created a correct and an erroneous version.

For all test cases, $\hat{s}$ (the starting state of the path) has been set to the global initial state, i.e., the global state where each state machine is in its initial state. For instances "coffee" and "mail", each possible combination of states for each state machine, and for the instance "philosophers" a meaningful selection of combination of states have been tested with the bound k set to $\{3,15,100\}$. Details on the outcomes of the test cases are presented on our project website. The results of all test cases are as expected. The bugs in the erroneous versions have been found. The approach performs well up to a bound of $k = 1000$. In Fig. 4 we exemplarily show the runtime measures for the coffee machine instance, which were evaluated on an Intel Core i5 CPU with 2.5 GHz running Linux. The red lines show a test case in which the goal was a partial global state containing a state of one out of three state machines, and the blue lines show a case where the goal was a complete global state, i.e., containing a state of each state machine. The dashed lines express the time needed to encode the problem and the solid lines give the total time, i.e., encoding time plus time spent by the SAT solver. As can be expected, the bottleneck for higher bounds is the task of solving the SAT instance.

6 Conclusion and Future Work

We presented a SAT-based approach to verify whether a combination of states in the state machine view of a software model can be reached through a path of bounded length from an

initial configuration. Using SAT allowed for a more direct translation than, e.g., using a model checker, and for a good integration within our existing framework which is implemented as plugin for the popular development environment Eclipse and therefore easily accessible to developers. We precisely defined the semantics of the state machine view of a software model. On this formal semantics we could build a formal problem definition. Based on this problem definition, the encoding to SAT turned out to be very intuitive, easy to understand and therefore easy to extend.

Immediately planned extensions of our encoding include the integration of a counter to handle the availability of multiple occurrences of a symbol representing an effect. Another important task is the optimization of the encoding in order to avoid unnecessary blowups and yield a more compact representation. On a longer time frame, the integration of transition guards and hierarchical state machines are planned. Before extending our approach, however, more extensive testing as well as a runtime comparison to alternative encodings will be necessary. To thoroughly test the tool, we will implement a solution verifier to automatically verify the solution returned by the SAT solver and apply our previously used approach of grammar-based fuzzing for MBE tools [14].

References

1. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A State/Event-based Model-Checking Approach for the Analysis of Abstract System Properties. Science of Computer Programming 76(2), 119–135 (Feb 2011)
2. Bézivin, J.: On the Unification Power of Models. SoSyM 4(2), 171–188 (2005)
3. Biere, A.: Bounded Model Checking. In: Handbook of Satisfiability, pp. 457–481. IOS Press (2009)
4. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. The MIT Press (1999)
5. Dubrovin, J., Junttila, T.A.: Symbolic Model Checking of Hierarchical UML State Machines. In: Int. Conf. on Application of Concurrency to System Design. pp. 108–117. IEEE (2008)
6. Knapp, A., Merz, S., Rauh, C.: Model Checking—Timed UML State Machines and Collaborations. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 395–416 (2002)
7. Le Berre, D., Parrain, A.: The Sat4j Library, Release 2.2, System Description. Journal on Satisfiability, Boolean Modeling and Computation 7, 59–64 (2010)
8. Lilius, J., Paltor, I.: vUML: A Tool for Verifying UML Models. In: ASE. pp. 255–258 (1999)
9. Niewiadomski, A., Penczek, W., Szreter, M.: Towards Checking Parametric Reachability for UML State Machines. In: Ershov Memorial Conference. LNCS, vol. 5947. Springer (2010)
10. Ober, I., Graf, S., Ober, I.: Validation of UML Models via a Mapping to Communicating Extended Timed Automata. In: Model Checking Software, LNCS, vol. 2989, pp. 127–145. Springer (2004)
11. Rintanen, J.: Planning and SAT. In: Handbook of Satisfiability, pp. 483–504. IOS Press (2009)
12. Selic, B.: What Will it Take? A View on Adoption of Model-based Methods in Practice. SoSyM 11, 513–526 (2012)
13. Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: Int. Conf. on Principles and Practice of Constraint Programming. LNCS, vol. 3709, pp. 827–831. Springer (2005)
14. Widl, M.: Test Case Generation by Grammar-based Fuzzing for Model-driven Engineering. In: Int. Haifa Verification Conference (2012)
15. Widl, M., Biere, A., Brosch, P., Egly, U., Heule, M., Kappel, G., Seidl, M., Tompits, H.: Guided Merging of Sequence Diagrams. In: Software Language Engineering. LNCS, vol. 7745, pp. 164–183. Springer (2013)