Abstraction for Model Checking

William Schultz

August 30, 2022

Abstraction, in the context of model checking, is generally aimed at reducing the size of the state space in an attempt to remove details that are irrelevant to the property being verified [2]. That is, broadly, abstraction is a fundamental tool in tackling the "state explosion" problem.

Abstraction for Kripke Structures

In general, an abstraction framework defines a set of concrete objects and abstract objects and a definition of how to map between them. For model checking, we typically use Kripke structures as our concrete objects. Recall that a *Kripe structure* $M = (AP, S, I, R, L)$ is defined as

- a set AP of atomic propositions
- a set of states S
- a set of initial states $I \subseteq S$
- a transition relation $R \subseteq S \times S$
- a labeling function $L : S \to 2^{AP}$

Simulation

To define a notion of abstraction for Kripke structures, we define a few standard relations between two structures M_1 and M_2. *Simulation* is a preorder (a reflexive and transitive partial order) in which the larger structure may have more behaviors, but possibly fewer states and transitions.

Let $M_1 = (AP_1, S_1, I_1, R_1, L_1)$ and $M_2 = (AP_2, S_2, I_2, R_2, L_2)$ be Kripke structures such that $AP_2 \subseteq AP_1$. A relation H is a *simulation relation from* M_1 *to* M_2 if for every $s_1 \in S_1$ and $s_2 \in S_2$ such that $H(s_1, s_2)$, both of the following conditions hold:

- For all $p \in AP_2$, $p \in L_1(s_1) \iff p \in L_2(s_2)$
- $\forall t_1 : R_1(s_1, t_1) \Rightarrow \exists t_2 : (R_2(s_2, t_2) \wedge H(t_1, t_2))$

The second condition states that for every transition of the "smaller" (i.e. more concrete) system M_1, there must exist a corresponding transition in the larger system M_2. We say that M_1 *is simulated by* M_2 (or M_2 *simulates* M_1) (denoted $M_1 \leq M_2$) if there exists a simulation relation H from M_1 to M_2 such that

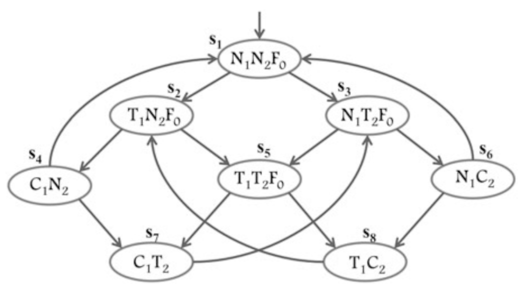
$$\forall s_1 \in I_1 : (\exists s_2 \in I_2 : H(s_1, s_2))$$

For example, consider the concrete Kripke structure M, modeling a mutual exclusion program where its atomic propositions $AP = \{N_1, T_1, C_1, N_2, T_2, C_2, F_0\}$:

Fig. 1 Process P_i

$v_i = Neutral$	$\to$	$v_i := Trying$
$v_i = Trying \ \wedge \ Flag = \mathrm{tt}$	$\to$	$v_i := Critical;\ Flag := \mathrm{ff}$
$v_i = Critical$	$\to$	$v_i := Neutral;\ Flag := \mathrm{tt}$

Fig. 2 Kripke structure *M* for the mutual exclusion program



and then an abstraction of this structure, M_1, with atomic propositions $AP_1 = \{C_1, C_2, F_0\}$:

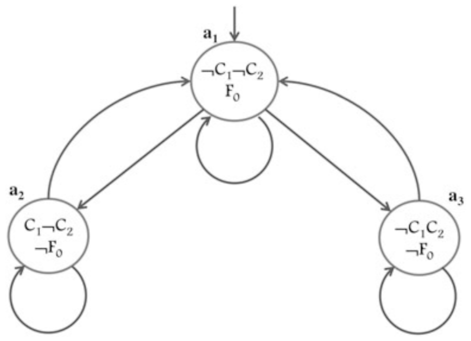


Fig. 3 Abstract Kripke structure M_1 for the mutual exclusion program

This abstraction basically only tracks whether a particular process is in the critical section or not, but ignores all other information. Note that $AP_1 \subseteq AP$. A simulation relation $H \subseteq S \times S_1$ from M to M_1 can then be defined as

$$H = \{(s_1, a_1), (s_2, a_1), (s_3, a_1), (s_4, a_2), (s_5, a_1), (s_6, a_3), (s_7, a_2), (s_8, a_3)\}$$

Bisimulation

One state is related to another by the bisimulation relation of they agree on their common atomic propositions and, in addition, for every successor of one state there is a corresponding successor of the other state, and *vice versa*.

Existential Abstraction

One way to define an abstract model (Kripke structure) from a concrete one is via a concretization function γ. We can define abstract Kripke structures by means of *existential abstraction* [1]. Given a set $\widehat{S}$ of abstract states, the *concretization function* $\gamma : \widehat{S} \to 2^S$ indicates, for each abstract state $\widehat{s}$, what set of concrete states are represented by $\widehat{s}$. Similarly, there is a transition from abstract state $\widehat{s}$ to another abstract state $\widehat{s}'$ if there is a transition from a state represented by $\widehat{s}$ to a state represented by $\widehat{s}'$. Essentially, we just take every transition between concrete states and add it into our abstract transition system, based on the abstract states that represent those concrete state transitions.

Counterexample-Guided Abstraction Refinement (CEGAR)

Regardless of how we choose our abstraction, our abstract model $\widehat{M}$ generally contains less information than the concrete model M, and so model checking $\widehat{M}$ may produce incorrect results. If a universal property is true in $\widehat{M}$ then it is also true in M, but if the abstract model produces an error, the concrete model may still be correct.

For example consider the following "traffic light" model and a simple abstraction of it

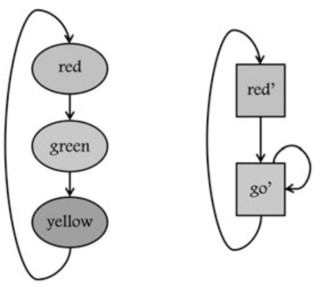


Fig. 6 Abstraction of a traffic light

If we wanted to check the universal CTL property **AGAF**($IsRed$) (i.e. along all paths, $IsRed$ holds infinitely often), this clearly holds in the concrete traffic light model, but fails in the abstract model. When an abstract counterexample does not correspond to any concrete counterexample, we call it *spurious*.

Consider another example of a spurious counterexample, shown as follows:

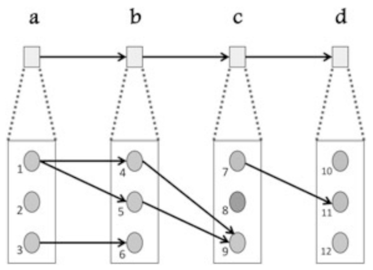


Fig. 7 Spurious counterexample. The abstract state *c* is a failure state

in this case, reachability of state 11 is not preserved by the abstraction. That is, it is not reachable in the lower level system, but it is reachable in the abstract system (consisting of abstract states $\{a, b, c, d\}$).

The framework of *counterexample-guided abstraction refinement (CEGAR)* deals with this issue. The main steps of CEGAR are as follows:

1. Given a concrete model M and some universal temporal formula to check, ψ, generate an initial abstract model $\widehat{M}$.
2. Model check $\widehat{M}$ with respect to ψ. If $\widehat{M}$ satisfies ψ, then conclude that the concrete model satisfies ψ and terminate. If a counterexample $\widehat{T}$ is found, check whether it is also a counterexample in the concrete model.
   - If it is, conclude that the concrete model does not satisfy the formula and stop.
   - Otherwise, the counterexample is spurious, and proceed to step 3.
3. Refine the abstract model, $\widehat{M}$, so that $\widehat{T}$ will not be included in the new, refined abstract model. Go back to step 2.

Note that refinement is typically done by *partitioning* an abstract state. That is, the set of concrete states represented by the abstract state is partitioned

Identifying Spurious Counterexamples

If we discover an abstract counterexample $\widehat{T}$, we need some way to check if this is a real counterexample in the concrete model. Assume that $\widehat{T}$ is a path $\widehat{s_1}, \ldots, \widehat{s_n}$ starting at the initial abstract state $\widehat{s_1}$. We can extend the concretization function γ to sequences of abstract states as follows: $\gamma(\widehat{T})$ is the set of concrete paths defined as:

$$\gamma(\widehat{T}) = \left\{ \langle s_1, \ldots, s_n \rangle \mid \bigwedge_{i=1}^{n} s_i \in \gamma(\widehat{s_i}) \wedge I(s_1) \wedge \bigwedge_{i=1}^{n} R(s_i, s_{i+1}) \right\}$$

Then, we need an algorithm to compute a sequence of sets of states that correspond to $\gamma\widehat{T}$. We let $S_1 = \gamma(\widehat{s_1}) \cap I$, and then define

$$S_i := \mathit{Image}(S_{i-1}, R) \cap \gamma(\widehat{s_i})$$

where $\mathit{Image}(S_{i-1}, R)$ is the set of successors, in M, of the states in S_{i-1}. Basically, we just want to symbolically execute concrete model, starting from the concretized version of the initial abstract counterexample state, and, at each step, check whether there is some concrete state in this image set that corresponds to the set of states from the abstract counterexample. We can formalize this into the following lemma. Specifically, the following are equivalent:

1. The path $\widehat{T}$ corresponds to a concrete counterexample.
2. The set $\gamma(\widehat{T})$ of concrete paths is non-empty.
3. For all $1 \leq i \leq n, S_i \neq \emptyset$.

Note that checking whether a counterexample is spurious involves computations on the concrete model.

SAT-based Abstraction

Main idea is to do bounded model checking and then use proofs of unsatisfiability in this case to provide an *explanation* of correctness, and to help us generate an abstraction for proving a property in the unbounded case. Using such a proof to generate an abstraction is called *proof-based abstraction*[3].

References

[1] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, sep 1994.

[2] Dennis Dams and Orna Grumberg. *Abstraction and Abstraction Refinement*, pages 385–419. Springer International Publishing, Cham, 2018.

[3] Kenneth L. McMillan and Nina Amla. Automatic abstraction without counterexamples. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'03, page 2–17, Berlin, Heidelberg, 2003. Springer-Verlag.