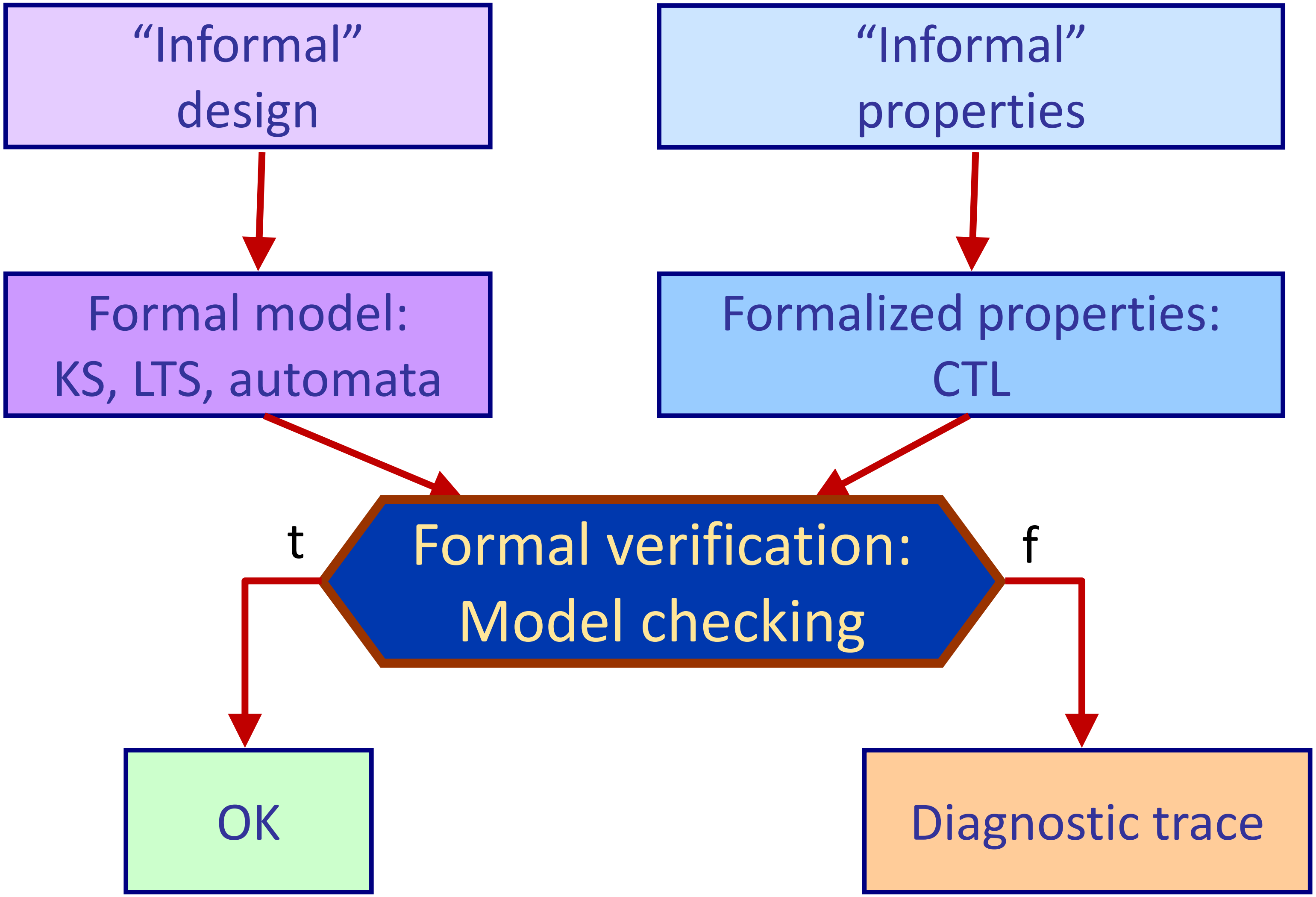
Bounded model checking

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Formal verification of CTL properties



Recap: Techniques for handling large state space

- CTL model checking: Symbolic technique

Set enumeration technique	Symbolic technique
Sets of labeled states	Characteristic functions (Boolean functions): ROBDD representation
Operations on sets of states	Efficient operations on ROBDDs

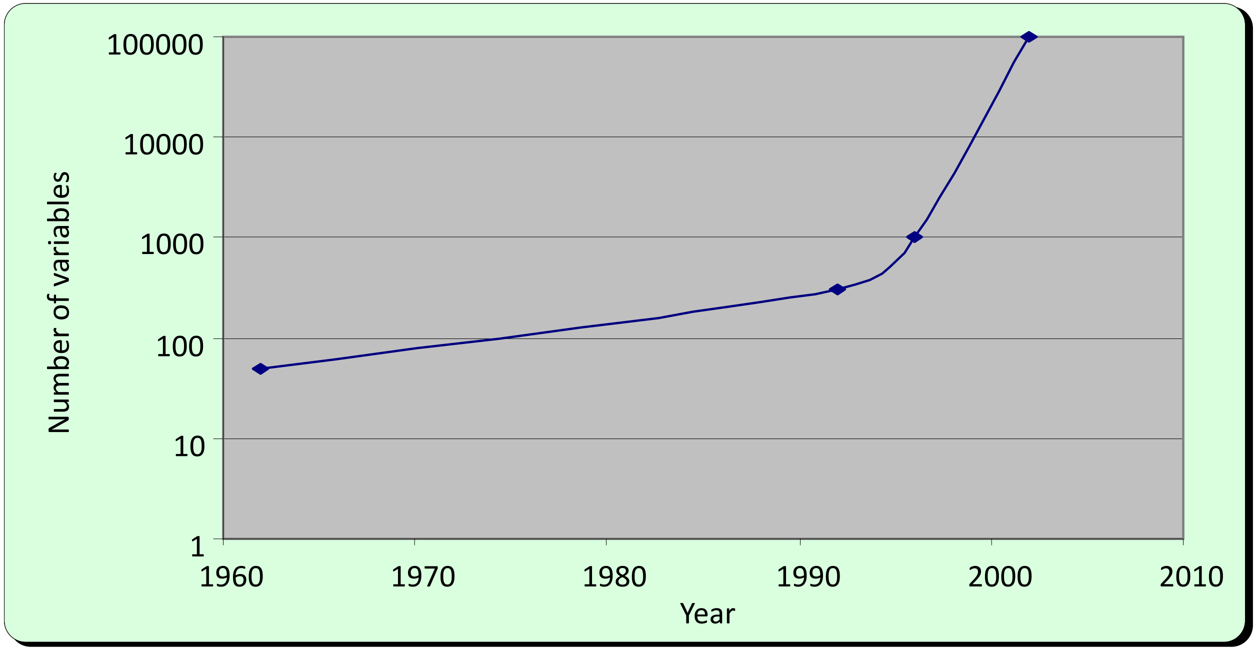
- Model checking of invariants: Bounded model checking
  - Model checking to a given depth:
    Searching for counterexamples with bounded length
    - A detected counterexample is always valid
    - Non-existing counterexample does not imply correctness
  - Background: Searching satisfying valuations for Boolean formulas with SAT/SMT techniques

The basic idea of bounded model checking

- We do not handle the state space “all in one”
- We perform checking by restricting the length of paths from the initial state
  - Partial verification: checking only up to a given bound in path length
  - The bound can be iteratively increased
  - In certain cases, the state space has a “diameter”: the length of the longest loop-free path
    - Increasing the bound to this length will result in complete checking

To be used: SAT solvers

- SAT solver:
  - Given a Boolean formula (Boolean function), it searches for a model, i.e., a variable assignment (substitution) that makes the formula true
  - Example: for formula $f(x_1,x_2,x_3)=x_1 \wedge x_2 \wedge \neg x_3$ substitution is $(x_1,x_2,x_3)=(1,1,0)$
- Hard problem, but efficient algorithms exist
  - zChaff, MiniSAT, …



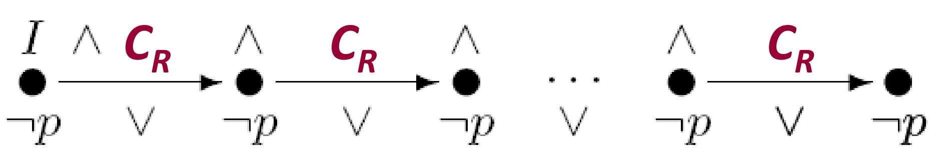
Approach and goals

- Mapping the bounded model checking problem (model + property) to a Boolean formula to be satisfied (by a SAT solver)
  - Model: Paths of bounded length are mapped to Boolean formula on the basis of the characteristic functions
    - Initial state
    - State transition relation to reach next states (along the path)
  - Property: Typically invariant properties mapped to Boolean formula as a characteristic function belonging to the property
    - Not limited to reachable states, but for all possible states
- The Boolean formula belonging to the model checking problem will be constructed in the following way:
  - If the SAT solver finds a substitution for the formula, then the substitution induces a counterexample for the property
  - If the SAT solver finds no substitution for the formula, then the property holds

Informal introduction

- How do we describe a path of bounded length?
  - Starting from the initial states: characteristic function $I(s)$
  - „Stepping forward” along potential transitions $s^0 \rightarrow s^1 \rightarrow s^2 \rightarrow s^3 \rightarrow \ldots$
    - Characteristic function of the transition relation: $C_R(s, s')$ with variables for s, s'
    - Step between s^0 and s^1: characteristic function $C_R(s^0, s^1)$ with separate variables for s^0, s^1
    - Second step: $C_R(s^1, s^2)$ with separate variables for s^1, s^2
    - i-th step: $C_R(s^i, s^{i+1})$ with separate variables for s^{i-1}, s^i
- How do we describe the property?
  - Invariant (states with a given local property): characteristic function $p(s)$
- The characterization of a counterexample (with conjunction):
  - Starting from the initial state: $I(s)$
  - „Stepping forward” n steps along the transition relation: $C_R(s^i, s^{i+1})$
  - To get a counterexample (somewhere $p(s^i)$ fails): $\neg p(s^i)$ disjunction on states of the path

A substitution for this formula corresponds to a counterexample:



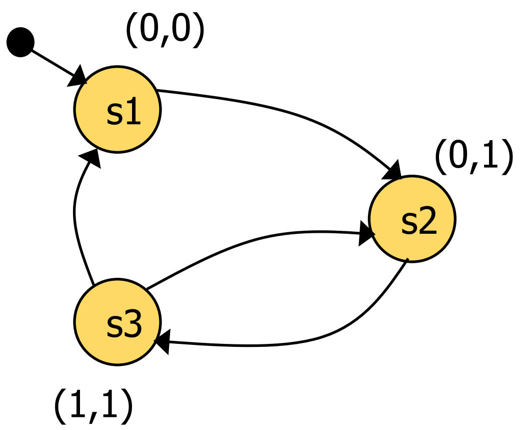
Notations

- Kripke structure M=(S,R,L)
- Logical formulas:
  - I(s): the characteristic formula of initial states with n variables
    - Background: Encoding states with a bit vector of length n
  - $C_R(s^i,s^{i+1})$: the characteristic formula of transitions in 2n variables
    - Disjunction of the characteristic function of individual transitions
  - path(): characteristic function of paths of length k with (k+1)*n variables

$$\mathrm{path}(s^0, s^1, \ldots, s^k) = \bigwedge_{0 \le i < k} C_R(s^i, s^{i+1})$$

  - p(s): characteristic function of the property
    - Based on the labeling of states with local property
    - In general: it can be constructed based on the state variables

Example: Encoding a model

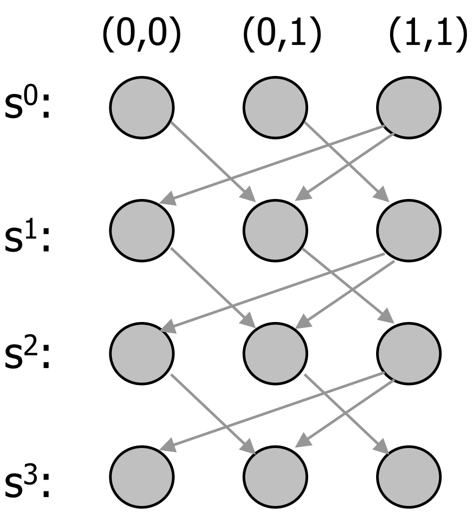


Initial state:

$I(x,y) = (\neg x \wedge \neg y)$

Transition relation:

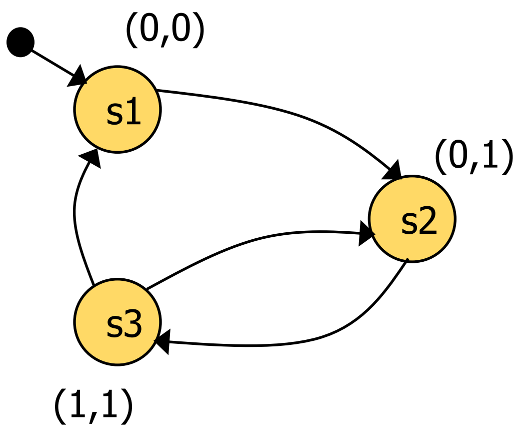
$$
\begin{aligned}
C_R(x,y,x',y') = {} & (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee \\
& \vee (\neg x \wedge y \wedge x' \wedge y') \vee \\
& \vee (x \wedge y \wedge \neg x' \wedge y') \vee \\
& \vee (x \wedge y \wedge \neg x' \wedge \neg y')
\end{aligned}
$$



Paths with 3 steps (from any state):

$$
\begin{aligned}
path(s^0,s^1,s^2,s^3) = {} & \\
& C_R(x^0,y^0,x^1,y^1) \wedge \\
& C_R(x^1,y^1,x^2,y^2) \wedge \\
& C_R(x^2,y^2,x^3,y^3)
\end{aligned}
$$

Example: Encoding a model

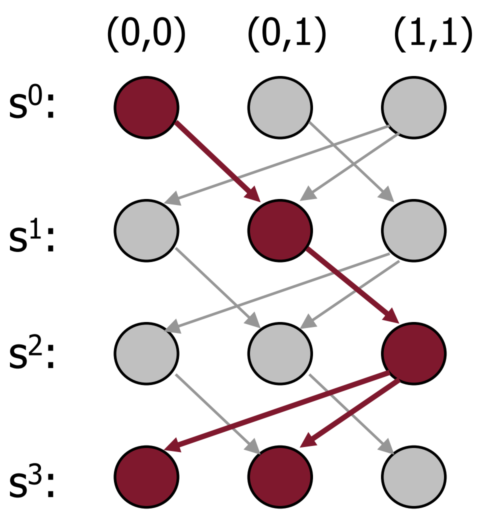


Initial state:

$I(x,y) = (\neg x \wedge \neg y)$

Transition relation:

$$
\begin{aligned}
C_R(x,y,x',y') = {} & (\neg x \wedge \neg y \wedge \neg x' \wedge y') \vee \\
& \vee (\neg x \wedge y \wedge x' \wedge y') \vee \\
& \vee (x \wedge y \wedge \neg x' \wedge y') \vee \\
& \vee (x \wedge y \wedge \neg x' \wedge \neg y')
\end{aligned}
$$



Paths with 3 steps from the initial state:

$$
\begin{aligned}
I(x^0,y^0) \wedge path(s^0,s^1,s^2,s^3) = {} & \\
= I(x^0,y^0) \wedge {} & \\
& C_R(x^0,y^0,x^1,y^1) \wedge \\
& C_R(x^1,y^1,x^2,y^2) \wedge \\
& C_R(x^2,y^2,x^3,y^3)
\end{aligned}
$$

Formalizing the problem

- Invariant to prove: Each path from the initial states ends in a state where p(s) holds

$$\forall i:\ \forall s^0, s^1, \ldots, s^i:\ (I(s^0) \wedge \mathrm{path}(s^0, s^1, \ldots, s^i) \Rightarrow\ p(s^i))$$

- Counterexample: If p(s) fails at some point then there exists an index i such that the following formula is satisfiable (a substitution exists):

$$I(s^0) \wedge \mathrm{path}(s^0, s^1, \ldots, s^i) \wedge \neg p(s^i)$$

  The substitution can be found by the SAT solver

  - That is, values for the (i+1)*n variables that define the path ($s^0, s^1, \ldots, s^i$)

- First idea: for i=0,1,2,..., check whether for paths of length i the following formula can hold:

$$I(s^0) \wedge \mathrm{path}(s^0, s^1, \ldots, s^i) \wedge \neg p(s^i)$$

Elements of the algorithm

- Iteration: i=0,1,2,... on the length of paths
- We are investigating loop-free paths: lfpath

Can be expressed in terms of the state variables

$$\mathrm{lfpath}(s^0, s^1, \ldots, s^k) = \mathrm{path}(s^0, s^1, \ldots, s^k) \wedge \bigwedge_{0 \le i < j \le k} s^i \ne s^j$$

- Termination condition during the iteration:
  - There is no loop-free path with length i from the initial state, that is, the following is not satisfiable:

$$I(s^0) \wedge \mathrm{lfpath}(s^0, s^1, \ldots, s^i)$$

  - There is no loop-free path with length i to a "bad state" (where p(s) does not hold), that is, the following is not satisfiable:

$$\mathrm{lfpath}(s^0, s^1, \ldots, s^i) \wedge \neg p(s^i)$$

- If the iteration stops, then p(s) holds invariably

The algorithm

$$i = 0$$

while True do

if $\text{not SAT}(I(s^0) \wedge \text{lfpath}(s^0, s^1, \ldots, s^i))$

or $\text{not SAT}((\text{lfpath}(s^0, s^1, \ldots, s^i) \wedge \neg p(s^i))$

then return True

if $\text{SAT}(I(s^0) \wedge \text{path}(s^0, s^1, \ldots, s^i) \wedge \neg p(s^i))$

then return $(s^0, s^1, \ldots, s^i)$

$i = i + 1$

end

No loop-free paths of length i from the initial states

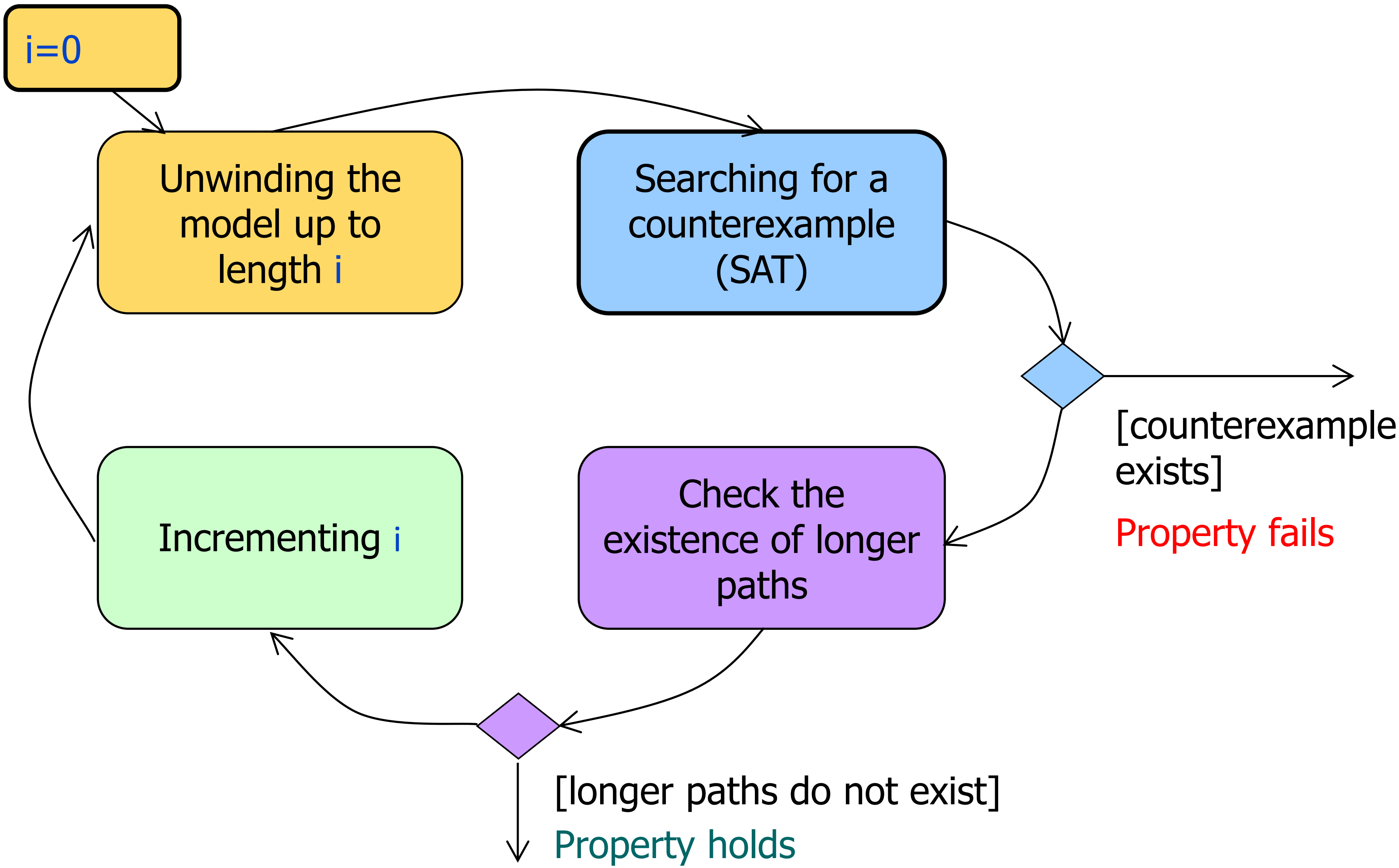
No loop-free paths of length i to a "bad state"

There is a path from an initial state to a "bad state"

Iteration

- If the result is True: the invariant holds
- If the result is a substitution of the $n*(i+1)$ variables inducing a path $(s^0,s^1,\ldots,s^i)$: it is a counterexample for the property $p(s)$

Bounded model checking with iteration



Refining the algorithm

- We do not start iterating from 0
  - Start with a given k, and try to generate the counterexample first
    - if such a counterexample exists, it is found quickly (without iterations)
    - If not: examine whether for k+1 the iteration terminates, and then increase k
  - It is not guaranteed that the length of the counterexample is minimal
  - Heuristics needed for estimating k if we aim to find a short counterexample
- Further restrictions on paths (encoded in the path formula):
  - On paths, no initial states are traversed after the first one
    - Not necessarily a loop – there might be many initial states
    - Similarly: No bad states are traversed before the last state of the path
  - Only the shortest path is considered between two states
    - Longer paths between the same pair of states are excluded
  - All initial states (if there are many) are considered “at once”
    - Those paths are avoided on which the end state can be reached by a shorter path from another initial state
    - Similarly for the bad states

Summary: BMC

- Efficient for checking invariant properties
- Sound method using loop-free paths
  - If there is a counterexample up to a certain bound, it will be found
  - A counterexample found is a valid counterexample
- Handling the state space
  - SAT solver: symbolic technique using Boolean formulas
  - For up to a given length of paths only a partial result is obtained
- Finding the shortest counterexample is possible
  - Useful for generating test sequences
- Automatic method
- Tool examples:
  - Symbolic Analysis Laboratory (SAL): sal-bmc
  - SAL sal-atg: used for automated test generation
  - CBMC: bounded model checker for C source code

Outlook: The results of Intel (hardware models)

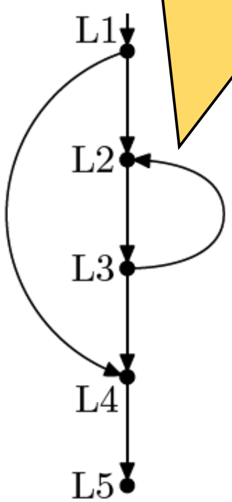
Model	k	Forecast (BDD)	Thunder (SAT)
Circuit 1	5	114	2.4
Circuit 2	7	2	0.8
Circuit 3	7	106	2
Circuit 4	11	6189	1.9
Circuit 5	11	4196	10
Circuit 6	10	2354	5.5
Circuit 7	20	2795	236
Circuit 8	28	—	45.6
Circuit 9	28	—	39.9
Circuit 10	8	2487	5
Circuit 11	8	2940	5
Circuit 12	10	5524	378
Circuit 13	37	—	195.1
Circuit 14	41	—	—
Circuit 15	12	—	1070

Bounded model checking based on software source code

Use for software: the problem of loops

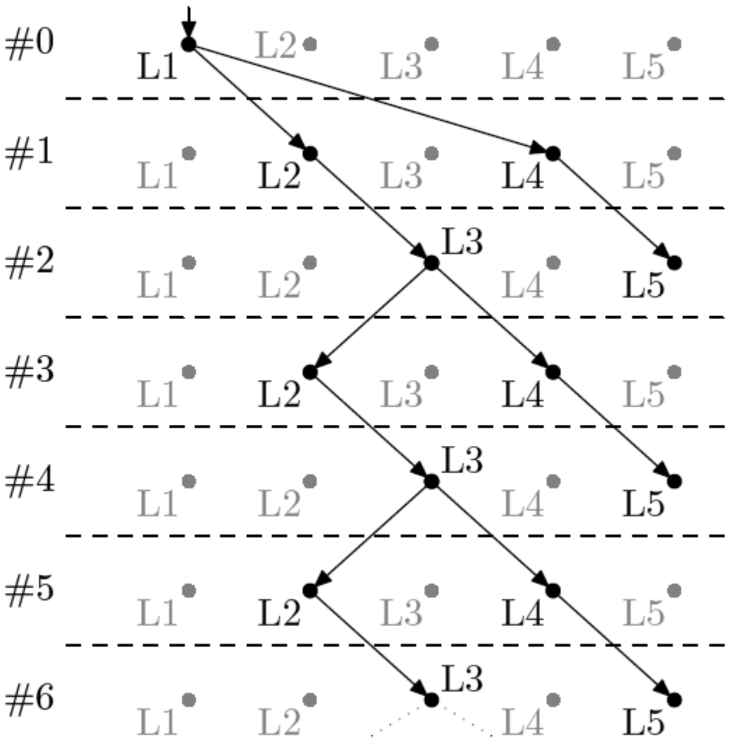
Control flow graph (CFG):

Loop in the program: variables are modified, resulting in new states!



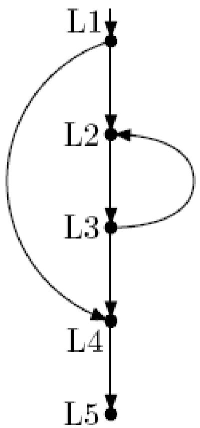
Traversing cycles might lead to new states

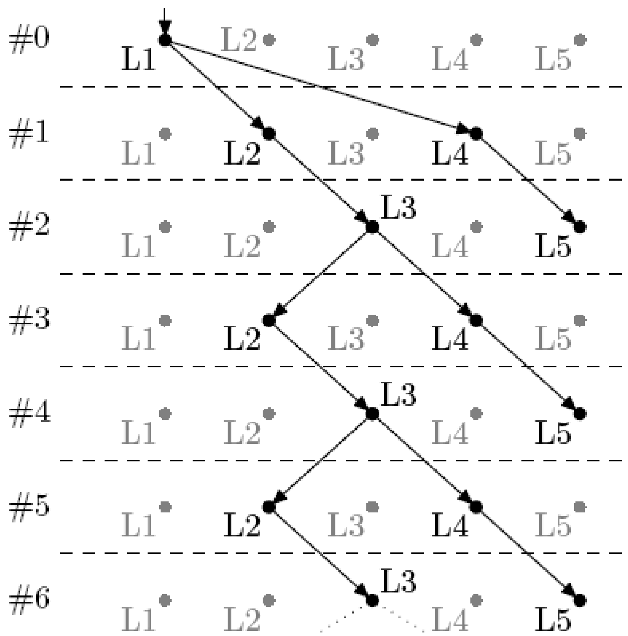
Path enumeration:

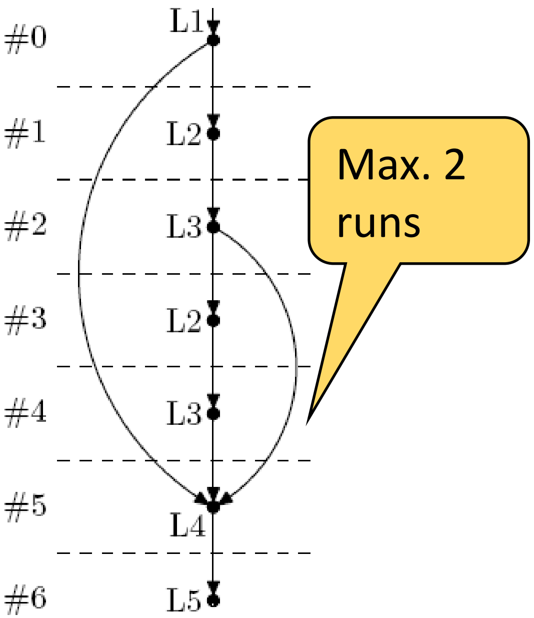


Handling the loops

- Possibilities for handling the loops:
  - Path enumeration:
    - Systematically along all possible paths in execution cycles
  - Loop unrolling:
    - Unrolling loops in a limited number of runs







Software model checking tools

- F-SOFT (NEC):
  - Path enumeration
  - Used for checking Unix system utilities (e.g. pppd)
- CBMC (CMU, Oxford University):
  - Supports C, SystemC
  - Loop unrolling
  - Support for certain system libraries in Linux, Windows, MacOS
  - Handling integer arithmetic:
    - Bit level („bit-flattening”, „bit-blasting”)
  - CBMC with SMT solving
    - Satisfiability Modulo Theories (SMT):
      SAT solving extended with first order theories, e.g. integer arithmetic
- SATURN:
  - Loop unrolling: at most 2 runs
  - Full Linux kernel was verifiable for Null pointer dereferences

Supplementary material: k-induction

The basic idea of k-induction

- Introduction: Let P_i be a series of properties
  - Traditional mathematical induction:

$$P_0 \wedge \forall i : \left(P_i \Rightarrow P_{i+1}\right) \;\Rightarrow\; \forall n : P_n$$

  - k-induction:

$$\bigwedge_{j=0}^{k-1} P_j \wedge \forall i : \left(\left(\bigwedge_{j=0}^{k-1} P_{i+j} \right) \Rightarrow P_{i+k} \right) \;\Rightarrow\; \forall n : P_n$$

- Idea: Application on state space to check invariants
  - Base case: The invariant holds on paths of length k from the initial state (this can be checked by bounded model checking)
  - Inductive step: If the invariant holds on paths of length k from any state, then it holds for the next states that follow the end states of each path (i.e., on paths of length k+1)
    - Single state transition from any state may not keep the property
    - But k successive transitions may keep the property to k+1

k-induction on the state space

- Formula: $\bigwedge_{j=0}^{k-1} P_j \wedge \forall i:\left(\left(\bigwedge_{j=0}^{k-1} P_{i+j}\right) \Rightarrow P_{i+k}\right) \Rightarrow \forall n: P_n$

- Its base case: $\bigwedge_{j=0}^{k-1} P_j$

  Corresponds to:

$$\forall s^0, s^1, \ldots, s^{k-1}: \left(I(s^0) \wedge path(s^0, s^1, \ldots, s^{k-1})\right) \Rightarrow \left(\forall 0 \leq j < k-1:\ P(s^j)\right)$$

- Inductive step: $\forall i:\left(\left(\bigwedge_{j=0}^{k-1} P_{i+j}\right) \Rightarrow P_{i+k}\right)$

  Corresponds to:

$$\forall i: \forall s^i, s^{i+1}, \ldots, s^{i+k}: \left(path(s^i, s^{i+1}, \ldots, s^{i+k}) \wedge \bigwedge_{j=i}^{i+k-1} P(s^i) \right) \Rightarrow P(s^{i+k})$$

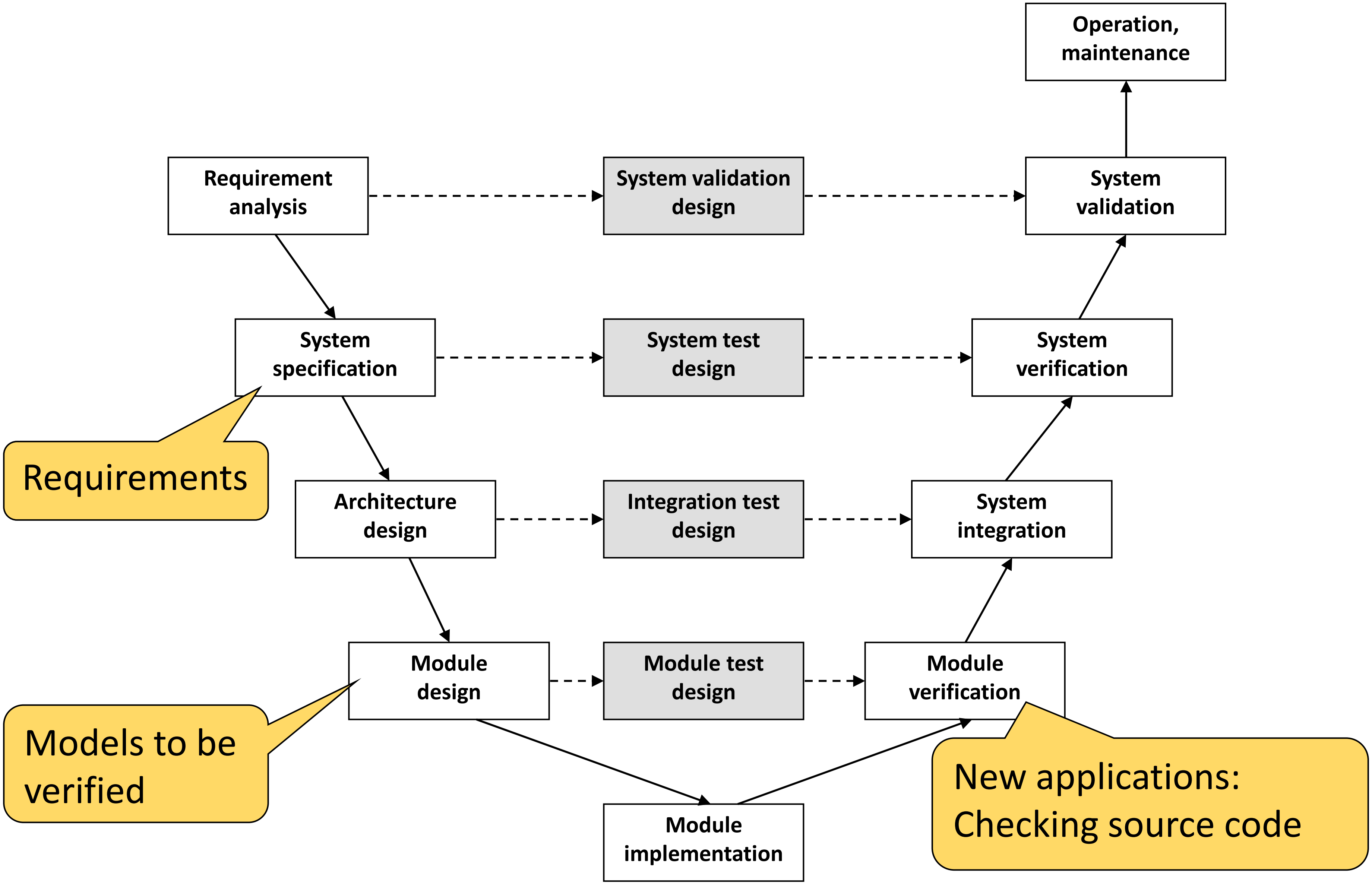
Using k-induction

- Cases for evaluating the invariant property:
  - If the base case (bounded model checking) provides a counter-example: The invariant does not hold
  - If there is no counterexample in the base case and no counterexample in the inductive step: The invariant holds
  - Otherwise: It is not known whether the invariant holds
    - A counterexample resulting from the inductive step may not hold (considering the given initial state of the model)
- Further steps if there is no decisive result:
  - Increasing the length of the induction
    - In case of longer paths decisive result may arise
  - Strengthening the invariant: P' is checked instead of P, where P' => P
  - Adding and extra invariant (additional knowledge)
    - If there is another invariant L then it restricts the paths considered:

$$\bigwedge_{j=0}^{k-1} P_j \wedge \forall i : \left(\left(\bigwedge_{j=0}^{k-1} (P_{i+j} \wedge L_{i+j}) \right) \Rightarrow P_{i+k} \right) \Rightarrow \forall n : P_n$$

Summary:
Properties of model checking

Model checking during the design



Efficient techniques for model checking

- Symbolic model checking
  - Characteristic formulas represented as ROBDD
  - Efficient for „well structured” problems
    - E.g. identical processes in a protocol
  - Size depends on variable ordering
- Bounded model checking for invariant properties
  - Based on satisfiability solving (SAT solver)
  - Searching for counterexamples of bounded length
    - A counterexample found is a valid counterexample
    - If no counterexample found, it is only a partial result (longer counterexamples might exist)
  - Good for test generation

Strengths of model checking

- It is possible to handle large state spaces
  - State spaces of size 10^{20}, but examples even for size 10^{100}
  - This is the state space of the system (e.g. network of automata)
  - Efficient techniques: symbolic, SAT based (bounded)
- General method
  - Software, hardware, protocols, …
- Fully automatic tool, no intuition or strong mathematical background is needed
  - Theorem proving is much difficult to apply
- Generates a counterexample that can be used for debugging

Turing Award in 2007 for establishing model checking:
E. M. Clarke, E. A. Emerson, J. Sifakis (1981)

Weaknesses of model checking

- Scalability
  - Uses explicit state space traversal
  - Efficient techniques exist, but good scalability can not be guaranteed
- Mainly for control driven applications
  - Complex data structures induce a large state space
- Hard to generalize the results
  - If the protocol is correct for 2 processes,
    is it correct for N processes?
- Formalizing requirements is hard
  - „Dialects" in temporal logic for different domains
  - IEEE standard: PSL (Property Specification Language)