Model checking for weak memory models

Ori Lahav Viktor Vafeiadis

30 August 2017

What is model checking?

Software model checking

Given a property Φ and a program C, check whether all (consistent) executions of C satisfy the property Φ.

The property, Φ:

- Traditionally, given in a temporal logic (e.g., LTL)
- Here, we consider only safety properties.
- These can be expressed as reachability of error states.

The program, C, and its semantics:

- A concurrent program with WMC semantics.
- Axiomatic WMM ensuring $(\mathtt{po} \cup \mathtt{rf})^+$ is acyclic.

Traditional MC approaches

Following an operational semantics...

- Explicit state MC
- Stateless MC (with POR)

Following an axiomatic semantics...

- Encode the problem in SAT/SMT

Graph-based *stateful* model checking

Goal: Enumerate all consistent execution graphs of P without

- generating the same graph multiple times; and
- generating any inconsistent graphs.

Naive approach:

- Record the set V of all graphs already generated.
- Initially, V contains only the empty execution graph.
- At each point, pick a graph $G \in V$ and an event a such that $G' = \mathit{Add}(G, a)$ is a consistent execution of P, and add G' to V.
- Repeat the previous until no new graphs can be added.

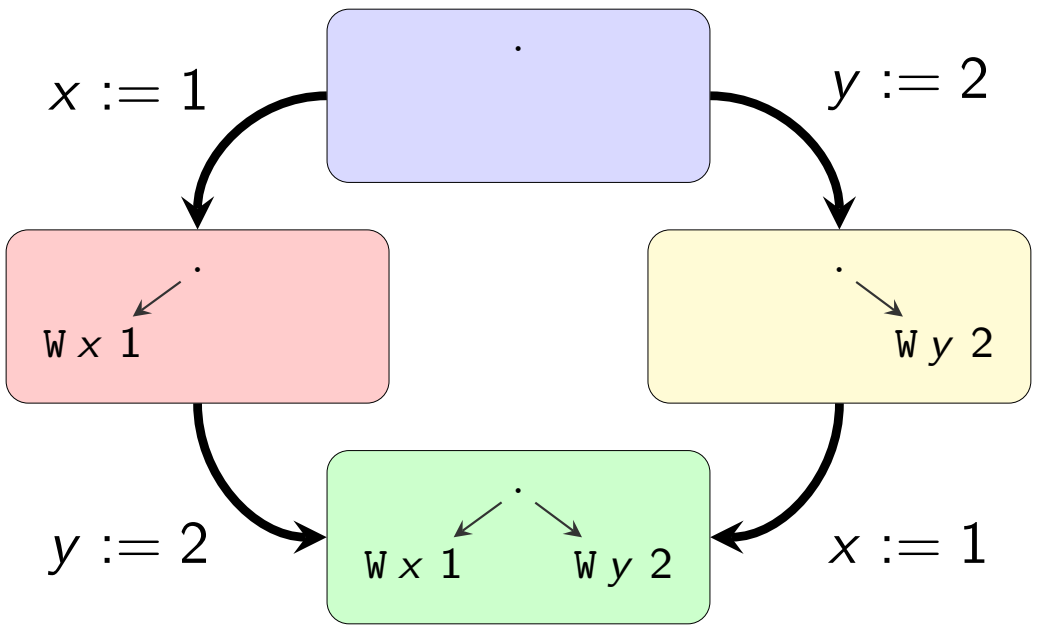
The naive algorithm is too naive

Observation

The order in which events are added is mostly irrelevant.

Example

$$x := 1 \parallel y := 2$$



Improving the naive algorithm

Fix an order in which events are added.

- *e.g.*, in increasing thread ID order.

When adding a read r:

- Consider all possible writes that r could read from.

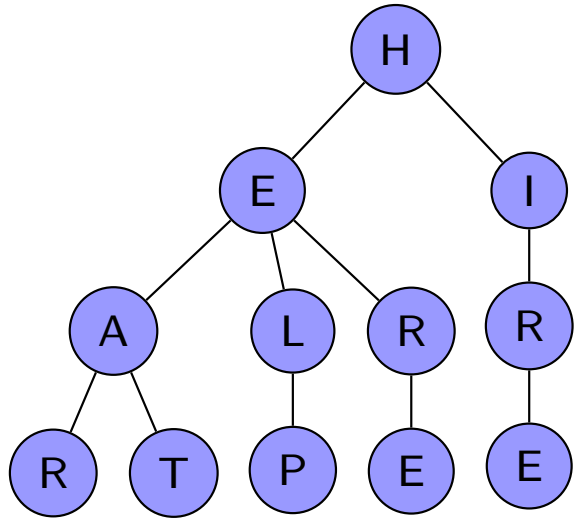
When adding a write w:

- Consider all possible placements of w in $\texttt{mo}$ *and also* whether any existing reads can read from w.
- For any subset of such reads, "revisit" them:
  - Change their $\texttt{rf}$-incoming edges to read from w.
  - Delete any events $(\texttt{po} \cup \texttt{rf})^+$ after them.

Representing sets of visited graphs

Use a trie

- A standard data structure for storing sets of strings.
- Particularly useful if strings often have a common prefix.



{HEAR, HEAT, HELP, HERE, HIRE}

Mapping executions to strings

- Visit events in some total order extending $(\texttt{po} \cup \texttt{rf})^+$.
  (preferably matching the event addition order)
- For reads, record where they read from.
- For writes, record their position in `mo`.

Graph-based *stateless* model checking

Goal: Enumerate all consistent execution graphs of P without

- generating the same graph multiple times;
- generating any inconsistent graphs; and
- recording the set of graphs already generated.

Key challenge:

- How to avoid repetition?

How can repetition arise?

- Revisiting the 'same' read in multiple subexecutions.
- The same event but reading from different writes.

$$x := 1 \;\|\; a := x \;\|\; x := 2$$

- The reads differ only in their $(\mathtt{po} \cup \mathtt{rf})^+$ suffix.

$$y := 1 \;\Big\|\; \begin{array}{l} a := x; \\ b := y \end{array} \;\Big\|\; x := 2$$

- We get the same graph after revisiting them.

Revisit sets

Record the set T of revisitable reads:

- *i.e.*, reads that *may* be revisited when extending G.

When adding a read r:

- Make r revisitable in only one of the subexecutions; more specifically, in one reading from a $(\mathtt{po} \cup \mathtt{rf})^+$-prior write.
- In all other cases, remove its $(\mathtt{po} \cup \mathtt{rf})^+$-prior reads from the revisit set, T.

When adding a write w and revisiting a set R of reads:

- Require that $R \subseteq T$ and $[R]; (\mathtt{po} \cup \mathtt{rf})^+; [R \cup \{w\}] = \emptyset$.
- Remove R and all their $(\mathtt{po} \cup \mathtt{rf})^+$-predecessors from T.

Properties

- Reads reachable from a revisitable read are revisitable:

$$\mathit{codom}([T]; (\mathtt{po} \cup \mathtt{rf})^{+}; [\mathtt{R}]) \subseteq T$$

- Every non-revisitable read in G is revisitable in some other visited execution.