IAS Seminar

Model Checking Embedded Systems

Lucas Cordeiro
lucascordeiro@ufam.edu.br

Career Summary

- BSc in Electrical Engineering, MSc/PhD in Computer Science
  - algorithms, software engineering, formal verification, and embedded systems
- 39 reviewed publications, including 6 journal papers and 33 workshop/conference contributions
  - distinguished paper awards at SAC'08 and ICSE'11, and two bronze medals at TACAS’12 and TACAS’13
- developer of XMPM, STB225, and ESBMC tools
- research collaborations with Southampton and Stellenbosh
- research funding from Samsung, Nokia, and Royal Society
- research team leader (one PhD, four MSc, and two BSc students)
  - acting as course leader of Electrical Engineering

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics
  - medical systems

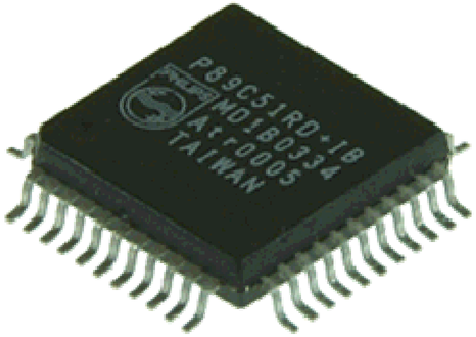
Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics
  - medical systems

- provide a number of **distinctive characteristics**
  - usually implemented in DSP, FPGA and µC (mass production)

Embedded systems are ubiquitous but their verification becomes more difficult.

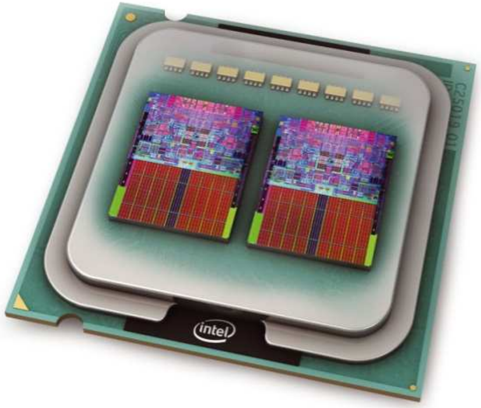
- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics
  - medical systems



- provide a number of **distinctive characteristics**
  - usually implemented in DSP, FPGA and μC (mass production)
  - functionality determined by software in read-only memory

Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics
  - medical systems



- provide a number of **distinctive characteristics**
  - usually implemented in DSP, FPGA and μC (mass production)
  - functionality determined by software in read-only memory
  - multi-core processors with scalable shared memory

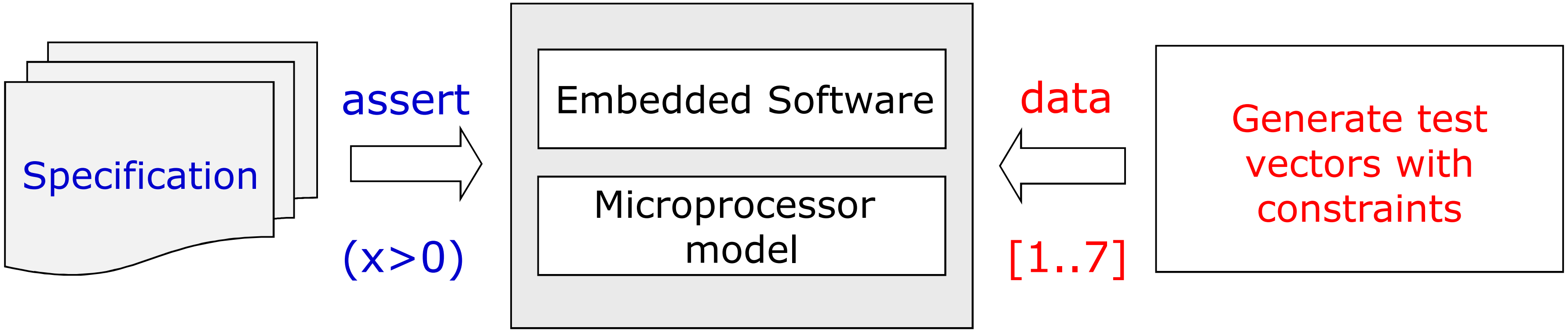
Embedded systems are ubiquitous but their verification becomes more difficult.

- embedded system is part of a well-specified larger system (**intelligent product**)
  - automobiles
  - airplanes
  - communication systems
  - consumer electronics
  - medical systems

- provide a number of **distinctive characteristics**
  - usually implemented in DSP, FPGA and µC (mass production)
  - functionality determined by software in read-only memory
  - multi-core processors with scalable shared memory
  - limited amount of energy

Verification Challenges

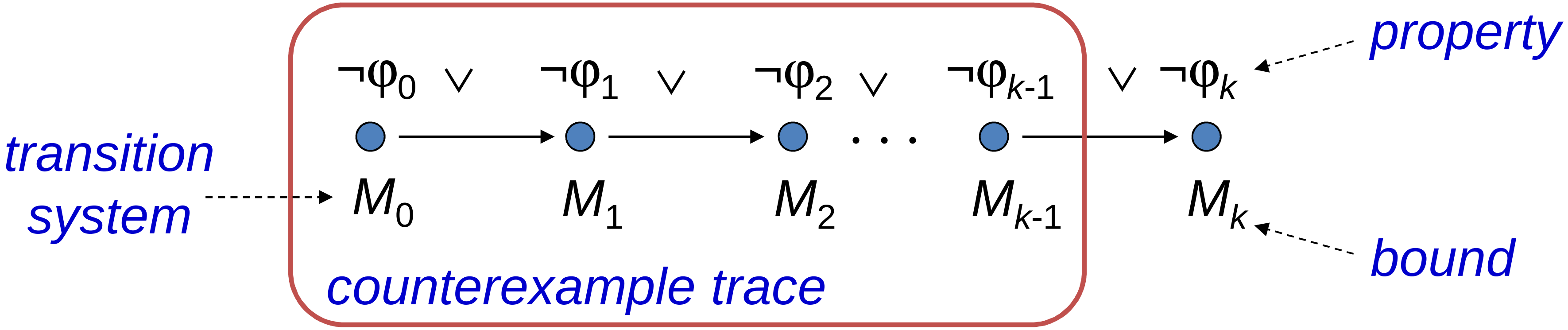
- **verification methodologies** for embedded systems



- verification of embedded systems raises **additional challenges**
  - handle concurrent software
  - meet time and energy constraints
  - legacy designs (usually written in low-level languages)
- improve **coverage** and reduce **verification time**

Bounded Model Checking (BMC)

Basic Idea: check negation of given property up to given depth



$\neg\varphi_0 \vee \neg\varphi_1 \vee \neg\varphi_2 \vee \neg\varphi_{k-1} \vee \neg\varphi_k$
M_0
M_1
M_2
M_{k-1}
M_k
. . .
property
transition system
bound
counterexample trace

- transition system *M* unrolled *k* times
  - for programs: loops, arrays, …
- translated into verification condition ψ such that

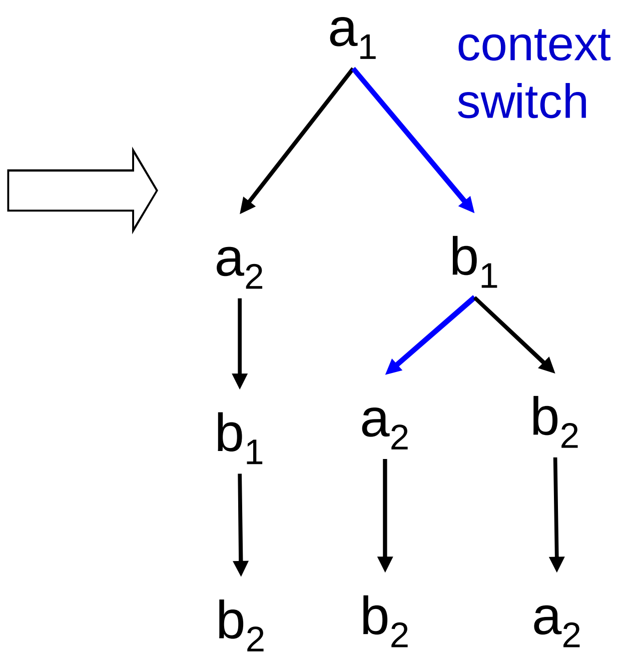
ψ satisfiable iff φ has counterexample of max. depth *k*

- has been applied successfully to verify (embedded) software

BMC of Multi-threaded Software

- concurrency bugs are tricky to **reproduce** because they usually occur under specific thread interleavings
  - most common errors: 67% related to atomicity and order violations, 30% related to deadlock [Lu et al.'08]

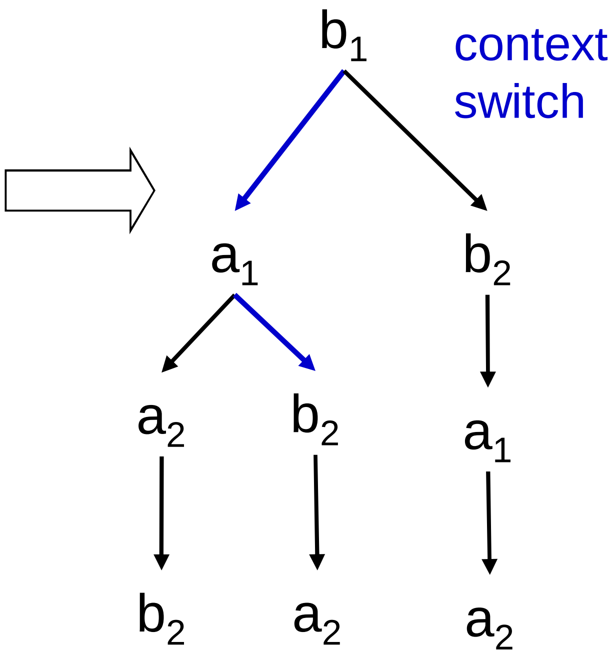
Thread T_1	Thread T_2
a_1	b_1
a_2	b_2



BMC of Multi-threaded Software

- concurrency bugs are tricky to **reproduce** because they usually occur under specific thread interleavings
  - most common errors: 67% related to atomicity and order violations, 30% related to deadlock [Lu et al.'08]

Thread T_1	Thread T_2
a_1	b_1
a_2	b_2

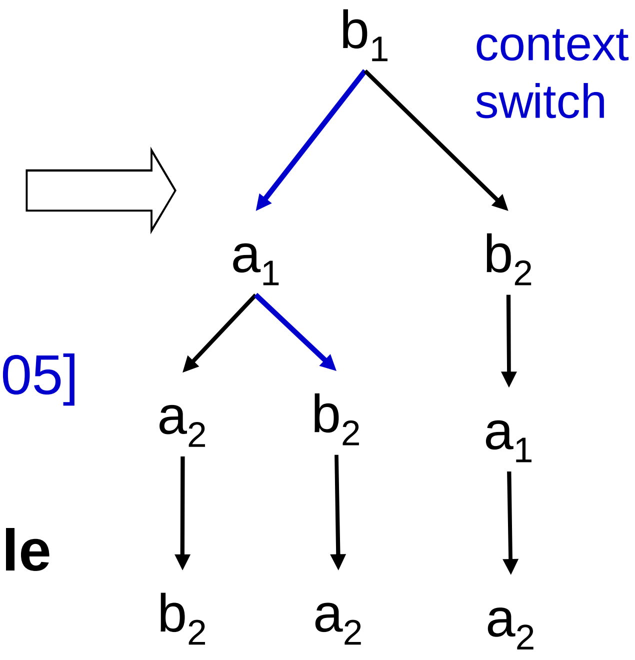


BMC of Multi-threaded Software

- concurrency bugs are tricky to **reproduce** because they usually occur under specific thread interleavings
  - most common errors: 67% related to atomicity and order violations, 30% related to deadlock [Lu et al.'08]

number of executions: $O(n^s)$

Thread T_1	Thread T_2
a_1	b_1
a_2	b_2

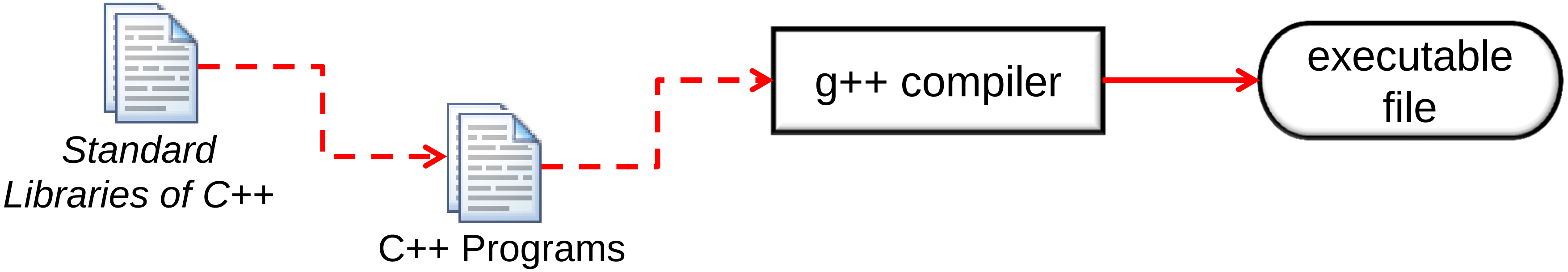


concurrency bugs are shallow [Qadeer&Rehof'05]

- hypothesis:
  - SAT/SMT solvers produce **unsatisfiable cores** that allow removing possible undesired models of the system

BMC of SystemC/C++

- SystemC consists of a set of C++ classes that simulates **concurrent processes** using plain C++
  - **object-oriented design** and **template classes**



the standard C++ library complicates the VCs unnecessarily

BMC of SystemC/C++

- SystemC consists of a set of C++ classes that simulates conc
  - – ol

```
template <class _Tp, class _Alloc>void vector<_Tp,
_Alloc>::_M_fill_insert(iterator __position, size_type __n,
const _Tp& __x){
 if (__n != 0) {
  if (size_type(_M_end_of_storage - _M_finish) >= __n) {
   _Tp __x_copy = __x;
   const size_type __elems_after = _M_finish - __position;
   iterator __old_finish = _M_finish;
   if (__elems_after > __n) {
    uninitialized_copy(_M_finish - __n, _M_finish, _M_finish);
    _M_finish += __n;
    copy_backward(__position, __old_finish - __n, __old_finish);
    fill(__position, __position + __n, __x_copy);
…
```

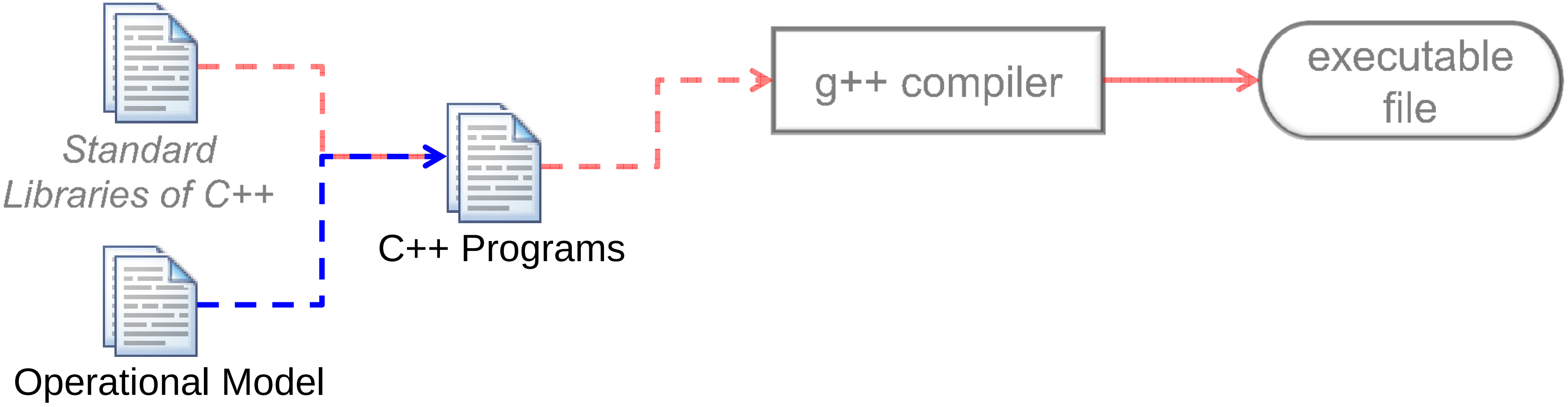
Standa
Libraries c

cutable
file

the stan

BMC of SystemC/C++

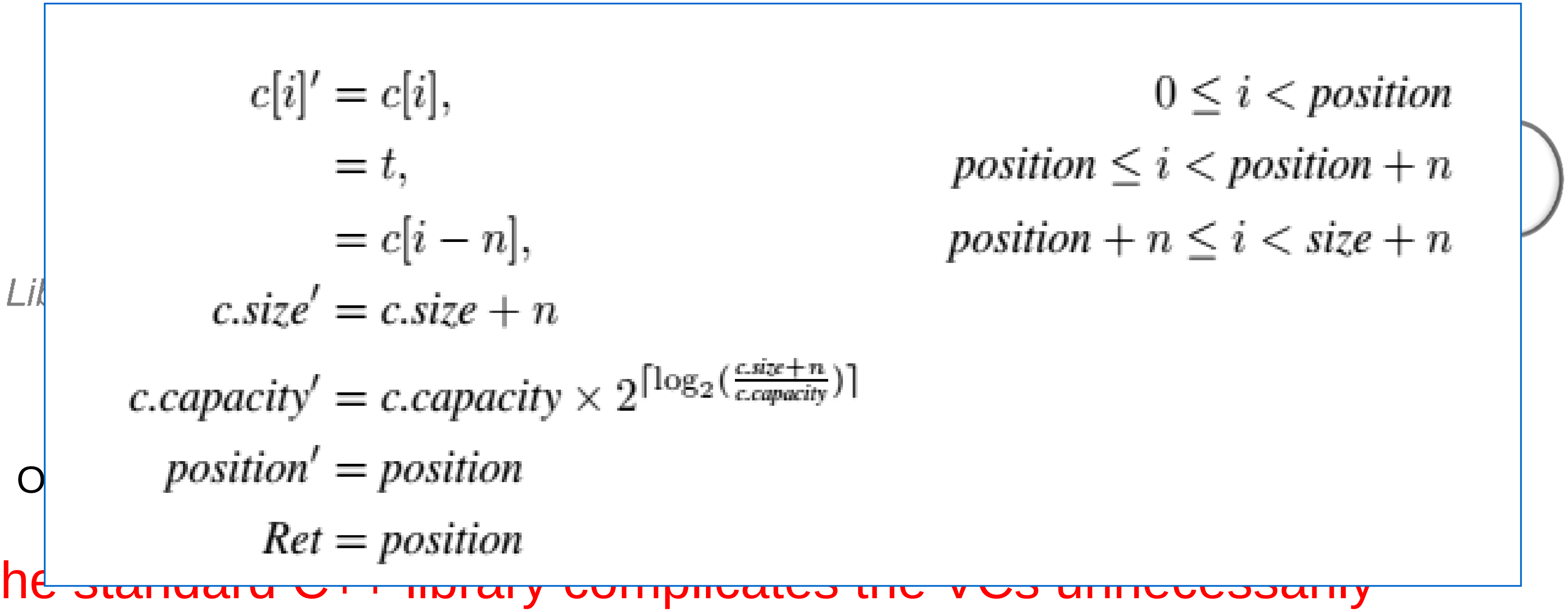
- SystemC consists of a set of C++ classes that simulates **concurrent processes** using plain C++
  - **object-oriented design** and **template classes**



the standard C++ library complicates the VCs unnecessarily

BMC of SystemC/C++

- SystemC consists of a set of C++ classes that simulates **concurrent processes** using plain C++



$$
\begin{aligned}
c[i]' &= c[i], & 0 \leq i < position \\
&= t, & position \leq i < position + n \\
&= c[i-n], & position + n \leq i < size + n \\
c.size' &= c.size + n \\
c.capacity' &= c.capacity \times 2^{\lceil \log_2 (\frac{c.size+n}{c.capacity}) \rceil} \\
position' &= position \\
Ret &= position
\end{aligned}
$$

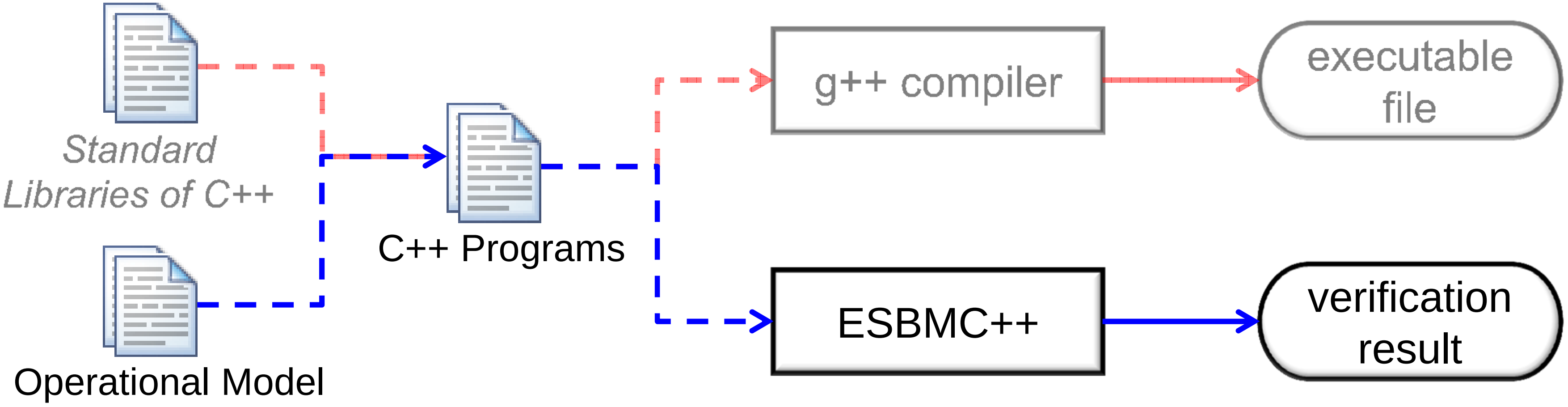
Li

O

the standard C++ library complicates the VCs unnecessarily

BMC of SystemC/C++

- SystemC consists of a set of C++ classes that simulates **concurrent processes** using plain C++
  - **object-oriented design** and **template classes**

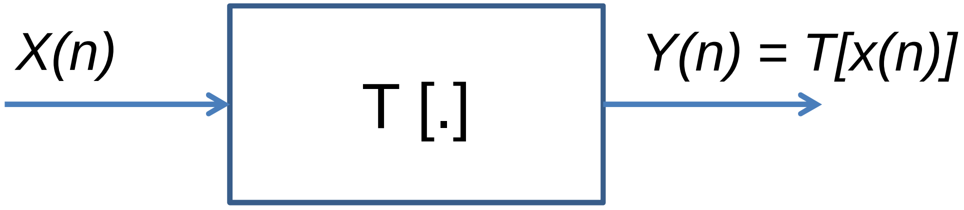


the standard C++ library complicates the VCs unnecessarily

- hypothesis:
  - abstract representation of the standard C++ libraries to conservatively approximate their semantics

BMC of Discrete-Time Systems

- discrete-time systems consist of a **mathematical operator** that maps one signal into another signal

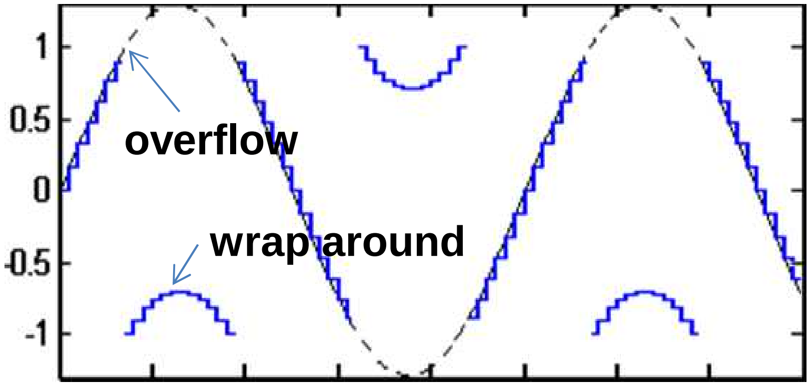

$X(n)$

$\boxed{T[.]}$

$Y(n) = T[x(n)]$

$$y(n) = -\sum_{k=1}^{N} a_k\, \mathrm{y(n-k)} + \sum_{k=0}^{M} b_k\, \mathrm{x(n-k)}$$

MAX = $2^{k-1} - 2^{-l}$

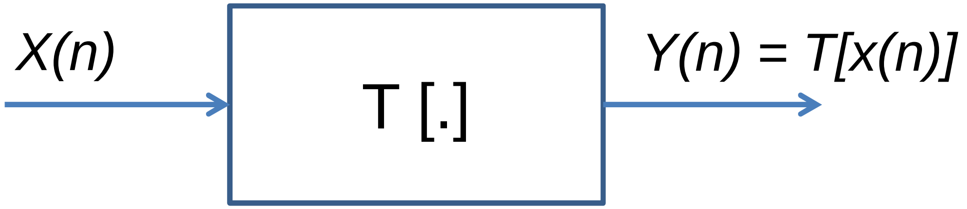


MIN = -2^{k-1}

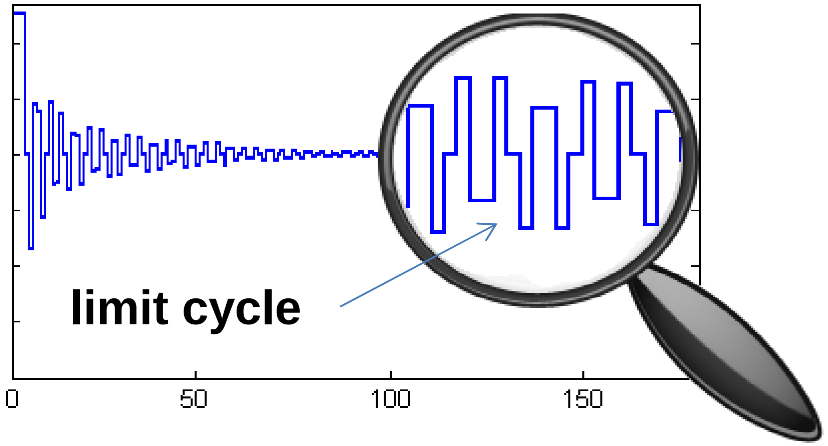
fixed-point implementation leads to errors due to the finite word-length

BMC of Discrete-Time Systems

- discrete-time systems consist of a **mathematical operator** that maps one signal into another signal



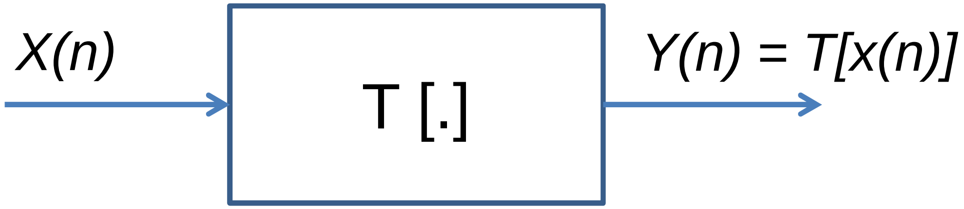
$$y(n) = -\sum_{k=1}^{N} a_k \, \mathrm{y(n-k)} + \sum_{k=0}^{M} b_k \, \mathrm{x(n-k)}$$



fixed-point implementation leads to errors due to the finite word-length

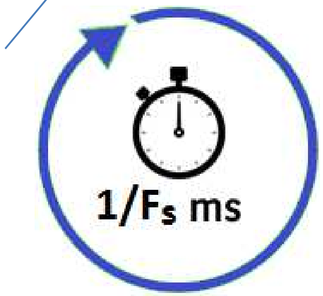
BMC of Discrete-Time Systems

- discrete-time systems consist of a **mathematical operator** that maps one signal into another signal



$l_{timing} \Leftrightarrow ((N \times T) \leq D)$

time constraint



$$y(n) = -\sum_{k=1}^{N} a_k \, \mathrm{y(n-k)} + \sum_{k=0}^{M} b_k \, \mathrm{x(n-k)}$$

fixed-point implementation leads to errors due to the finite word-length

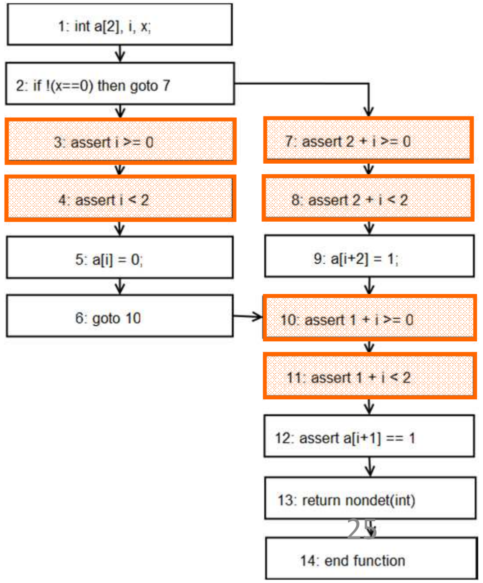
- hypothesis:
  - discrete-time systems realization has a **rigid structure**
  - simplify the **models** according to the **property** to be verified

Software BMC using ESBMC

- program modelled as state transition system
  - *state*: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

} crucial

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```



Software BMC using ESBMC

- program modelled as state transition system
  - *state*: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

  crucial
- front-end converts unrolled and optimized program into SSA

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

$g_1 = x_1 == 0$

$a_1 = a_0$ WITH $[i_0{:=}0]$

$a_2 = a_0$

$a_3 = a_2$ WITH $[2+i_0{:=}1]$

$a_4 = g_1\ ?\ a_1 : a_3$

$t_1 = a_4\,[1+i_0] == 1$

Software BMC using ESBMC

- program modelled as state transition system
  - *state*: program counter and program variables
  - derived from control-flow graph
  - checked safety properties give extra nodes
- program unfolded up to given bounds
  - loop iterations
  - context switches
- unfolded program optimized to reduce blow-up
  - constant propagation
  - forward substitutions

  } crucial
- front-end converts unrolled and optimized program into SSA
- extraction of *constraints C* and *properties P*
  - specific to selected SMT solver, uses theories
- satisfiability check of $C \wedge \neg P$

```
int main() {
  int a[2], i, x;
  if (x==0)
    a[i]=0;
  else
    a[i+2]=1;
  assert(a[i+1]==1);
}
```

$$C := \left[\begin{array}{l} g_1 := (x_1 = 0) \\ \wedge\, a_1 := store(a_0, i_0, 0) \\ \wedge\, a_2 := a_0 \\ \wedge\, a_3 := store(a_2, 2+i_0, 1) \\ \wedge\, a_4 := ite(g_1, a_1, a_3) \end{array}\right]$$

$$P := \left[\begin{array}{l} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\, 2+i_0 \geq 0 \wedge 2+i_0 < 2 \\ \wedge\, 1+i_0 \geq 0 \wedge 1+i_0 < 2 \\ \wedge\, select(a_4, i_0+1) = 1 \end{array}\right]$$

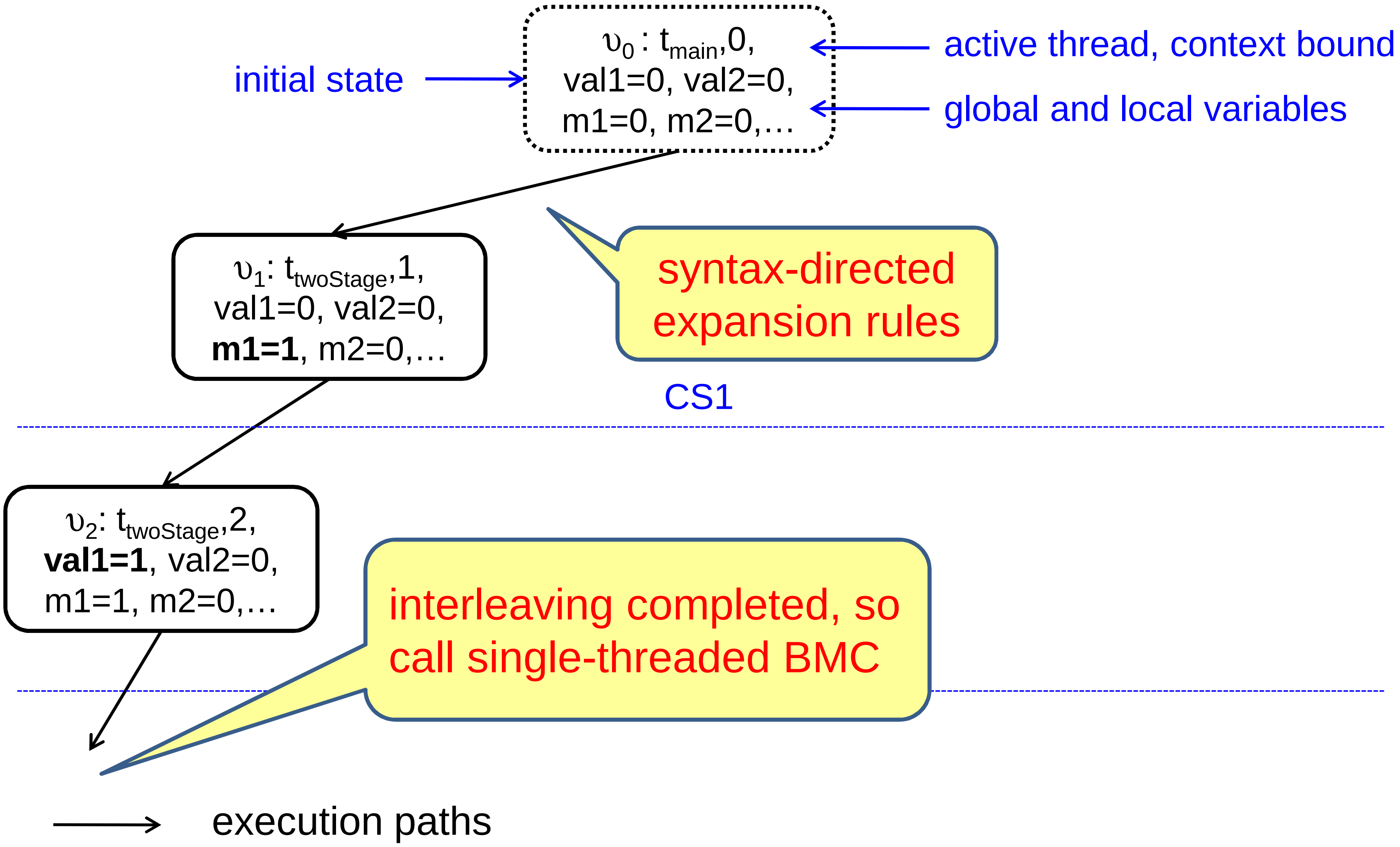
Context-Bounded Model Checking in ESBMC

Idea: iteratively generate all possible interleavings and call the BMC procedure on each interleaving

… combines

- **symbolic** model checking: on each individual interleaving
- **explicit state** model checking: explore all interleavings
  - bound the number of context switches allowed among threads

Lazy Exploration of the Reachability Tree



initial state

υ_0 : t_{main},0,
val1=0, val2=0,
m1=0, m2=0,…

active thread, context bound

global and local variables

υ_1: $t_{twoStage}$,1,
val1=0, val2=0,
m1=1, m2=0,…

syntax-directed expansion rules

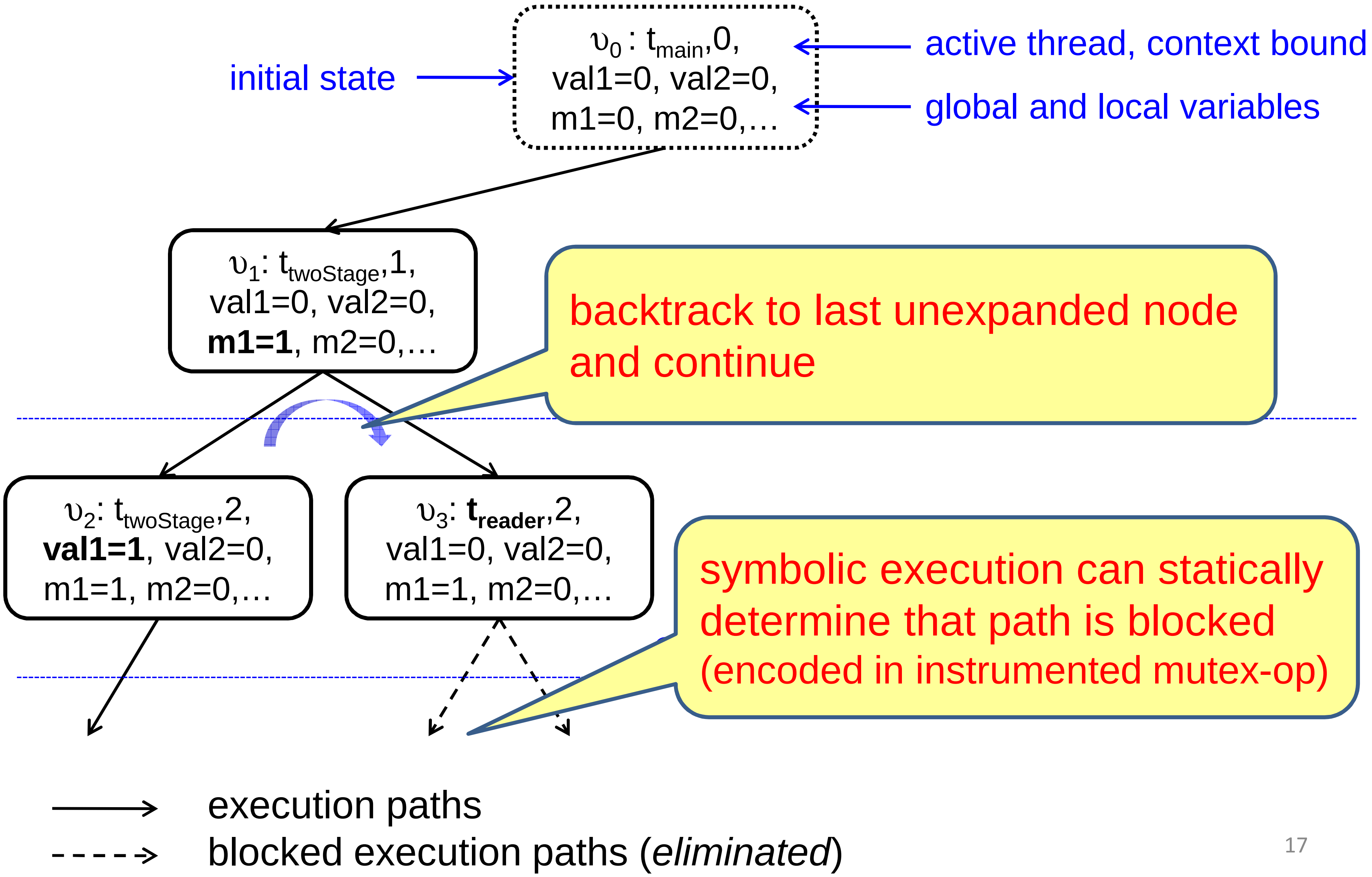
CS1

υ_2: $t_{twoStage}$,2,
val1=1, val2=0,
m1=1, m2=0,…

interleaving completed, so call single-threaded BMC

execution paths

Lazy Exploration of the Reachability Tree


initial state
active thread, context bound
global and local variables
υ_0 : t_{main},0,
val1=0, val2=0,
m1=0, m2=0,…
υ_1: $t_{twoStage}$,1,
val1=0, val2=0,
m1=1, m2=0,…
υ_2: $t_{twoStage}$,2,
val1=1, val2=0,
m1=1, m2=0,…
υ_3: $\mathbf{t_{reader}}$,2,
val1=0, val2=0,
m1=1, m2=0,…

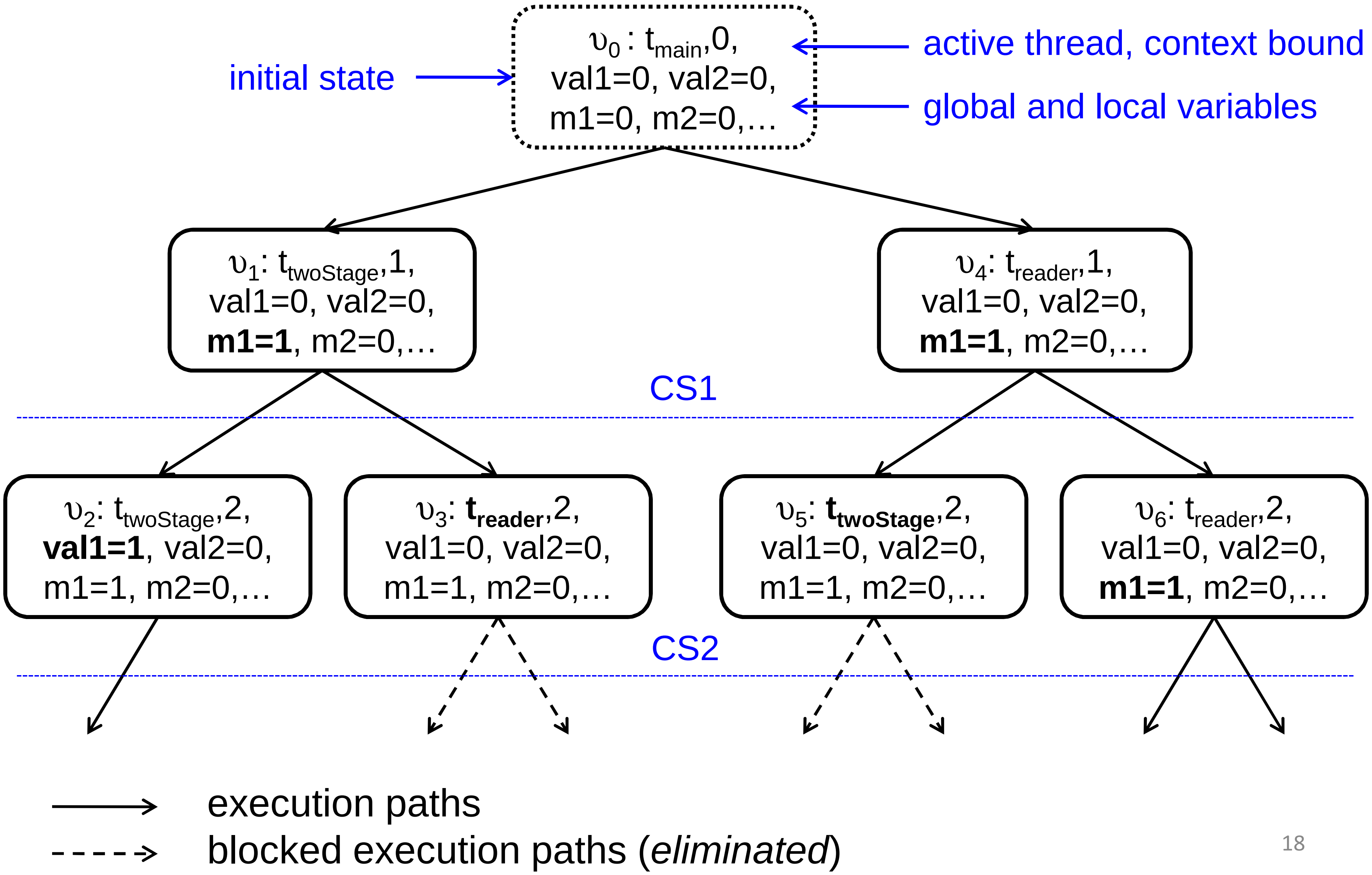
backtrack to last unexpanded node and continue

symbolic execution can statically determine that path is blocked (encoded in instrumented mutex-op)

execution paths

blocked execution paths (*eliminated*)

Lazy Exploration of the Reachability Tree



υ_0 : t_{main},0, val1=0, val2=0, m1=0, m2=0,…

initial state

active thread, context bound

global and local variables

υ_1: $t_{twoStage}$,1, val1=0, val2=0, **m1=1**, m2=0,…

υ_4: t_{reader},1, val1=0, val2=0, **m1=1**, m2=0,…

CS1

υ_2: $t_{twoStage}$,2, **val1=1**, val2=0, m1=1, m2=0,…

υ_3: $\mathbf{t_{reader}}$,2, val1=0, val2=0, m1=1, m2=0,…

υ_5: $\mathbf{t_{twoStage}}$,2, val1=0, val2=0, m1=1, m2=0,…

υ_6: t_{reader},2, val1=0, val2=0, **m1=1**, m2=0,…

CS2

execution paths

blocked execution paths (*eliminated*)

Achievements

- proposed first SMT-based context-BMC for full C
  - verifies single- and multi-threaded software (ASE’09, distinguished paper award at ICSE’11, TSE’12)
    - discrete-time systems (SBrT’13) and C++ (ECBS’13)
  - combines plain BMC with k-induction (TACAS’13, SBESC’13)
  - *found undiscovered bugs related to arithmetic overflow, buffer overflow, and invalid pointer in standard benchmarks*
    - *confirmed by the benchmark’s creators (NOKIA, NEC, NXP)*
  - most prominent BMC tool (two bronze medals in the overall ranking at TACAS’12 and TACAS’13)
- users of our ESBMC model checker
  - Airbus, Fraunhofer-Institut (Germany), LIAFA laboratory (France), University of Tokyo (Japan), Nokia Institute of Technology (Brazil)