NSWI101: System Behaviour Models and Verification

12. Counter-Example Guided Abstraction Refinement

Jan Kofroň
Motivation

Verification of programs is an 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, ...

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

\begin{verbatim}
x = *;
while (x < 0)
    x++;
assert (x != 0);
\end{verbatim}

System model

Property specification

assertion violations

Model Checker

Property satisfied

Property violated
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

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

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 $\rightarrow$ it always terminates
   - no error found $\rightarrow$ program is safe, terminate
   - error found $\rightarrow$ analyse error and either
     - it is real error—report it and terminate, or
     - it is spurious—refine abstraction, i.e., extend set of predicates

May not terminate—inevitable due to undecidability of software verification
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 $\rightarrow$ it always terminates
   - no error found $\rightarrow$ **program is safe**, terminate
   - error found $\rightarrow$ 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
Concrete C program
OVERVIEW

Concrete C program

Abstraction

Concrete C program

Abstraction
OVERVIEW

Concrete C program

Abstraction

Boolean program

BP model checker

Refinement

Predicates

Error analysis

Property satisfied

Property violated

infeasible

feasible
Overview

Concrete C program

Abstraction

Boolean program

BP model checker

OK

Property satisfied
OVERVIEW

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

Concrete C program

Abstraction

Boolean model checker

Error analysis

Error

feasible

Property violated

OK

Property satisfied

Boolean program

Concrete C program

Refinement

Predicates
OVERVIEW

Concrete C program

Abstraction

Refinement

infeasible

Error

feasible

Property violated

Property satisfied

Error analysis

BP model checker

OK

Boolean program

Jan Kofroň: Behaviour Models and Verification
OVERVIEW

Concrete C program

Abstraction

BP model checker

Error analysis

Refinement

Predicates

Boolean program

Error

infeasible

feasible

Property violated

Property satisfied

Refinement

Predicates

Abstraction

Concrete C program

Boolean program

BP model checker

Error analysis

Error
Two operations of CEGAR loop are hard:

- Checking error trace feasibility
- Performing refinement, i.e., finding new predicates
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
Start with empty set of predicates—data replaced by non-deterministic values:

\[
x = ? \\
\text{if } (x \geq 20) \\
x = x \% 20; \\
\text{if } (x \geq 10) \\
x = x / 2; \\
\text{assert}(x < 10);
\]
Start with empty set of predicates—data replaced by non-deterministic values:

\[ x = ? \]

\[ \text{if } (x \geq 20) \]
\[ x = x \% 20; \]

\[ \text{if } (x \geq 10) \]
\[ x = x / 2; \]

\[ \text{assert}(x < 10); \]

\[ \Rightarrow \]

\[ x = ? \]

\[ \text{if } (*) \]
\[ \text{skip}; \]

\[ \text{if } (*) \]
\[ \text{skip}; \]

\[ \text{assert}(*); \]

Orange means false, green means true
Example

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

\[
\text{\begin{align*}
  &x = ? \\
  \text{if} &\ (\ast) \\
  &\text{skip ;} \\
  \text{if} &\ (\ast) \\
  &\text{skip ;} \\
  \text{assert} &\ (\ast) ;
\end{align*}}
\]

\[
\text{\begin{align*}
  &x = ? \\
  \text{if} &\ (x \geq 20) \\
  &x = x \% 20 ; \\
  \text{if} &\ (x \geq 10) \\
  &x = x / 2 ; \\
  \text{assert} &\ (x < 10) ;
\end{align*}}
\]

Symbolic execution yields path predicates: \{\neg(x \geq 20), \neg(x \geq 10), \neg(x < 10)\}
Path predicates form path condition: \( P_1 \equiv (x < 20) \land (x < 10) \land (x \geq 10) \)
Path predicates form path condition: \( P_1 \equiv (x < 20) \land (x < 10) \land (x \geq 10) \)

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

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

\( P_1 \) 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: $P_1 \equiv (x < 20) \land (x < 10) \land (x \geq 10)$
- $P_1$ 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:
  - $A = (x < 20) \land (x < 10)$
  - $B = (x \geq 10)$
  - Interpolant $I_1$ of $(A, B)$ is $x < 10$
Model check Boolean program and perform symbolic execution along error path:

\[ x = ? \]

\[
\text{if } (*)
\text{ assume}(*); \\
\]

\[
\text{if } (!I_1)
\text{ assume}(*); \\
\]

\text{assert}(I_1);
Model check Boolean program and perform symbolic execution along error path:

\[ x = ? \]

if (\( \ast \))
assume(\( \ast \));

if (!I\_1)
assume(\( \ast \));

assert(I\_1);

\[ x = ? \]

if (\( x \geq 20 \))
x = x \% 20;

if (\( x \geq 10 \))
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)\}
Path predicates form path condition:

\[ P_2 \equiv \neg(x_1 \geq 20) \land \neg(x_1 < 10) \land x_2 = x_1/2 \land \neg(x_2 < 10) \]
Path predicates form path condition:
\[ P_2 \equiv \neg(x_1 \geq 20) \land \neg(x_1 < 10) \land x_2 = x_1/2 \land \neg(x_2 < 10) \]

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

\( P_2 \) is unsatisfiable \( \rightarrow \) error path found is **spurious** \( \rightarrow \) refinement needed

Predicate to add is computed using interpolation over predicates of \( P_2 \) : \( (A, B) \)

\[ A = \neg(x_1 \geq 20) \land \neg(x_1 < 10) \land x_2 = x_1/2 \]
\[ B = \neg(x_2 < 10) \]
Path predicates form path condition:
\[ P_2 \equiv \neg(x_1 \geq 20) \land \neg(x_1 < 10) \land x_2 = x_1/2 \land \neg(x_2 < 10) \]

- \( P_2 \) is unsatisfiable \( \rightarrow \) error path found is \textbf{spurious} \( \rightarrow \) refinement needed
- Predicate to add is computed using interpolation over predicates of \( P_2 \) : \((A, B)\)
  - \( A = \neg(x_1 \geq 20) \land \neg(x_1 < 10) \land x_2 = x_1/2 \)
  - \( B = \neg(x_2 < 10) \)
- Interpolant \( I_2 \) of \((A, B)\) is \( x < 20 \)
  - new predicate to be added to refine abstract program
Model check Boolean program:

\[
x = ?
\]

\[
\text{if } (!I_2) \\
\text{assume}(I_2);
\]

\[
\text{if } (!(!I_1)) \\
\text{assume}(I_2); \\
\text{assume}(I_1);
\]

\[
\text{assert}(I_1);
\]

⇒ 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
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