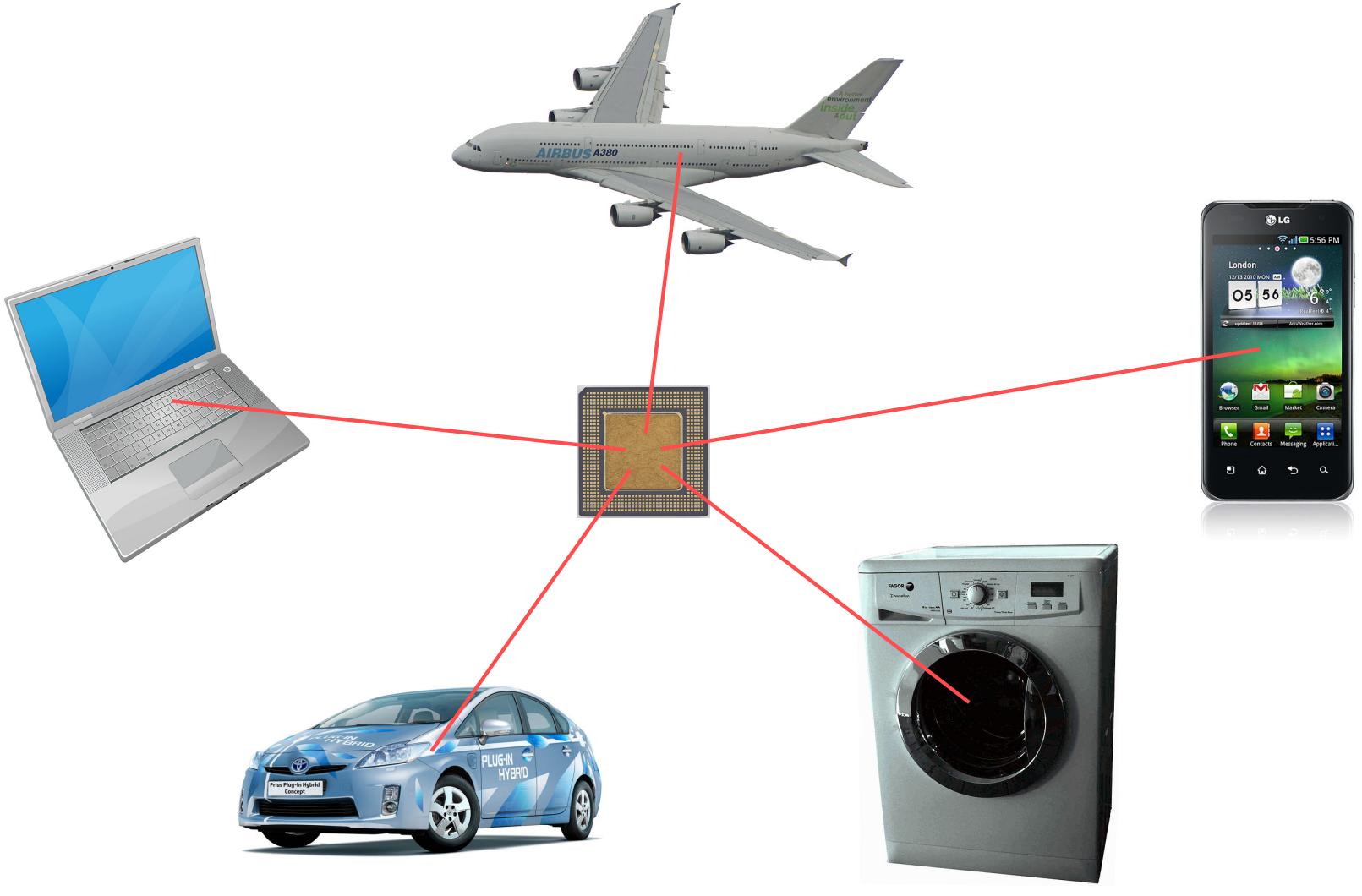
Formal Hardware Verification

Conception des systèmes sur puces

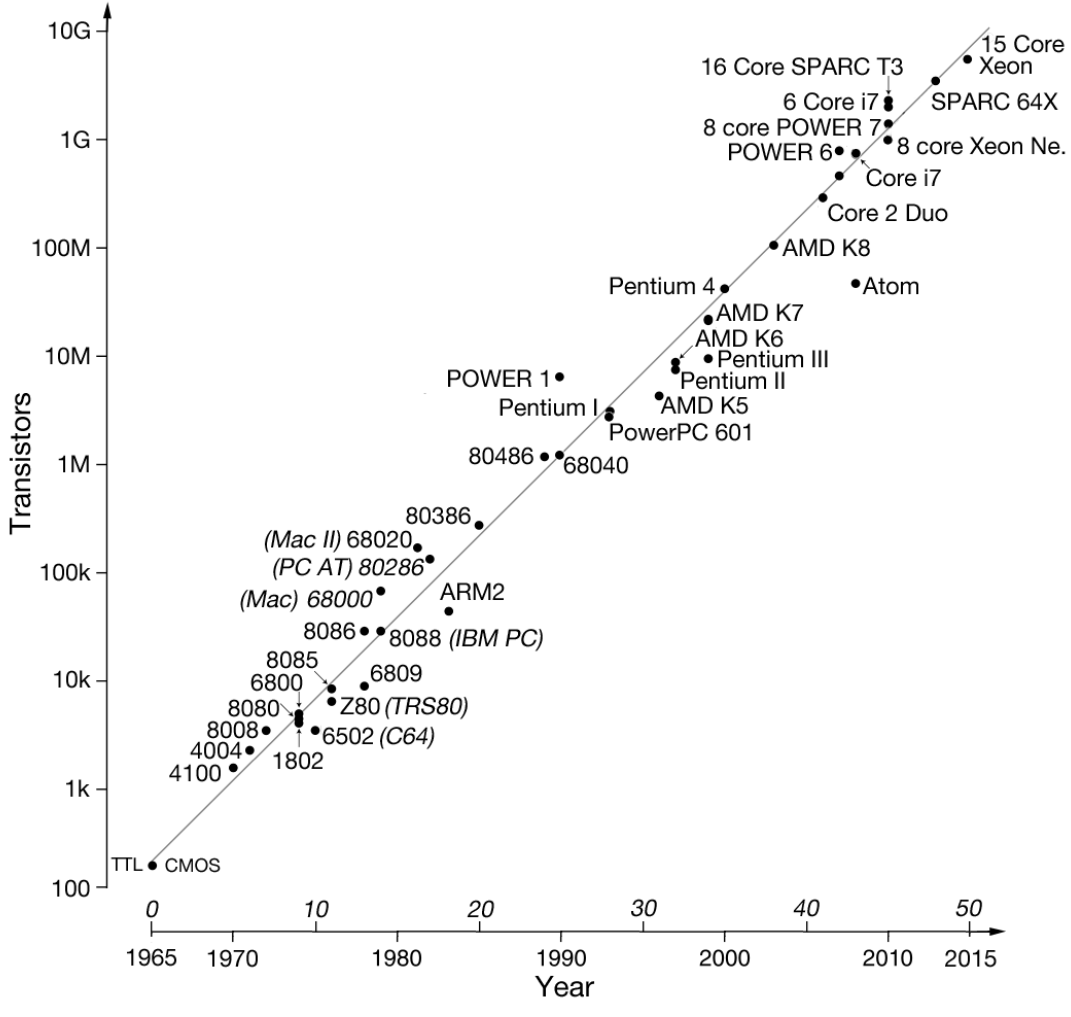
Ulrich Kühne

2016/2017

Motivation

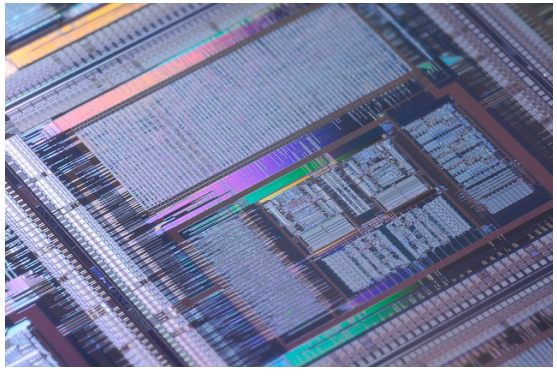


Motivation



[Source: www.elektormagazine.com/articles/moores-law]

Motivation

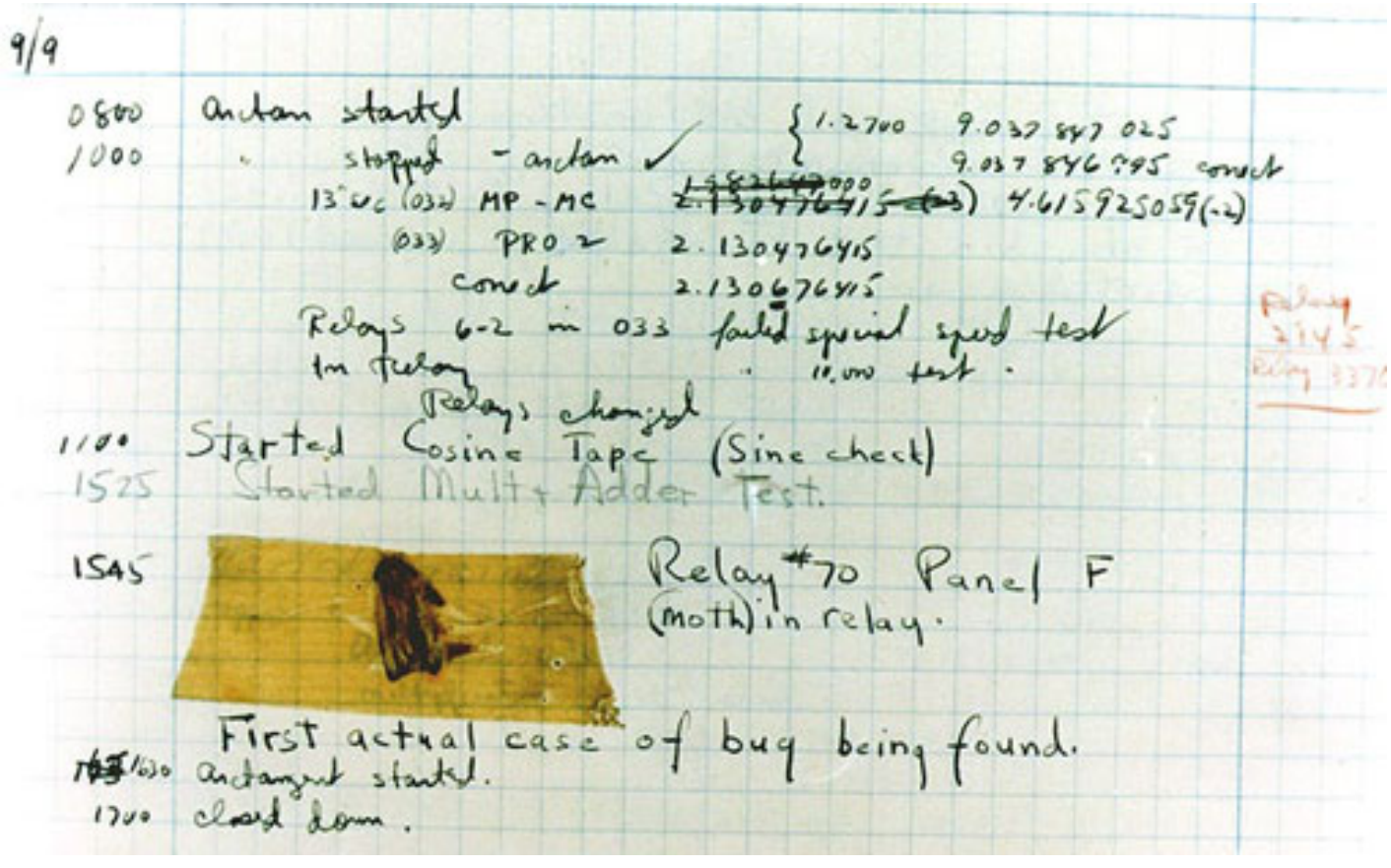


[© photo by mark.sze]

- Transistor count of > 3 billion
- Gate level models are huge
- Big designer teams (several hundreds)
- Big correctness issues
- Late bugs are extremely expensive

Motivation

The First Bug (1947)



[Photo: U.S. Naval Historical Center]

Motivation

The Pentium FDIV Bug (1994)

let $x = 4195835,\ y = 3145727 \quad \Rightarrow \quad x - \frac{x}{y} \cdot y = 256$

- Bug in floating point unit $\Rightarrow$ $ 450 Mio. loss for Intel

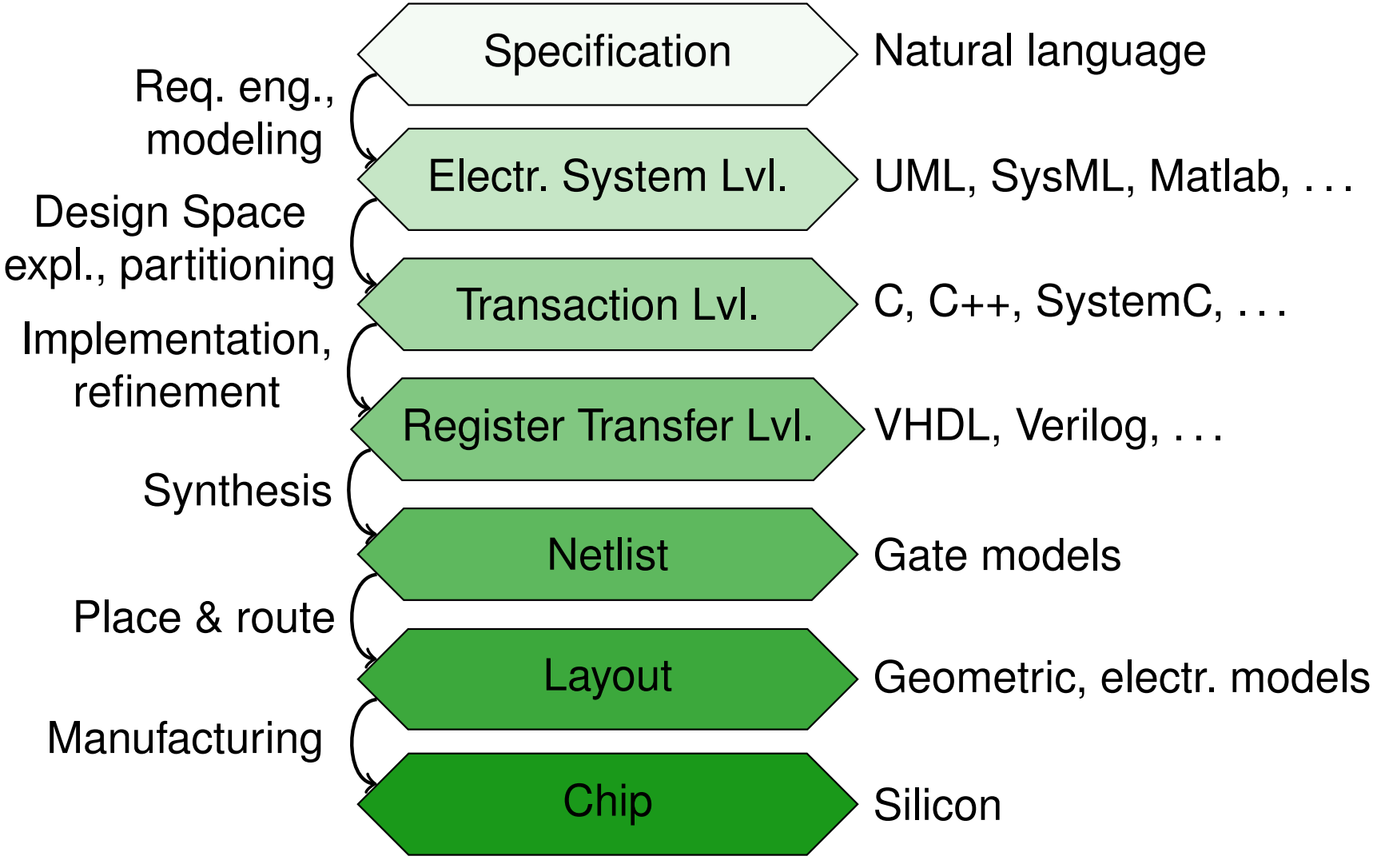
820 Chipset MTH Bug (2000)

- Error in memory translator hub
- Recall of around 1 Mio. motherboards
- $ 253 Mio. financial loss

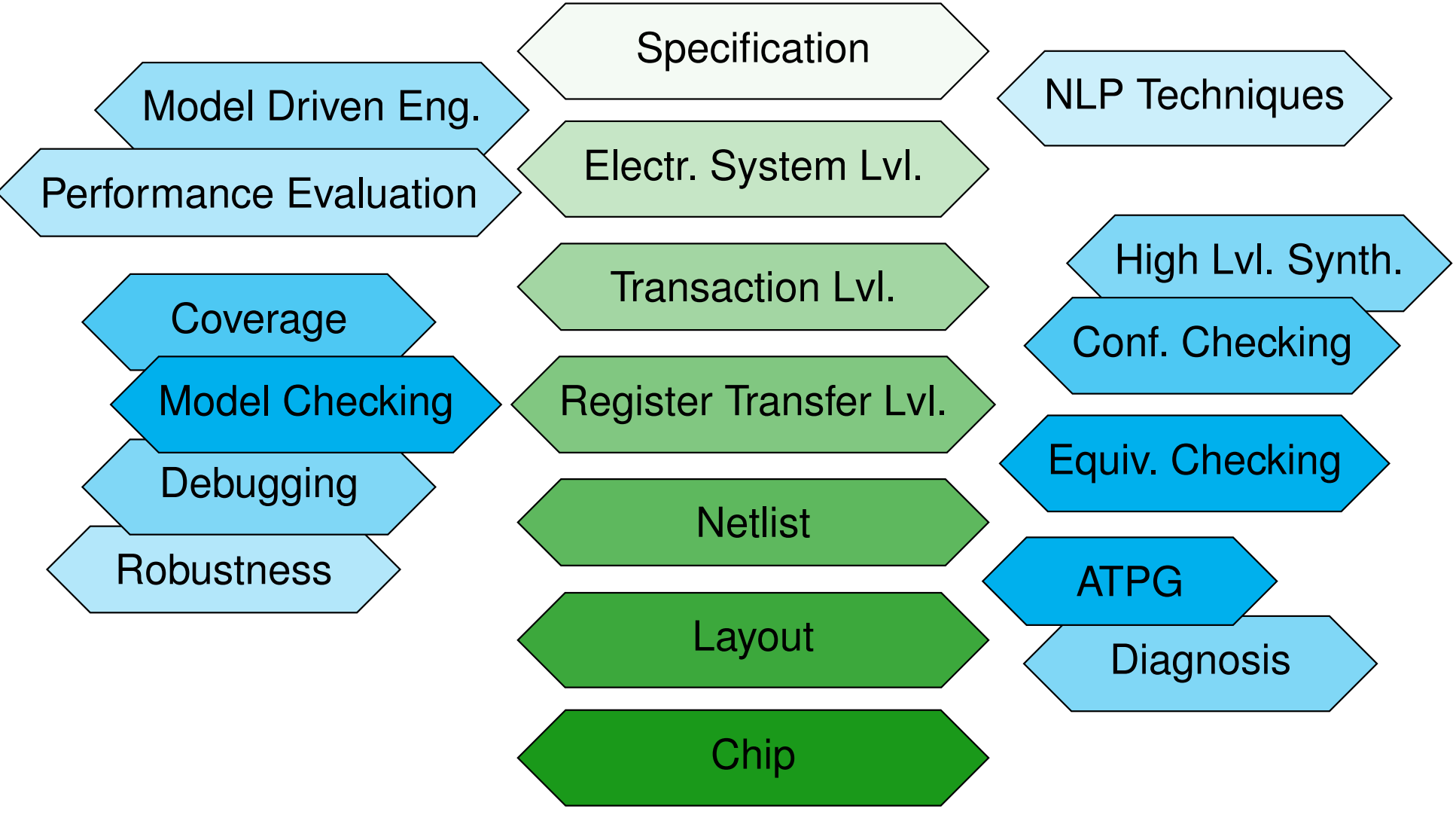
AMD Stack Pointer Bug (2012)

- Specific instruction sequence causes wrong stack pointer update
- Found by linux developer and reproduced on 48 core system

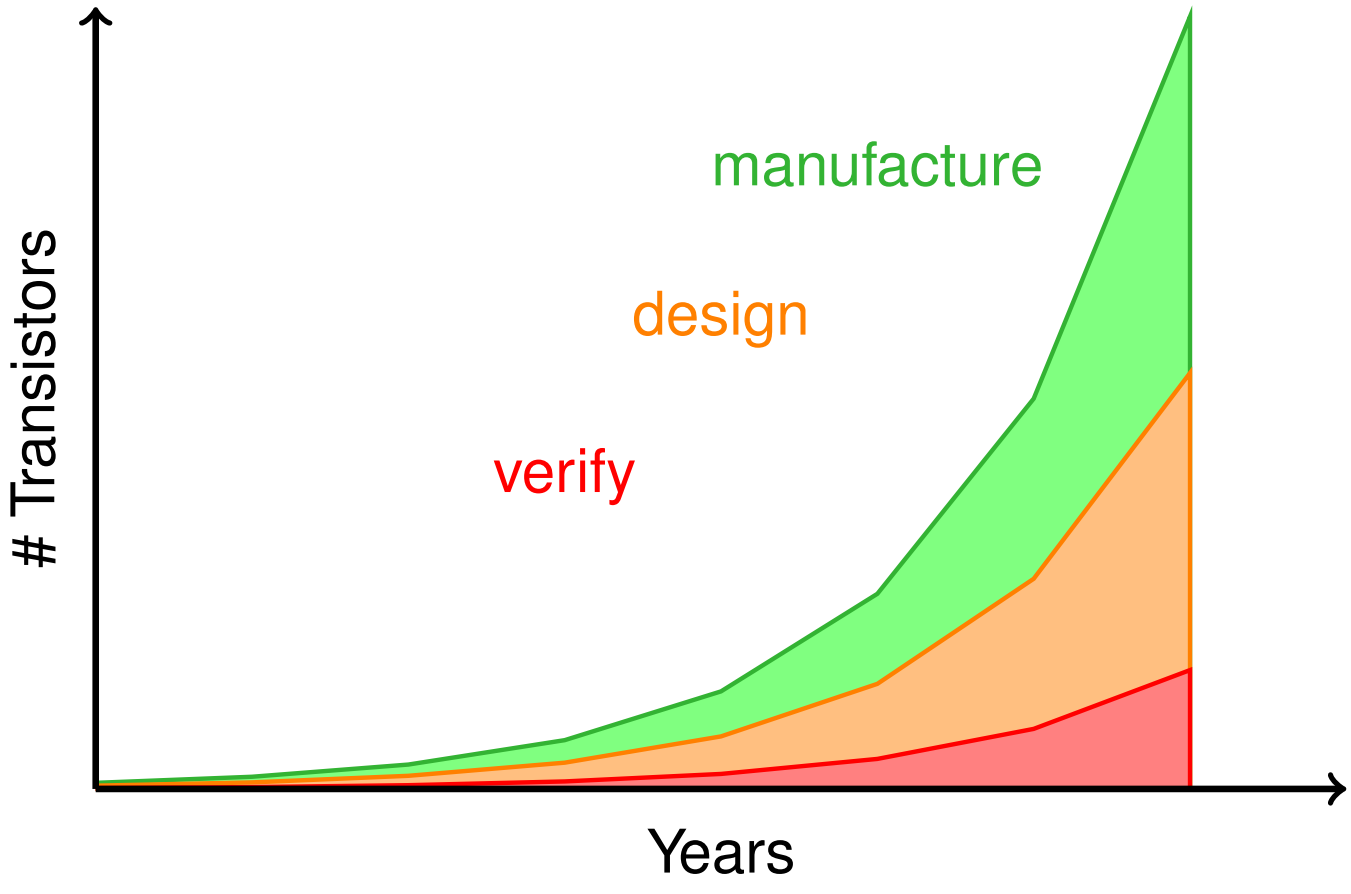
Levels of Abstraction



Quality Assurance



Design Gap – Verification Gap

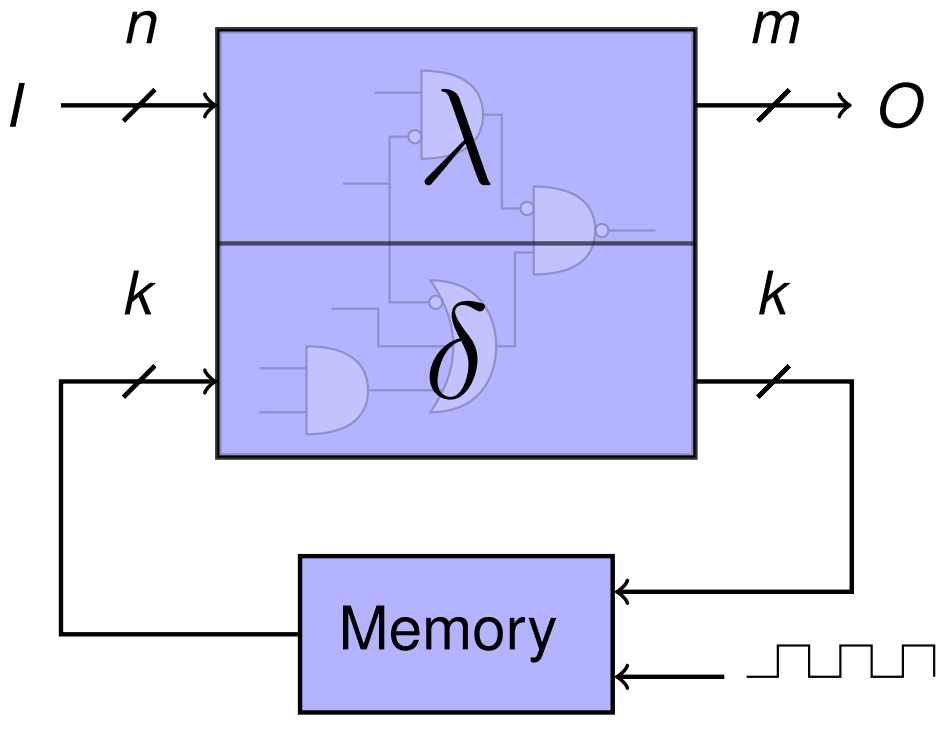


Outline

Functional Verification

- Dynamic verification (= simulation) still standard technology
- Pentium 4 overall simulated cycles < one minute at operation speed [Bentley, 2005]
- Full coverage is infeasible
- Increasing use of formal methods

Sequential Circuit Model



Mealy Machine:

$$\mathcal{M} = (I, O, S, S_0, \delta, \lambda)$$

$$\delta : S \times I \rightarrow S$$
$$\lambda : S \times I \rightarrow O$$
$$S_0 \subseteq S$$

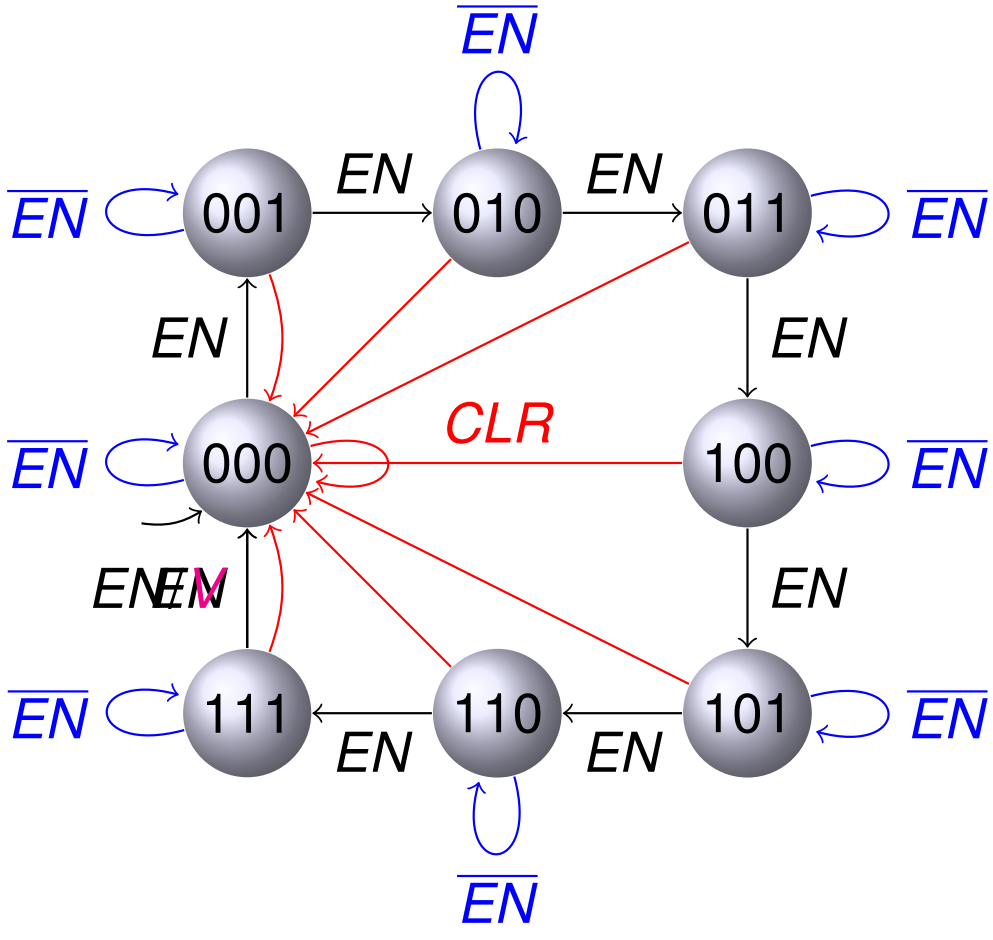
$$I = \{0,1\}^n$$
$$O = \{0,1\}^m$$
$$S = \{0,1\}^k$$

```
module count(CLK, EN, CLR,
             S0, S1, S2, V);

   input      CLK, EN, CLR;
   output reg S0, S1, S2;
   output     V;

   assign V = S0 & S1 & S2 &
              !CLR & EN;

   always @(posedge CLK) begin
      if (CLR) begin
         {S2, S1, S0} <= 0;
      end else begin
         if (EN) begin
            {S2, S1, S0}
              <= {S2, S1, S0} + 1;
         end
      end
   end
endmodule // count
```

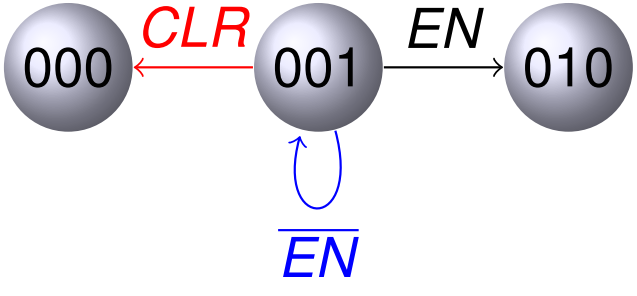


Verification Model

Mealy Machine:

$$\mathcal{M} = (I, O, S, S_0, \delta, \lambda)$$

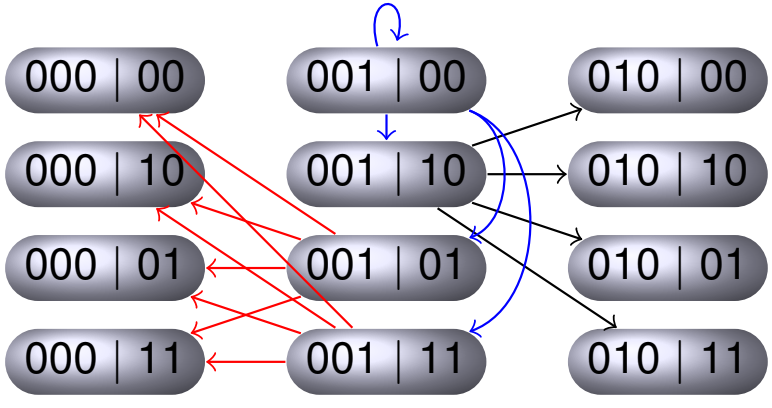
$$\delta : S \times I \to S$$
$$\lambda : S \times I \to O$$
$$S_0 \subseteq S$$

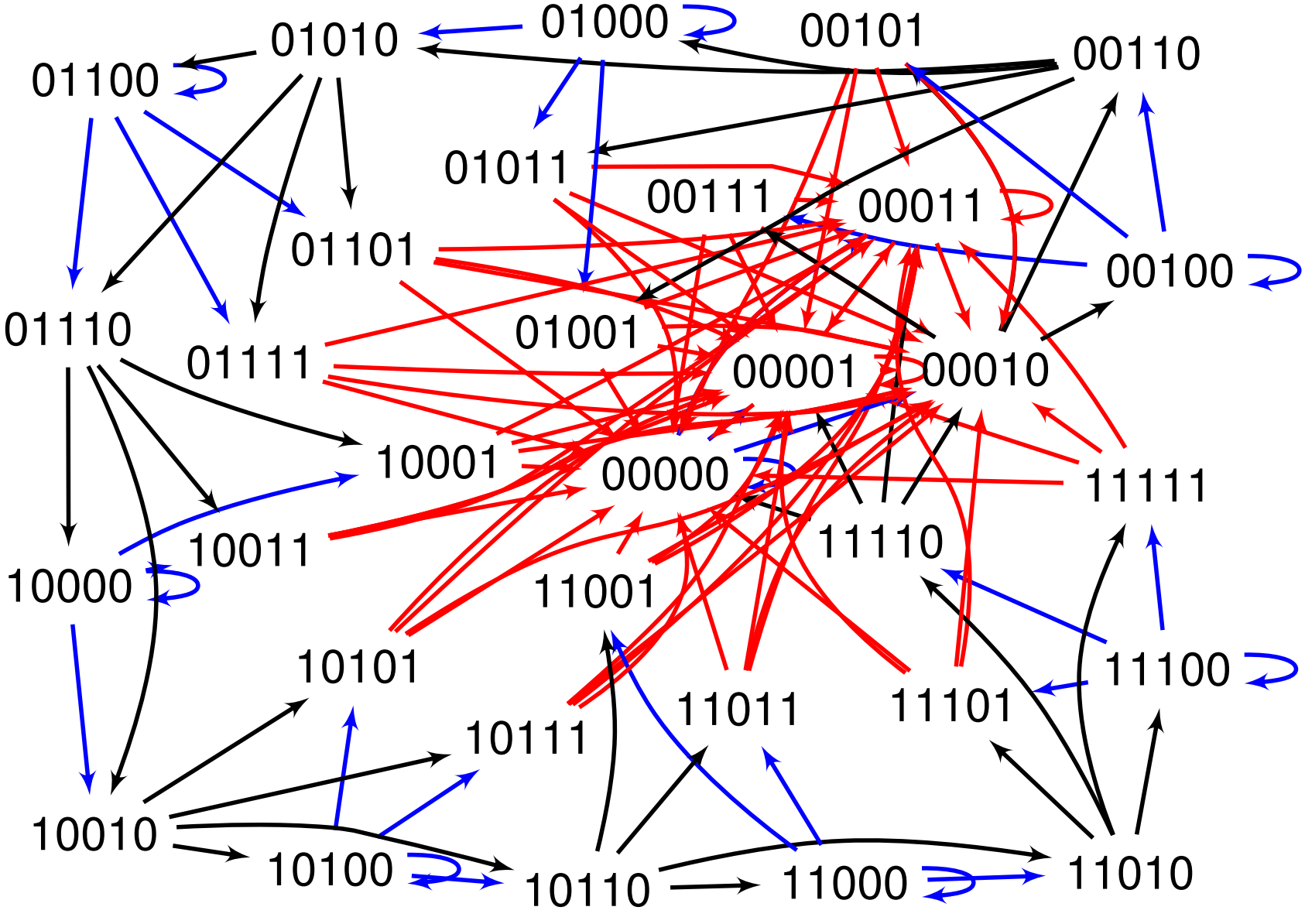


Kripke Structure:

$$\mathcal{K} = (S, S_0, \delta, \mathcal{V}, \mathcal{L})$$

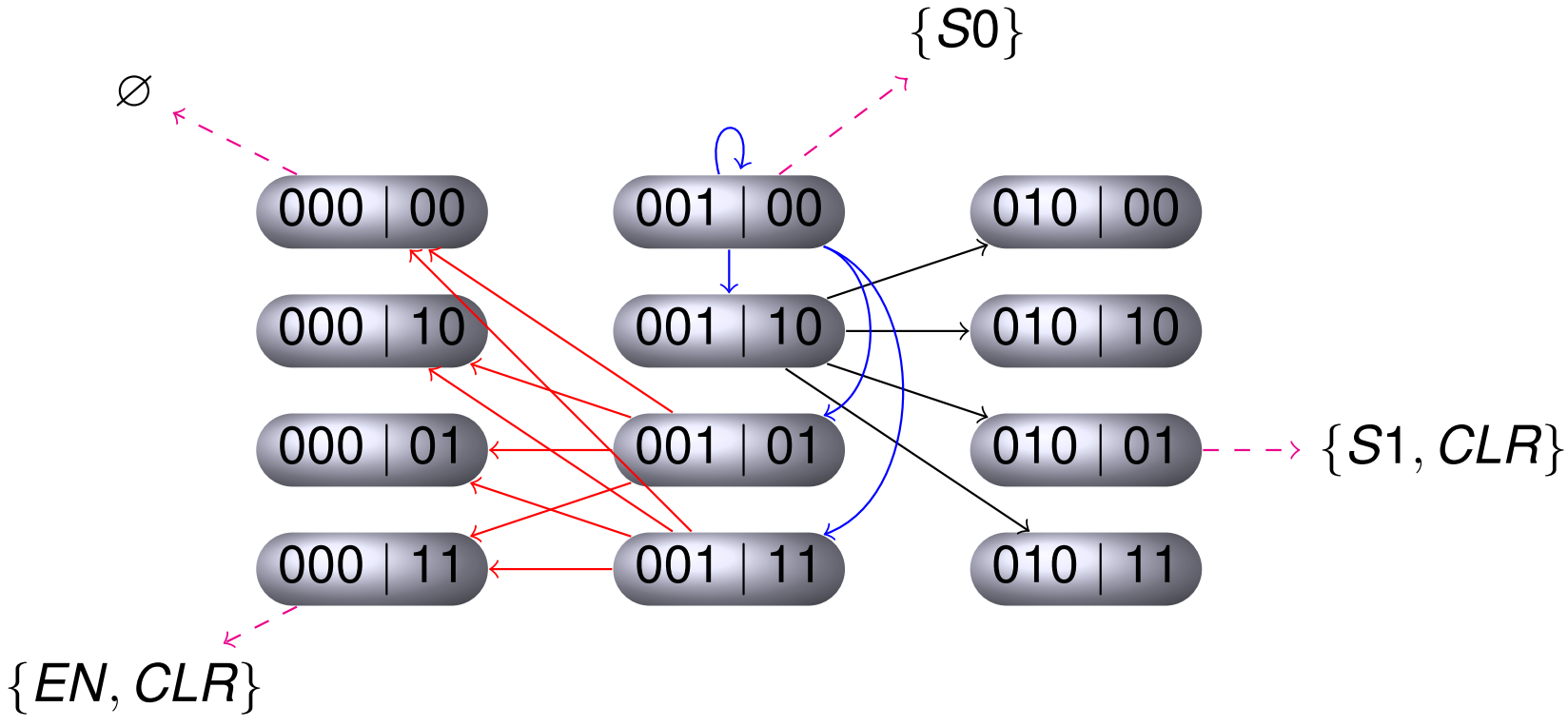
$\delta \subseteq S \times S$	transition relation
$S_0 \subseteq S$	initial states
$\mathcal{V}$	propositional variables
$\mathcal{L} : S \to 2^{\mathcal{V}}$	labelling function




01010
01000
00101
00110
01100
01011
00111
00011
01101
00100
01110
01001
01111
00001
00010
10001
00000
11111
10011
11110
10000
11001
10101
11100
11011
11101
10111
10010
10100
10110
11000
11010

Labelling Function

Propositional variables $\mathcal{V} = \{S2, S1, S0, EN, CLR, V\}$



What do we want to verify?

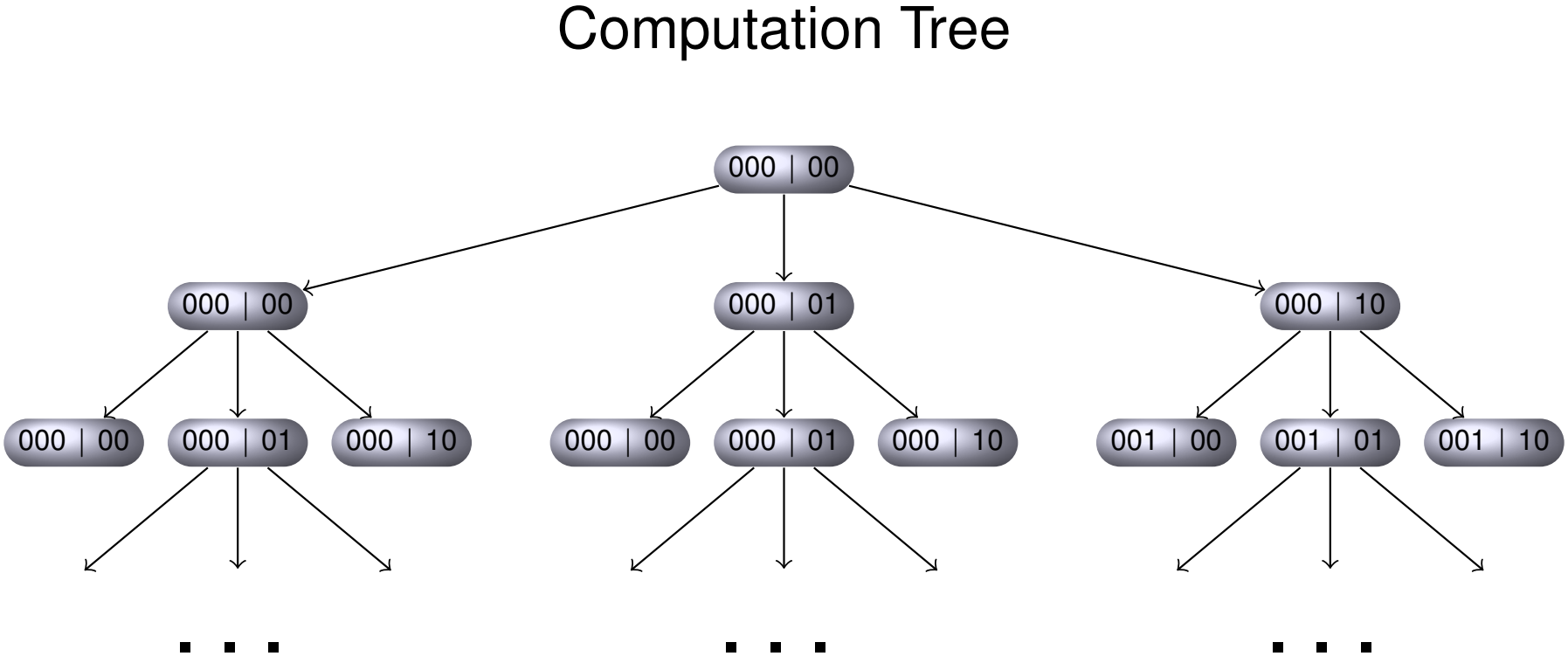
Safety

Something bad will never happen, e.g.
“The stack pointer will never overflow”
“The traffic lights will never be green at the same time”

Liveness

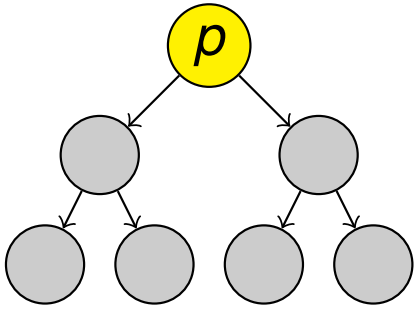
Something good will eventually happen, e.g.
“Every request will be granted”
“The cache and the main memory will eventually be consistent”

How to specify such properties?



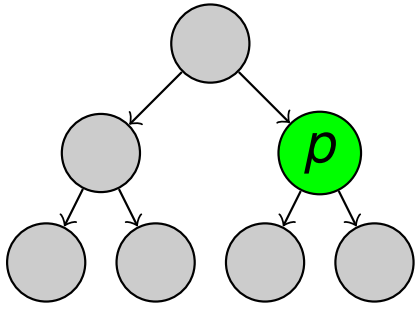
How to specify such properties?

Some property *p* holds
(in the initial state)



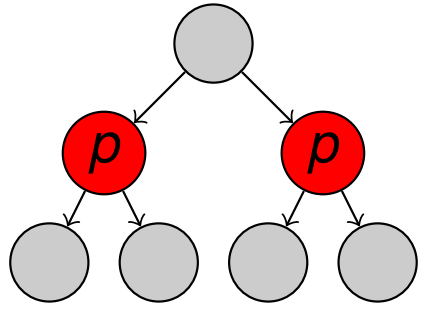
p

p holds in
some next state



EX p

p holds in
all next states



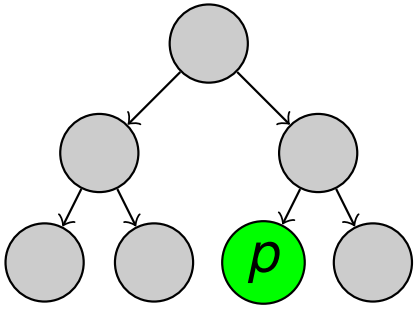
AX p

path
quantifier

next
operator

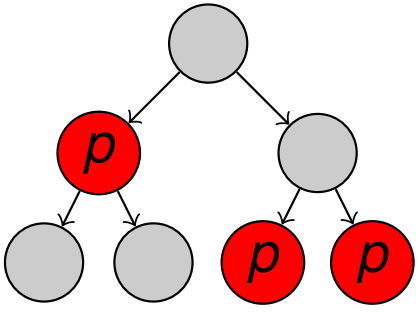
Further Modalities

p holds in some future state



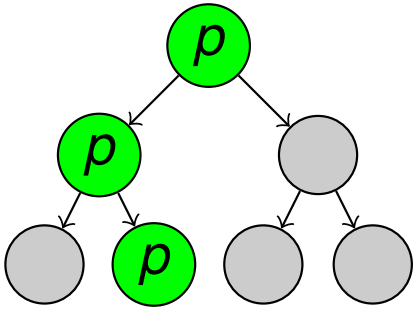
EF p

p holds eventually



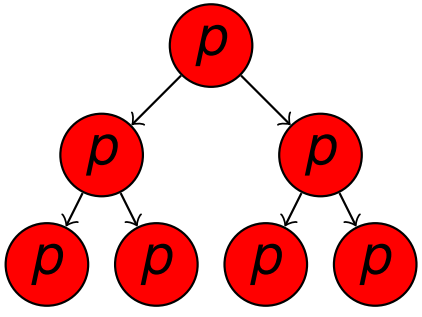
AF p

p holds globally on some path



EG p

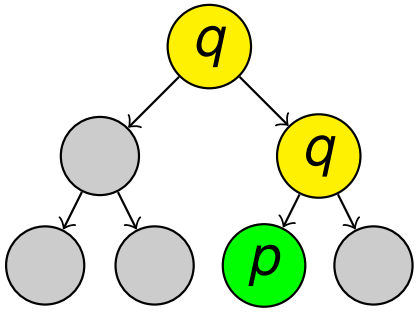
p holds globally on all paths



AG p

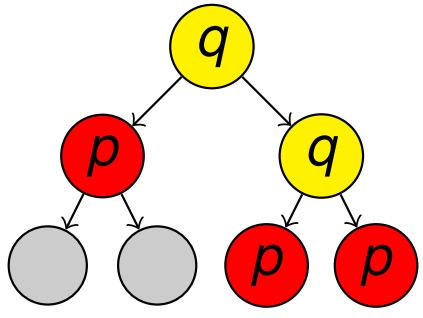
Until Modalities

On some path, q holds until p holds



$E(qUp)$

On all paths, q holds until p holds



$A(qUp)$

Computation Tree Logic

A CTL formula over propositional variables $\mathcal{V}$ has the form

$$
\begin{array}{rll}
\mathit{CTL} ::= & p, \quad \text{where } p \in \mathcal{V} & \\
\mid & \varphi \wedge \psi & \mid \quad \neg\varphi \\
\mid & \mathsf{EX}\ \varphi & \mid \quad \mathsf{AX}\ \varphi \\
\mid & \mathsf{EF}\ \varphi & \mid \quad \mathsf{AF}\ \varphi \\
\mid & \mathsf{EG}\ \varphi & \mid \quad \mathsf{AG}\ \varphi \\
\mid & \mathsf{E}(\varphi\ \mathsf{U}\ \psi) & \mid \quad \mathsf{A}(\varphi\ \mathsf{U}\ \psi)
\end{array}
$$

What do we want to verify?

Safety

"The stack pointer will never overflow" $\mathsf{AG}\ (\mathsf{sp} < 4096)$

"The traffic lights will never be green at the same time" $\neg\mathsf{EF}\ (\mathsf{tl}_1 \wedge \mathsf{tl}_2)$

Liveness

"Every request will be granted" $\mathsf{AG}\ (\mathsf{req} \rightarrow \mathsf{AF}\ \mathsf{gnt})$

"The cache and the main memory will eventually be consistent" $\mathsf{AF}\ (\mathsf{mem}_i = \mathsf{cache}_i)$

Model Checking

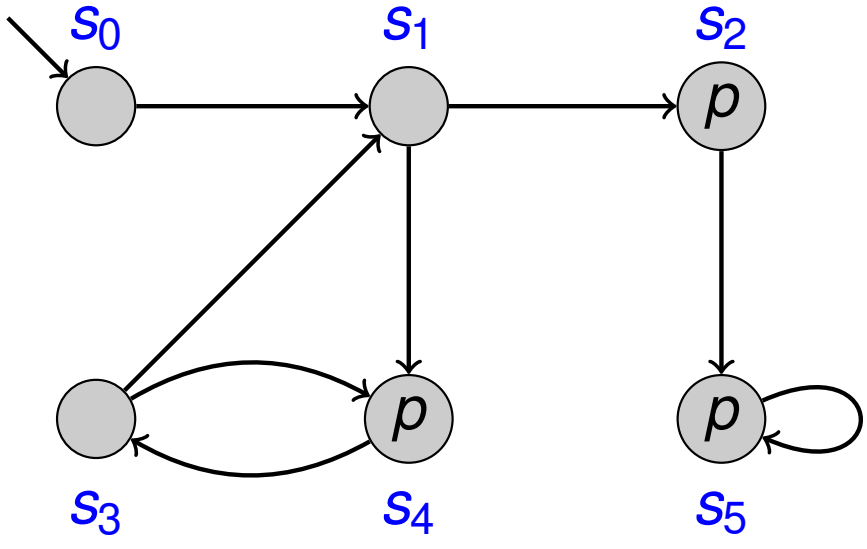
Model Checking

Given a Kripke Structure $\mathcal{K}$ and a CTL formula φ,
check if $\mathcal{K} \models \varphi$.

How do we do this?

1. Compute all states in which φ holds:
   $\tau(\varphi) = \{s \in S \mid \mathcal{K}, s \models \varphi\}$
2. Check if the initial states are a subset of those states:
   $S_0 \setminus \tau(\varphi) = \varnothing$

Example

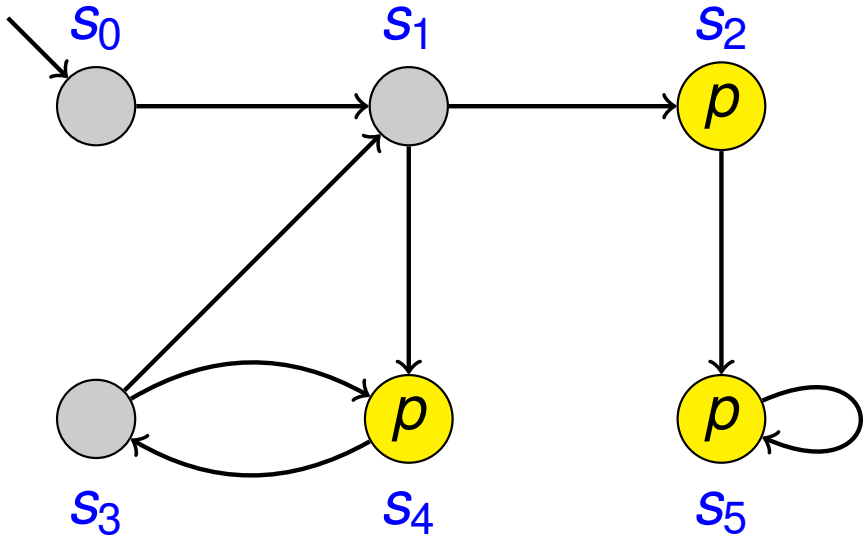


$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Example

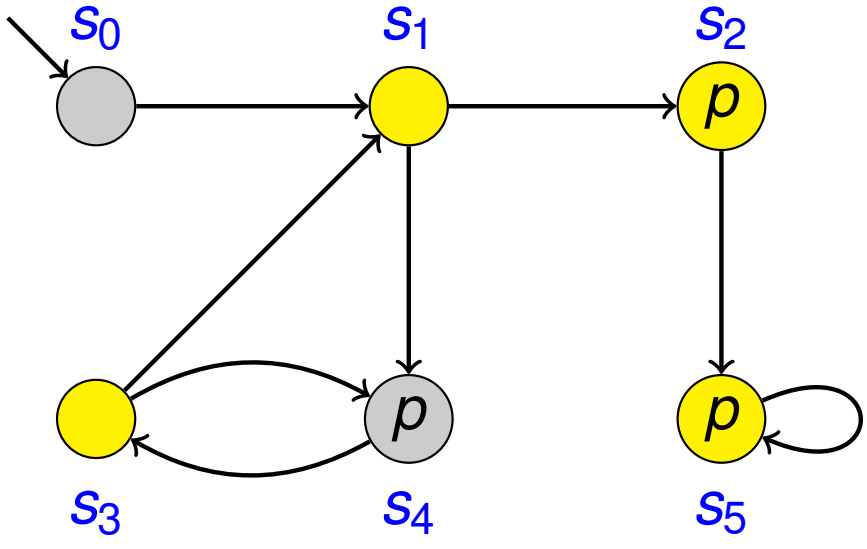


$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Example

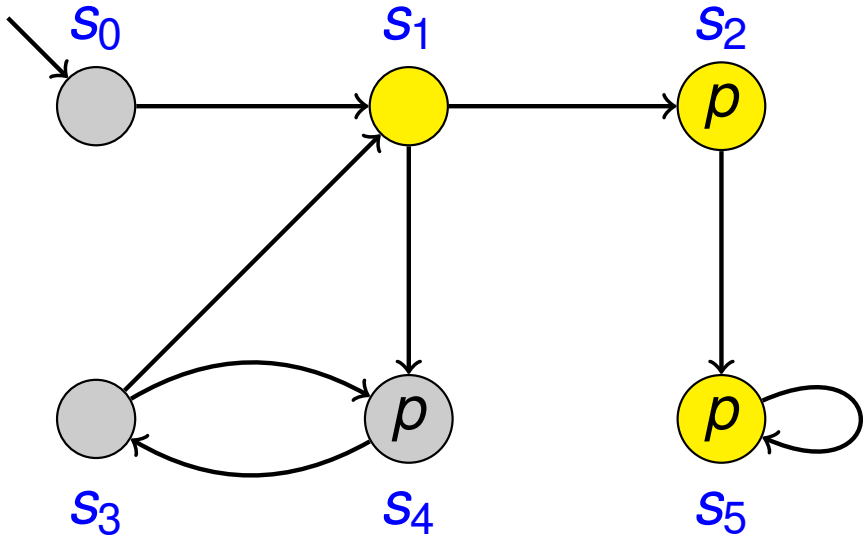


$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Example

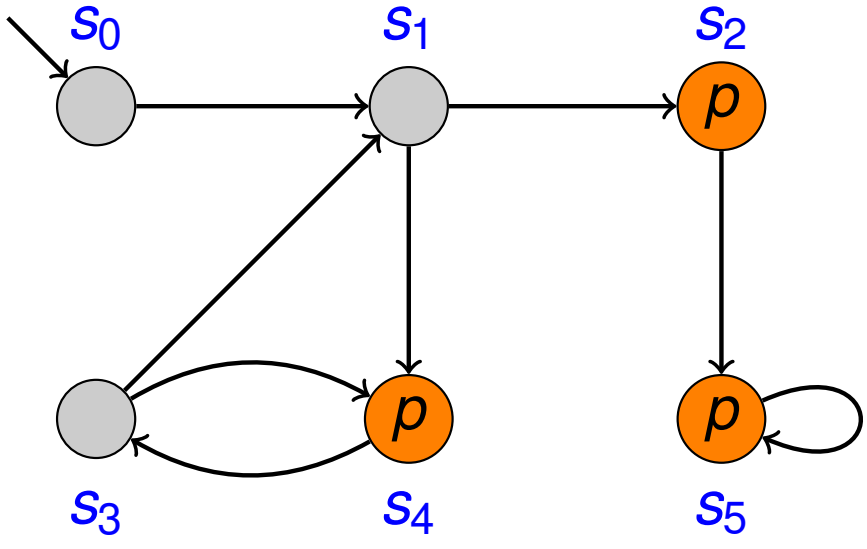


$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Example

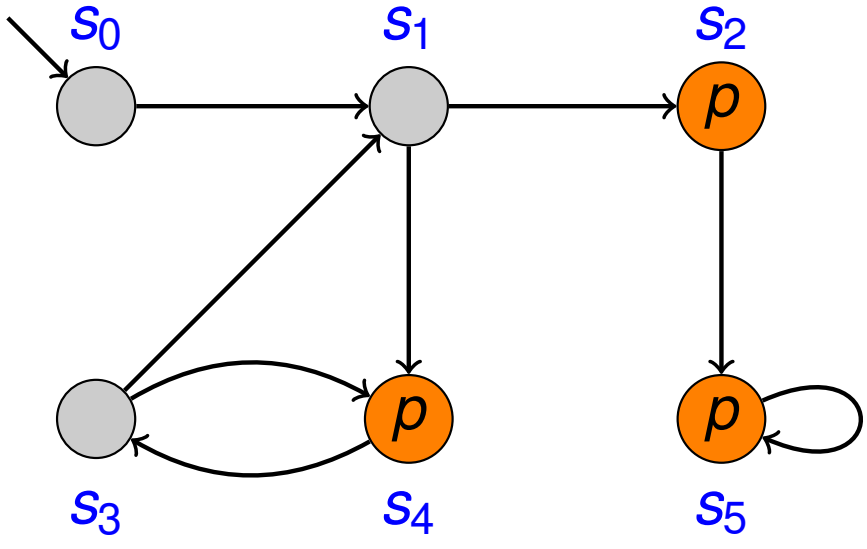


$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

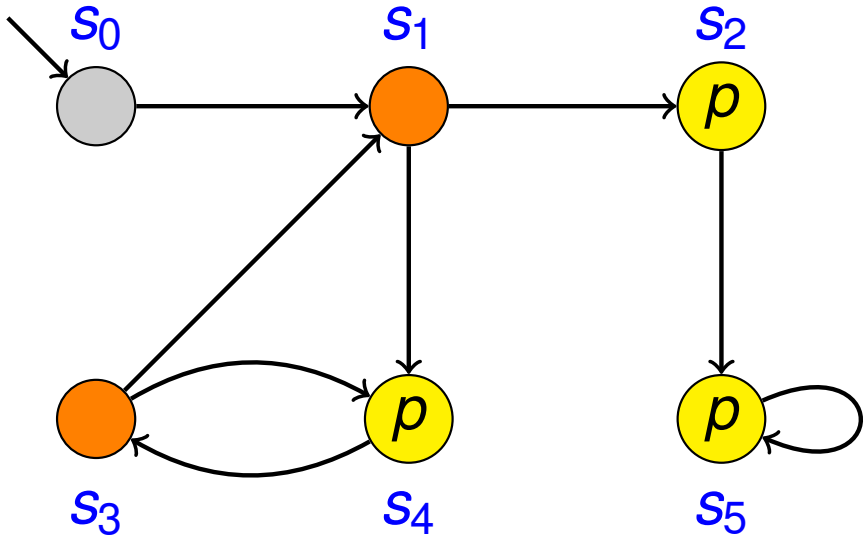
$$\tau(\mathrm{EF}\ p) = \{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) = \{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{EF}\ \varphi = \varphi \vee \mathrm{EX}\ \mathrm{EF}\ \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\text{EX } p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\text{AX } p) = \{s_1, s_2, s_5\}$$

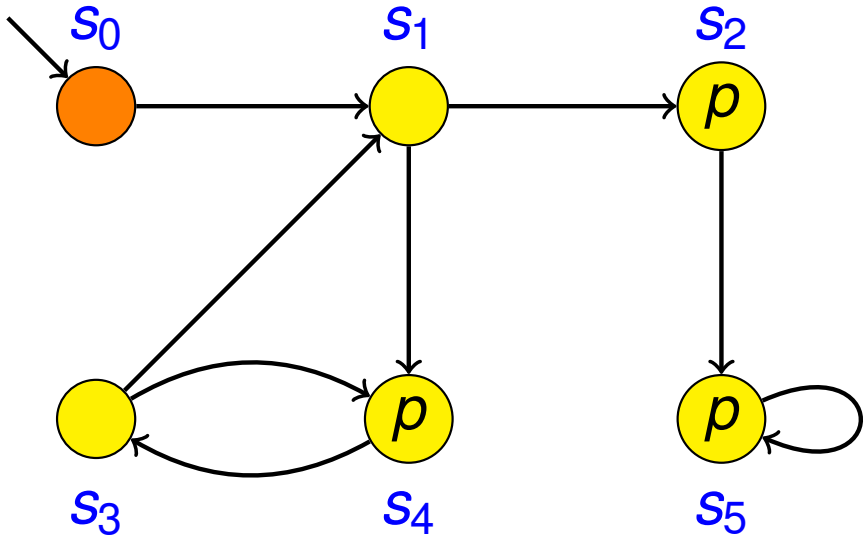
$$\tau(\text{EF } p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\text{AG } p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\text{EF } \varphi = \varphi \vee \text{EX EF } \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

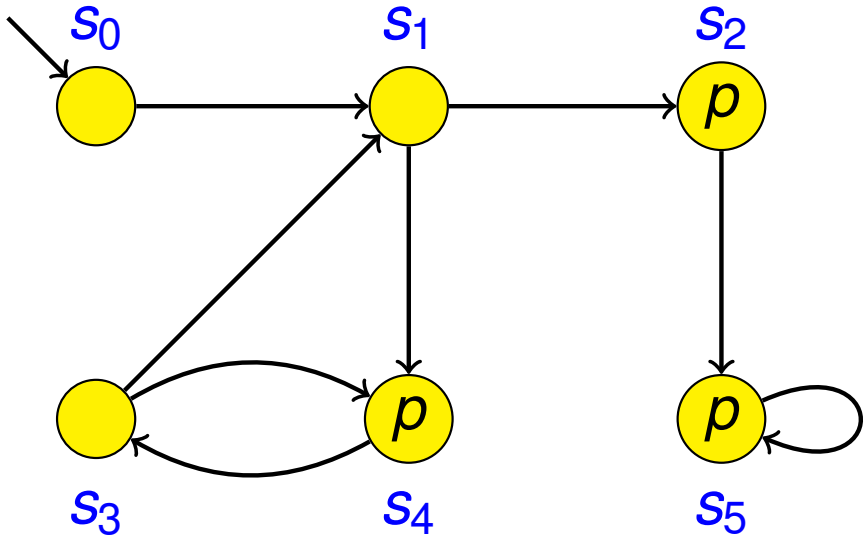
$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{EF}\ \varphi = \varphi \vee \mathrm{EX}\ \mathrm{EF}\ \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

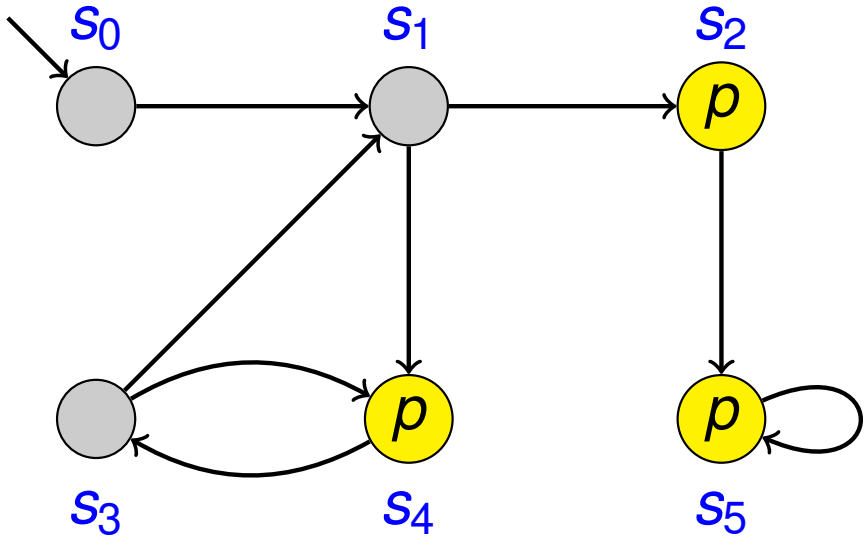
$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{EF}\ \varphi = \varphi \vee \mathrm{EX}\ \mathrm{EF}\ \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

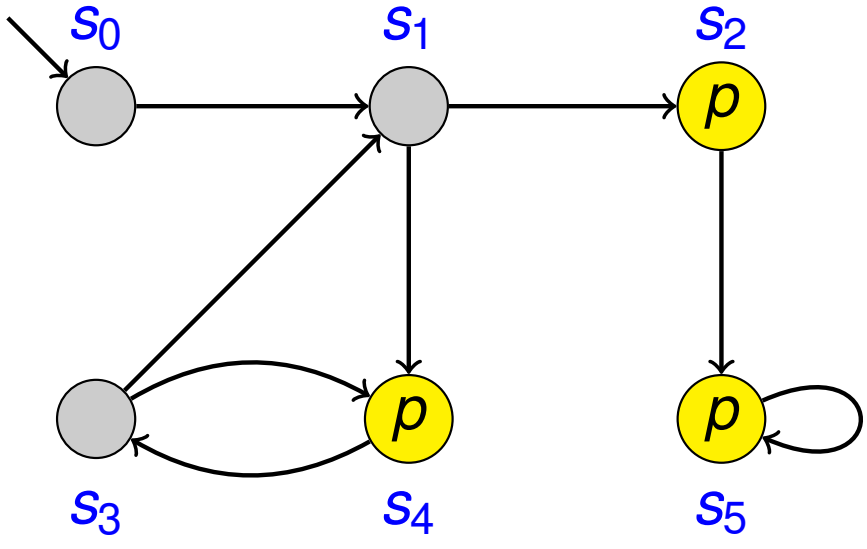
$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{AG}\ \varphi = \varphi \wedge \mathrm{AX}\ \mathrm{AG}\ \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

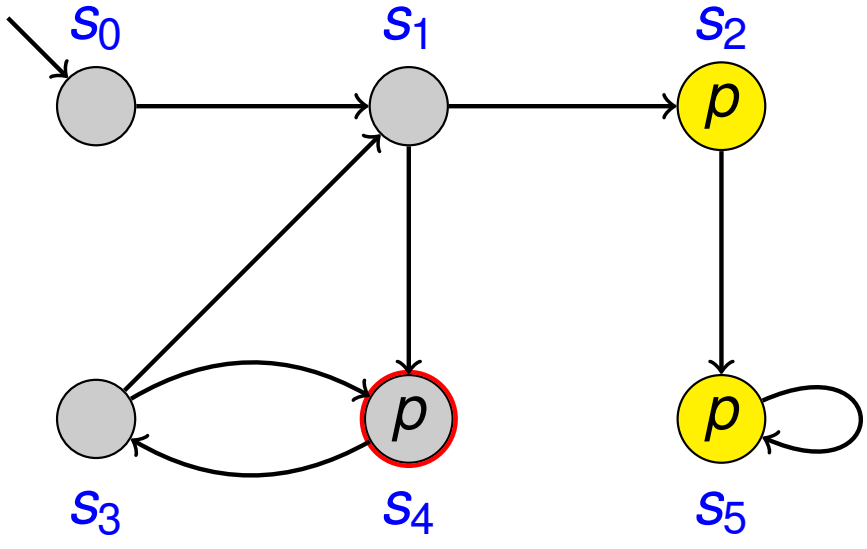
$$\tau(\mathrm{EF}\ p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{AG}\ \varphi = \varphi \wedge \mathrm{AX}\ \mathrm{AG}\ \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\text{EX } p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\text{AX } p) = \{s_1, s_2, s_5\}$$

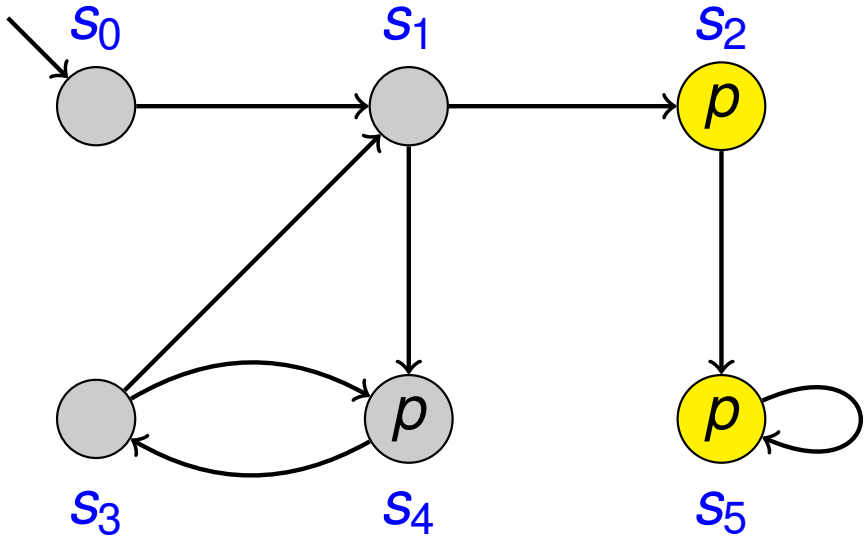
$$\tau(\text{EF } p) =$$
$$\{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\text{AG } p) =$$
$$\{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\text{AG } \varphi = \varphi \wedge \text{AX AG } \varphi$$

Example



$$\tau(p) = \{s_2, s_4, s_5\}$$
$$\tau(\mathrm{EX}\ p) = \{s_1, s_2, s_3, s_5\}$$
$$\tau(\mathrm{AX}\ p) = \{s_1, s_2, s_5\}$$

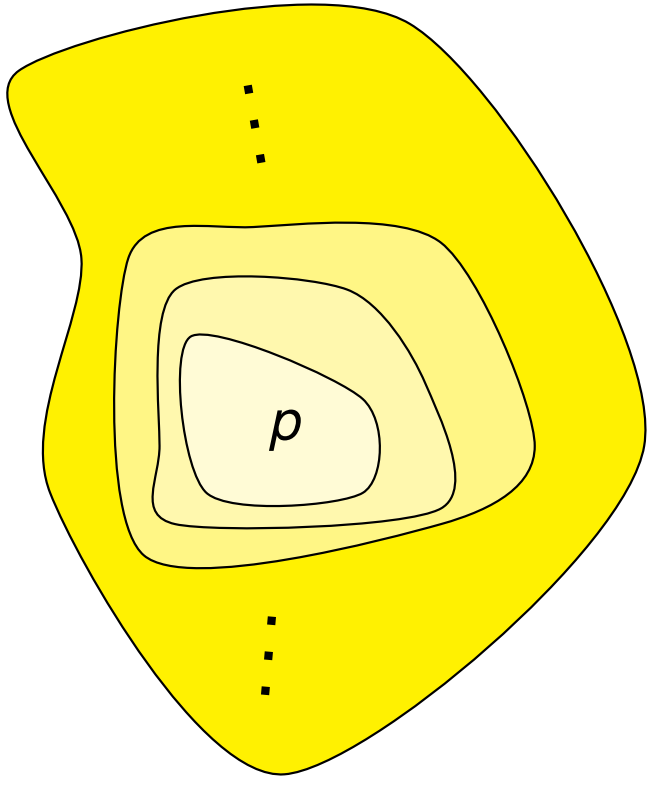
$$\tau(\mathrm{EF}\ p) = \{s_2, s_4, s_5\} \cup \{s_1, s_3\} \cup \{s_0\}$$

$$\tau(\mathrm{AG}\ p) = \{s_2, s_4, s_5\} \cap \{s_2, s_5\}$$

Expansion rule:

$$\mathrm{AG}\ \varphi = \varphi \wedge \mathrm{AX}\ \mathrm{AG}\ \varphi$$

Fixed Point Algorithm for EF p



$$S_0 = p$$
$$S_1 = p \cup \text{EX}\ p$$
$$S_2 = p \cup \text{EX}\ p \cup \text{EX EX}\ p$$
$$\ldots$$
$$S_n = p \cup \bigcup_{i=1}^{n} \text{EX}^i p = S_{n-1}$$

$$\Rightarrow S_n = \tau(\text{EF}\ p)$$

Fixed Points

Let $f : \mathcal{P}(S) \to \mathcal{P}(S)$ a set-valued function and $Z \subseteq G$.

- Z is called a fixed point of f if $f(Z) = Z$
- Z is the least fixed point of f it is a fixed point and for all other fixed points U of f it holds that $Z \subseteq U$.
- Z is the greatest fixed point of f it is a fixed point and for all other fixed points U of f it holds that $U \subseteq Z$.

Fixed Points (2)

A function $f : \mathcal{P}(S) \to \mathcal{P}(S)$ is called monotone if for all $X, Y \subseteq S$

$$X \subseteq Y \Rightarrow f(X) \subseteq f(Y) \tag{1}$$

Knaster-Tarski Theorem

Let $f : \mathcal{P}(S) \to \mathcal{P}(S)$ be a monotone function. Then f has a least and a greatest fixed point.

- $\bigcup_{n \geq 1} f^n(\varnothing)$ is the least fixed point of f.
- $\bigcap_{n \geq 1} f^n(S)$ is the greatest fixed point of f.

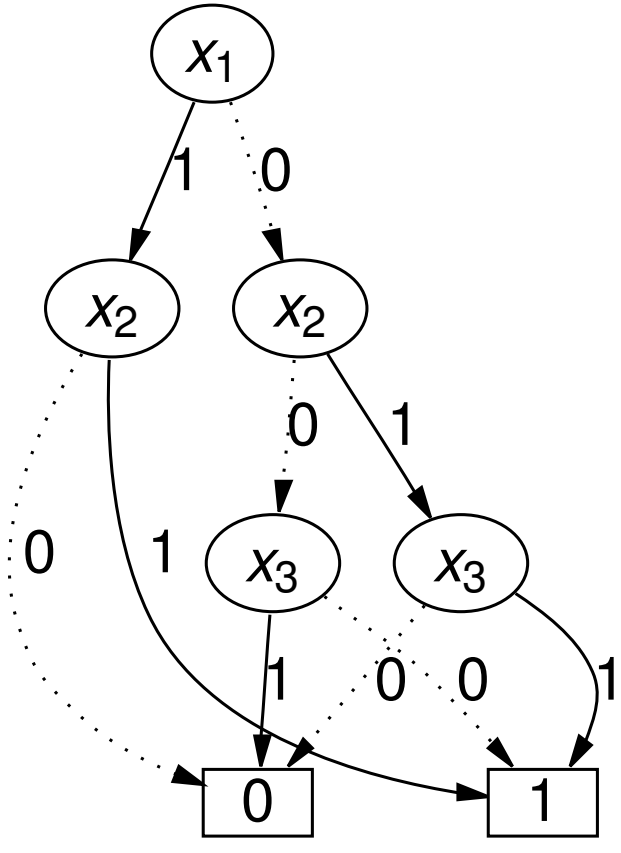
CTL Model Checking

Let $\mathcal{K} = (S, S_0, \delta, \mathcal{V}, \mathcal{L})$ be a Kripke structure.

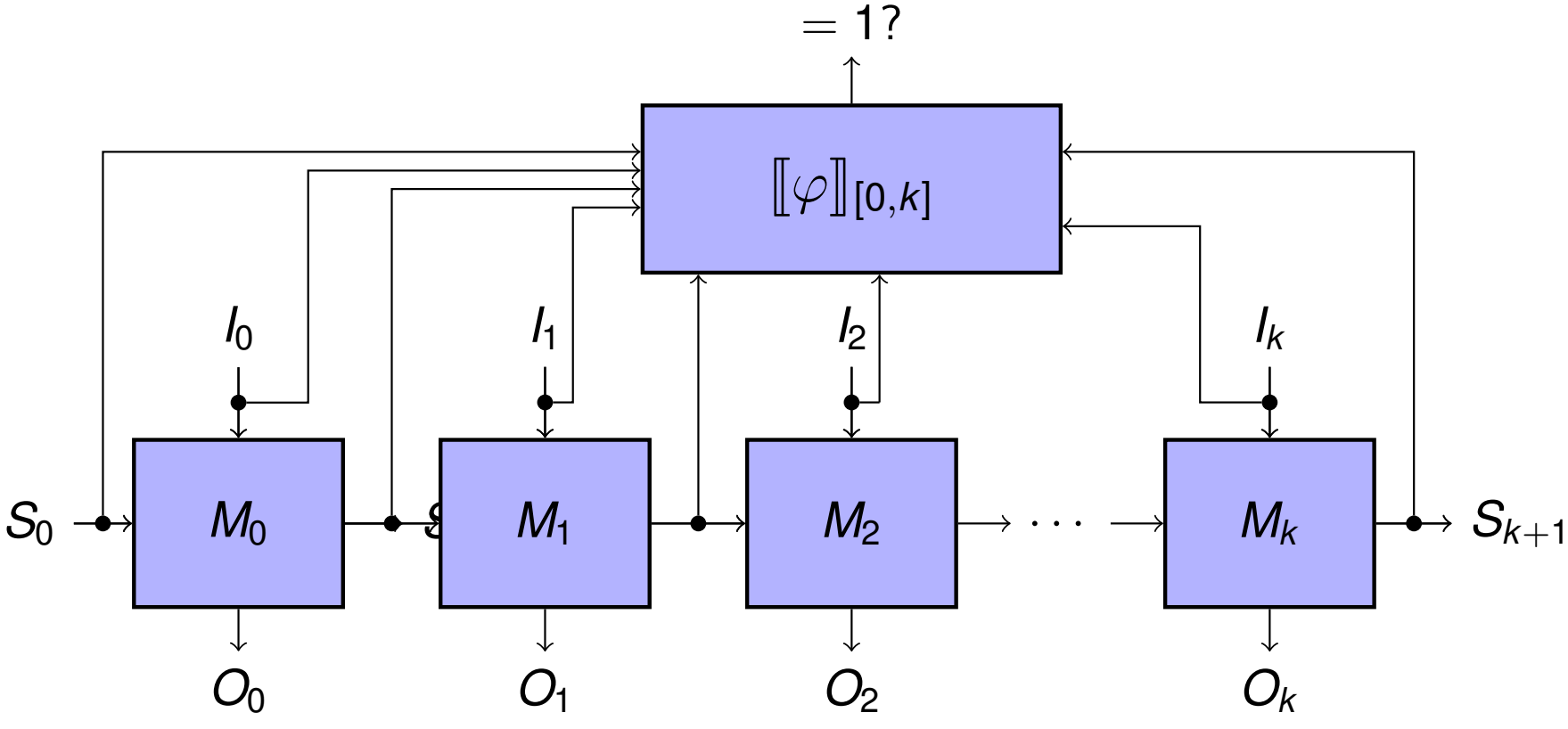
$$
\begin{array}{lcl}
\tau(p) & = & \{s \in S \mid p \in \mathcal{L}(s)\} \\
\tau(\varphi \wedge \psi) & = & \tau(\varphi) \cap \tau(\psi) \\
\tau(\varphi \vee \psi) & = & \tau(\varphi) \cup \tau(\psi) \\
\tau(\neg\varphi) & = & S \setminus \tau(\varphi) \\
\tau(\mathsf{EF}\ \varphi) & = & \mathsf{lfp}Z.\ \tau(\varphi) \cup \mathsf{EX}\ (Z) \\
\tau(\mathsf{AF}\ \varphi) & = & \mathsf{lfp}Z.\ \tau(\varphi) \cup \mathsf{AX}\ (Z) \\
\tau(\mathsf{EG}\ \varphi) & = & \mathsf{gfp}Z.\ \tau(\varphi) \cap \mathsf{EX}\ (Z) \\
\tau(\mathsf{AG}\ \varphi) & = & \mathsf{gfp}Z.\ \tau(\varphi) \cap \mathsf{AX}\ (Z) \\
\tau(\mathsf{E}(\varphi\ \mathsf{U}\ \psi)) & = & \mathsf{lfp}Z.\ \tau(\psi) \cup (\tau(\varphi) \cap \mathsf{EX}\ (Z)) \\
\tau(\mathsf{A}(\varphi\ \mathsf{U}\ \psi)) & = & \mathsf{lfp}Z.\ \tau(\psi) \cup (\tau(\varphi) \cap \mathsf{AX}\ (Z))
\end{array}
$$

Symbolic Model Checking

- Complexity of CTL model checking depending on state space
- Use of symbolic representations
- Binary Decision Diagrams (BDDs)
- State space explosion still a problem
- Works for small (or very regular) systems
- Popular tool: NuSMV [Cimatti et al., 2002]

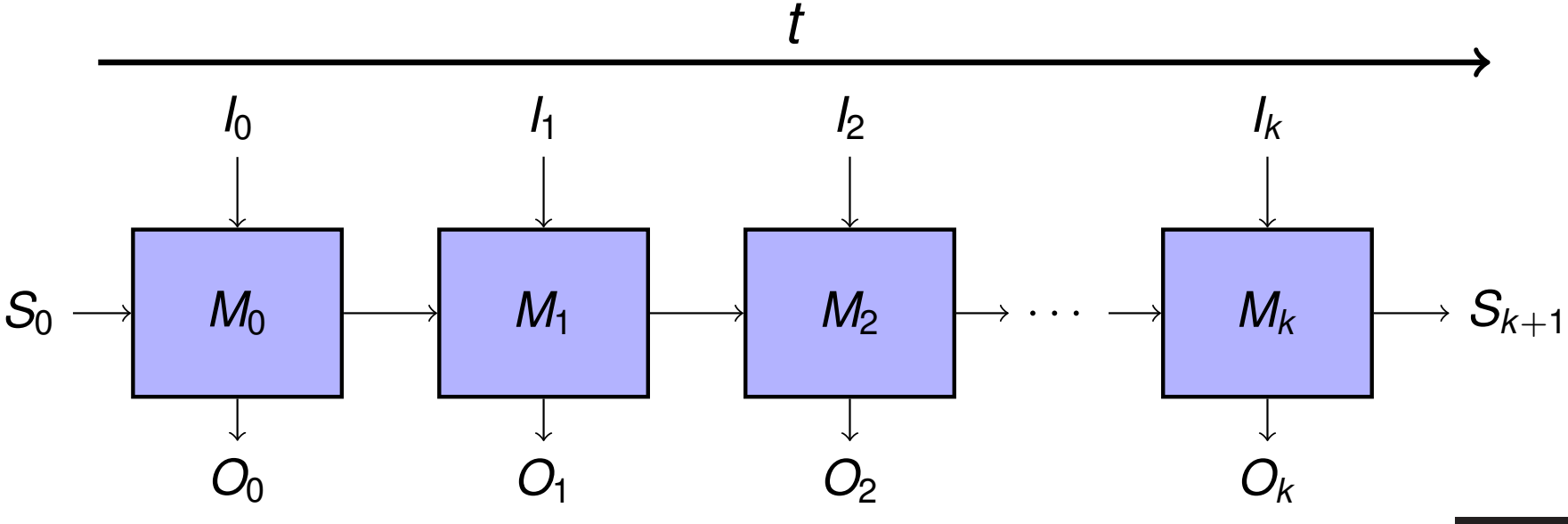


Bounded Model Checking



Bounded Model Checking

- Avoid reachability computation by unrolling
- Encode bounded property
- Works well for safety checking
- Original method is incomplete (bug hunting only)
- Alternatively, use temporal induction



Advancements in Model Checking

- Symbolic model checking with BDDs [Burch et al., 1992]
- SAT-based bounded model checking [Biere et al., 1999]
- Counter-example guided abstraction refinement [Clarke et al., 2003]
- Inductive invariant checking [Bradley, 2011]

Outline

Boolean Satisfiability (SAT)

SAT Problem

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ (in conjunctive normal form), is there an assignment $X \in \{0,1\}^n$, such that $f(X) = 1$?

Conjunctive Normal Form

A Boolean formula over variables $X = \{x_0 \dots x_n\}$ is in conjunctive normal form if it is a conjunction of clauses $(\ell_{1,1} \vee \ell_{1,2} \vee \cdots \vee \ell_{1,m_0}) \wedge \cdots \wedge (\ell_{k,1} \vee \cdots \vee \ell_{k,m_k})$. A clause is a disjunction of literals $\ell = x_i$ or $\ell = \neg x_i$ for some $x_i \in X$.

- NP-complete problem [Cook, 1971]

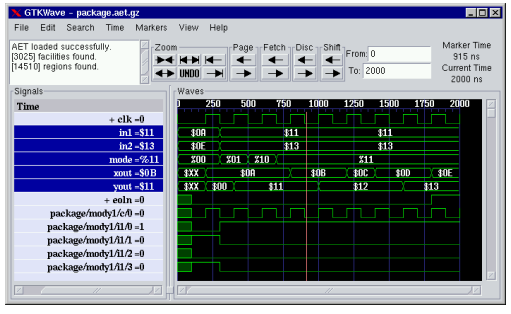
SAT-Based BMC

$\mathcal{M} \models \varphi$

CNF

SAT Solver

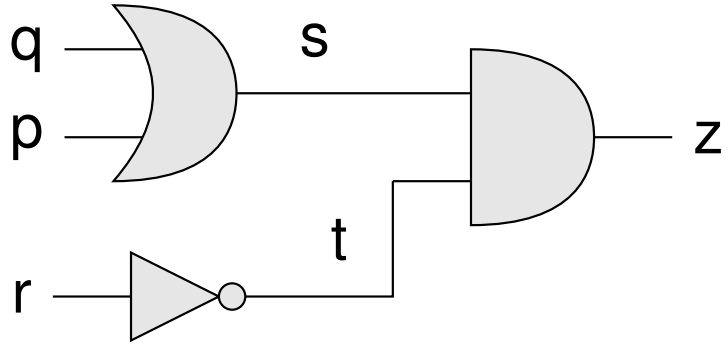
Counterexample



SAT

UNSAT

Tseitin Transformation



$$z \leftrightarrow (p \vee q) \wedge \neg r$$

$$s \leftrightarrow p \vee q$$
$$t \leftrightarrow \neg r$$
$$z \leftrightarrow s \wedge t$$

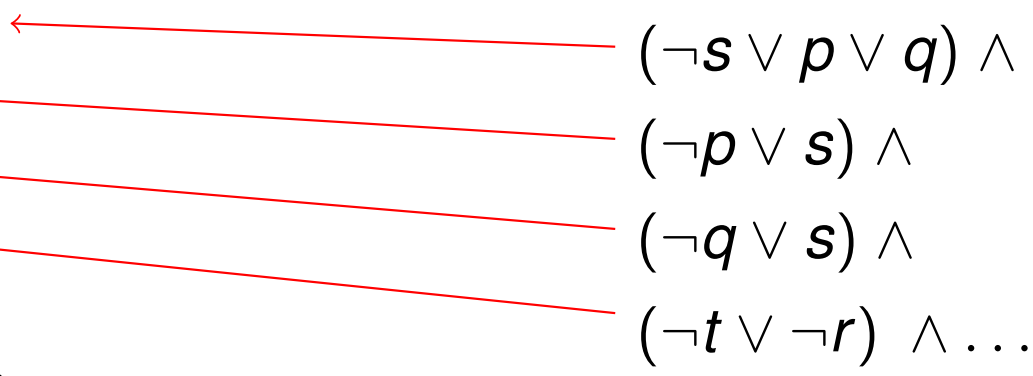
$$\begin{aligned}
s \leftrightarrow p \vee q &\equiv (s \rightarrow p \vee q) \wedge (p \vee q \rightarrow s) \\
&\equiv (\neg s \vee p \vee q) \wedge (\neg(p \vee q) \vee s) \\
&\equiv (\neg s \vee p \vee q) \wedge ((\neg p \wedge \neg q) \vee s) \\
&\equiv (\neg s \vee p \vee q) \wedge (\neg p \vee s) \wedge (\neg q \vee s)
\end{aligned}$$

$$t \leftrightarrow \neg r \equiv (\neg t \vee \neg r) \wedge (t \vee r)$$

$$z \leftrightarrow s \wedge t \equiv (\neg s \vee \neg t \vee z) \wedge (s \vee \neg z) \wedge (t \vee \neg z)$$

How it Looks Like in Practice...

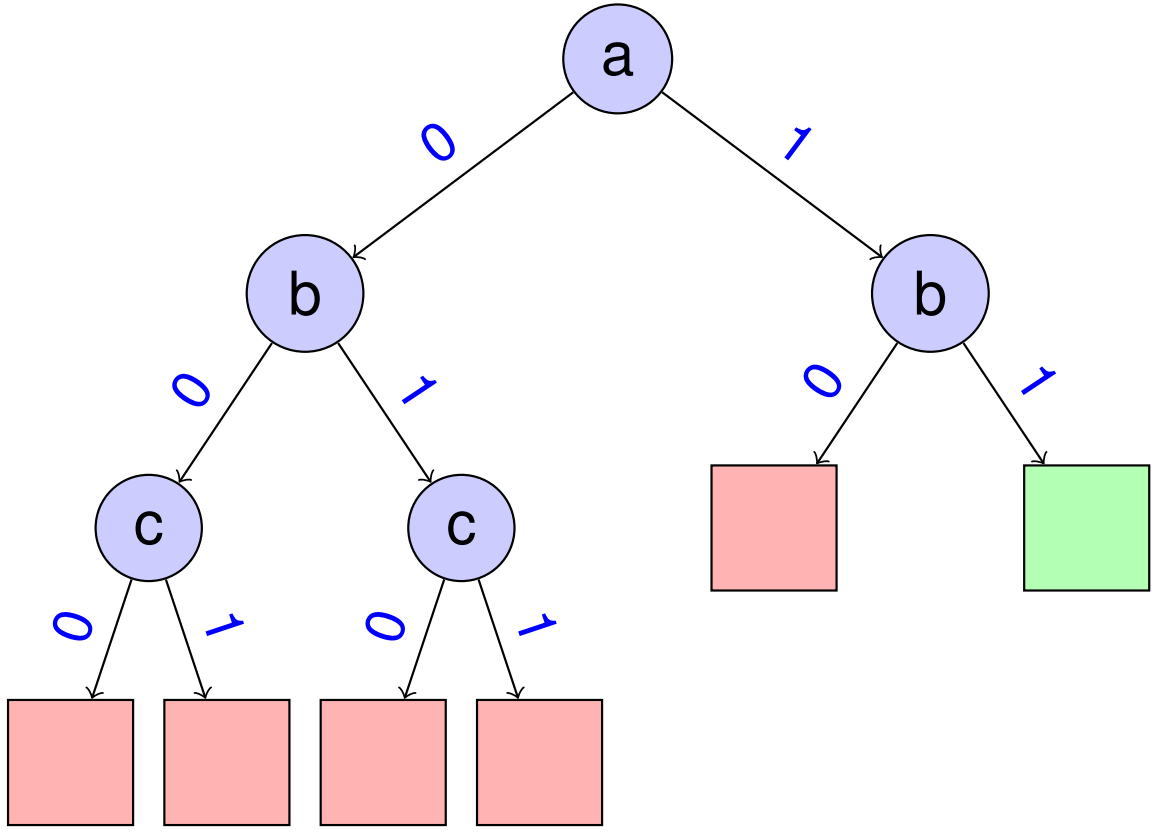
```
c  example circuit
c
p cnf 6 8
-4 1 2 0
-1 4 0
-2 4 0
-5 -3 0
5 3 0
-4 -3 6 0
4 -6 0
5 -6 0
```



$$(\neg s \vee p \vee q) \wedge$$
$$(\neg p \vee s) \wedge$$
$$(\neg q \vee s) \wedge$$
$$(\neg t \vee \neg r) \wedge \ldots$$

SAT Solving in a Nutshell

$(\neg a \vee b \vee c)$
$(a \vee c \vee d)$
$(a \vee c \vee \neg d)$
$(a \vee \neg c \vee d)$
$(a \vee \neg c \vee \neg d)$
$(\neg b \vee \neg c \vee d)$
$(\neg a \vee b \vee \neg c)$
$(\neg a \vee \neg b \vee c)$



Advancements in SAT Solving

- **1962:** DPLL backtracking algorithm [Davis et al., 1962]
- **1996:** Conflict learning [Silva and Sakallah, 1996]
- **2001:** Local search, decision heuristics, engineering ...

- Modern solvers handle millions of variables and clauses
- Popular solvers:
  - MiniSAT
  - Glucose
  - Lingeling
- Extensions of SAT:
  - Satisfiability Modulo Theories (SMT)
  - Quantified Boolean Formulae (QBF)
  - Max-SAT

Summary Decision Procedures

- Separation of verification problem and decision engine
- Many many applications in hardware & software verification
- Convenient and standardized APIs for ease of use
- Active field of research
- Popular SMT solvers:
  - MathSAT
  - Z3 (Microsoft Research)
  - STP (Stanford)
  - Yices

Outline

System Verilog Assertions (Reminder)

```
module monitor( foo.MONITOR I );

   property slave_data_notunknown_when_ready;
      @(posedge I.clk)
        I.ready | -> $isunknown(I.s) == 0;
   endproperty

   assert_slave_data_notunknown_when_ready: assert property (slave_data_notunknown_when_ready)
     else $error("%m: ready is asserted but data from slave is non valid");

   property slave_ready_until_valid;
      @(posedge I.clk)
        $rose(I.ready) | -> I.ready throughout I.valid [->1]; //ou I.ready [*0:$] ##1 I.valid;
   endproperty

   assert_slave_ready_until_valid: assert property(slave_ready_until_valid)
     else $error("%m:slave's ready must be held until valid is set");

   property slave_data_held_when_ready;
      bit [7:0] s;
      @(posedge I.clk) disable iff (I.nrst == 0)
        (I.ready && !I.valid , s = I.s) | => s == I.s; //ou $stable(I.s);
   endproperty

   assert_slave_data_held_when_ready: assert property(slave_data_held_when_ready)
     else $error("%m: data must be held stable when slave is ready");
endmodule
```

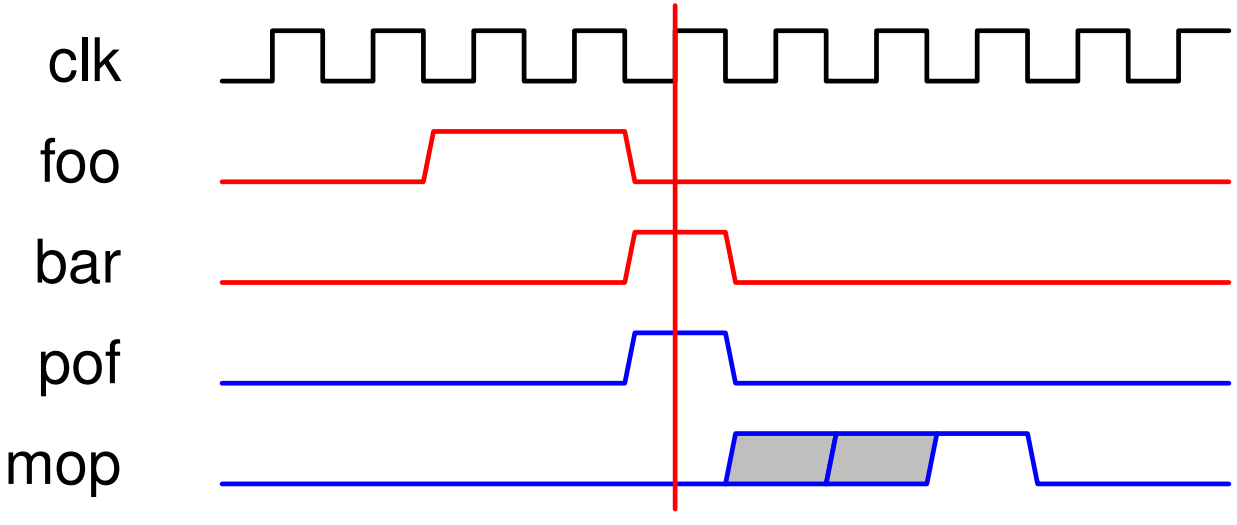
Basic Property Structure

```
// basic property structure
property foo;
   @(posedge clk) disable iff (rst)
     expr;
endproperty // foo

// verification directives
assert_foo: assert property(foo);
assume_foo: assume property(foo);
```

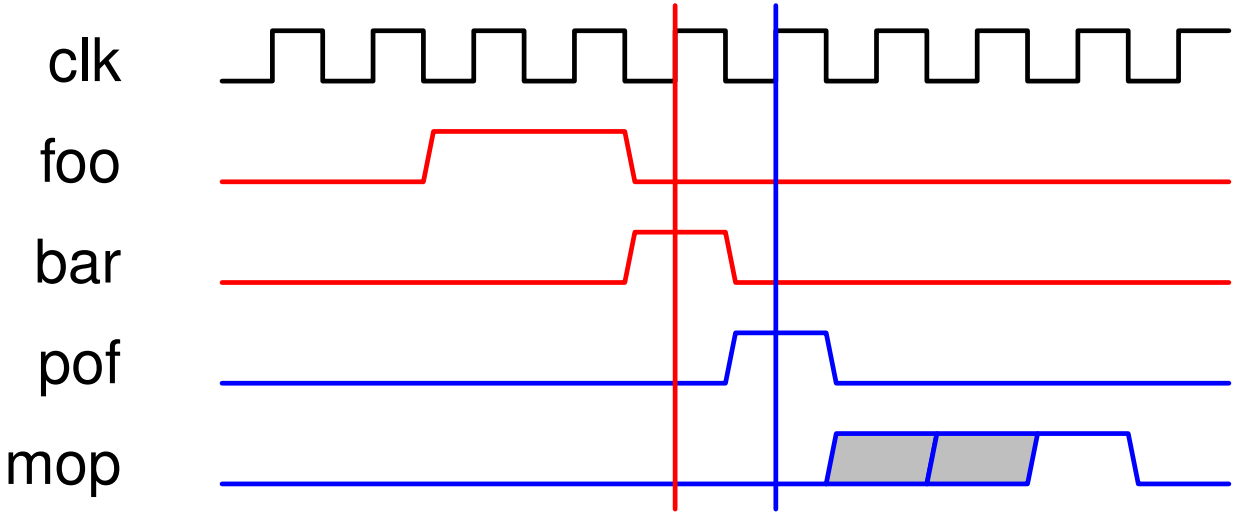
Sequences and Suffix Implication

```
// suffix implication
foo ## bar |-> pof ##[1:3] mop;
```



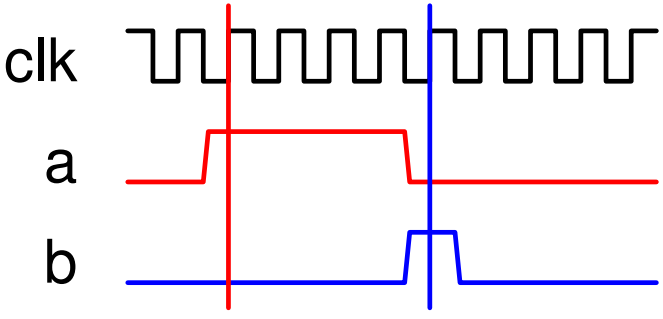
Non-Overlapping Suffix Implication

```
// non-overlapping suffix implication
foo ## bar |=> pof ##[1:3] mop;
```

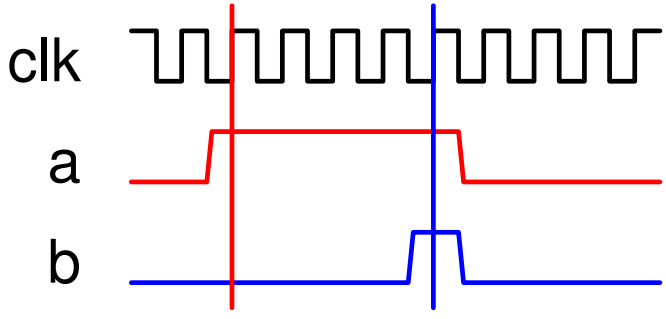


Until

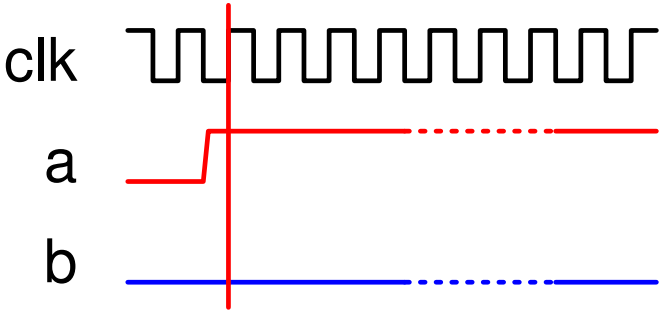
```
a until b
a s_until b
```




```
a until_with b
a s_until_with b
```

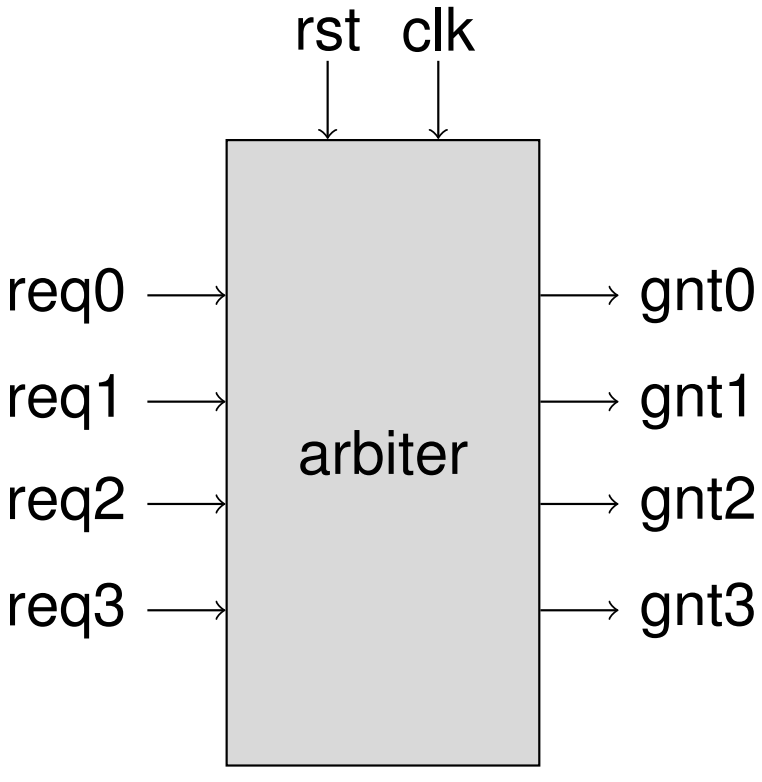



```
a until b
a until_with b
```



- Overlapping vs. non-overlapping
- Weak vs. strong

Round-Robin Arbiter



- Arbitration of four masters
- Single request always granted
- Fair arbitration of multiple (simultaneous) requests

Practical Exercise

Let's get to work. . .

References I


Bentley, B. (2005).
Validating a modern microprocessor.
In Etessami, K. and Rajamani, S., editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 2–4. Springer Berlin Heidelberg.


Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. (1999).
Symbolic model checking using SAT procedures instead of BDDs.
In *Design Automation Conference (DAC)*, pages 317–320.


Bradley, A. R. (2011).
SAT-based model checking without unrolling.
In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 70–87.


Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. (1992).
Symbolic model checking: 10^{20} States and beyond.
Information and Computation, 98(2):142–170.

References II

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002).
NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking.
In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark. Springer.

Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2003).
Counterexample-guided abstraction refinement for symbolic model checking.
Journal of the ACM, 50(5):752–794.

Cook, S. (1971).
The complexity of theorem proving procedures.
In *3. ACM Symposium on Theory of Computing*, pages 151–158.

Davis, M., Logemann, G., and Loveland, D. (1962).
A machine program for theorem-proving.
Commun. ACM, 5(7):394–397.

References III

Silva, J. a. P. M. and Sakallah, K. A. (1996).
Grasp – a new search algorithm for satisfiability.
In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '96, pages 220–227, Washington, DC, USA. IEEE Computer Society.