

Abstraction

<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Motivating example

```
1:  int sum(int from, int to) {
2:      int total = 0;
3:      for (int i = from; i <= to; i++) {
4:          total += i;
5:      }
6:      return total;
7:  }

8:  main() {
9:      int x = sum(1, 1000);
10:     assert(x > 0);
11: }
```

Abstraction

- Goal: smaller reachable program state space
- Approaches
 - Reducing the size of variables' data domains
 - Ignoring concrete values of certain variables
- Benefits
 - Mitigating the state space explosion
 - Improved scalability (performance)

Data abstraction

- Using abstract domains for program variables
- Tracking only abstract states of the program
- Abstract state = set of concrete states
- Process: mapping **concrete** to **abstract**
 - data types, values, operations, program states

Example: Signs abstraction

- Abstract data type
 - `int` \rightarrow { NEG, ZERO, POS }

Example: Signs abstraction

- Abstract data type
 - `int` \rightarrow { NEG, ZERO, POS }
- Abstract values
 - $\alpha(x) \subseteq$ { NEG, ZERO, POS }
- Abstract operation +

	NEG	ZERO	POS
NEG	{ NEG }	{ NEG }	{ NEG, ZERO, POS }
ZERO	{ NEG }	{ ZERO }	{ POS }
POS	{ NEG, ZERO, POS }	{ POS }	{ POS }

Construction of abstract programs

- Transformation of program source code

```
int x = 0;  
  
...  
  
int y;  
y = x + 2;
```

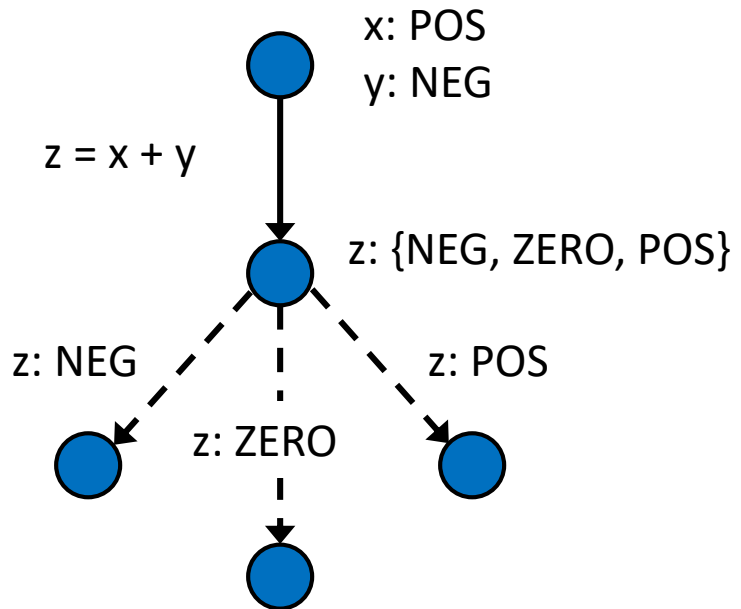


```
Signs x = Signs.ZERO;  
  
...  
  
Signs y;  
y = Signs.add(x, Signs.POS);
```

Abstract state space

- Non-deterministic choice
 - assignment, branching condition (if-else, loops)

```
int x = 5;  
int y = -2;  
z = x + y;
```



Other simple data abstractions

- Interval abstraction
 - Example: $x < 0$, $0 \leq x \leq 10$, $x > 10$
- Combining intervals with concrete values
 - Example: $x < 0$, $x = 0$, $x = 1$, $x = 2$, $x = 3$, $x = 4$, $x > 4$

Predicate abstraction



Predicate abstraction

- Data type
 - Predicates about program variables
 - Theories: linear integer arithmetic, equality, arrays
 - Example: $x = 0$, $x > 0$, $y + z \geq 2$, $u = v$, $select(a, 1) = 5$
- Abstract state
 - Some valuation of all the predicates

Example

```
1:  int sum(int from, int to) {
2:      int total = 0;
3:      for (int i = from; i <= to; i++) {
4:          total += i;
5:      }
6:      return total;
7:  }

8:  int x = sum(1, 1000);
9:  assert(x > 0);
```

Boolean program

```
bool P1 = false;
bool P2 = false;

// int total = 0;
P2 = true;

// int i = from;
P1 = *;

// total += i;
if (P1 && P2) P2 = true;
else P2 = *;
```

Predicates

P1: $i > 0$

P2: $total \geq 0$

Deriving predicate valuations

- Weakest preconditions
 - Predicate p : $\text{total} \geq 0$
 - Statement s : $\text{total} += i$;
 - $\text{WP}(s,p) \equiv \text{total} + i \geq 0$
- Querying the SMT solver
 - Example: $p1 \ \&\& \ !p2 \rightarrow \text{WP}(s,p)$ is valid ?
- Processing results
 - 1) $p1 \ \&\& \ !p2 \rightarrow \text{WP}(s,p)$ is valid \rightarrow if $(p1 \ \&\& \ !p2)$ $p = \text{true}$;
 - 2) $p1 \ \&\& \ !p2 \rightarrow \text{WP}(s,p)$ is valid \rightarrow if $(p1 \ \&\& \ !p2)$ $p = \text{false}$;
 - 3) both valid or none valid \rightarrow if $(p1 \ \&\& \ !p2)$ $p = *$;

Optimizations

- Goal: reduce the number of queries for SMT
- Possible approaches
 - Compute new valuation only for predicates that refer to variables modified by the given concrete assignment statement
 - We must be very careful though: aliasing
 - For generating branches of the big `if-else` statements in the abstract boolean program, consider only predicates that refer to variables read by the assignment statement

Verification using predicate abstraction

- Using model checker for boolean programs
 - Much easier task than for general programs (C, Java)
 - Well-known optimizations: symbolic model checking
- Practical challenges
 - Translating counterexamples back to source code
 - Encoding properties into reachability of assertions

Abstraction: characteristics



Abstraction: characteristics



Over-approximation

- Abstract program captures **all possible behaviors** of the original concrete program
 - Behavior: possible control flow path, thread interleaving
- Purpose: complete verification (all reachable states)
- Examples
 - Simple data abstraction
 - Predicate abstraction
- Problem: imprecise abstraction
 - Captures some infeasible execution paths → **spurious errors**
 - Branch conditions replaced with a non-deterministic choice

Under-approximation

- Abstract program captures only a **certain subset** of all **possible behaviors** of the concrete program
 - selected thread interleavings, reduced data domains
- Purpose: fast error detection (subset of reachable states)
- Examples
 - Normal tests (used in SW industry)
 - State space traversal with heuristics
 - Context-bounded search (traversal)
 - Bounded model checking in general
- Problem: imprecise abstraction
 - Omits some feasible execution paths → **missed errors**

Abstractions: characteristics



Over-approximation

Error in abstraction



Error in concrete program

Error-free abstraction



Error-free concrete program

Spurious errors

Under-approximation

Error in abstraction



Error in concrete program

Error-free abstraction



Error-free concrete program

Missed errors

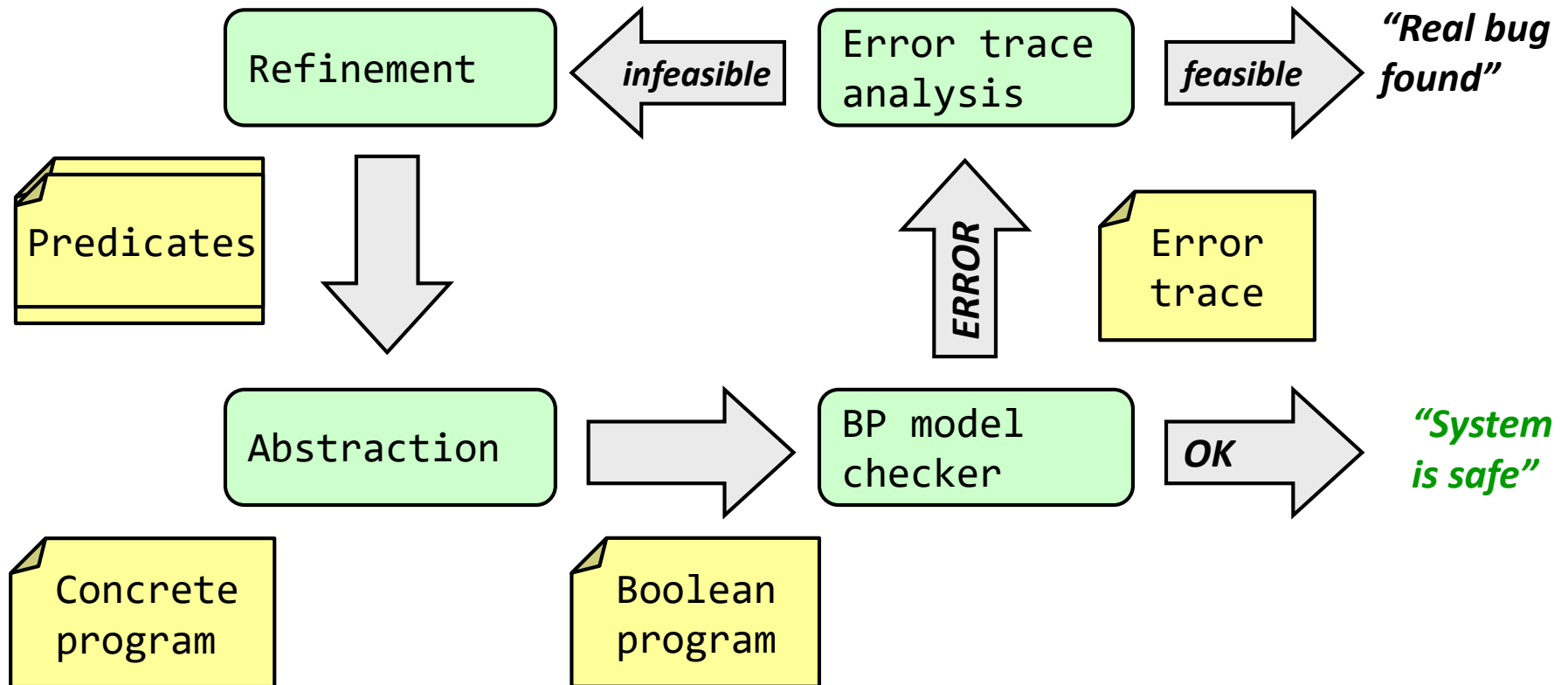
Abstraction: issues

- Very hard to get right
 - Too precise → state explosion
 - Too coarse → spurious errors
- Possible remedy
 - Start with coarse abstraction
 - Employ iterative refinement



Counter-Example Guided Abstraction Refinement

- Automated iterative refinement based on spurious errors



Picture created by Ondřej Šerý

Challenges

- Checking error trace feasibility
- Inferring additional predicates

Checking error trace feasibility

- Simulate the abstract error trace on the concrete program
- Record the path condition *PaC* using symbolic execution
 - Options selected at choice points (if-else, loops, non-determinism)
- Create path formula that encodes the whole error trace
 - The `assume` statement: clauses from the *PaC* (selected branches)
- Check satisfiability of the path formula (query the SMT solver)
- Example
 - Error trace

```
index = 1; total = total + index; assume index > 1000
```
 - Path formula

```
(index0 = 1) && (total1 = total0 + index0) && (index0 > 1000)
```

Inferring additional predicates

- Divide path formula ϕ into two parts ϕ^- and ϕ^+
 - such that $\phi^- \ \&\& \ \phi^+$ is unsatisfiable
- Then derive a Craig interpolant ψ for ϕ^- and ϕ^+
 - Logic formula ψ such that
 - $\phi^- \rightarrow \psi$, $\phi^+ \ \&\& \ \psi$ is unsatisfiable, and
 - ψ uses symbols common to ϕ^- and ϕ^+
- Finally generate additional predicates from ψ
- Example
 - Path formula
 - $(\text{index0} = 1) \ \&\& \ (\text{total1} = \text{total0} + \text{index0}) \ \&\& \ (\text{index0} > 1000)$
 - $\phi^-: \text{index0} = 1 \ \&\& \ \text{total1} = \text{total0} + \text{index0}$
 - $\phi^+: \text{index0} > 1000$
 - $\psi: \text{index0} = 1$ // newly inferred predicate in this case
- Disclaimer
 - Bad choices of inferred predicates may lead to non-termination
 - Tools generate predicates that may look strange (not intuitive)

- Static Driver Verifier (SDV)
 - SLAM: verification engine that uses CEGAR
- Purpose
 - Analyzing third party Windows device drivers
 - Specific rules about proper usage of Windows kernel API
 - Major source of kernel crashes (infamous “blue screens”)
 - Drivers have feasible code size and a strict environment
- Many extensions developed in the last decade
- Additional information
 - <https://www.microsoft.com/en-us/research/project/slam/>
 - Many research papers, slides, download, user guides

Optimizations

- Lazy abstraction
 - Set of predicates specific to each code location
 - Tools: BLAST
- Method summaries
 - Logic formula relating inputs and outputs
 - Summaries computed using interpolants
 - Tools: Whale, FunFrog, ...

Tools

- BLAST
 - <https://www.sosy-lab.org/~dbeyer/Blast/index-epfl.php>
- CPAchecker
 - <http://cpachecker.sosy-lab.org/>
- UFO/Whale
 - <https://bitbucket.org/arieg/ufo/wiki/Home>
- Wolverine
- ... and many others

Further reading

- T. Ball, R. Majumdar, T. Millstein, and S.K. Rajamani. **Automatic Predicate Abstraction of C Programs**. PLDI 2001
- E.M. Clarke, D. Kroening, N. Sharygina, and K. Yorav. **Predicate Abstraction of ANSI-C Programs Using SAT**. Formal Methods in System Design, 25(2-3), 2004
- T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. **Lazy Abstraction**. POPL 2002
- D. Beyer, T.A. Henzinger, R. Jhala, and R. Majumdar. **The Software Model Checker BLAST**. STTT, 9(5-6), 2007
- K.L. McMillan. **Lazy Abstraction with Interpolants**. CAV 2006
- A. Albarghouthi, A. Gurfinkel, and M. Chechik. **Whale: An Interpolation-based Algorithm for Inter-procedural Verification**. VMCAI 2012
- T. Ball, V. Levin, and S.K. Rajamani. **A Decade of Software Model Checking with SLAM**. Communications of the ACM, 54(7), 2011