Predicate Abstraction with CPAchecker

Philipp Wendler

UNIVERSITÄT PASSAU

SoSy-Lab
Software Systems

Predicate Abstraction

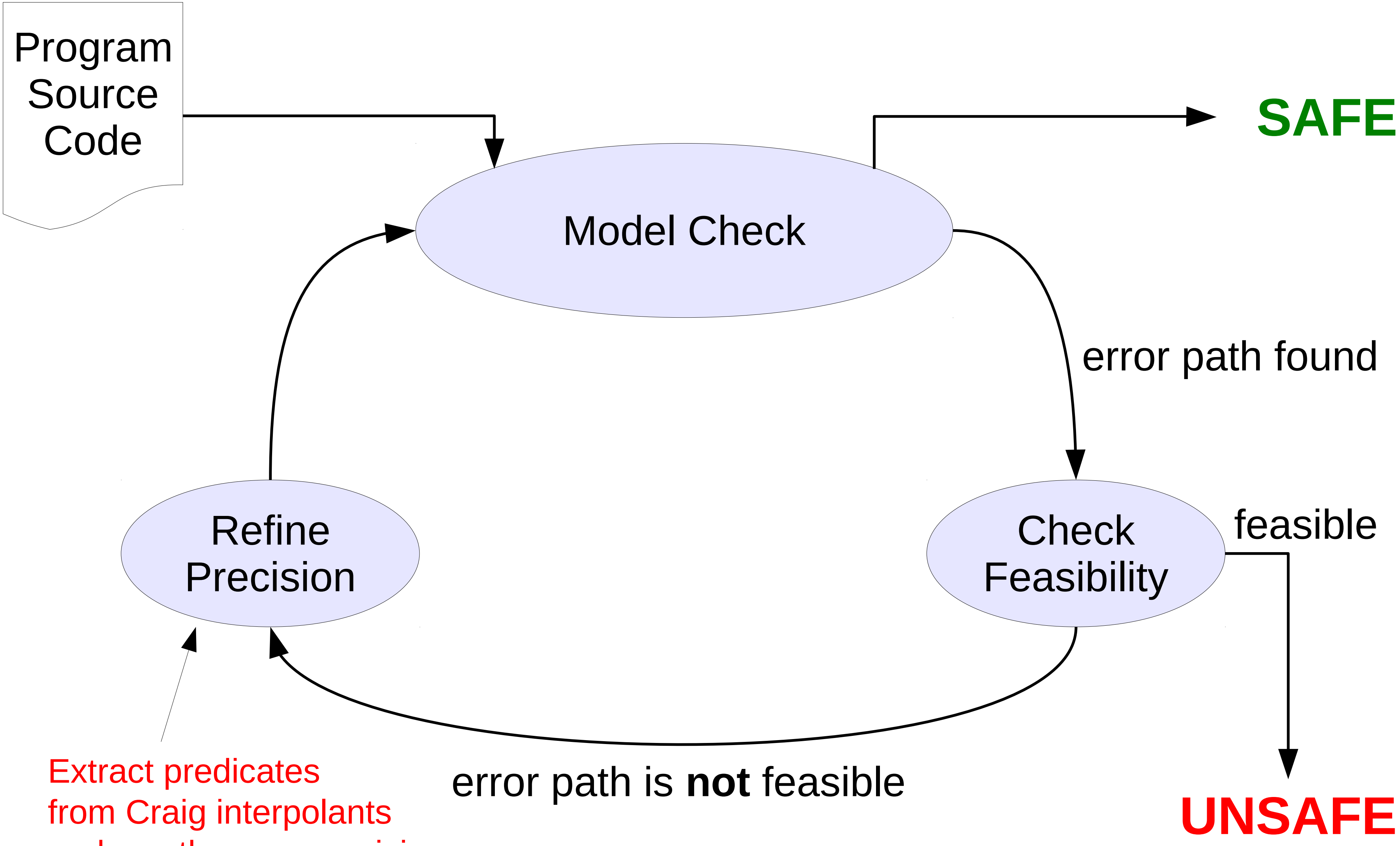
- Traditional abstract domain for software model checking
- Powerful but expensive
- Given finite set π of predicates over program variables (precision),
  abstract state is boolean combination of predicates
- Predicates are usually atoms such as (x > 0)
- Abstract state is represented as BDD
- SMT solver is used for computing successors
  (Given a state and a program statement,
  what combination of predicates holds afterwards?)

Predicate Abstraction (2)

2 possibilities:

- Cartesian abstraction:
  - Strongest conjunction of predicates
  - Looses relations between predicates,
    e.g. (x > 0) => (y > 0)
- Boolean abstraction:
  - Strongest boolean combination of predicates
  - Uses All-SMT query over predicates
    for successor computation

Reminder: CEGAR



Extract predicates from Craig interpolants and use them as precision

Example Program

Program

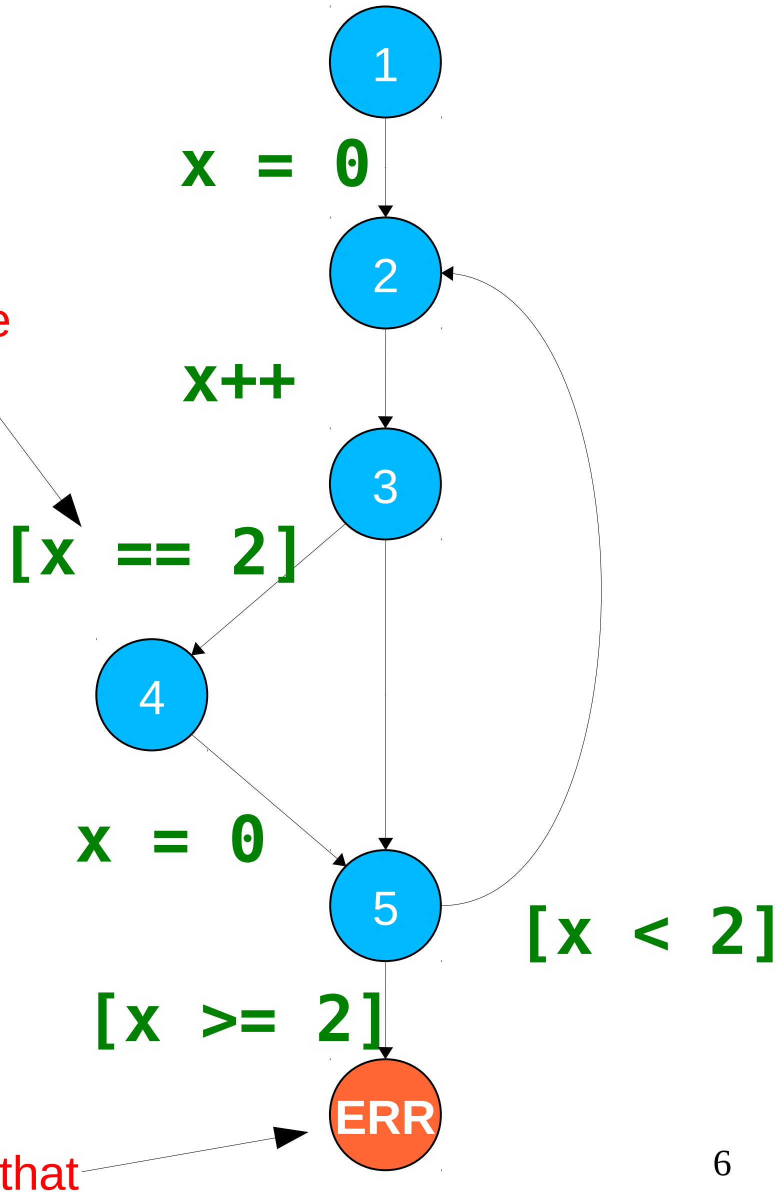
```
int x = 0;
while (true)
  x++;

  if (x == 2)

    x = 0;

  assert(x < 2);
```

Control-Flow Automaton

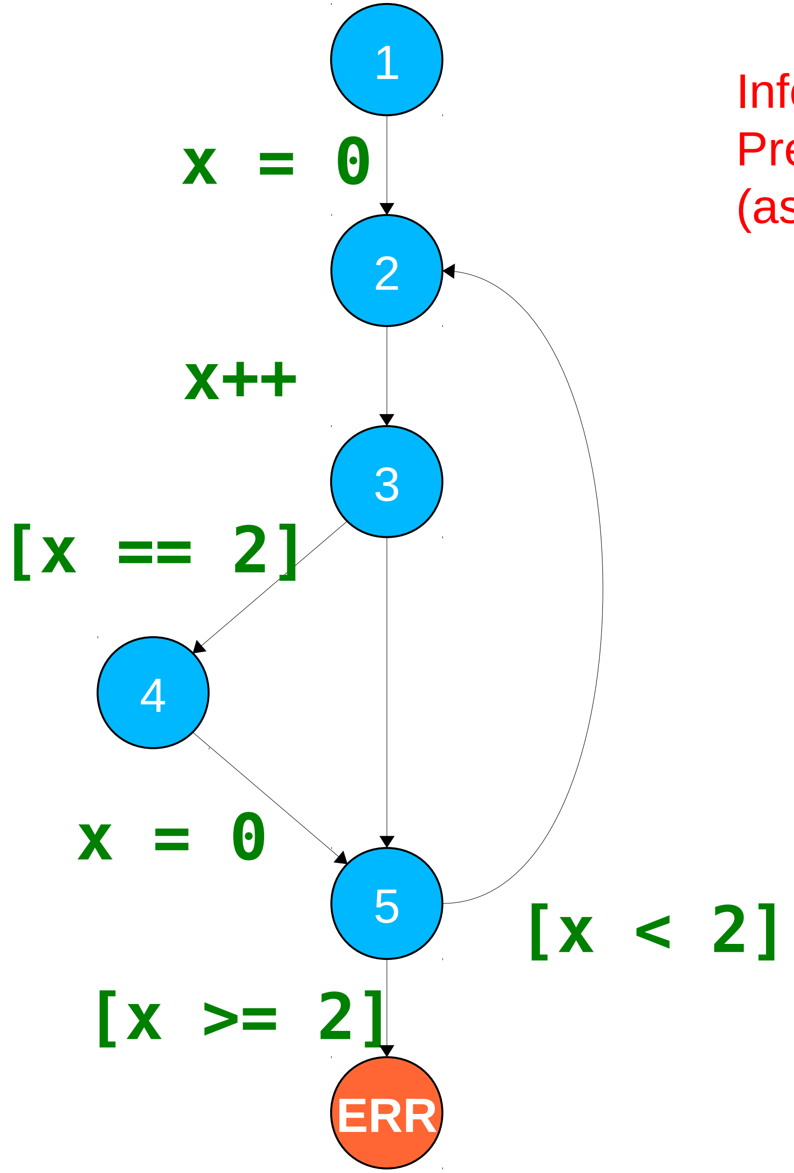


Conditional Edge (Assumption)

Error Location that violates specification

Predicate Abstraction

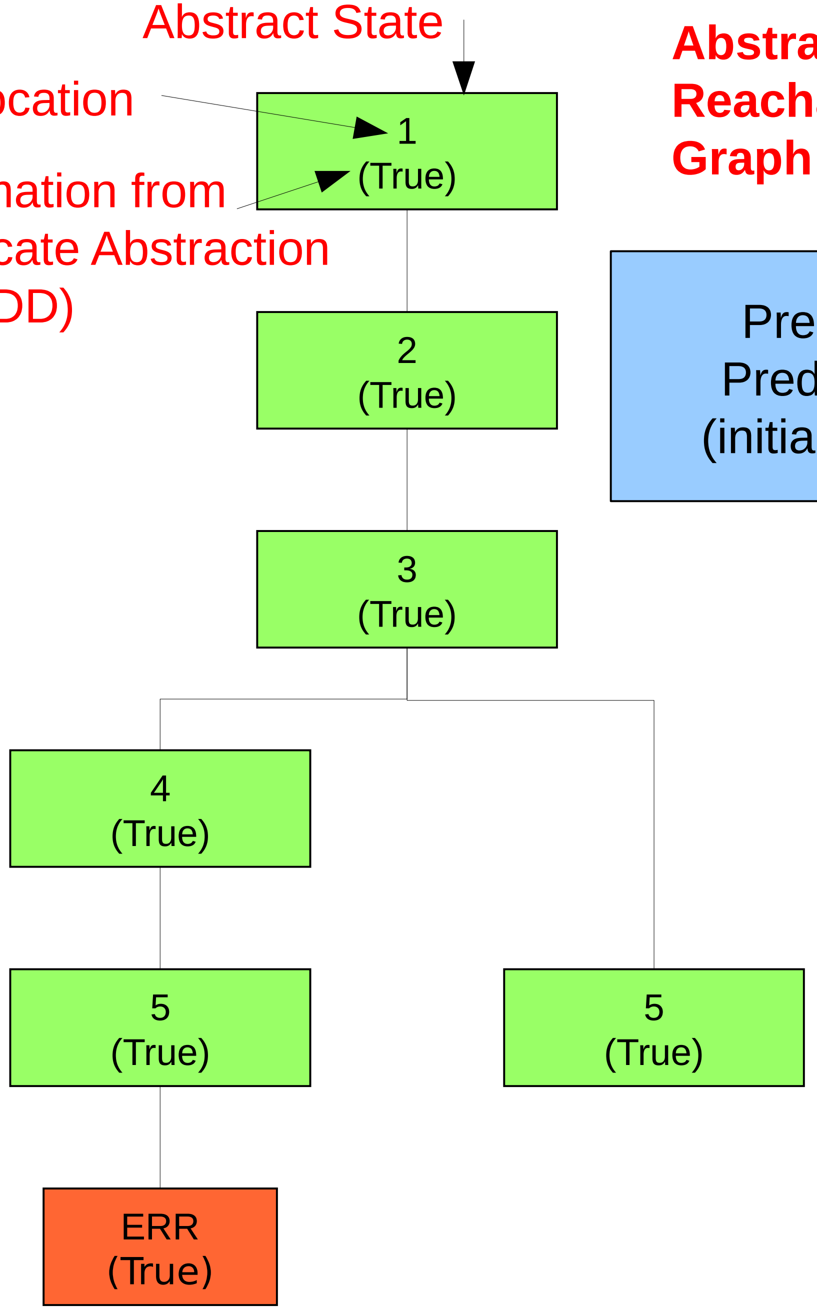
Control-Flow Automaton



Abstract State

Location

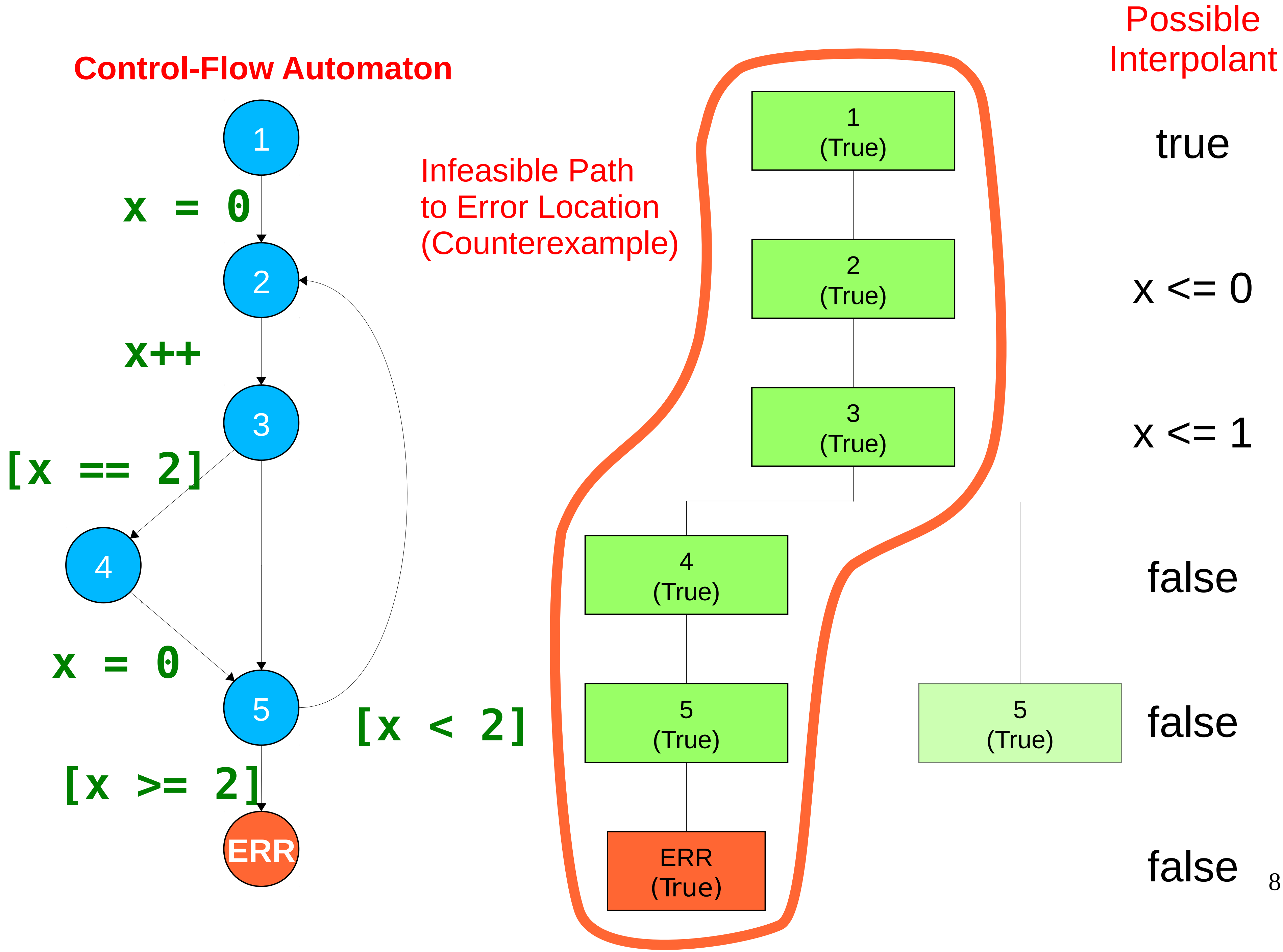
Information from Predicate Abstraction (as BDD)



Abstract Reachability Graph

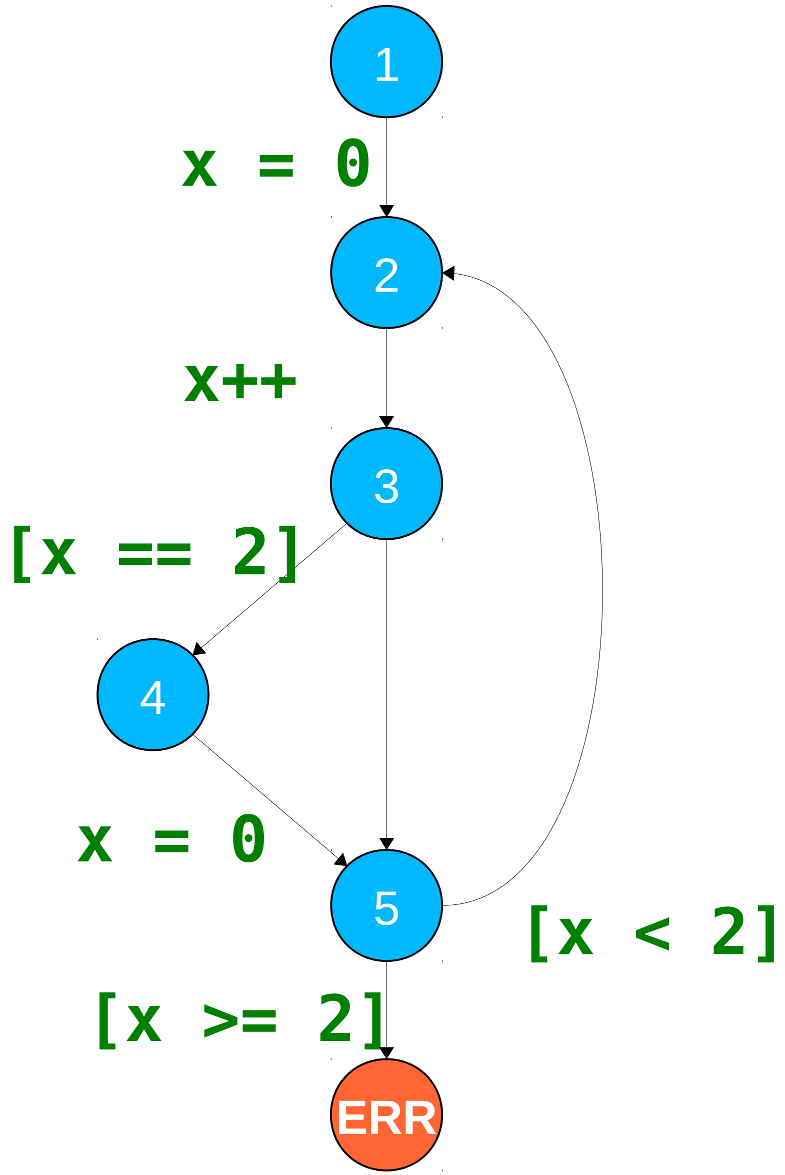
Precision = Predicate set (initially empty)

Predicate Abstraction

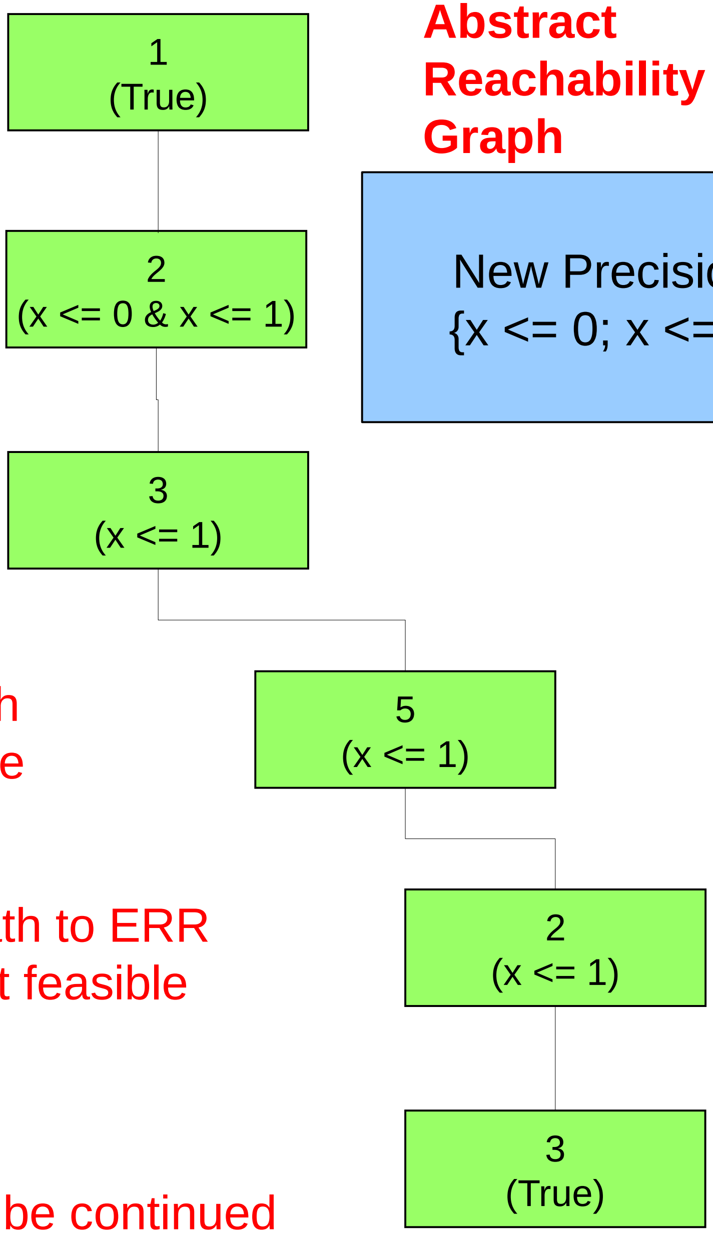


Predicate Abstraction

Control-Flow Automaton



Recomputed Abstract Reachability Graph



New Precision:
{x <= 0; x <= 1}

Left branch not feasible

Path to ERR not feasible

To be continued

Demo

- Run CPAchecker with SBE on induction2.c
- ARG
- ARGRefinements
- Predicates from predmap.txt
- Introduce bug in program
- Error path

Optimizations

- Lazy abstraction:
  - Different predicates per location and per path
  - Incremental analysis instead of restart from scratch after refinement
- Adjustable-Block Encoding:
  - Handle loop-free blocks of statements at once
  - Abstract only between blocks
    (less abstractions, less refinements)

Demo 2

- Run CPAchecker with ABE-L on induction2.c
- ARG
- Predicate Mapping from predmap.txt

CPAchecker

- Framework for Software Verification
  - Written in Java
  - Open Source: Apache 2.0 License
  - 38 contributors so far
    from 7 universities/institutions
  - 280.000 lines of code
    (170.000 without blank lines and comments)
  - Started 2007

http://cpachecker.sosy-lab.org

CPAchecker

- Among world's best software verifiers:
  http://sv-comp.sosy-lab.org/2014/results/
- In 3 consecutive years:
  http://sv-comp.sosy-lab.org/2013/results/
  http://sv-comp.sosy-lab.org/2012/results/
- Used for Linux driver verification
  with real bugs found and fixed in Linux

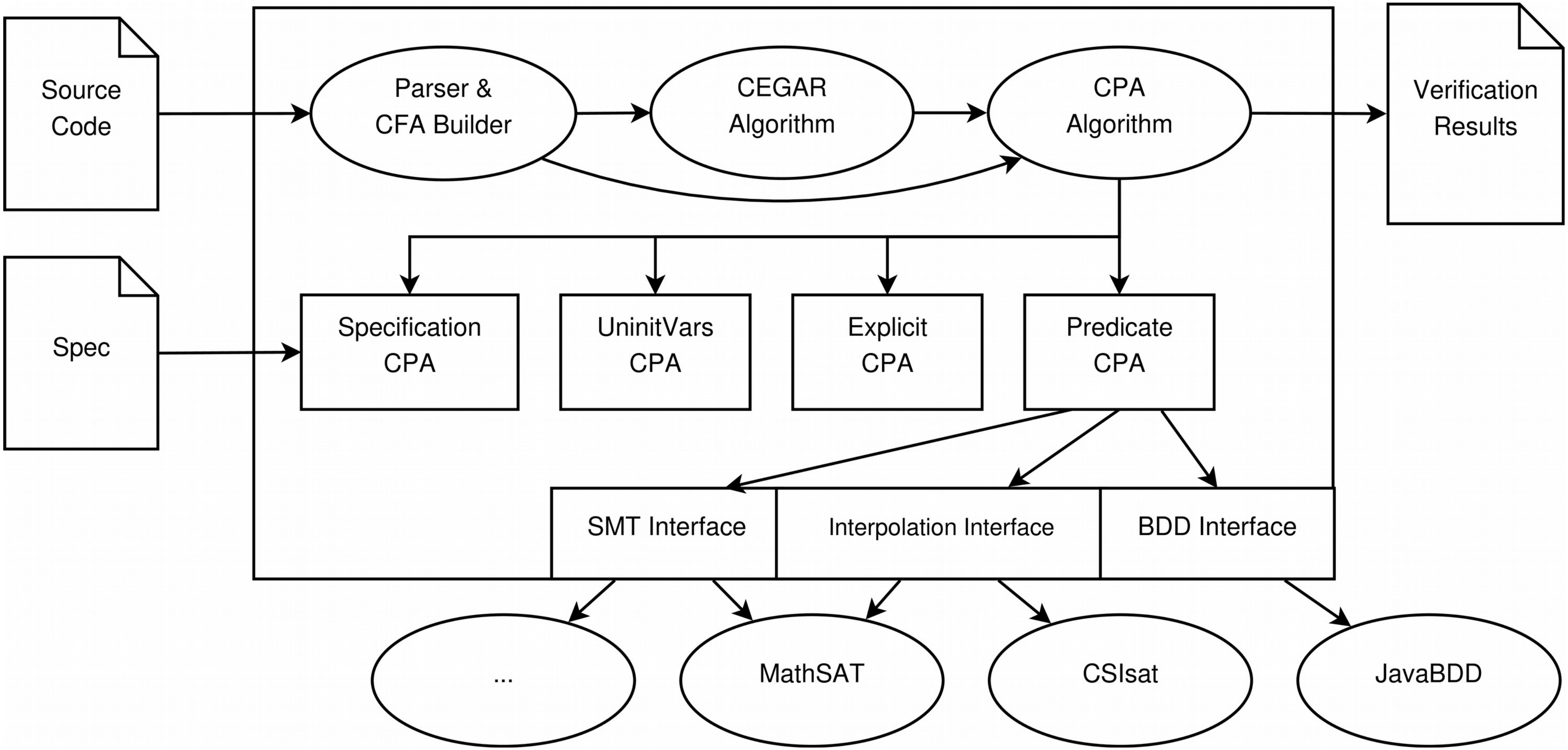
CPAchecker

- Every analysis is implemented as a “Configurable Program Analysis” (CPA)
- E.g. predicate abstraction, explicit-value analysis, intervals, octagon, BDDs, and more
- Algorithms are central and implemented only once
- Completely modular,
  and thus flexible and easily extensible

CPAchecker

- Further available analyses:
  - IMPACT algorithm
  - Bounded model checking
  - k-Induction
  - Conditional Model Checking

CPAchecker


Source Code
Spec
Parser & CFA Builder
CEGAR Algorithm
CPA Algorithm
Verification Results
Specification CPA
UninitVars CPA
Explicit CPA
Predicate CPA
SMT Interface
Interpolation Interface
BDD Interface
...
MathSAT
CSIsat
JavaBDD

Try CPAchecker

- Online at Google AppEngine:
  http://cpachecker.appspot.com
- Download for Linux/Windows:
  http://cpachecker.sosy-lab.org
  - Run `scripts/cpa.sh` | `scripts\cpa.bat`
  - `-predicateAnalysis <FILE>`
  - Windows/Mac: -setprop cpa.predicate.solver=smtinterpol
- Example program: http://bit.ly/1lpipUv
- Look at output / CPALog.txt for problems
- Open .dot files with dotty / xdot (www.graphviz.org)
- If there is a counterexample:
  `scripts/report-generator.py`

Specification

- Model Checkers check only what you specified
- CPAchecker's default:
  - Label `ERROR`
  - Calling function `__assert_fail()`
  - `assert(pred)` needs to be pre-processed
- SV-COMP:
  - Calling function `__VERIFIER_error()`
  - `-spec sv-comp-reachability`

Limitations

- Of presented analysis:
  - Linear arithmetic over reals
    (no overflows, no bit operators)
  - No checks for memory safety
  - Heap allocations with bounded size
- Other analyses do not have these limitations
- For bitvectors:
  `-predicateAnalysis-bitprecise`