SAT-based Model-Checking

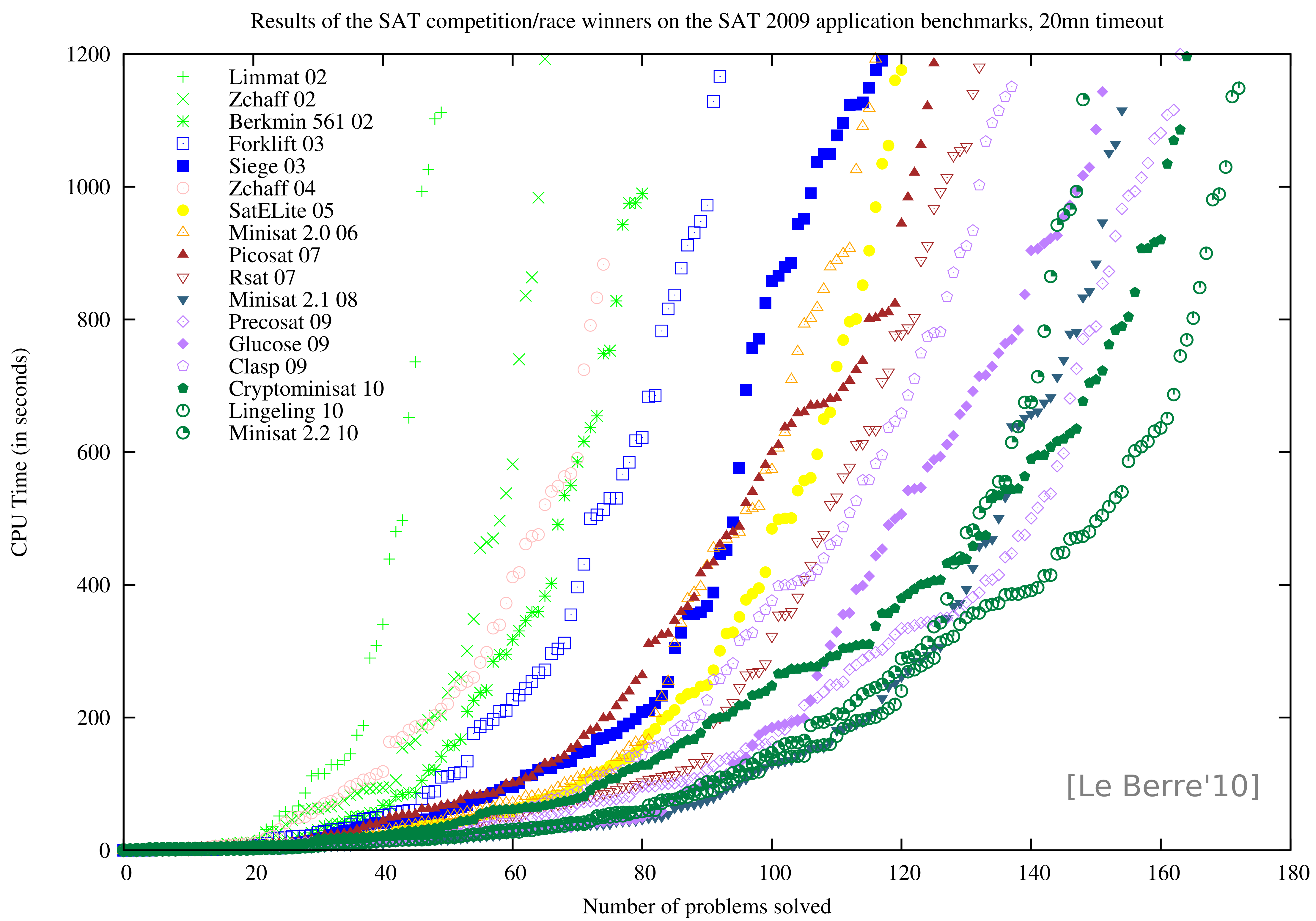
Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University
Linz, Austria

1st International SAT/SMT Summer School 2011

MIT, Cambridge, USA

Tuesday, June 14, 2011



Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout
CPU Time (in seconds)
Number of problems solved
Limmat 02
Zchaff 02
Berkmin 561 02
Forklift 03
Siege 03
Zchaff 04
SatELite 05
Minisat 2.0 06
Picosat 07
Rsat 07
Minisat 2.1 08
Precosat 09
Glucose 09
Clasp 09
Cryptominisat 10
Lingeling 10
Minisat 2.2 10
[Le Berre'10]
0
200
400
600
800
1000
1200
0
20
40
60
80
100
120
140
160
180

[ClarkeEmerson'82] [QuielleSifakis'82] Turing Award 2007

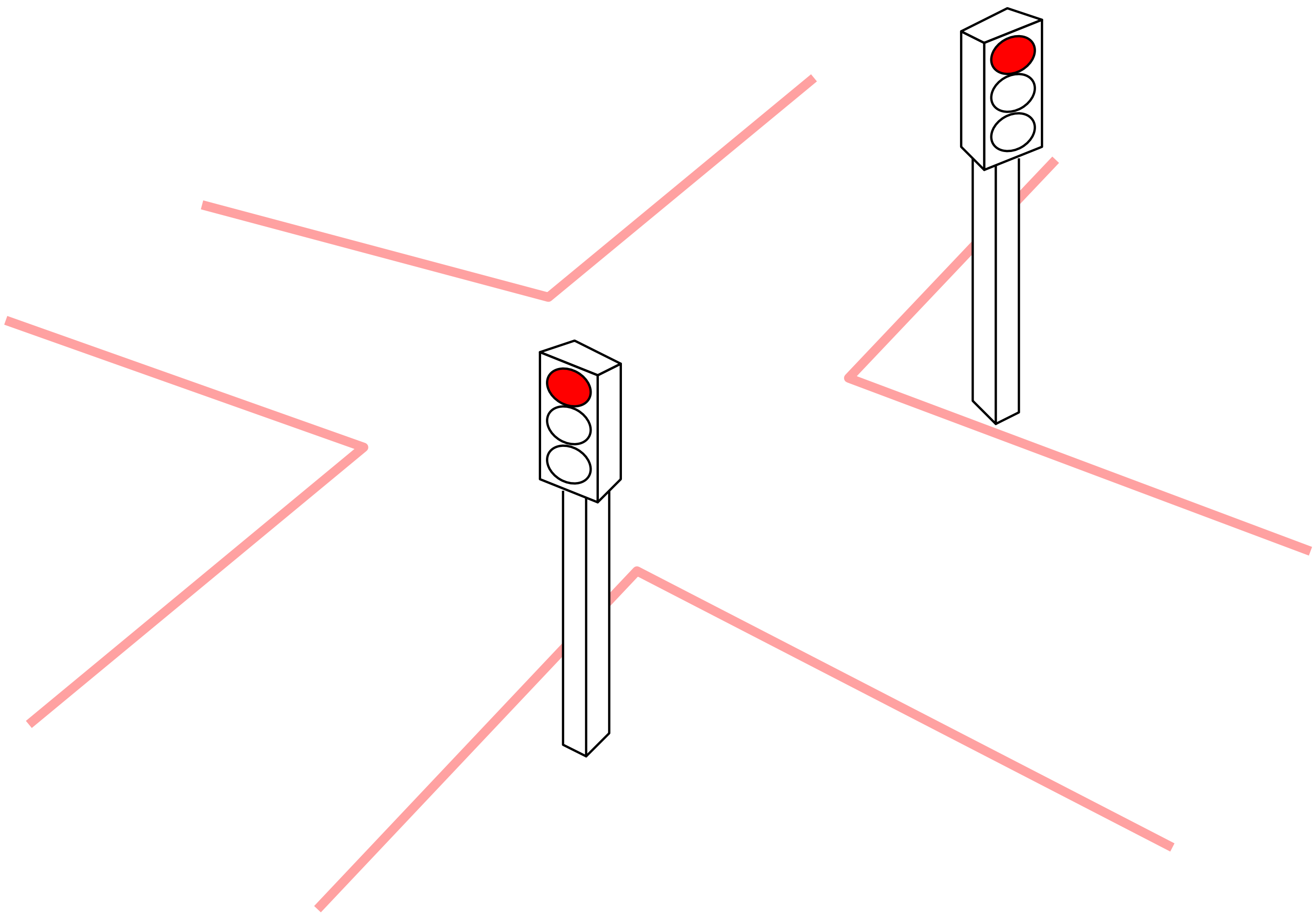
- check **algorithmically** temporal / sequential properties
  - systems are originally **finite state** e.g. circuits
  - simple model: finite state automaton
- **comparison** of automata can be seen as model checking
  - check that the output streams of two finite state systems "match"
  - process algebra: simulation and bisimulation checking
- **temporal logics** as specification mechanism: LTL, CTL PSL, SVA
  - safety, liveness and more general temporal operators, fairness

- fixpoint algorithms with symbolic representations:
  - software, cyber physical systems
  - termination guaranteed if finite quotient structure exists

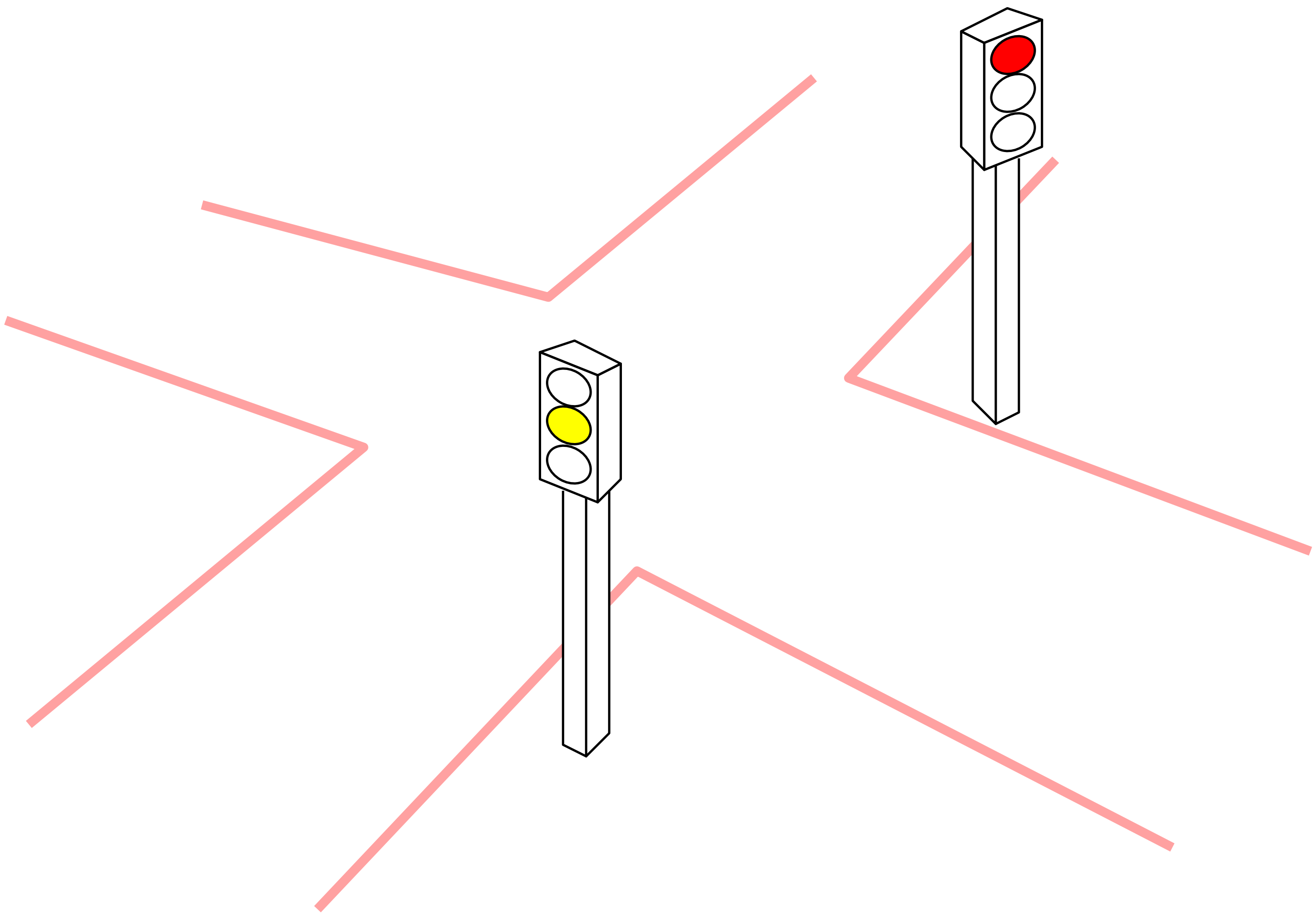
- key to success: abstract interpretation, e.g. predicate abstraction

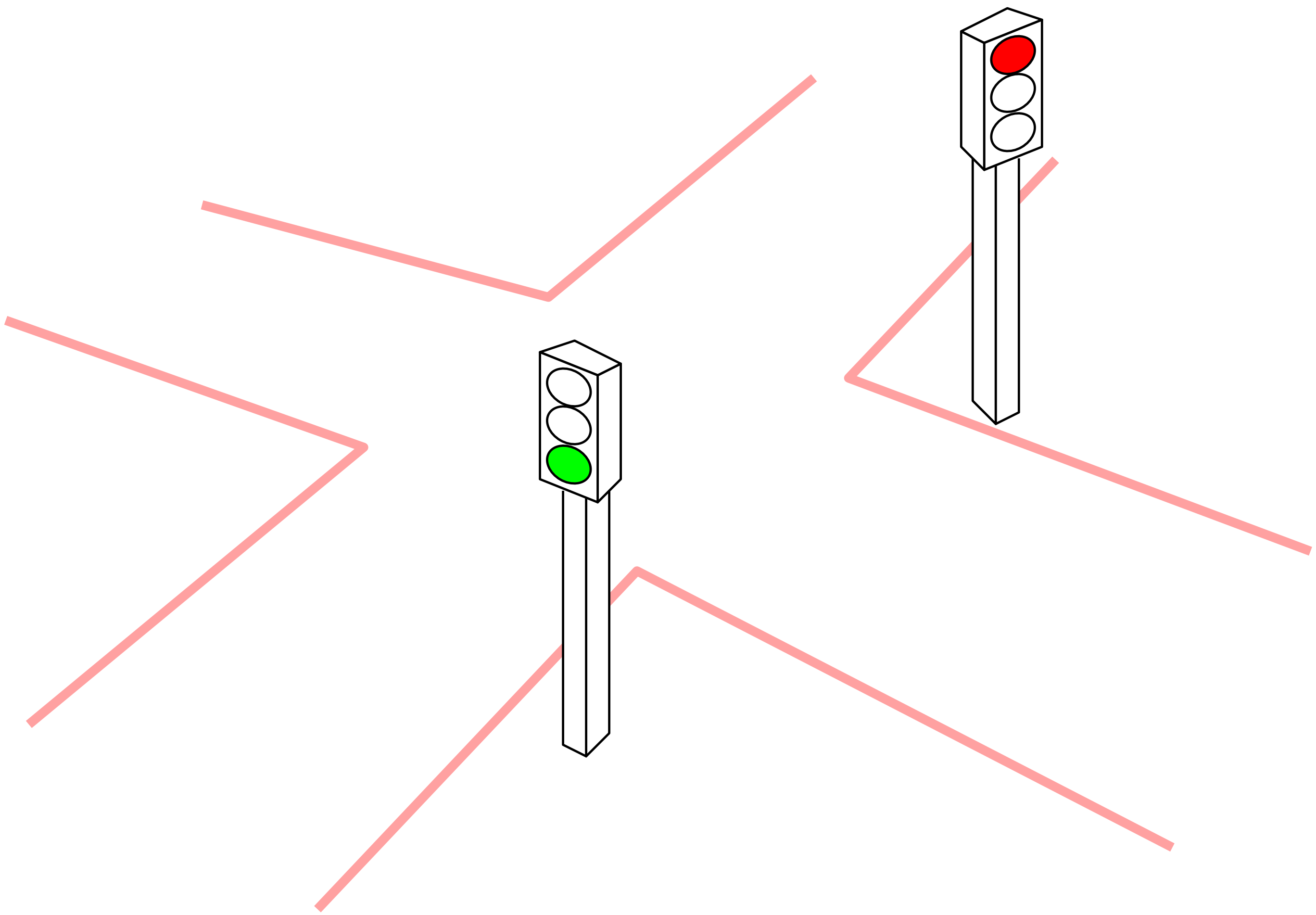
- otherwise drop completeness
  - simply run model checker *for some time*
  - hope that model checking algorithm still converges
  - trade completeness for scalability: bounded model checking
  - actually in practice: complexity is an issue not decidability

Traffic Light Controller (TLC)

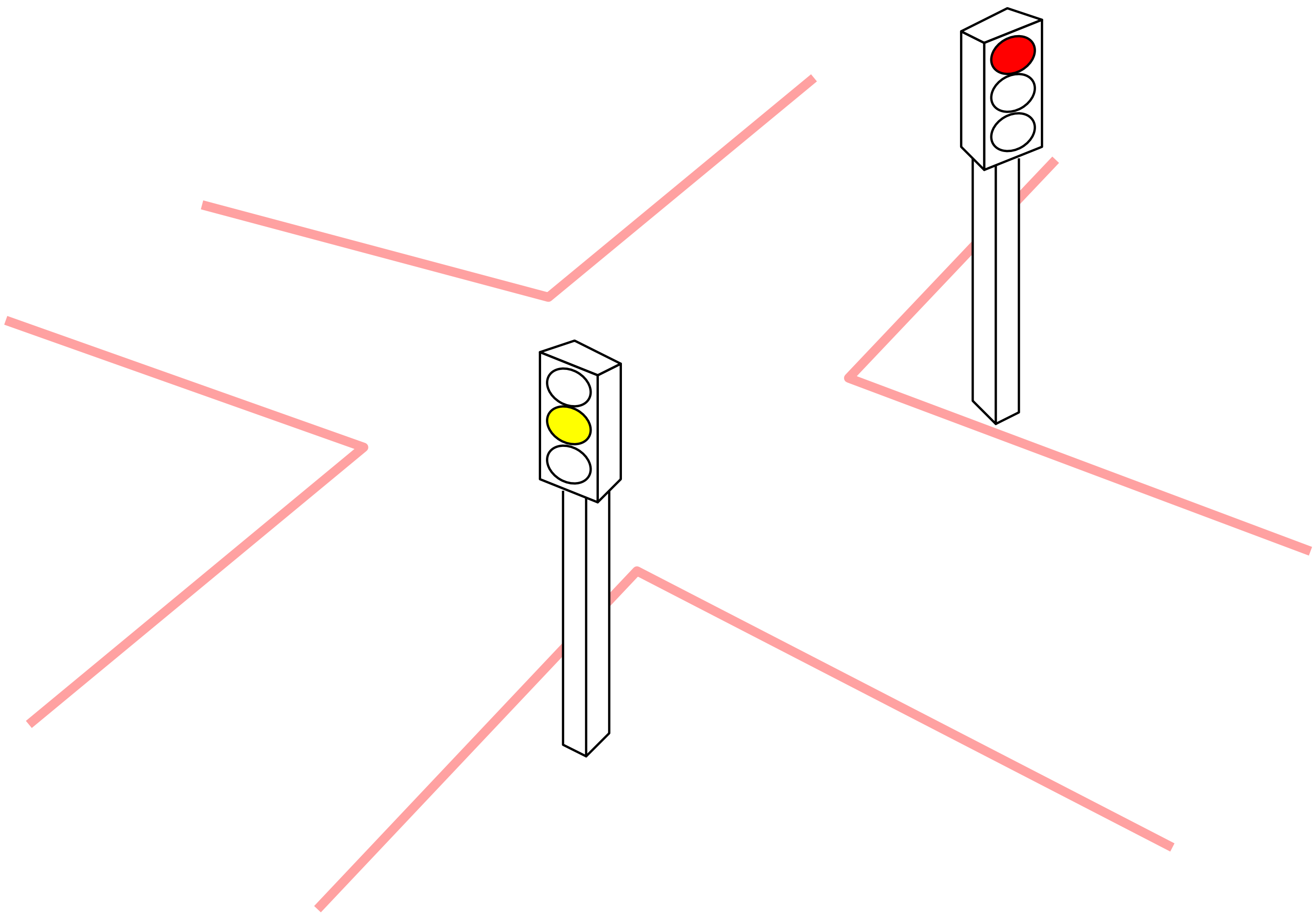


Traffic Light Controller (TLC)

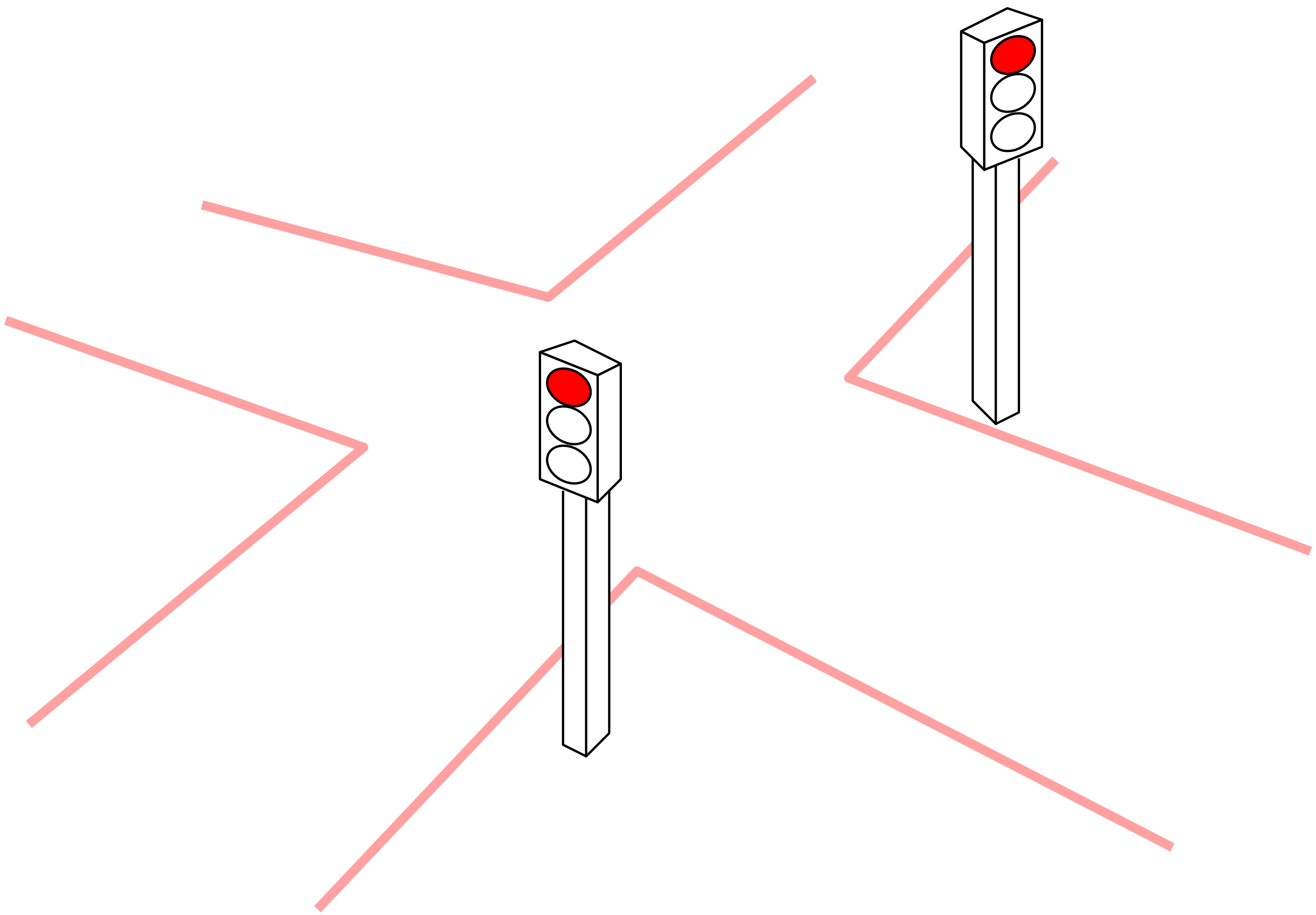




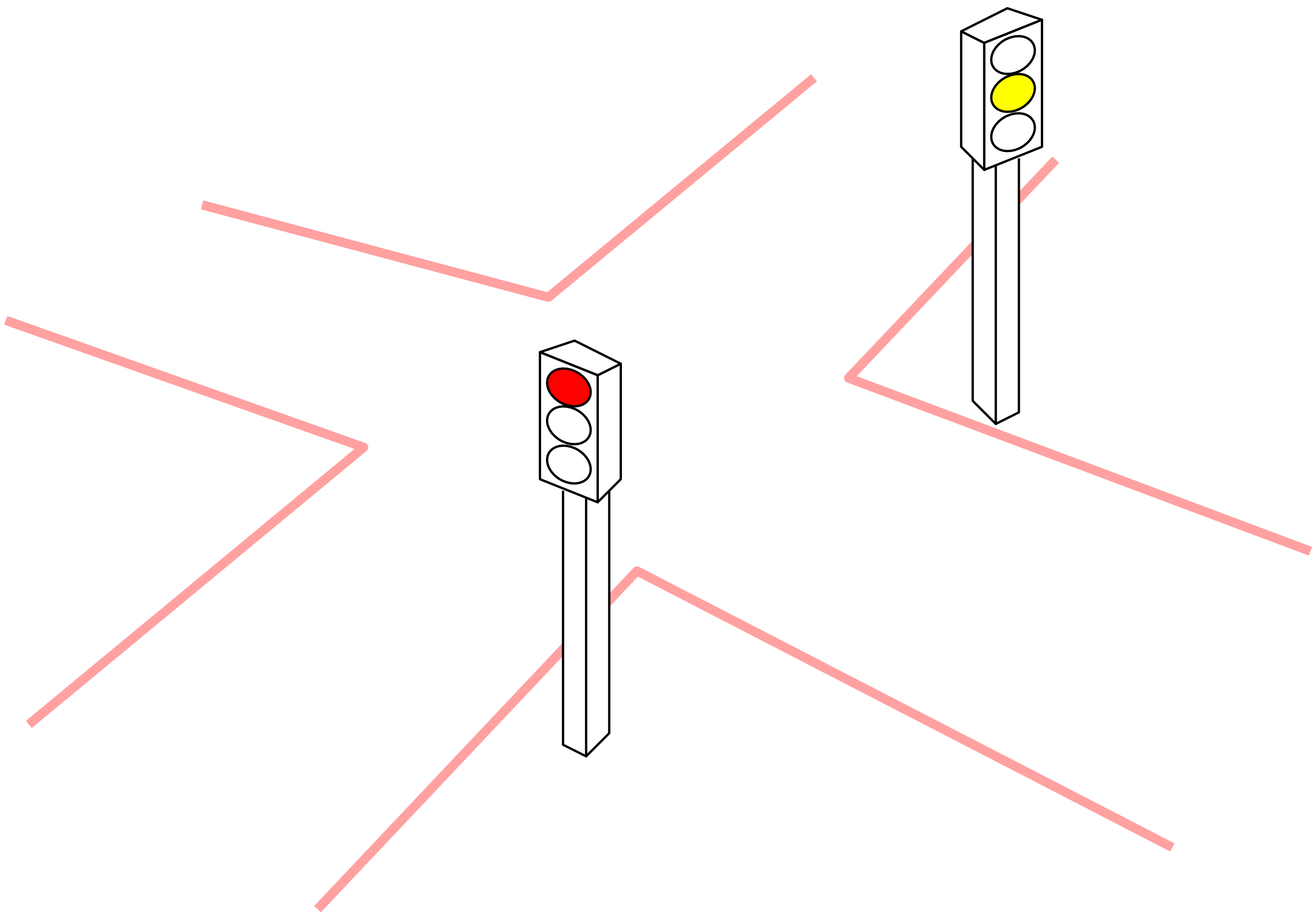
Traffic Light Controller (TLC)



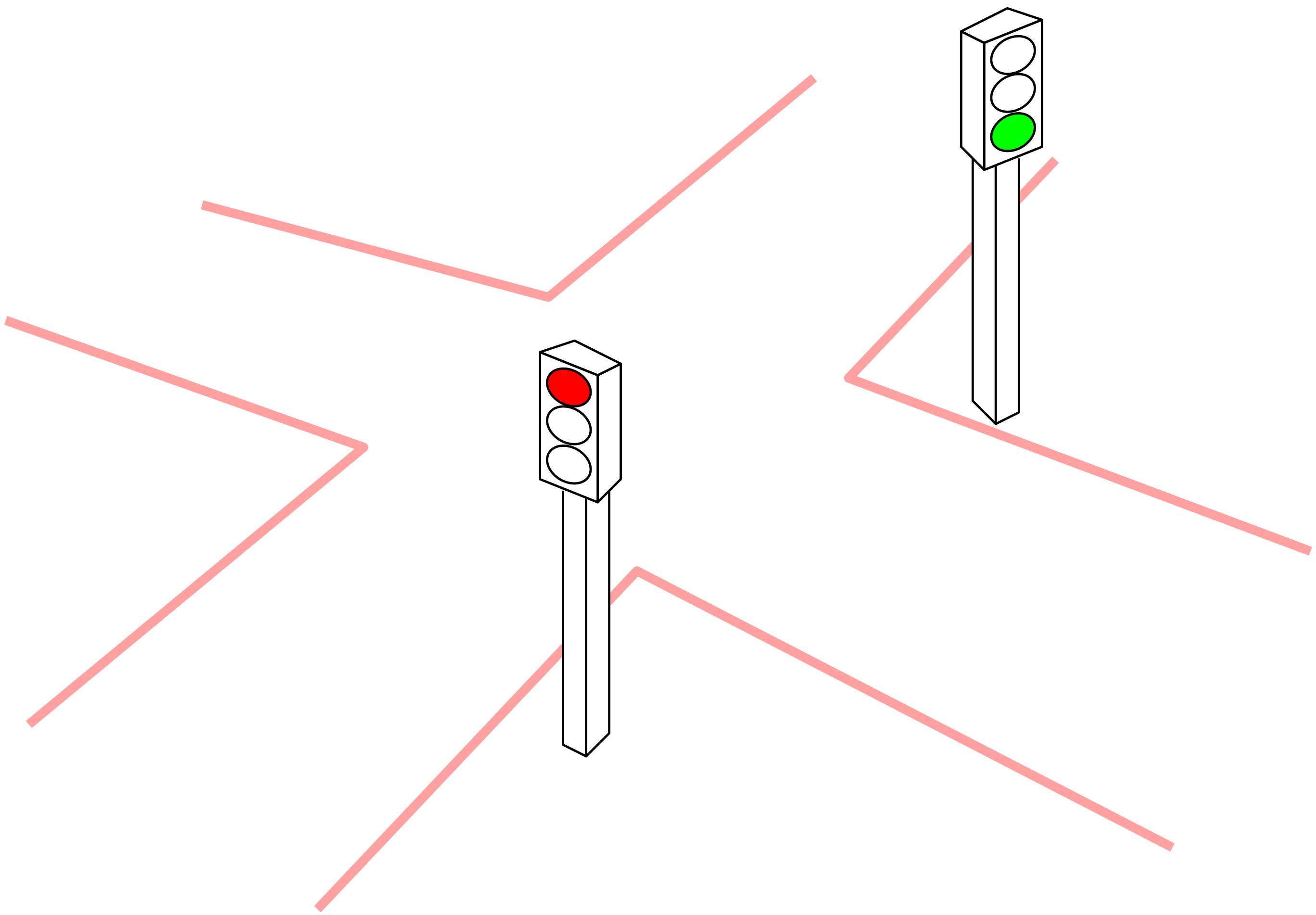
Traffic Light Controller (TLC)



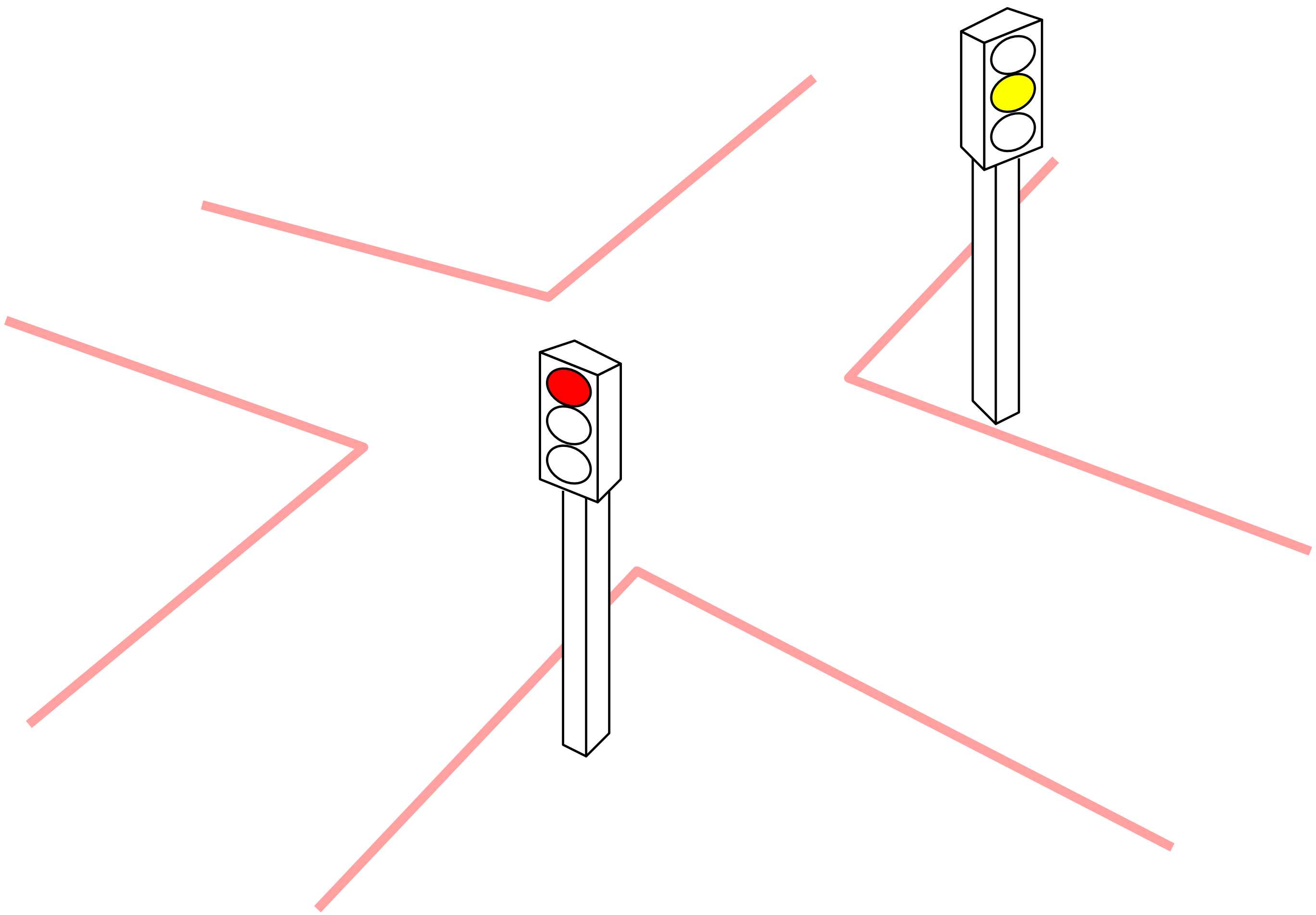
Traffic Light Controller (TLC)

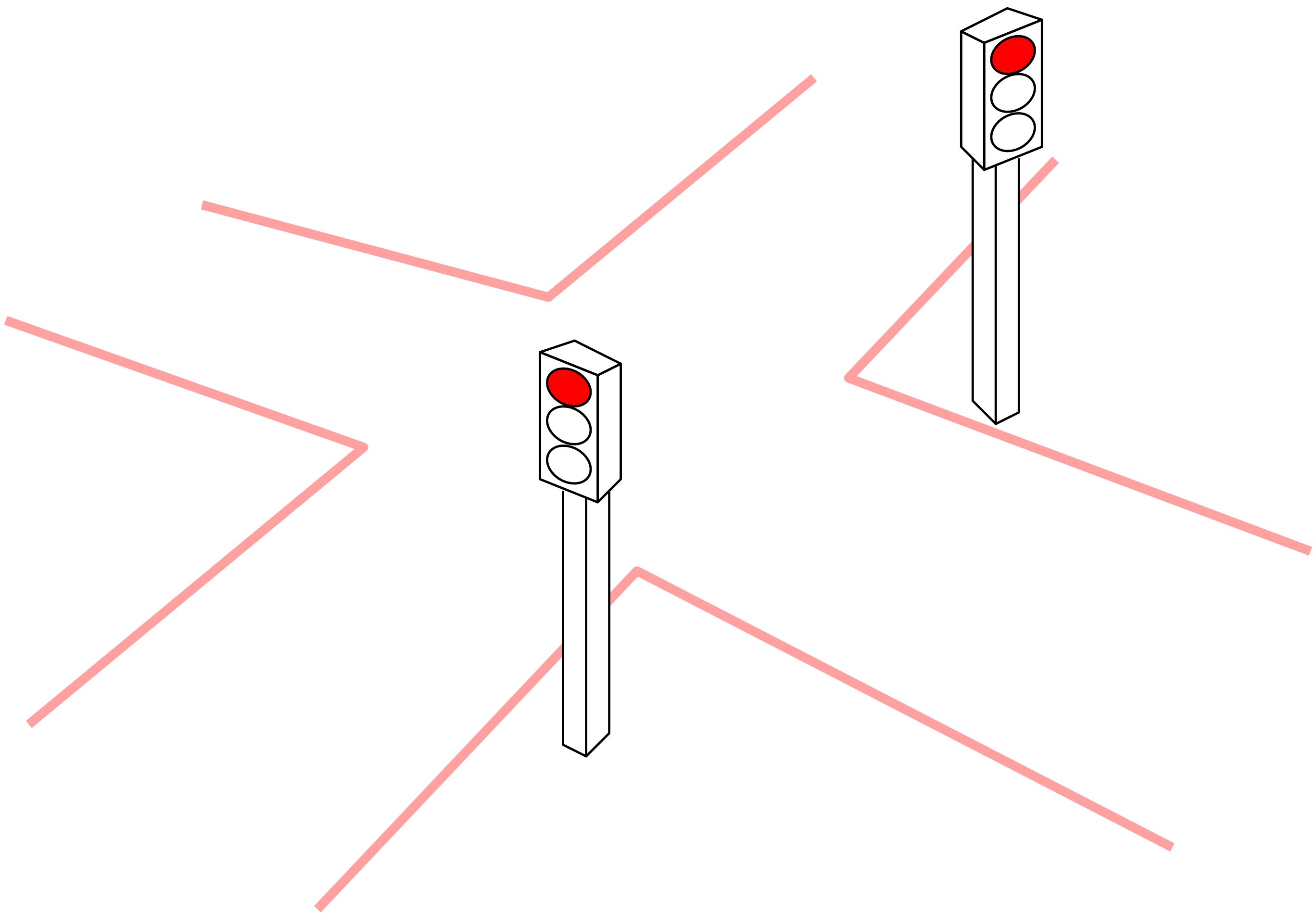


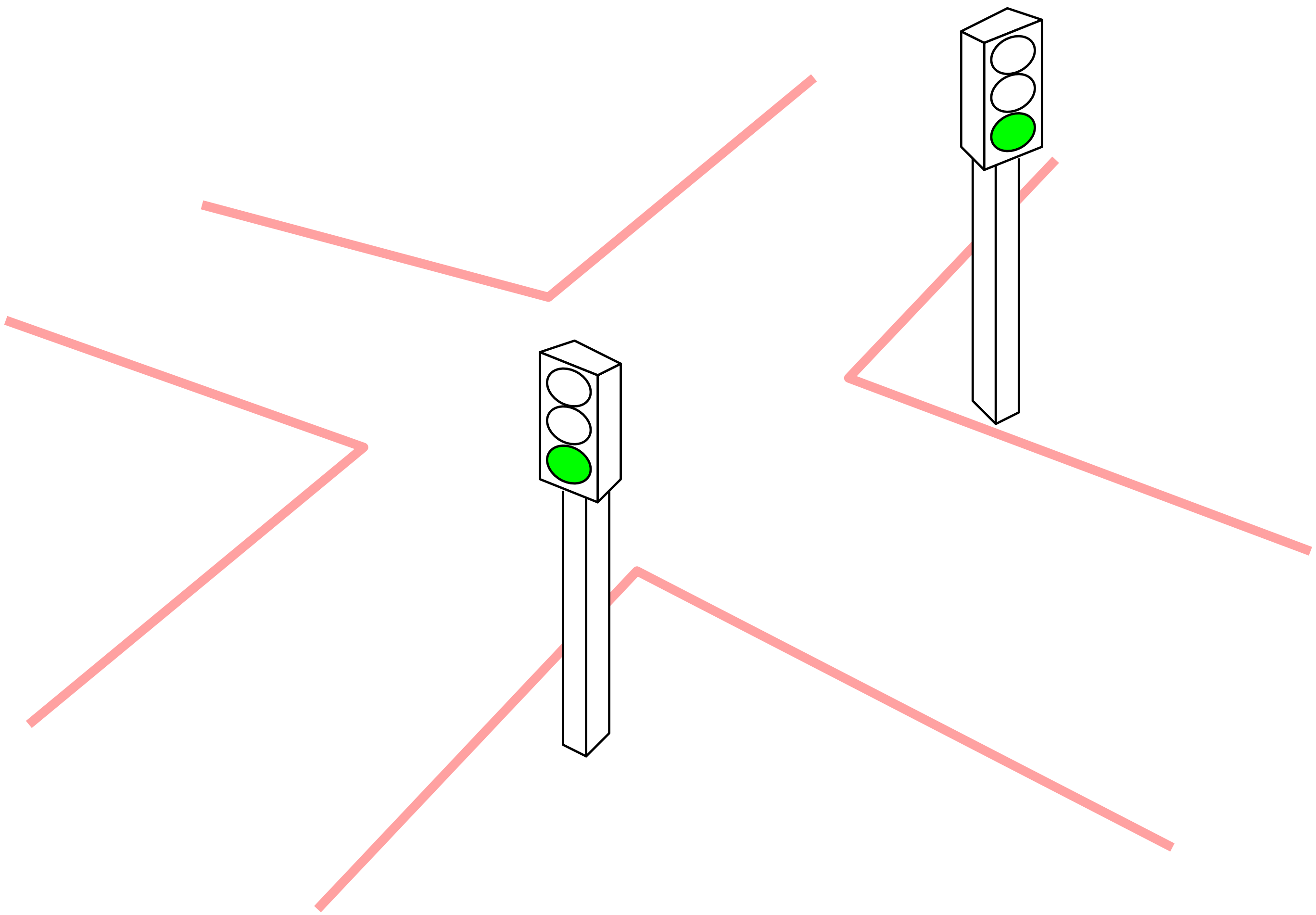
Traffic Light Controller (TLC)



Traffic Light Controller (TLC)







the two traffic lights should never show a green light at the same time

- state space is the set of assignments to variables of the system
  - state space is finite if the **range** of variables is finite
  - this notion works for inifinite state spaces as well
- TLC example:
  - single assignment $\sigma\colon \{\mathit{southnorth}, \mathit{eastwest}\} \rightarrow \{\mathit{green}, \mathit{yellow}, \mathit{red}\}$
  - set of assignments is isomorphic to $\{\mathit{green}, \mathit{yellow}, \mathit{red}\}^2$
  - eg state space is isomorphic to the crossproduct of variable ranges
- not all states are reachable: $(\mathit{green}, \mathit{green})$

- safety properties specify **invariants** of the system

- simple generic algorithm for checking safety properties:

  1. iteratively generate all reachable states

  2. check for violation of invariant for newly reached states

  3. terminate if all newly reached states can be found

- compare with **assertions**

  – used in run time checking: `assert` in C and VHDL

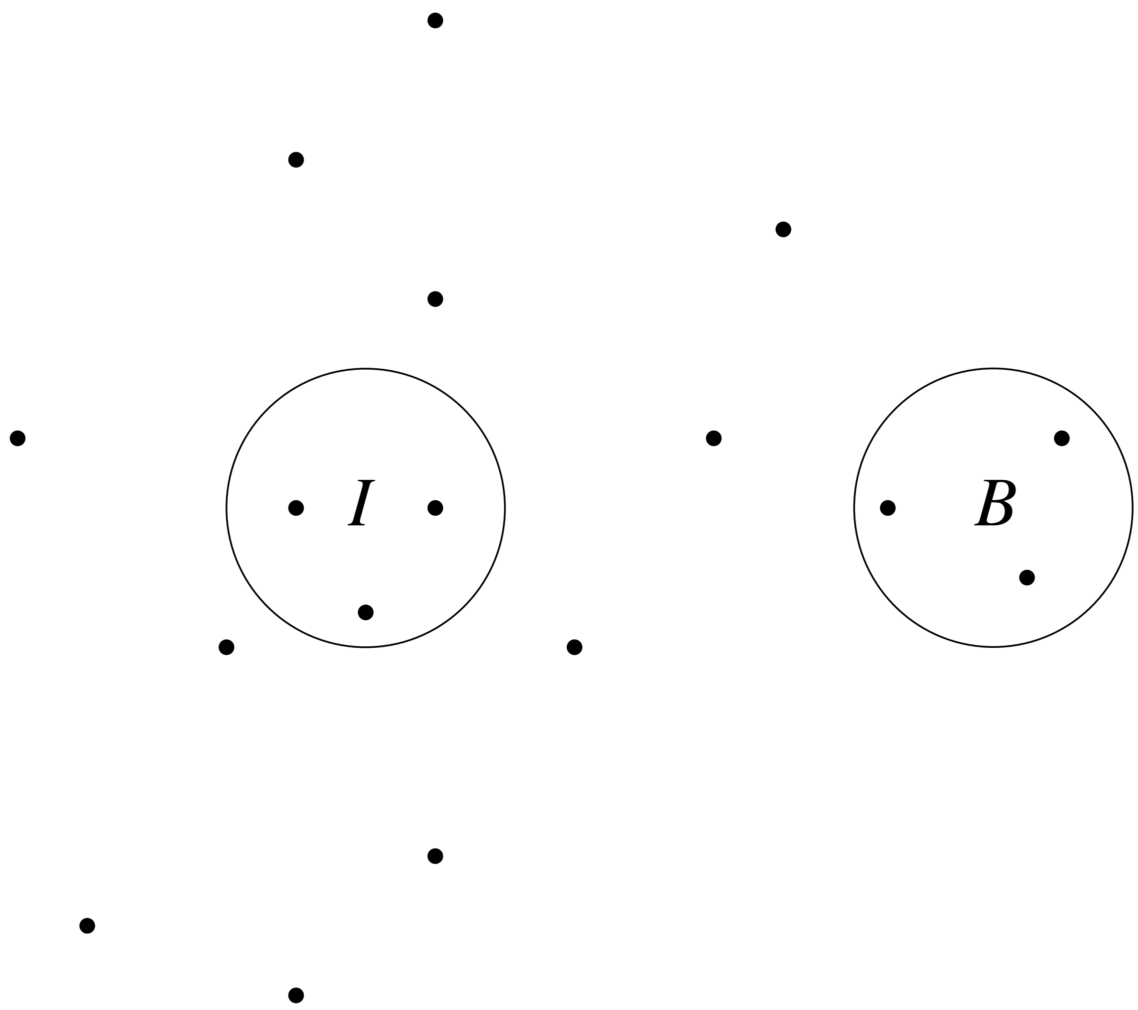
  – contract checking: `require`, `ensure`, etc. in Eiffel

- set of states S, initial states I, transition relation T

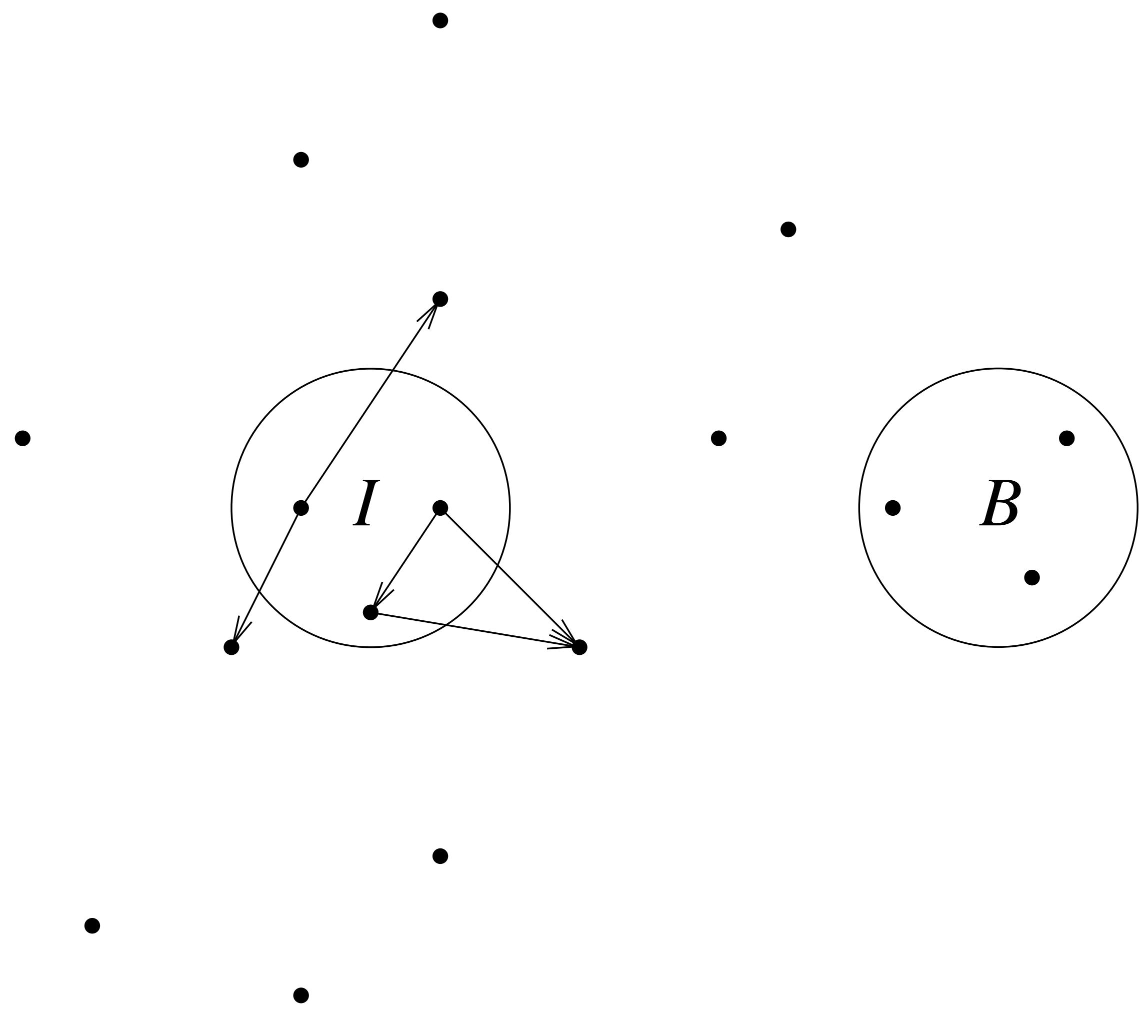
- bad states B reachable from I via T?

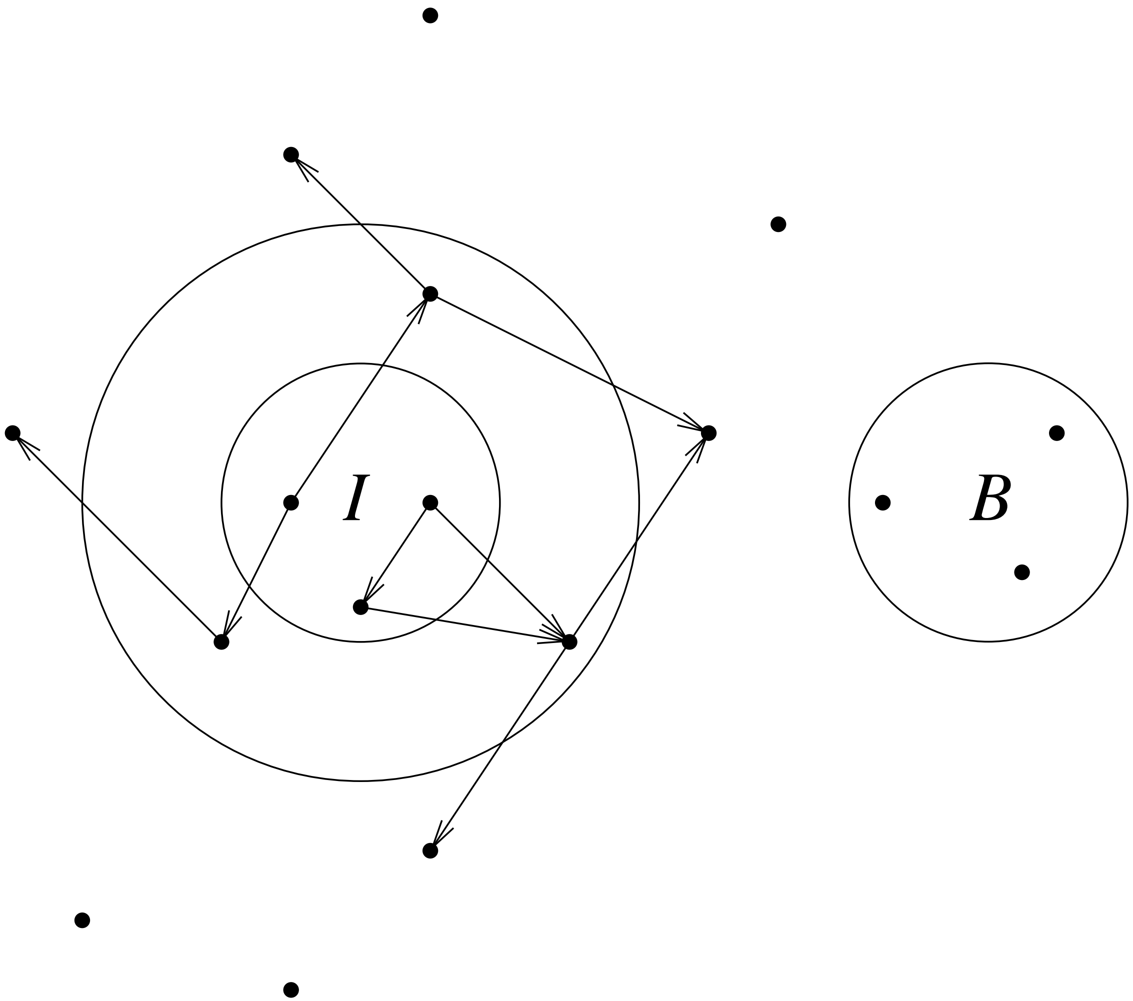
- symbolic representation of T (ciruit, program, parallel product)
  - avoid explicit matrix representations, because of the
  - state space explosion problem, e.g. n-bit counter: $|T| = O(n)$, $|S| = O(2^n)$
  - makes reachability PSPACE complete [Savitch'70]

- on-the-fly [Holzmann'81'] for protocols
  - restrict search to reachable states
  - simulate and hash reached concrete states


I
B

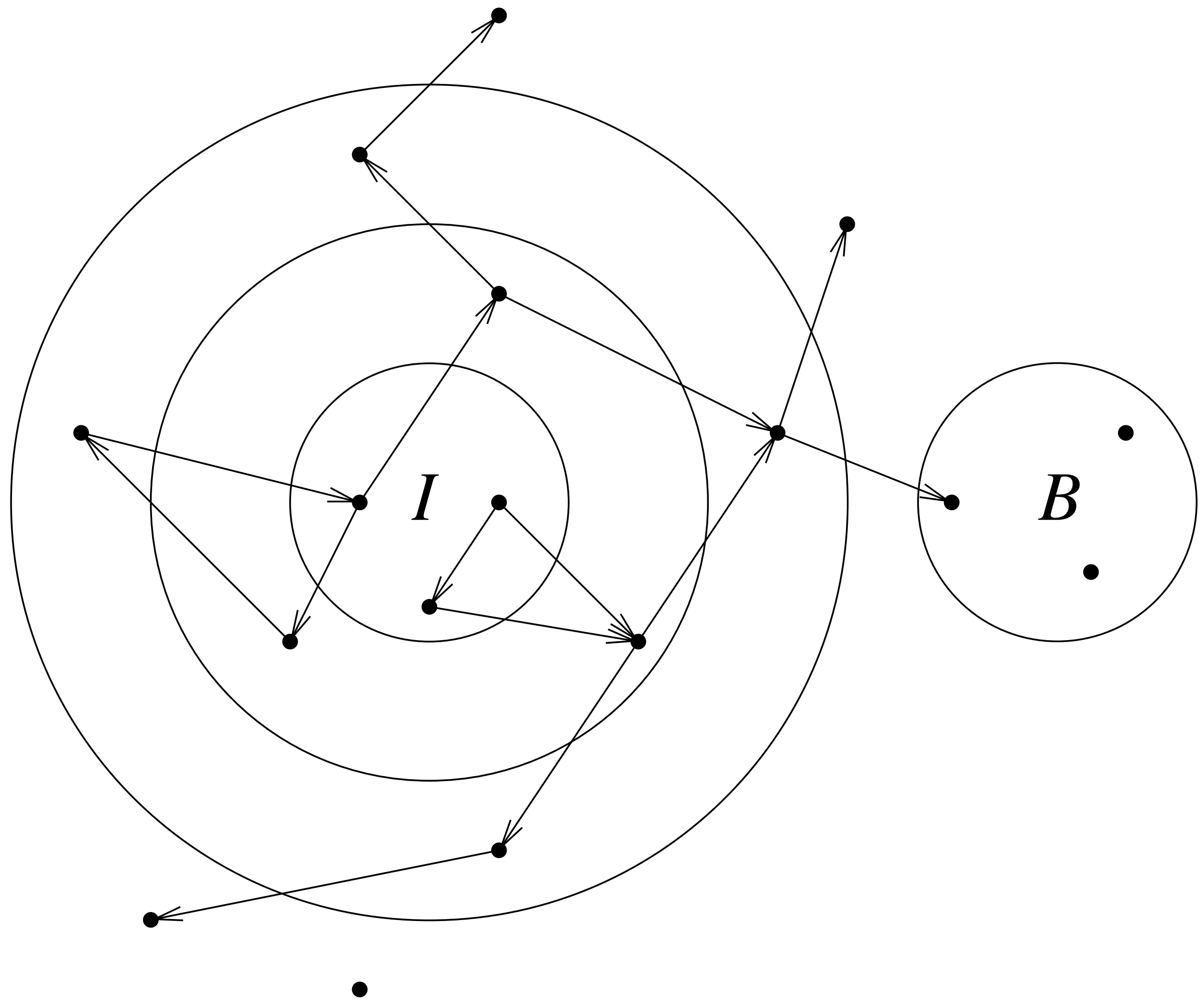
Forward Fixpoint: Step 1



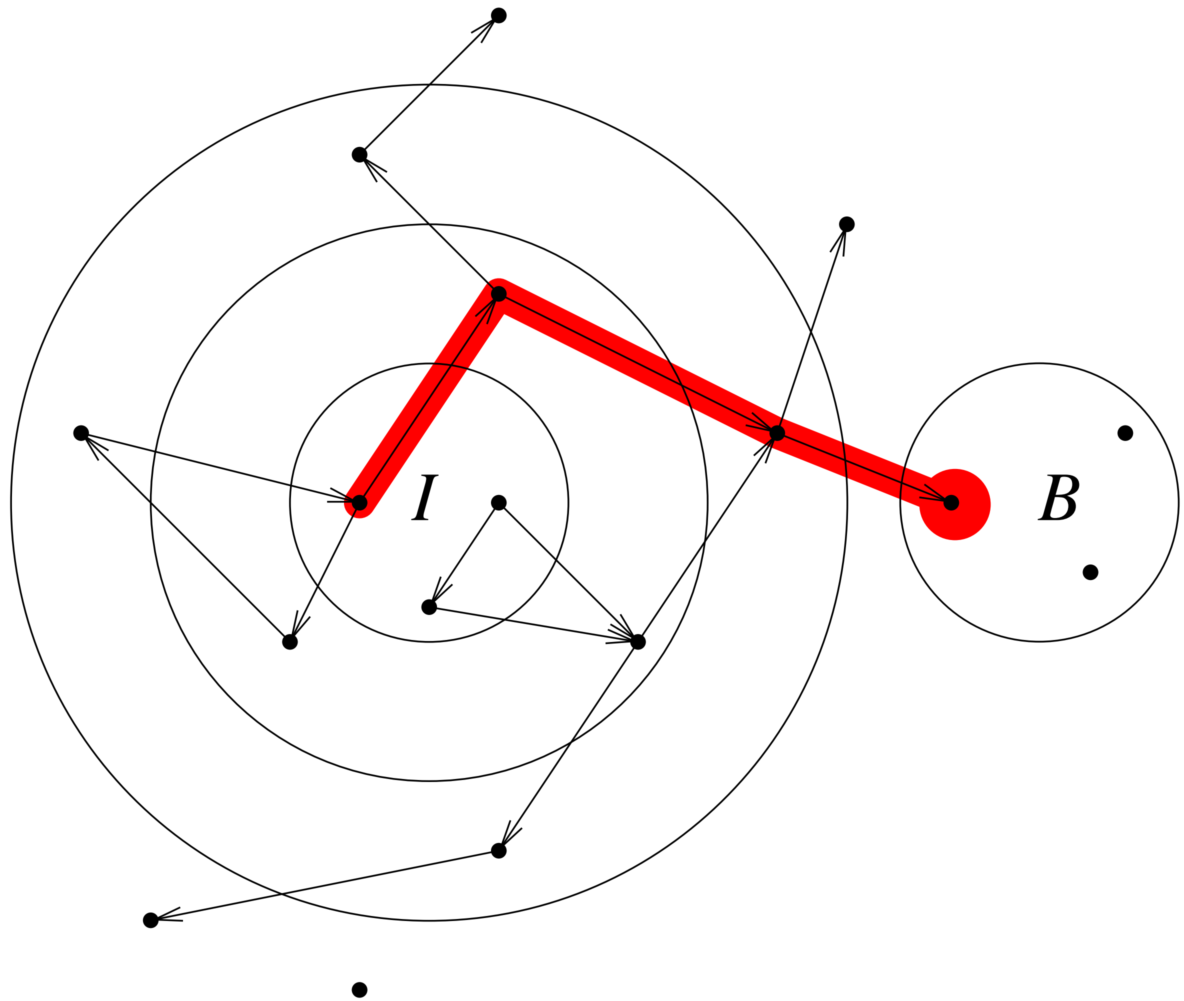


I
B

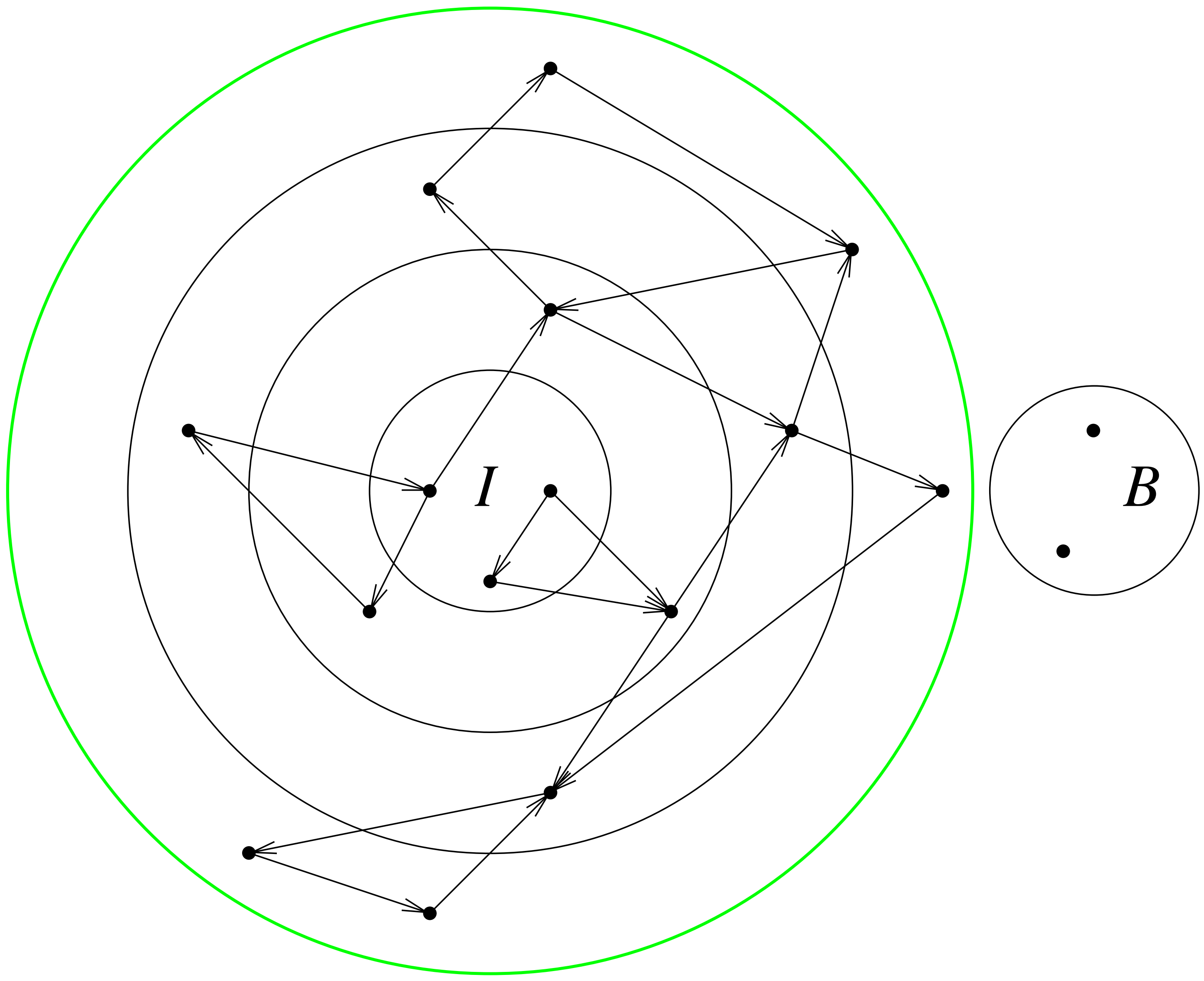
Forward Fixpoint: Step 3



Forward Fixpoint: Bad State Reached



Forward Fixpoint: Termination, No Bad State Reachable



initial states I, transition relation T, bad states B

<u>model-check</u>$^{\mu}_{\text{forward}}$ (I, T, B)

$S_C = \emptyset$; $S_N = I$;

while $S_C \neq S_N$ **do**

  if $B \cap S_N \neq \emptyset$ **then**

    return “found error trace to bad states”;

  $S_C = S_N$;

  $S_N = S_C \cup \boxed{Img(S_C)}$;

done;

return “no bad state reachable”;

```
MODULE trafficlight (enable)
VAR
  light : { green, yellow, red };
  back : boolean;
ASSIGN
  init (light) := red;
  next (light) :=
    case
      light = red & !enable : red;
      light = red & enable : yellow;
      light = yellow & back : red;
      light = yellow & !back : green;
      TRUE : yellow;
    esac;
  next (back) :=
    case
      light = red & enable : FALSE;
      light = green : TRUE;
      TRUE : back;
    esac;
MODULE main
VAR
  southnorth : trafficlight (TRUE);
  eastwest : trafficlight (TRUE);
SPEC
  AG !(southnorth.light = green & eastwest.light = green)
```

Counterexample

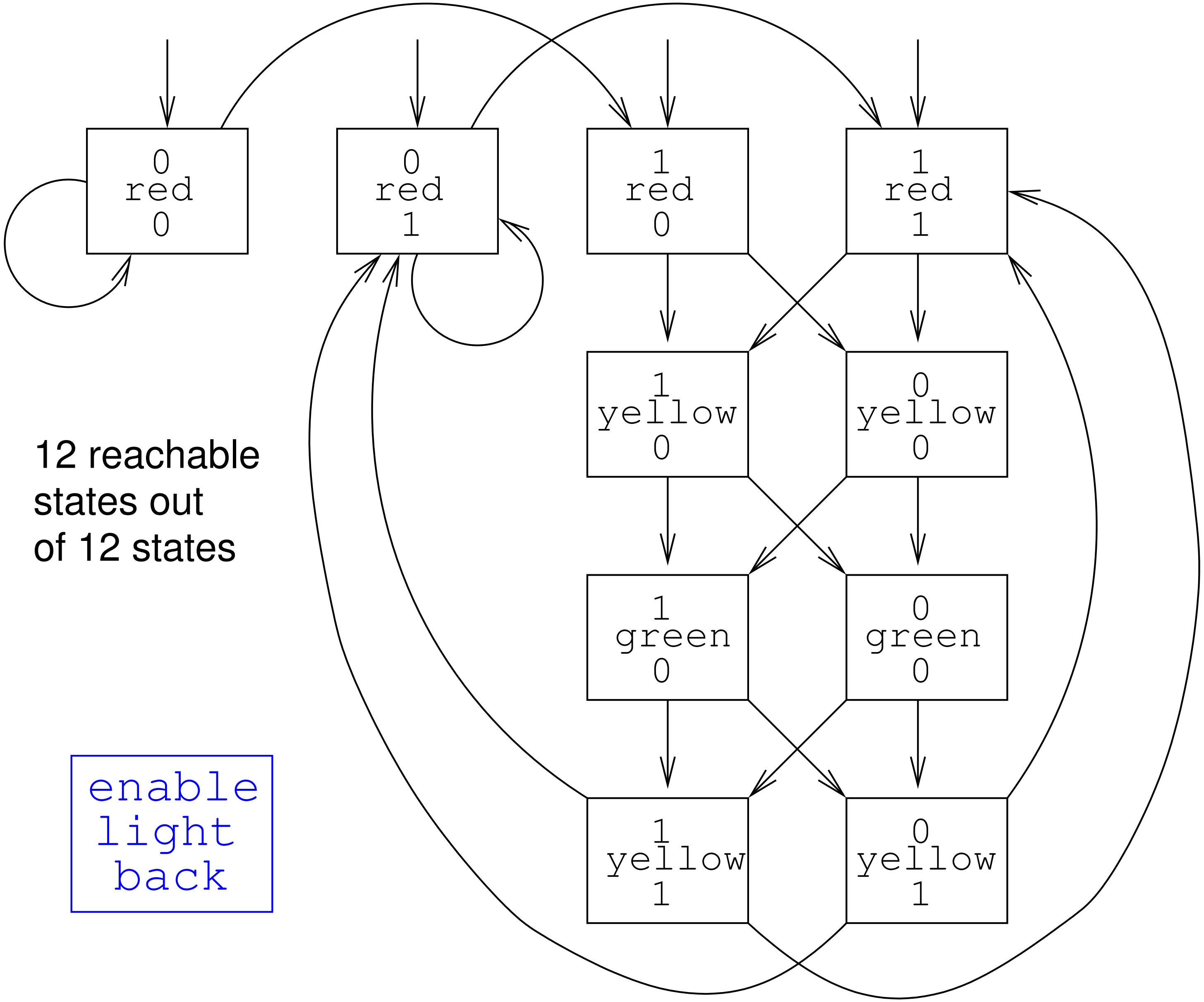
```
*** This is NuSMV 2.5.2 (compiled on Mon May 30 11:42:23 UTC 2011)
*** Copyright (c) 2010, Fondazione Bruno Kessler

-- specification AG !(southnorth.light = green & eastwest.light = green)  is false
-- as demonstrated by the following execution sequence

Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  enablesouthnorth = FALSE
  enableeastwest = FALSE
  southnorth.light = red
  southnorth.back = FALSE
  eastwest.light = red
  eastwest.back = FALSE
-> State: 1.2 <-
  enablesouthnorth = TRUE
  enableeastwest = TRUE
-> State: 1.3 <-
  enablesouthnorth = FALSE
  enableeastwest = FALSE
  southnorth.light = yellow
  eastwest.light = yellow
-> State: 1.4 <-
  southnorth.light = green
  eastwest.light = green
```

- symbolic model checker implemented by Ken McMillan at CMU (early 90'ies)

- input language: finite models + temporal specification (CTL + fairness)
  - hierarchical description, similar to hardware description language (HDL)
  - integer and enumeration types, arithmetic operations

- original version relies on Binary Decision Diagrams (BDDs)

- NuSMV an up-to-date version from FBK, Trento
  - also uses SAT/SMT technology
  - additionally LTL



12 reachable states out of 12 states

enable light back

- compilation of finite model into pure propositional domain — like HW synthesis

- first step is to **flatten** the hierarchy
  - recursive instantiation of all submodules
  - name and parameter substitution
  - may increase program size exponentially

- second step is to **encode** variables with **boolean** variables

light		light@1	light@0
green	$\mapsto$	0	0
yellow	$\mapsto$	0	1
red	$\mapsto$	1	0

logarithmic/binary encoding

```
MODULE main
VAR
  enablesouthnorth : boolean;
  enableeastwest : boolean;
  southnorth.light : {green, red, yellow};
  southnorth.back : boolean;
  eastwest.light : {green, red, yellow};
  eastwest.back : boolean;
ASSIGN
  init(southnorth.light) := red;
  next(southnorth.light) :=
    case
      southnorth.light = red & !enablesouthnorth : red;
      southnorth.light = red & enablesouthnorth : yellow;
      southnorth.light = yellow & southnorth.back : red;
      southnorth.light = yellow & !southnorth.back : green;
      1 : yellow;
    esac;
  next(southnorth.back) :=
    case
      southnorth.light = red & enablesouthnorth : 0;
      southnorth.light = green : 1;
      1 : southnorth.back;
    esac;
  init(eastwest.light) := red;
  next(eastwest.light) :=
    case
      eastwest.light = red & !enableeastwest : red;
      eastwest.light = red & enableeastwest : yellow;
      eastwest.light = yellow & eastwest.back : red;
      eastwest.light = yellow & !eastwest.back : green;
      1 : yellow;
    esac;
  next(eastwest.back) :=
    case
      eastwest.light = red & enableeastwest : 0;
      eastwest.light = green : 1;
      1 : eastwest.back;
    esac;
SPEC
  AG !(southnorth.light = green & eastwest.light = green)
```

- initial state predicate I represented as boolean formula

```
!eastwest.light@0 & eastwest.light@1
```

(equivalent to `init(eastwest.light) := red`)

- transition relation T represented as boolean formula

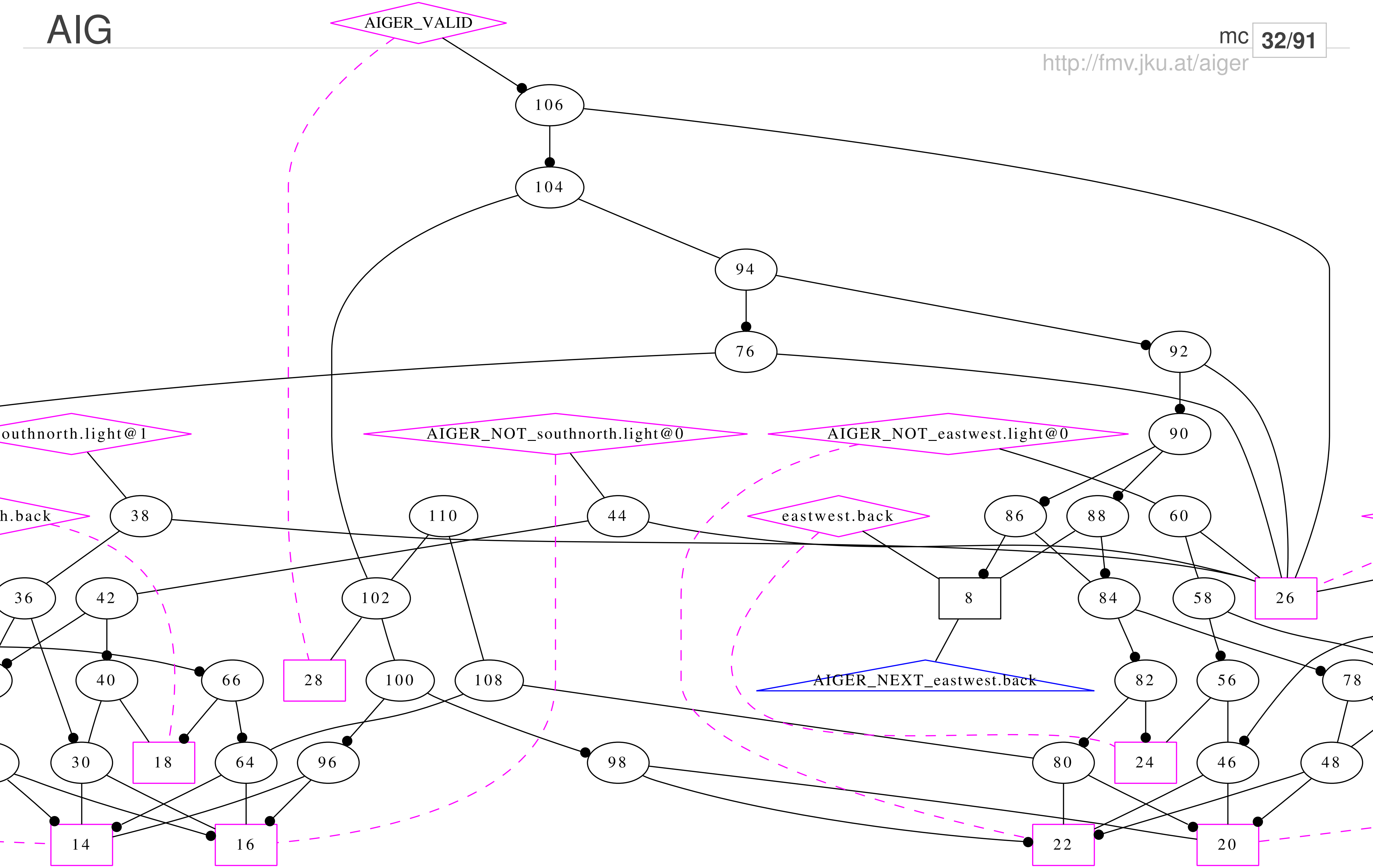
- encoding of atomic predicates p as boolean formulae

```
!eastwest.light@1 & !eastwest.light@0
```

(equivalent to `eastwest.light != green`)

```
MODULE main
VAR
  enablesouthnorth : boolean;
  enableeastwest : boolean;
  southnorth.light@1 : boolean; --TYPE-- green red yellow
  southnorth.light@0 : boolean;
  southnorth.back : boolean;
  eastwest.light@1 : boolean; --TYPE-- green red yellow
  eastwest.light@0 : boolean;
  eastwest.back : boolean;
DEFINE
  .MACRO1 := southnorth.light@1 | !southnorth.light@0;
  .MACRO0 := enablesouthnorth | .MACRO1;
  .MACRO2 := !southnorth.light@1 | southnorth.light@0;
  .MACRO3 := !southnorth.light@1 & !southnorth.light@0;
  .MACRO5 := eastwest.light@1 | !eastwest.light@0;
  .MACRO4 := enableeastwest | .MACRO5;
  .MACRO6 := !eastwest.light@1 | eastwest.light@0;
  .MACRO7 := !eastwest.light@1 & !eastwest.light@0;
ASSIGN
  init(southnorth.light@1) := FALSE;
  init(southnorth.light@0) := TRUE;
  next(southnorth.light@1) := .MACRO0 & .MACRO2;
  next(southnorth.light@0) := !.MACRO0 | southnorth.back & !.MACRO2;
  next(southnorth.back) := (!enablesouthnorth | .MACRO1) & (southnorth.back | .MACRO3);
  init(eastwest.light@1) := FALSE;
  init(eastwest.light@0) := TRUE;
  next(eastwest.light@1) := .MACRO4 & .MACRO6;
  next(eastwest.light@0) := !.MACRO4 | eastwest.back & !.MACRO6;
  next(eastwest.back) := (!enableeastwest | .MACRO5) & (eastwest.back | .MACRO7);
INVAR
  (!southnorth.light@1 | !southnorth.light@0) &
  (!eastwest.light@1 | !eastwest.light@0)
SPEC
  AG (!.MACRO3 | !.MACRO7)
```

AIG


AIGER_VALID
106
104
94
76
92
90
outhnorth.light@1
AIGER_NOT_southnorth.light@0
AIGER_NOT_eastwest.light@0
h.back
38
110
44
eastwest.back
86
88
60
36
42
102
8
84
58
26
40
66
28
100
108
AIGER_NEXT_eastwest.back
82
56
78
30
18
64
96
98
80
24
46
48
14
16
22
20

[BiereCimattiClarkeZhu-TACAS'99]

- uses SAT for model checking
  - historically not the first *symbolic model checking* approach
  - scales better than original BDD based techniques
    [CouterBerthetMadre'89] [BurchClarkeMcMillanDillHwang'90] [McMillan'93]
- mostly incomplete in practice
  - validity of a formula can often not be proven
  - focus on counter example generation
  - only counter example up to certain length (the bound k) are searched

0: terminate? $S_C^0 = S_N^0$ $\forall s_0[\neg I(s_0)]$

0: bad state? $B \cap S_N^0 \neq \emptyset$ $\exists s_0[I(s_0) \wedge B(s_0)]$

1: terminate? $S_C^1 = S_N^1$ $\forall s_0, s_1[I(s_0) \wedge T(s_0, s_1) \rightarrow I(s_1)]$

1: bad state? $B \cap S_N^1 \neq \emptyset$ $\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge B(s_1)]$

2: terminate? $S_C^2 = S_N^2$ $\forall s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow$
$I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]$

2: bad state? $B \cap S_N^1 \neq \emptyset$ $\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge B(s_2)]$

Falsification Part of Fixpoint Algorithm

[BiereCimattiClarkeZhu-TACAS'99]

~~0: terminate?~~ $\cancel{S_C^0 = S_N^0}$ $\cancel{\forall s_0[\neg I(s_0)]}$

0: bad state? $B \cap S_N^0 \neq \emptyset$ $\boxed{\exists s_0[I(s_0) \wedge B(s_0)]}$

~~1: terminate?~~ $\cancel{S_C^1 = S_N^1}$ $\cancel{\forall s_0, s_1[I(s_0) \wedge T(s_0, s_1) \rightarrow I(s_1)]}$

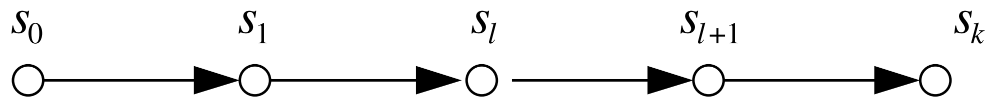
1: bad state? $B \cap S_N^1 \neq \emptyset$ $\boxed{\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge B(s_1)]}$

~~2: terminate?~~ $\cancel{S_C^2 = S_N^2}$ $\cancel{\forall s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow}$
$\cancel{I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]}$

2: bad state? $B \cap S_N^1 \neq \emptyset$ $\boxed{\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge B(s_2)]}$

[BiereCimattiClarkeZhu-TACAS'99]

checking safety property $\mathbf{G}p$ for a bound k as SAT problem:



$$B \vee B \vee B \vee B \vee B$$

$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \bigvee_{i=0}^{k} B(s_i)$$

check occurrence of B in the first k states with $B \equiv \neg p$

$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge B(s_k)$$

in incremental check only last state can be bad

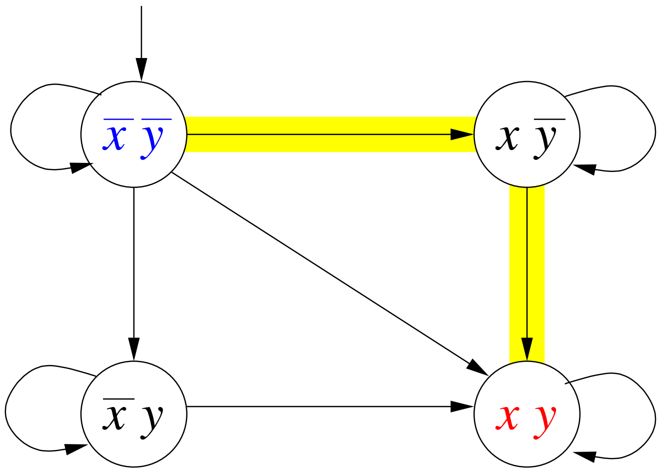
Checking Simple Saftey Properties with BMC

[BiereCimattiClarkeZhu-TACAS'99]

$$I \equiv \bar{x}\bar{y}$$

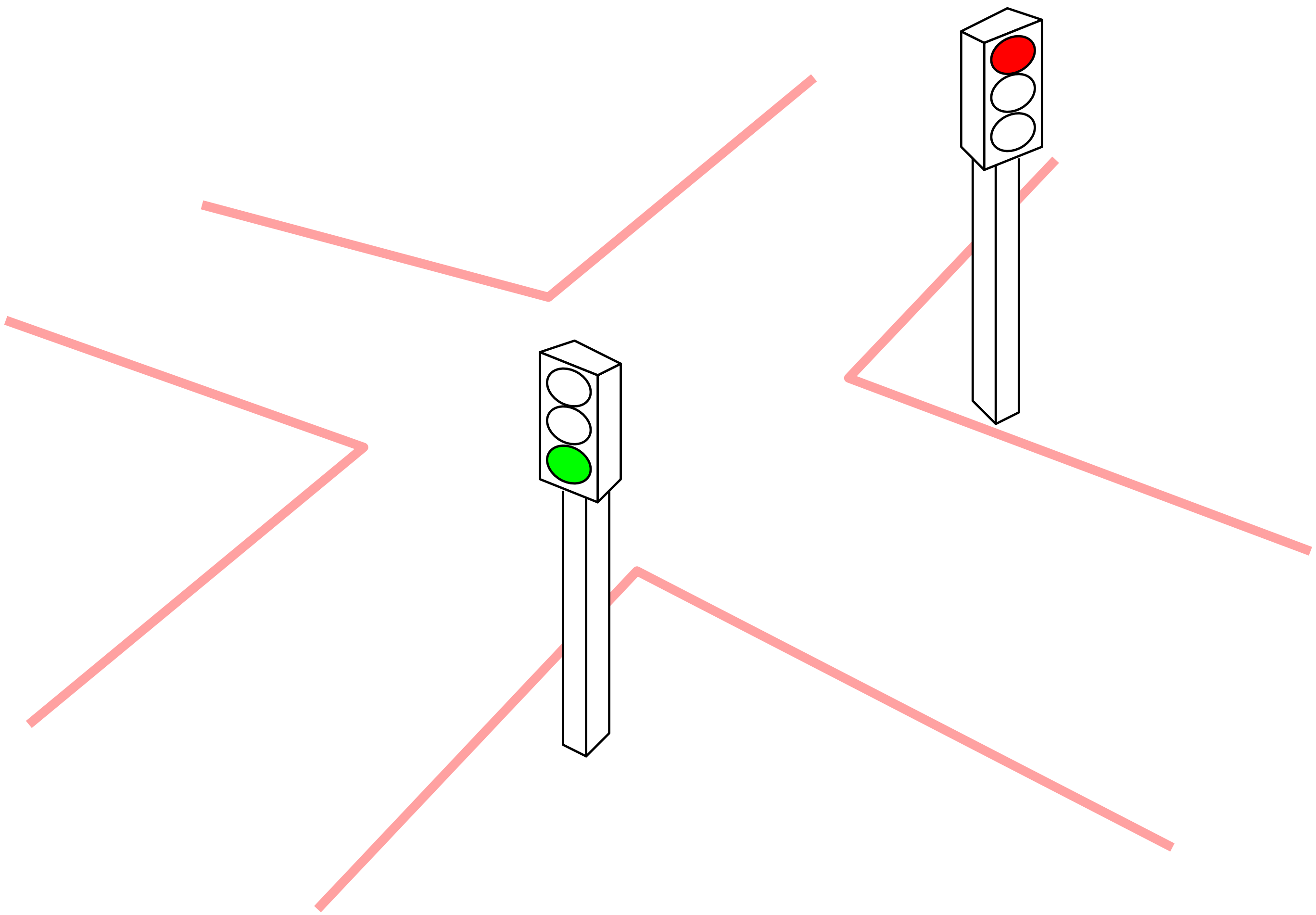
$$T \equiv (x \rightarrow x')(y \rightarrow y')$$

$$B \equiv xy$$

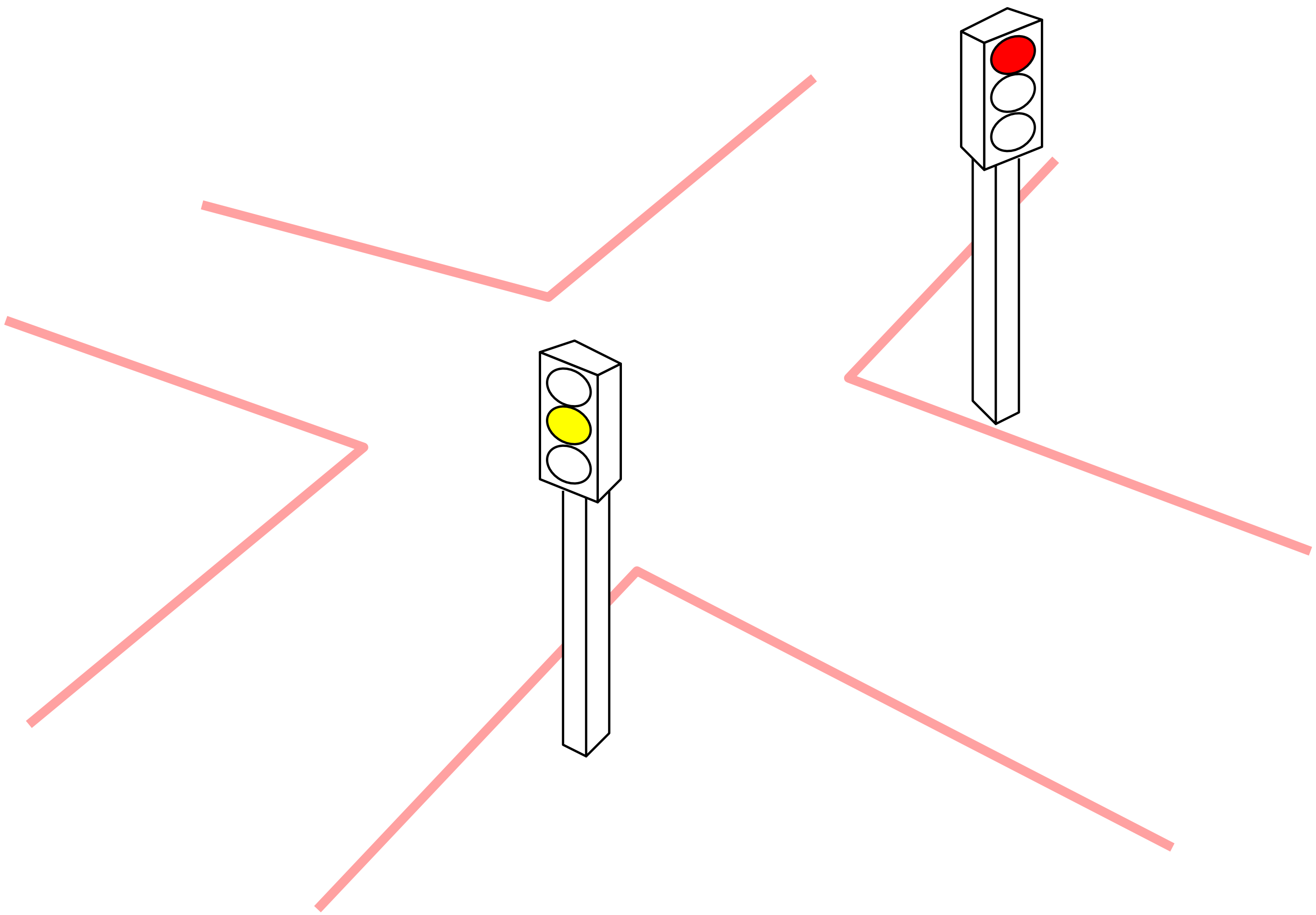


$$\begin{array}{ccccccc} I(s_0) & \wedge & T(s_0, s_1) & \wedge & T(s_1, s_2) & \wedge & B(s_2) \\ \bar{x}_0\bar{y}_0 & \wedge & (x_0 \rightarrow x_1)(y_0 \rightarrow y_1) & \wedge & (x_1 \rightarrow x_2)(y_1 \rightarrow y_2) & \wedge & x_2y_2 \end{array}$$

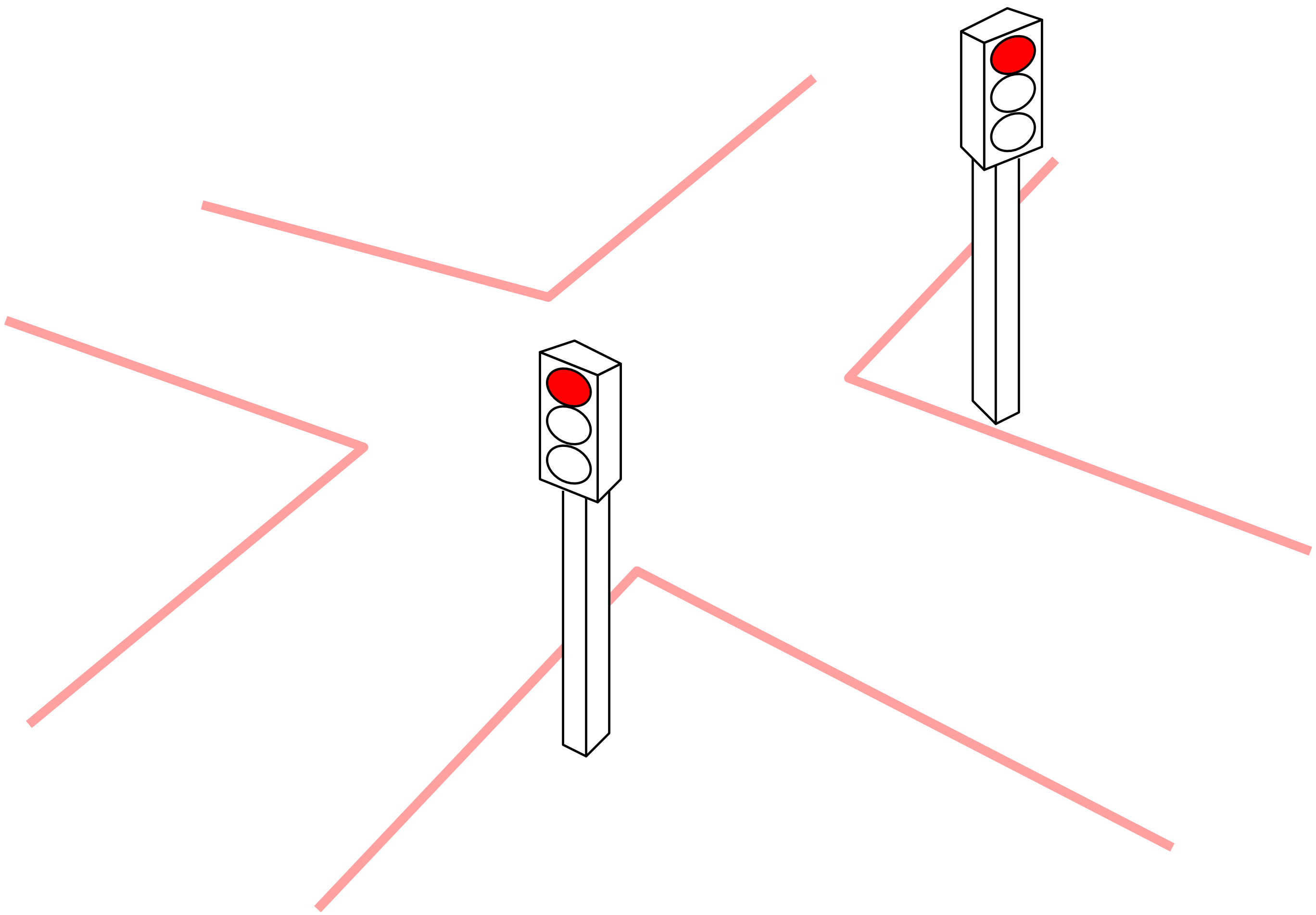
satisfying assignment: $(x_0, y_0) = (0,0), \quad (x_1, y_1) = (1,0), \quad (x_2, y_2) = (1,1)$



traffic lights showing red should eventually show green



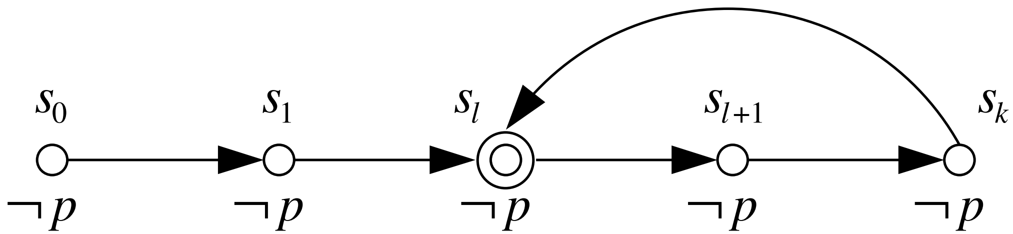
traffic lights showing red should eventually show green



traffic lights showing red should eventually show green

[BiereCimattiClarkeZhu-TACAS'99]

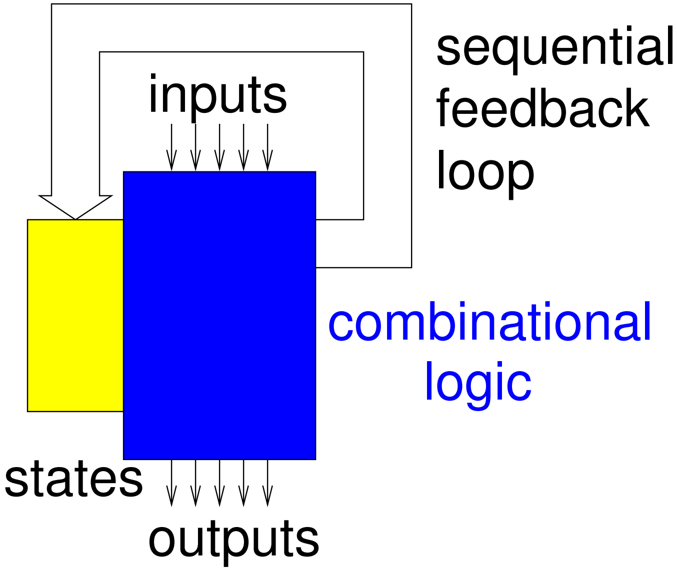
generic counter example trace of length k for liveness **$\mathbf{F}\,p$**



think of $\neg p$ = "no progress"

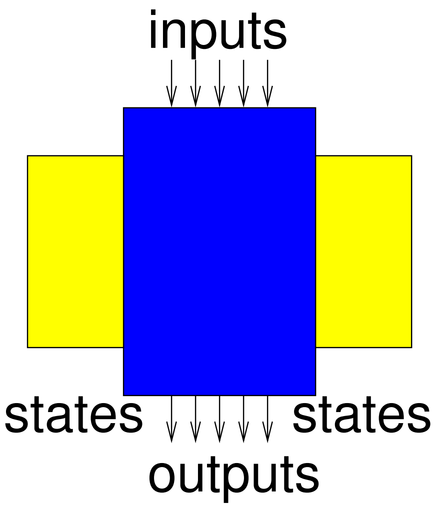
$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_k, s_{k+1}) \wedge \bigvee_{l=0}^{k} s_l = s_{k+1} \wedge \bigwedge_{i=0}^{k} \neg p(s_i)$$

for finite systems liveness can always be reformulated as safety [BiereArthoSchuppan02]



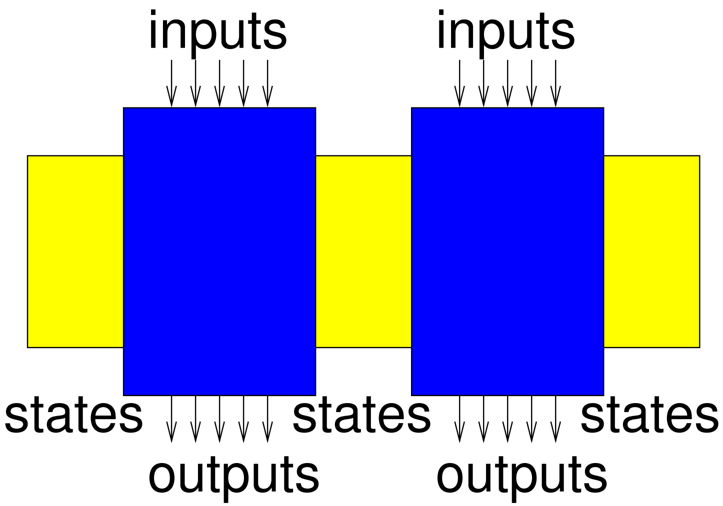
sequential circuit

Time Frame Expansion in HW



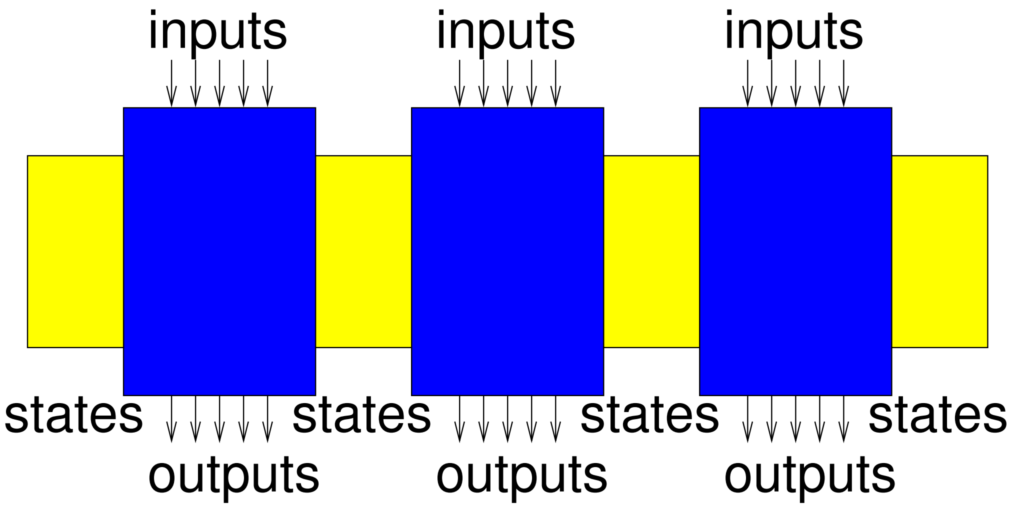
break sequential loop

Time Frame Expansion in HW



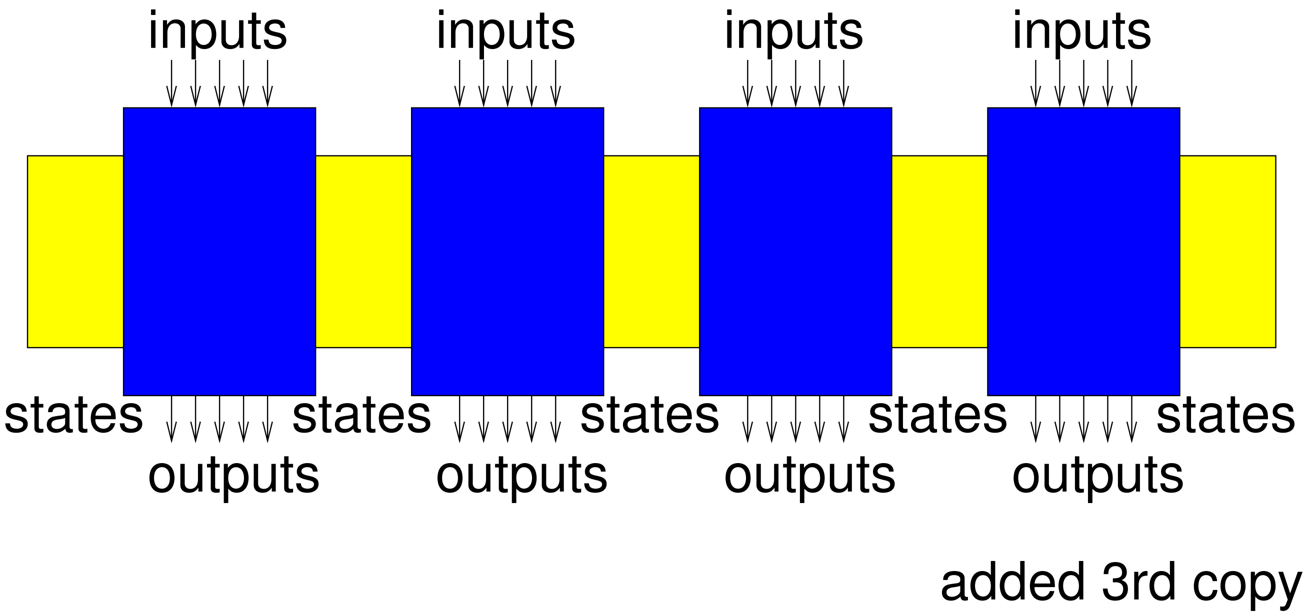
added 1st copy

Time Frame Expansion in HW

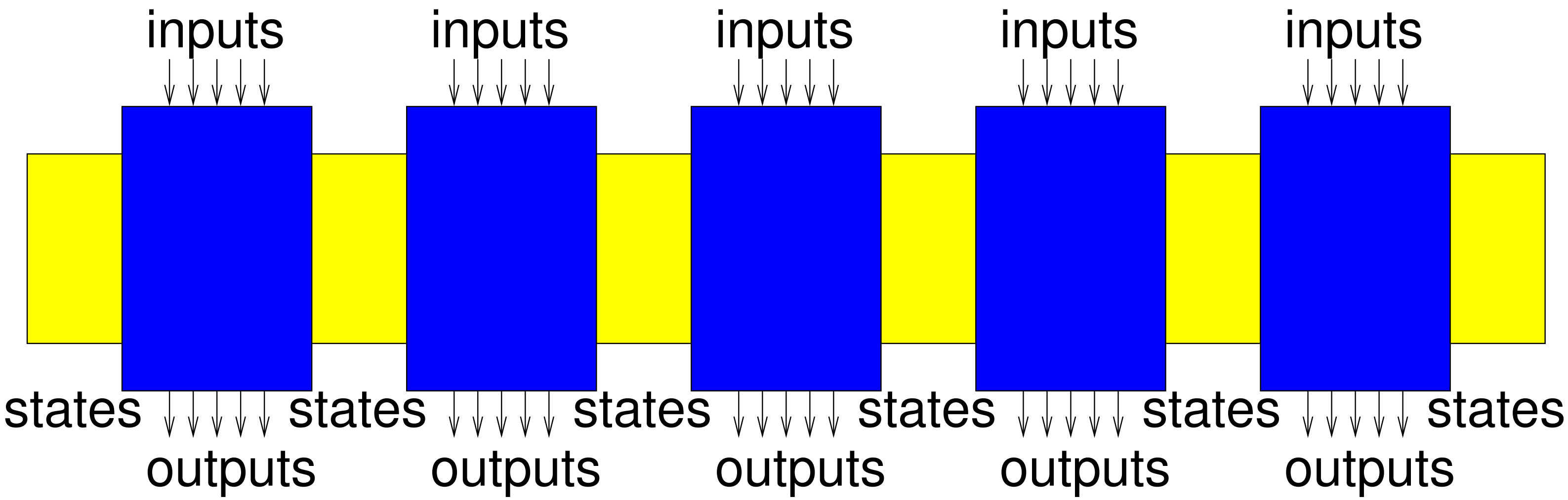


added 2nd copy

Time Frame Expansion in HW

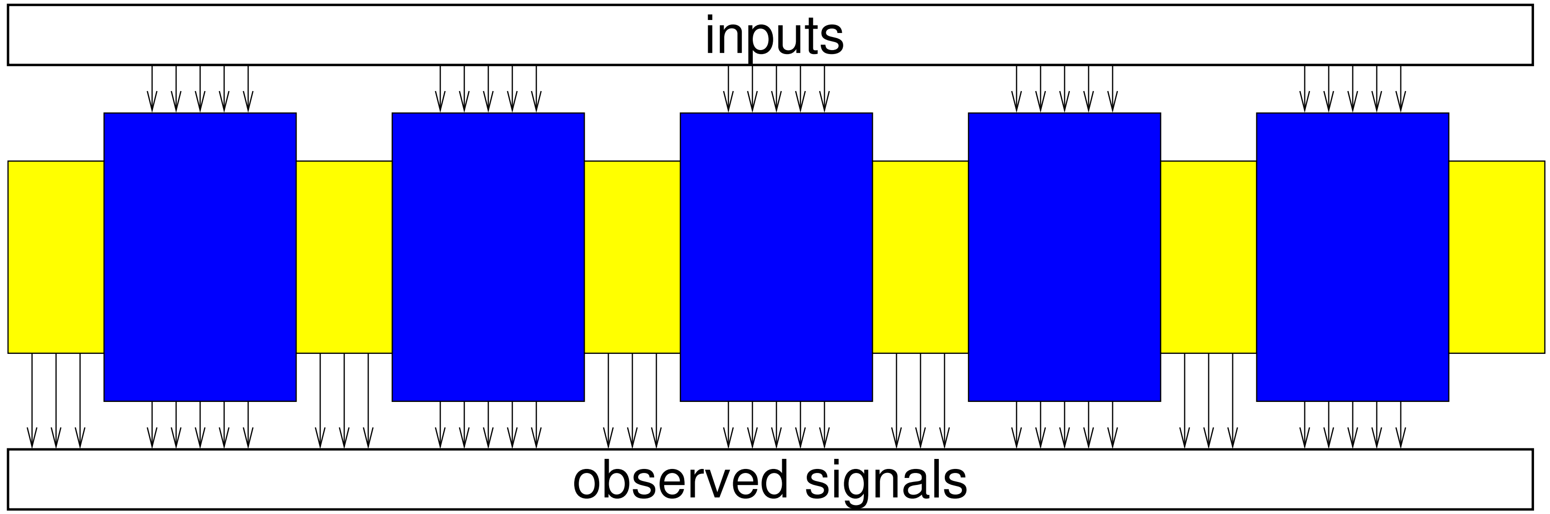


Time Frame Expansion in HW

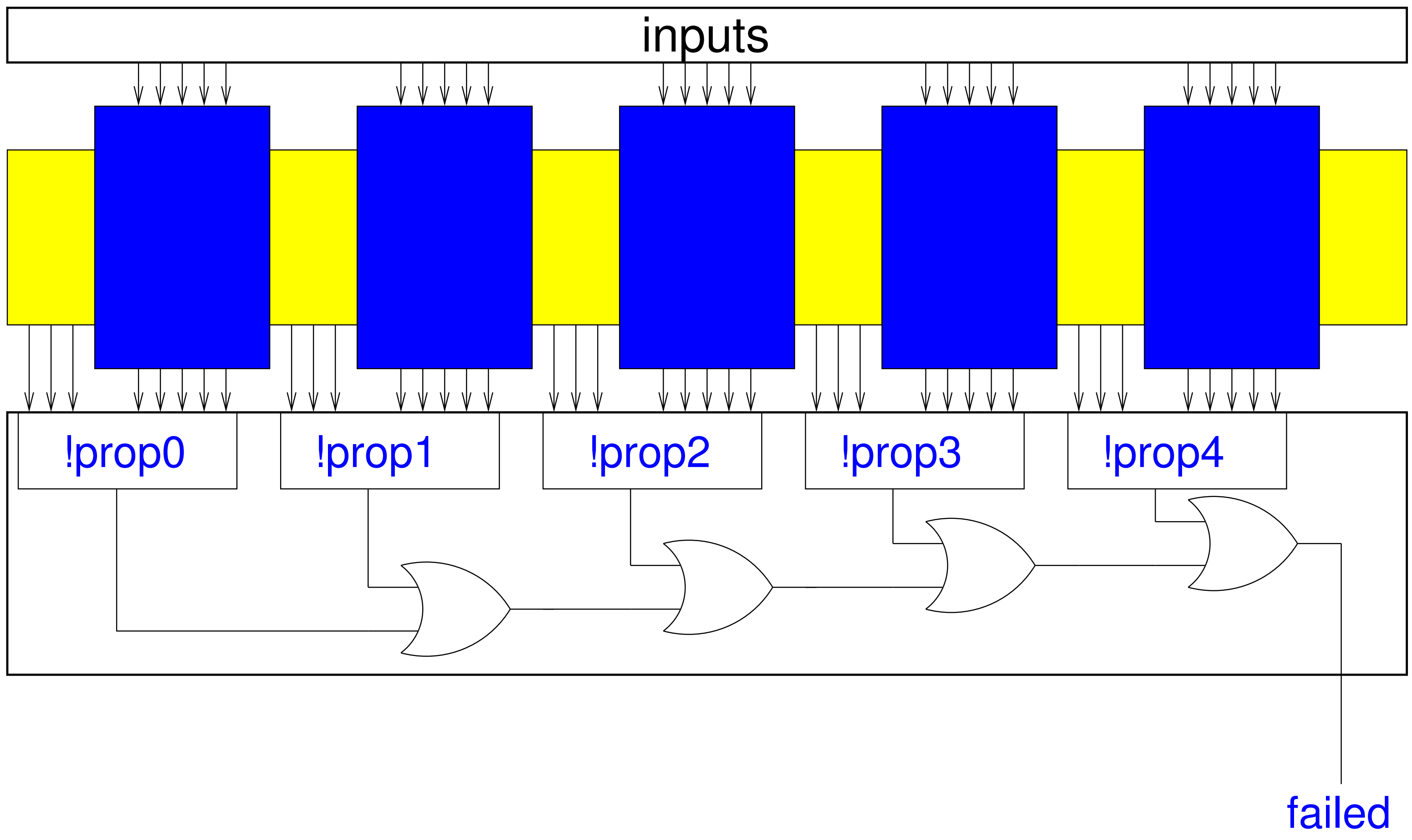


added 4th copy

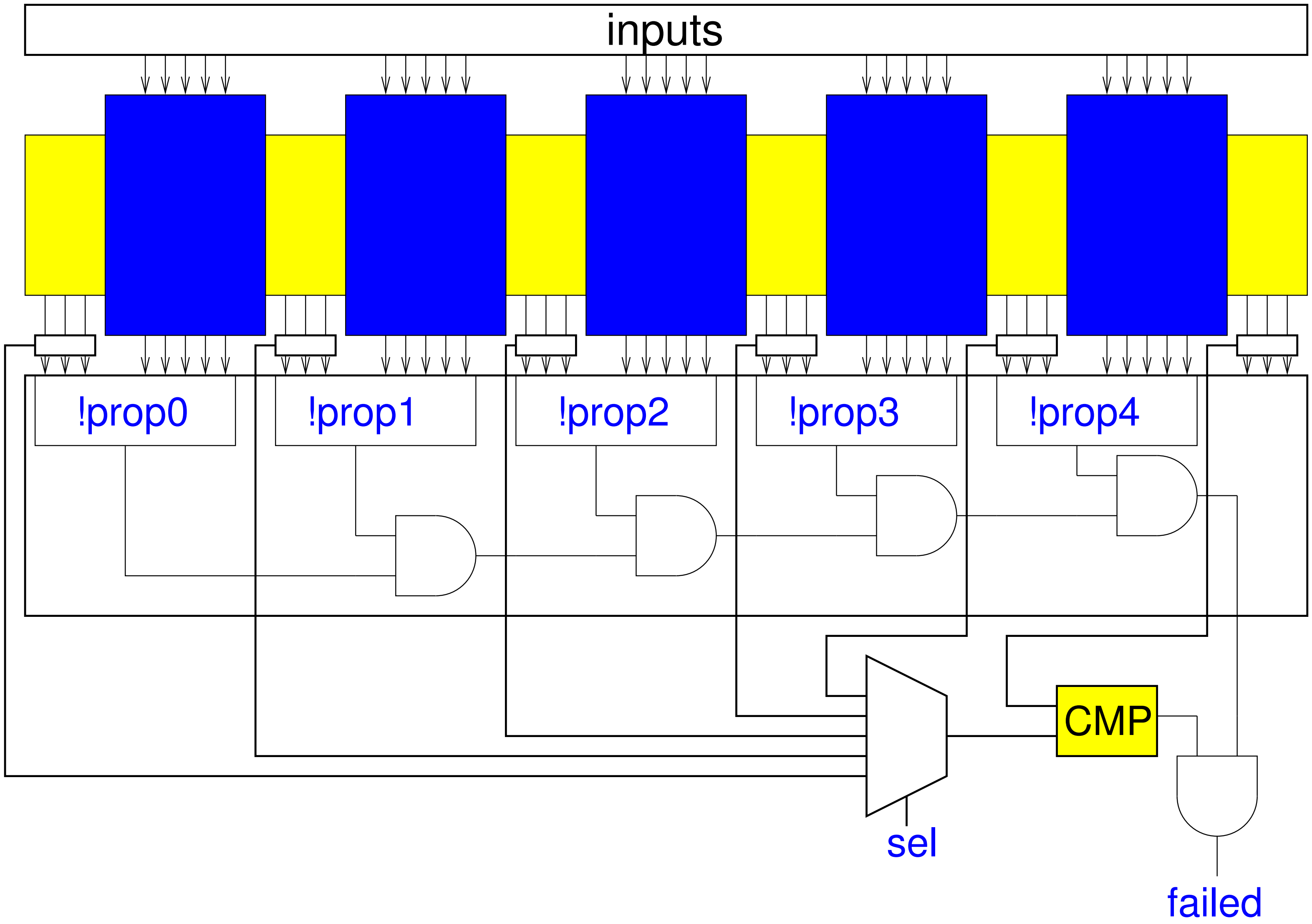
Time Frame Expansion in HW



Bounded Model Checking Safety in HW



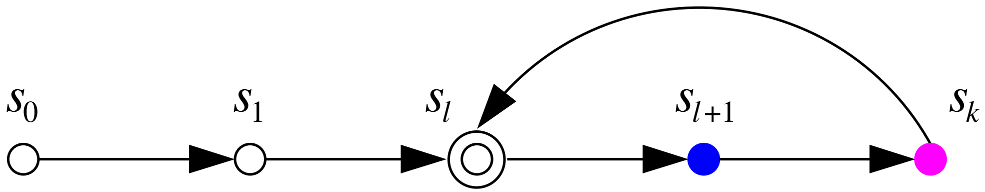
find inputs for which failed becomes true



find inputs for which failed becomes true

$$(\mathbf{GF} f) \quad \wedge \quad (\mathbf{GF} g)$$

path $\pi = (s_0, s_1, s_2, \ldots)$ is **fair** iff all fairness constraints occur infinitely often on π



$$I(s_0) \wedge T(s_0, s_1) \wedge \cdots \wedge T(s_k, s_{k+1}) \wedge \bigvee_{l=0}^{k} \left(s_l = s_{k+1} \wedge \bigvee_{j=l}^{k} f(s_j) \wedge \bigvee_{j=l}^{k} g(s_j) \right)$$

generalizes to tableau constructions for LTL

Completeness in Bounded Model Checking

- find bounds on the maximal length of counter examples [BiereCimattiClarkeZhu99]
  - also called **completeness threshold** [KroeningStrichman03]
  - exact bounds are hard to find ⇒ approximations
- **induction**
  - try to find inductive invariants (algorithmically and/or manually)
  - algorithmic generalization of inductive invariants: k-induction
- use of SAT for **quantifier elimination** as with BDDs
  - then model checking becomes fixpoint calculation
  - interpolation as approximation of quantifier elimination
- **relative inductive reasoning** as in IC3 by Aaron Bradley

Distance: length of shortest path between two states

$$\delta(s,t) \equiv \min\{n \mid \exists s_0,\ldots,s_n[s = s_0, t = s_n \text{ and } T(s_i,s_{i+1}) \text{ for } 0 \leq i < n]\}$$

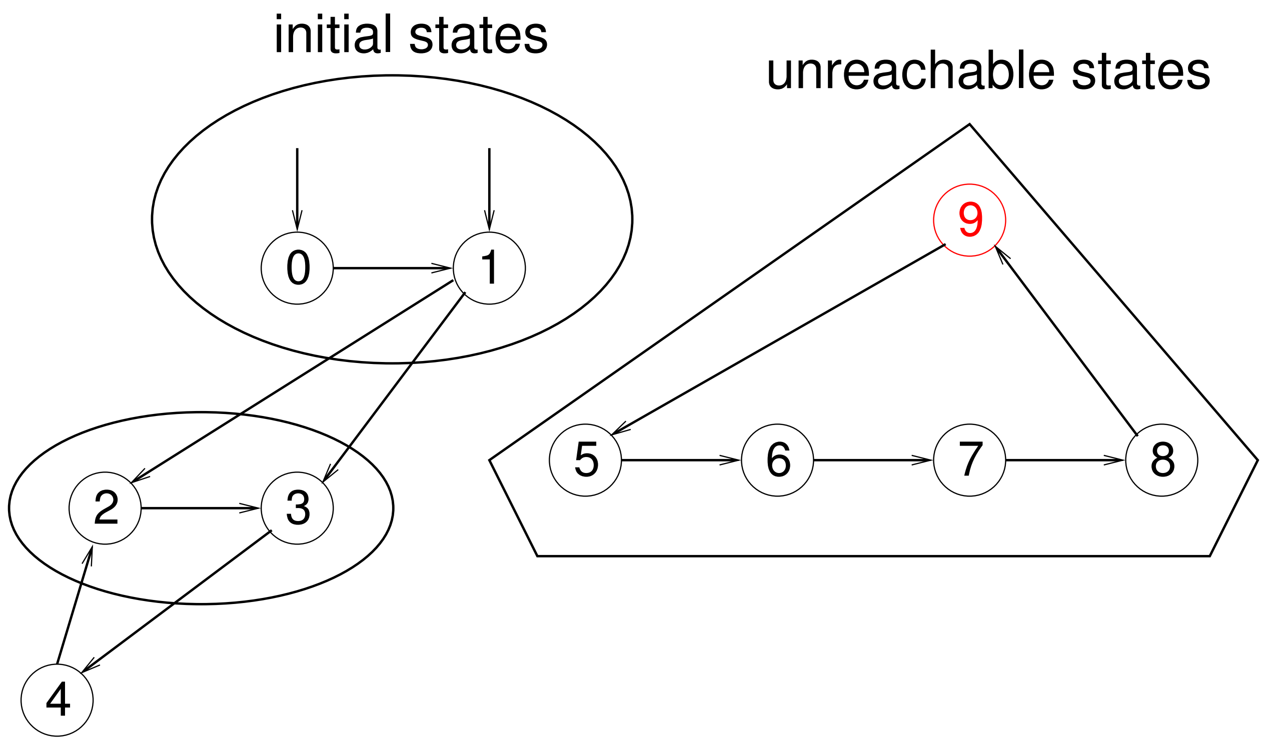
Diameter: maximal distance between two connected states

$$d(T) \equiv \max\{\delta(s,t) \mid T^*(s,t)\}$$

Radius: maximal distance of a reachable state from the initial states

$$r(T,I) \equiv \max\{\delta(s,t) \mid T^*(s,t) \text{ and } I(s) \text{ and } \delta(s,t) \leq \delta(s',t) \text{ for all } s' \text{ with } I(s')\}$$

Diameter Example



(forward) radius = 2 diameter = 4 backward radius = 4

- number of steps needed to reach a bad state reached can be bounded by radius
  - works both for *forward* radius and *backward* radius
  - so we can use the minimum of the two
- radius **completeness threshold** for safety properties
  - safety properties: max. k for doing bounded model checking bounded
  - if no counter example of this length can be found the safety property holds

reformulation:

radius max. length r of an initialized path leading to a state t, such there is no other path from an initial state to t with length less than r.

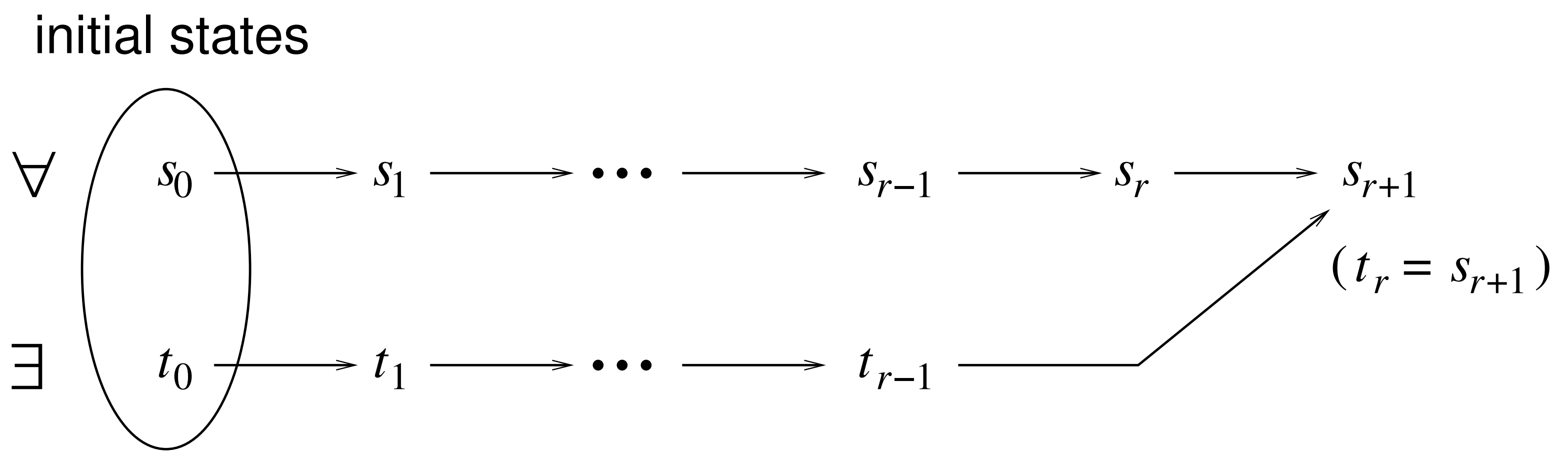
Thus radius r is the minimal number which makes the following formula valid:

$$\forall s_0,\ldots,s_{r+1}[\,(I(s_0)\wedge\bigwedge_{i=0}^{r}T(s_i,s_{i+1}))\rightarrow$$
$$\exists\, n\leq r\,[\,\exists t_0,\ldots,t_n[\,I(t_0)\wedge\bigwedge_{i=0}^{n-1}T(t_i,t_{i+1})\wedge t_n=s_{r+1}\,]\,]\,]$$

Quantified Boolean Formula (QBF)

to prove un/satisfiable of QBF is PSPACE complete

Visualization of Reformulation



we allow t_{i+1} to be identical to t_i in the lower path

- we can not find the real radius / diameter with SAT efficiently

- over approximation idea:
  - drop requirement that there is no shorter path
  - enforce *different* (no reoccurring) states on single path instead
  - also called simple paths

reoccurrence diameter:

length of the longest *simple path*

reoccurrence radius:

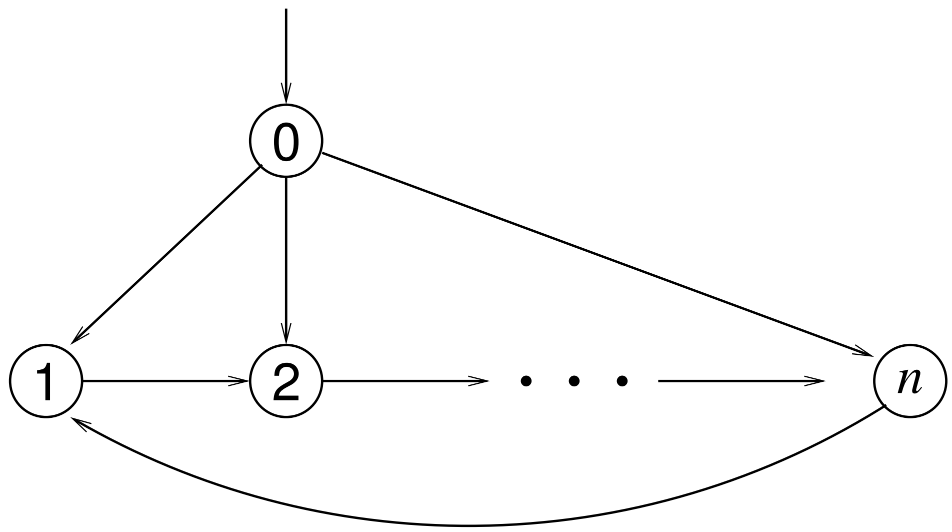
length of the longest initialized *simple path* *reoccurring radius* is minimal r which makes the following formula valid:

$$I(s_0) \wedge \bigwedge_{i=0}^{r} T(s_i, s_{i+1}) \rightarrow \bigvee_{0 \leq i < j \leq r+1} s_i = s_j$$

which is valid iff the following formula is unsatisfiable:

$$I(s_0) \wedge \bigwedge_{i=0}^{r} T(s_i, s_{i+1}) \wedge \underbrace{\bigwedge_{0 \leq i < j \leq r+1} s_i \neq s_j}_{\text{simple path constraints}}$$

Bad Example for Reoccurrence Radius



radius 1, reoccurrence radius n

k-Induction

for $k = 0 \ldots \infty$ check

1. k-induction base case:

$$I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge B(s_k) \wedge \bigwedge_{0 \le i < k} \neg B(s_i) \qquad \text{satisfiable?}$$

2. k-induction induction step:

$$T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge B(s_k) \wedge \bigwedge_{0 \le i < k} \neg B(s_i) \qquad \text{unsatisfiable?}$$

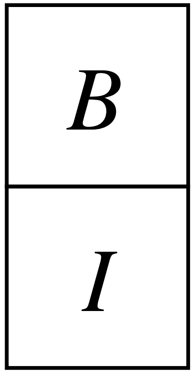
if base case satisfiable (= BMC), then bad state reachable

if inductive step unsatisfiable, then bad state **unreachable**

incomplete without simple path constraints

Incremental SAT Solving for BMC and k-Induction

[EénSörensson'03]



$k = 0$ base case

Incremental SAT Solving for BMC and k-Induction

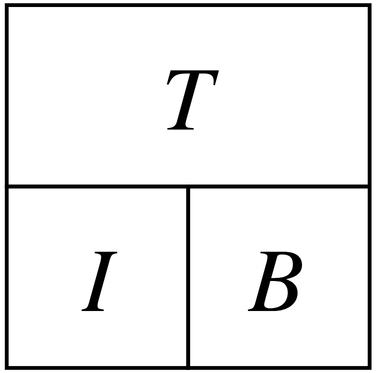
[EénSörensson'03]

$$\boxed{B}$$

$k = 0$ inductive step

Incremental SAT Solving for BMC and k-Induction

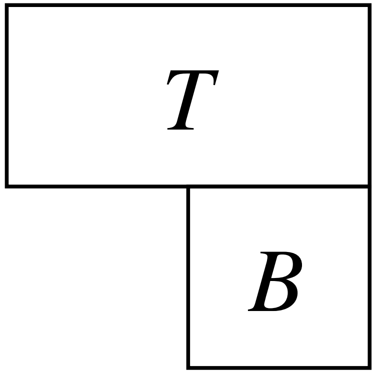
[EénSörensson'03]



$k = 1$ base case

Incremental SAT Solving for BMC and k-Induction

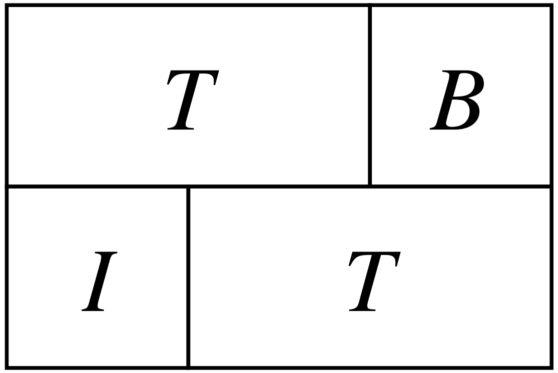
[EénSörensson'03]



$k = 1$ inductive step

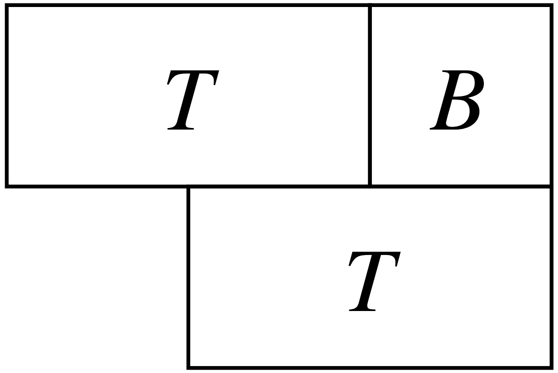
Incremental SAT Solving for BMC and k-Induction

[EénSörensson'03]



$k = 2$ base case

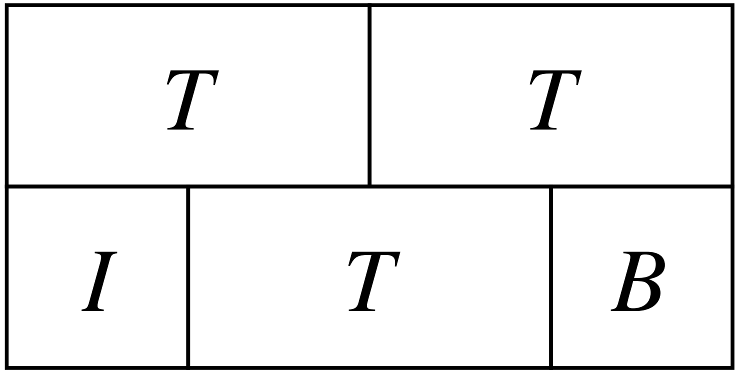
[EénSörensson'03]



$k = 2$ inductive step

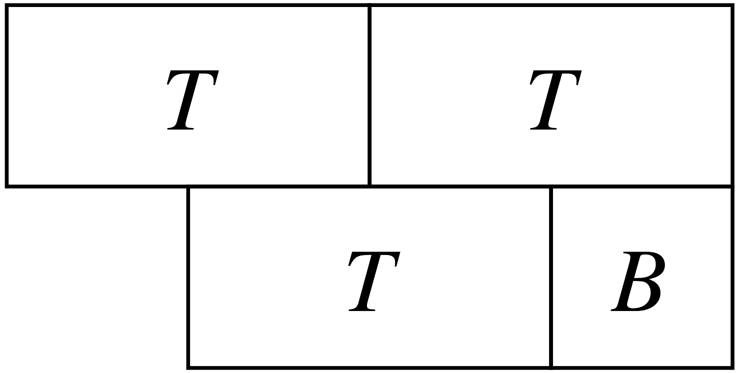
Incremental SAT Solving for BMC and k-Induction

[EénSörensson'03]



$k = 3$ base case

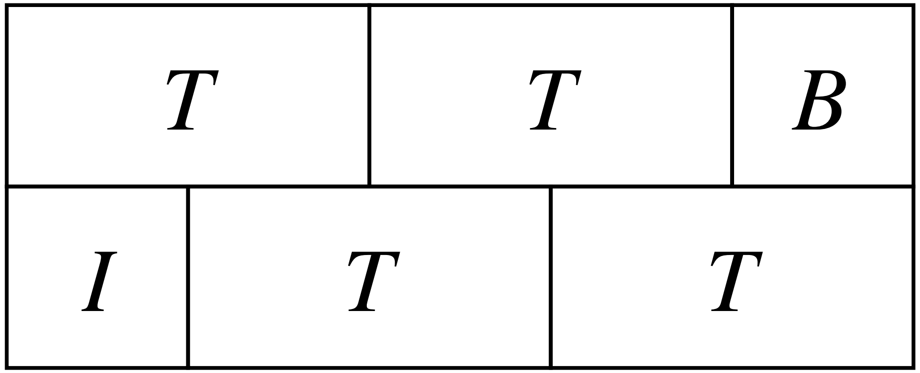
[EénSörensson'03]



$k = 3$ inductive step

Incremental SAT Solving for BMC and k-Induction

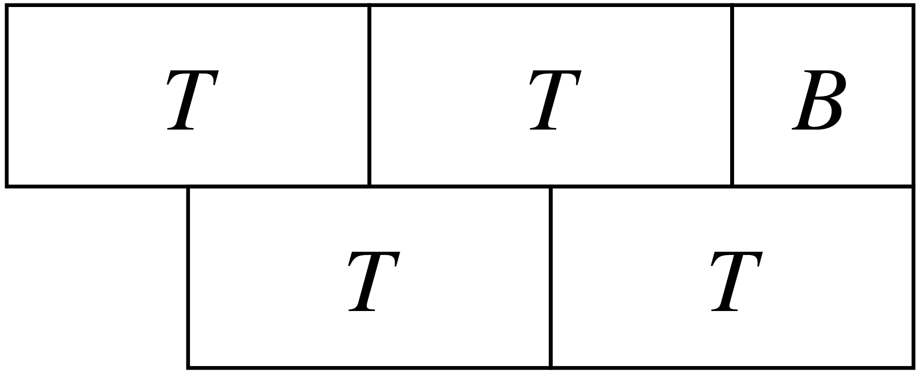
[EénSörensson'03]



$k = 4$ base case

Incremental SAT Solving for BMC and k-Induction

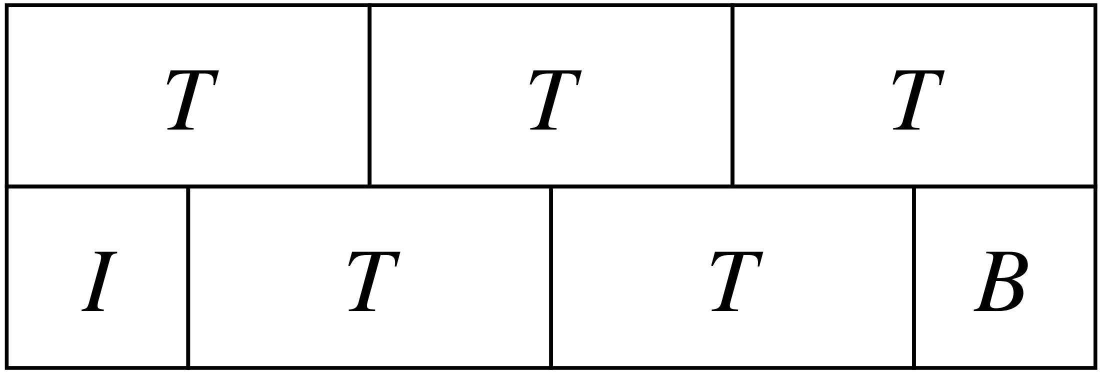
[EénSörensson'03]



$k = 4$ inductive step

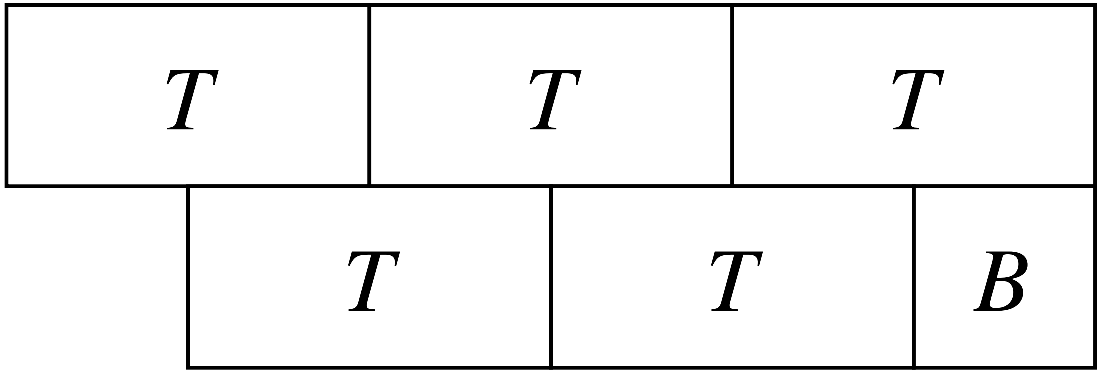
Incremental SAT Solving for BMC and k-Induction

[EénSörensson'03]



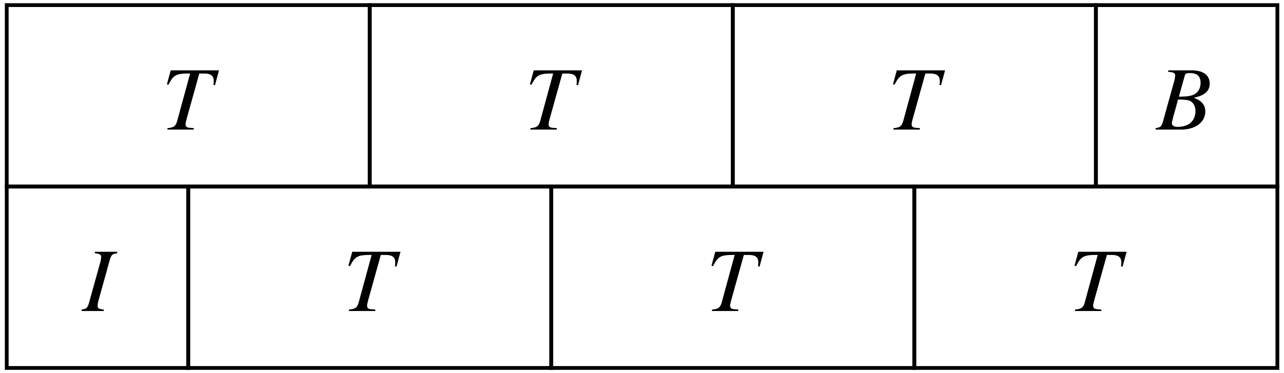
$k = 5$ base case

[EénSörensson'03]



$k = 5$ inductive step

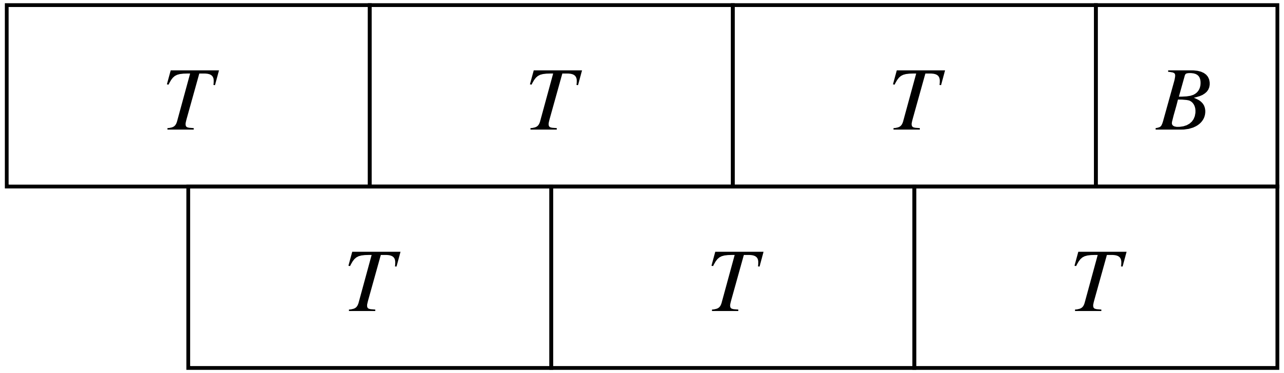
[EénSörensson'03]



$k = 6$ base case

Incremental SAT Solving for BMC and k-Induction

[EénSörensson'03]



$k = 6$ inductive step

Simple Path Constraints

- bounded model checking: [BiereCimattiClarkeZhu'99]

$$I(s_1) \wedge T(s_1,s_2) \wedge \ldots \wedge T(s_{k-1},s_k) \wedge \bigvee_{0 \leq i \leq k} B(s_i) \qquad \text{satisfiable?}$$

- reoccurrence diameter checking: [BiereCimattiClarkeZhu'99]

$$T(s_1,s_2) \wedge \ldots \wedge T(s_{k-1},s_k) \ \wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j \qquad \text{unsatisfiable?}$$

- k-induction base case: [SheeranSinghStålmarck'00]

$$I(s_1) \wedge T(s_1,s_2) \wedge \ldots \wedge T(s_{k-1},s_k) \ \wedge B(s_k) \wedge \bigwedge_{0 \leq i < k} \neg B(s_i) \qquad \text{satisfiable?}$$

- k-induction induction step: [SheeranSinghStålmarck'00]

$$T(s_1,s_2) \wedge \ldots \wedge T(s_{k-1},s_k) \ \wedge B(s_k) \wedge \bigwedge_{0 \leq i < k} \neg B(s_i) \ \wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j \qquad \text{unsatisfiable?}$$

- automatic abstraction refinement = lemmas on demand of simple path constraints [EénSörensson'03]

Special Cases

let $G = \neg B$ denote the "good states":

- 0-induction base case: $\boxed{I(s_0) \wedge B(s_0) \text{ satisfiable}}$ iff initial bad state exists
- 0-induction inductive step: $\boxed{B(s_0) \text{ unsatisfiable}}$ iff $\neg B$ propositional tautology
- 1-induction base: $\boxed{I(s_0) \wedge T(s_0, s_1) \wedge B(s_1) \text{ satisfiable}}$ iff bad state reachable in one step
- 1-induction inductive step: $\boxed{\neg B(s_0) \wedge T(s_0, s_1) \wedge B(s_1) \text{ unsatisfiable}}$ iff G inductive

  assuming 0-induction base case was unsatisfiable and thus $I \models G$

where $G = \neg B$ is called **inductive** iff 1. $I \models G$ and 2. $G \wedge T \models G'$

SAT-based Deductive Model Checking

[BiereCimattiClarkeFujitaZhu'00]

task is to prove that p is an invariant — $\mathbf{G}p$ holds on the model

- guess a formula G stronger than p: $G \models p$ — 1st check
- show G inductive: $I \models G, \quad G \wedge T \models G'$ — 2nd, 3rd check
- all three checks can be formulated as UNSAT checks
- if one check fails refine G based on satisfying assignment

manual process and thus complete on finite state systems

there are also automatic abstraction/refinement versions of this approach

CEGAR [ClarkeGrumbergJhaLuVeith'00]

Interpolant

Definition I interpolant of A and B iff

(1) $A \Rightarrow I$ (2) $V(I) \subseteq G = V(A) \cap V(B)$ (3) $I \wedge B$ unsatisfiable

Note: $A \wedge B$ unsatisfiable as a consequence.

Intuition: I abstraction of A over the common (global/interface) variables G of A and B which still is inconsistent with B.

strongest interpolant $\exists L_A[A]$ with $L_A = V(A) \backslash G$

Let A and B formulas in CNF.
From a refutational resolution proof of $A \wedge B$ generate interpolant I.

next slide

Many applications, approx. quantifier elimination, gives fast model checking algorithm.

Extracting Interpolants from Refutations

[McMillan'03, McMillan'05] + [Biere'09] (BMC chapter in Handbook)

Definition interpolating quadruple $(A,B)\ c\ [\,f\,]$ is *well-formed* iff

$$\text{(W1)}\quad V(c) \subseteq V(A) \cup V(B) \qquad\qquad \text{(W2)}\quad V(f) \subseteq G \cup (V(c) \cap V(A)) \subseteq V(A)$$

Definition well-formed interpolating quadruple $(A,B)\ c\ [\,f\,]$ is *valid* iff

$$\text{(V1)}\quad A \Rightarrow f \qquad\qquad \text{(V2)}\quad B \wedge f \Rightarrow c$$

Definitition proof rules for interpolating quadrupels

$$\text{(R1)}\quad \frac{}{(A,B)\ c\ [\,c\,]}\ c \in A \qquad\qquad \frac{(A,B)\ c \dot\vee l\ [\,f\,] \qquad (A,B)\ d \dot\vee \bar{l}\ [\,g\,]}{(A,B)\ c \vee d\ [\,f \wedge g\,]}\ |l| \in V(B) \quad \text{(R3)}$$

$$\text{(R2)}\quad \frac{}{(A,B)\ c\ [\,\top\,]}\ c \in B \qquad\qquad \frac{(A,B)\ c \dot\vee l\ [\,f\,] \qquad (A,B)\ d \dot\vee \bar{l}\ [\,g\,]}{(A,B)\ c \vee d\ [\,f|_{\bar{l}} \vee g|_{l}\,]}\ |l| \notin V(B) \quad \text{(R4)}$$

Theorem proof rules produce well-formed and valid interpolating quadruples

$$\overbrace{I(s_{-1}) \wedge T(s_{-1},s_0)}^{A} \quad \wedge \quad \overbrace{T(s_0,s_1) \wedge T(s_1,s_2) \wedge T(s_2,s_3) \wedge \bigvee_{i=0}^{3} \neg G(s_i)}^{B}$$

interpolant $P_1(s_0)$ let $R_1 \equiv I \vee P_1$

$$R_1(s_{-1}) \wedge T(s_{-1},s_0) \quad \wedge \quad T(s_0,s_1) \wedge T(s_1,s_2) \wedge T(s_2,s_3) \wedge \bigvee_{i=0}^{3} \neg G(s_i)$$

interpolant $\boxed{R_2(s_0) \Leftarrow R_1(s_{-1}) \wedge T(s_{-1},s_0)}$ let $R_2 \equiv R_1 \vee P_2$

$$R_2(s_{-1}) \wedge T(s_{-1},s_0) \quad \wedge \quad T(s_0,s_1) \wedge T(s_1,s_2) \wedge T(s_2,s_3) \wedge \bigvee_{i=0}^{3} \neg G(s_i)$$

$$\vdots$$

$$R_{n-1}(s_{-1}) \wedge T(s_{-1},s_0) \quad \wedge \quad T(s_0,s_1) \wedge T(s_1,s_2) \wedge T(s_2,s_3) \wedge \bigvee_{i=0}^{3} \neg G(s_i)$$

interpolant $P_n(s_0)$

---

until $R_n \equiv R_{n-1}$ fix-point guaranteed for k = backward radius of $\neg G$

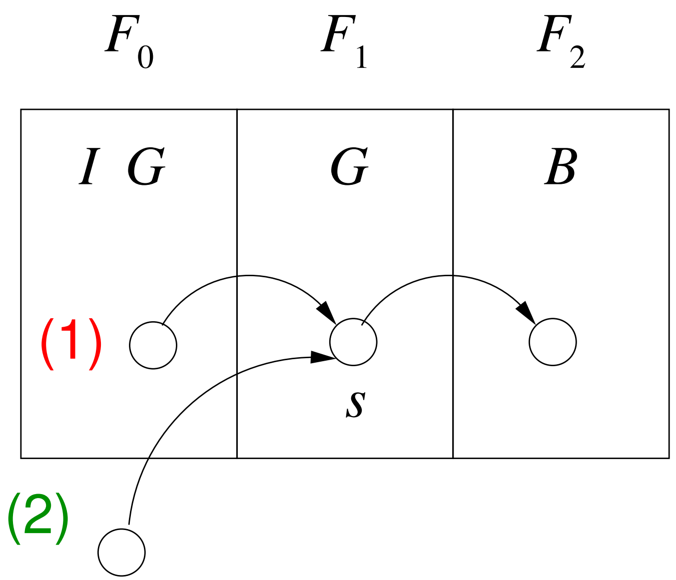
SAT-based Model Checking without Unrolling

[Bradley'11] + [EénMishchenkoBrayton'11]

I initial states

G good states

B bad states



$F_0 \supseteq F_1 \supseteq F_2$

sets of rel. ind. clauses

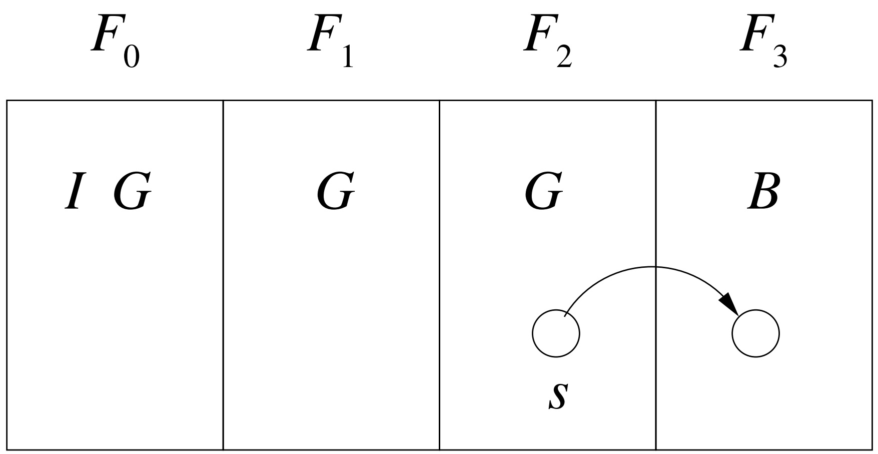
new key concept in [Bradley'11]:

clause c relative inductive w.r.t. F iff $c \wedge F \wedge T \Rightarrow c'$ iff $c \wedge F \wedge T \wedge \bar{c}'$ unsatisfiable

(1) s is reachable from F_0 then bad is reachable transitively

(2) otherwise exists $c \subseteq \bar{s}$ rel. ind. w.r.t. F_0 can be added to F_1 and maybe to F_2

as soon the last set is good, i.e. $F_k \Rightarrow G$ increase k



propagate all relative inductive clauses of last set to new set

if all can been propagated F_k is an inductive invariant stronger than G

Let $F_0, \ldots, F_k$ be a sequence of *sets of clauses*.

---

monotonic iff $\boxed{F_i \supseteq F_{i+1}}$ for $i = 0 \ldots k-1$

(relative) **inductive** iff $\boxed{F_i\, T \Rightarrow F'_{i+1}}$ for $i = 0 \ldots k-1$

initialized iff $\boxed{I \equiv F_0}$

good iff $\boxed{F_i \Rightarrow G}$ for $i = 0 \ldots k-1$ last set might be bad if $F_k \wedge B$ satisfiable

---

F is k-**adequat** iff all states s satisfying F are at least k steps away from B
[McMillan'03]

sequence monotonic and inductive $\Rightarrow$ F_{k-j} j-adequat

CHECK (s, i) {

while $\bar{s} \wedge F_{i-1} \wedge T \wedge s'$ satisfiable {

if $i = 1$ **throw** SATISFIABLE

choose cube t with $t \models \bar{s} \wedge F_{i-1} \wedge T \wedge s'$

CHECK $(t, i-1)$

}

choose clause $c \subseteq \bar{s}$ with $c \wedge F_{i-1} \wedge T \wedge \bar{c}'$ unsatisfiable

F_j := $F_j \cup \{c\}$ for all $j = 1 \ldots i$ and if possible for higher j

}

MAIN (s, i) {

$F_0 = I$, $F_1 = \top$, $k = 1$ do not forget to check base cases first

forever {

CHECK (B, k)

k := $k+1$, F_k := all rel. ind. clauses of F_{k-1} w.r.t. F_{k-1}

if $F_k \subseteq F_{k-1}$ **throw** UNSATISFIABLE

}

}

- implemented in IC3 by Aaron Bradley
  - as single engine model checker extremely successful in HWMCC'10
    Hardware Model Checking Competition 2010
  - based on rather out-dated SAT solver (ZChaff from 2004)
- independent implementations such as [EénMishchenkoBrayton IWLS'11]
  - seem to be faster than BDDs, k-induction, interpolation
  - might be much easier to lift to SMT-based model checking than interpolation
  - opportunities for improvement: structural SAT/SMT solving

- affiliated to FMCAD'11, Novemeber 2011, Austin
  - we expect new benchmarks from industry
  - and improved implementations
- checking *multiple* properties
- checking *liveness* properties
- new AIGER format

http://fmv.jku.at/hwmcc11

given (symbol encoding of) an *infinite* path $\pi = (s_0, s_1, \ldots)$.

$$
\begin{array}{rll}
s_i \models p & \text{iff} & p(s_0) \\
s_i \models \mathbf{X} f & \text{iff} & s_{i+1} \models f \\
s_i \models \mathbf{G} f & \text{iff} & \forall j \leq i [s_j \models f] \\
s_i \models \mathbf{F} f & \text{iff} & \exists j \leq i [s_j \models f]
\end{array}
$$

How to define/encode bounded semantics for a *lasso*?

where lasso is a path $\pi = (s_0, s_1, \ldots, s_k)$ and $s_l = s_k$ for one l

Simple and Linear Translation of full LTL for BMC

[LatvalaBiereHeljankoJunttila FMCAD'04]

evaluate semantics on loop in two iterations

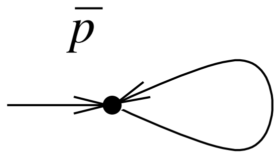
$\langle\ \rangle$ = 1st iteration $\quad$ $[\ ]$ = 2nd iteration

$:=$	$i < k$	$i = k$
$[p]_i$	$p(s_i)$	$p(s_k)$
$[\neg p]_i$	$\neg p(s_i)$	$\neg p(s_k)$
$[\mathbf{X}f]_i$	$[f]_{i+1}$	$\bigvee_{l=0}^{k}(T(s_k,s_l)\wedge[f]_l)$
$[\mathbf{G}f]_i$	$[f]_i\wedge[\mathbf{G}f]_{i+1}$	$\bigvee_{l=0}^{k}(T(s_k,s_l)\wedge\langle\mathbf{G}f\rangle_l)$
$[\mathbf{F}f]_i$	$[f]_i\vee[\mathbf{F}f]_{i+1}$	$\bigvee_{l=0}^{k}(T(s_k,s_l)\wedge\langle\mathbf{F}f\rangle_l)$
$\langle\mathbf{G}f\rangle_i$	$[f]_i\wedge\langle\mathbf{G}f\rangle_{i+1}$	$[f]_k$
$\langle\mathbf{F}f\rangle_i$	$[f]_i\vee\langle\mathbf{F}f\rangle_{i+1}$	$[f]_k$

- LTL semantics on *single path* the same as CTL semantics
  - symbolically implement fixpoint calculation for (A)CTL
  - fixpoint computation terminates after 2 iterations (not k)
  - boolean fixpoint equations

- easy to implement and optimize, fast
  - generalized to past time [LatvalaBiereHeljankoJunttila VMCAI'05]
  - minimal counter examples for past time [SchuppanBiere TACAS'05]
  - incremental (and complete) [LatvalaHeljankoJunttila CAV'05]

Why Not Just Try to Satisfy Boolean Equations directly?



recursive expansion $\mathbf{F}p \equiv p \vee \mathbf{XF}p$

checking $\mathbf{G}\overline{p}$ implemented as search for witness for $\mathbf{F}p$

Kripke structure: single state with self loop in which p does not hold

incorrect translation of $\mathbf{F}p$:

$$\overbrace{I(s_0) \wedge T(s_0, s_0)}^{\text{model constraints}} \;\wedge\; \underbrace{([\mathbf{F}p] \leftrightarrow p(s_0) \vee [\mathbf{F}p])}_{\text{translation}} \;\wedge\; \overbrace{\underbrace{[\mathbf{F}p]}_{x}}^{\text{assumption}}$$

since it is satisfiable by setting $x = 1$ though $p(s_0) = 0$

(x fresh boolean variable introduced for $[\mathbf{F}p]$)