NSWI101: System Behaviour Models and Verification

12. Counter-Example Guided Abstraction Refinement

Jan Kofroň

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of
Distributed and
Dependable
Systems

Motivation

Verification of programs is undecidable problem

- due to loops, threads, recursion, dynamic memory allocation, ...

In many cases, we can verify them

- however, not using just brute force
- by employing kind of abstraction

One significant source of undecidability is data non-determinism

- user input, random values, ...

Motivation

Verification of programs is undecidable problem

- due to loops, threads, recursion, dynamic memory allocation, ...

In many cases, we can verify them

- however, not using just brute force
- by employing kind of abstraction

One significant source of undecidability is data non-determinism

- user input, random values, ...

Counter-Example Guided Abstraction Refinement—CEGAR

CEGAR

```
x = *;
while (x < 0)
  x++;
assert (x != 0);
```

System model

assertion violations

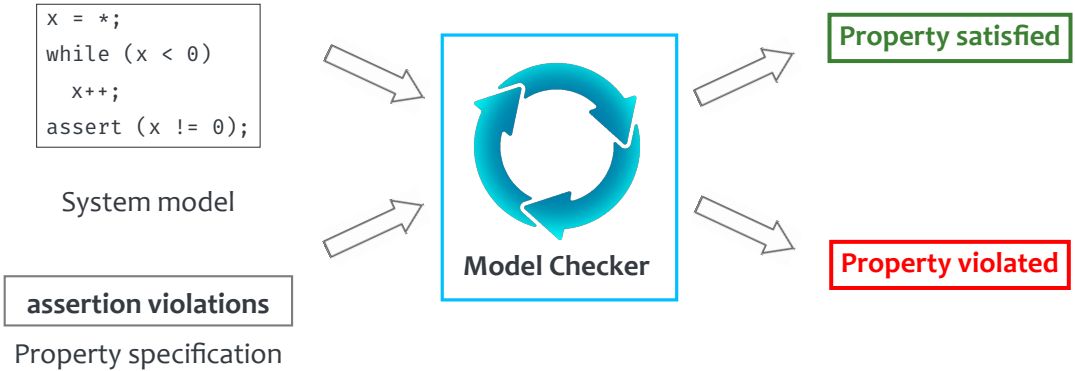
Property specification

Model Checker

Property satisfied

Property violated

CEGAR


x = *;
while (x < 0)
x++;
assert (x != 0);
System model
assertion violations
Property specification
Model Checker
Property satisfied
Property violated

Overview

1. Initial abstraction is created via replacing all data in tests with non-deterministic Boolean values (predicates) and all data updates with skips
   - Boolean program over-approximates original program

Overview

1. Initial abstraction is created via replacing all data in tests with non-deterministic Boolean values (predicates) and all data updates with skips
   - Boolean program over-approximates original program
2. Boolean program is model checked
   - number of program paths is finite → it always terminates
   - no error found → **program is safe**, terminate
   - error found → analyse error and either
     - it is **real error**—report it and terminate, or
     - it is **spurious**—refine abstraction, i.e., extend set of predicates

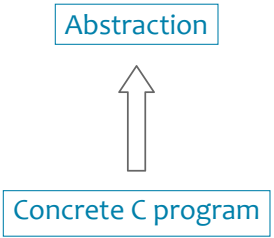
Overview

1. Initial abstraction is created via replacing all data in tests with non-deterministic Boolean values (predicates) and all data updates with skips
   - Boolean program over-approximates original program
2. Boolean program is model checked
   - number of program paths is finite → it always terminates
   - no error found → **program is safe**, terminate
   - error found → analyse error and either
     - it is **real error**—report it and terminate, or
     - it is **spurious**—refine abstraction, i.e., extend set of predicates
3. Repeat from step 2
   - May not terminate—inevitable due to undecidability of software verification

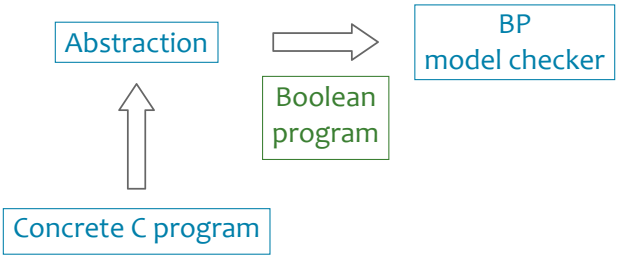
Overview

Concrete C program

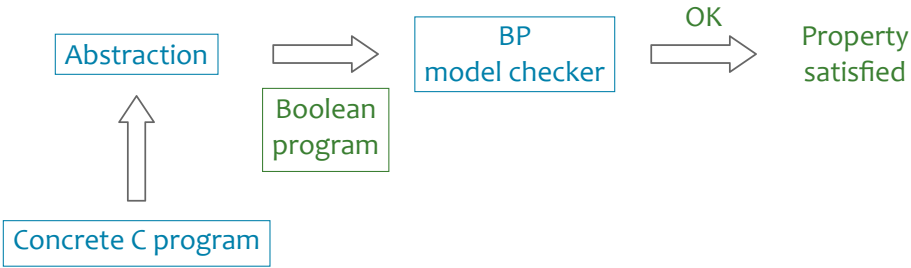
Overview



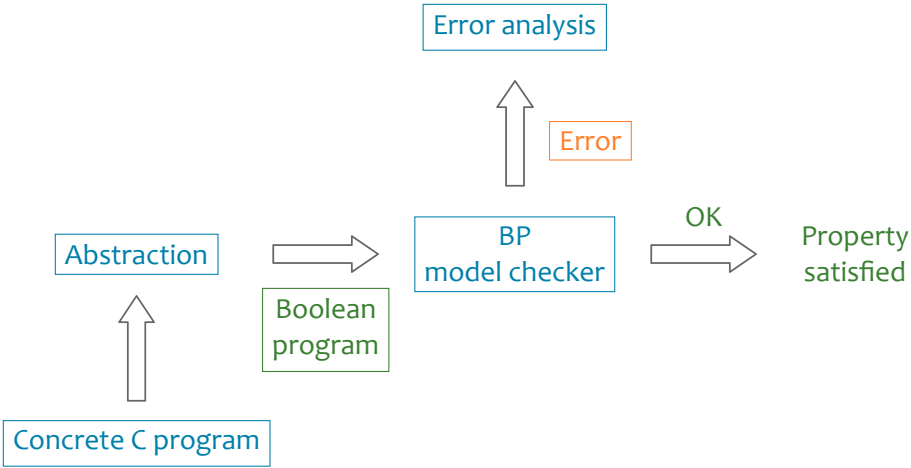
Overview


Abstraction
BP
model checker
Boolean
program
Concrete C program

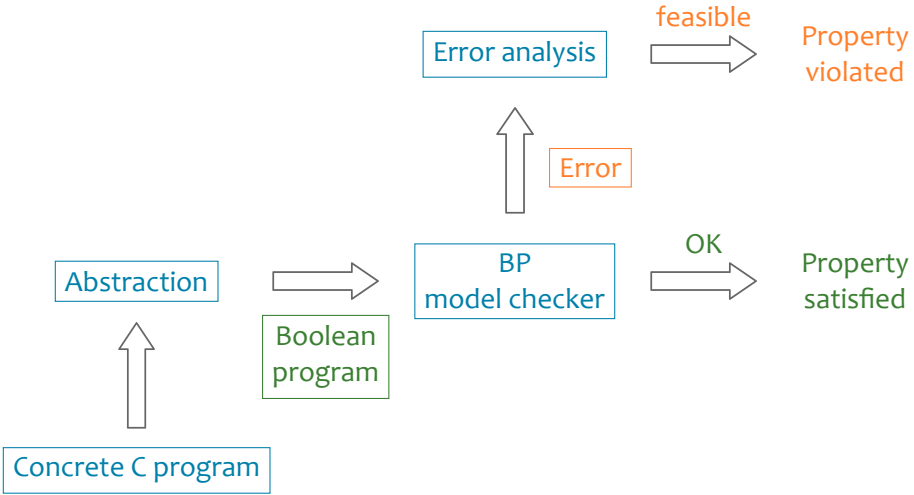
Overview


Abstraction
Concrete C program
Boolean program
BP model checker
OK
Property satisfied

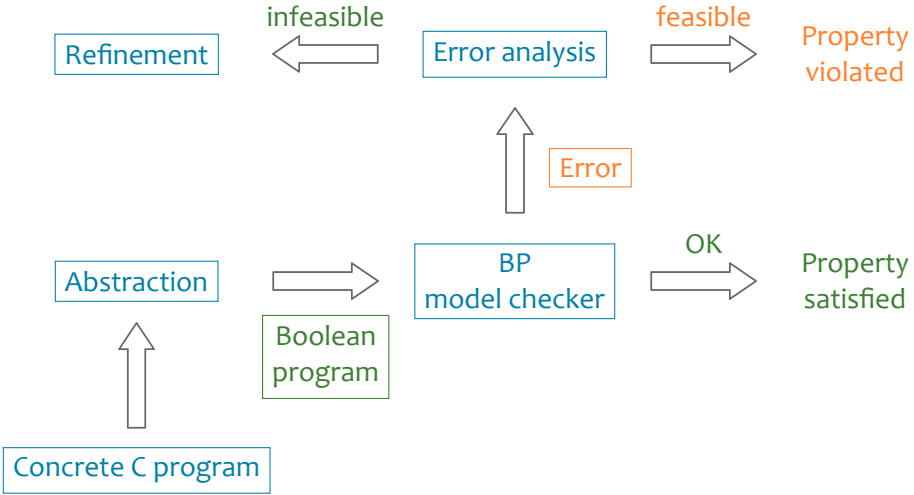
Overview


Error analysis
Error
OK
Abstraction
BP
model checker
Property
satisfied
Boolean
program
Concrete C program

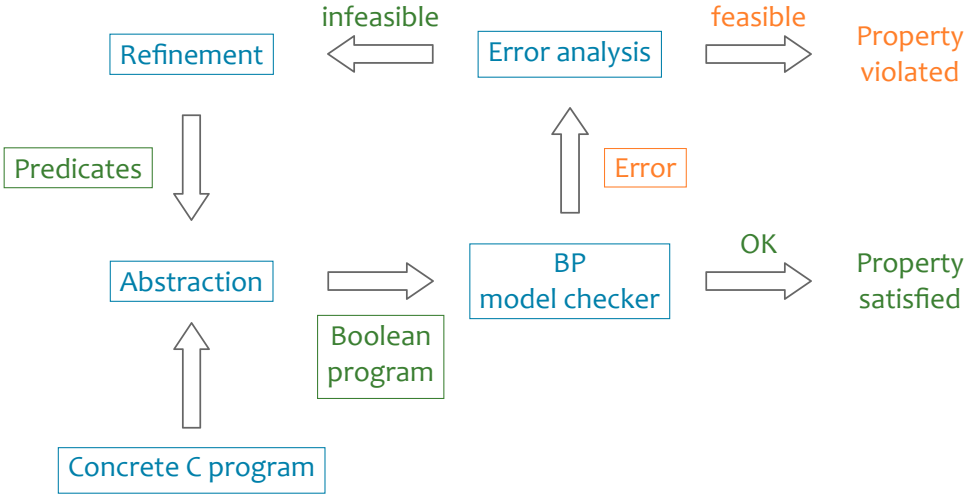
Overview


Error analysis
feasible
Property violated
Error
BP model checker
OK
Property satisfied
Abstraction
Boolean program
Concrete C program

Overview


Refinement
infeasible
Error analysis
feasible
Property violated
Error
Abstraction
BP model checker
OK
Property satisfied
Boolean program
Concrete C program

Overview


Refinement
infeasible
Error analysis
feasible
Property violated
Predicates
Error
Abstraction
BP model checker
OK
Property satisfied
Boolean program
Concrete C program

Challenges

Two operations of CEGAR loop are hard:

- Checking error trace feasibility
- Performing refinement, i.e., finding new predicates

Error Trace Feasibility

We need to simulate abstract error trace on concrete program:

1. record path condition using symbolic execution
2. create path formula encoding error trace found
3. check path formula satisfiability using SMT solver
   - 3.1 satisfiable formula $\rightarrow$ real error
   - 3.2 unsatisfiable formula $\rightarrow$ spurious error $\rightarrow$ need for abstraction refinement

Example

Start with empty set of predicates—data replaced by non-deterministic values:

```
x = ?

if (x >= 20)
  x = x % 20;

if (x >= 10)
  x = x / 2;

assert(x < 10);
```

```
x = ?

if (*)
  skip;

if (*)
  skip;

assert(*);
```

Example

Start with empty set of predicates—data replaced by non-deterministic values:

```
x = ?

if (x >= 20)
  x = x % 20;

if (x >= 10)
  x = x / 2;

assert(x < 10);
```

```
x = ?

if (*)
  skip;

if (*)
  skip;

assert(*);
```

Orange means **false**, **green** means **true**

Example

Model check Boolean program and perform symbolic execution along error path:

```
x = ?

if (*)
  skip;

if (*)
  skip;

assert(*);
```

```
x = ?

if (x >= 20)
  x = x % 20;

if (x >= 10)
  x = x / 2;

assert(x < 10);
```

Symbolic execution yields path predicates: $\{\neg(x \geq 20), \neg(x \geq 10), \neg(x < 10)\}$

Example

- Path predicates form path condition: $P_1 \equiv (x < 20) \wedge (x < 10) \wedge (x \geq 10)$

Example

- Path predicates form path condition: $P_1 \equiv (x < 20) \wedge (x < 10) \wedge (x \geq 10)$
- P_1 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed

Example

- Path predicates form path condition: $P_1 \equiv (x < 20) \wedge (x < 10) \wedge (x \geq 10)$
- P_1 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed
- Predicate to add is computed using interpolation over predicates of $P_1 : (A, B)$

Example

- Path predicates form path condition: $P_1 \equiv (x < 20) \wedge (x < 10) \wedge (x \geq 10)$
- P_1 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed
- Predicate to add is computed using interpolation over predicates of $P_1 : (A, B)$
- Interpolant is computed for each program location

Example

- Path predicates form path condition: $P_1 \equiv (x < 20) \wedge (x < 10) \wedge (x \geq 10)$
- P_1 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed
- Predicate to add is computed using interpolation over predicates of $P_1 : (A, B)$
- Interpolant is computed for each program location
- Predicate for refining abstraction $\equiv$ an interpolant before the first inconsistent transition:
  - $A = (x < 20) \wedge (x < 10)$
  - $B = (x \geq 10)$
  - Interpolant I_1 of (A, B) is **x < 10**

EXAMPLE

Model check Boolean program and perform symbolic execution along error path:

```
x = ?

if (*)
  assume(*);

if (!I_1)
  assume(*);

assert(I_1);
```

Example

Model check Boolean program and perform symbolic execution along error path:

```
x = ?

if (*)
  assume(*);

if (!I_1)
  assume(*);

assert(I_1);
```

```
x = ?

if (x >= 20)
  x = x % 20;

if (x >= 10)
  x = x / 2;

assert(x < 10);
```

Symbolic execution path predicates: $\{\neg(x_1 \geq 20), x_1 \geq 10, x_2 = x_1/2, \neg(x_2 < 10)\}$

Example

- Path predicates form path condition:
  $P_2 \equiv \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2 \wedge \neg(x_2 < 10)$

Example

- Path predicates form path condition:
  $P_2 \equiv \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2 \wedge \neg(x_2 < 10)$
- P_2 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed

Example

- Path predicates form path condition:
  $P_2 \equiv \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2 \wedge \neg(x_2 < 10)$
- P_2 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed
- Predicate to add is computed using interpolation over predicates of P_2 : (A, B)
  - $A = \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2$
  - $B = \neg(x_2 < 10)$

Example

- Path predicates form path condition:
  $P_2 \equiv \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2 \wedge \neg(x_2 < 10)$
- P_2 is unsatisfiable $\rightarrow$ error path found is **spurious** $\rightarrow$ refinement needed
- Predicate to add is computed using interpolation over predicates of P_2 : (A, B)
  - $A = \neg(x_1 \geq 20) \wedge \neg(x_1 < 10) \wedge x_2 = x_1/2$
  - $B = \neg(x_2 < 10)$
- Interpolant I_2 of (A, B) is **x < 20**
  - new predicate to be added to refine abstract program

EXAMPLE

Model check Boolean program:

```
x = ?

if (!I_2)
  assume(I_2);

if (!(I_1))
  assume(I_2);
  assume(I_1);

assert(I_1);
```

None of the four paths violates the assert condition ⇒ **the program is safe!**

Tools: SDV and SLAM

- **Static Driver Verifier** (SDV) from Microsoft Research
- Employs results of SLAM project—verification engine that uses CEGAR
- Purpose: Analysing third party Windows device drivers
- Specific rules about proper usage of Windows kernel API
- Drivers have feasible code size and a strict environment
- `https://www.microsoft.com/en-us/research/project/slam/`

Tools: Blast

- Blast employs lazy abstraction—abstracting just those parts of state space to avoid spurious errors
- More efficient in terms of memory usage and time consumption
- Traverses just fraction of entire state space
- `http://mtc.epfl.ch/software-tools/blast/index-epfl.php`