Towards Verification Modulo Theories of asynchronous systems via abstraction refinement

Gianluca Redondi
Fondazione Bruno Kessler
Trento, Italy
gredondi@fbk.eu

Alessandro Cimatti
Fondazione Bruno Kessler
Trento, Italy
cimatti@fbk.eu

Alberto Griggio
Fondazione Bruno Kessler
Trento, Italy
griggio@fbk.eu

Abstract—This paper introduces a new algorithm designed to verify safety properties for asynchronous compositions of symbolic transition systems. The approach combines under-approximation and over-approximation: on one side, it zooms in on a selected set of components, while forcing the remaining ones to stutter; on the other, the selected components are individually abstracted and re-composed. This strategy can be advantageous for scenarios involving large numbers of components, where only a small subset of key components allows to produce the right invariants for the system. We detail the application of our algorithm to a class of parameterized symbolic transition systems, by using a form of slicing as an abstraction. Our experimental results, although preliminary, show the potential of the approach.

Index Terms—Compositional Approach, CEGAR, Parameterized Systems

I. Introduction

This paper focuses on the problem of asynchronous verification, where multiple systems with shared variables undergo transitions independently. The problem is studied in the literature across various formalisms. In this paper, we adopt the formalism of Verification Modulo Theories [3], using symbolic transition systems defined by formulae in SMT. We describe an algorithm that we aim to use in a verification project related to railway interlocking logic [1], [5], which is characterized by numerous components and a large number of variables. However, the verification of safety properties may necessitate examining only a select few of these components. Additionally, many variables within these components are not integral to managing safety operations. Hence, our algorithm seeks to abstract numerous system variables while potentially concentrating on the pertinent components.

Initially, we outline the algorithm in a generic scenario, not focusing on particular abstraction or class of systems. Then, we narrow our focus to a more concrete use case. We define a class of transition systems capable of modeling parameterized systems and instantiate the aforementioned approach by providing specific procedures for abstraction and refinement. Such a scenario can model, for example, the interlocking logic we are interested in.

We have developed a prototype of the algorithm and tested it on some artificial benchmarks and a simplified case study related to interlocking logic. The outcomes are encouraging, suggesting that this algorithm could also perform effectively on the entire logic system once we acquire the comprehensive system descriptions.

The paper is organized as follows: Section 2 provides the necessary background, and it studies the connections between abstraction and composition. Section 3 details the procedure for the verification of systems defined via asynchronous composition. Section 4 specializes the algorithm in the case of parameterized systems and presents the experimental evaluation. Finally, Section 5 describes our conclusions and outlines avenues for future work.

II. Background

A. Preliminaries

Our models of computation are symbolic transition systems, i.e. triples of the form $(X, I(X), T(X, X'))$, where X is a set of variables, called the *state variables* of the system, and $I(X)$, $T(X, X')$ are formulae over some theory $\mathcal{T}$. Given a model $\mathcal{M}$ for $\mathcal{T}$, a state s is a valuation of the state variables X in the universe of $\mathcal{M}$. A state is initial iff it is a model of $I(X)$, i.e., $s \models I(X)$. A pair of states s, s' denotes a transition iff $s, s' \models T(X, X')$. A state s is reachable iff there exists a path π such that $\pi[i] = s$ for some i.

A formula $\phi(X)$ is an invariant of the transition system $C = (X, I(X), T(X, X'))$ iff it holds in all the reachable states. Following the standard model checking notation, we denote this as $C \models \phi(X)$. We say that ϕ is inductive for C if $I(X) \models \phi(X)$ and $\phi(X) \wedge T(X, X') \models \phi(X')$.

In the following, we will use the notion of *case-defined functions* defined in a theory $\mathcal{T}$. These functions are defined by a sequence of couples $\{(case_i, value_i)\}_{i=1}^n$ where each $case_i$ is a predicate and each $value_i$ is a term (they correspond to statements of the form *if case_1 then value_1, else ...*); moreover, all the case predicates are required to be mutually exclusive, and their disjunction is a valid formula. Although case-defined functions are not standard in the SMT setting, they can be easily handled by using, for example appropriate if-then-else terms.

B. Abstraction

Let C be $(X, I(X), T(X, X'))$ and $\tilde{C}$ be $(\tilde{X}, \tilde{I}(\tilde{X}), \tilde{T}(\tilde{X}, \tilde{X}'))$. Let S be the set of states of C, and $\tilde{S}$ be the set of states of $\tilde{C}$. Let α be a relation between S and $\tilde{S}$; we write $\alpha(s, \tilde{s})$ to denote that two states s and $\tilde{s}$ are in relation.

Definition 1. *We say that $\tilde{C}$ α-simulates C (written $C \rightarrow_\alpha \tilde{C}$) if the following two conditions hold:*

i. *For each initial state s of C, there exists an initial state $\tilde{s}$ of $\tilde{C}$ such that $\alpha(s, \tilde{s})$ holds.*

ii. *For each pair $(s, \tilde{s})$ such that $\alpha(s, \tilde{s})$ holds, and for each $s' \in S$ such that $s, s' \models T(X, X')$, there exists a state $\tilde{s}'$ such that $\alpha(s', \tilde{s}')$ holds and $\tilde{s}, \tilde{s}' \models \tilde{T}(\tilde{X}, \tilde{X}')$.*

If α is clear in the context, we might say that $\tilde{C}$ simulates (or *abstracts*) C.

Let $V \subseteq X \cap \tilde{X}$ be a set of variables, and $F(V)$ be a formula. We have the following facts about simulations:

Definition 2. *We say that the simulation $C \rightarrow_\alpha \tilde{C}$ preserves the formula F if, for all states s such that $s \models \neg F$, then for all $\tilde{s}$ such that $\alpha(s, \tilde{s})$, $\tilde{s} \models \neg F$.*

Proposition 1. *Given a simulation $C \rightarrow_\alpha \tilde{C}$ that preserves F, $\tilde{C} \models F \Rightarrow C \models F$.*

Proposition 2. *Given a simulation $C \rightarrow_\alpha \tilde{C}$ that preserves F, if F is inductive for $\tilde{C}$, then F is inductive for C.*

In many cases, the abstract variables $\tilde{X}$ of the system $\tilde{C}$ are different from the original variables X. In this paper, we consider only the case $\tilde{X} \subseteq X$ for the sake of simplicity.

If a counterexample is found in $\tilde{C}$, in general, this does not imply the existence of a counterexample in C. We say that a counterexample π in $\tilde{C}$ is spurious if there exists no path $s_0, \ldots, s_k$ in C such that $s_n \models \neg F$ and, for all $0 \leq i \leq k$, $\alpha(s_i, \pi[i])$. In such cases, the abstraction yields no helpful information and needs to be refined.

C. Asynchronous Composition

Let $C_1 = (X_1, I_1(X_1), T(X_1, X_1'))$ and $C_2 = (X_2, I(X_2), T(X_2, X_2'))$ be two symbolic transition systems. If V is a set of variables, we denote with $Inertia(V)$ the formula $\bigwedge_{v \in V}(v = v')$.

Definition 3. *The asynchrounous product between C_1 and C_2, is the transition system $C_1 \parallel C_2 = (X_1 \cup X_2, I_{C_1 \parallel C_2}(X_1, X_2), T_{C_1 \parallel C_2}(X_1, X_2, X_1', X_2'))$, where:*

- *$I_{C_1 \parallel C_2}(X_1, X_2)$ is the formula $I_1(X_1) \wedge I_2(X_2)$;*
- *$T_{C_1 \parallel C_2}$ is the formula $(T_1(X_1, X_1') \wedge Inertia(X_2 \setminus X_1)) \vee (T_2(X_2, X_2') \wedge Inertia(X_1 \setminus X_2))$.*

Given a set of variables $V \subseteq X_1 \cup X_2$ and a formula $F(V)$, asynchronous verification amounts to prove or disprove if $(C_1 \parallel C_2) \models F(V)$. More generally, if $V \not\subseteq X_i$, we may write $C_i \models F(V)$ with the meaning that we add to the X_i the remaining $V \setminus X_i$ variables, and we modify T_i by adding inertia on $V \setminus X_i$. We have:

Proposition 3. *If a formula F is not inductive for $C_1 \parallel C_2$, then there exists an $i \in \{1, 2\}$ such that F is not inductive for C_i.*

We say that two transition system C_1 and C_2 are *compatible* if each partial assignment to the shared variables can be extended to an initial state of C_1 if and only if it can be extended to an initial state of C_2. In practice, this means that the shared variables are initialized in the same way in the two systems.

Proposition 4. *Consider $C_1 \rightarrow_{\alpha_1} \tilde{C}_1$ and $C_2 \rightarrow_{\alpha_2} \tilde{C}_2$ two simulation relations. Suppose that $\tilde{C}_1$ and $\tilde{C}_2$ are compatible. Consider $\alpha_1 \parallel \alpha_2$ (called the product simulation) defined as*

$$\alpha_1 \parallel \alpha_2(s, \tilde{s}) \text{ iff } \alpha_1(s|_{X_1}, \tilde{s}|_{X_1}) \text{ and } \alpha_2(s|_{X_2}, \tilde{s}|_{X_2}).$$

Then, $C_1 \parallel C_2 \rightarrow_{\alpha_1 \parallel \alpha_2} \tilde{C}_1 \parallel \tilde{C}_2$.

By definition of product simulation, we have the following corollary:

Corollary 1. *Consider $C_1 \rightarrow_{\alpha_1} \tilde{C}_1$ and $C_2 \rightarrow_{\alpha_2} \tilde{C}_2$ two simulation relation such that they both preserve a formula F. Assume that $\tilde{C}_1$ and $\tilde{C}_2$ are compatible. Then, $\alpha_1 \parallel \alpha_2$ also preserves F.*

III. A COMPOSITIONAL APPROACH WITH ABSTRACTION REFINEMENT

In this section, we outline a procedural framework that remains parametric, considering a generic family of transition systems, a generic abstraction procedure, and a target invariant property denoted as F. In the next section, we delve into a case study where we provide a more concrete setting.

Suppose that we have a finite family of transition systems $\{C_i\}_{i \in I}$. Let $C = \parallel_{i \in I} C_i$ be the asynchronous composition of the systems, and consider a formula $F(V)$ with $V \subseteq \bigcup_{i \in I} X_i$. The problem that we face is to prove or disprove whether $C \models F$. The problem is solved if either we find a counterexample, i.e. a path π of finite length n, such that $\pi[n] \models \neg F$, or if we find an inductive invariant Ψ for F. If F is not inductive itself, then by consecutive applications of Proposition 3 it follows that there exists a subset J of I such that F is not inductive for $\parallel_{j \in J} C_j$.

To describe our algorithm, we assume to have some subprocedures, namely: (i) a model checker, capable of automatically proving if an invariant holds in a transition system. If so, the model checker provides an inductive invariant for it. Otherwise, the model checker find a counterexample; (ii) a theorem prover, capable of checking whether a formula is inductive for a transition system, or if a counterexample can be simulated (e.g by bounded model checking). As a preprocessing step, we suppose to identify the set of components J for which the property is not already inductive. Moreover, let $\tilde{C}$ be a transition system such that there exists a simulation $\parallel_{j \in J} C_j \rightarrow \tilde{C}$. The only property that we require on the abstraction is that the simulation should preserve all inductive invariants found by the model checker. We consider the following procedure, depicted in Figure 1:

- we start by asking a model checker if $\tilde{C} \models F$. The model checker can either find an inductive invariant, Ψ, or a counterexample, π;
- If an invariant is found, we ask the prover to check if Ψ is also inductive for the whole asynchronous composition C. Note that, since the simulation preserves Ψ, we already know that it is inductive for the components $\{C_j\}_{j \in J}$ by Proposition 2.
- If the prover proves the induction, then we are done. Otherwise, there must exist a new set of components $J' \subseteq I \setminus J$ for which the induction check fails. We thus update the set J to be equal to $J \cup J'$, and we restart the loop by updating the abstraction $\tilde{C}$.
- Suppose instead that the model checker finds a counterexample in $\tilde{C}$. Then, we ask the prover if the counterexample can be simulated by C. If so, the algorithm terminates with a counterexample. Otherwise, we refine the abstraction to remove the abstract counterexample.

The key differences of our approach from conventional CEGAR methods in compositional verification such as [13], [7] lies in the fact that the system $\tilde{C}$ doesn't abstract entire composition $C = \parallel_{i \in I} C_i$; instead, it abstracts only the under-approximation $\parallel_{j \in J} C_j$. We could eventually abstract all components, when $J = I$, and $C \to \tilde{C}$ is simulation - but our method is best suited for situations where this should not happen.

Even when $\tilde{C}$ doesn't represent the entire system, the algorithm's soundness is evident. This is because we conduct an additional induction check to verify whether the invariant identified during model checking is also inductive for the broader composition C.

IV. VERIFICATION OF CONCURRENT PARAMETERIZED SYSTEMS

In this section, we illustrate the application of the procedure outlined in Section III through a specific use case. Our algorithm was conceived to facilitate the verification of interlocking logic within railway stations, as part of a larger initiative to integrate various formal methods into railway design [1]. In this scenario, the system's components are each represented by parameterized transition systems. These components interact by sharing multiple variables; our objective is to ascertain the general safety of the composition of them. Despite the presence of a vast number of components, only a subset is critical for ensuring safety. Furthermore, while the systems may involve a large number of variables, we recognize that only a limited number are pertinent to safety concerns. Consequently, our approach to abstraction focuses on narrowing down to these essential variables.

The systems are modeled with a subclass of the formalism of [14] and similar to [9], where parameterized systems are modeled as symbolic transition systems with state variables that are functions from an uninterpreted theory (with finite but unbounded universes) to a generic theory. Moreover, quantifiers occur in the system description to model the unboundedness of possible instances.

Definition 4. *An array-based transition system $C = (X, I(X), T(X, X'))$ is a symbolic transition system where:*
- *X are a set function symbols;*
- *$I(X)$ is a formula of the form $\forall j. \bigwedge_{x \in Y} x(j) = val_x$ - where val_x is a constant of the appropriate signature, and $Y \subseteq X$;*
- *$T(X, X')$ is a disjunction of formulae of the form (called transition rules)*

$$\exists i. (\phi_G(i, X) \land \phi_U(i, X, X')) \qquad (1)$$

where $\phi_G(i, X)$ is called the guard, and $\phi_U(i, X, X')$ is the functional update, i.e., a formula of the form

$$\forall j. \bigwedge_{x \in X} x'(j) = F_x(i, j, X, X')$$

with $\{F_x\}_{x \in X}$ a family of case-defined function.

We start by defining the simulation relation that we will use.

Definition 5. *Let $V \subseteq X$ a set of variables. We define a relation α between two assignments $s, \tilde{s}$ of X and V by*

$$\alpha(s, \tilde{s}) \Leftrightarrow s|_V \equiv \tilde{s},$$

i.e. we ask that the two states are in relation iff they assign the same value to the variables in V.

Given a (sub)set of variables V and a transition system C defined as in Definition 4, we now define a new transition system $\tilde{C}^V$ such that there exists a simulation between the two systems. The new variables will be $V \cup B \cup E$, with B and E sets of fresh input variables. The abstract initial formula of the system, denoted $\tilde{I}(V)$, is simply obtained from I by dropping the conjuncts that are not assigning variables in V; that is, we have that

$$\tilde{I}(V) = \bigwedge_{v \in V} \forall j. v(j) = val_v.$$

For the abstract transition formula, denoted as $\tilde{T}(V, B, E, V')$, we need more steps. We will work on the single transition rules of the concrete transition, that are of the form (1). The abstract transition will be a disjunction of formulae of the form $\exists i. (\tilde{\phi}_G(i, V) \land \tilde{\phi}_U(i, V, B, E, V'))$ where
- $\tilde{\phi}_G(i, V)$ is obtained by ϕ_G by replacing each atom that contains variables in $X \setminus V$ with the constant $true$, if it occurred positively in the formula, or $false$ otherwise;
- $\tilde{\phi}_U(i, V, B, E, V')$ is the formula $\bigwedge_{v \in V} \forall j. v'(j) = \tilde{F}_v(i, j, V, B, E, V')$ where $\tilde{F}_v$ is a case-define function with a sequence of $\tilde{case}_i(V, B)$ statements and a sequence of corresponding terms $\tilde{val}_i(V, B, E)$ such that:
  - $\tilde{case}_i(V, B)$ is either equal to $case_i(V)$ if the original case predicate is defined only over the V variables, or is a fresh boolean constant $b \in B$ otherwise;
  - $\tilde{val}_i(V, B, E)$ is either equal to $val_i(V)$ if the original term is defined only over V, or is a fresh constant $e \in E$ of the appropriate type otherwise.

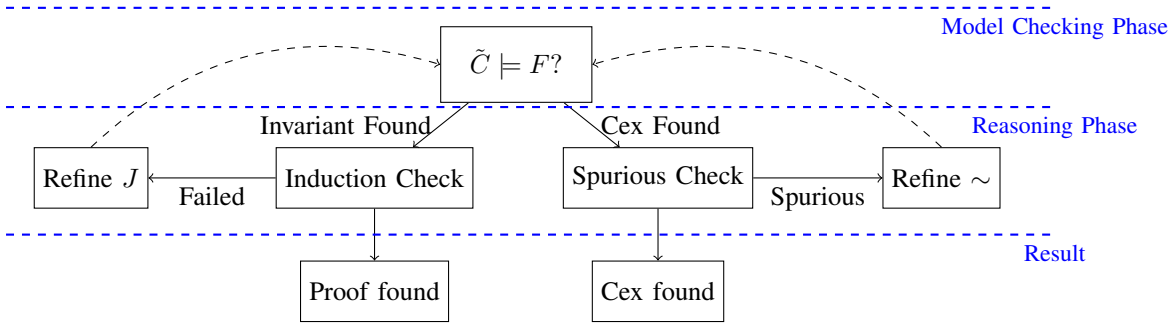Let $\tilde{C}^V = (\{V, B, E\}, \tilde{I}(V), \tilde{T}(V, B, E, V'))$. We have:

Fig. 1. The procedure

Proposition 5. $\tilde{C}^V$ simulates C.

Moreover, since the simulation relation is the equality on V, we have that:

Proposition 6. *The simulation preserves each formula $F(V)$ defined only over the set of variables V.*

Therefore, in order for the abstraction $\sim^V$ to preserve the property, we need that the set of variables V always include all the state variables occuring in the property to prove. Thus, from Corollary 1, we have that, given any family of *compatible* array-based transition systems:

Corollary 2. $\parallel C_i \rightarrow \parallel \tilde{C}_i^V$ *via the product simulation. Moreover, the product simulation preserves all the formulas defined over V.*

Thanks to the latter, we can use as an abstraction for $\parallel_{i \in J} C_i$ the composition of the individual abstractions, i.e. $\parallel_{i \in J} \tilde{C}_i^V$.

A. Refinement

Suppose that, during the model checking phase, we found an abstract counterexample π. For refinement, we check as usual the satisfiability of the concrete unrolling. In the case of satisfiability, we are in the presence of a real counterexample, and we can exit from the procedure. In case of unsatisfiability, we are in presence of a spurious counterexample, and we follow this refinement procedure: we start by computing an unsat core of the latter formula. Then, let V' be the set of variables that occur in at least one literal of the core and not in V. We update V to be $V \cup V'$. We have the following result that ensures that ensures that V' is never empty:

Proposition 7. *In case a spurious counterexample in $\parallel_{i \in J} \tilde{C}_i^V$ is found, then there exists a literal in the unsat core of the concrete unrolling that contains a variable not occurring in V.*

This refinement allows us to have a notion of progress, since at each spurious counterexample we decrease the number of variables that not abstracted.

B. Implementation and first results

We developed the algorithm presented in the preceding section using Python3, leveraging the SMT solvers Z3 and Mathsat, along with the parameterized model checker Lambda [4]. Given that Mathsat lacks support for quantified formulae, we exclusively utilized Z3 for the 'Induction Check' sub-procedure. For 'Spurious Check' and 'Refinement' sub-procedures, both Mathsat5 and Z3 were employed. Lambda, capable of processing system descriptions in the VMT language [6], is able to synthesize inductive invariants for systems as defined in Definition 4.

We tested the algorithm on a simplified case study of the railway logic, with 5 parameterized components (with a total of 15 variables) and two properties. For the set of abstracted variables V, we always initialize it with the set of state variables occuring in the property to prove. The first property was verified by the algorithm in 4.2 seconds, by using only 5 variables and 2 components, and one refinement step. Instead, the monolithic approach (i.e. using the model checker on the entire system description of the composition) took around 9 seconds. On the other hand, the second property was a false assertion whose counterexample involved most of the components and variables used. In that case, the monolithic approach can find a counterexample faster then the compositional algorithm. Although these outcomes are not entirely satisfactory, it's crucial to acknowledge that our case study was quite limited compared to the actual system. In reality, the system comprises over 100 components, each with numerous variables, and a full symbolic description has yet to be achieved. A second source of benchmarks derives from two parameterized protocols, which we have adapted by integrating N components capable of altering certain shared variables. We test the algorithm on properties that are true and are independent of the modifications enacted by these additional components. The results are depicted in Figure 2. The x-axis represents the number of components, while the y-axis measures the time taken by the procedures in seconds (presented on a logarithmic scale). Compared to the monolithic approach (in orange), our algorithm's verification process (in blue) is significantly less time-consuming and remains relatively constant. A virtual machine to replicate the results is available here, together with an extended version of this paper.
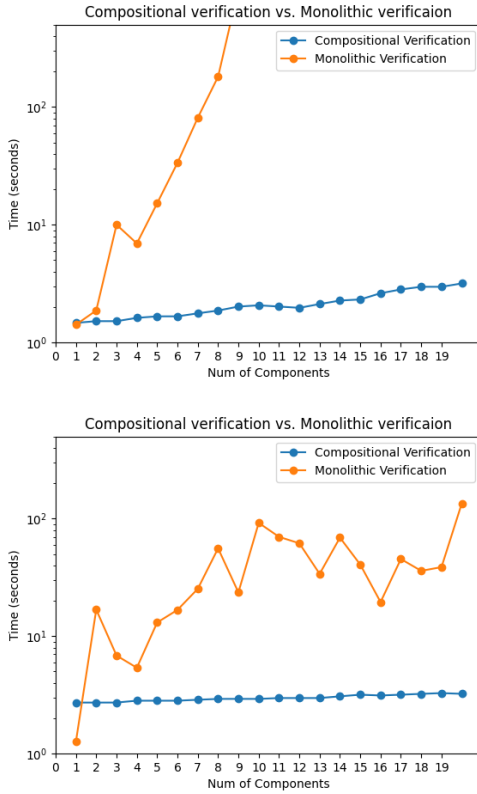
Fig. 2. Results on protocols with additional components

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced a method for verifying the safety properties of asynchronous compositions of symbolic transition systems defined over an SMT theory $\mathcal{T}$. We believe that this method is particularly suited for scenarios requiring the verification of properties across a large family of components, where the inductive invariant can be identified by examining a subset of those components. If the procedure terminates by abstracting only a subset of the components, then we can determine *a posteriori* that a split invariant [10] between the abstracted and non-abstracted components is found.

We applied this general algorithm to a family of symbolic transition systems designed to describe parameterized systems and defined a form of splicing as an abstraction strategy that concentrates on a subset of the system's variables. A prototype of the algorithm was developed and tested on simple benchmarks. The initial results are promising, and we plan to extend its application to a comprehensive verification project focused on interlocking logic, where the previously described situation (numerous components with many variables) frequently arises. Moreover, future work may integrate assumption-guarantee methods in our approach, such as those found in [2], [12], [11], [8] to further simplify the verification.

REFERENCES

[1] CAVADA, R., CIMATTI, A., GRIGGIO, A., AND SUSI, A. A formal IDE for railways: Research challenges. In *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops - AI4EA, F-IDE, CoSim-CPS, CIFMA, Berlin, Germany, September 26-30, 2022, Revised Selected Papers* (2022), P. Masci, C. Bernardeschi, P. Graziani, M. Koddenbrock, and M. Palmieri, Eds., vol. 13765 of *Lecture Notes in Computer Science*, Springer, pp. 107–115.

[2] CIMATTI, A., DORIGATTI, M., AND TONETTA, S. OCRA: A tool for checking the refinement of temporal contracts. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013* (2013), E. Denney, T. Bultan, and A. Zeller, Eds., IEEE, pp. 702–705.

[3] CIMATTI, A., GRIGGIO, A., MOVER, S., ROVERI, M., AND TONETTA, S. Verification modulo theories. *Formal Methods Syst. Des. 60*, 3 (2022), 452–481.

[4] CIMATTI, A., GRIGGIO, A., AND REDONDI, G. Verification of SMT systems with quantifiers. In *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings* (2022), A. Bouajjani, L. Holík, and Z. Wu, Eds., vol. 13505 of *Lecture Notes in Computer Science*, Springer, pp. 154–170.

[5] CIMATTI, A., GRIGGIO, A., AND REDONDI, G. Towards the verification of a generic interlocking logic: Dafny meets parameterized model checking, 2024.

[6] CIMATTI, A., GRIGGIO, A., AND TONETTA, S. The VMT-LIB language and tools. *CoRR abs/2109.12821* (2021).

[7] CLARKE, E., GRUMBERG, O., JHA, S., LU, Y., AND VEITH, H. Counterexample-guided abstraction refinement. In *Computer Aided Verification* (Berlin, Heidelberg, 2000), E. A. Emerson and A. P. Sistla, Eds., Springer Berlin Heidelberg, pp. 154–169.

[8] GHEORGHIU BOBARU, M., PĂSĂREANU, C. S., AND GIANNAKOPOULOU, D. Automated assume-guarantee reasoning by abstraction refinement. In *Computer Aided Verification* (Berlin, Heidelberg, 2008), A. Gupta and S. Malik, Eds., Springer Berlin Heidelberg, pp. 135–148.

[9] GHILARDI, S., AND RANISE, S. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Log. Methods Comput. Sci. 6*, 4 (2010).

[10] GIANNAKOPOULOU, D., NAMJOSHI, K. S., AND PASAREANU, C. S. Compositional reasoning. In *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Springer, 2018, pp. 345–383.

[11] GUPTA, A., MCMILLAN, K. L., AND FU, Z. Automated assumption generation for compositional verification. In *Computer Aided Verification* (Berlin, Heidelberg, 2007), W. Damm and H. Hermanns, Eds., Springer Berlin Heidelberg, pp. 420–432.

[12] LIMBRÉE, C., CAPPART, Q., PECHEUR, C., AND TONETTA, S. Verification of railway interlocking - compositional approach with ocra. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification* (Cham, 2016), T. Lecomte, R. Pinger, and A. Romanovsky, Eds., Springer International Publishing, pp. 134–149.

[13] LOISEAUX, C., GRAF, S., SIFAKIS, J., BOUAJJANI, A., AND BENSALEM, S. Property preserving abstractions for the verification of concurrent systems. *Form. Methods Syst. Des. 6*, 1 (jan 1995), 11–44.

[14] REDONDI, G., CIMATTI, A., GRIGGIO, A., AND MCMILLAN, K. Invariant checking for smt-based systems with quantifiers. *ACM Trans. Comput. Logic* (aug 2024). Just Accepted.