Formal Methods in Computer-Aided Design 2022

Split Transition Power Abstraction for Unbounded Safety

Martin Blicha*[‡], Grigory Fedyukovich[†], Antti E. J. Hyvärinen* and Natasha Sharygina*
*Università della Svizzera Italiana, Lugano, Switzerland
{blichm,hyvaeria,sharygin}@usi.ch
[†]Florida State University, Tallahassee, FL, USA
grigory@cs.fsu.edu
[‡]Charles University, Prague, Czech Republic

***Abstract*—Transition Power Abstraction (TPA) is a recent symbolic model checking approach that leverages Craig interpolation to create a sequence of symbolic abstractions for transition paths that double in length with each new element. This doubling abstraction allows the approach to find bugs that require long executions much faster than traditional approaches that unfold transitions one at a time, but its ability to prove system safety is limited. This paper proposes a novel instantiation of the TPA approach capable of proving unbounded safety efficiently while preserving the unique capability to detect deep counterexamples. The idea is to split the transition over-approximations in two complementary parts. One part focuses only on reachability in fixed number of steps, the second part complements it by summarizing all shorter paths. The resulting split abstractions are suitable for discovering safe transition invariants, making the SPLIT-TPA approach much more efficient in proving safety and even improving the counterexample detection. The approach is implemented in the constrained Horn clause solver GOLEM and our experimental comparison against state-of-the-art solvers shows it to be both competitive and complementary.**

I. Introduction

Automated formal verification by means of model checking is popular because of the ability to both 1) find error paths for unsafe systems, and 2) *prove* the absence of error paths for safe systems. Recent techniques based on Satisfiability Modulo Theories (SMT) as well as the continuing improvements of SMT solvers [1, 12, 16, 27, 35] enable scalable applications of model checking to software verification [3]. Specifically, the idea of building a safe inductive invariant incrementally—pioneered by the hardware model checking algorithm IC3/PDR [8, 17]—has been successfully applied in several IC3-inspired approaches [10, 11, 18, 24, 29, 30], thus improving the capabilities of verification tools significantly.

Although this progress is undeniably encouraging, model checking still suffers from scalability issues associated with an exhaustive exploration of a system's states. For many systems, a large set of states need to be observed to eventually detect a counterexample or synthesize an invariant. Multi-phase loops [39] often exhibit such behaviour, in particular. A recently introduced approach based on Transition Power Abstraction (TPA) [5] successfully attacks the first part of the problem. It uses abstraction to summarize the reachability of an exponentially increasing number of steps. Thus TPA can quickly focus on the essential part of the search space and not waste time examining short paths that cannot lead to a counterexample. Interestingly, the abstractions that enable TPA to detect long counterexample paths quickly can also be used to prove safety by discovering safe transition invariants [5]. However, the required condition that the over-approximating relation must be closed under composition with transition relation is rarely satisfied, and the algorithm performs rather poorly on safe systems.

In this paper we leverage the ideas from TPA that enable a fast exploration of large parts of the state space to detect invariants in the system that hold only after a specific (often very large) number of transitions. Our new approach, called SPLIT-TPA, also uses the idea of the transition power abstraction sequence but computes the abstractions in a different way that generates significantly more suitable candidates for transition invariants. In the original TPA sequence n^{th} element over-approximates reachability *up to* 2^n steps of the transition relation. The TPA sequence is used to check reachability by doubling the number of explored states at every iteration of the verification run. At the same time the sequence is expanded and its elements are refined as a direct consequence of information learned in these bounded reachability checks.

The novelty of SPLIT-TPA lies in splitting the over-approximating sequence into two complementary parts: $\text{TPA}^{=}$ and $\text{TPA}^{<}$. Elements of $\text{TPA}^{=}$ summarize paths of a fixed number of steps: n^{th} element covers *exactly* 2^n steps of the transition relation. The elements of $\text{TPA}^{<}$ complement the first sequence: n^{th} element summarizes all paths of length less than 2^n. The abstractions of $\text{TPA}^{=}$ sequence allow SPLIT-TPA to discover a special type of safe transition invariants, which are not possible to obtain in the original TPA algorithm. These invariants are composed of two orthogonal parts: one part summarizes safe transitions up to a specific bound; the second part summarizes unbounded safety, but only from that specific bound onwards. The final invariant is a disjunction of these two orthogonal parts which together cover any number of transitions. This specific structure makes these invariants suitable for proving safety of a large class of problems including some challenging instances that cannot be tackled by other state-of-the-art approaches.

We have implemented SPLIT-TPA in our publicly available CHC solver GOLEM and compared it against the original TPA approach and other state-of-the-art solvers ELDARICA and SPACER. On a set of challenging public benchmarks representing multi-phase loops [39], SPLIT-TPA significantly outperforms TPA on the safe version of these benchmarks and

https://doi.org/

is able to prove safe several benchmarks that state-of-the-art tools are not able to solve. Moreover, SPLIT-TPA outperforms TPA also on the unsafe version of these benchmarks.

The rest of the paper is organized as follows. Section II presents the necessary background. Section III gives a detailed overview of the TPA algorithm from [5]. Our novel instantiation is presented in Section IV. In Section V we show how the transition invariants from SPLIT-TPA can be translated into state invariants. The experiments are described in Section VI. Finally, we discuss the related work in Section VII and conclude in Section VIII.

II. Preliminaries

We assume a finite set of (typed) variables $\vec{x}$, called *state variables*, and we associate with it a primed copy $\vec{x}'$. A formula $S(\vec{x})$ over the state variables is a *state* formula and a formula $T(\vec{x}, \vec{x}')$ is a *transition* formula. A *state* s is an interpretation of $\vec{x}$ that assigns value to each $x \in \vec{x}$. For a formula $S(\vec{x})$ and a state s we say s is an S-state iff $s \models S$. We identify state formulas with sets of states where they hold and freely move between these two representations. Similarly, we identify transition formulas with binary relations over the set of states. The identity relation $Id(x, x')$ corresponds to the transition formula $x = x'$. For readability we typically drop the vector notation and use x, x' instead of $\vec{x}, \vec{x}'$. Additional copies of the state variables are denoted as x'', x''', or in general $x^{(n)}$ for x with n primes added. Given binary relations R_1 and R_2, $R_1 \circ R_2$ represents *relational composition* of R_1 and R_2, $R_1 \cup R_2$ represents their union. For $R = R_1 \circ R_2$, $R(x, z) \equiv \exists y : R_1(x, y) \wedge R_2(y, z)$. Similarly, for $R = R_1 \cup R_2$, $R(x, y) \equiv R_1(x, y) \vee R_2(x, y)$. For a binary relation R and a set A, we denote the restriction of the domain of R to A as $A \lhd R = \{(x, y) \mid (x, y) \in R \text{ and } x \in A\}$ and the restriction of codomain as $R \rhd A = \{(x, y) \mid (x, y) \in R \text{ and } y \in A\}$. In terms of logical formulas, $(A \lhd R)(x, y) \equiv R(x, y) \wedge A(x)$, $(R \rhd A)(x, y) \equiv R(x, y) \wedge A(y)$.

Transition system is a pair $\mathcal{S} = \langle Init, Tr \rangle$ where $Init(\vec{x})$ defines the initial states and $Tr(\vec{x}, \vec{x}')$ is a defines the transition relation of the system. A *safety problem* is a triple $\langle Init, Tr, Bad \rangle$ where $\langle Init, Tr \rangle$ is a transition system and $Bad(\vec{x})$ represents erorr states. Relation Tr^n denotes the composition of n copies of the transition relation and represents reachability in exactly n steps. $Tr^0 = Id$.

A set of states S is a *k-inductive invariant* iff

- $Init(x^{(0)}) \wedge Tr^i(x^{(0)}, x^{(i)}) \implies S(x^{(i)})$ for $0 \leq i < k$,
- $\bigwedge_{i=0}^{k-1} S(x^{(i)}) \wedge Tr(x^{(i)}, x^{(i+1)}) \implies S(x^{(k)})$.

S is an *inductive invariant* if it is 1-inductive.

A binary relation R is a (full) *transition invariant* iff $R \supseteq Tr^*$, where Tr^* is a reflexive transitive closure of Tr. We say that R is a *left-grounded* transition invariant iff $Init \lhd R \supseteq Init \lhd Tr^*$. Similarly, R is a *right-grounded* transition invariant iff $R \rhd Bad \supseteq Tr^* \rhd Bad$. R is a *grounded* transition invariant if it is either left-grounded or right-grounded. Note that a full transition invariant is also both left-grounded and right-grounded. We say R is *safe* iff $\forall x, x' : x \in Init \wedge x' \in Bad \implies (x, x') \notin R$, or in other words, $Init(x) \wedge R(x, x') \wedge Bad(x')$ is unsatisfiable. If a safe grounded transition invariant exists, then Bad is not reachable from $Init$, and the system is safe.

A *Craig interpolant* [15] for an unsatisfiable $A \wedge B$ is a formula I such that (i) $A \implies I$; (ii) $I \wedge B \implies \bot$; (iii) I uses only common symbols of A and B.

III. An overview of TPA

Here we give a brief overview of the TPA algorithm as introduced in [5]. The main procedure is given in Algorithm 1 and resembles the typical main loop of bounded model checking that checks bounded reachability for gradually increasing bound. The main difference is that TPA increases this bound in *exponential* steps (IsReachable($n, Init, Bad$) checks for paths of length $\leq 2^{n+1}$), instead of in one-step increments, as is typical for bounded model checking algorithms. This allows TPA to detect much longer counterexamples compared to state-of-the-art competitors, as witnessed in [5].

Algorithm 1: IsSafeTPA($\langle Init, Tr, Bad \rangle$): TPA's main procedure

```
input : transition system $\mathcal{S} = \langle Init, Tr, Bad \rangle$
global: TPA sequence $ATr^{\leq 0}, \ldots, ATr^{\leq n}, \ldots$ (lazily
        initialized to $true$)
1 $ATr^{\leq 0} \leftarrow Id \vee Tr$;   $n \leftarrow 0$;   $res \leftarrow \emptyset$
2 while $res = \emptyset$ do
3     $res \leftarrow$ IsReachable($n, Init, Bad$)
4     $n \leftarrow n + 1$
5 return UNSAFE
```

The key ingredient that allows efficient bounded reachability checks is the *transition power abstraction* sequence. It is a sequence of relations where nth element *over-approximates* reachability in *up to* 2^n steps of Tr. The construction and refinement of the TPA sequence happen as part of the bounded reachability check, inside the procedure IsReachable, given in Algorithm 2.

Algorithm 2: IsReachable(n, Src, Tgt): Reachability query using TPA sequence

```
input : level $n$, source states $Src$, target states $Tgt$
output: subset of target states truly reachable in $\leq 2^{n+1}$ steps
global: TPA sequence $ATr^{\leq 0}, \ldots, ATr^{\leq n}, \ldots$
1  while $true$ do
2      $q \leftarrow Src(x) \wedge ATr^{\leq n}(x, x') \wedge ATr^{\leq n}(x', x'') \wedge Tgt(x'')$
3      $sat\_res \leftarrow$ CheckSat($q$)
4      if $sat\_res =$ UNSAT then
5          $Itp(x, x'') \leftarrow$ GetItp($ATr^{\leq n}(x, x') \wedge ATr^{\leq n}(x', x'')$,
                   $Src(x) \wedge Tgt(x'')$)
6          $ATr^{\leq n+1} \leftarrow ATr^{\leq n+1} \wedge Itp[x'' \mapsto x']$
7          return $\emptyset$
8      else
9          if $n = 0$ then return QE($\exists x, x'\ q$)$[x'' \mapsto x]$
10         $Inter \leftarrow$ QE($\exists x, x''. q$)$[x' \mapsto x]$
11         $InterReach \leftarrow$ IsReachable($n - 1, Src, Inter$)
12         if $InterReach = \emptyset$ then continue
13         $TgtReach \leftarrow$ IsReachable($n - 1, InterReach, Tgt$)
14         if $TgtReach \neq \emptyset$ then return $TgtReach$
```

This procedure returns a subset of reachable states of Tgt if there exists a path from Src to Tgt of length at most 2^{n+1}. If no such path exists, it returns an empty set. First, it checks existence of an *abstract* path consisting of *two* steps of $ATr^{\leq n}$, the n^{th} element of the TPA sequence (lines 2-3). If no such abstract path exists (line 4), then no real path of length $\leq 2^{n+1}$ exists (line 7). Additionally, $n+1^{\text{st}}$ element of the TPA sequence is constructed or refined using *Craig interpolation* [15] (lines 5-6).

If an abstract path does exist, the procedure attempts to refine it to a real path. The refinement begins by applying quantifier elimination (QE) to determine a set of candidate *intermediate* states (line 10). These are states that can be reached from Src by one step of $ATr^{\leq n}$ and also can reach Tgt by another step of $ATr^{\leq n}$. Given a set of intermediate states, the procedure *recursively* determines the existence of a *real* path from Src to the intermediate states (line 11) and then the existence of a real path from the truly reachable intermediate states to Tgt (line 13). The bound for these recursive calls is decremented, and $n = 0$ represents the base case where no recursive calls are needed as $ATr^{\leq 0}$ represents true reachability in the system (line 9). If any of the two abstract steps cannot be refined, the procedure tries to find a new abstract path and repeats the whole process. The strengthening of $ATr^{\leq n}$ in the recursive call to IsReachable with $n-1$ guarantees that refuted abstract paths cannot repeat, and the procedure makes progress.

Note that instead of full quantifier elimination, any under-approximation can be used in IsReachable. In particular, experiments in [5] showed that TPA works much better with *model-based projection* [4, 30].

One way to understand the procedure IsReachable in TPA is that it mimics bounded reachability checks using a sequence of (precise) relations $R^{\leq n}$ defined inductively as

$$\begin{aligned} R^{\leq 0} &= Id \cup Tr, \\ R^{\leq n+1} &= R^{\leq n} \circ R^{\leq n}. \end{aligned} \tag{1}$$

However, this precise sequence is over-approximated by the TPA sequence. The over-approximation keeps the satisfiability queries manageable: Each $ATr^{\leq n}$ is a formula only over *two* copies of the state variables, no matter how large n is. This is guaranteed by using *Craig interpolation* to compute the abstractions. Compared to that, representing relation $R^{\leq n}$ *precisely* requires $2^n + 1$ copies of the state variables.

The TPA algorithm has been designed to detect long counterexample paths quickly and in this has achieved significant improvements over the state-of-the-art. Interestingly, the TPA sequence can also provide candidates for safe transition invariant, which could be used to prove safety. However, the capabilities of TPA in this respect are very limited, as also exhibited by the experimentation in [5].

Fig. 1 illustrates the limitations of TPA in generating safe transition invariants. The loop on the left has been studied extensively in the context of loop invariants, e.g., in [39]. We scaled the constants to better demonstrate the behaviour of TPA. TPA proves safety up to $8192 = 2^{13}$ iterations

```
x=0; y=5000;
while(x<10000){
    if(x>=5000)
        y=y+1
    x=x+1;
}
assert(y==10000);
```

```
v=0; w=0;
assume(x>z);
while(v<1000){
    if(x<z)
        v=v+1;
    else
        w=w+1;
    x=x+1;
    z=z+2;
}
assert(w>0);
```

Fig. 1. Examples of multi-phase loops

of the loop very quickly. Each of the first 13 top-level calls to IsReachable determines bounded safety with a single satisfiability query. In the process, TPA learns that $ATr^{\leq n} \equiv x' \leq x + 2^n$ for $n = 1 \ldots 13$. It utilizes the fact that x must be incremented more than 2^{13} times to exit the loop and reach the `assert`. However, in the next iteration of Algorithm 1 an abstract path of two steps of $ATr^{\leq 13}$ is discovered and the refinement process in IsReachable begins. To make progress, the algorithm must refine the over-approximating relation $ATr^{\leq 13}$ in order to show that the error is not reachable in two steps of $ATr^{\leq 13}$. This requires learning a suitable relation between variables x and y. However, since $ATr^{\leq 13}$ must capture *all* paths of length $\leq 2^{13}$, it is not easy to learn such a relation. At least in our implementation, TPA is continuously discovering and refuting new abstract paths, making very little progress in refining the elements of the TPA sequence with each refutation. Due to this slow progress, the algorithm fails to prove safety in a reasonable amount of time.

The second loop depicted on the right of Fig. 1 is benchmark 17 from the suite of multi-phase benchmarks used in our experiments (Section VI). The behaviour of TPA is similar to the previous case, but this time it can find a safe invariant, though at a considerable cost, as illustrated below. It uses variable v and the fact that at least 1000 increments are required and quickly proves bounded safety up to 2^9 iterations of the loop. In the next iteration of its main procedure TPA spends a considerable amount of time in IsReachable refining the abstraction and capturing the behaviour of the other variables and the relations between them. Finally, after proving safety up to 2^{11} iterations of the loop, it manages to discover a safe transition invariant.

We will see in the next section that split-TPA is able to prove the first loop safe and it can find a safe transition invariant for the second loop much faster.

IV. Split Transition Power Abstraction

In this section we present split-TPA, a new instantiation of the TPA approach suitable for proving unbounded safety. We start by revisiting $R^{\leq}$ from Eq. (1) and show that the idea of *splitting* the TPA sequence arises naturally from a redundancy present in the inductive definition of $R^{\leq}$. Then we show how split-TPA performs bounded reachability checks with the split sequences and how it discovers safe transition invariants.

A. Overview

As mentioned previously, the TPA algorithm has been designed to be a simple and efficient procedure for detecting deep counterexample paths. It can also prove safety by discovering a safe transition invariant for the system. However, the only source of candidates for the required safe transition invariants are the elements $ATr^{\leq n}$ of the TPA sequence. $ATr^{\leq n}$ can be proved to be a transition invariant if it is closed under composition with one step of Tr. The problem is that this condition is rarely fulfilled. The abstractions $ATr^{\leq n}$ are primarily constructed as proofs of bounded safety in the system: they must summarize all paths of lengths from 0 to 2^n and they must be safe. While it is possible that such bounded proof is in fact an unbounded proof, in many cases these abstractions are not closed under composition with Tr, and the bounded proofs do not generalize to unbounded proofs.

Our solution to TPA's lack of ability to prove unbounded safety in practice is to introduce *new* source of candidates for transition invariants. We split the over-approximating TPA sequence into two complementary parts: TPA$^{=}$ and TPA$^{<}$. Elements of TPA$^{=}$ summarize paths of fixed length and the corresponding elements of TPA$^{<}$ summarize all shorter paths. While TPA$^{<}$ leads to similar transition invariants as TPA, TPA$^{=}$ leads to invariants with different structure and different properties, which allows SPLIT-TPA to prove safety of some challenging problems.

The idea of splitting is motivated not only by the need for another source of candidates for invariants, but also by a possible redundancy in the TPA algorithm, which could lead to unnecessary work. TPA sequence is based on the sequence $R^{\leq}$ from Eq. (1). The intuition behind this inductive definition is that every path of length $\leq 2^{n+1}$ can be obtained as a concatenation of two paths of length $\leq 2^n$. However, there can be multiple ways to decompose such a path into two smaller paths (see Fig. 2) and proving one such decomposition infeasible does not entail that others are infeasible as well.

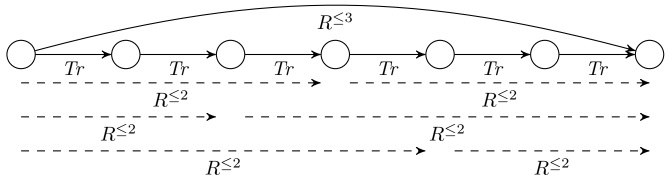

Fig. 2. Three different ways of decomposing path of length 6 into two paths of length at most 4

Splitting arises naturally from an attempt to fix this redundancy. The reasoning is as follows: Instead of concatenating two steps of $R^{\leq n}$ to obtain $R^{\leq n+1}$, we replace one of these steps with a step of $R^{=n} = Tr^{2^n}$, which represents reachability in *exactly* 2^n steps. However, $R^{\leq n} \circ R^{=n}$ covers only paths of length from 2^n to 2^{n+1}. To keep the smaller lengths covered as well, we can add $R^{\leq n}$. The result, $R^{\leq n+1} = R^{\leq n} \cup R^{\leq n} \circ R^{=n}$, *almost* gives us the unique deconstruction we are seeking. The exceptions are paths of length *exactly* 2^n which are covered by both $R^{\leq n}$ and $R^{\leq n} \circ R^{=n}$. The final step is a realization that this last redundancy is removed by replacing the relation $R^{\leq n}$ by $R^{<n}$. The sequence $R^{<}$ has the following inductive definition:[1]

$$\begin{aligned} R^{<0} &= Id, \\ R^{<n+1} &= R^{<n} \cup R^{<n} \circ R^{=n}, \end{aligned} \tag{2}$$

with the sequence $R^{=}$ also defined inductively:

$$\begin{aligned} R^{=0} &= Tr, \\ R^{=n+1} &= R^{=n} \circ R^{=n}. \end{aligned} \tag{3}$$

Notice that we have effectively *split* the $R^{\leq}$ sequence into two sequences $R^{<}$ and $R^{=}$, because $R^{\leq n} = R^{<n} \cup R^{=n}$. Now, decomposing a path according to the inductive definitions from Eq. (2) and (3) is *unique*. For example, there is only one way to decompose the path of length six from Fig. 2, now viewed as one step of $R^{<3}$, according to Eq. (2): first two steps are covered by $R^{<2}$ and the last four steps are covered by $R^{=2}$.

Following the TPA template, we do not use the sequences $R^{<}$ and $R^{=}$ directly. We build over-approximating sequences TPA$^{<}$ and TPA$^{=}$ whose representation in terms of copies of state variables does not blow up with increasing n. The elements of the over-approximating sequences TPA$^{<}$ and TPA$^{=}$ are denoted as $ATr^{<n}$ and $ATr^{=n}$, respectively, and we require that

$$ATr^{<n} \supseteq R^{<n} = Id \cup Tr \cup Tr^2 \cup \dots \cup Tr^{2^n - 1}, \tag{4}$$

$$ATr^{=n} \supseteq R^{=n} = Tr^{2^n}. \tag{5}$$

SPLIT-TPA uses these over-approximating sequences TPA$^{<}$ and TPA$^{=}$ both for bounded reachability checks and for detecting safe transition invariants. We will see later that TPA$^{=}$ sequence allows SPLIT-TPA to find interesting invariants and prove safety of challenging problems. The main procedure of SPLIT-TPA is similar to Algorithm 1 and is given in Algorithm 3.

Algorithm 3: ISSAFESPLITTPA($\langle Init, Tr, Bad \rangle$): SPLIT-TPA's main procedure

```
input  : transition system $\mathcal{S} = \langle Init, Tr, Bad \rangle$
global : TPA$^{<}$ sequence $ATr^{<0}, \dots, ATr^{<n}, \dots$
         TPA$^{=}$ sequence $ATr^{=0}, \dots, ATr^{=n}, \dots$ (lazily
initialized to $true$)
1 $ATr^{<0} \leftarrow Id$;   $ATr^{=0} \leftarrow Tr$;   $n \leftarrow 0$
2 while true do
3     if ISREACHABLELT($n, Init, Bad$) $\neq \emptyset$ or
        ISREACHABLEEQ($n, Init, Bad$) $\neq \emptyset$ then return
        UNSAFE
4     if HASTRANSITIONINVARIANT($\mathcal{S}, n$) then return SAFE
5     $n \leftarrow n + 1$
```

In the rest of this section we present the implementation of the methods ISREACHABLELT and ISREACHABLEEQ for bounded reachability checks and the implementation of the method HASTRANSITIONINVARIANT for discovering safe transition invariant.

[1]An alternative inductive definition $R^{<n+1} = R^{<n} \cup R^{=n} \circ R^{<n}$ leads to a different variant of our algorithm.

B. Bounded reachability checks with TPA$^=$ and TPA$^<$

SPLIT-TPA performs the bounded reachability check at level n in two phases. First, all paths of length *strictly smaller* than 2^{n+1} are checked in ISREACHABLELT. Then all paths of length *exactly* 2^{n+1} are checked in ISREACHABLEEQ.

To implement ISREACHABLEEQ, we can reuse Algorithm 2, with the modification that all references to TPA sequence and its elements $ATr^{\leq n}$ are replaced by TPA$^=$ sequence and its elements $ATr^{=n}$ (we do not repeat the pseudocode for the sake of space). To understand why this works, compare the inductive definitions of the underlying sequences $R^{\leq}$ and $R^{=}$ from Eq. (1) and (3). The induction step is *the same* in both cases. The only difference is the base case: TPA$^=$ sequence starts with $ATr^{=0} = R^{=0} = Tr$, as opposed to $ATr^{\leq 0} = R^{\leq 0} = Id \cup Tr$. The output of ISREACHABLEEQ is either a non-empty subset of Tgt that is truly reachable from Src in exactly 2^{n+1} steps of Tr, or an empty set if no path from Src to Tgt of length 2^{n+1} exists.

The procedure ISREACHABLELT is designed to complement ISREACHABLEEQ by covering all paths with $<2^{n+1}$ steps. The implementation is given in Algorithm 4. It follows the inductive definition of $R^<$ from Eq. (2) in the same manner as ISREACHABLEEQ follows the inductive definition of $R^=$.

Algorithm 4: Reachability query using TPA$^<$ sequence

```
input : level $n$, source states $Src$, target states $Tgt$
output: subset of target states truly reachable in $<2^{n+1}$ steps
global : TPA$^<$ sequence $ATr^{<0}, \ldots, ATr^{<n}, \ldots$,
         TPA$^=$ sequence $ATr^{=0}, \ldots, ATr^{=n}, \ldots$
1  while true do
2      $opt1 \leftarrow ATr^{<n}[x' \mapsto x'']$
3      $opt2 \leftarrow ATr^{<n}(x, x') \wedge ATr^{=n}(x', x'')$
4      $q \leftarrow Src(x) \wedge (opt1 \vee opt2) \wedge Tgt(x'')$
5      $sat\_res, model \leftarrow$ CHECKSAT($q$)
6      if $sat\_res$ = UNSAT then
7          $Itp(x, x'') \leftarrow$ GETITP($opt1 \vee opt2, Src(x) \wedge Tgt(x'')$)
8          $ATr^{<n+1} \leftarrow ATr^{<n+1} \wedge Itp[x'' \mapsto x']$
9          return $\emptyset$
10     else
11         if $n = 0$ then return QE($\exists x, x' : q$)$[x'' \mapsto x]$
12         if $model \models opt1$ then
13             $TgtReach \leftarrow$ ISREACHABLELT($n-1, Src, Tgt$)
14             if $TgtReach = \emptyset$ then continue
15             return $TgtReach$
16         else
17             $Inter \leftarrow$ QE($\exists x, x'' :$
                 $Src(x) \wedge opt2 \wedge Tgt(x''), x'$)$[x' \mapsto x]$
18             $InterReach \leftarrow$ ISREACHABLELT($n-1, Src, Inter$)
19             if $InterReach = \emptyset$ then continue
20             $TgtReach \leftarrow$
                 ISREACHABLEEQ($n-1, InterReach, Tgt$)
21             if $TgtReach = \emptyset$ then continue
22             return $TgtReach$
```

ISREACHABLELT first assembles the query for an abstract path (lines 2–4) and sends it to the satisfiability solver (line 5). Following the inductive definition of Eq. (2), the abstract path consists of either one step of $ATr^{<n}$ or a step of $ATr^{<n}$ followed by a step of $ATr^{=n}$. If no such abstract path exists (line 6), the procedure reports that no real path of length $<2^{n+1}$ exists (line 9). Before reporting the result, it uses Craig interpolation [15] to refine the abstraction at the next level (line 8).

If an abstract path exists (line 10), the procedure checks whether there is a corresponding real path. On level 0 (line 11), the discovered abstract path is real, and the procedure returns a reachable subset of target states. On other levels, the procedure first needs to determine which abstract path has been found and then try to refine it.

The first possibility is that the abstract path is a single step of $ATr^{<n}$ (line 12). The refinement of this single abstract step is checked with a single recursive call. If the refinement is not successful, the procedure attempts to find a new abstract path (line 14). Otherwise, the reached target states from the recursive call are returned (line 15).

The second possibility is that abstract path consists of one step of $ATr^{<n}$ followed by one step of $ATr^{=n}$ (line 16). One after another, the procedure attempts to refine these abstract steps into a real path by calling the corresponding procedures ISREACHABLELT and ISREACHABLEEQ with decreased bound. If any of the two steps cannot be refined, that abstract path has been refuted and the procedure attempts to find a new abstract path (lines 19, 21). If both abstract steps have been successfully refined, a reachable subset of target states is reported (line 22).

Similarly to Algorithm 2, quantifier elimination can be replaced by its under-approximation, such as model-based projection, and we do so in our implementation.

The correctness of the reachability procedures guarantees the correctness of UNSAFE answer of SPLIT-TPA.

Lemma 1: If ISREACHABLEEQ(n, Src, Tgt) or ISREACHABLELT(n, Src, Tgt) returns a non-empty set Res, then $Res \subseteq Tgt$ and every state in Res can be reached from some state in Src in exactly 2^{n+1} steps (for ISREACHABLEEQ) or in $<2^{n+1}$ steps (for ISREACHABLELT).

Proof: By induction on n, relying on the properties of quantifier elimination (QE) and the fact that $ATr^{<0} = Id$ and $ATr^{=0} = Tr$ represent true reachability. ■

Theorem 1: If SPLIT-TPA (Algorithm 3) returns UNSAFE, then there exists a counterexample path in the system, i.e., some bad state is reachable from some initial state.

Proof: Follows directly from Lemma 1. ■

C. Proving safety by discovering safe transition invariants

If a bounded safety has been proved on level n in Algorithm 3, i.e., there is no counterexample path of length $\leq 2^{n+1}$ in the system, then the algorithm attempts to extend the bounded proofs to unbounded ones. The procedure HASTRANSITIONINVARIANT tries to construct a (grounded) transition invariant based on the elements of TPA$^=$ and TPA$^<$ sequences. If a safe transition invariant is found, SPLIT-TPA has proven *unbounded safety*.

We have identified sufficient conditions for the elements $ATr^{<n}$ and $ATr^{=n}$ that guarantee the existence of a transition

invariant. These conditions are formalized in Lemma 2 and Lemma 3, respectively.

Lemma 2: Assume that for some n, $Init \lhd ATr^{<n} \circ Tr \subseteq Init \lhd ATr^{<n}$. Then $ATr^{<n}$ is a left-grounded transition invariant.

If $Tr \circ ATr^{<n} \rhd Bad \subseteq ATr^{<n} \rhd Bad$, then $ATr^{<n}$ is a right-grounded transition invariant.

Proof: Suppose that $s \in Init$ and $(s,t) \in Tr^*$, i.e., t is reachable from s. We show that $(s,t) \in ATr^{<n}$ by induction on d, the length of minimal path from s to t.

Base case $d < 2^n$: $(s,t) \in ATr^{<n}$ holds by Eq. (4).

Induction step: Suppose that the claim holds for all paths of length d. We show that then it also holds for all paths of length $d+1$. Consider a path between s and t of length $d+1$. Then t has a predecessor m on this path, i.e., m lies d steps from s and reaches t in 1 step. Then $(s,m) \in ATr^{<n}$ by the induction hypothesis. Since $(m,t) \in Tr$ it follows that $(s,t) \in ATr^{<n} \circ Tr$. Since $s \in Init$, it follows by the assumption of the lemma that $(s,t) \in ATr^{<n}$.

We have shown that if $s \in Init$ and $(s,t) \in Tr^*$ then $(s,t) \in ATr^{<n}$. Thus $ATr^{<n}$ is a left-grounded transition invariant. The case of the right-grounded transition invariant is analogous. In the inductive case, we consider m the successor of s on the path from s to t. ■

Note that with a slightly stronger assumption we can use the same proof idea to discover full transition invariants:

Observation 1: If $ATr^{<n} \circ Tr \subseteq ATr^{<n}$ or $Tr \circ ATr^{<n} \subseteq ATr^{<n}$ then $ATr^{<n}$ is a transition invariant.

Discovering transition invariants based on TPA$^{<}$ sequence is similar to how the invariants were detected in TPA sequence in [5]. This is not surprising, as the elements $ATr^{<n}$ and $ATr^{\leq n}$ have similar properties. The key advantage of SPLIT-TPA is the additional ability to discover transition invariants by detecting fixed points in the TPA$^{=}$ sequence.

Lemma 3: Assume that for some n, $Init \lhd ATr^{<n} \circ ATr^{=n} \circ ATr^{=n} \subseteq Init \lhd ATr^{<n} \circ ATr^{=n}$ then $Init \lhd Tr^* \subseteq Init \lhd ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$.

If $ATr^{=n} \circ ATr^{=n} \circ ATr^{<n} \rhd Bad \subseteq ATr^{=n} \circ ATr^{<n} \rhd Bad$ then $Tr^* \rhd Bad \subseteq ATr^{<n} \cup ATr^{=n} \circ ATr^{<n} \rhd Bad$.

Proof: The proof uses the same ideas as the proof of Lemma 2. Suppose that $s \in Init$ and $(s,t) \in Tr^*$, i.e., t is reachable from s. We proceed by induction on d, the length of minimal path from s to t.

Base case $d < 2^{n+1}$: It follows by Eq. (4) and (5) that $(s,t) \in ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$.

Induction step: Assuming the claim holds for all paths of length d, we show that it also holds for all paths of length $d+2^n$. Consider a path between s and t of length $d+2^n$. There exists m on this path that lies d steps from s and reaches t in exactly 2^n steps. Then $(m,t) \in ATr^{=n}$ by Eq. (5) and $(s,m) \in ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$ by induction hypothesis. Consider the two cases:

- $(s,m) \in ATr^{<n}$: It follows that $(s,t) \in ATr^{<n} \circ ATr^{=n}$.
- $(s,m) \in ATr^{<n} \circ ATr^{=n}$: It follows that $(s,t) \in ATr^{<n} \circ ATr^{=n} \circ ATr^{=n}$. Then $(s,t) \in ATr^{<n} \circ ATr^{=n}$ by the assumption of the lemma.

We have shown that if $s \in Init$ and $(s,t) \in Tr^*$ then $(s,t) \in ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$. Thus $ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$ is a left-grounded transition invariant. For the right-grounded transition invariant, in the induction step pick m that lies exactly 2^n steps from s (and reaches Bad in d steps). ■

Similarly to Lemma 2, full transition invariants can be discovered by checking a stronger condition:

Observation 2: If $ATr^{=n} \circ ATr^{=n} \subseteq ATr^{=n}$ then both $ATr^{<n} \cup ATr^{<n} \circ ATr^{=n}$ and $ATr^{<n} \cup ATr^{=n} \circ ATr^{<n}$ are full transition invariants.

Note that transition invariants obtained using Lemma 3 are *disjunctive by definition*. The disjunctive structure reflects the inductive nature of the proof of Lemma 3. $ATr^{<n}$ corresponds to the base case and represents the bounded part of the proof; $ATr^{=n}$ corresponds to the induction step and represents the unbounded part of the proof. Since the induction step makes 2^n steps of Tr instead of 1, the unbounded proof corresponds to k-induction rather than induction.

The procedure HASTRANSITIONINVARIANT checks the conditions of Lemma 2 and Lemma 3 using an SMT solver. For example, $ATr^{=n} \circ ATr^{=n} \circ ATr^{<n} \rhd Bad \subseteq ATr^{=n} \circ ATr^{<n} \rhd Bad$ *iff* $ATr^{=n}(x,x') \wedge ATr^{=n}(x',x'') \wedge ATr^{<n}(x'',x''') \wedge Bad(x''') \wedge \neg ATr^{=n}(x,x'')$ is *unsatisfiable*. When the procedure discovers a grounded transition invariant it must also verify that the invariant is *safe*, i.e., it does not relate any initial with any bad state. This can also be checked with a single satisfiability query. In the case of transition invariant detected using conditions of Lemma 2, the check is not even necessary. The invariant, which is $ATr^{<n}$ for some n, is guaranteed to be safe after ISREACHABLELT proved bounded safety on level $n-1$.

The detection of safe (grounded) transition invariants as described above allows SPLIT-TPA to prove safety and the correctness is guaranteed by Lemma 2 and Lemma 3.

Theorem 2: If SPLIT-TPA returns SAFE, there is no counterexample path from $Init$ to Bad in $\mathcal{S}$.

To demonstrate the behaviour of SPLIT-TPA, recall the loops from Fig. 1. For the first loop, similarly to TPA, SPLIT-TPA quickly proves bounded safety up to $8192 = 2^{13}$ iterations of the loop, and in the process learns that $ATr^{<n} \equiv x' < x + 2^n$ and that $ATr^{=n} \equiv x' \leq x + 2^n$ for $n = 1 \dots 13$. In the next iteration of its main loop, SPLIT-TPA discovers an abstract path consisting of a step of $ATr^{<13}$ followed by a step of $ATr^{=13}$. After some time spent in the refinement, the algorithm manages to refute all abstract paths and proves bounded safety for $< 2^{14}$ iterations. As part of the refinement, it strengthens $ATr^{=13}$ to include the facts $x' = x + 8192$ and $x \leq 1808$. With this strengthened information, it can easily prove that no path of length exactly $2^{14} = 16384$ exists because it is not possible to make two steps of the abstract relation $ATr^{=13}$ from the initial state. In addition, it learns that $ATr^{=14} \equiv x \leq -6384$. This satisfies the condition of Observation 2, namely $ATr^{=14} \circ ATr^{=14} \subseteq ATr^{=14}$. Thus SPLIT-TPA concludes at this point that the system is safe.

When analyzing the second loop, SPLIT-TPA behaves differently than TPA. After proving bounded safety in the first iteration of Algorithm 3, SPLIT-TPA learns that $ATr^{=1} \equiv x > z \implies w' \geq w + 2$. In the next iteration, $ATr^{=1}$ is strengthened with facts $x \geq z \implies w' \geq w + 1$ and $x < z \implies w' \geq w$. These three facts together concisely over-approximate the change to w after precisely two iterations of the loop. Moreover, $ATr^{=1}$ with these three components is closed under composition, i.e., $ATr^{=1} \circ ATr^{=1} \subseteq ATr^{=1}$. Thus, SPLIT-TPA concludes already at this point that the system is safe (based on Observation 2). The transition invariant, using $\vec{a} = (x, z, v, w)$, is then $ATr^{<1}(\vec{a}, \vec{a}'') \vee (ATr^{<1}(\vec{a}, \vec{a}') \wedge ATr^{=1}(\vec{a}', \vec{a}''))$, where

$$\begin{aligned} ATr^{<1}(\vec{a}, \vec{a}') \equiv\ & w' \geq w \wedge v' \leq v \wedge \\ & ((x' \geq x \wedge z' \leq z) \vee (x' \geq x + 1 \wedge z' \leq z + 2)), \end{aligned}$$

$$\begin{aligned} ATr^{=1}(\vec{a}, \vec{a}') \equiv\ & x > z \rightarrow w' \geq w + 2 \wedge \\ & x \geq z \rightarrow w' \geq w + 1 \wedge \\ & x < z \rightarrow w' \geq w. \end{aligned}$$

Note that the exact value of $ATr^{<1}$ is not important in this case, as long as it over-approximates all paths of length <2.

V. From transition invariants to state invariants

In Section IV-C, we have shown how SPLIT-TPA can prove a transition system safe by finding a safe transition invariant. However, many applications require a proof of safety in the form of a safe inductive (state) invariant. Here we show that (k-)inductive invariants can be obtained from the discovered transition invariants by quantifying over the source or target states. This follows Lemma 2 and Lemma 3 and their proofs.

Lemma 4: Assume that for some n, the following holds:

$$\mathit{Init} \triangleleft ATr^{\leq n} \circ Tr \subseteq \mathit{Init} \triangleleft ATr^{\leq n}.$$

Then the following is an inductive invariant:

$$\mathit{Inv}(x') \equiv \exists x : \mathit{Init}(x) \wedge ATr^{\leq n}(x, x').$$

Proof: Analogous to the proof of Lemma 2. Intuitively, *Inv* represents all states reachable by one step of $ATr^{<n}$ from *Init*. Since $ATr^{<n}$ is a left-grounded transition invariant by Lemma 2, making one additional step of *Tr* cannot end up outside this set. Also, $\mathit{Init} \subseteq \mathit{Inv}$, because $\mathit{Id} \subseteq ATr^{\leq n}$, i.e., *Inv* holds in the initial states. ■

Lemma 5: Assume that for some n, the following holds:

$$Tr \circ ATr^{\leq n} \triangleright \mathit{Bad} \subseteq ATr^{\leq n} \triangleright \mathit{Bad}.$$

If $ATr^{<n}$ is safe, then the following is an inductive invariant:

$$\mathit{Inv}(x) \equiv \neg(\exists x' : ATr^{\leq n}(x, x') \wedge \mathit{Bad}(x')).$$

Proof: Analogous to the proof of Lemma 4. ■

Compared to Lemma 2, the proof of Lemma 3 uses an inductive step of size 2^n. Following that proof we can turn the transition invariant from TPA$^=$ into 2^n-inductive invariant.

Lemma 6: Assume that for some n, the following holds:

$$\mathit{Init} \triangleleft ATr^{<n} \circ ATr^{=n} \circ ATr^{=n} \subseteq \mathit{Init} \triangleleft ATr^{<n} \circ ATr^{=n}.$$

Then the following is 2^n-inductive invariant:

$$\begin{aligned} \mathit{Inv}(x'') \equiv\ & \exists x, x' : \mathit{Init}(x) \wedge \\ & (ATr^{<n}(x, x'') \vee (ATr^{<n}(x, x') \wedge ATr^{=n}(x', x''))). \end{aligned}$$

Proof: We follow the proof of Lemma 3. *Inv* represents the set of states reachable from *Init* either by one step of $ATr^{<n}$ or by a combined step of $ATr^{<n}$ and $ATr^{=n}$. It follows that *Inv* over-approximates the set of states reachable from *Init* in less than 2^{n+1} steps of *Tr*. Thus, *Inv* satisfies the base step of k-induction (for $k = 2^n$).

For the inductive step, we need to prove that making 2^n steps of *Tr* from an *Inv*-state leads again to an *Inv*-state. We rely on Eq. (5), i.e., $ATr^{=n} \supseteq Tr^{2^n}$. If s is an *Inv*-state, then it is reachable from some initial state i either in one step of $ATr^{<n}$ or in one step of $ATr^{<n} \circ ATr^{=n}$. Moreover, all states reachable from s in 2^n steps of *Tr* are reachable from s by one step of $ATr^{=n}$. Thus, in the first case, they are reachable from i in one step of $ATr^{<n} \circ ATr^{=n}$. In the second case, they are reachable from i in one step of $ATr^{<n} \circ ATr^{=n} \circ ATr^{=n}$. Based on the assumption of the lemma, they are reachable from i also in one step of $ATr^{<n} \circ ATr^{=n}$. ■

Lemma 7: Assume that for some n, the following holds:

$$ATr^{=n} \circ ATr^{=n} \circ ATr^{<n} \triangleright \mathit{Bad} \subseteq ATr^{=n} \circ ATr^{<n} \triangleright \mathit{Bad}.$$

If $ATr^{<n}(x, x'') \vee (ATr^{=n}(x, x') \wedge ATr^{<n}(x', x''))$ is safe then the following is 2^n-inductive invariant:

$$\begin{aligned} \mathit{Inv}(x) \equiv\ & \neg(\exists x', x'' : \mathit{Bad}(x'') \wedge \\ & (ATr^{<n}(x, x'') \vee (ATr^{=n}(x, x') \wedge ATr^{<n}(x', x'')))). \end{aligned}$$

Proof: Analogous to the proof of Lemma 6. ■

Note that in each given case, the (k-)inductive invariants are quantified and quantifier elimination must be applied if quantifier-free inductive invariants are required. Inductive invariants can be obtained from k-inductive invariants by quantifying over the intermediate states [29].

VI. Experiments

We have implemented SPLIT-TPA in our Horn solver GOLEM[2]. In our experiments we used GOLEM 0.1.0, which uses OPENSMT 2.3.2 for SMT solving and interpolation.

The goal of the experiments was to compare SPLIT-TPA to TPA [5], which is also available in GOLEM, and to state-of-the-art tools ELDARICA 2.0.8 [26], Z3-SPACER [30] implemented in Z3 4.8.17 [35], and GSPACER [22] a more recent version enriched with global guidance. All experiments were conducted on a machine with AMD EPYC 7452 32-core processor and 8x32 GiB of memory. We used a timeout of 5 minutes for every task.[3]

For the evaluation we used the set of benchmarks representing multi-phase loops [39], which are known to be challenging for automated analysis techniques. We used both the safe

[2]https://github.com/usi-verification-and-security/golem.git

[3]Full results at http://verify.inf.usi.ch/content/split-tpa-experiments, artifact at https://doi.org/10.5281/zenodo.6988735

TABLE I
SUMMARY OF THE EXPERIMENTS ON MULTI-PHASE BENCHMARKS.

Benchmark suite	SPLIT-TPA	TPA	Z3SPACER	GSPACER	ELDARICA
multi-phase safe	19 (7)	12 (0)	6 (0)	24 (3)	26 (4)
multi-phase unsafe	37 (3)	35 (2)	20 (0)	17 (0)	17 (0)

Solved (unique) instances out of 54 benchmarks.

TABLE II
FULL RESULTS ON SAFE (LEFT) AND UNSAFE BENCHMARKS (RIGHT)

Ben.	SPLIT-TPA	TPA	Z3SPACER	GSPACER	ELDARICA
01	**26.28**	TO	TO	TO	TO
02	TO	TO	133.28	**<1**	TO
03	TO	TO	TO	TO	**1.33**
04	TO	TO	TO	**<1**	3.70
05	**<1**	**<1**	**<1**	**<1**	1.19
06	TO	TO	TO	TO	**3.95**
07	TO	TO	TO	**<1**	1.32
08	TO	TO	TO	TO	TO
09	TO	TO	TO	TO	TO
10	TO	TO	TO	TO	TO
11	TO	TO	TO	**5.68**	TO
12	TO	TO	TO	TO	**1.62**
13	**<1**	**<1**	ERR	**<1**	1.16
14	**53.94**	TO	TO	TO	118.78
15	TO	TO	TO	TO	TO
16	TO	TO	TO	TO	TO
17	**<1**	37.50	TO	**<1**	7.53
18	**<1**	**<1**	TO	**<1**	3.66
19	TO	TO	**<1**	**<1**	1.22
20	TO	TO	TO	TO	TO
21	**<1**	10.39	TO	**<1**	15.45
22	TO	TO	TO	TO	TO
23	**<1**	**<1**	ERR	**<1**	1.79
24	TO	TO	TO	TO	TO
25	TO	45.93	TO	TO	**9.33**
26	2.60	1.55	TO	**<1**	TO
27	TO	TO	TO	TO	TO
28	**<1**	TO	TO	TO	1.61
29	**3.94**	TO	TO	118.98	34.22
30	TO	TO	TO	TO	**20.48**
31	TO	TO	TO	**<1**	1.60
32	TO	TO	TO	**11.49**	TO
33	TO	TO	TO	TO	TO
34	TO	TO	TO	**<1**	5.86
35	TO	TO	TO	**<1**	1.80
36	**<1**	**<1**	TO	**<1**	1.92
37	**<1**	**<1**	**<1**	**<1**	14.33
38	TO	**<1**	TO	**<1**	1.36
39	TO	TO	67.41	58.73	**2.48**
40	**109.05**	TO	TO	TO	ERR
41	TO	TO	TO	TO	TO
42	TO	TO	TO	**<1**	4.37
43	TO	TO	TO	**5.20**	TO
44	TO	TO	TO	TO	TO
45	TO	TO	TO	TO	TO
46	TO	288.20	13.07	**<1**	1.28
47	TO	TO	TO	TO	TO
48	**47.00**	TO	TO	TO	TO
49	**122.96**	TO	TO	TO	TO
50	TO	TO	TO	TO	TO
51	TO	TO	TO	TO	TO
52	**235.24**	TO	TO	TO	TO
53	**147.28**	TO	TO	TO	TO
54	**133.63**	TO	TO	TO	TO

Ben.	SPLIT-TPA	TPA	Z3SPACER	GSPACER	ELDARICA
01	14.53	**10.12**	TO	TO	TO
02	**<1**	**<1**	1.25	TO	TO
03	**<1**	**<1**	**<1**	**<1**	1.16
04	TO	TO	TO	TO	TO
05	**<1**	**<1**	**<1**	**<1**	1.18
06	TO	TO	TO	TO	TO
07	TO	TO	TO	TO	TO
08	TO	TO	TO	TO	TO
09	TO	TO	TO	TO	TO
10	**20.40**	233.78	TO	TO	TO
11	**152.28**	TO	TO	TO	TO
12	TO	TO	TO	TO	TO
13	**<1**	**<1**	**<1**	**<1**	1.13
14	**<1**	**<1**	**<1**	8.91	89.78
15	TO	TO	TO	TO	TO
16	TO	TO	TO	TO	TO
17	**14.84**	15.81	181.59	TO	TO
18	**<1**	**<1**	**<1**	**<1**	1.57
19	**<1**	**<1**	**<1**	**<1**	20.74
20	TO	TO	TO	TO	TO
21	**<1**	**<1**	**<1**	**<1**	10.63
22	TO	TO	TO	TO	TO
23	**<1**	**<1**	**<1**	**<1**	1.17
24	**<1**	TO	96.64	TO	TO
25	**<1**	**<1**	**<1**	**<1**	1.19
26	2.01	**1.46**	TO	TO	TO
27	**<1**	**<1**	TO	TO	TO
28	**<1**	**<1**	TO	TO	162.43
29	**<1**	**<1**	2.76	32.56	45.75
30	**<1**	**<1**	**<1**	**<1**	10.22
31	TO	TO	TO	TO	TO
32	**<1**	**<1**	**<1**	**<1**	7.17
33	**<1**	**<1**	**<1**	**<1**	1.21
34	**<1**	**<1**	**<1**	**<1**	1.15
35	**<1**	**<1**	**<1**	**<1**	1.20
36	16.68	**14.45**	TO	TO	TO
37	**<1**	**<1**	**<1**	**<1**	13.37
38	**262.18**	TO	TO	TO	TO
39	TO	TO	TO	ERR	TO
40	**<1**	**<1**	**<1**	133.07	ERR
41	TO	**4.60**	TO	TO	TO
42	**18.31**	40.39	TO	TO	TO
43	TO	TO	TO	TO	TO
44	**34.18**	TO	TO	TO	TO
45	TO	TO	TO	TO	TO
46	TO	**239.05**	TO	TO	TO
47	**5.71**	6.79	TO	TO	TO
48	17.52	**12.10**	TO	TO	TO
49	32.59	**12.49**	TO	TO	TO
50	TO	TO	TO	TO	TO
51	**6.71**	11.57	TO	TO	TO
52	**70.83**	82.43	TO	TO	TO
53	57.42	**33.00**	TO	TO	TO
54	40.74	**15.15**	TO	TO	TO

TO: timeout; ERR: memory out or other inconclusive answer.

versions of the benchmarks from CHC-COMP repository[4] and the unsafe versions of the benchmarks from [5]. The results are summarized in Table I and times for each tool/benchmark pair are given in Table II.

Regarding safety, Table I shows that SPLIT-TPA overall solved 7 more benchmarks than TPA, but still less than GSPACER or ELDARICA. However, it solved seven benchmarks *uniquely* (the other competitors did not solve them). This indicates that SPLIT-TPA is quite orthogonal to the existing techniques for proving safety.

The results on unsafe benchmarks show that SPLIT-TPA not only preserves the capability of TPA to detect deep counterexample, but it was even able to outperform it by solving two more benchmarks overall.

[4]https://github.com/chc-comp/aeval-benchmarks

Besides the multi-phase benchmarks, we also evaluated the tools on a general benchmark set from the LRA-TS category of CHC-COMP 2021, the latest edition with a publicly available selected benchmark set.[5] Out of 498 benchmarks, SPLIT-TPA proved 128 benchmarks safe and 72 unsafe. TPA proved 62 benchmarks safe and 71 unsafe. Even though the performance of SPLIT-TPA still lacks behind Z3-SPACER and GSPACER (ELDARICA does not support arithmetic over reals) on CHC-COMP benchmarks, it still achieved a significant improvement over TPA, especially on safe benchmarks.

To better understand the advantage of SPLIT-TPA over TPA, we collected statistics from the runs of SPLIT-TPA on safe instances to see which transition invariants it used to prove safety. In our implementation $\mathrm{TPA}^{<}$ is checked before $\mathrm{TPA}^{=}$. Moreover, each sequence element is first checked for a full transition invariant. This is followed by checks for left-grounded and finally right-grounded transition invariant.

On CHC-COMP2021 LRA-TS benchmarks, out of 128 benchmarks proven safe, 63 invariants were discovered from $\mathrm{TPA}^{<}$ and 65 invariants were discovered with $\mathrm{TPA}^{=}$. Regardless of the sequence, 81 were full transition invariants and 47 were left-grounded transition invariants. Surprisingly, no (purely) right-grounded transition invariants were discovered. For safe multi-phase benchmarks the results were similar. Out of 19 invariants, 15 invariants were found with $\mathrm{TPA}^{=}$ and 4 invariants were found with $\mathrm{TPA}^{<}$. Fifteen of these invariants were full transition invariants and 4 were left-grounded. Again, no purely right-grounded transition invariant was found. These statistics confirm the essential role of the $\mathrm{TPA}^{=}$ sequence in SPLIT-TPA as a source of transition invariants.

VII. RELATED WORK

Many model-checking algorithms search for a safe inductive invariant to prove safety. Candidates for inductive invariants are typically obtained from proofs of bounded safety. The algorithms try to construct the safe inductive invariant either in monolithic [32, 34, 38] or incremental way [8, 10, 17, 24, 30]. Our work follows a similar strategy, but it primarily computes *transition* invariants, not state invariants.

Transition invariants have been introduced in [36] as a proof rule for program verification, especially termination and other liveness properties. Transition predicate abstraction [37] has been introduced as a way to compute transition invariants. In contrast, we use transition invariants to prove safety, with candidates automatically obtained from proofs of bounded safety using Craig interpolation.

Craig interpolation [15] is a popular abstraction technique widely used in model checking. We use standard algorithms to compute interpolants from proofs of unsatisfiability [6, 13, 33]. The integration of domain-specific knowledge [31] is future work.

While in most model checking algorithms interpolants are used as over-approximations of *states*, we use them to over-approximate *transitions*. The idea of abstracting transition

[5]https://github.com/chc-comp/chc-comp21-benchmarks/tree/main/LRA-TS

relation with interpolants originates from [28]. However, they maintained an abstraction of only a single step of the transition relation. We build two sequences of relations over-approximating doubling number of steps of the transition relation, which are useful both for detecting deep counterexamples and as a source of candidates for safe transition invariant.

Loop acceleration [2, 7, 19] is a loop analysis technique that can prove safety and detect deep counterexamples. However, on its own, it is applicable only to limited types of integer loops. Acceleration have also been successfully integrated into interpolation-based model checking [9, 25] where interpolants computed from accelerated paths lead to much better abstraction refinement in the traditional CEGAR algorithm [14]. In contrast, SPLIT-TPA computes transition interpolants, not state interpolants. It also does not try to capture all possible behaviour of a loop (by accelerating it). Instead, it builds over-approximations of (exponentially increasing) bounded number of iterations. By relying purely on Craig interpolation it can handle transition relations where acceleration is not possible.

The k-induction principle [20] has been successfully used as a replacement for basic inductive reasoning in IC3-style algorithms [21, 23, 29]. k-inductive invariants can be more compact than inductive invariants and for some theories k-induction is a strictly stronger proof rule [29]. SPLIT-TPA uses both inductive reasoning (applied to $\mathrm{TPA}^{<}$) and k-inductive reasoning (applied to $\mathrm{TPA}^{=}$) to discover transition invariants. We believe that SPLIT-TPA's success on challenging systems can be in large part attributed to the inclusion of k-inductive reasoning, which was missing in TPA [5].

VIII. Conclusion

In this work we have presented SPLIT-TPA, a novel instantiation of a recently introduced TPA approach. Splitting the transition power abstraction into two complementary parts makes the algorithm more efficient in proving safety by detecting safe transition invariants while still retaining and even improving the capability of detecting long counterexamples. The advantage of our instantiation has been confirmed experimentally on a set of challenging multi-phases benchmarks and on an extensive general benchmark set from CHC-COMP. The experiments also show that SPLIT-TPA is both competitive and complementary compared to state-of-the-art in safety verification. As the next step, we plan to study extensions of SPLIT-TPA from transition systems to general CHC systems.

Acknowledgments

This work was partially supported by Swiss National Science Foundation grant 200021_185031, Czech Science Foundation grant 20-07487S, and the National Science Foundation (of the United States) grant 2106949. The authors thank the anonymous reviewers for their valuable feedback and suggestions.

References

[1] Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer International Publishing, Cham (2022)

[2] Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: acceleration from theory to practice. International Journal on Software Tools for Technology Transfer **10**(5), 401–424 (2008)

[3] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. Journal of Automated Reasoning **60**(3), 299–335 (Mar 2018)

[4] Bjørner, N., Janota, M.: Playing with quantified satisfaction. In: Fehnker, A., McIver, A., Sutcliffe, G., Voronkov, A. (eds.) LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations. EPiC Series in Computing, vol. 35, pp. 15–27. EasyChair (2015)

[5] Blicha, M., Fedyukovich, G., Hyvärinen, A., Sharygina, N.: Transition power abstraction for deep counterexample detection. In: Tools and Algorithms for Construction and Analysis of Systems (2022)

[6] Blicha, M., Hyvärinen, A.E.J., Kofroň, J., Sharygina, N.: Decomposing Farkas interpolants. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 3–20. Springer International Publishing, Cham (2019)

[7] Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification. pp. 227–242. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[8] Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 70–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[9] Caniart, N., Fleury, E., Leroux, J., Zeitoun, M.: Accelerating interpolation-based model-checking. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 428–442. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[10] Cimatti, A., Griggio, A.: Software model checking via IC3. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification. pp. 277–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[11] Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 modulo theories via implicit predicate abstraction. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 46–61. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[12] Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 93–107. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[13] Cimatti, A., Griggio, A., Sebastiani, R.: Efficient generation of Craig interpolants in satisfiability modulo theories. ACM Trans. Comput. Logic **12**(1), 7:1–7:54 (Nov 2010)

[14] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification. pp. 154–169. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[15] Craig, W.: Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. The Journal of Symbolic Logic **22**(3), 269–285 (1957)

[16] Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer-Aided Verification (CAV'2014). Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (July 2014)

[17] Een, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design. pp. 125–134. FMCAD '11, FMCAD Inc, Austin, TX (2011)

[18] Fedyukovich, G., Prabhu, S., Madhukar, K., Gupta, A.: Solving Constrained Horn Clauses Using Syntax and Data. In: FMCAD. pp. 170–178. IEEE (2018)

[19] Frohn, F.: A calculus for modular loop acceleration. In: Biere, A., Parker, D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 58–76. Springer International Publishing, Cham (2020)

[20] Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) Computer Aided Verification. pp. 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)

[21] Gurfinkel, A., Ivrii, A.: K-induction without unrolling. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 148–155 (Oct 2017)

[22] Hari Govind, V.K., Chen, Y., Shoham, S., Gurfinkel, A.: Global guidance for local generalization in model checking. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. pp. 101–125. Springer International Publishing, Cham (2020)

[23] Hari Govind, V.K., Vizel, Y., Ganesh, V., Gurfinkel, A.: Interpolating strong induction. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 367–385. Springer International Publishing, Cham (2019)

[24] Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing – SAT 2012. pp. 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[25] Hojjat, H., Iosif, R., Konečný, F., Kuncak, V., Rümmer, P.: Accelerating interpolants. In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis. pp. 187–202. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[26] Hojjat, H., Rümmer, P.: The ELDARICA Horn Solver. In: FMCAD. pp. 158–164. IEEE (2018)

[27] Hyvärinen, A.E.J., Marescotti, M., Alt, L., Sharygina, N.: OpenSMT2: An SMT solver for multi-core and cloud computing. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing – SAT 2016. pp. 547–553. Springer International Publishing, Cham (2016)

[28] Jhala, R., McMillan, K.L.: Interpolant-based transition relation approximation. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification. pp. 39–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[29] Jovanović, D., Dutertre, B.: Property-directed k-induction. In: 2016 Formal Methods in Computer-Aided Design (FMCAD). pp. 85–92 (Oct 2016)

[30] Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Formal Methods in System Design **48**(3), 175–205 (Jun 2016)

[31] Leroux, J., Rümmer, P., Subotić, P.: Guiding Craig interpolation with domain-specific abstractions. Acta Informatica **53**(4), 387–424 (2016)

[32] McMillan, K.L.: Interpolation and SAT-based model checking. In: Computer Aided Verification. pp. 1–13. Springer, Berlin Heidelberg (2003)

[33] McMillan, K.L.: An interpolating theorem prover. Theoretical Computer Science **345**(1), 101 – 121 (2005)

[34] McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) Computer Aided Verification. pp. 123–136. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

[35] de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[36] Podelski, A., Rybalchenko, A.: Transition invariants. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004. pp. 32–41 (2004)

[37] Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 132–144. POPL '05, Association for Computing Machinery, New York, NY, USA (2005)

[38] Rümmer, P., Hojjat, H., Kuncak, V.: On recursion-free Horn clauses and Craig interpolation. Formal Methods In System Design **47**(1), 1–25 (2015)

[39] Sharma, R., Dillig, I., Dillig, T., Aiken, A.: Simplifying loop invariant generation using splitter predicates. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification. pp. 703–719. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)