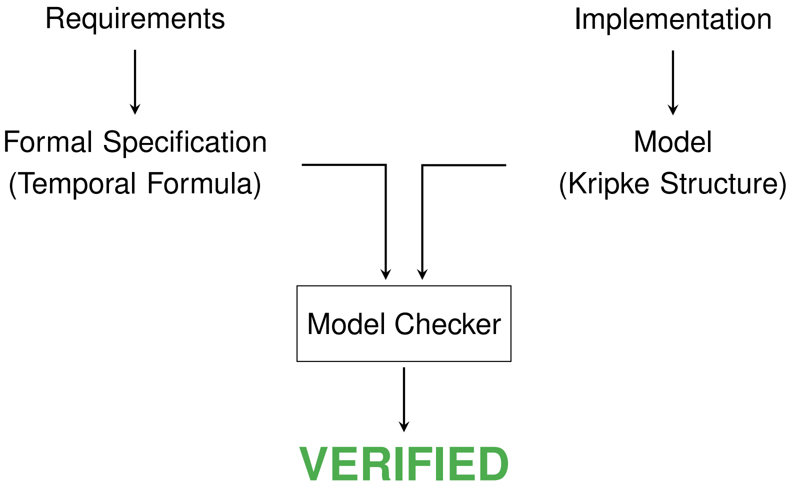
DEBUGGING: SOFTWARE BMC

WS 2017/2018

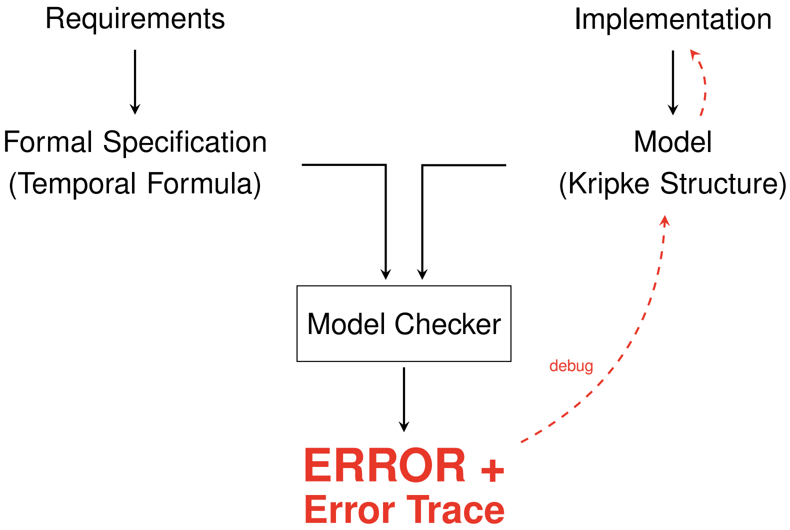
Martina Seidl
Institute for Formal Models and Verification

JOHANNES KEPLER
UNIVERSITY LINZ

Model Checking



Model Checking



Types of Model Checking

General question: Given a system K and a property p, does p hold for K (i.e., for all initial states of K) ?

- Explicit state model checking
  - enumeration of the state space
  - state explosion problem
- Symbolic model checking
  - representation of model checking problem as logical formula (e.g., in propositional logic (SAT) or QBF)

Bounded Model Checking

basic idea: search for a counter-example of bounded length k

- encoding in propositional logic (or extensions)
- use SAT solvers to find such a counter-example:
  formula is satisfiable iff a bug is found, i.e., an execution of program that violates the claim.
- benefits:
  - bit-precise encoding of the real semantics
  - powerful SAT solvers
  - difficulty of the problem is controllable (by selection of k)
- drawback: incomplete for k that is too small

$\Rightarrow$ can be used for debugging

Bounded Model Checking of ANSI-C Programs

- idea:
  - unwind program into equation
  - check equation using SAT
- benefits:
  - completely automated
  - treatment of pointers and dynamic memory is possible
- properties:
  - simple assertions
  - run time errors (pointers/arrays)
  - run time guarantees (WCET)

for example implemented in tool CBMC

From C to SAT

- removal of side effects
  example: `j=i++` is rewritten to `j=i; i=i+1`
- control flow is made explicit
  example: `continue, break` are replaced by `goto`
- transformation of loops to `while (...) ...`
- `while (...) ...` loops are unwound
  - all loops must be bounded
    $\rightarrow$ analysis may become incomplete
  - constant loop bounds are found automatically, others must be specified by user
  - to ensure sufficient unwinding, “unwinding assertions” are added

From C to SAT: Loop Unwinding

original function:

```
void f (...) {
  ...
  while (cond) {
    body;
  }
  rest;
}
```

with unwounded loop:

```
void f (...) {
  ...
  if (cond) {
    body;
    if (cond) {
      body;
      if (cond) {
        body;
        assert(!cond);
      }
    }
  }
  rest;
}
```

after last iteration an assertion is added:

violated if program runs longer than bound permits

From C to SAT: SSA

single static assignment (SSA) form: fresh variable for LHS of each assignment

example:

```
x = x + y;
x = x * 2;
a[i] = 100;
```

is translated to

```
x1 = x0 + y0;
x2 = x1 * 2;
a1[i0] = 100;
```

from which the following SMT formula can be derived

$$(x_1 = x_0 + y_0) \wedge (x_2 = x_1 * 2) \wedge (a_1[i_0] = 100)$$

From C to SAT: Conditionals

- for each join point, new variables with selectors are added
- example:

original program:

```
if (v)
  x = y;
else
  x = z;

w = x;
```

$\Rightarrow$

rewritten program:

```
if (v0)
  x0 = y0;
else
  x1 = z0;

x2 = v0 ? x0 : x1;

w1 = x2;
```

From C to SAT: Example

```
int main () {
  int x, y;
  y = 1;
  if (x)
    y-;
  else
    y++;

  assert
    (y==2 || y==3);


}
```

⇒

```
int main () {
  int x, y;
  y1 = 1;
  if(x0)
    y2 = y1-1;
  else
    y3 = y1+1;

  y4 = x0 ? y2 : y3;

  assert
    (y4==2 || y4==3);
}
```

⇒

$$((y_1 = 8) \wedge (y_2 = y_1 - 1) \wedge (y_3 = y_1 + 1) \wedge (y_4 = x_0 ? y_2 : y_3))$$

$$\rightarrow ((y_4 \leftrightarrow 2) \vee (y_4 \leftrightarrow 3))$$