Safety Model Checking with Complementary Approximations

Jianwen Li, Shufang Zhu, Yueling Zhang, Geguang Pu, Moshe Y. Vardi

Presented by Marek Chalupa

IA072 – spring 2021, March 12, 2021

Complementary Approximate Reachability (CAR)

Technique for verifying invariant properties of boolean transition systems.

- Inspired by symbolic reachability and IC3/PDR
- Aims to find a property violation as fast as possible
- Works complementary to IC3/PDR

Preliminaries

We use propositional logic (boolean variables, connectives $\wedge, \vee, \neg, \implies, \iff$). A variable or its negation is called <u>literal</u>, conjunction of literals is <u>cube</u> and disjunction of literals is <u>clause</u>.

Preliminaries

We use propositional logic (boolean variables, connectives $\wedge, \vee, \neg, \implies, \iff$). A variable or its negation is called literal, conjunction of literals is cube and disjunction of literals is clause.

A formula over variables $X = \{x_1, \ldots, x_n\}$ is satisfiable (SAT) if there exists an assignment $\alpha : X \to \{true, false\}$ to its variables that evaluates the formula to *true*. We allow α to be partial function, in which case we call the assignment partial assignment.

Preliminaries

We use propositional logic (boolean variables, connectives $\wedge, \vee, \neg, \implies, \iff$). A variable or its negation is called literal, conjunction of literals is cube and disjunction of literals is clause.

A formula over variables $X = \{x_1, \ldots, x_n\}$ is satisfiable (SAT) if there exists an assignment $\alpha : X \to \{true, false\}$ to its variables that evaluates the formula to *true*. We allow α to be partial function, in which case we call the assignment partial assignment.

If a formula ϕ is unsatisfiable (UNSAT), we can find its minimal unsat core (MUC) which is a subformula ψ of ϕ which is UNSAT but every subformula of ψ is SAT.

Preliminaries – cont.

Boolean transition system (BTS) is a tuple $(\bar{x}, I, T)$ consisting of

- state (boolean) variables $\bar{x} = \{x_1, x_2, ..., x_n\}$,
- a propositional formula $I(\bar{x})$ describing initial states,
- a propositional formula $T(\bar{x}, \bar{x}')$ describing transition relation

Where $\bar{x}'$ are the next-state versions of state variables: $\{x_1', x_2', ..., x_n'\}$.

Preliminaries – cont.

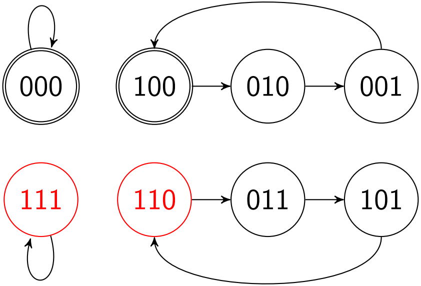
Boolean transition system (BTS) is a tuple $(\bar{x}, I, T)$ consisting of

- state (boolean) variables $\bar{x} = \{x_1, x_2, ..., x_n\}$,
- a propositional formula $I(\bar{x})$ describing initial states,
- a propositional formula $T(\bar{x}, \bar{x}')$ describing transition relation

Where $\bar{x}'$ are the next-state versions of state variables: $\{x_1', x_2', ..., x_n'\}$.

A predicate over $\bar{x}$ represents a set of states. Notably, a conjunction of n (different) literals represents a single <u>state</u>.

BTS Example



$$
\begin{aligned}
\bar{x} &= \{x_1, x_2, x_3\} \\
I(\bar{x}) &= \neg x_2 \wedge \neg x_3 \\
T(\bar{x}, \bar{x}') &= (x_1 \iff x_2') \wedge (x_2 \iff x_3') \wedge (x_3 \iff x_1') \\
P(\bar{x}) &= \neg x_1 \vee \neg x_2
\end{aligned}
$$

Preliminaries – cont.

From now on, we fix some BTS $(\bar{x}, I, T)$ and a set of states P called property.

Notation: we do not write arguments to named formulas. That is, we abbreviate $I(\bar{x})$ to I and $T(\bar{x}, \bar{x}')$ to T. Formula F' is the formula F with all variables primed. E.g., if $F = x \wedge y$ then $F' = x' \wedge y'$. If $F = z \wedge y'$, then $F' = z' \wedge y''$, etc.

Preliminaries – cont.

From now on, we fix some BTS $(\bar{x}, I, T)$ and a set of states P called <u>property</u>.

Notation: we do not write arguments to named formulas. That is, we abbreviate $I(\bar{x})$ to I and $T(\bar{x}, \bar{x}')$ to T. Formula F' is the formula F with all variables primed. E.g., if $F = x \wedge y$ then $F' = x' \wedge y'$. If $F = z \wedge y'$, then $F' = z' \wedge y''$, etc.

A set of states is <u>invariant</u> of BTS if it is a superset of reachable states. A set S of states is <u>inductive</u> (closed under reachability) if

$$\begin{aligned} I &\implies S \\ S \wedge T &\implies S' \end{aligned}$$

Model Checking of BTSs

Assume we want to show that all states reachable from I satisfy a predicate P.

The basic option is to enumerate states reachable in 0, 1, 2, ... steps.

Model Checking of BTSs

Assume we want to show that all states reachable from I satisfy a predicate P.

The basic option is to enumerate states reachable in 0, 1, 2, ... steps. But that is not efficient for large systems.

Model Checking of BTSs

Assume we want to show that all states reachable from I satisfy a predicate P.

The basic option is to enumerate states reachable in 0, 1, 2, ... steps. But that is not efficient for large systems.

Another option is to try to find an inductive invariant F such that $F \implies P$, i.e.,

- $I \implies F$
- $F \wedge T \implies F'$
- $F \implies P$

Model Checking of BTSs – cont.

A useful technique for finding inductive invariants for BTSs is IC3/PDR:

- IC3/PDR maintains an over-approximation F of states reachable in at most k steps (k is being increased if found insufficient)
- it iteratively blocks states that fail the inductiveness of F until F becomes inductive or a counter-example (real error) is found.
- IC3/PDR generalizes the blocked states to speed-up the convergence to an invariant.
- Problem 1: the generalization has a big over-head.
- Problem 2: it may take a long time to find a counter-example.

Model Checking of BTSs – cont.

CAR is inspired by IC3/PDR but tries to solve:

- Problem 1 (the generalization in IC3/PDR has a big over-head) by using MUC for the generalization. MUC are provided virtually free by SAT solvers.
- Problem 2 (it may take a long time to find a counter-example) by using also an under-approximation of states that reach bad states.

CAR Algorithm (main loop)

```
assume  B_i, F_i = false, true for all i > 0
F_0, B_0 ← I, ¬P

# check the first two steps from init
if sat(F_0 ∧ B_0) or sat(F_0 ∧ T ∧ B_0'):
    return unsafe (+ cex)

for i in  1, 2, ... :
    F_i ← P
    cex ← strengthen
    # a counter-example (real error) found
    if cex: return unsafe (+ cex)

    # some frame got inductive
    if ∃j ≤ i : F_j ⟹ ⋁_{m<j} F_m:
        return safe
```

CAR Algorithm (the core)

```
def strengthen():

    while sat((⋁_n F_n) ∧ T ∧ (⋁_n B'_n)):
        j ← the minimal j s.t. sat(F_j ∧ T ∧ B'_k) for some k
        c_1 ← partial_assignment(F_j ∧ T ∧ B'_k)|_x̄
        if j = 0: return c_1 # found error

        B_{k+1} ← B_{k+1} ∨ c_1
        φ ← F_{j-1} ∧ T ∧ c'_1
        if sat(φ):
            c_2 ← partial_assignment(φ)|_x̄
            B_{k+2} ← B_{k+2} ∨ c_2
        else:
            c_2 ← (unsat_core(φ)|_{c'_1})[x̄' ← x̄]
            F_j ← F_j ∧ ¬c_2
```

Final notes

- CAR can run also backwards (with T^{-1} from $\neg - P$ to I)
- Experiments: in paper.

Final notes

- CAR can run also backwards (with T^{-1} from $\neg - P$ to I)
- Experiments: in paper.

Thank you!

MUNI
FACULTY
OF INFORMATICS