SAT-based verification temporal induction

Mary Sheeran, Chalmers

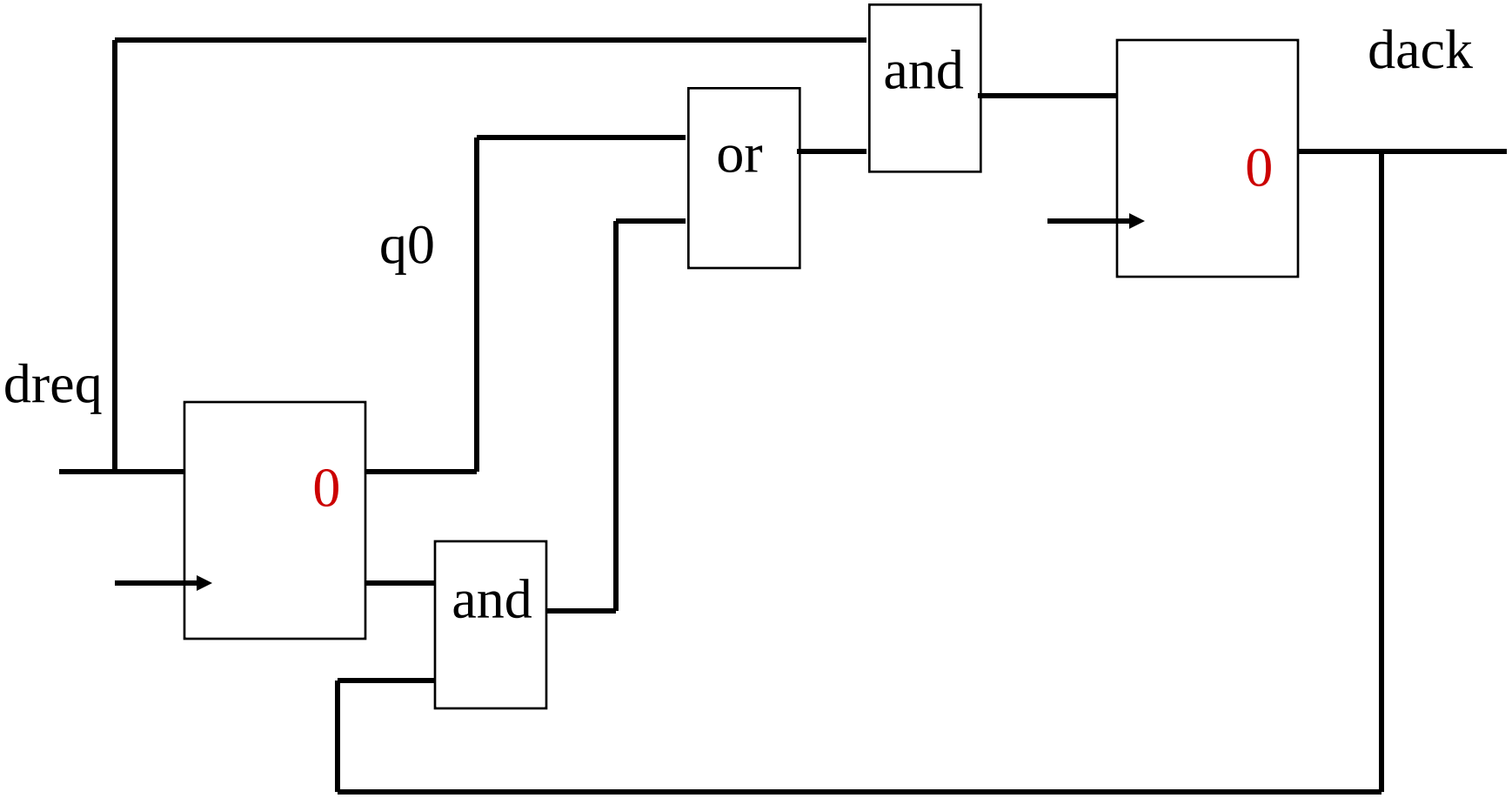
(Revised by Emil Axelsson)

SAT-based verification now hot

- Used here in Sweden since 1989 mostly in safety critical applications (railway control program verification)
- Bounded Model Checking a sensation in 1998
- SAT-based safety property verification in Lava since 1997
- Basic complete temporal induction method described here invented by Stålmarck during a talk on inductive proofs of circuits by Koen Claessen
- SAT-based Induction (engine H) and BMC used in Jasper Gold. Also in IBM SixthSense, at Intel etc.

Bounded Model Checking (BMC)

- Look for bugs up to a certain length
- Proposed for use with SAT
- Used successfully in large companies, most often for safety properties (Intel, IBM)
- Can be extended to give proofs and not just bug-finding in the particular case of safety properties. (Stålmarck et al discovered this independently of the BMC people.)
- See also work by McMillan on SAT-based unbounded model checking


and
dack
or
0
q0
dreq
0
and

Representing circuit behaviour as formulas

$I(q0,dack) = \neg q0 \wedge \neg dack$

$T((q0,dack),(q0',dack'))$

$$= (q0' \leftrightarrow dreq) \wedge (dack' \leftrightarrow dreq \wedge (q0 \vee (\neg q0 \wedge dack)))$$

Representing circuit behaviour as formulas

$I(q0,dack) = \neg q0 \wedge \neg dack$

Initial state(s)

$T((q0,dack),(q0',dack'))$

Transition relation

$$= (q0' \leftrightarrow dreq) \wedge (dack' \leftrightarrow dreq \wedge (q0 \vee (\neg q0 \wedge dack)))$$

Representing circuit behaviour as formulas

$I(q0,dack) = \neg q0 \wedge \neg dack$

Initial state(s)

$T((q0,dack),(q0',dack'))$

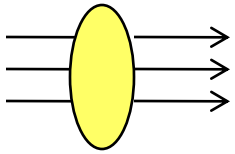
Transition relation

$= (q0' \leftrightarrow dreq) \wedge (dack' \leftrightarrow dreq \wedge (q0 \vee (\neg q0 \wedge dack)))$

new state depends also on input

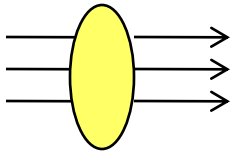
Picturing transition relations

Draw $I(s)$ as



Picturing transition relations

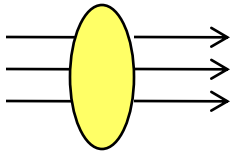
Draw I (s) as



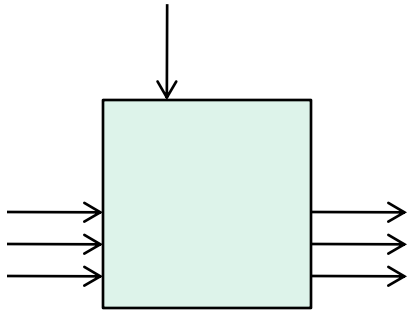
Constraint is only on the state holding elements
not on inputs

Picturing transition relations

Draw I (s) as



Draw T (s,s') as



Composing transitions into paths

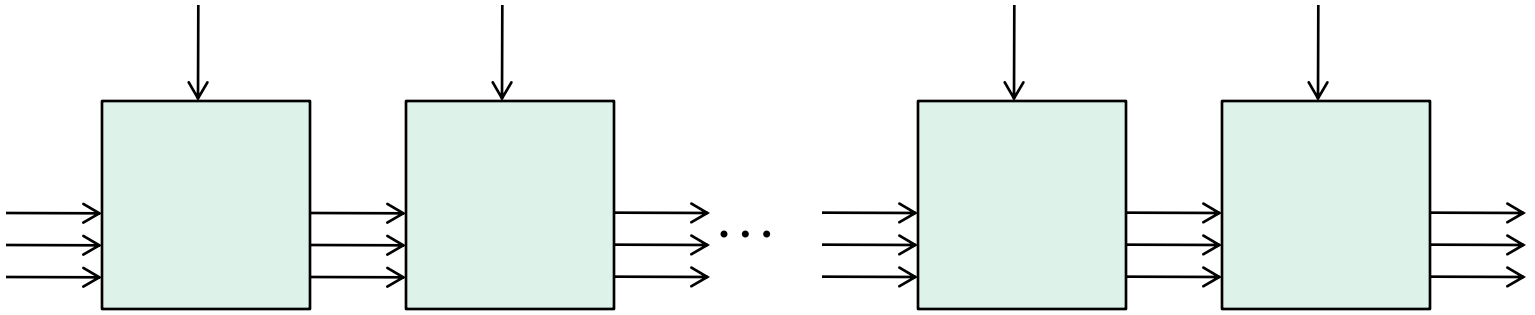
$$T^i(s_0, \ldots, s_i)$$

$$= T(s_0, s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{i-1}, s_i)$$

Composing transitions into paths

$T^i(s_0, \ldots, s_i)$

$T(s_0, s_1) \wedge T(s_1, s_2) \wedge \ldots \wedge T(s_{i-1}, s_i)$



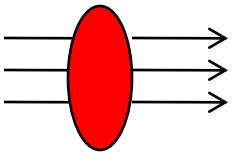
i copies

Representing the bad states

Similar to use of formula for initial states

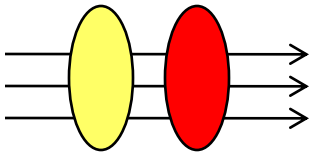
$B(q0,dack) = \neg q0 \wedge dack$

Drawing the bad states



B(s)

BMC for simple safety properties

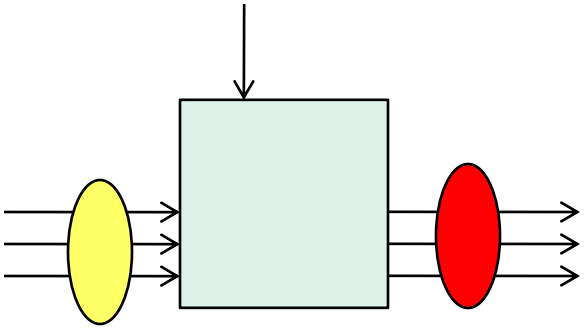


Corresponding formula:

$$I(s_0) \wedge B(s_0)$$

If the formula is satisfiable, we have a bug already in the initial state!

BMC for simple safety properties

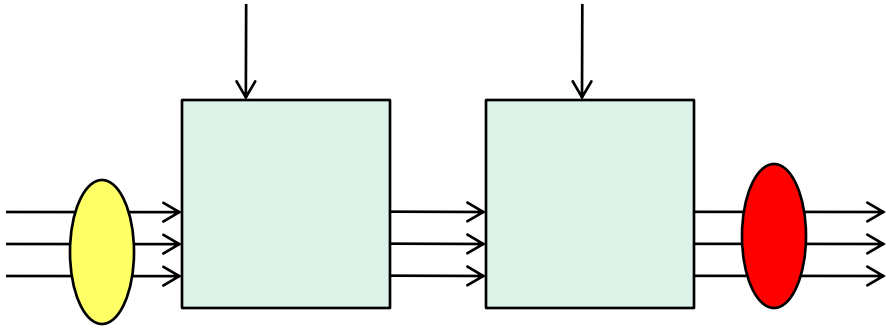


Corresponding formula:

$$I(s_0) \wedge T(s_0,s_1) \wedge B(s_1)$$

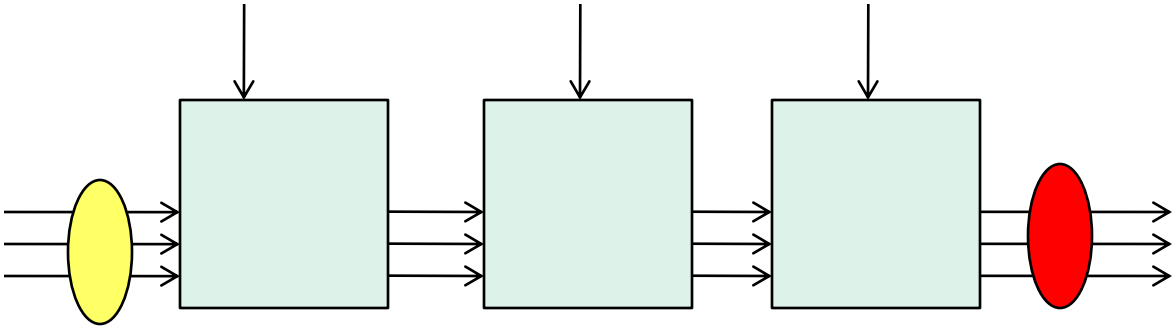
Satisfiable => bug after one step

BMC for simple safety properties



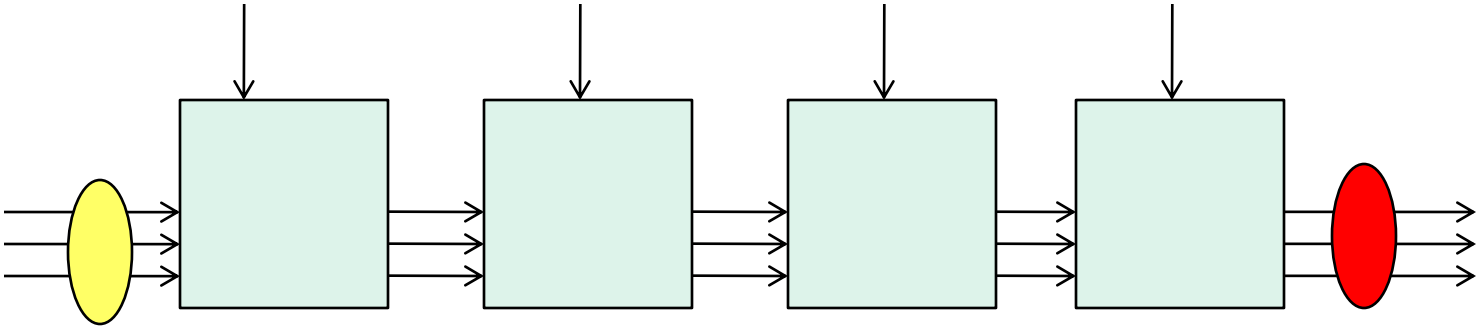
Satisfiable => bug after two steps

BMC for simple safety properties



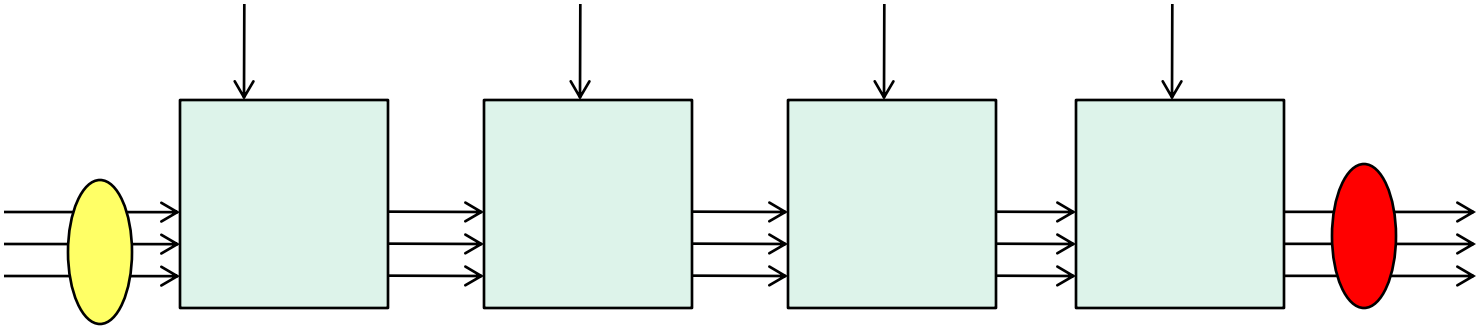
Satisfiable => bug after three steps

BMC for simple safety properties



Satisfiable => bug after four steps

BMC for simple safety properties



Forumula is $I(s_0) \wedge T^4(s_0, s_1, s_2, s_3, s_4) \wedge B(s_4)$

Call this $Base_4$ and generalise to $Base_i$

Improvement: Start with bound n

Choose a bound n

If the formula

$$I(s_0) \wedge T^n(s_0, \ldots, s_n) \wedge (B(s_0) \vee B(s_1) \vee \ldots \vee B(s_n))$$

is satisfiable, then there is a bug somewhere in the first n steps through the transition system

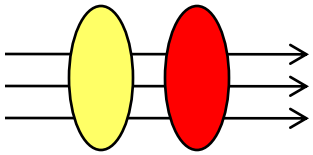
BMC

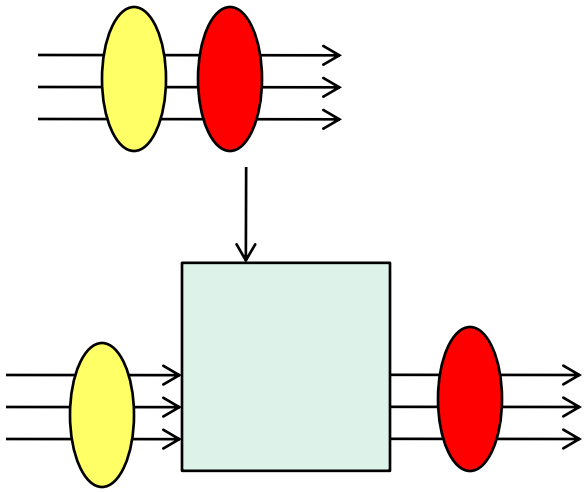
Above description covers simple safety properties

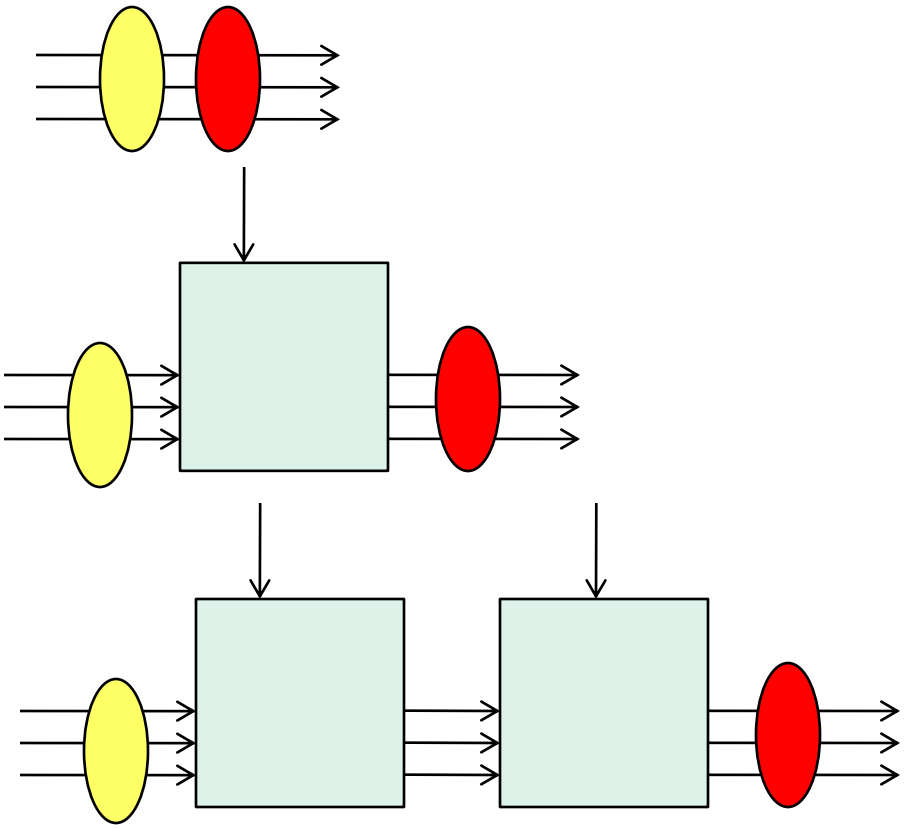
Original BMC papers cover more complex properties

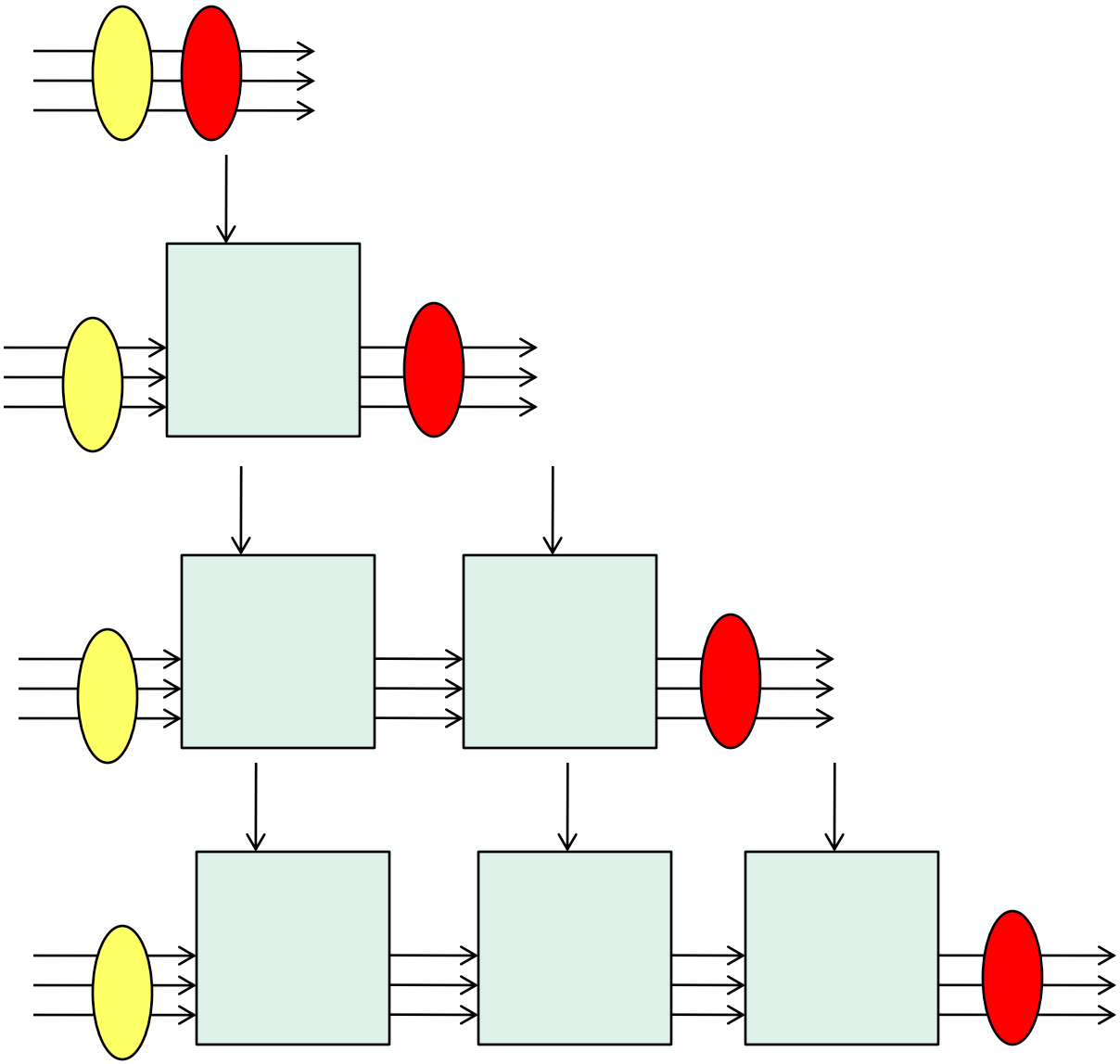
Note complete lack of quantifiers!

(Remember: BDDs support quantification; SAT doesn’t)









If system is bad

- Base_0
- Base_1
- Base_2

and so on

- Finds a shortest countermodel
- Error trace for debugging

What if system is good?

When to stop iterating?

When $I(s_0) \wedge T^i(s_0, \ldots, s_i)$ is UNSAT?

What if system is good?

When to stop iterating?

When $I(s_0) \wedge T^i(s_0, \ldots, s_i)$
is UNSAT?

No, the system never gets stuck. Will eventually come back to already visited states.

Need to check for absense of loops.

Extra formulas for loop-free ”the unique states condition”

$$U^k(s_0, \ldots, s_k) = \bigwedge_{0 \le i < j \le k} (s_i \neq s_j)$$

Quadratic number of comparisons

States are vectors of bits, so

if $s=(a,b,c,d)$ then

$s_0 \neq s_1$ is $\neg (a_0 \leftrightarrow a_1) \vee$

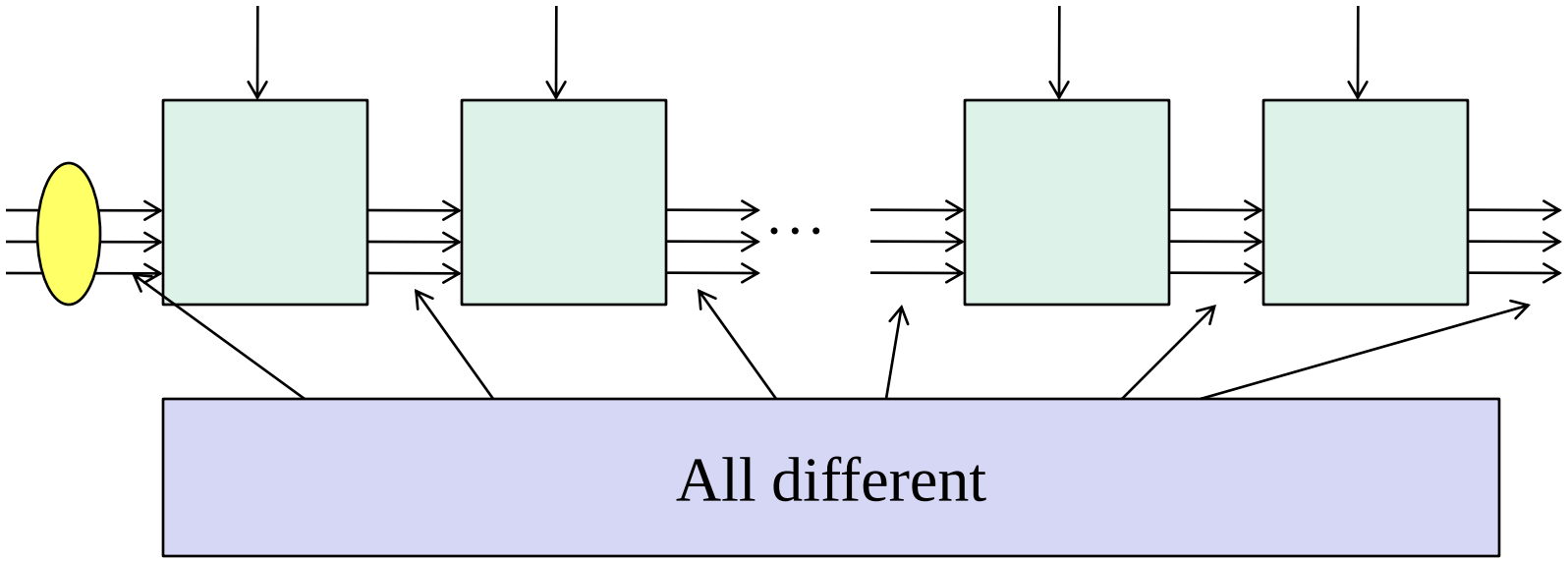
$\neg (b_0 \leftrightarrow b_1) \vee$

$\neg (c_0 \leftrightarrow c_1) \vee$

$\neg (d_0 \leftrightarrow d_1)$

We can stop if

$$I(s_0) \wedge T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i)$$

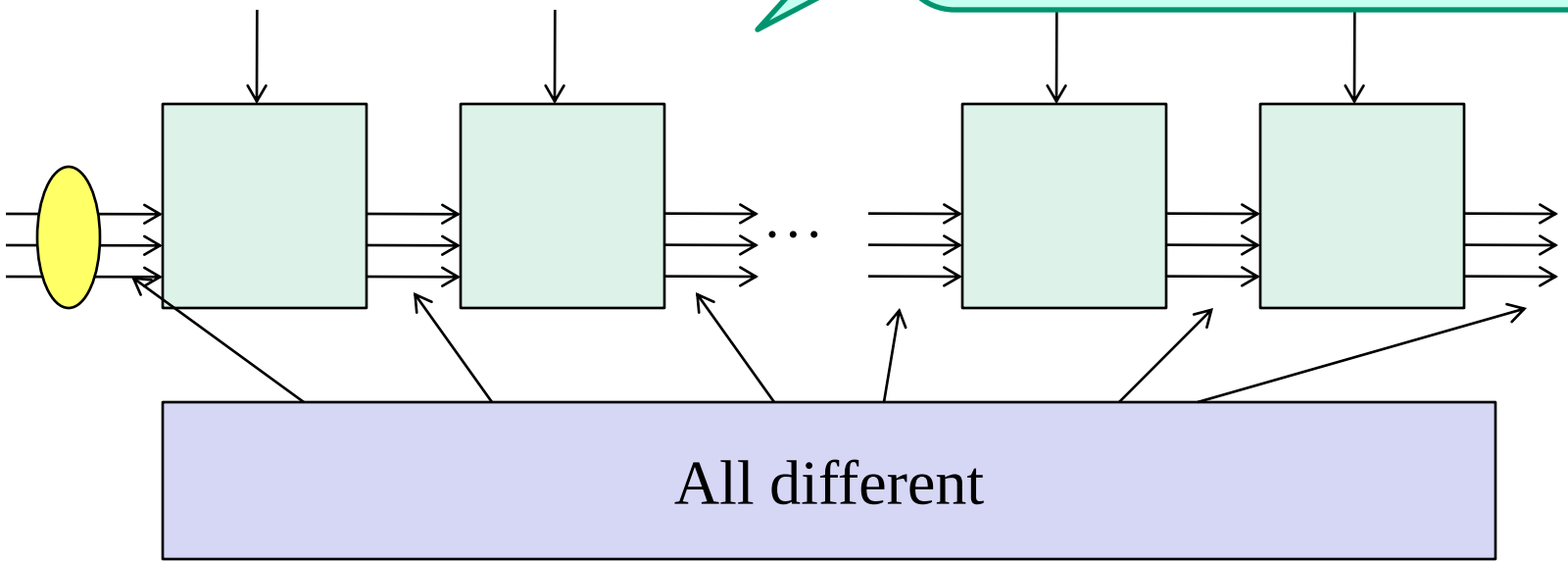


is UNSAT

We can stop if

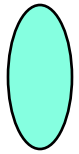
$$I(s_0) \wedge T^i(s_0, \ldots, s_i)$$

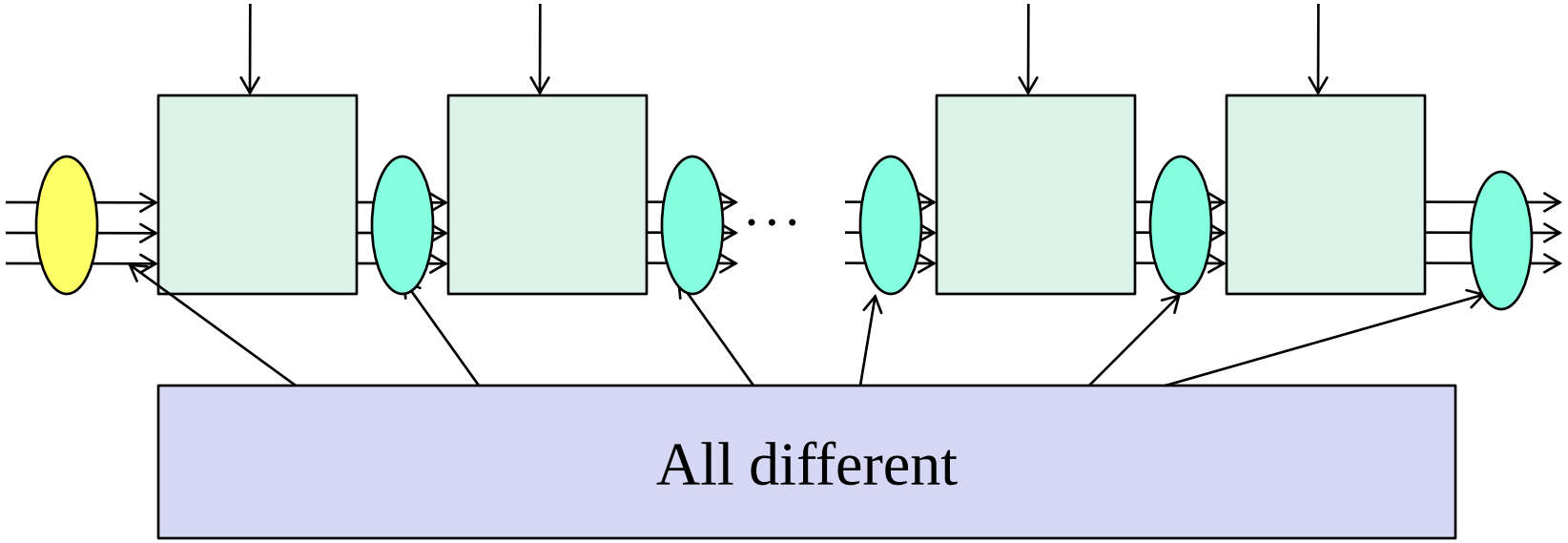
No loop-free paths of length i starting from inital states



is UNSAT

Optimization: Only consider shortest paths

- Don’t want to go back to an initial state
- Draw non-initial as 

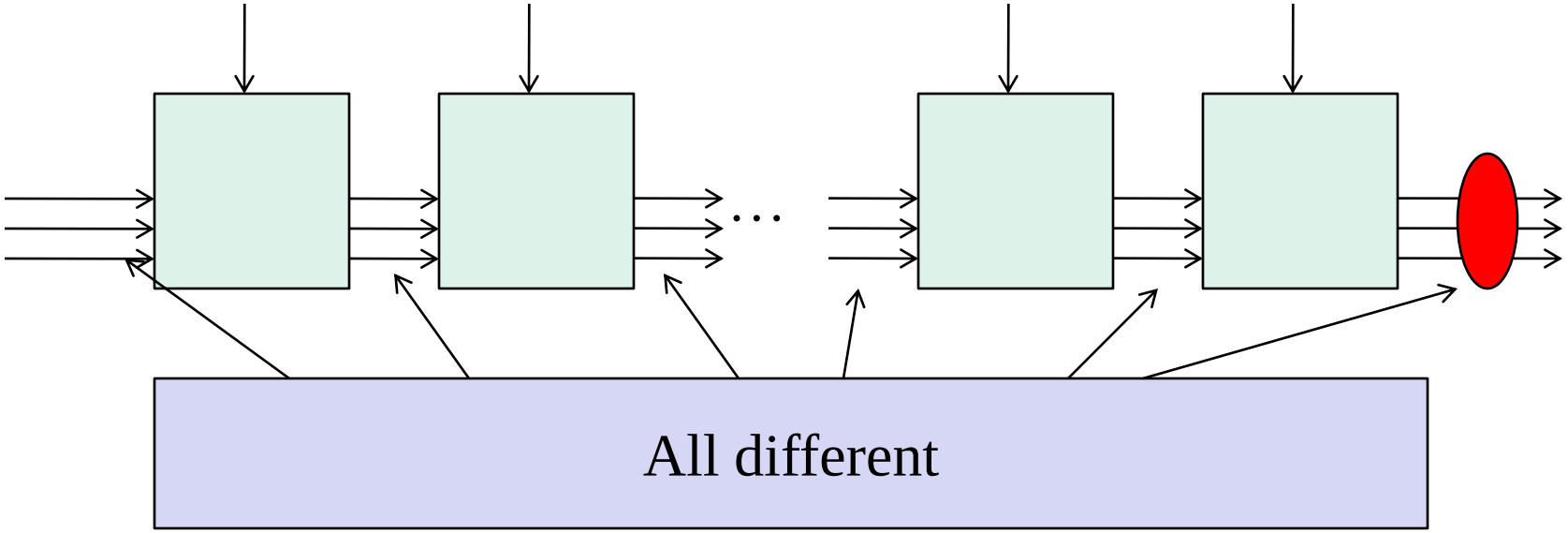


Backwards termination condition

We can also terminate when we are sure that there are no longer paths leading to a bad state.

Backwards termination condition

$$T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i) \wedge B(s_i)$$

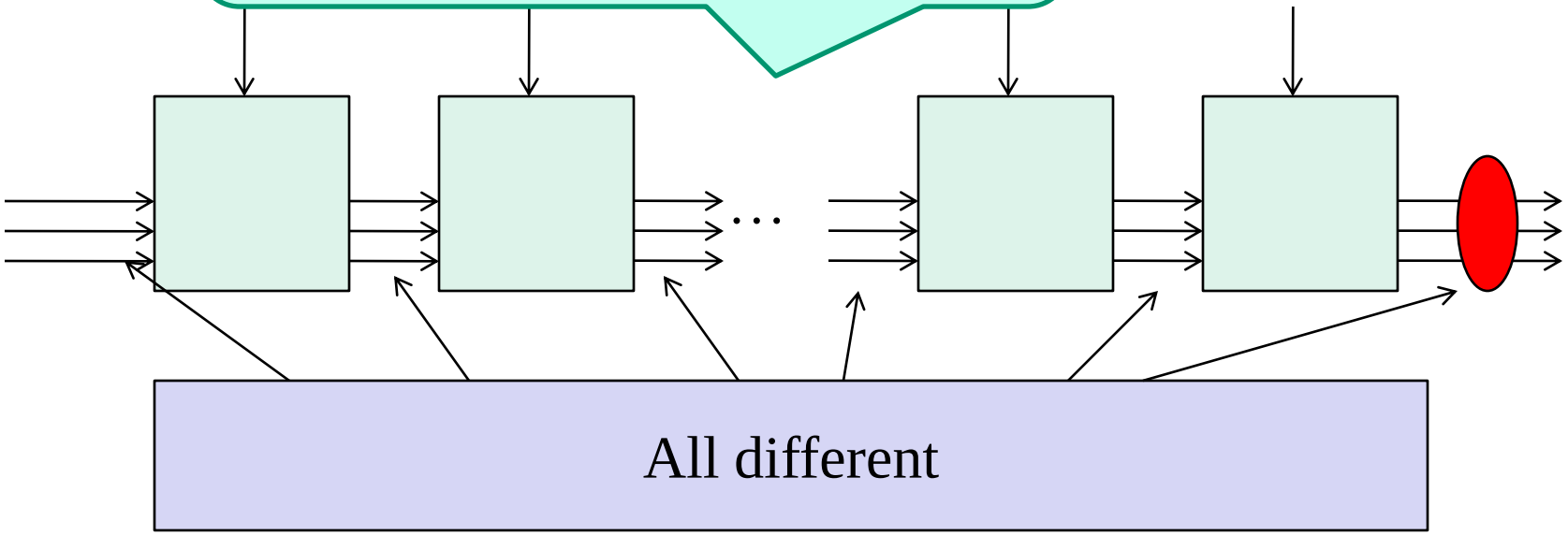


is UNSAT

Backwards termination condition

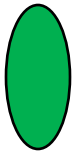
T $s_i) \wedge B(s_i)$

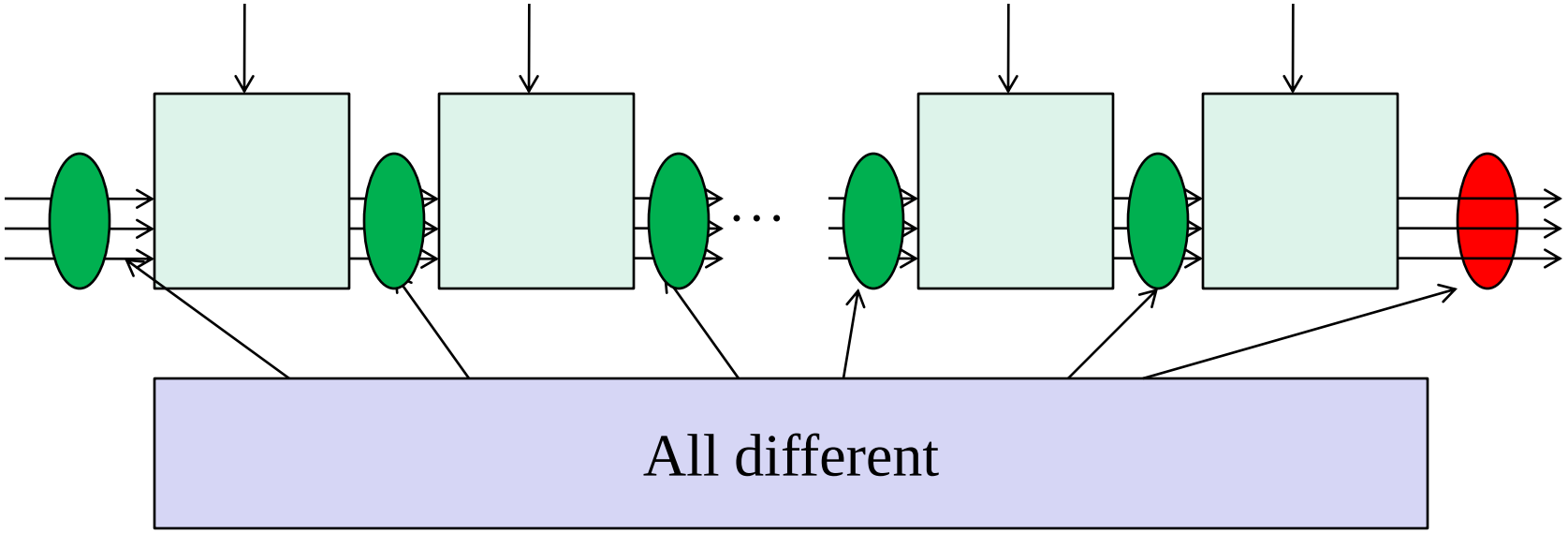
No loop-free paths of length
i ending in a bad state



is UNSAT

Optimization: Only interested in shortest paths

Draw Good = not Bad ($\neg$B) as 



This is a much better choice (may terminate much more quickly)

What do we have?

A BMC formula:

$$I(s_0) \wedge T^i(s_0, \ldots, s_i) \wedge B(s_i)$$

Two different termination conditions:

$$T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i) \wedge B(s_i)$$

$$I(s_0) \wedge T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i)$$

Define

$$Base_k = I(s_0) \wedge T^k(s_0, \ldots, s_k) \wedge B(s_k)$$

$$Step1_k = T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i) \wedge B(s_i)$$

$$Step2_k = I(s_0) \wedge T^i(s_0, \ldots, s_i) \wedge U^i(s_0, \ldots, s_i)$$

Alternatively

$$\text{Step1}_k = T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge$$

$$\bigwedge_{0 \le j \le k} \neg B(s_j) \wedge B(s_{k+1})$$

$$\text{Step2}_k = T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge$$

$$I(s_0) \wedge \bigwedge_{1 \le j \le k+1} \neg I(s_j)$$

Alternatively

$$\text{Step1}_k = T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge$$

$$\bigwedge_{0 \le j \le k} \neg B(s_j) \wedge B(s_{k+1})$$

$$\text{Step2}_k = T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge$$

$$I(s_0) \wedge \bigwedge_{1 \le j \le k+1} \neg I(s_j)$$

Won't be needed if there is only one initial state

Starts to look like induction!

UNSAT ($Step1_k$) means:

> "If we have visited i+1 unique good states, the next state is guaranteed to be good."

UNSAT ($Step2_k$) means:

> "If we have visited i+1 unique non-initial states, the state state i+2 cycles ago is guaranteed to be non-initial."

Induction

UNSAT ($Base_k$) and (UNSAT ($Step1_k$) or UNSAT ($Step1_k$))

=> Property holds (cannot reach a bad state from an initial state), by induction

Induction

UNSAT ($Base_k$) and (UNSAT ($Step1_k$) or UNSAT ($Step1_k$))

=> Property holds (cannot reach a bad state from an initial state), by induction

But:

SAT ($Base_k$) and not (UNSAT ($Step1_k$) or UNSAT ($Step1_k$))

=> ??

Then we need to increase k.

Temporal induction (Stålmarck)

k=0
while True do {
    if Sat($Base_k$) return False (and counter example)
    if Unsat($Step1_k$) or Unsat($Step2_k$) return True
    k=k+1
}

Will terminate eventually!

Temporal induction

Most presentations consider only the Step1 case but I like to keep things symmetrical

Much overlap between formulas in different iterations.
Was part of the inspiration behind the development (here at Chalmers) of the incremental SAT-solver miniSAT (open source, see minisat.se)
(see paper by Een and Sörensson in the list later)

In reality need to think hard about what formulas to give the SAT-solver.

Temporal induction

The method is sound and complete (see papers)

Gives the right answer. Gives proof, not just bug-finding.

Algorithm given above leads to a shortest counter-example

May also want to take bigger steps and sacrifice this property (though this may make less sense when using an incremental SAT-solver)

The method can be strengthened further (still ongoing research).

Definitely met with scepticism initially

Conclusion

BMC: the work-horse of formal hardware verification

SAT-based temporal induction is also much used

See our tutorial paper for info. on the history and the necessary development of SAT-solvers

Much research now concentrates on raising the level of abstraction at which formal reasoning is done

Satisfiability Module Theories (SMT) is the hot topic