SAT-based verification temporal induction

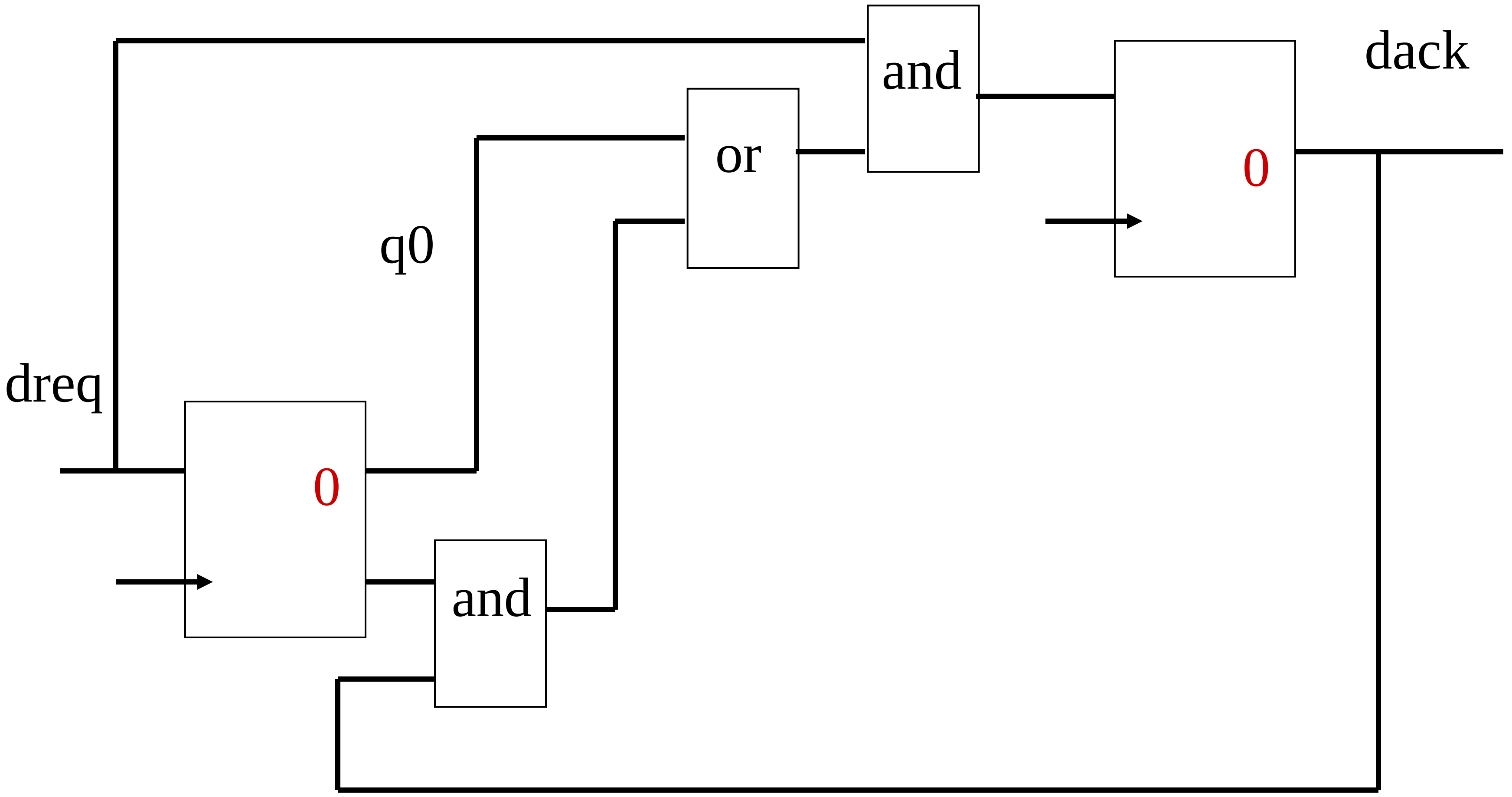
Mary Sheeran, Chalmers

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



dack
and
or
0
q0
dreq
0
and

Representing circuit behaviour as formulas

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

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

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

Representing circuit behaviour as formulas

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

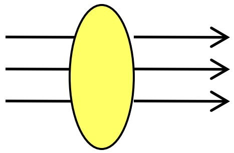
$T((q0,dack),(q0',dack'))$

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

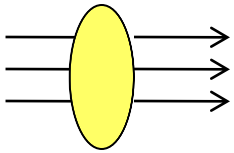
new state depends also on input

Picturing transition relations

Draw I (s) as



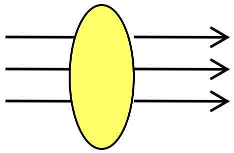
Picturing transition relations

Draw I (s) as 

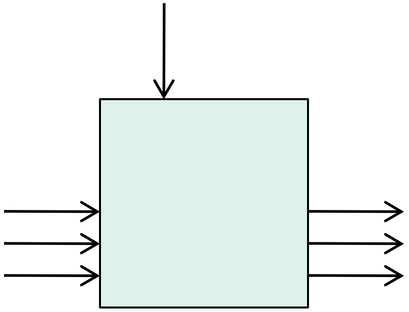
Constraint is only on the state holding elements
not on inputs

Picturing transition relations

Draw I (s) as



Draw T (s,s’) as



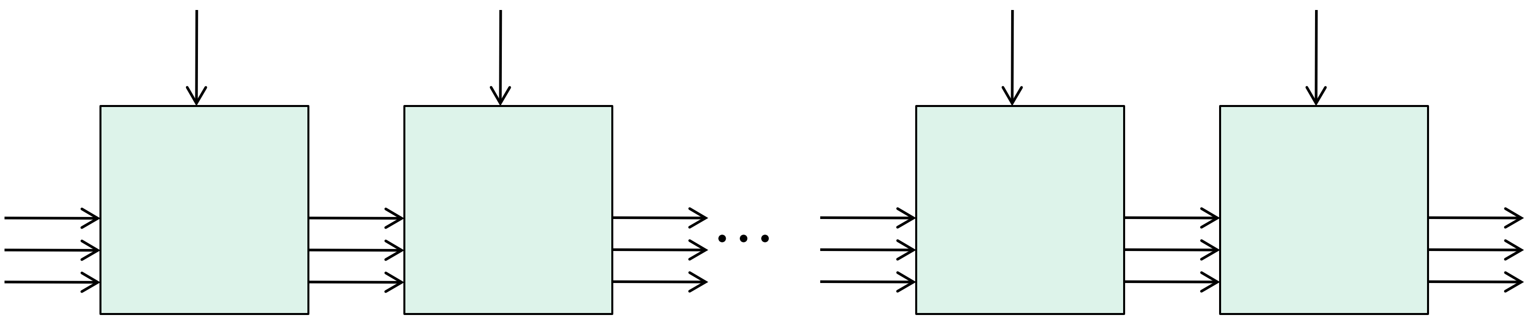
Composing transitions into paths

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

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

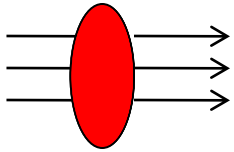
Representing the bad states

Similar to use of formula for initial states

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

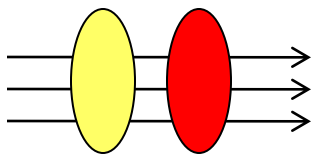
or may be using an observer

Drawing the bad states



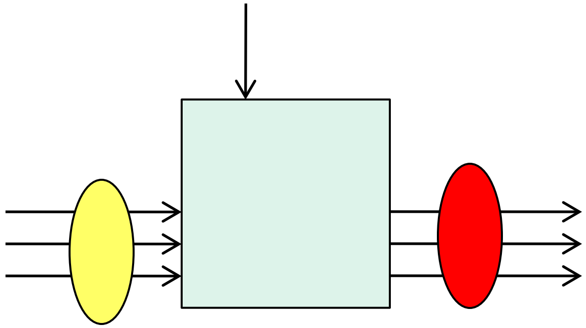
B(s)

BMC for simple safety properties



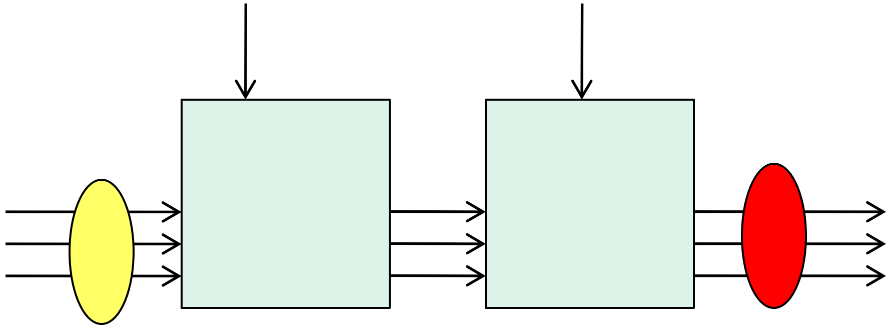
If the corresponding formula is satisfiable, we have a bug already in the initial state!

BMC for simple safety properties



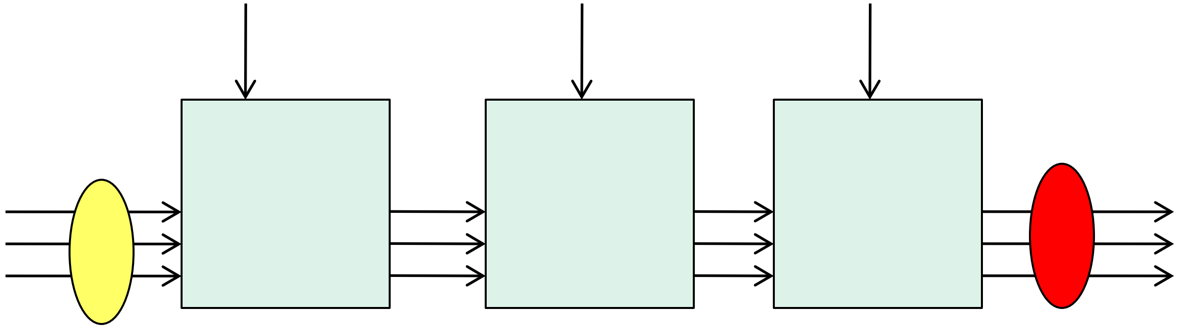
Satisfiable => bug after one step

BMC for simple safety properties



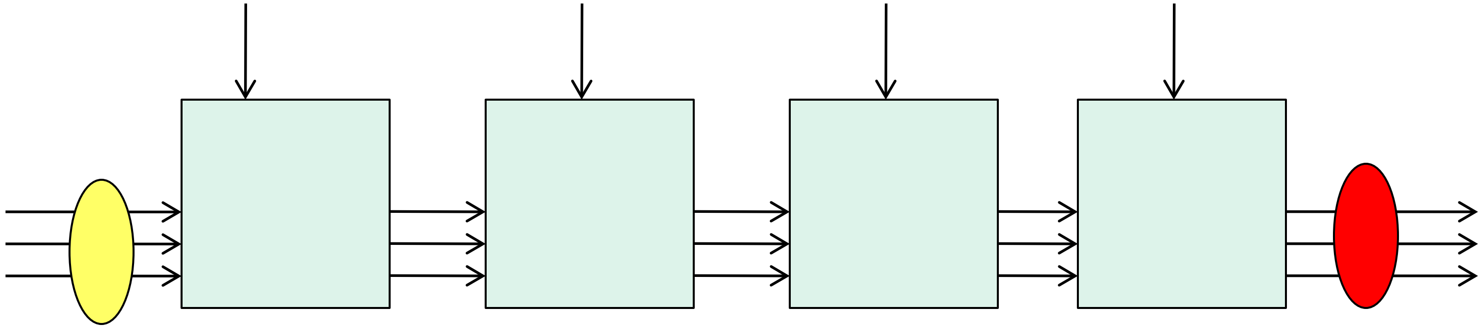
bug after two steps?

BMC for simple safety properties



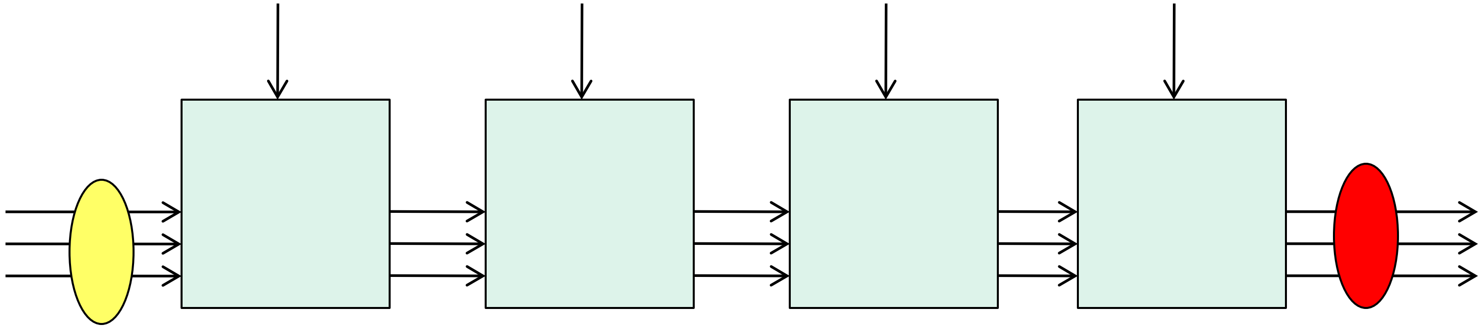
bug after three steps?

BMC for simple safety properties



bug after four steps?

BMC for simple safety properties



bug after four steps?

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

Call this Base_4 and generalise to Base_i

Can 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

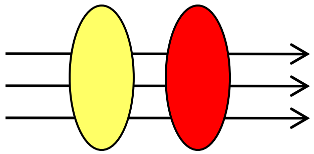
Above description covers simple safety properties

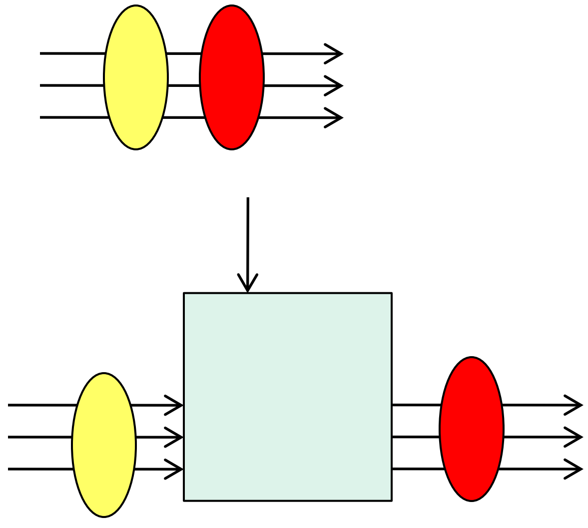
Original BMC papers cover more complex properties

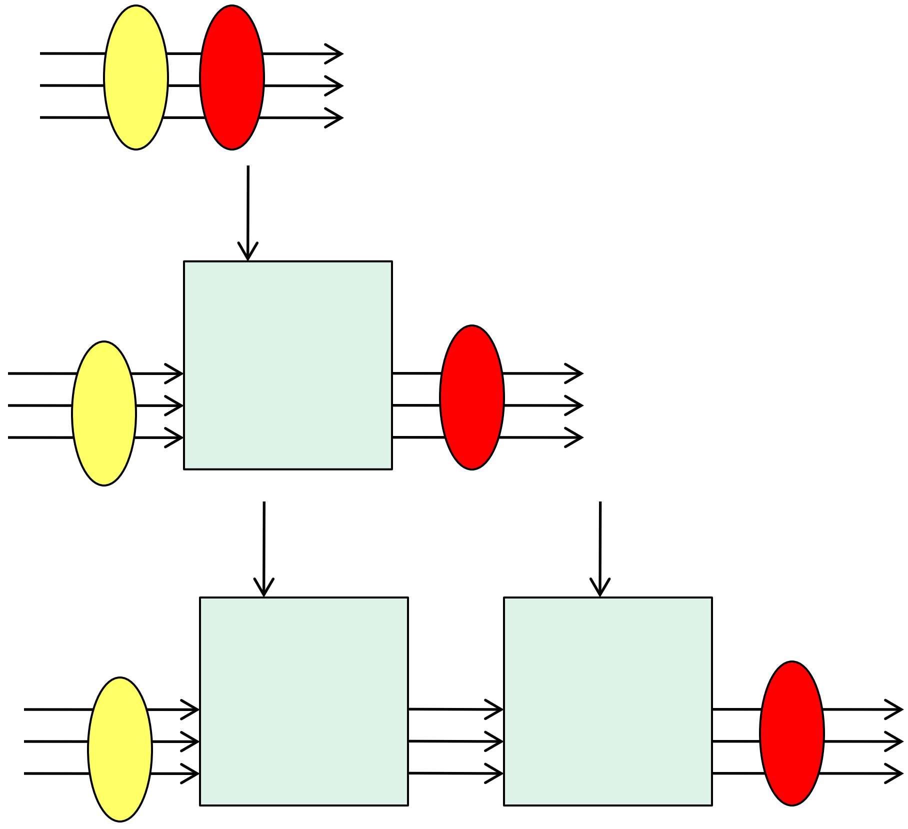
Note complete lack of quantifiers! Key point.

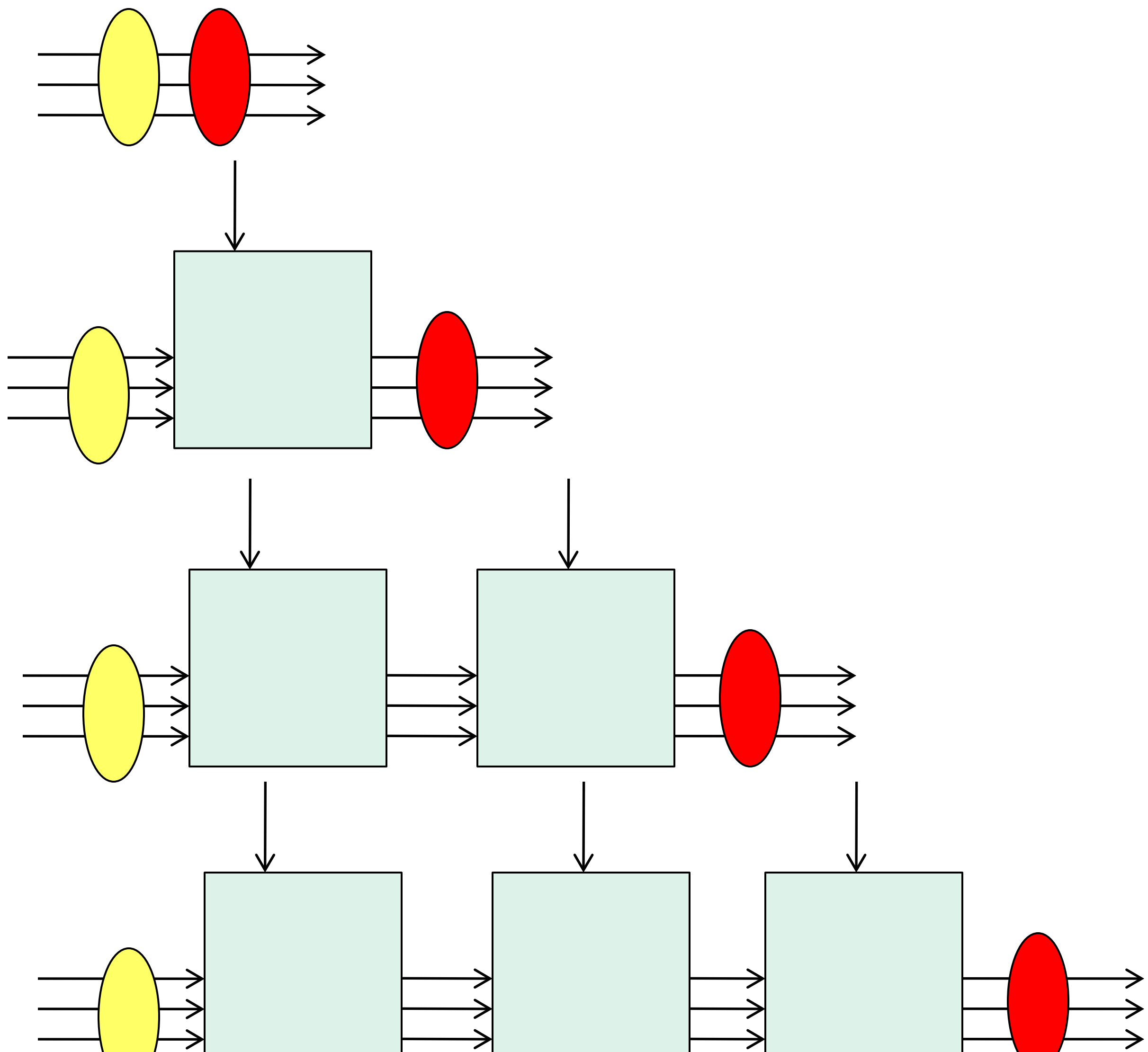
Temporal induction

Start thinking along the same lines









If system is bad

- Base_0
- Base_1
- Base_2

and so on

- Finds a shortest countermodel
- Error trace for debugging

But when can we stop?

when

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

UNSAT ?

Not quite, but

when there is no such path that is loop-free

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

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

Size??

States are vectors of bits, so

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

$s_0 \neq s_1$ is $\neg\,(\mathbf{a_0} \text{ <-> } \mathbf{a_1}) \vee$

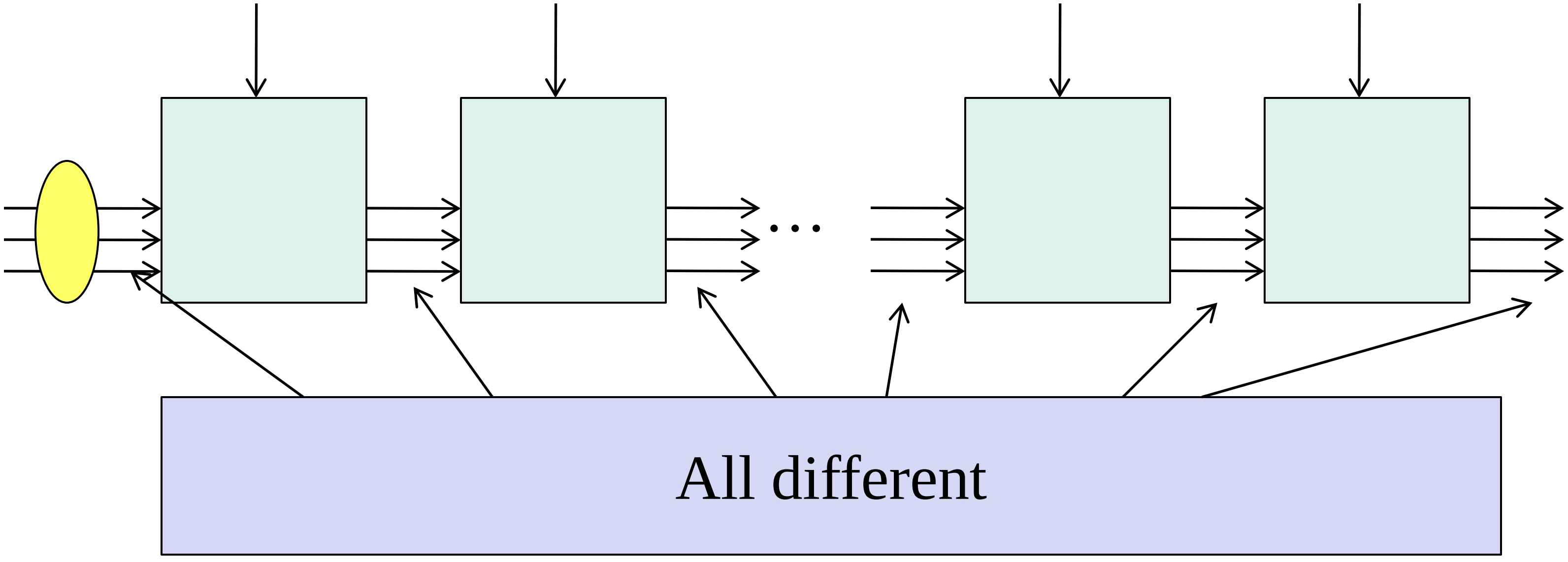
$\neg\,(\mathbf{b_0} \text{ <-> } \mathbf{b_1}) \vee$

$\neg\,(\mathbf{c_0} \text{ <-> } \mathbf{c_1}) \vee$

$\neg\,(\mathbf{d_0} \text{ <-> } \mathbf{d_1})$

We can stop if

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

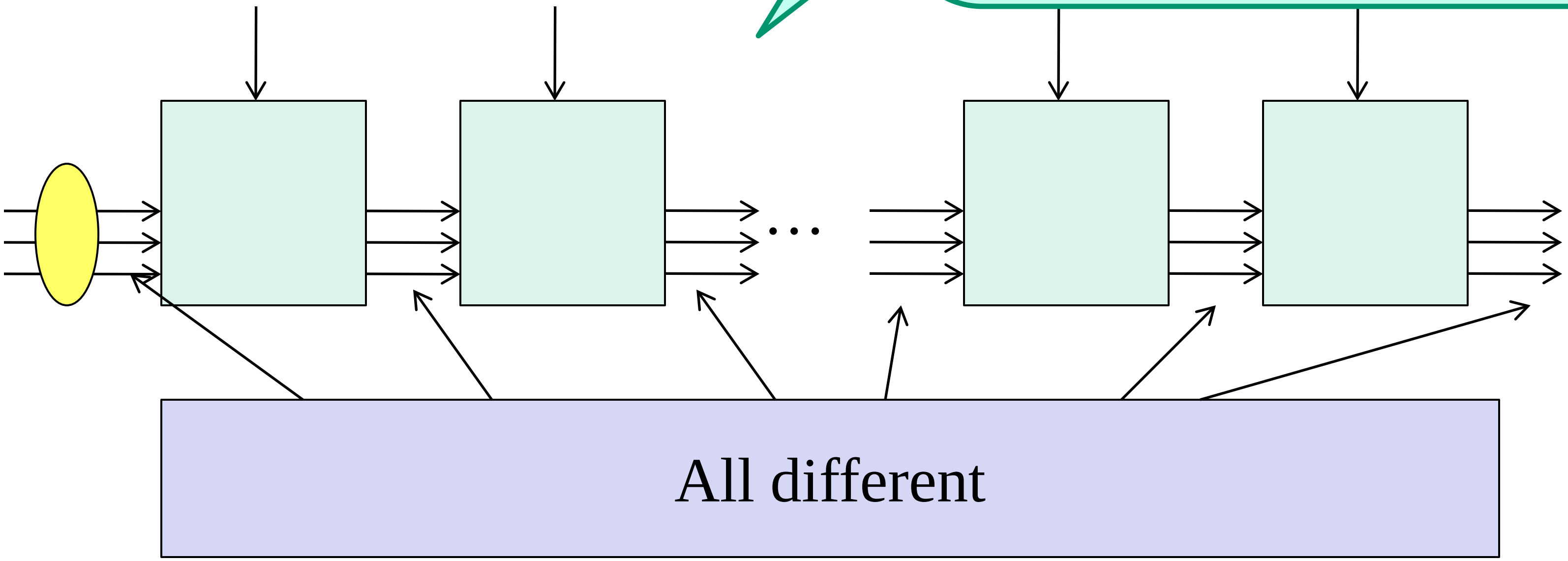


is UNSAT

We can

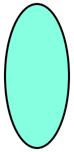
$$I(s_0) \wedge T^i(s_0, \ldots, s_i)$$

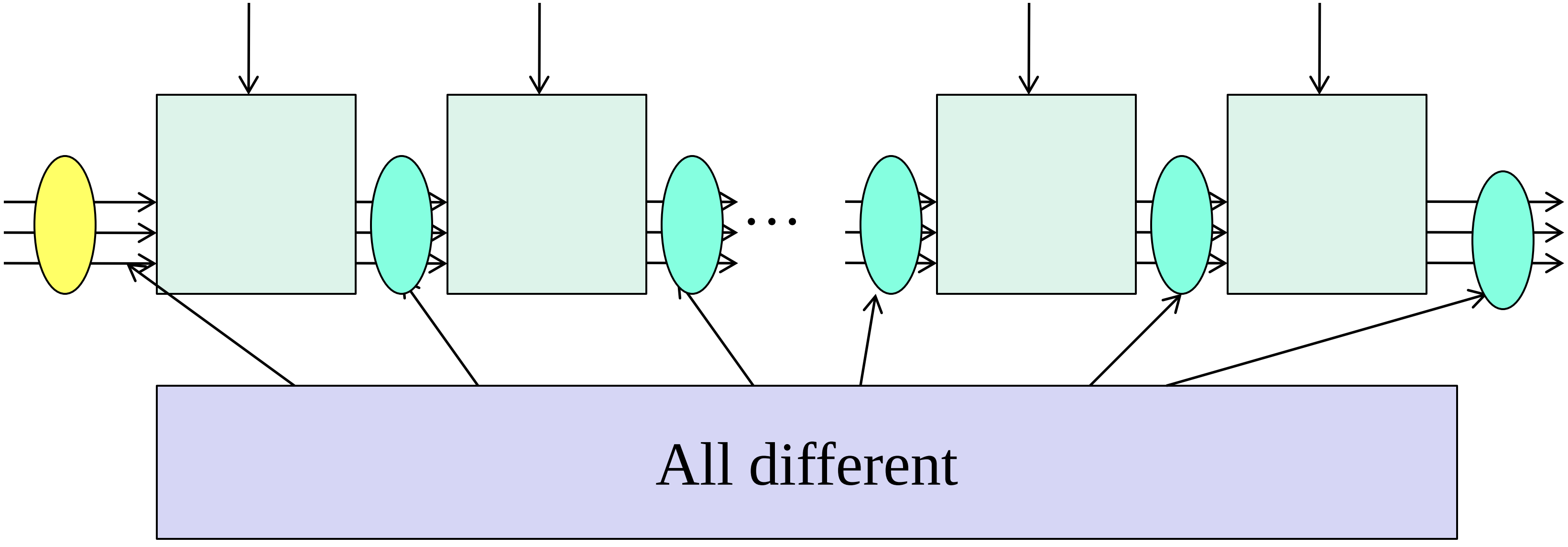
No loop-free paths of length i starting from inital states



is UNSAT

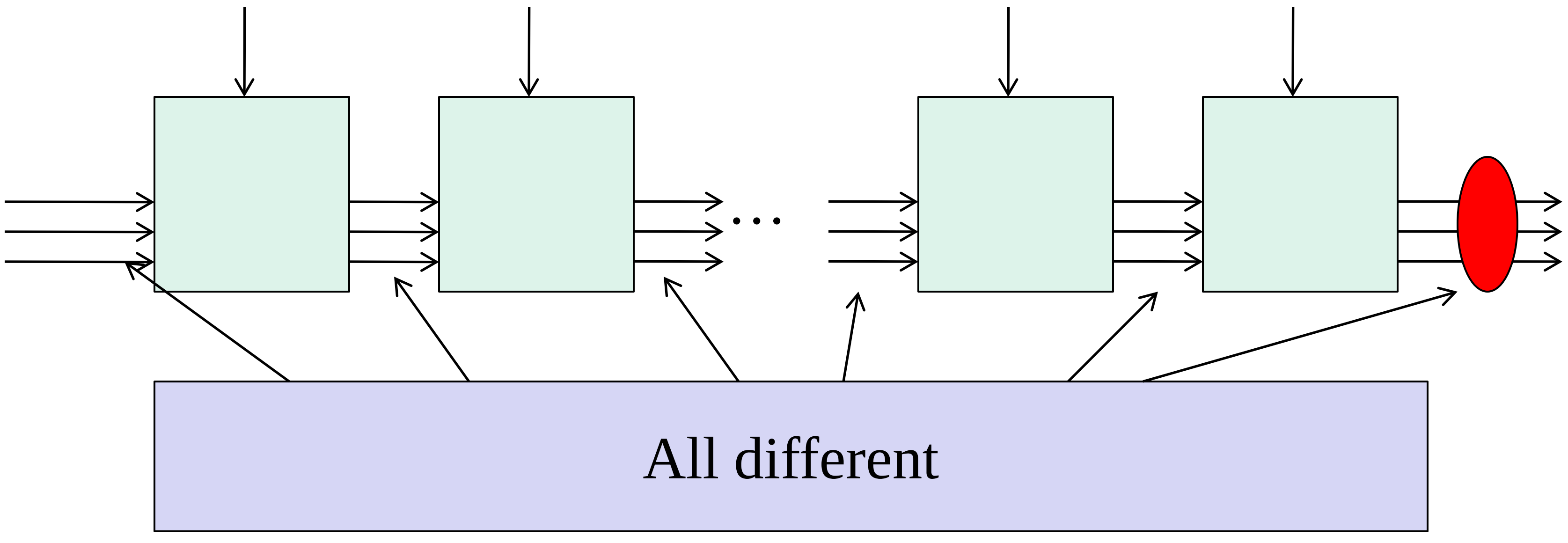
Only interested in shortest paths

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



Symmetrically, we can stop if

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

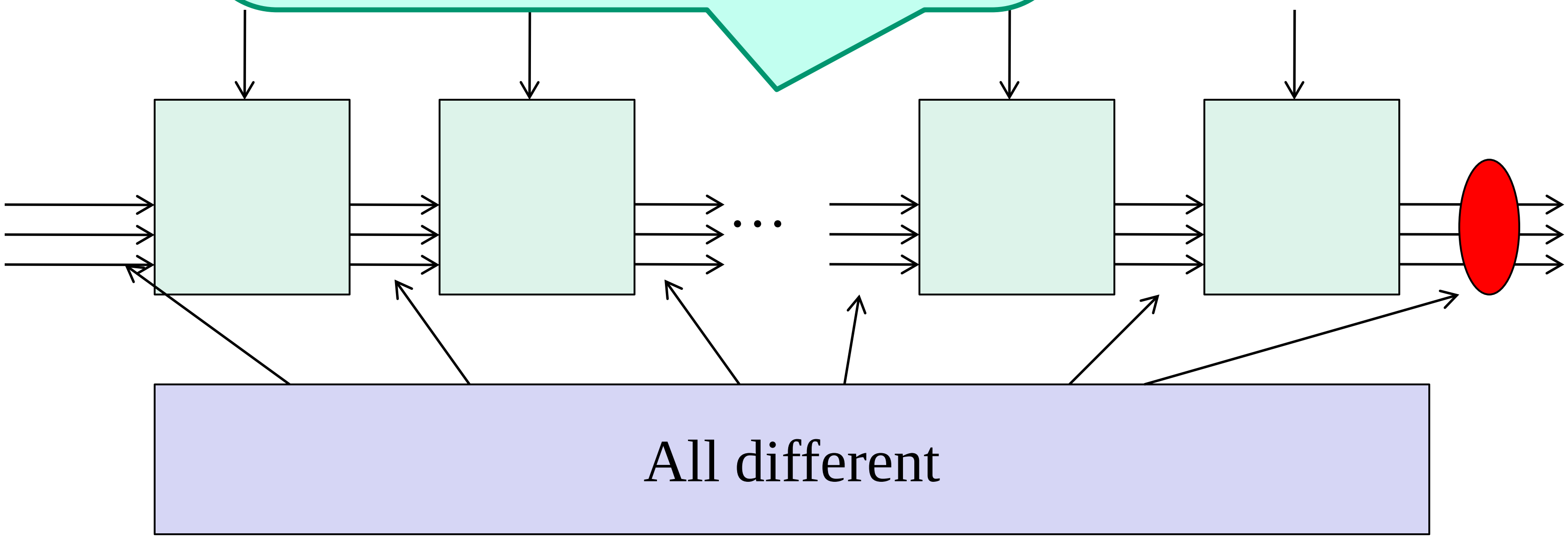


is UNSAT

No loop-free paths ending in a bad state

S an stop if

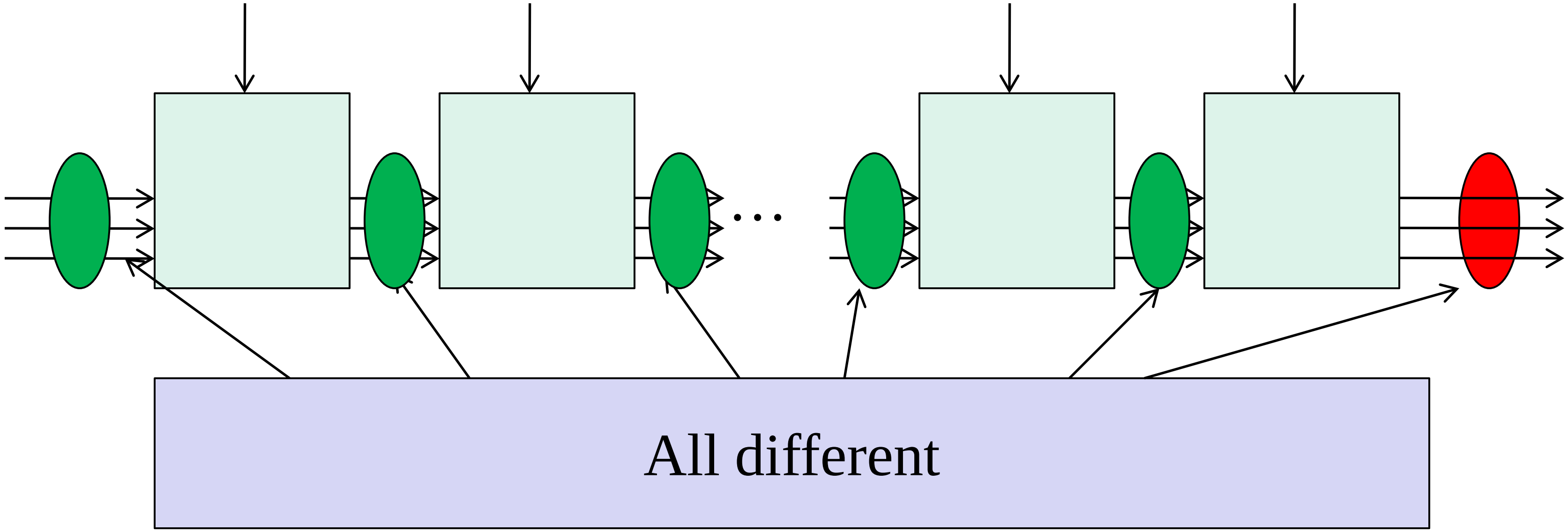
T $s_i) \wedge B(s_i)$



is UNSAT

Only interested in shortest paths

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



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

Define

$$\text{Base}_k = I(s_0) \wedge T^k(s_0, \ldots, s_k) \wedge B(s_k)$$

$$\text{Step1}_k = T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge \bigwedge_{0 \le j \le k} \neg B(s_j) \wedge B(s_{k+1})$$

Define

$$\mathrm{Base}_k = I(s_0) \wedge T^k(s_0, \ldots, s_k) \wedge B(s_k)$$

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

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

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

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

Define

$$\text{Base}_k = I(s_0) \wedge T^k(s_0, \ldots, s_k) \wedge B(s_k)$$

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

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

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

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

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

Temporal induction (Stålmarck)

```
i=0
while True do {
    if Sat(Base_i) return False    (and counter example)
    if Unsat(Step1_i)  or Unsat(Step2_i)   return True
    i=i+1
}
```

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, later slides)

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

Is it really induction?

To make this easier to see,   rewrite

$$\neg\ \mathrm{Base}_k \ = \neg\ (I(s_0)\ \wedge T^k(s_0,.., s_k) \wedge B(s_k\ ))$$

Let $P\ = \neg\ B$ (want to prove that P holds in all reachable states)

Rewrite as

$$(I(s_0)\ \wedge T^k(s_0, \ldots , s_k)\ )\ \Rightarrow\ P(s_k\ )$$

Is it really induction?

To make this easier to see, rewrite

$$\neg\,\mathrm{Base}_k \;=\; \neg\,(I(s_0)\;\wedge\; T^k(s_0,..,s_k)\;\wedge\quad B(s_k\,))$$

Let $P = \neg B$ (want to prove that P holds in all reachable states)

Rewrite as

Now add facts from previous iterations

$$P(\cancel{s_k}\,))$$

$$\bigwedge_{0 \le j \le k} P(s_j\,)$$

Is it really induction?

To make this easier to see, rewrite

$$\neg\, \mathrm{Base}_k \;=\; \neg\;\; (I(s_0) \wedge T^k(s_0,..,s_k) \wedge B(s_k\,))$$

Let $P = \neg\, B$ (want to prove that P holds in all reachable states)

Rewrite as

$$(I(s_0) \wedge T^k(s_0,\, .\, .\, .\, ,\, s_k)\,) \;\Rightarrow\; \bigwedge_{0 \le j \le k} P(s_j\,)$$

Is it really induction?

$$(I(s_0) \wedge T^k(s_0, \ldots, s_k)) \Rightarrow \bigwedge_{0 \leq j \leq k} P(s_j)$$

P holds in cycles 0 to k

Working with the strengthend Step1

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

$$\bigwedge_{0 \leq j \leq k} P(s_j) \wedge \neg\, P(s_{k+1})\,)$$

$$=$$

$$(T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge \bigwedge_{0 \leq j \leq k} P(s_j)$$

$$\Rightarrow \quad P(s_{k+1})$$

$$(T^{k+1}(s_0, \ldots, s_{k+1}) \wedge U^{k+1}(s_0, \ldots, s_{k+1}) \wedge \bigwedge_{0 \le j \le k} P(s_j))$$

$$\Rightarrow \quad P(s_{k+1})$$

If P holds in cycles 0 to k
then it also holds in the next cycle

Strengthened induction, depth k

$$(I(s_0) \wedge T^k(s_0, .., s_k)) \Rightarrow \bigwedge_{0 \leq j \leq k} P(s_j)$$

$$(T^{k+1}(s_0, .., s_{k+1}) \wedge U^{k+1}(s_0, .., s_{k+1}) \wedge \bigwedge_{0 \leq j \leq k} P(s_j)) \Rightarrow P(s_{k+1})$$

P holds in all reachable states

Strengthened induction, depth k

$(I(s_0) \wedge T^k(s_0 \ldots s_k)) \Rightarrow \ldots$

NO QUANTIFIERS

Can all be done with a SAT-solver

$(T^{k+1}(s_0 \ldots \quad \bigwedge_{0 \leq j \leq k} P(s_j)))$

P holds in all reachable states

induction, depth k

$$(I(s_0) \wedge T^k(s_0, .., s_k)) \Rightarrow \bigwedge_{0 \le j \le k} P(s_j)$$

$$(T^{k+1}(s_0, .., s_{k+1}) \wedge \bigwedge_{0 \le j \le k} P(s_j))$$

$$\Rightarrow \quad P(s_{k+1})$$

P holds in all reachable states

induction, depth k

$$(I(s_0) \wedge T^k(s_0, .., s_k)) \Rightarrow \bigwedge_{0 \le j \le k} P(s_j)$$

$(T^{k+1}(s_0, ..,$ $\bigwedge_{0 \le j \le k} P(s_j))$

$\Rightarrow$

is SOUND
conclusion is correct
if base and step proven

P holds in all reachable states

induction, depth k

$(I(s_0) \wedge T^k(s_0, .., s_k))$ => $\bigwedge_{0 \le j \le k} P(s_j)$

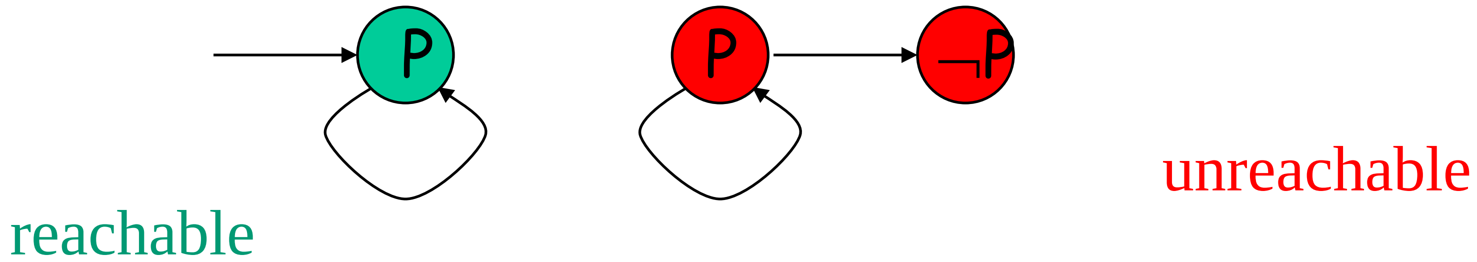
$(T^{k+1}(s_0, ..,$ $\bigwedge_{0 \le j \le k} P(s_j))$

=>

but NOT COMPLETE

P holds in all reachable states

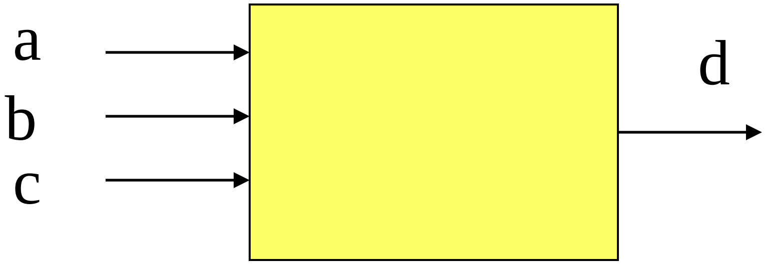
Some properties are not k-inductive no matter how big you make k



But there is a path from an initial to a bad state if and only if there is such a path without repeated states (loop-free, simple)

So Stålmarck's eureka step was vital and brilliant!

Symbolic Trajectory Evaluation (STE)



[a is v, _ , c is not v, _] ⟶ [_ , _ , _ , d is true]

antecedent

consequent

STE

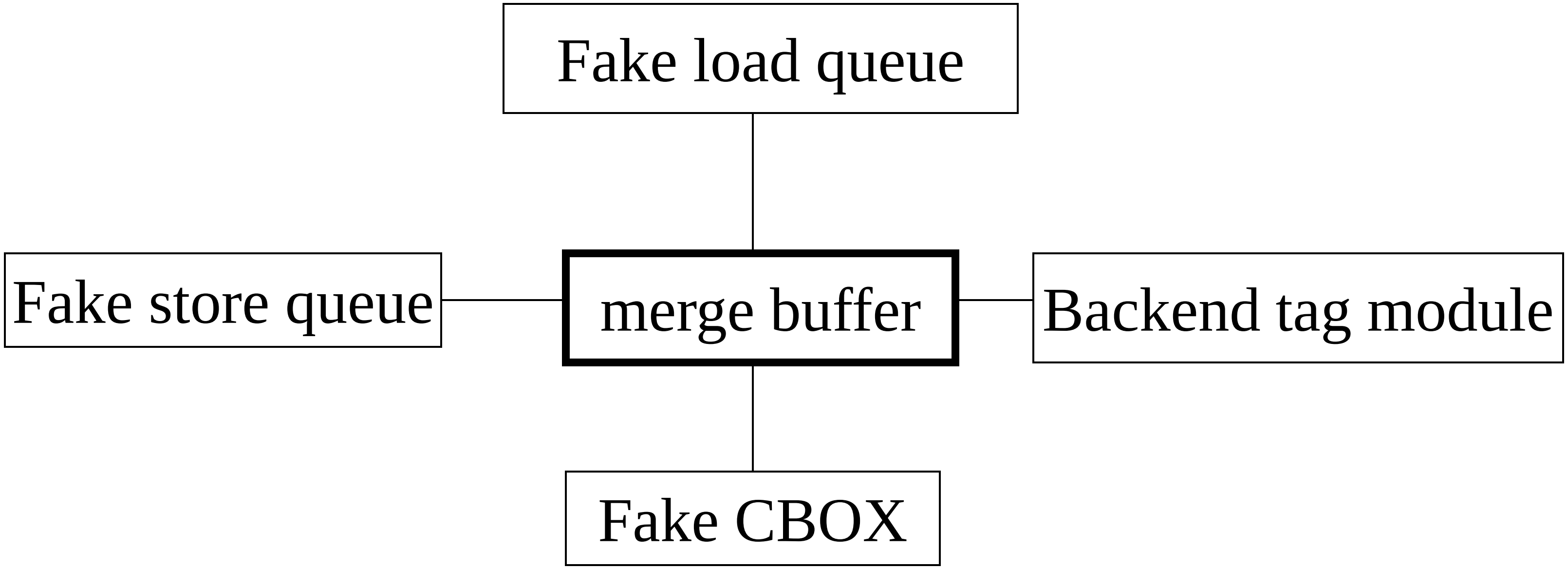
We already saw Symbolic Simulation.

Don’t just have concrete values (and X) flowing in the circuit. Have BDDs or formulas flowing

A single run of a symbolic simulator checks an STE property requiring many concrete simulations

STE is symbolic simulation plus proof that the consequent holds

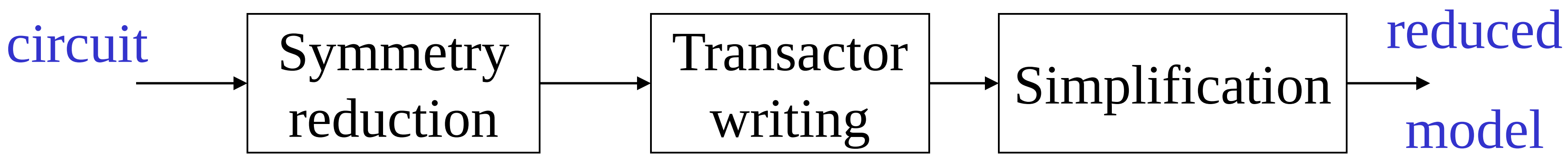
Use of BMC and STE in verifying the Alpha



Aim: to automatically find violations of properties like
Same address cannot be in two entries at once
that is, bug finding during development

Reducing the problem

- Initial circuit: 400 inputs, 14 400 latches, 15 pipeline stages



- Reduced model has 10 inputs, 600 latches

Results

- Real bugs found, from 25 -144 cycles
- SAT-based BMC on 32 bit PC 20 -10k secs.
- Custom SMV on 64 bit Alpha took much longer (but went to larger sizes)
- STE quick to run, but writing specs takes time and expertise
- Promising results in real development

NOTE: Done by Per Bjesse, who used to assist on this course ☺. (paper on links page)

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