Lecture notes:
Verification with Small and Short Worlds (S^{2}W)

Presenter: Jaesung Park

Feb 28, 2017

1 Overview

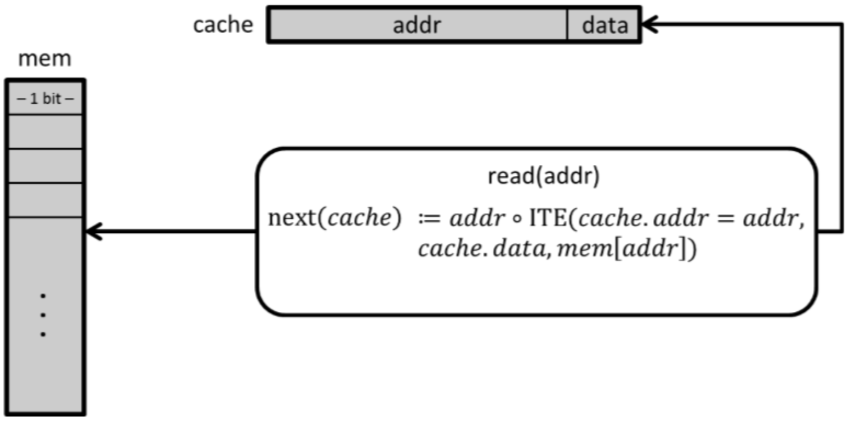
1. Problem statement
   (a) System model for large arrays and data structures
   (b) Safety property to verify
2. Methodology of S^{2}W
   (a) Induction, if we are lucky
   (b) Small World, restricting the search space
   (c) Short World, bounding the number of steps
3. Evaluation
   (a) Table Look-ahead Buffer (TLB) of Bochs x86 emulator
   (b) Set-associative cache and Content Adressable Memory (CAM)
   (c) Shadow paging

2 Problem Statement

2.1 System Model

$S = (\mathcal{I}, \mathcal{O}, \mathcal{V}, \mathit{Init}, \mathcal{A})$: system model

$\mathcal{I}$: a finite set of input variables
$\mathcal{O}$: a finite set of input variables
$\mathcal{V}$: a finite set of input variables
Init : a finite set of input variables
$\mathcal{A}$: a finite set of input variables


cache
addr
data
mem
– 1 bit –
read(addr)
next(*cache*) ≔ *addr* ∘ ITE(*cache*. *addr* = *addr*, *cache*. *data*, *mem*[*addr*])

$\mathcal{I} = \{addr\}$
$\mathcal{O} = \{out\}$
$\mathcal{V} = \{mem, cache\}$
$\mathit{Init} = (mem_0, cahe_0)$
$\mathcal{A}$: updating *cache* and return the value in *mem*

2.2 Problem Definition

The goal is to verify **G**Φ, the temporal safety property we are interested in.
G : the temporal operator ”always”
Φ : the temporal operator ”always”

In the example,

$$\Phi_2 = \forall x. (addr = x) \rightarrow ((cache.addr = addr \wedge cache.addr \neq 0) \rightarrow cache.data = mem[addr]).$$

3 Methodology of S^{2}W

3.1 Induction

Using one-step induction,

$$\begin{gathered} Init(\mathcal{V}) \rightarrow \Phi(\mathcal{V}) \\ \Phi(\mathcal{V}) \land \mathcal{R}(\mathcal{V}, \mathcal{V}') \rightarrow \Phi(\mathcal{V}'). \end{gathered}$$

If both checks pass, the verification is complete.
Otherwise, S^{2}W continues.

In the example, induction fails when

$$mem[i] := a, \quad mem[j] := b, \quad cache.addr := i, \quad cache.data := z,$$

and $read(i)$ is given.

3.2 Small World

Instantiate the free variables in Φ with symbols.
It defines a *dependence set* ($\mathcal{U}$).

In the example,

$$\Phi_2 = \forall x. (addr = x) \rightarrow ((cache.addr = addr \land cache.addr \neq 0) \rightarrow cache.data = mem[addr]).$$

In the form of $\forall x. \Phi(x)$, replacing x with a fixed symbolic value a, we verify $\Phi(a)$,
and $\mathcal{U} = \{mem[a], cache\}$.

In result, the search space is restricted.

This way, the state space is reduced from $2^3 4$ states to the following 16 states:

$$\begin{aligned} &\{cache.addr = a, cache.addr \neq a, cache.addr = 0, cache.addr \neq 0\} \\ &\quad \times \{cache.data = 0, cache.data = 1\} \times \{mem[a] = 0, mem[a] = 1\}. \end{aligned}$$

3.3 Short World

Diameter D of the abstract model, the smallest integer where for every reachable states, there is a sequence of inputs of length $\leq D$.

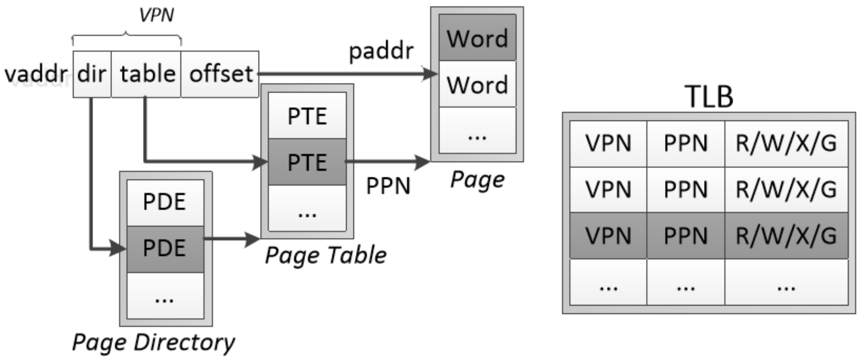
Find the upper bound of D denoted by k, then run Bounded Model Checking (BMC) with the maximum number of steps k.

In result, the search space is restricted.

In the example, at most 2 steps are used to reach all reachable states.

4 Evaluation

4.1 Bochs' TLB

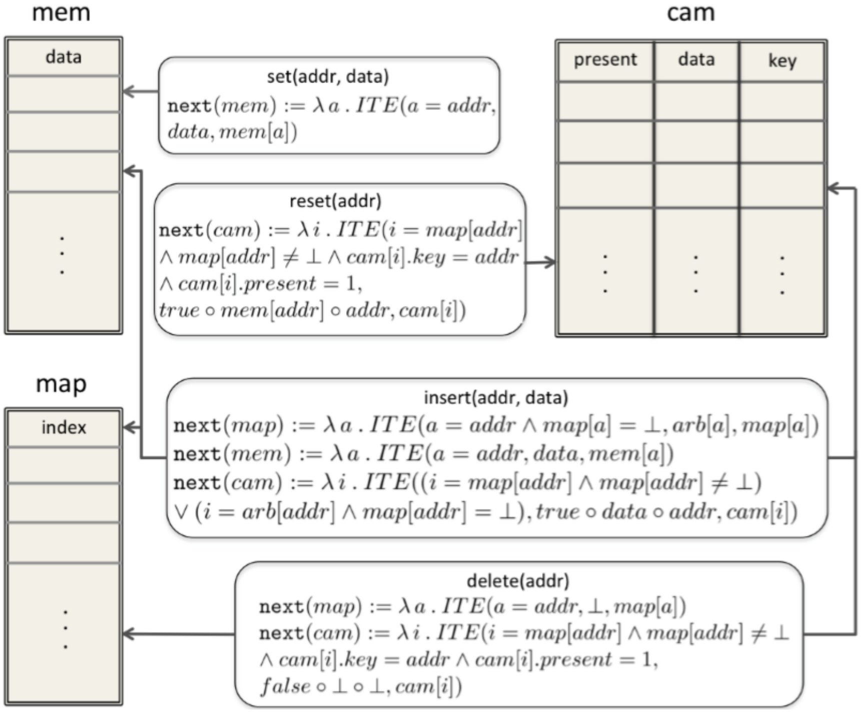


Safety property to verify:

”Does TLB indicate the correct a physical address with respect to a virtual address, if TLB has the entry?”

1. *Induction*: fails.
2. *Small World*: looking only at a specific location of memory and page entry.
3. *Short World*: bounded by 9 steps.

BMC took 25 45 minutes in S^{2}W.


mem
data
cam
present
data
key
map
index
set(addr, data)
$\texttt{next}(mem) := \lambda\, a\,.\, ITE(a = addr,$
$data, mem[a])$
reset(addr)
$\texttt{next}(cam) := \lambda\, i\,.\, ITE(i = map[addr]$
$\wedge\, map[addr] \neq \bot \wedge cam[i].key = addr$
$\wedge\, cam[i].present = 1,$
$true \circ mem[addr] \circ addr, cam[i])$
insert(addr, data)
$\texttt{next}(map) := \lambda\, a\,.\, ITE(a = addr \wedge map[a] = \bot, arb[a], map[a])$
$\texttt{next}(mem) := \lambda\, a\,.\, ITE(a = addr, data, mem[a])$
$\texttt{next}(cam) := \lambda\, i\,.\, ITE((i = map[addr] \wedge map[addr] \neq \bot)$
$\vee\, (i = arb[addr] \wedge map[addr] = \bot), true \circ data \circ addr, cam[i])$
delete(addr)
$\texttt{next}(map) := \lambda\, a\,.\, ITE(a = addr, \bot, map[a])$
$\texttt{next}(cam) := \lambda\, i\,.\, ITE(i = map[addr] \wedge map[addr] \neq \bot$
$\wedge\, cam[i].key = addr \wedge cam[i].present = 1,$
$false \circ \bot \circ \bot, cam[i])$

4.2 Content Addressable Memory

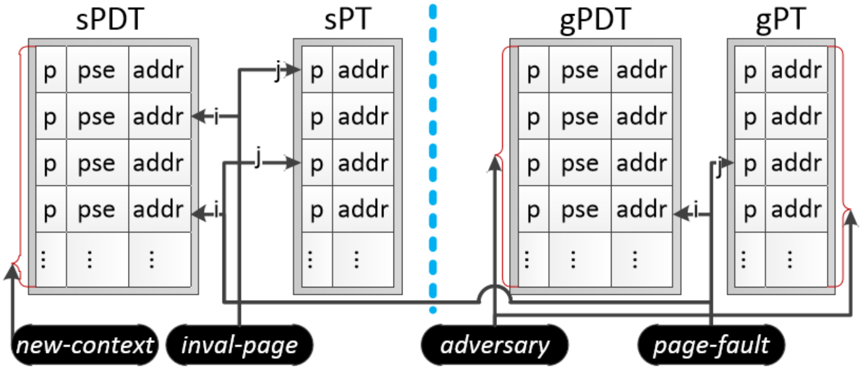
Safety property to verify:

> ”Do the CAM and memory have the same data for all keys present in CAM?”

1. *Induction*: fails.
2. *Small World*: looking only at *mem*[*a*], *map*[*a*] and *cam*[*map*[*a*]].
3. *Short World*: bounded by 5 steps.

BMC took 5 15 seconds in S^{2}W.

4.3 Shadow Paging



Safety property to verify:

(1) ”Is the address under fixed limit, if page size extension bit is on?”
(2) ”Is the address under fixed limit, if page size extension bit is off?”

1. *Induction*: (1) success, (2) fails.
2. *Small World*: (2) looking only at sPDT[a_i] and SPT[a_j].
3. *Short World*: (2) bounded by 4 steps.

BMC took < 1 minute in S^{2}W for verifying (2).