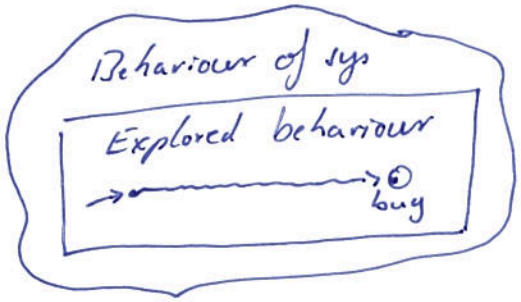
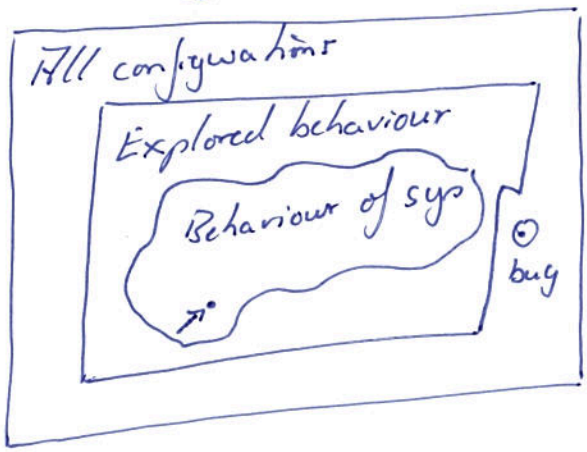
9. Reachability within a bounded number of rounds

(Abdulla, Bouajjani, Parlato CAV 2011)

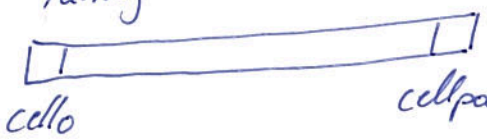
- Often, algorithmic verification problems are undecidable or have a high complexity.
  Therefore, practical verification methods focus on one of the following:
  - ↳ Under-approximate system analysis = bug hunting.
  - ↳ Over-approximate system analysis = prove sys correct





- In these approximations, reachability can be solved with "low" complexity.

Low still means $\geq$ PSPACE as long as we consider variables:



= Boolean while-program with variables $x_0, \ldots, x_{poly(n)}$.

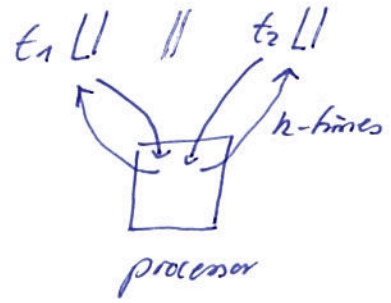
(Dexter Kozen 1977)

Goal:
- One can show that TSO-reachability has non-primitive recursive (time and space) complexity.
- Develop an under-approximation that works in PSPACE.

Inspiration:
- Reversal-bounded counter machines (Ibarra 1978, see also Applied Automata Theory notes).
- Context-bounded analysis of recursive and parallel programs (Qadeer, Rehof 2005)

$t_1 LI \parallel t_2 LI$ = Turing machine.

Context-bounding:



Practice:
Most bugs show up within few context switches.

Idea:
- Assume each thread can be active in k-rounds (will be defined in a moment).
- With this, given a program P construct a new program P'
  - of size linear in the size of P and
  - of the same type (same set of instructions)

such that

$$Reach_{TSO}^{k\text{-rounds}}(P) = Reach_{SC}(P').$$

This equality means that for every control state pc we have

pc is reachable within k-rounds when P is executed under TSO

iff

pc is reachable when P' is executed under SC.

- That P' is executed under SC means the analysis does not need to simulate store buffers.
  $\Rightarrow$ Standard verification tools apply to P'.

Definition (Rounds):
Consider a program $P = \{t_1, \dots, t_n\}$.
- We augment the TSO-transition relation with thread identifiers:

$$\rightarrow_{TSO} \subseteq CF \times TID \times CF.$$

A transition

$cf \xrightarrow{t}_{TSO} cf'$ is defined as before but additionally indicates that
- thread t performed the operation or
- the buffer of t has been used to update the memory.

- Each computation

$$\rho = cf_0 \xrightarrow{t_{i_0}} cf_1 \xrightarrow{t_{i_1}} cf_2 \xrightarrow{t_{i_2}} \dots \xrightarrow{t_{i_{n-1}}} cf_n$$

can be represented as a sequence of <u>rounds</u>, computation segments where the same thread is active (transitions have the same thread identifier):

$$\rho = \rho_0 \dots \rho_k$$

with

$$\rho_0 = cf_0 \xrightarrow{t_{j_0}} cf_1 \xrightarrow{t_{j_0}} cf_2 \dots \xrightarrow{t_{j_0}} cf_{n_0}$$

$$\rho_1 = cf_{n_0} \xrightarrow{t_{j_1}} cf_{n_0+1} \xrightarrow{t_{j_1}} cf_{n_0+2} \dots \xrightarrow{t_{j_1}} cf_{n_1}$$

$$\vdots$$

$$\rho_k = cf_{n_{k-1}} \xrightarrow{t_{j_k}} cf_{n_{k-1}+1} \xrightarrow{t_{j_k}} cf_{n_{k-1}+2} \dots \xrightarrow{t_{j_k}} cf_{n_k},$$

where $t_{j_m} \neq t_{j_n}$ for $m \neq n \in \{0, \dots, k\}$.

- A <u>k-round computation</u> is a computation from $\left(\xrightarrow{t_0}{}^{*} \dots \xrightarrow{t_n}{}^{*}\right)^k$, so every thread has at most k rounds.

We use

$$Reach_{TSO}^{k\text{-rounds}}(P) := \{ pc \in LOC^{TID} \mid \exists cf = (pc, val, buf) \in CF : cf \text{ is reachable by a } k\text{-round computation} \}.$$

We study the following

<u>k-round reachability problem:</u>

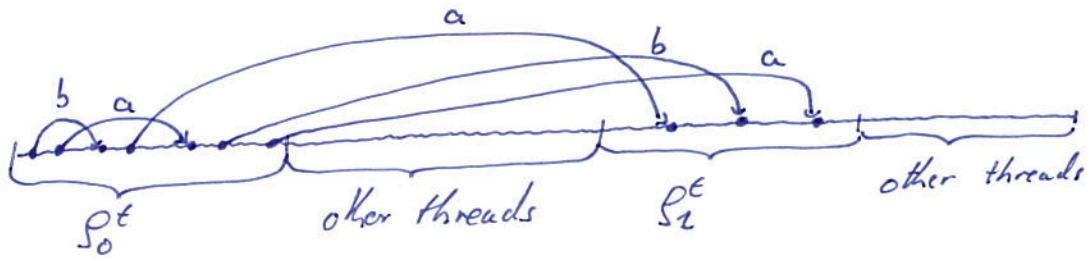
<u>Given</u>: Program P, control state pc, bound $k \in \mathbb{N}$.

<u>Question</u>: $pc \in Reach_{TSO}^{k\text{-rounds}}(P)$?

Remark:

- This is reachability / emptiness modulo a bounded language
  $B = w_1^* \dots w_n^*$.
- Esparza, Ganty 2011:
  $L(\Pi_1) \cap L(\Pi_2) \cap L(B) = \emptyset$
  is NP-complete for $L(\Pi_1), L(\Pi_2)$ context-free (as opposed to undecidable in the general case).
  Relies on Parikh-images.

9.1 2-round case



↳ Focus on the behaviour of thread t,
project configurations to what is visible to t.

↳ Inspect the interaction with other threads
at round-switch points:

$$\rho_0^t = (pc(t), rval(t), mval, buf(t)) \xrightarrow{t}{}^*_{TSO} (\underline{pc'(t)}, \underline{rval'(t)}, mval', \underline{buf'(t)})$$

$$\rho_1^t = (\underline{pc'(t)}, \underline{rval'(t)}, \tilde{mval}, \underline{buf'(t)}) \xrightarrow{t}{}^*_{TSO} (pc''(t), rval''(t), mval'', buf''(t))$$

underlining $=$ means these components coincide at the end of round 0 and at the beginning of round 1.

Goal:
- Use a finite number of addresses to encode unbounded store buffers.
- Make use of two observations.

Observation 1:

↳ To execute a load, we need to know
- whether a store to the address is still buffered
- if so, need the value of the last such store
- otherwise, need the value from memory.

↳ Since in a round only t is active, only the buffer of t will change the memory. This means, to execute a load we need

last written value per address $\oplus$ memory content at the beginning of the round.

↳ Introduce a function

$view \in DOM^{DOM}$

so that $view[a]$ gives the value to load from address a.

Observation 2:

↳ The order in which store operations of t on different addresses are consumed by the memory is not important. As t is running in isolation, no partner thread could observe a difference.

↳ Moreover, only the last store per address is relevant.

↳ Introduce, for each round j, two functions

$mask_j \in \mathbb{B}^{DOM}$

$queue_j \in DOM^{DOM}$:

$mask_j[a] = true$ iff there is a store operation on address a in the buffer of t that is used to update the memory at round j.

$queue_j[a] = v$ is the value of the last such store operation.

Together this means we replace thread-local configurations in P

$(pc(t), rval(t), mval, buf(t))$

by

$(pc(t), rval(t), view, mval, mask_0, mask_1, queue_0, queue_1)$ in P'.

How to mimic the behaviour of P by P':

- We construct for a 2-round computation of t in P a 2-round computation of t in P' such that the above invariants on the relationship between the configurations at switch-points hold.

Special case: All store operations issued in round 0 go to memory in round 0:

↳ In P', thread t immediately writes the stores to the shared memory, without buffering.

↳ Thread t will not notice a difference (for single-threaded programs, $[\![0]\!] = s()$).

↳ The other threads are not active.

General case: Some store operations go to memory in round 0 some go to memory in round 1:

In P', the task is to

- ensure that the shared memory contains $mval'$
- pass (a summary of) $buf'(t)$ to the second round.

Behaviour of t in P:

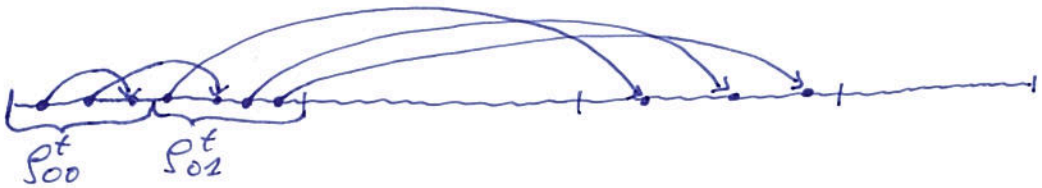
↳ We can understand ρ_0^t as concatenation of two computations:

$$\rho_0^t = \rho_{00}^t \cdot \rho_{01}^t$$

where

ρ_{00}^t = All stores issued in ρ_{00}^t go to memory in round 0,

ρ_{01}^t = All stores issued in ρ_{01}^t go to memory in round 1.



↳ By FIFO, the stores for round 0 are issued before the stores for round 1.

Then:

$mval'$ = Result of executing stores from S_{00}^t (1)

$buf'(t)$ = All stores produced in S_{01}^t (2).

Simulation of round 0 in P':

↳ During the simulation of round 0 by P', guess the end of S_{00}^t non-deterministically.

↳ Stores in S_{00}^t are written to memory immediately. This ensures the content of the shared memory in P' is precisely $mval'$, due to (1) above.

↳ During the simulation of S_{01}^t, P' performs two operations:

- maintain the view of t by updating the view-addresses.
- keep in $mask_1$ and $queue_1$ the information about the last values stored to each address in S_{01}^t.

This ensures that, at the end of round 0, $mask_1$ and $queue_1$ represent a summary of $buf'(t)$, due to (2).

Simulation of round 1 in P':

↳ Start with a new state $\widetilde{mval}$ of the shared memory.

↳ Update the shared memory using $mask_1$ and $queue_1$, immediately at the beginning of the computation.

↳ Rebuild view as follows:

- Starting from view at the end of round 0, change the valuation of all addresses a for which no stores were pending in the buffer ($mask_1[a] = false$):

$$view[a] := \widetilde{mval}[a].$$

Theorem (Atig, Bouajjani, Parlato 2011)

k-round TSO-reachability is PSPACE-complete, and reducible to ordinary SC-reachability.

-7- Note: In the actual reduction, one makes rounds atomic.