




Ternary Simulation as Abstract Interpretation (Work in Progress)

Nils Froleyks  Emily Yu  Armin Biere 

Abstract

We introduce a formalization of ternary simulation as abstract interpretation along with a widening operator to speed up convergence. With the same goal, we present a subsumption algorithm that can determine termination earlier than the usual approach using hash sets. Additionally, we introduce a narrowing operator that utilizes recent advances in backbone extraction, allowing to increase the overapproximation precision in simulation at any time. The experiments evaluate the presented techniques in the context of hardware model checking.

1 Introduction

Optimizing reachability analysis and model checking is an important topic in formal verification of hardware designs. Ternary simulation [1] is a powerful method utilized in various preprocessing techniques as well as tightly integrated in standard model checking approaches like PDR [2]. For example, phase abstraction [1] leverages ternary simulation for identifying a group of oscillating signals used to simplify the original hardware designs.

By using a three-valued logic $\{1, 0, X\}$, at initialization the inputs are assigned unknown values (X) while latch variables are assigned according to their reset definitions; the successor states are then computed based on the three-valued semantics. Differently from using conventional symbolic execution, as a result an over-approximation of reachable states is obtained at termination. This also allowed us in recent work on model checking certificates to provide an alternative notion of cube semantics [3].

Even though ternary simulation enables fast computation in practice even on industrial designs [4], the key challenge, is the exponential time for convergence in the worst case, due to the PSPACE complete nature of symbolic reachability analysis, also known as the “state explosion” problem. Furthermore, while ternary simulation has been implemented in numerous state-of-the-art tools, there are still significant gaps between its precise theory and practical applications.

On another matter, abstract interpretation [5] helps to obtain sound and precise overapproximations of the state space, and has been commonly used in the static analysis of software systems [6]. It relies on a logical approximation relation between concrete models and abstract predicates to produce a sound fixpoint approximation.

Related to ternary simulation is Symbolic Trajectory Evaluation (STE) [7], which can be characterized as a combination of symbolic execution and ternary simulation, but with the ternary value functions encoded as BDDs.

Previous work [8] has shown, from a theoretical point of view, the Galois connection between a ternary model and Boolean model as a form of abstract interpretation. We focus on practical applications of abstract interpretation and further utilize narrowing and widening operators to refine lossy unknown values and ensure early termination.

In this paper, we make an attempt to bridge the gap by formalizing ternary simulation as a form of abstract interpretation, which is more succinct than the formalism presented in [8], and at the same time closer to practical implementations of simulation.

This enables us to leverage the framework of abstract interpretation with decades of extensive research to enhance bit-level hardware verification. We begin by defining an abstract domain and transformation functions for ternary simulation. Furthermore, we introduce a widening operator in the strict sense of abstract interpretation, guaranteeing early termination to avoid exponential computation. We also present a weaker widening operator, which might be more suitable for practical applications and an algorithm for efficiently determining convergence. More importantly, we further enhance the technique by making use of backbone extraction [9] in defining the narrowing operator. Lastly, we also demonstrate the effectiveness of our method in the experimental evaluation.

2 Circuit

In the rest of the paper, we assume standard semantics for Boolean operators [10] and use the notations from abstract interpretation theory [11].

Since ternary simulation relies on the structure of a Boolean circuit, our definition is more detailed than a simple transition relation and defines transition functions in the form of an and-inverter graph (AIG).

Definition 1 (Circuit). A Boolean circuit C is represented by the tuple (I, L, A, R, F, D) where $I = \{i_1, \dots, i_{\#I}\}$, then $L = \{l_{\#I+1}, \dots, l_{\#I+\#L}\}$ and $A = \{a_{\#I+\#L+1}, \dots, a_{\#I+\#L+\#A}\}$ are *inputs*, *latches* and (AND) *gates* respectively and are ordered continuously. Let $\bar{S} = S \cup \{\neg \ell \mid \ell \in S\}$ for any such set S denote the set of literals over S and VAR the inverse operation (determining variables from literals). With that, the set of *all* literals is $\Lambda = \bar{I} \cup \bar{L} \cup \bar{A}$.

The (total) functions $R : L \rightarrow \{0, 1\}$ and $F : L \rightarrow \Lambda$ define the initial state of the circuit and the transition behavior of the latches respectively. The (total) function $D : A \rightarrow \Lambda \times \Lambda$ gives the definition of each AND gate and has to be stratified, i.e.: $\forall a_i \in A : D(a_i) = (a_j, a_k) \Rightarrow j < i \wedge k < i$.

Definition 2 (Cubes and States). For a set of literals S we further define the following sets:

- $c \in \mathbb{P}(S)$ is called a *cube* with $\mathbb{P}(S)$ the power set of S ,
- $\mathbb{P}_1(S) = \{c \in \mathbb{P}(S) \mid \ell \in c \Rightarrow \neg\ell \notin c\}$ is the set of *consistent* cubes, and
- $\mathbb{P}_2(S) = \{s \in \mathbb{P}_1(S) \mid |s| = |S|\}$ are the *complete* cubes.

Additionally, we define $v : \mathbb{P}_1(\Lambda) \times \Lambda \rightarrow \{0, 1, X\}$ to get the value of a literal in a consistent cube and further use $\tau : \Lambda \times \{0, 1, X\} \rightarrow \Lambda \cup \{\varepsilon\}$ to change the sign of a literal:

$$v(c, \ell) = \begin{cases} 1, & \text{if } \ell \in c \\ 0, & \text{if } \neg\ell \in c \\ X, & \text{otherwise} \end{cases} \quad \tau(\ell, b) = \begin{cases} \ell, & \text{if } b = 1 \\ \neg\ell, & \text{if } b = 0 \\ \varepsilon, & \text{otherwise} \end{cases}$$

These are the only functions that explicitly deal with three-valued semantics $\{0, 1, X\}$. Otherwise our formalism uses cubes instead of full assignments mapping to $\{0, 1, X\}$. To further simplify the exposition we use ε as the neutral element of set-addition, i.e., any set containing ε is equivalent to the same set without it.

The stratification condition in Def. 1 ensure that the directed graph induced by D is a acyclic, i.e., a *directed acyclic graph* (DAG), with inputs and latches as leaves. This makes the transition function of a circuit well-defined:

Definition 3 (Transition). For a circuit $C = (I, L, A, R, F, D)$ we define the following functions:

- $ext_I : \mathbb{P}_2(L) \rightarrow \mathbb{P}(\mathbb{P}_2(I \cup L))$, $ext_I(s) = \{s' \in \mathbb{P}_2(I \cup L) \mid s' \Rightarrow s\}$ that expands a state over latches with all possible inputs.
- $ext_A : \mathbb{P}_2(I \cup L) \rightarrow \mathbb{P}_2(\Lambda)$ is defined as fixpoint of $\varphi(s) = \{\tau(a, \text{AND}(v(s, l), v(s, r))) \mid (a, (l, r)) \in D\}$ It can be computed in a single pass over the gates due to the stratification assumption. Since s is a complete state and therefore $v(s, \cdot)$ maps to $\{0, 1\}$, AND computes the standard Boolean function (“ \wedge ”).
- $next_L(s) = \{\tau(\ell, v(s, n)) \mid (\ell, n) \in F\}$ extracts the successor state from next-state function of a latch.
- $f^L : \mathbb{P}_2(L) \rightarrow \mathbb{P}(\mathbb{P}_2(L))$, with $f^L(s) = \{next_L(ext_A(s')) \mid s' \in ext_I(s)\}$, maps a state to the set of its successors in C .

3 Ternary Simulation as Abstract Interpretation

In this section we define the abstract transformation from a concrete circuit to a ternary circuit. We will consider two lattices (Σ, \subseteq) , with $\Sigma = \mathbb{P}(\mathbb{P}_2(L))$ for the concrete semantics of the circuit and (Ω, \Rightarrow) , with $\Omega = \mathbb{P}(L)$ for abstract semantics utilizing ternary simulation.

Note, that we prefer to use the set of cubes Ω instead of three-valued states: $\{0, 1, X\}^{|L|} \cup \perp$. Here we use \perp to denote “no-state” (inconsistent cube in $\mathbb{P}(L)$), not to be confused with the ternary state (X, X, \dots) representing all states equivalent to the empty cube in our abstract domain.

Theorem 4. (Σ, \subseteq) and (Ω, \Rightarrow) are complete lattices.

We further define two functions to translate between them:

Definition 5. The *abstraction function* $\alpha : \Sigma \rightarrow \Omega$ is defined as $\alpha(\sigma) = \bigcap_{s \in \sigma} s$.

Definition 6. The *concretization function* $\gamma : \Omega \rightarrow \Sigma$ is defined as $\gamma(\omega) = \{s \mid s \in \mathbb{P}_2(L) \wedge s \Rightarrow \omega\}$.

Theorem 7. The tuple $(\gamma, \Sigma, \Omega, \alpha)$ is a Galois connection.

Proof. For sets of states $\sigma \in \Sigma$ and cube $\omega \in \Omega$, we have:

$$\begin{aligned} & \alpha(\sigma) \Rightarrow \omega && \text{by Def. 5} \\ \Leftrightarrow & \left(\bigcap_{s \in \sigma} s \right) \Rightarrow \omega \\ \Leftrightarrow & \forall s \in \sigma : s \Rightarrow \omega && \text{with } \sigma \subseteq \mathbb{P}_2(L) \\ \Leftrightarrow & \sigma \subseteq \{s \mid s \in \mathbb{P}_2(L) \wedge s \Rightarrow \omega\} && \text{by Def. 6} \\ \Leftrightarrow & \sigma \subseteq \gamma(\omega) && \square \end{aligned}$$

For the concrete transition function, we define the transition of a set of states as the union of their successors.

Definition 8. The *concrete transition function* $f : \Sigma \rightarrow \Sigma$ is defined as $f(\sigma) = \bigcup_{s \in \sigma} f^L(s)$.

On the abstract side we finally formally define ternary simulation as the abstract transition function that allows us to transition in the abstract domain.

Definition 9. We define $f^\# : \Omega \rightarrow \Omega$ as the *abstract transition function* with $f^\#(\omega) = next_L(ext_A^X(\omega))$, where $ext_A^X : \mathbb{P}_1(I \cup L) \rightarrow \mathbb{P}_1(\Lambda)$ follows the definition of ext_A except AND is replaced by AND^X defined by the table below:

l	r	$\text{AND}^X(l, r)$
0	-	0
-	0	0
1	1	1
1	X	X
X	1	X

where ‘-’ denotes either 0 or 1.

We have to show that $f^\#$ is indeed a valid abstraction of f .

Theorem 10. $(f \circ \gamma)(\omega) \subseteq (\gamma \circ f^\#)(\omega)$

For that we will first define the depth of a gate.

Definition 11 (Depth). The *depth* of a gate $a \in A$ is defined as the length of the longest path from a to a leave in the DAG induced by D . The depth of an input I or latch L is 0.

We state a connection between the transition functions.

Lemma 12. For $\omega \in \mathbb{P}_1(L)$, $s \in \gamma(\omega)$, $a \in I \cup L \cup A$, if $v(ext_A^X(\omega), a) \in \{0, 1\}$ then $v(ext_A(s), a) = v(ext_A^X(\omega), a)$.

Proof. The proof proceeds by induction over the depth n of a . For $n = 0$, a is either an input or a latch literal, where only the latch might be in ω in which case it will also be in both extensions. Now for any gate a at depth $n + 1$ let $(l, r) = D(a)$, both l and r have depth no greater than n . Further, in case $c = X$ the claim holds trivially. Otherwise, neither l nor r are X , thus AND equals AND^X or exactly one of them is X and the other one is 0. In the latter case, both extensions in $\gamma(\omega)$ result in 0. \square

We continue to the proof of the Theorem 10.

Proof of Theorem 10. Suppose there is a $\sigma \in (f \circ \gamma)(\omega)$ that differs from all states in $(\gamma \circ f^\#)(\omega)$ in at least one literal. Let ω be as stated in the theorem, $\sigma = \gamma(\omega)$ and σ' a state in $(\gamma \circ f^\#)(\omega)$ that differs from σ in the fewest number of literals, one of them being ℓ . We consider two cases: (1) $v(f^\#(\omega), \ell) = X$: By definition of γ the set $\gamma(f^\#(\omega))$ also contains a state that is the same as σ' but matches $\sigma \in \ell$, contradicting our assumption. (2) $v(f^\#(\omega), \ell) = c$, with $c \in \{0, 1\}$: Let $a = F(\ell)$, then $v(\text{ext}_A^X, a) = c$. By Lemma 12 and Def 8 σ matches σ' on ℓ again contradicting our assumption. \square

4 Widening

Even though ternary simulation can be implemented very efficiently, it is exponential in the size of the circuit. In fact, this exponential behavior is easily exposed by adding a 64-bit counter that is independent of the rest of the design.

The widening operators, as introduced in [12], promise to alleviate that problem by guaranteeing a faster convergence. However the demands on the properties of such a widening operator are quite strong. We will introduce ∇ that fulfills both the *covering* and *termination* property [13] and additionally the more conservative operator $\bar{\nabla}$ that does not meet these strict criteria, but exhibit superior performance in our application. A similar operation has been introduced as *X-saturation* in [4]. We do believe that $\bar{\nabla}$ is sufficient to guarantee termination in linear time, if the operator is applied periodically. However, the proof of this claim remains open.

Definition 13. $\nabla : \mathbb{P}(L) \times \mathbb{P}(L) \rightarrow \mathbb{P}(L)$, $a \nabla b = a \cap b$.

Theorem 14. ∇ is:

1. *covering*: $\forall a, b \in \mathbb{P}(L) : a \Rightarrow a \nabla b$, and $b \Rightarrow a \nabla b$
2. *terminating*: For an ascending chain $\{a_i\}_{i \geq 0}$, the chain $b_0 = a_0, b_{i+1} = b_i \nabla a_{i+1}$ stabilizes after a finite number of terms.

Definition 15. $\bar{\nabla} : \mathbb{P}(L) \times \mathbb{P}(L) \rightarrow \mathbb{P}(L)$ with $a \bar{\nabla} b = b \setminus \{\ell\}$ and where neither $\ell \in b$, $\neg \ell \in a$ nor $\text{VAR}(\ell)$ have been removed by widening before.

5 Narrowing

Ternary simulation can produce a high number of spurious traces, which is even more true if widening is used. Narrowing operators [13] increase the precision of the simulation at any point, thus removing a set of spurious traces while still maintaining a valid over-approximation.

Our narrowing operator for ternary simulation Δ relies on the backbone of the transition between two cubes. The backbone of a satisfiable formula, is the set of literals that hold true in all assignments. It is only applicable to two cubes a, b if $(f^L)^n(a) \Rightarrow b$, were n is the number of function applications. For simplicity we will only define it for a single step of the transition function.

Definition 16. $\Delta : \mathbb{P}(L) \times \mathbb{P}(L) \rightarrow \mathbb{P}(L)$ with $a \Delta b = F^{-1}(B(a \wedge D \wedge F(b)))$, and where

- $F(b) = \{\tau(F(\ell), v(b, \ell)) \mid \ell \in \text{VAR}(b)\}$ denotes the “primed” version of a cube b ,
- $F^{-1}(c) = \{\tau(\ell, v(c, n)) \mid (\ell, n) \in F\}$ the inverse and
- $B(\Phi)$ denotes the backbone of a Boolean formula Φ .

6 Termination via Subsumption

In both, collection semantics of abstract interpretation [11] and ternary simulation [4], termination is defined as a subsumption check, i.e., the simulation terminates if a cube implies a previously encountered cube. At that point the encountered cubes represent an overapproximation of all reachable states. Such a check can be implemented using BDDs, however as the authors of [4] state: “In practice, the performance of such approach is prohibitive”. They instead use a hash table and only terminate, when an exact match to a previously encountered cube is found.

We utilize a different algorithm used for *forward subsumption* detection in SAT solving [14, 10]. The algorithm relies on a one-watch data structure, i.e., each cube appears in the watch-list of one of its literals.

```

subsumed (cube c)
1  mark all literals in c
2  for literal  $\ell$  in c do
3      for cube  $c'$  in watch[ $\ell$ ] do
4           $\ell' \leftarrow$  unmarked literal in  $c'$ 
5          if  $\ell' =$  invalid then // No such literal
6              lassos.add(pred(c),  $c'$ )
7          if  $|c'| = |c|$  then // Exact match
8              return lassos
9          else watch[ $\ell'$ ].add( $c'$ )
10     watch[ $\ell$ ].clear()
11  unmark all literals

```

Algorithm 1 Subsumption check. Identifies all previous cubes that imply c , and thereby induce a *lasso* in the state space. Requires one literal of each cube to be *watched*.

Whenever the algorithm reaches line 6, a sound over-approximation is found. However, for some applications it can be beneficial to consider more than one *cube lasso*. For example in phase abstraction [1] the length of both the stem and the loop of the lasso should be divisible by some small number.

7 Evaluation

Our preliminary implementation does not cover the entirety of Sect. 4 and 5. The simulation itself is fairly efficient, calculating all the gates in a single linear pass, using a few basic bit operations. However, we do not reorder gates to allow for more efficient packing / random access of state bits or parallelization using SIMD / threads.

	# _{Latches}	base	widening	termination
bob12m04	43950	199 0.06	199 0.07	153 0.05
bob12m15	448	133 1.62	133 1.60	12 0.40
bobsmnut1	644	107 0.21	107 0.19	106 0.18
shift1add	27	20 1.20	20 1.25	1 0.01
6s376r	4708	145 766.34	116 16.30	116 4.20
6s47	815	to	8 35.63	6 0.30
6s100	97598	to	36 37.53	36 41.48
6s107	1568	to	746 16.93	746 1.18
6s149	12781	to	4 47.81	4 12.88
6s202b41	68881	to	8574 45.71	8574 28.35
6s204b16	28986	to	4034 19.33	4034 13.51
6s205b20	68842	to	8727 46.56	8727 28.08
6s355rb ₈₇₄₀	15091	to	221 37.07	221 15.51
6s400rb ₇₈₁₉	14665	to	221 36.96	221 11.00
6s342rb ₁₂₂	56838	to	to	1894 354.60
cucnt128	128	to	to	0 1.82
cucnt32	32	to	to	0 0.01

Table 1 We evaluated three versions: **base** ternary simulation with hashing and no widening, a configuration using **widening** ($\bar{\nabla}$) if the cube has not reduced in size in a few thousand iterations, and one that employs *both* widening and the early **termination** using the forward subsumption algorithm. Presented is the runtime and the number of *transients* that could be found for each circuit. Our benchmarkset included all 20815 circuits from the HWMCC (2007-2020)[15]. The table lists all instances where either transients were lost to the optimization or any of the configurations timed out (^{to}).

Termination is implemented both with hashing and forward-subsumption for comparison. We also provide an implementation of the widening operator $\bar{\nabla}$. Ours differs from the *X-saturation* described in [4] in that we do not eliminate all non-fixed latches, but only pick a single one, which has not been affected by widening before. As $\bar{\nabla}$ removes too many literals, it is not evaluated. We do not yet have an implementation for narrowing.

All experiments were conducted on our cluster with Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz, with a time limit of 2 hours. As an example application of ternary simulation and to gauge its precision, we extract *transients*. Transients are latches that assume a constant value after a finite number of steps (constant in the loop of any cube lasso). The results are shown in Table 1.

Considering the high number of benchmarks, both widening and earlier termination had very little impact on the number of identified transient. However, they did help with a number of related benchmarks that originally exceeded the two-hour time limit. Note that the final configuration using both techniques solved all instances.

Acknowledgement

This work is supported by the Austrian Science Fund (FWF) under the project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, the ERC-2020-AdG 101020093 and by a gift from Intel Corporation.

8 Literature

- [1] P. Bjesse and J. H. Kukula, “Automatic generalized phase abstraction for formal verification,” in *ICCAD*. IEEE Computer Society, 2005, pp. 1076–1082.
- [2] N. Eén, A. Mishchenko, and R. K. Brayton, “Efficient implementation of property directed reachability,” in *FMCAD*. FMCAD Inc., 2011, pp. 125–134.
- [3] E. Yu, N. Froylyks, A. Biere, and K. Heljanko, “Towards compositional hardware model checking certification,” in *FMCAD*, 2023, pp. 44–54.
- [4] M. L. Case, J. Baumgartner, H. Mony, and R. Kanzelman, “Approximate reachability with combined symbolic and ternary simulation,” in *FMCAD*, P. Bjesse and A. Slobodová, Eds. FMCAD Inc., 2011, pp. 109–115.
- [5] P. Cousot, *Méthodes Itératives de Construction et d’approximation de Points Fixes d’opérateurs Monotones Sur Un Treillis, Analyse Sémantique Des Programmes*, 1978.
- [6] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, “The astree analyzer,” in *ESOP*, ser. Lecture Notes in Computer Science, vol. 3444. Springer, 2005, pp. 21–30.
- [7] C. H. Seger and R. E. Bryant, “Formal verification by symbolic evaluation of partially-ordered trajectories,” *Formal Methods Syst. Des.*, vol. 6, no. 2, pp. 147–189, 1995.
- [8] C.-T. Chou, “The mathematical foundation of symbolic trajectory evaluation,” in *CAV’99*. Springer, 1999, pp. 196–207.
- [9] A. J. Parkes, “Clustering at the phase transition,” in *AAAI/IAAI*. AAAI Press / The MIT Press, 1997, pp. 340–345.
- [10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability: Second Edition*, ser. Frontiers in Artificial Intelligence and Applications. Washington: IOS Press, 2021, no. vol. 336.
- [11] P. Cousot and R. Cousot, “Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints,” in *POPL*. ACM, 1977, pp. 238–252.
- [12] P. Cousot and R. Cousot, “Basic concepts of abstract interpretation,” in *IFIP*, ser. IFIP, R. Jacquart, Ed., vol. 156. Kluwer/Springer, 2004, pp. 359–366.
- [13] A. Cortesi and M. Zanioli, “Widening and narrowing operators for abstract interpretation,” vol. 37, no. 1, pp. 24–42, 2011.
- [14] L. Zhang, “On Subsumption Removal and On-the-Fly CNF Simplification,” in *Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds. Berlin, Heidelberg: Springer, 2005, pp. 482–489.
- [15] M. Preiner, A. Biere, and N. Froylyks, “Hardware model checking competition 2020,” 2020.