Notes on Masterthesis

Sally Chau

11.12. 2018, December

Contents

1 Hierarchical LTL Model Checking

- For a (recursive) input program $\mathcal{P}$ construct a respective Recursive State Machine $\mathcal{A}_{\mathcal{P}}$.
- Given a RSM $\mathcal{A}$ and an LTL formula φ check whether $\mathcal{A} \models \varphi$.
- If $\mathcal{A} \not\models \varphi$ return a counterexample and check whether it is spurious or not.

1.1 Translate input program $\mathcal{P}$ to a Recursive State Machine $\mathcal{A}_{\mathcal{P}}$.

(Jimple) Input program $\mathcal{P}$

- Input program $\mathcal{P}$ consists of a set of procedures (including an initial procedure (`main`)).
- Each procedure contains a set of statements.
- Statements can be

- assignments
- branching statements (`if`/ `else`, `skip`, `goto`)

  Why are these branching statements?

- invoke statements (procedure calls)
- return statements (procedure returns)

- Each statement is referenced by a numerical program location.
- Global and local variables ~> There are only distiguished by scoping

Recursive State Machine (RSM) $\mathcal{A} = \langle A_1, ..., A_k \rangle$ over a finite alphabet Σ, where each component state machine $A_i = (N_i \cup B_i, Y_i, En_i, Ex_i, \delta_i)$ consists of

- a set N_i of nodes and a (disjoint) set B_i of boxes,
- a labeling $Y_i : B_i \mapsto \{1, \cdots, k\}$ that assigns to every box an index of one of the component state machines A_i, $1 \leq i \leq k$,
- a set of entry nodes $En_i \subseteq N_i$,
- a set of exit nodes $Ex_i \subseteq N_i$,
- a transition relation δ_i, where transitions are of the form (u, σ, v), where
  - the source u is either a node of N_i, or a pair (b, x), where b is a box in B_i and x is an exit node in Ex_j for $j = Y_i(b)$
  - the label σ is in Σ
  - the destination v is either a node in N_i or a pair (b, e), where b is a box in B_i and e is an entry node in En_j for $j = Y_i(b)$

Algorithm (Sketch)

- Given a (recursive) input program $\mathcal{P}$, compute an according RSM $\mathcal{A}_\mathcal{P}$ that represents the control flow in $\mathcal{P}$. ~> Do you already have an idea how to do this in the intraprocedural case?
- Let k be the number of procedures in $\mathcal{P}$. Then $\mathcal{A}_\mathcal{P}$ has k component state machines, one for each procedure. Therefore, $\mathcal{A}_\mathcal{P} = \langle A_1, \cdots, A_k \rangle$.
- Each procedure i is represented by a respective component state machine $A_i = (N_i \cup B_i, Y_i, En_i, Ex_i, \delta_i)$, where
  - the set N_i of nodes composes of the program locations of procedure i. Each statement (except for procedure calls) is represented by a node $s \in N_i$. Index node s with respective program location of $\mathcal{P}$.

Is this sufficient? I would have expected that program states are relevant here.

Otherwise you only check the control flow graph.

I think this is clarified by your Algorithm on the next page :)

- For every distinct procedure j that is called by procedure i, we introduce a box $b \in B_i$ with $Y_i(b) = j$. If there are no procedure calls, $B_i = \emptyset$. Thus, procedure call statements are represented by boxes. seems legit.
- The labeling Y_i is as described above.
- En_i composes of the entry point of procedure i.
- Ex_i composes of the return (exit) point(s) of procedure i.
- δ_i is determined by the control flow of $\mathcal{P}$.
  * internal transitions: transitions given by control flow graph that stay within procedure i
  * call transition: transitions from a node s to an entry node of a box b
  * return transition: transitions from exit node of a box b to the last node visited in the component that called b

1.2 Model check RSM $\mathcal{A}$ for LTL formula φ

Model checking RSM $\mathcal{A}$ happens on-the-fly: as soon as a counterexample is found, the process will be aborted. In order to save unnecessary computations, the state space of the program $\mathcal{P}$ (computation of heap configurations of each state of $\mathcal{A}$) will be computed on-the-fly as well.

Input: RSM $\mathcal{A}$, LTL formula φ

Output: `true` if $\mathcal{A} \models \varphi$, `false` and counterexample otherwise

We may also abort with a timeout/mem-out

Algorithm (Sketch)

Ah, so you really combine state space generation. Nice.

Please keep in mind that I would also like to perform no model checking, i.e. construct the whole state space. This would be a good sanity check for your implementation.

- Let the component state machine A_1 of $\mathcal{A}$ refer to the `main`-method of the program $\mathcal{P}$. Let $en_1 \in En_1$ be the initial node/ entry node of A_1. Let en_1 be the current node.
- Get the current set of assertions Φ for formula φ.
- Get the respective program statement for the current node.
- Execute the program statement and compute the heap configuration HC for the current state,
  - according to the implemented methods in `InterproceduralAnalysis.run()` → `StateSpaceGenerator.generate()` → `stateSemanticsCommand.computeSuccessors(ProgramState)` which executes a single step of the abstract program semantics on the given program state and computes the set of successor states.

This should include the state labeling. I just mention it for completeness.

If you think the current MC implementation is useful, feel free to adapt it. However, it's fine (and maybe easier) to implement it from scratch.

- After the computation of the heap configuration of the **current** state, model check its heap configuration HC for the current formula/ set of assertions using the tableau method. ✓
  - need to change the tableau method implementation such that single states and a (sub)formula or a set of assertions can be taken as an input to the method call
  - should return model checking result, and next set of assertions
- If the tableau method returns a final value (`true` or `false`), the process is done. ✓
- Otherwise, the next state according to the tableau method execution needs to be analysed. The next state to be checked is determined by the control flow of the program $\mathcal{P}$/ the transition relation of the RSM $\mathcal{A}$. ✓
- If the next state is not a call or return state, the above procedure is repeated.
- If the next state enters procedure i, start a new state space for procedure i in order to generate contracts. Therefore, input last assertions of tableau method into the new model checking layer. Continue model checking procedure for internal nodes of procedure i. ✓
- At the end (exit) of procedure i, return to calling position with updated assertions and model checking results. ✓
- Compute contracts (pre-/ and post-conditions) after the whole state space of procedure i has been computed. ✓
- Store model checking results (assertions that hold/ do not hold before/after model checking) together with contracts (as a kind of summary) in a table, so that the contracts can be reused in case procedure i is called with the same (sub)set of assertions at another program location. ✓

I am not 100% sure what the best approach here is. Ideally, I would like to throw away the state space, but probably that's not possible?

→ this is ok for now

- If the contract for procedure i has already been computed, but the assertion or formula φ has not been model checked and stored yet, start a new model checking subroutine for i and the stored state space. The state space of i does not need to be computed again. Add the model checking result to the table of model checking results of i.
- Continue this process until a counterexample has been found or the tableau method terminates. ✓

Is this meant as an alternative to the tableaux method?
I am not sure how you want to encode state space gen in a PDA.

1.2.1 Model checking RSM $\mathcal{A}$ as a PDA

The procedure described above can also be described by a run of a pushdown automaton (PDA) $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$, where

- Q is a finite set of states
- Σ is a finite input alphabet
- Γ is a finite stack alphabet
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the transition relation
- $q_0 \in Q$ is the start state
- $Z \in \Gamma$ is the initial stack symbol
- $F \subseteq Q$ is the set of accepting states

For an input program $\mathcal{P}$ a respective PDA $\mathcal{M}_\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ composes of

I do not understand how this is supposed to work. Somehow I'm missing the connection to state space generation and tableaux-based model-checking.

Moreover, what is the benefit of formalizing your algorithm as a PDA?

Personally, I think the algorithm is easier to understand. Maybe I am missing something though. Let's discuss this on Thursday.

- a finite set Q of states, where for each procedure i in the program $\mathcal{P}$, there is a state $q_i \in Q$,
- the finite input alphabet Σ, is described later,
- the finite stack alphabet Γ coincides with the set of indices of the procedures of $\mathcal{P}$,
- the transition relation $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ with two kinds of transitions
  - call transitions, and
  - return transitions,
- the start state $q_0 \in Q$ is the `main` procedure of $\mathcal{P}$,
- the initial stack symbol $Z \in \Gamma$,
- $F \subseteq Q$ is the set of accepting states, which is either the return statement of the `main` procedure or defined via termination of the tableau method.

The stack alphabet consists of objects of the form $\Phi^C_{q_i,q_j} \in \Sigma \cup \{\epsilon\}$ and $\Phi^R_{q_j,q_i} \in \Sigma \cup \{\epsilon\}$. These are the sets of assertions to be checked in q_j and q_i, respectively. These objects summarize the model checking information passed between the procedures.

To-Do

- Pre- and post-conditions of the procedures need to be included in the summary objects, as they are required for a possible table look up, and also for continuing the tableau method/ state space generation in the next procedure/ state.

A separate table containing information on pre- and post-conditions of each procedure keeps track of previously seen model checking processes (combinations of pre-/ post-conditions and set of assertions). If the table look up finds an entry, the model checking process can be replaced by the known result. The table look up happens only if pre- and post-conditions of a procedure have been computed, therefore if the stace space has been computed.

<u>The transition relations</u>:
For every call transition, there is also a return transition (for non-faulty and finite programs $\mathcal{P}$).

- <u>Call transitions</u>:

  For every procedure j that is called by procedure i, there is a call transition in $\mathcal{M}_\mathcal{P}$. Thus, for nodes $q_i, q_j \in Q$, an input $\Phi^C_{q_i,q_j} \in \Sigma \cup \{\epsilon\}$, topmost stack symbol $A \in \Gamma$, a string $\alpha \in \Gamma^*$, we have $(q_i, \Phi^C_{q_i,q_j}, A, q_j, \alpha) \in \delta$ if

  - procedure i calls procedure j
  - the index of the calling procedure i is pushed to the top of the stack

- <u>Return transitions</u>:

  For every procedure j that is called by procedure i, there is a return transition in $\mathcal{M}_\mathcal{P}$. Thus, for nodes $q_i, q_j \in Q$, an input $\Phi^C_{q_i,q_j} \in \Sigma \cup \{\epsilon\}$, topmost stack symbol $A \in \Gamma$, a string $\alpha \in \Gamma^*$, we have $(q_j, \Phi^C_{q_j,q_i}, A, q_i, \alpha) \in \delta$ if

  - procedure j returns to procedure i
  - the index of the calling procedure i is popped from the top of the stack

The model checking itself happens within the nodes $q \in Q$. The PDA $\mathcal{M}_{\mathcal{P}}$ thus abstracts from the concrete model checking process and rather represents the flow of the assertions and model checking results passed between the procedures involved.

1.2.2 Table of model checking results

In order to avoid multiple computation of the same model checking results for the same procedures, we introduce a table that stores the previously computed result together with pairs of pre- and post-conditions of a procedure in a table. Prior to model checking a procedure, we check

- Are pre-/ post-conditions available for the current procedure?
- Is there a matching set of pre-/ post-conditions to the current state?
- Has the set of assertions been model checked before for the current procedure?

If all above qeustions can be answered with 'yes', a table look up can check for the model checking results that must have been previously computed. Thus, the model checking process does not need to be repeated for the procedure and the set of assertions.

To-Do

- How can we structure the table such that a look up is as fast as possible?
- Is there a way to reuse or imply model checking results from set of assertions that do not coincide with the current set of assertions by 100%? e.g. expansion rules

1.3 Analyse counterexample for spuriosity