AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

COMBINATORIAL TESTING

PETER GORM LARSEN
PROFESSOR
(PGL@ENG.AU.DK)

PLAN

- Need for testing and verification
- Combinatorial testing
- Test example
- Model checking
- State space explosion

NEED FOR TESTING &VERIFICATION

- The Ariane 5 rocket exploded on June 4, 1996, less than forty seconds after it was launched.
- The investigation committee found a software error in the computer responsible to calculate movement.
- An exception occurred when a large 64-bit floating point number was converted to a 16-bit integer.
- This conversion was not protected by code for handling exceptions and caused the computer to fail, incorrect attitude data was transmitted.

NEED FOR TESTING &VERIFICATION

- No longer feasible to shut down a malfunctioning system in order to restore it.
- Systems need to be analyzed not only for compiling and static errors.
- The principal validation methods for complex systems are simulation, testing, and verification (model checking, deduction).

NEED FOR TESTING &VERIFICATION

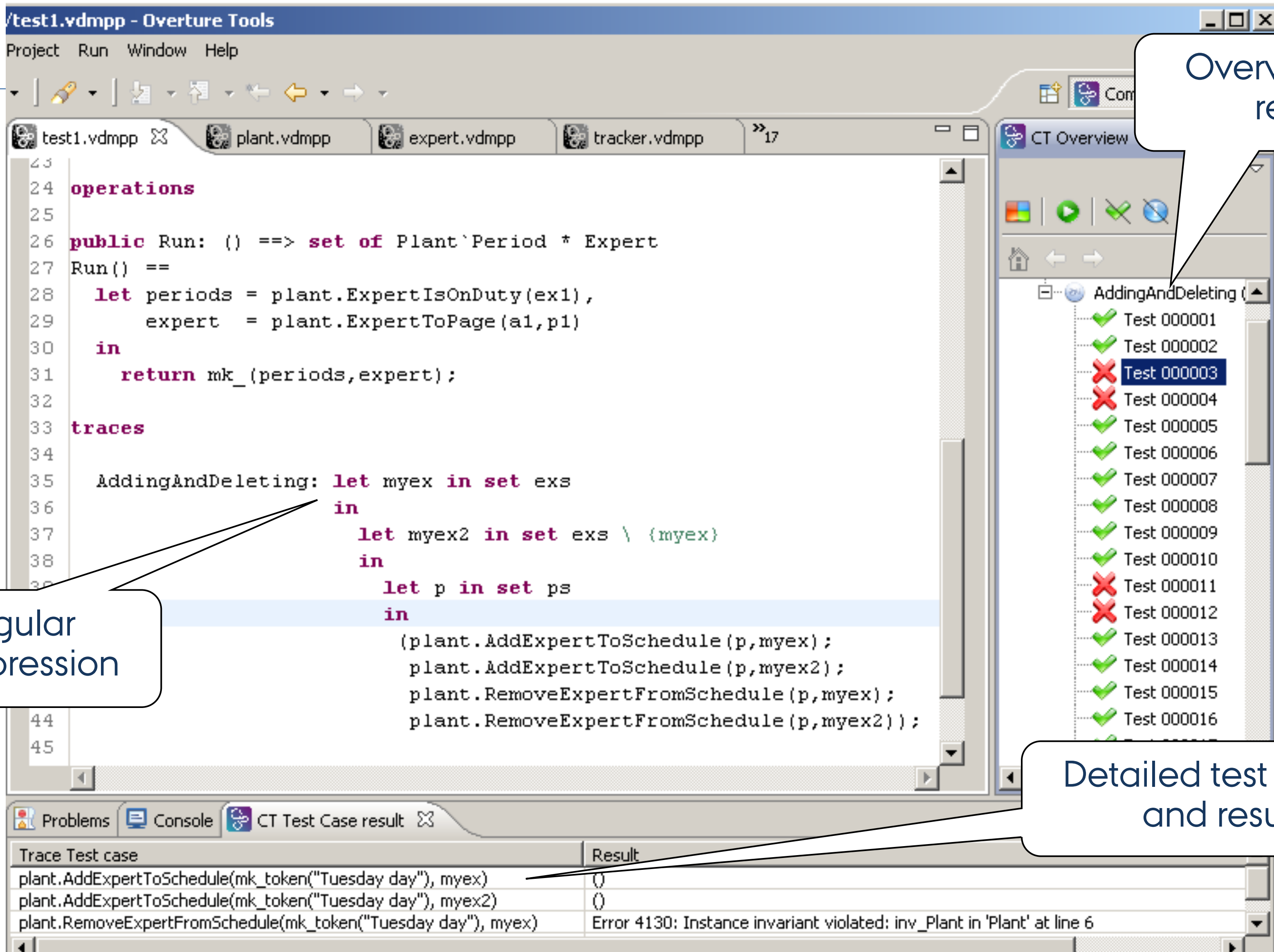
- The analysis is expensive to be performed on the actual software.
- Huge state space, combinatorial explosion
- Combining different techniques to achieve V&V: Cheap to expensive (small hummer, big hummer)

TESTING

- Testing involves making experiments before deploying the system in the field.
- Can be performed on a model or the actual system
- Typically inject signals/inputs at certain points in the system and observe the resulting signals/outputs at other points.
- Testing is a cost-efficient way to find many errors
- However, checking all potential executions and interleaving using testing is rarely possible.

AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

COMBINATORIAL TESTING PERSPECTIVE



Overview of results

Regular expression

Detailed test case and results

EXAMPLE

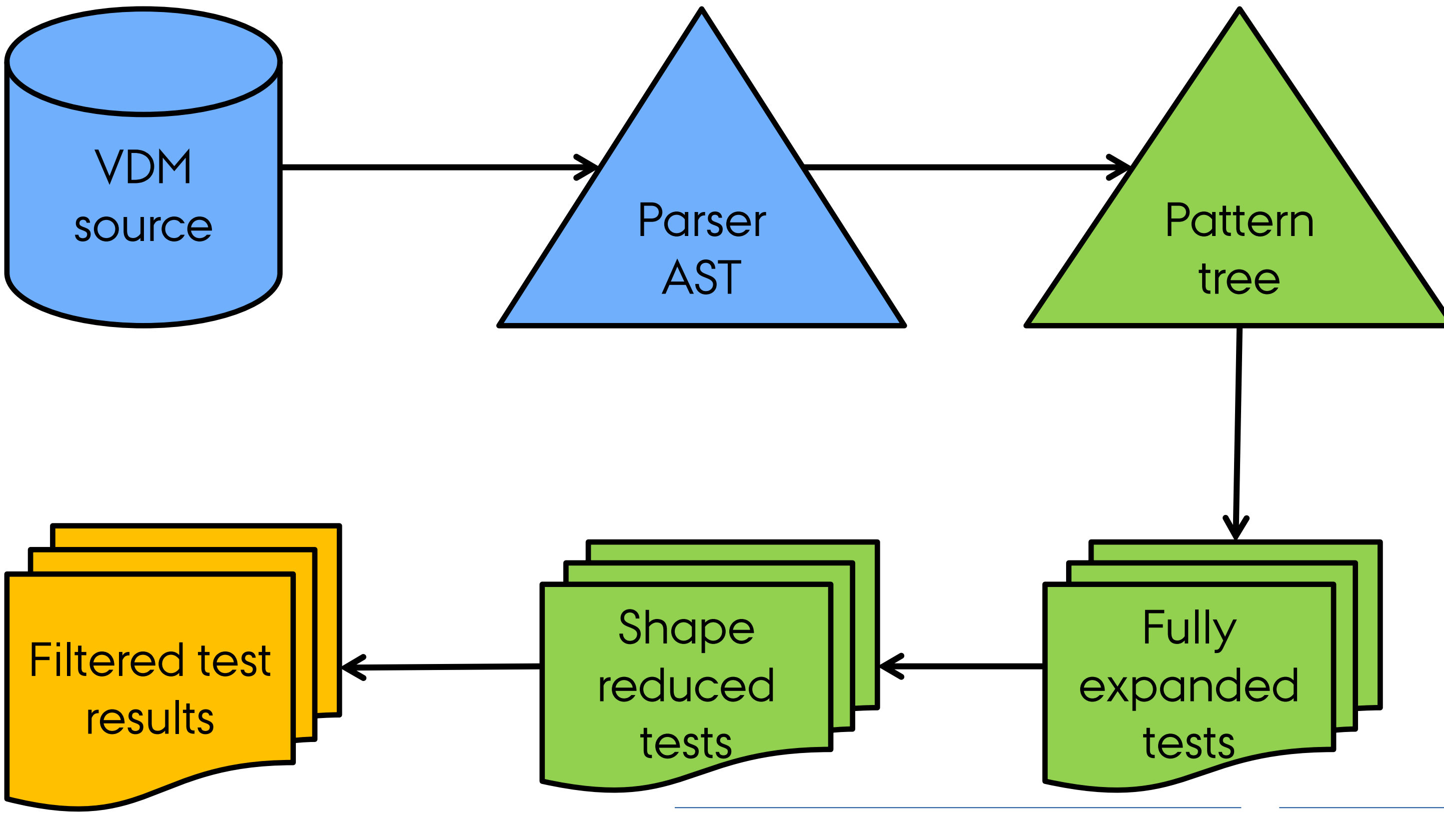
Trace:

```
TTest: (let x in set {1, 2}
         in t.Insert(x)){3}
```

Generated Testcases:

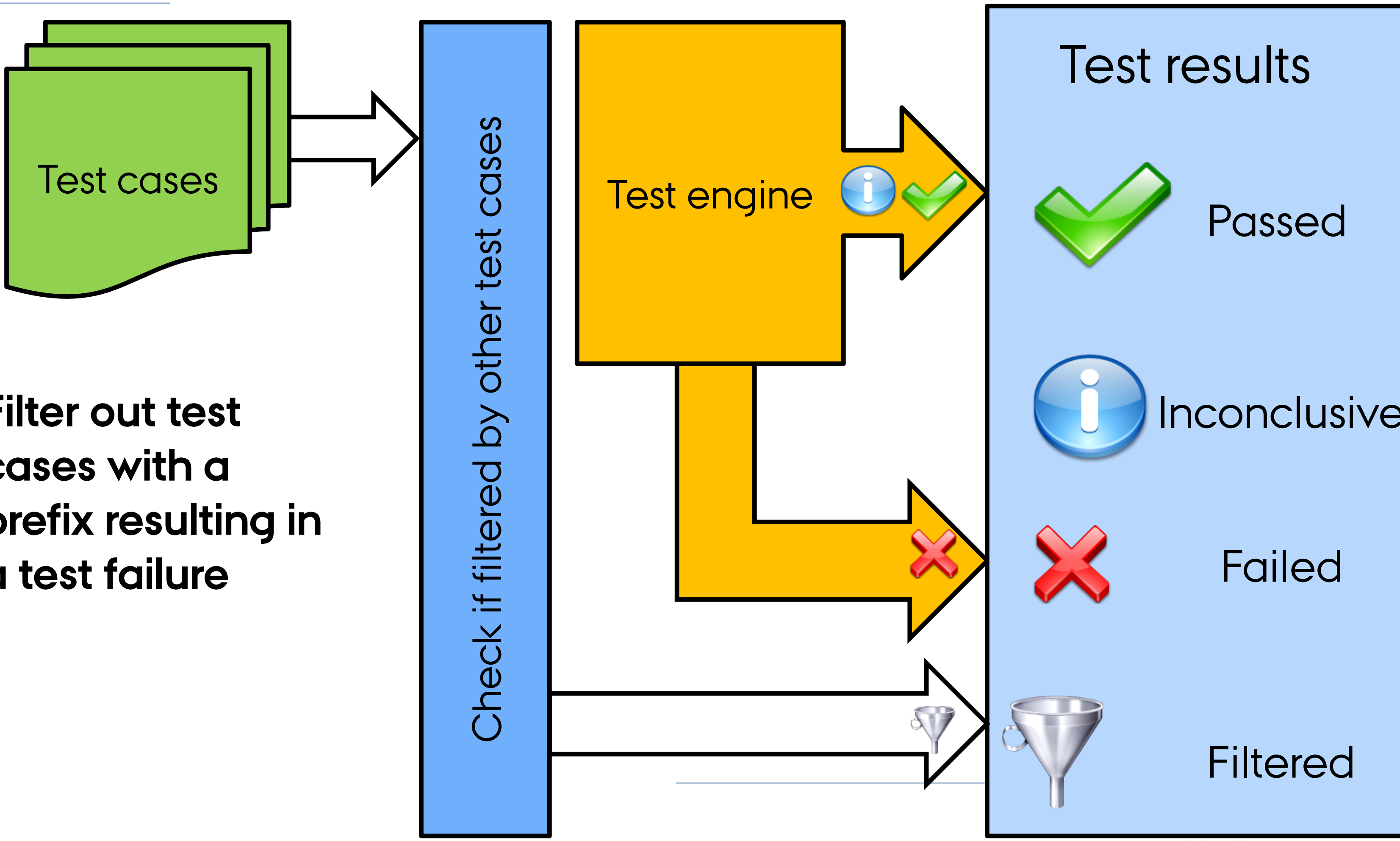
```
TC1: t.Insert(1);t.Insert(1);t.Insert(1)
TC2: t.Insert(1);t.Insert(1);t.Insert(2)
TC3: t.Insert(1);t.Insert(2);t.Insert(1)
TC4: t.Insert(1);t.Insert(2);t.Insert(2)
TC5: t.Insert(2);t.Insert(1);t.Insert(1)
TC6: t.Insert(2);t.Insert(1);t.Insert(2)
TC7: t.Insert(2);t.Insert(2);t.Insert(1)
TC8: t.Insert(2);t.Insert(2);t.Insert(2)
```

COMBINATORIAL TESTING OVERVIEW



AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

TEST CASE EXECUTION



Filter out test cases with a prefix resulting in a test failure

AN EXAMPLE

```
operations
public insert : int ==> ()
insert(val)== skip
pre val > 1;

traces
T1 : let x in set {1,2} in insert(x);
```

ANOTHER EXAMPLE

```
class A
values
obj : A = new A();

operations
public op : nat ==> nat
op (x) == return x;

traces
T2:
let x,y in set {1, ..., 10}
in
(obj.op(x);obj.op(y));

end A
```

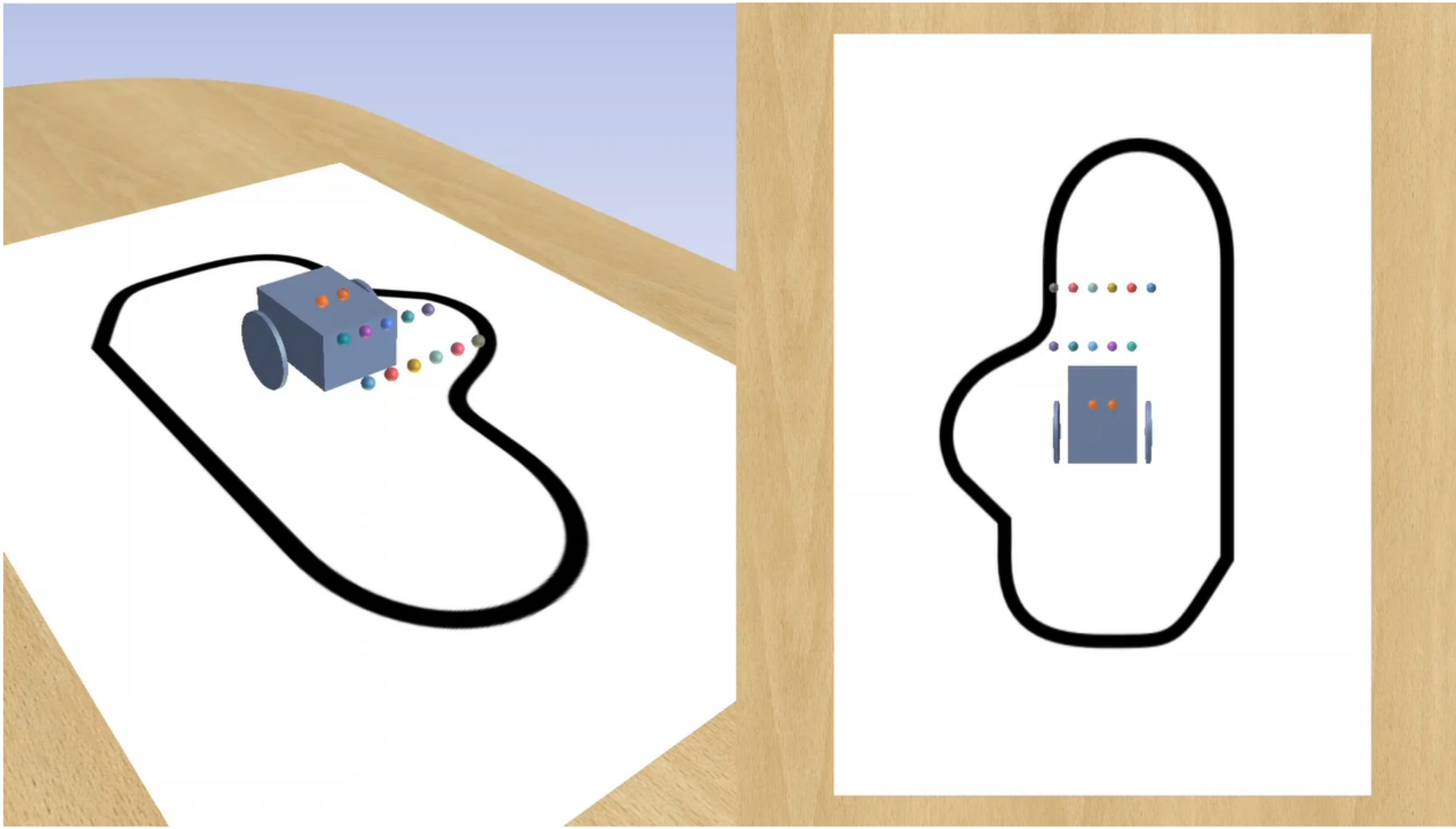
How many test cases will this generate?

REGULAR EXPRESSIONS IN TRACES

- Let definitions (local naming)
- Let be such that definitions (all selection)
- Repeat traces (fixed or variable number of times)
- Concurrency traces (all interleavings)
- Application expression (calling operations)
- Trace alternatives

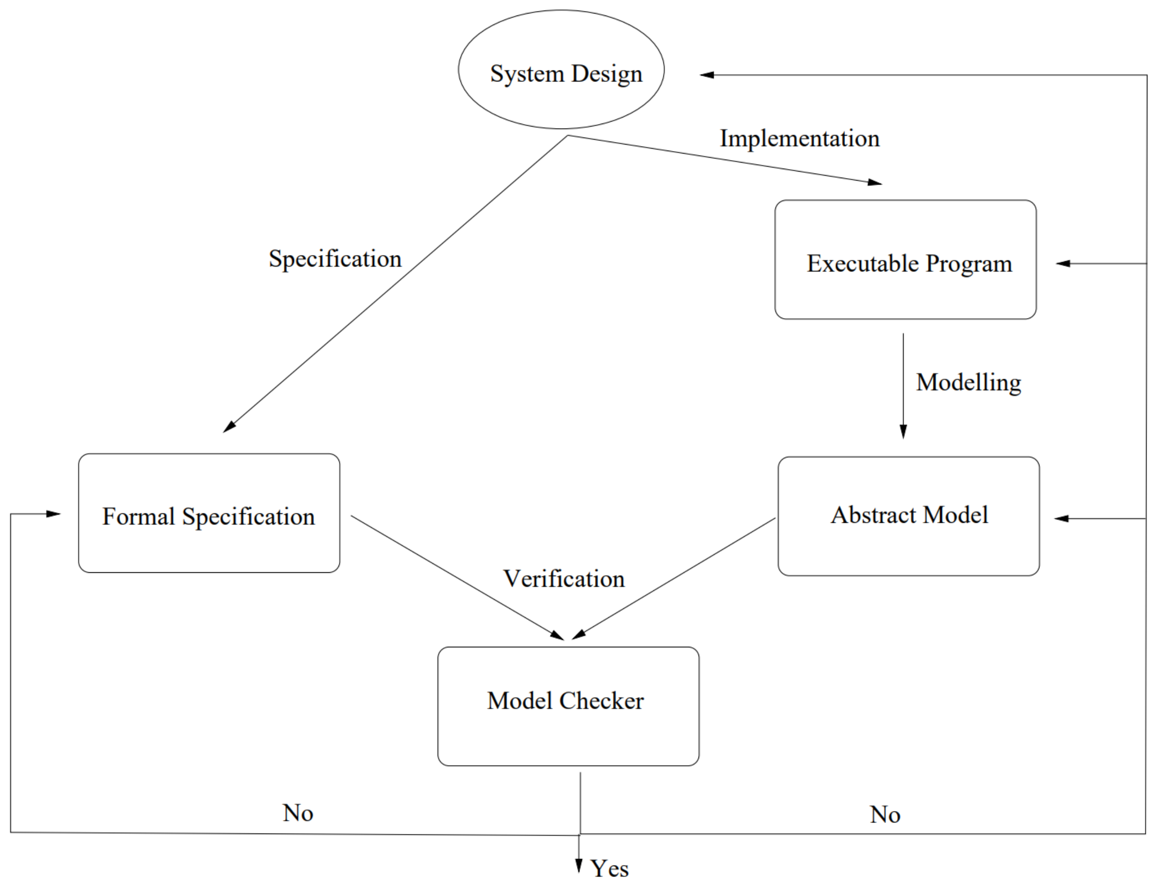
AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

DESIGN SPACE EXPLORATION



AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

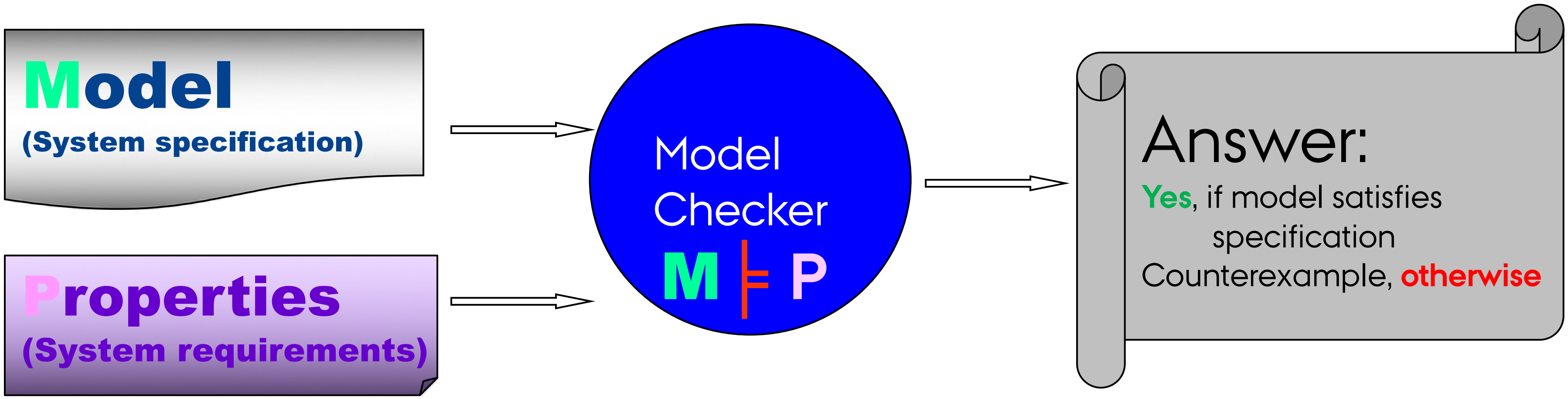
MODEL CHECKING (1)



MODEL CHECKING (2)

- Model checking is applicable for large systems, but with finite state space.
- Constant and static state space.
- Normally it uses an exhaustive search of the finite state space of the system.
- Automatic exploration of the state space.

MODEL CHECKING (3)



- Model specifies the system behavior in a formal way.
- Properties:
  - Safety: Invariants, deadlocks, reachability, etc
  - Liveness: faireness, response, infinite traces, etc

SPECIFICATION/MODELING

- Description of the behavior of individual processes and potential synchronizations.

Example: 2 concurrent processes M1 and M2 competing for a semaphore S, each process has three states: Non-critical (N), Trying (T), Critical (C). Semaphore available (S_0) or taken (S_1).

$$
\begin{array}{lcl}
N_1 & \rightarrow & T_1 \\
T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 \\
C_1 & \rightarrow & N_1 \wedge S_0
\end{array}
\quad || \quad
\begin{array}{lcl}
N_2 & \rightarrow & T_2 \\
T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\
C_2 & \rightarrow & N_2 \wedge S_0
\end{array}
$$

PROPERTIES DESCRIPTION

› A property describes a requirement that must always be satisfied by the system execution, no matter how the system execution evolves.

› Properties are usually described using a logic, or a syhntax that is compatible with the specification language.

Example: process1 and process2 must not be in a critical state at the same time.

$$\boxed{M1 || M2 \models A[]\ \mathrm{not}(C_1 \wedge C_2)}$$

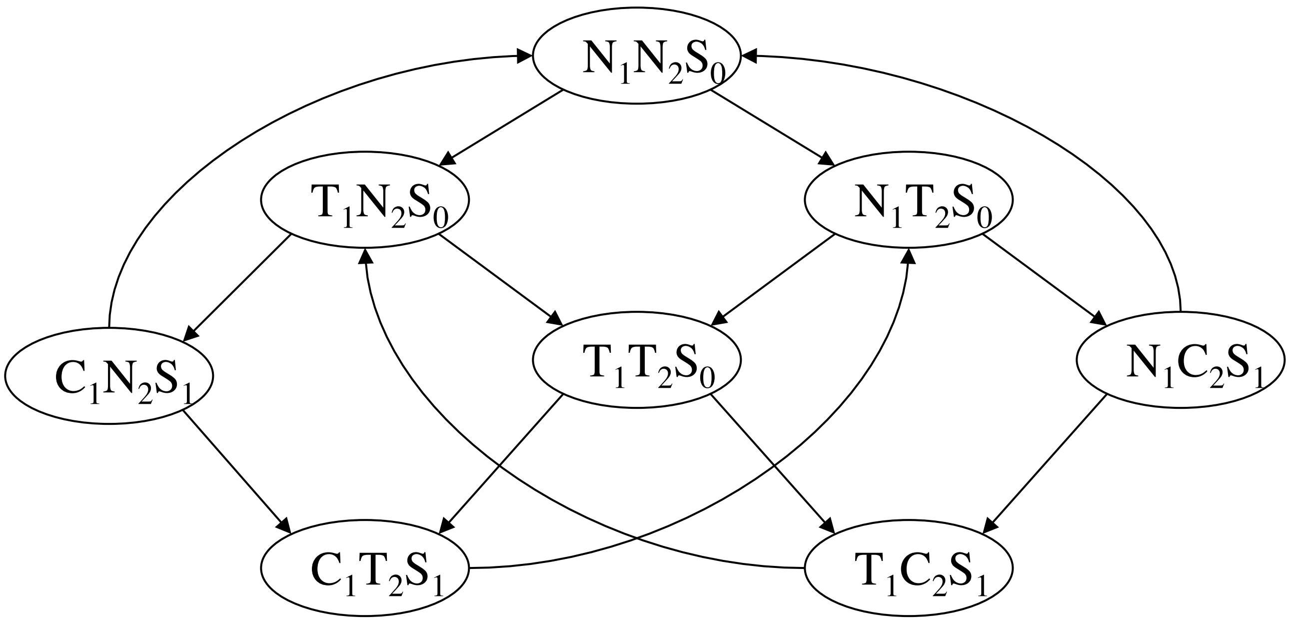
› Related to VDM syntax, a property can be: all invariants, pre-conditions and post-conditions must be satisfied.

VERIFICATION (1)

- To check whether a property is satisfied or not, the model checker calculates first the state space.
- State space is formed by all potential executions.
- The verification visits each state in the statespace and checks whether the property is satisfied or not.
- It terminates with yes/no (counterexample).

VERIFICATION (2)

Example: state space for M1||M2.



Property $\boxed{M1 || M2 \models A[]\ not(C_1 \wedge C_2)}$ is satisfied.

STATE SPACE EXPLOSION

- The composition of two concurrent processes is modeled by taking the Cartesian product of the corresponding state spaces.
- In general, state space = $[M_1]*[M_2]*..*[M_n]$.
- Even a small system (<10 proc) can end up in millions of states due to data/time variables.

Challenging drawback: state space explosion.

SYMBOLIC STATE SPACE

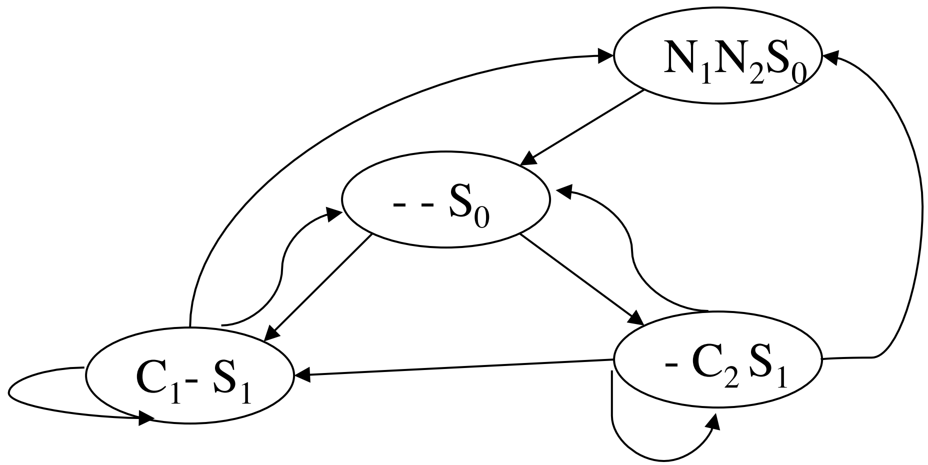
Two main approaches have been proposed to cope with the state explosion problem:

- Symbolic algorithms.
- Partial order reduction.

In symbolic model checking, logical formulas may be associated with the set of states that validate the formula. Thus compacting the state space.

SYMBOLIC ALGORITHMS

- Example: formula: inUse(S)=y/n by one of the processes.



- State space can be reduced to n/[p], n: original state space, p:number pf processes.

SUMMARY

- What have I presented today?
  - Combinatorial testing possibilities
  - Model checking and symbolic algorithms.
- What do you need to do now?
  - Read chapter 13 for next week
  - Continue working on your mini-projects