SMT-Based Verification of Parameterized Systems*

Arie Gurfinkel
SEI/CMU, USA
University of Waterloo, Canada
arie.gurfinkel@uwaterloo.ca

Sharon Shoham
Tel Aviv University, Israel
sharon.shoham@gmail.com

Yuri Meshman
Technion, Israel
syurim@gmail.com

ABSTRACT

It is well known that verification of safety properties of sequential programs is reducible to satisfiability modulo theory of a first-order logic formula, called a verification condition (VC). The reduction is used both in deductive and automated verification, the difference is only in whether the user or the solver provides candidates for inductive invariants. In this paper, we extend the reduction to parameterized systems consisting of arbitrary many copies of a user-specified process, and whose transition relation is definable in first-order logic modulo theory of linear arithmetic and arrays. We show that deciding whether a parameterized system has a universally quantified inductive invariant is reducible to satisfiability of (non-linear) Constraint Horn Clauses (CHC). As a consequence of our reduction, we obtain a new automated procedure for verifying parameterized systems using existing PDR and CHC engines. While the new procedure is applicable to a wide variety of systems, we show that it is a decision procedure for several decidable fragments.

CCS Concepts

•**Theory of computation** → **Automated reasoning;** *Logic and verification; Verification by model checking;*

Keywords

model checking, parameterized systems, safety verification, invariant inference

*This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with CMU for the operation of the SEI, a FFRDC and by EU – Seventh Framework Programme (FP7) under ERC grant agreements No. 321174-VSSC and No. 615688 – ERC-COG-PRIME. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US DoD. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM-0003426

FSE'16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2950330

1. INTRODUCTION

Many mutual exclusion algorithms, bus protocols, distributed algorithms, telecommunication protocols, and cache coherence protocols are designed to be run by an arbitrary number of threads or processes. Such protocols give rise to *parameterized systems*, where the number of processes is a parameter of the system. Any value of the parameter defines an instance of the parameterized system.

In this work, we are interested in verifying safety properties of parameterized systems. Namely, we would like to verify that the system is safe for *any value of the parameter.* We consider parameterized systems where each instance consists of N identical processes, executing asynchronously. Safety holds if every configuration that is reachable from an initial state via an execution of the system (with any number of processes) satisfies the safety property.

Due to the unbounded number of processes, the set of possible configurations of the system is infinite. Therefore, parameterized systems are a special case of infinite-state transition systems. One of the most useful techniques for proving safety of such systems, already advocated by Floyd [13], is using *inductive invariants.* Given a set of initial configurations Init, a transition relation Tr and a safety property P, we say that an invariant Inv is *inductive* for $\langle \mathit{Init}, \mathit{Tr}, P\rangle$ if Inv satisfies the following three conditions: (i) $\mathit{Init} \Rightarrow \mathit{Inv}$ (**initiation**). (ii) $\mathit{Inv} \Rightarrow P$ (**safety**). (iii) Inv is closed under Tr, i.e., for every step of Tr which starts in a configuration satisfying Inv, Inv also holds after executing Tr (**consecution**). It is well known that P holds in all the configurations that are reachable from Init via steps of Tr if and only if there exists an inductive invariant for $\langle \mathit{Init}, \mathit{Tr}, P\rangle$ (in a sufficiently powerful language).

Our goal in this work is to find inductive invariants that prove safety of a parameterized system (for all values of the parameter). To do so, we model the initial configurations Init, the transition relation Tr, and the safety property P of a parameterized system in first-order logic (modulo theory), and look for inductive invariants of the same form. Since we consider instances of the system with an arbitrary number of processes, we cannot fix the set of processes in the formulas describing Init, Tr and P. Likewise, a typical inductive invariant will not refer to a fixed set of processes. We, therefore, model process identifiers as logical variables, and allow for quantification over process identifiers both in Init, Tr, P, and in the invariants.

We observe that the verification condition of such a system can be written as a set of *Constrained Horn Clauses* (CHCs), where the invariant is given by an uninterpreted predicate $Inv(\cdot)$. The CHCs encode the initiation, safety and consecution requirements, and contain *quantified constraints*. Searching for an inductive invariant then amounts to solving the set of CHCs, i.e., finding an interpretation of $Inv(\cdot)$ that satisfies all clauses. To tackle this problem, we focus our attention on *universally quantified invariants* (or *universal invariants* in short) where process identifiers are universally quantified. Despite their simplicity, universal invariants are powerful enough to prove safety of many parameterized systems.

We show that under certain natural restrictions, searching for a universal invariant with a *fixed* number of universal quantifiers is reducible to the problem of solving quantifier-free CHCs. This reduction allows us to use existing CHC solvers to find *universal* inductive invariants.

As a consequence of our reduction, we obtain a new automated procedure for verifying parameterized systems using existing PDR and CHC engines. Our procedure is applicable to a wide variety of systems, and is a decision procedure for several decidable fragments, including Petri nets.

Interestingly, in the case of one-quantifier invariants, our reduction produces Owicki-Gries proof rule [20]. For a larger number of quantifiers, our reduction generalizes Owicki-Gries.

Our main contributions are: (a) A safety verification condition (VC) for parameterized systems in the form of CHCs with *quantified* constraints. A solution to the CHCs is an inductive invariant that certifies safety of all instances of the system. (b) Inference of universally quantified solutions for the VC with a fixed number of quantifiers. We reduce finding a solution with k universal quantifiers to solving *quantifier-free* CHCs. (c) A sufficient condition for a system with infinitely many processes to soundly approximate a system with unbounded (but finite) number of processes. (d) A procedure, by combining our technique with an iterative search for a finite counterexample, for verifying safety of parameterized systems, which is complete for interesting decidable fragments. (e) An implementation and initial experience verifying interesting parameterized protocols.

2. OVERVIEW

In this section we provide a short overview of our approach for safety verification of parameterized systems.

2.1 Motivating Example

Figure 1 presents an example of a parameterized system which describes an arbitrary number of agents moving around asynchronously. All agents run the code depicted in Figure 1. Each agent has a unique identifier i, and its own copy of the local variables, denoted $v[i]$. Agent i only changes its own copies of the local variables, but its functionality also depends on the local variables of other agents via universal global guards. In this example, an agent starts in state CHOOSE. It then computes some (local) *desired* location and changes its state to TRY. From TRY it changes its state to WAIT and sets *desired* to its *next* location. However, this change only happens under a global universal guard which makes sure that the *desired* location is different than both the *curr* and *next* locations of all the agents with higher ids. Next, the agent changes to MOVE. This transition is only enabled when *next* is different than both the *curr* and *next* locations of all the agents with smaller ids. Finally, in MOVE, the agent changes its *curr* location to *next*. Note that the last two steps are not performed atomically: first

local
  $pc : \{\text{CHOOSE}, \text{TRY}, \text{WAIT}, \text{MOVE}\}$;
  $curr, next, desired : \text{Location}$
def $\texttt{proc}(i)$ **:**
  do
    $pc[i] = \text{CHOOSE}$ **:** $desired[i] := *$; $pc[i] := \text{TRY}$;
    $pc[i] = \text{TRY} \wedge \forall j\,.\, i < j \Rightarrow curr[j] \neq desired[i] \wedge next[j] \neq desired[i]$ **:**
      $next[i] := desired[i]$; $pc[i] := \text{WAIT}$;
    $pc[i] = \text{WAIT} \wedge \forall j\,.\, j < i \Rightarrow next[i] \neq curr[j] \wedge next[i] \neq next[j]$ **:**
      $pc[i] := \text{MOVE}$;
    $pc[i] = \text{MOVE}$ **:**
      $curr[i] := next[i]$; $pc[i] := \text{CHOOSE}$;
def $\texttt{init}(i, j)$ **:**
  $pc[i] = \text{CHOOSE} \wedge curr[i] = next[i] \wedge (i \neq j \Rightarrow curr[i] \neq curr[j])$
def $\texttt{bad}(i, j)$ **:**
  $i \neq j \wedge curr[i] = curr[j]$

Figure 1: Collision avoidance [7].

the agent changes its state to MOVE, and only in the next step it actually moves. The safety property requires that no collisions occur.

This example has two sources of infinity which make the verification task challenging. First, the locations are taken from an infinite domain. Second, and more importantly, the number of agents is unbounded. Fortunately, the theory of Linear Integer Arithmetic allows SMT-based verifiers to deal with the first source of infinity. However, to tackle the second obstacle and verify the system for *any* number of agents, we need to consider *quantified* invariants that prove correctness of all instances of the system simultaneously, while also taking into account infinite domains of variables.

2.2 Our Approach

We view the problem of finding an invariant for the parameterized system as the problem of finding a solution to a set of Constrained Horn Clauses (CHCs) which capture the initiation, consecution and safety requirements of an inductive invariant. In order to formulate the problem in this way, we model the initial states, the transitions and the bad states using first-order logic formulas. This modeling reveals several subtle points that we discuss at length in Section 5.

Having formulated the set of CHCs, all that remains is to solve it. However, since the solution we seek is quantified, we cannot use off-the-shelf solvers. We, therefore, develop a specialized approach targeted at finding *simple universal invariant*. By *universal* we mean that the invariant is definable by a formula with a prefix of universal quantifiers, and by *simple* we mean that quantifier-free body of the invariant does not use functions over the quantified variables. For example, if i is a quantified variable, then $i + 1$ is not allowed in the body.

Our solution strategy is to (a) fix the number of quantifiers expected in the invariant, (b) instantiate the quantifiers eagerly, and (c) use existing solvers for inference of quantifier-free safe inductive invariants to discover solutions of the reduced CHC system. In the rest of this section, we demonstrate this strategy on two special cases of one and two quantifiers. The general case is presented in Section 6.

One Quantifier. When applying our search strategy for in-

$$
\begin{aligned}
Init(i,i,\overline{v}) &\Rightarrow I_1(i,\overline{v}) \\
I_1(i,\overline{v}) \wedge Tr(i,\overline{v},\overline{v}') &\Rightarrow I_1(i,\overline{v}') && (1)\\
I_1(i,\overline{v}) \wedge I_1(j,\overline{v}) \wedge Tr(j,\overline{v},\overline{v}') \wedge j \neq i &\Rightarrow I_1(i,\overline{v}') && (2)\\
I_1(i,\overline{v}) \wedge I_1(j,\overline{v}) &\Rightarrow \neg Bad(i,j,\overline{v})
\end{aligned}
$$

Figure 2: $VC_1(T)$ for one-quantifier invariants.

$$
\begin{aligned}
& Init(i,j,\overline{v}) \wedge Init(j,i,\overline{v}) \wedge \\
& \quad Init(i,i,\overline{v}) \wedge Init(j,j,\overline{v}) \Rightarrow I_2(i,j,\overline{v}) \\
& \quad I_2(i,j,\overline{v}) \wedge Tr(i,\overline{v},\overline{v}') \Rightarrow I_2(i,j,\overline{v}') && (3)\\
& \quad I_2(i,j,\overline{v}) \wedge Tr(j,\overline{v},\overline{v}') \Rightarrow I_2(i,j,\overline{v}') && (4)\\
& I_2(i,j,\overline{v}) \wedge I_2(i,k,\overline{v}) \wedge I_2(j,k,\overline{v}) \wedge \\
& \quad Tr(k,\overline{v},\overline{v}') \wedge k \neq i \wedge k \neq j \Rightarrow I_2(i,j,\overline{v}') && (5)\\
& \quad I_2(i,j,\overline{v}) \Rightarrow \neg Bad(i,j,\overline{v})
\end{aligned}
$$

Figure 3: $VC_2(T)$ for two-quantifier invariants.

variants with one quantifier, i.e., of the form $\forall i\,.\, I_1(i,\overline{v})$, on a system T, we obtain the set of constraints $VC_1(T)$ depicted in Figure 2. $VC_1(T)$ is based on a logical formulation of T that uses quantified id variables in order to refer to arbitrary agents. Technically, the initial states are constraint via a formula $\forall i,j\,.\, Init(i,j,\overline{v})$ which allows the initial constraint to refer to pairs of agents. The transition relation formula has the form $\exists i\,.\, Tr(i,\overline{v},\overline{v}')$, which reflects the property that some agent i moves in each step, and the bad states are constraint by $\exists i,j\,.\, Bad(i,j,\overline{v})$.

In systems with universally guarded commands, $Tr(i,\overline{v},\overline{v}')$ is itself a quantified formula. To complete the reduction to a set of CHCs, we additionally instantiate the quantifiers in Tr. The details are presented in Section 6, which also addresses existentially guarded transitions.

$VC_1(T)$ reduces the problem of finding a 1-quantifier solution of a set of linear CHC, to the problem of finding a quantifier-free solution of a set of non-linear CHC.

If Tr does not update any shared variable, as is the case in the collision avoidance example, then (2) is unnecessary and the set of CHCs is linear.

Relation to Owicki-Gries proof rule. $VC_1(T)$ is a parametric extension of the Owicki-Gries proof-rule for partial correctness of concurrent programs [20]. In particular, (1) corresponds to invariant preservation under a step of a process, and (2) to preservation under an interference.

Two Quantifiers. In practice, a one-quantifier invariant is useful only when Tr itself does not contain any global transitions (i.e., transitions guarded by quantified guards). This is not the case for the example in Figure 1, and $VC_1(T)$ does not have a solution.

We, therefore, extend our approach to two-quantifier invariants of the form $\forall i,j\,.\, I_2(i,j,\overline{v})$. Figure 3 depicts the set of constraints $VC_2(T)$ obtained by our reduction. To simplify the presentation, we again do not present additional instantiations that result from universal or existential guards in Tr. These are explained in Section 6.

Similarly to VC_1, the clause (3) corresponds to local preservation of the invariant, and the clauses (4) and (5) correspond to preservation under interference. As in $VC_1(T)$, if Tr does not update any shared variables, as is the case in the example in Figure 1, then (5) is removed and the system $VC_2(T)$ becomes linear.

$VC_2(T)$ captures the common *self-vs-other* proof pattern in verification of parameterized systems. That is, it can express reasoning about arbitrary many processes that reduces to reasoning about the current process *self* and its environment as abstracted by some *other* process. More refined capturing of the environment is possible with additional quantifiers. Intuitively, each additional quantifier corresponds to exposing an additional process in the environment. By solving $VC_2(T)$, an inductive invariant with 2 universal quantifiers is found for the example in Figure 1.

Completeness. While our approach is in general incomplete, it is complete in interesting special cases. Specifically, we show that for parameterized systems that form monotonic transition systems (such as Petri nets), iterating our approach with increasing number of quantifiers is guaranteed to terminate and find an inductive invariant. This matches known decidability results for such systems.

3. BACKGROUND

In this section, we give a brief overview of notation and other key concepts used in the paper.

We write $\overline{x}$ for a vector of elements $\langle x_1,\ldots,x_m\rangle$, x_i for the ith element of $\overline{x}$ and $|\overline{x}|$ for the length of $\overline{x}$. For a formula φ and a variable x, we write $\varphi(x)$ to denote that x is free in φ. Note that φ might have other free variables in addition to x. For a term a of `array` sort, we write $a[i]$ for the ith element of a (`(select a i)` in SMT-LIB syntax), and $a[i := v]$ for an array obtained from a by replacing the ith element by v (`(store a i v)` in SMT-LIB syntax). For a vector $\overline{a} = \langle a_1,\ldots,a_m\rangle$ of `array` terms we write $\overline{a}[i]$ as a shorthand for the vector $\langle a_1[i],\ldots,a_m[i]\rangle$.

Constrained Horn Clauses. Let $\mathcal{T}$ be a first-order theory over some signature including equality, and $\mathcal{V}$ and $\mathcal{P}$ be sets of variables and predicates, respectively. A *Constrained Horn Clause (CHC)* is a formula:

$$\forall \mathcal{V} \cdot (\phi \wedge p_1(\overline{x}_1) \wedge \cdots \wedge p_k(\overline{x}_k) \Rightarrow h(\overline{x})), \text{ for } k \geq 0$$

where ϕ is a (possibly quantified) constraint in $\mathcal{T}$; $\overline{x}_i, \overline{x} \subseteq \mathcal{V}$ are (possibly empty) vectors of variables; $p_i(\overline{x}_i)$ is an application $p(t_1,\ldots,t_n)$ of an n-ary predicate symbol $p \in \mathcal{P}$ for first-order terms t_i; and $h(\overline{x})$ is either defined analogously to p_i or is $\mathcal{P}$-free (i.e., no $\mathcal{P}$ symbols occur in h).

h is called the *head* of the clause and $\phi \wedge p_1(\overline{x}_1) \wedge \cdots \wedge p_k(\overline{x}_k)$ is called the *body*. A clause is *linear* if its body contains at most one predicate symbol, and *non-linear* otherwise. For simplicity of presentation, we usually omit the outermost universal quantifier. Unless otherwise specified, we assume that the theory $\mathcal{T}$ is the standard SMT combination of the theories of Linear Integer Arithmetic and Arrays.

A set C of CHCs is satisfiable modulo a theory $\mathcal{T}$ (with a canonical model $\mathcal{M}_\mathcal{T}$) if there exists an extension $\mathcal{J}$ of $\mathcal{M}_\mathcal{T}$ that interprets all of the predicate symbols in $\mathcal{P}$ such that each clause $c \in C$ is true under $\mathcal{J}$. In this case, we say that $\mathcal{J}$ is a *solution* of C. In practice, we are interested in $\mathcal{T}$-definable solutions. For simplicity of presentation, we identify the solution with the FO formulas that define the interpretation of the predicate symbols in $\mathcal{P}$ (when exist). For example, we say that the solution is $p(x,y) = x > y$,

$\langle proc \rangle$	$::=$	`def proc`(i) : `do` $\langle gcmds \rangle$	
$\langle gcmds \rangle$	$::=$	$\langle guard \rangle$: $\langle cmd \rangle$	guraded command
	$\mid$	$\langle gcmds \rangle \mid \langle gcmds \rangle$	nondeterministic choice
$\langle guard \rangle$	$::=$	$\varphi_\ell(i, \overline{v})$	local guard
	$\mid$	$\exists\, j \bowtie i \,.\, \varphi_g(i, j, \overline{v})$	existential global guard
	$\mid$	$\forall\, j \bowtie i \,.\, \varphi_g(i, j, \overline{v})$	universal global guard
$\langle cmd \rangle$	$::=$	$\ell[i] := expr$	assignmnet to local variable
	$\mid$	$b := expr$	assignment to shared variable
	$\mid$	$\langle cmd \rangle$; $\langle cmd \rangle$	sequential composition

Figure 4: Syntax of a process: $\varphi_\ell(i, \overline{v})$ denotes a formula over $Local(\overline{v})[i]$ and $Shared(\overline{v})$; $\varphi_g(i, j, \overline{v})$ denotes a formula over $Local(\overline{v})[i]$, $Shared(\overline{v})$, and $Local(\overline{v})[j]$; ℓ denotes a local variable, while b denotes a shared variable; $expr$ denotes an expression over $Shared(\overline{v})$ and $Local(\overline{v})[i]$.

meaning that the solution is a FO model $\mathcal{J}$ such that $\mathcal{J}(p) = \{(x, y) \mid x < y\}$.

Universal Horn Clauses (UHC) extend CHC by allowing universal quantification over predicates from $\mathcal{P}$ that appear in the body. Satisfiability and solution of UHC are defined analogously to CHC.

4. SAFETY OF PARAMETERIZED SYSTEMS

In this section, we present our syntax, semantics, and the safety verification problem of parameterized systems.

4.1 Syntax

We consider parameterized systems that consist of an arbitrary number of copies of a given process P with a (global) initial condition *Init* and a (global) error condition *Bad*.

Processes. A process P is defined by a set of variables $\overline{v}$ and a set of guarded commands, with the meaning that these guarded commands reside in a loop, where in each iteration one of the commands whose guard evaluates to true is chosen non-deterministically and is executed atomically. The process is parameterized by an identifier, denoted i. Figure 4 presents the syntax of a process (explained next).

Variables. As a generalization of several interaction modes between copies of the process, we consider two classes of variables in P: (i) *local variables*, denoted $Local(\overline{v})$, and (ii) *shared variables*, denoted $Shared(\overline{v})$. Technically, all copies of P share the same copy of the shared variables, while each copy of P has its own copy of each local variable. Local variables are accessed via a process id, e.g., $v[i]$ or $v[j]$.

We allow processes to read (but not write) local variables of other processes. Thus, the local variables can be further sub-divided into process-private (not read by others) and distributed-shared (written by one, read by others). However, this distinction is irrelevant for our purpose.

Each copy of P interacts with the other copies via shared variables or *global guards* that examine the local variables of other processes in a restricted manner, as explained below.

Universal and Existential Guards. Global guards are either existential or universal, and do not explicitly refer to specific process indices. We assume a static topology and a linear order on the process indices, where a process can only distinguish between the processes with greater or smaller indices.

Formally, a global guard has the form $\mathbb{Q}j \bowtie i \,.\, \varphi(i, j, \overline{v})$, where $\mathbb{Q} \in \{\exists, \forall\}$, $\bowtie \in \{<, >, \neq\}$, and $\varphi(i, j, \overline{v})$ is a formula over $Local(\overline{v})[i]$, $Shared(\overline{v})$, and $Local(\overline{v})[j]$. If $\mathbb{Q} = \forall$, the guard is called *universal*, and if $\mathbb{Q} = \exists$, the guard is called *existential*. The relation $\bowtie$ refers to the order between the copies of the process induced by the linear topology. An existential guard $\exists j \bowtie i \,.\, \varphi$ is satisfied by the ith copy of P iff there exists a copy j of P that satisfies $j \bowtie i$ and whose local state together with the local state of i satisfies φ. Similarly, a universal guard $\forall j \bowtie i \,.\, \varphi$ is satisfied by the ith copy of P iff the local state of each of the copies j of P that satisfy $j \bowtie i$ satisfies φ.

Other guards are called *local*. In particular, if-then-else commands are written using local guards. Note that local guards can refer to shared variables. We classify guarded commands as *local* or *global* based on their guards.

Initial and Bad States. The description of a parameterized system includes a description of the initial states, as well as a description of the bad states:

$$\text{def init}(i, j) : \varphi_{init}(i, j, \overline{v}) \qquad \text{def bad}(i, j) : \varphi_{bad}(i, j, \overline{v})$$

where $\varphi(i, j, \overline{v})$ is a formula over $Shared(\overline{v})$, $Local(\overline{v})[i]$ and $Local(\overline{v})[j]$. Informally, the initial condition φ_{init} has to hold for all pairs of processes in the initial state, while a bad condition is satisfied if some pair of processes satisfy φ_{bad}.

4.2 Asynchronous Operational Semantics

Each value $n \in \mathbb{N}$ defines an *instance* of the parameterized system which consists of n copies of P, which are linearly ordered. We refer to each process via an index $0 \leq i \leq n-1$. The semantics of the instance is given by a transition system, denoted $T_n(P)$ (or simply T_n when P is clear from the context), defined as follows.

Let L denote the set of *local* states, i.e., L consists of all possible valuations of the local variables $Local(\overline{v})$, and let G denote the set of *global* states, which consists of all possible valuations of shared variables $Shared(\overline{v})$.

The states of T_n are given by configurations. A *configuration* of T_n is a tuple $c \in L^n \times G$ of n local states from L and a global state in G. Index n in the tuple, denoted $c[n]$, represents the global state, shared by all the processes, and for $0 \leq i \leq n-1$, the ith index in the tuple c, denoted $c[i]$, represents the local state of the ith copy of P. We denote the set of all configurations of T_n by C_n, i.e. $C_n = L^n \times G$.

The transitions of T_n, denoted TR_n, correspond to the execution of guarded commands by the individual processes. Therefore, $TR_n = \bigcup_{i=0}^{n-1} TR_n^i$, where $TR_n^i \subseteq C_n \times C_n$ is the set of transitions of process i. Each transition of process i corresponds to the execution of a guarded command. Technically, a guarded command induces the transition $(c, c') \in TR_n^i$ if c satisfies the guard at i (as defined below), $c'[i]$ and $c'[n]$ are updated based on the command, and $c'[j] = c[j]$ for every $j \neq i$. Satisfaction of local guards by process i is defined as usual: c satisfies a local guard φ at i if $(c[i], c[n])$ satisfy φ (where $c[i]$ is an interpretation for $Local(\overline{v})[i]$ and $c[n]$ is an interpretation for $Shared(\overline{v})$). c satisfies a global guard $\mathbb{Q}j \bowtie i \,.\, \varphi$ at i if

- $\mathbb{Q} = \forall$ and for every $0 \leq j \leq n-1$ s.t. $j \bowtie i$, $(c[i], c[j], c[n])$ satisfy φ, or
- $\mathbb{Q} = \exists$ and there exists $0 \leq j \leq n-1$ s.t. $j \bowtie i$ and $(c[i], c[j], c[n])$ satisfy φ,

where $c[i]$, $c[j]$, and $c[n]$ are interpretations of $Local(\overline{v})[i]$, $Local(\overline{v})[j]$, and $Shared(\overline{v})$, respectively.

We define the set of *initial* configurations of T_n, denoted C_n^I, as the set of all configurations $c \in C_n$ such that ($c[i]$, $c[j]$, $c[n]$) satisfy φ_{init} for every $0 \le i, j \le n-1$, and similarly, we define the set of *bad* configurations, denoted C_n^B, as the set of all configurations $c \in C_n$ such that ($c[i]$, $c[j]$, $c[n]$) satisfy φ_{bad} for some $0 \le i, j \le n-1$.

Parameterized Systems as Infinite State Transition Systems. The semantics of a parameterized system defined by P is the infinite-state transition system $T(P) = (C, TR)$, where $C = \bigcup_{n \ge 1} C_n$ is the set of configurations of all instances and $TR = \bigcup_{n \ge 1} TR_n$ is the set of transitions of all instances. We lift the definition of initial and bad states to $T(P)$ similarly: $C^I = \bigcup_{n \ge 1} C_n^I$ and $C^B = \bigcup_{n \ge 1} C_n^B$. When P is clear from the context, we simply write T instead of $T(P)$.

Safety. Traces in a transition system are defined in the usual way, as sequences of states (configurations) where each two consecutive states are connected by the transition relation. A transition system (e.g., T_n or T) is *safe* if there is no trace from an initial state to a bad state. T is safe if and only if T_n is safe for every $n \ge 1$.

5. VERIFICATION CONDITIONS FOR PARAMETERIZED SYSTEMS

In this section, we provide verification conditions for verifying the safety of a parameterized system in the form of sets of Constrained Horn Clauses (CHC), such that the verification condition is satisfiable iff the system is safe. Further, a solution of the CHCs provides a witness of safety in the form of an inductive invariant.

We present two variants of the verification condition. One, called N-aware, explicitly encodes the number of processes via a logical variable. Another, called N-oblivious, ignores this number, and can be thought of as encoding a system with infinitely many processes.

In the following, we denote formulas used in the N-aware formulation by F^*, and formulas used in the N-oblivious formulation by F^ω. We write F° to denote either of them.

Logical Variables. Given a process P, to model the parameterized system using logical formulas, we introduce a logical variable for every variable $v \in \overline{v}$ and adopt the same notation for it. The variables that correspond to $v \in \mathit{Local}(\overline{v})$ are of sort `array`, and the shared variables $v \in \mathit{Shared}(\overline{v})$ inherit their sorts from the program. For the N-aware verification condition, we add to the set of variables a variable N of sort `integer` that captures the (arbitrary) number of processes. We denote by $\overline{u}$ the set of variables which consists of both $\overline{v}$ and N in the N-aware formulation, and consists of $\overline{v}$ only in the N-oblivious formulation. We introduce variables (e.g., i, j) of sort `integer` to refer to process identifiers.

Arrays conveniently represent configurations of instances of the parameterized system with any number of processes.

Transition Relation Formula. As usual, formulas representing transitions are defined over two copies of the variables, where the first (unprimed) copy refers to the state before the transition and the second copy (primed) refers to the "next" state obtained after the transition. We denote by $\overline{v}'$ the copy of $\overline{v}$ used to describe the next-state variables. A next-state copy of N is not needed, since we consider static topologies. Therefore, in the N-aware formulation, $\overline{u}'$ stands for $(\overline{v}', N)$.

role	formula
local transition	$\exists i \,.\, \varphi^\circ_{lgrd}(i, \overline{u}) \wedge \mathit{Tr}_{cmd}(i, \overline{v}, \overline{v}')$
universal transition	$\exists i \,.\, \forall j \,.\, \varphi^\circ_{ugrd}(i, j, \overline{u}) \wedge \mathit{Tr}_{cmd}(i, \overline{v}, \overline{v}')$
existential transition	$\exists i \,.\, \exists j \,.\, \varphi^\circ_{egrd}(i, j, \overline{u}) \wedge \mathit{Tr}_{cmd}(i, \overline{v}, \overline{v}')$
initial condition	$\forall i \,.\, \forall j \,.\, \mathit{Init}^\circ(i, j, \overline{u})$
error condition	$\exists i \,.\, \exists j \,.\, \mathit{Bad}^\circ(i, j, \overline{u})$

Figure 5: Quantified formulas representing transitions, initial states and bad states of parameterized systems. F° stands for either F^*, representing the N-aware formulation, in which case $\overline{u}$ stands for $(N, \overline{v})$, or for F^ω, representing the N-oblivious formulation, in which case $\overline{u}$ stands for $\overline{v}$.

notation	definition
$\varphi^\circ_{lgrd}(i, \overline{u})$	$\mathit{range}^\circ(i) \wedge \varphi_\ell(i, \overline{v})$
$\varphi^\circ_{ugrd}(i, j, \overline{u})$	$\mathit{range}^\circ(i) \wedge (\mathit{range}^\circ(j) \wedge j \bowtie i \Rightarrow \varphi_g(i, j, \overline{v}))$
$\varphi^\circ_{egrd}(i, j, \overline{u})$	$\mathit{range}^\circ(i) \wedge \mathit{range}^\circ(j) \wedge j \bowtie i \wedge \varphi_g(i, j, \overline{v})$
$\mathit{Init}^\circ(i, j, \overline{u})$	$\mathit{range}^\circ(i) \wedge \mathit{range}^\circ(j) \Rightarrow \varphi_{init}(i, j, \overline{v})$
$\mathit{Bad}^\circ(i, j, \overline{u})$	$\mathit{range}^\circ(i) \wedge \mathit{range}^\circ(j) \wedge \varphi_{bad}(i, j, \overline{v})$

Figure 6: Quantifier-free formulas used in the logical formulation of parameterized systems. In the N-aware formulation, $\mathit{range}^\circ(i) \triangleq i \in [0, N)$, while in the N-oblivious formulation, $\mathit{range}^\circ(i) \triangleq \top$.

We associate every command *cmd* with a quantifier-free transition relation formula, denoted $\mathit{Tr}_{cmd}(i, \overline{v}, \overline{v}')$. The formula Tr_{cmd}, defined over $i, \overline{v}, \overline{v}'$, captures the semantics of the command when executed by process i while all other processes are idle. That is, $\mathit{Tr}_{cmd}(i, \overline{v}, \overline{v}')$ encodes the values of $\mathit{Local}(\overline{v}')[i]$ and $\mathit{Shared}(\overline{v}')$ based on the update performed by *cmd*, and implies that for every $v \in \mathit{Local}(\overline{v})$ and for every $j \ne i$, $v'[j] = v[j]$. For example, for the command $\mathit{next}[i] := \mathit{desired}[i]; \mathit{pc}[i] := \text{WAIT}$ from Figure 1, Tr_{cmd} is

$$\mathit{next}' = \mathit{next}[i := \mathit{desired}[i]] \wedge \mathit{pc}' = \mathit{pc}[i := \text{WAIT}] \wedge \mathit{curr}' = \mathit{curr} \wedge \mathit{desired}' = \mathit{desired}$$

Note that N does not appear in these formulas since commands do not explicitly refer to N. Thus, these formulas are the same both in the N-aware and in the N-oblivious case.

From the command formulas, a transition relation formula is constructed for each guarded command of P, describing the effect of some process executing the guarded command. The formulas for both the N-aware case and the N-oblivious case are listed in Figure 5 and Figure 6. Formulas corresponding to local guarded commands have the form $\exists i \,.\, \rho(i, \overline{u}, \overline{u}')$, formulas corresponding to universal guarded commands have the form $\exists i \,.\, \forall j \,.\, \rho(i, j, \overline{u}, \overline{u}')$, and formulas corresponding to existential guarded commands have the form $\exists i \,.\, \exists j \,.\, \rho(i, j, \overline{u}, \overline{u}')$, where ρ is quantifier-free. The N-aware, respectively N-oblivious, transition relation formula of the parameterized system, denoted $\mathit{Tr}^\circ(\overline{u}, \overline{u}')$, is the disjunction of these formulas over all guarded commands of P.

Initial and Bad States Formulas. The initial and bad states formulas are shown in Figures 5 to 6. They have the form $\forall i.\forall j.\mathit{Init}^\circ(i, j, \overline{u})$ and $\exists i.\exists j.\mathit{Bad}^\circ(i, j, \overline{u})$, respectively, where $\mathit{Init}^\circ(i, j, \overline{u})$ and $\mathit{Bad}^\circ(i, j, \overline{u})$ are quantifier-free. Abusing notation, we also refer to the quantified formulas above as $\mathit{Init}^\circ(\overline{u})$ and $\mathit{Bad}^\circ(\overline{u})$, respectively.

Note that the N-oblivious formulation corresponds to removing the range constraints on id variables in Tr, $Init$ and Bad. For example, it does not require that $i \in [0, N)$ in the definition of Tr and that that $j \in [0, N)$ in the formulas for global guards. Intuitively, the N-oblivious formulas encode a transition system whose configurations consist of infinite sequences of local states. This can be thought of as letting infinitely many processes interact.

5.1 Verification Conditions for Safety

Let T be a parameterized transition system, and let $\overline{u}$, $Init^\circ(\overline{u})$, $Tr^\circ(\overline{u}, \overline{v}')$, $Bad^\circ(\overline{u})$ be defined as above, where $\circ \in \{*, \omega\}$ and $\overline{u}$ is defined accordingly.

Definition 5.1. *For $\circ \in \{*, \omega\}$, the* verification condition $VC^\circ(T)$ *for T is defined via the following set of CHCs over variables $\overline{u}, \overline{u}'$ and over the predicate $Inv(\overline{u})$.*

$$Init^\circ(\overline{u}) \Rightarrow Inv(\overline{u})$$
$$Inv(\overline{u}) \wedge Tr^\circ(\overline{u}, \overline{u}') \Rightarrow Inv(\overline{u}')$$
$$Inv(\overline{u}) \Rightarrow \neg Bad^\circ(\overline{u})$$

The N-aware verification condition, denoted $VC^*(T)$, is obtained when $\circ = *$ and $\overline{u}$ is $(\overline{v}, N)$. The N-oblivious verification condition, denoted $VC^\omega(T)$, is obtained when $\circ = \omega$ and $\overline{u} = \overline{v}$. We omit T when it is clear from the context.

Note that VC^* is defined over a set of variables which includes N, while the set of variables of VC^ω excludes N.

By splitting the disjunction in $Tr^\circ(\overline{u}, \overline{u}')$ into multiple clauses and pulling out quantifiers when possible, the verification condition $VC^\circ(T)$ can be equivalently rewritten into the set of UHCs depicted in Figure 7 (note that $Init^\circ$ and φ°_{ugrd} might contain universal quantifiers).

The following lemma shows that VC^* captures the safety of the parameterized transition system.

Lemma 5.1. *$VC^*(T)$ is satisfiable iff T is safe.*

It is tempting to claim (and is often implicitly assumed in other works) that $VC^\omega(T)$ is SAT iff T is safe. Unfortunately, this is incorrect, as demonstrated by the following examples.

Example 1 (Unsoundness). *Consider the process P from Figure 8. Initially, all copies of the process are in location I ("initial"). From location I the process makes an unconditional move to location T ("trying"). If all other processes are in T, the process moves to E ("error"). The bad states are a process in an error location. Clearly, the system is unsafe for any number of processes. However, VC^ω is* SAT. *A satisfying interpretation for Inv includes all infinite configurations where a finite prefix consists of I and T locations, while the infinite suffix (which is of course nonempty) consists of I locations, thus blocking the universal transition that leads to error. As a side note, we point out that in this example no interpretation for Inv that satisfies VC^ω is expressible by a universally quantified formula.*

Example 2 (Unsoundness). *Consider the process from Figure 9. The process has a Boolean variable b. Initially, each copy of the process nondeterministically selects a Boolean value for b. If all processes have completed their selection and in addition $\forall i, j \,.\, i \neq j \Rightarrow b[i] \neq b[j]$, then process i can move to an "error" location. If two processes reach an "error" location, a bad state is encountered. Clearly, T is unsafe since T_2 is unsafe (whereas T_n for every $N > 2$ is safe). However, VC^ω is* SAT. *Specifically, the solution is:*

$$Inv = (\forall i, j \,.\, i \neq j \Rightarrow (pc[i] \neq E \vee pc[j] \neq E)) \wedge$$
$$(\forall i \,.\, pc[i] \neq I \Rightarrow b[i] \in [0, 1]) \wedge$$
$$(\forall i, j \,.\, (pc[i] = E \wedge i \neq j) \Rightarrow (pc[j] \neq I \wedge b[i] \neq b[j])).$$

These examples show that in the general case, the N-aware and N-oblivious variants of the verification conditions do not coincide. Specifically, the N-oblivious formulation might be satisfiable even though the system is not safe.

One way to fix the unsoundness is to restrict the system.

Definition 5.2 (Weak Monotonicity). *We say that T is* weakly monotonic *if there exists a quantifier free formula $\psi(i, Local(\overline{v}))$ such that $(\forall i.\varphi_{init}(i, j, \overline{v})) \models \psi(j, Local(\overline{v}))$, and for every formula $\varphi_g(i, j, \overline{v})$ that appears in a global universal guard, $\psi(j, Local(\overline{v})) \models \varphi_g(i, j, \overline{v})$.*

Intuitively, T is weakly monotonic if no universal guard can be blocked by a process in an initial state (as witnessed by the formula $\psi(j, Local(\overline{v}))$). For example, if processes have an initial location, then $\psi(i, Local(\overline{v}))$ can state that process i is in its initial location.

Weak monotonicity ensures that adding additional processes which remain in their initial state does not disable any global universal transition. In particular, if there are no universal guards, then T is weakly monotonic. Note that the classical notion of monotonicity [1, 12] is stronger as it would require that adding additional processes in *any* local state would not disable any global universal transition, which essentially forbids universal guards altogether.

Lemma 5.2. *If T is weakly monotonic, then $VC^\omega(T)$ is satisfiable iff T is safe.*

Proof. ($\Rightarrow$): Consider a solution (model) $\mathcal{J}$ for $VC^\omega(T)$. We need to show that T is safe. Assume to the contrary that it is unsafe. Then there exists $n \in \mathbb{N}$ such that there exists an error trace in T_n. We construct a corresponding trace over infinite configurations, where each infinite configuration in the trace agrees with the corresponding finite one on the prefix of the configuration of length n, and the suffix of the infinite configuration corresponds to processes that have not moved since their initialization. The existence of such a trace contradicts the fact that $\mathcal{J}$ satisfies $VC^\omega(T)$. We construct the corresponding trace inductively. The first configuration consists of an arbitrary extension of the initial configuration that satisfies the initial condition. To extend the trace we consider the guarded command that is being executed. If it is a local or an existential command, then the same command is enabled also in the infinite configuration. If it is a universal command, then weak monotonicity ensures that it is also enabled in the infinite configuration where the additional processes remained in their initial state.

($\Leftarrow$): Suppose T is safe. To show that VC^ω is SAT it suffices to show that the system defined over infinite configurations is safe. Assume to the contrary that it is not. Then there is a (finite) error trace which consists of infinite configurations. We use it to construct an error trace for T_n for some $n \in \mathbb{N}$, in contradiction. To define the value n for which T_n is unsafe, we consider all the indices of processes that move along the trace, as well as the indices of all the witnesses to existential guarded commands. We define n to

$$\begin{aligned}
\mathit{Init}^\circ(\overline{u}) &\Rightarrow \mathit{Inv}(\overline{u}) & \\
\mathit{Inv}(\overline{u}) \wedge \varphi^\circ_{lgrd}(i,\overline{u}) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}') &\Rightarrow \mathit{Inv}(\overline{v}') & \text{for every local guraded command} \\
\mathit{Inv}(\overline{u}) \wedge (\forall j \,.\, \varphi^\circ_{ugrd}(i,j,\overline{u})) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}') &\Rightarrow \mathit{Inv}(\overline{v}') & \text{for every universal guarded command} \\
\mathit{Inv}(\overline{u}) \wedge \varphi^\circ_{egrd}(i,j,\overline{u}) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}') &\Rightarrow \mathit{Inv}(\overline{v}') & \text{for every existential guarded command} \\
\mathit{Inv}(\overline{u}) &\Rightarrow \neg \mathit{Bad}^\circ(\overline{u}) &
\end{aligned}$$

Figure 7: Verification condition for safety of a parameterized system.

def proc(i) **:**
  do
    $pc[i] = I \ : \ pc[i] := T$;
    $\forall j \neq i \,.\, pc[j] = T \ : \ pc[i] := E$
def init(i,j) **:** $pc[i] = I$;
def bad(i,j) **:** $pc[i] = E$;

Figure 8: Parameterized system demonstrating unsoundness of VC^ω.

def proc(i) **:**
  do
    $pc[i] = I \ : \ pc[i] := D; b[i] := 1$;
    $pc[i] = I \ : \ pc[i] := D; b[i] := 0$;
    $(\forall j \neq i \,.\, pc[i] = D \wedge pc[j] \neq I \wedge b[j] \neq b[i]) \ : \ pc[i] := E$;
def init(i,j) **:** $pc[i] = I$;
def bad(i,j) **:** $i \neq j \wedge pc[i] = E \wedge pc[j] = E$;

Figure 9: Parameterized system demonstrating unsoundness of VC^ω.

be the maximum over all these indices. With this choice of n, we can now consider the trace which consists of the infinite configurations truncated to configurations of length n. This is a legal trace of T_n, as every local and universal guard that was enabled in the infinite configuration is clearly enabled in its sub-configuration, and every existential guard that was enabled is still enabled since the witness process is also present in the configuration. □

COROLLARY 5.1. *Under the weak monotonicity condition, $VC^*(T)$ is satisfiable if and only if $VC^\omega(T)$ is satisfiable.*

Note that the systems described in Example 1, Example 2, and Figure 1 are *not* weakly monotonic.

6. INFERRING UNIVERSAL INVARIANTS

In this section, we present an approach for determining whether the verification condition $VC(T)$ has a model definable by a universally quantified FO-formula (or, equivalently, whether there is a safe inductive universally quantified invariant for T). More precisely, we consider models definable by *simple* universally quantified formulas:

DEFINITION 6.1. *A formula $\varphi(\overline{u})$ is a* simple k-universal *formula if $\varphi(\overline{u}) \equiv \forall i_1,\ldots,i_k \,.\, \varphi_{QF}(i_1,\ldots,i_k,\overline{u})$, where φ_{QF} is a quantifier-free formula such that the variables $i_1,\ldots,i_k$ are not used as arguments of functions in φ_{QF}.*

Our strategy for determining the existence of a simple universal invariant is to (a) fix the number of quantifiers expected in the invariant, (b) instantiate the quantifiers eagerly, and (c) use existing solvers for inference of quantifier-free safe inductive invariants to discover models of the reduced system. The two special cases of one and two-quantifiers were already discussed in Section 2. We now present the general case, prove its soundness and investigate its completeness.

Let $k \geq 1$ be the number of universal quantifiers in the (simple) universal invariant we seek. That is, we consider solutions where $\mathit{Inv}(\overline{u})$ is of the form

$$\mathit{Inv}(\overline{u}) = \forall i_1,\ldots,i_k \,.\, \mathit{Inv}_k(i_1,\ldots,i_k,\overline{u}).$$

To reduce the search for a quantified solution to a search for a quantifier-free one, we define an operator U^k, parameterized by a number k, that takes a UHC system VC (e.g., $VC = VC^*$ or $VC = VC^\omega$) and returns a CHC system $U^k(VC)$ such that $U^k(VC)$ has a quantifier free solution iff VC has a simple k-universal solution. U^k performs three transformations: *restriction*, *case splitting*, and *instantiation*, which are described next.

We describe each transformation on each individual clause of VC separately. We show the case of the consecution clauses only. The initiation and safety clauses are handled similarly (when applicable). Let *gcmd* be a guarded command with the corresponding consecution clause

$$\mathit{Inv}(\overline{u}) \wedge \rho_{gcmd}(i,\ldots) \Rightarrow \mathit{Inv}(\overline{u}) \tag{6}$$

where ρ_{gcmd} is defined as $\varphi^\circ_{lgrd}(i,\overline{u}) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}')$, $\forall j \,.\, \varphi^\circ_{ugrd}(i,j,\overline{u}) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}')$, $\varphi^\circ_{egrd}(i,j,\overline{u}) \wedge \mathit{Tr}_{cmd}(i,\overline{v},\overline{v}')$, whenever *gmcd* is local, universal, or existential, respectively (see Figure 7).

Restriction. The first transformation replaces the predicate *Inv* by a predicate Inv_k that is universally quantified with k quantifiers. Formally, the result of the restriction transformation of (6) is

$$(\forall i_1,\ldots,i_k \,.\, \mathit{Inv}_k(i_1,\ldots,i_k,\overline{u})) \wedge \rho_{gcmd}(i,\ldots) \Rightarrow \mathit{Inv}_k(i_1,\ldots,i_k,\overline{u}) \tag{7}$$

Clearly, (7) has a quantifier-free-definable model iff (6) has a model definable using k universal quantifiers.

Case Splitting. Recall that in the formula $\rho_{gcmd}(i,\ldots)$, the next-state variables $\overline{v}'$ are indexed only by i (whereas, in a global guard, the current-state $\overline{v}$ variables might also be indexed by another variable). Intuitively, this captures that local variables of a process are changed only by its own moves. The case-splitting transformation splits the consecution premise (7) for *gcmd* into $k+1$ premises. Intuitively, each new premise represents a different process making a transition and changing its state. The first k premises consider the steps of the processes that the invariant refers to (hence, they can change local and shared variables), while the $k+1$ premise considers an interference from another different *interfering* process. Note that the interfering process can only affect the shared variables.

Formally, case-splitting transforms the clause (7) into k clauses, for $1 \leq j \leq k$:

$$(\forall i_1, \ldots, i_k \, . \, Inv_k(i_1, \ldots, i_k, \overline{u})) \land \rho_{gcmd}(i_j, \ldots) \Rightarrow Inv_k(i_1, \ldots, i_k, \overline{u}') \quad (8)$$

Additionally, if *gcmd* might update a shared variable, case-splitting adds the $(k+1)$st interference clause:

$$(\forall i_1, \ldots, i_k \, . \, Inv_k(i_1, \ldots, i_k, \overline{u})) \land \rho_{gcmd}(i_{k+1}, \ldots) \land \bigwedge_{1 \leq j \leq k} i_j \neq i_{k+1} \Rightarrow Inv_k(i_1, \ldots, i_k, \overline{u}') \quad (9)$$

The case-split VC has a quantifier-free solution iff the restricted VC has a quantifier-free solution. This follows from the logical equivalence between the two sets of CHCs. In particular, whenever *gcmd* does not update any shared variables, the interference clause (9) is unnecessary. Intuitively, this means that *gcmd* can cause no interference when executed by some other process.

Case-splitting is motivated by obtaining clauses with fewer free variables. This is beneficial for the instantiation step performed next, since the set of possible instantiations depends on the free variables.

Instantiation. The restriction and case-splitting transformations result in a VC that contains universal quantifiers in the body. The instantiation transformation applies a finite eager instantiation to the remaining universal quantifiers. The result is a quantifier-free CHC VC that can be solved by existing CHC solvers.

We begin with the following definitions:

Definition 6.2 (Instantiation Elements). *Given a clause c, the set of* instantiation elements *for c, denoted $C(c)$, is the smallest set X that contains (i) the variables $i_1, \ldots, i_k$ whenever $Inv_k(i_1, \ldots, i_k, \overline{u})$ is the head of c, (ii) every term x that appears as index to some* `array` *logical variable corresponding to a local program variable in c, and (iii) every term y such that $y \bowtie x$ (or $x \bowtie y$) is in c for $x \in X$.*

Intuitively, $C(c)$ includes all variables that are used to index an array or compared with a variable already in $C(c)$. Note that in the N-aware formulation, 0 and N are in $C(c)$.

Definition 6.3 (Full Instantiations). *Let $k \in \mathbb{N}$ and let C be a finite set of instantiation elements. We define finite instantiations $FI(k, C) = C^k$ to be the set of all k-tuples constructed from C.*

Note that $FI(k, C)$ also contains tuples in which the same term appears multiple times.

Definition 6.4 (Reduced Instantiations). *Let $k \in \mathbb{N}$, C be a finite set of instantiation elements, and $\prec$ be an arbitrary total order on terms. We define reduced instantiations $RI(k, C) = \{\langle y_1, \ldots, y_k \rangle \in C^k \mid \forall i, j \, . \, i < j \Rightarrow y_i \prec y_j\}$ to be the set of all strictly $\prec$-increasing k-tuples constructed from C.*

Note that the definition of $RI(k, C)$ is parameterized by the choice of the term order $\prec$. For example, if $C = \{x, y\}$ and $x \prec y$, then $RI(2, C) = \{(x, y)\}$.

The instantiations transformation is applied on each clause c of the VC that contains universal quantifiers. All occurrences of $(\forall i_1, \ldots, i_k \, . \, Inv_k(i_1, \ldots, i_k, \overline{u}))$ are replaced by $(\bigwedge_{\overline{i} \in RI(k, C(c))} Inv_k(\overline{i}, \overline{u}))$; and all other universally quantified formulas $\forall \overline{i} \, . \, \varphi(\overline{i}, \overline{u})$ are replaced by their complete instantiations $(\bigwedge_{\overline{i} \in FI(|\overline{i}|, C(c))} \varphi(\overline{i}, \overline{u})$. The latter include (a) the transition relation formulas for universal guarded commands, where only one variable is instantiated (and therefore the full instantiations are the same as the reduced ones), and (b) the initial states formula, where pairs of variables are instantiated.

Example 3. *Consider the consecution clause* (8) *obtained from VC^ω after case splitting. If the clause corresponds to a universal guarded command, we obtain from it the following clause in $U^k(VC^\omega)$:*

$$Inv_k(i_1, \ldots, i_k, \overline{v}) \land \bigwedge_{1 \leq h \leq k} Tr_{ugcmd}(i_j, i_h, \overline{v}, \overline{v}') \Rightarrow Inv_k(i_1, \ldots, i_k, \overline{v}')$$

where $Tr_{ugcmd}(i_j, i_h, \overline{v}, \overline{v}') = (i_h \bowtie i_j \Rightarrow \varphi_g(i_j, i_h, \overline{v})) \land Tr_{cmd}(i_j, \overline{v}, \overline{v}')$. Here, reduced instantiations are used for Inv_k, resulting in one instantiation, and full instantiations are used for the transition relation formula (in fact the instantiation where $i_h = i_j$ can be removed since it simplifies to true due to the structure of $Tr_{ugcmd}(i_h, i_j, \overline{v}, \overline{v}')$). Note that the obtained clause is linear.

From a consecution clause that corresponds to an existential command, we obtain the following clause in $U^k(VC^\omega)$:

$$\bigwedge_{\overline{i} \in RI(k, C(c))} Inv_k(\overline{i}, \overline{v}) \land Tr_{egcmd}(i_j, x, \overline{v}, \overline{v}') \Rightarrow Inv_k(i_1, \ldots, i_k, \overline{v}')$$

where $C(c) = \{i_1, \ldots, i_k, x\}$, hence there are $k+1$ instantiations in $RI(k, C(c))$, and $Tr_{egcmd}(i_j, x, \overline{v}, \overline{v}') = x \bowtie i_j \land \varphi_g(i_j, x, \overline{v}) \land Tr_{cmd}(i_j, \overline{v}, \overline{v}')$. In this case, the obtained clause is non-linear.

Lemma 6.1. *The fully and reduced instantiated VCs are equi-satisfiable.*

Proof. Since the reduced instantiations are a subset of the full instantiations, one direction follows trivially.

For the other direction, consider a quantifier-free solution $M(\overline{x}, \overline{u})$ for Inv_k of the fully instantiated VC with $\overline{x} = \langle x_1, \ldots, x_k \rangle$. To transform M into a solution H of the reduced instantiated VC, we define

$$Y = \{\overline{y} = \langle y_1, \ldots, y_k \rangle \mid \forall 1 \leq i \leq k \, . \, \exists 1 \leq j \leq k \, . \, y_i = x_j\}$$

and $H(\overline{x}, \overline{u}) = \bigwedge_{\overline{y} \in Y} M(\overline{y}, \overline{u})$. That is, Y is the set of all k-tuples constructed from $x_1, \ldots, x_k$ (including tuples that have repetitions, i.e., $y_i = y_j$), and H is the reflexive symmetric closure of M. We have that $(\bigwedge_{\overline{x} \in RI(k,C)} H(\overline{x}, \overline{u})) \equiv (\bigwedge_{\overline{x} \in FI(k,C)} M(\overline{x}, \overline{u}))$.

Each conjunct of $H(\overline{x}, \overline{u})$ corresponds to $M(\overline{y}, \overline{u})$ for $\overline{y} \in Y$, whose permutations are all part of $\bigwedge_{\overline{x} \in FI(k,C)} M(\overline{x}, \overline{u})$. Therefore, each premise with the head $M(\overline{y}, \overline{u})$ and body $(\bigwedge_{\overline{x} \in RI(k,C)} H(\overline{x}, \overline{u})) \land \rho_{gcmd}(i, \ldots)$ follows by a proper renaming of the variables in $\overline{y}$. Hence, H is a solution of the reduced instantiated VC. $\square$

If the instantiated VC has a quantifier-free solution, then the original VC has a k-universal solution. However, the opposite direction is not true in general.

The reduced instantiations of Inv_k reduce an exponential number of instantiations, $|C(c)|^k = |FI(k, C(c)|$, to $\binom{|C(c)|}{k} = |RI(k, C(c)|$. In the typical case, $|C(c)| - k = O(1)$ which makes the number of reduced instantiations $|C(c)|^{O(1)}$. For example, if $C(c) = \{i_1, \ldots, i_k\}$, then the number of instantiations is just 1, resulting in *linear* constraints. This is the case for all the clauses that correspond to local guarded commands and for all the clauses that correspond to universal guarded commands in VC^ω, except for the interference clauses. The interference clauses have an additional instantiation element, resulting in $k+1$ instantiations. Clauses corresponding to existential guarded commands also have additional instantiation elements.

EXAMPLE 4. *Having presented the general reduction for k-universal invariants, we now revisit the special cases of one-quantifier and two-quantifier invariants that were presented in Section 2. The sets of constraints displayed in Figure 2 and Figure 3 correspond to $U^1(VC^\omega)$ and $U^2(VC^\omega)$ respectively, for the case where all transitions in T have local guards (but may update shared variables). Universal and existential guards would be handled by $U(VC^\omega)$ as demonstrated in Example 3. For example, for a universal guarded command, the constraint* (3) *in Figure 3 would become*

$$Inv_2(i, j, \overline{v}) \wedge Tr(i, j, \overline{v}, \overline{v}') \Rightarrow Inv_2(i, j, \overline{v}').$$

For an existential command, constraint (3) *would become*

$$Inv_2(i, j, \overline{v}) \wedge Inv_2(j, k, \overline{v}) \wedge Inv_2(i, k, \overline{v}) \wedge Tr(i, k, \overline{v}, \overline{v}') \Rightarrow Inv_2(i, j, \overline{v}').$$

In the rest of the section, we establish the soundness and relative completeness of the transformation $U^k(VC)$.

Soundness. The soundness of the transformation $U^k(VC)$ follows directly from the soundness of each of the steps. This claim is robust — it applies both to the N-aware VC^* as well as to the N-oblivious VC^ω.

LEMMA 6.2. *Let VC be one of VC^* or VC^ω. If $U^k(VC)$ has a quantifier-free solution, then VC has a solution with k universal quantifiers.*

The proof of Lemma 6.2 is straightforward: If $H(\bar{i}, \overline{u})$ is a solution of $U^k(VC)$, then $\forall \bar{i} \,.\, H(\bar{i}, \overline{u})$ is a solution for VC.

In the rest of this section, we show one of our main results — under the transformation U^k, N-oblivious VC^ω soundly approximates N-aware VC^*. Thus, in many cases a simpler formulation VC^ω (i.e., fewer instantiations) is sufficient.

LEMMA 6.3. *If $U^k(VC^\omega)$ has a quantifier-free solution, then $U^k(VC^*)$ has a quantifier-free solution.*

PROOF. The proof is constructive. Given a quantifier-free solution $H(i_1, \ldots, i_k, \overline{v})$ for $Inv_k(i_1, \ldots, i_k, \overline{v})$ in $U^k(VC^\omega)$, we construct a quantifier-free solution $S(i_1, \ldots, i_k, N, \overline{v})$ for $Inv_k(i_1, \ldots, i_k, N, \overline{v})$ in $U^k(VC^*)$. S is defined as follows:

$$S = (\bigwedge_{j=1}^{k} i_j \in [0, N)) \Rightarrow H.$$

That is, each identifier variable i_j ($1 \leq j \leq k$) is restricted to the legal range of process identifiers.

To show that S is a solution to $U^k(VC^*)$, we need to show that it satisfies all clauses. We prove the claim for the consecution premises (the proof of the other clauses is similar). Consider a consecution premise c in $U^k(VC^*)$ with head $Inv_k(i_1, \ldots, i_k, N, \overline{v})$ and let σ be an assignment that satisfies the body. If the value of at least one of $i_1, \ldots, i_k$ in σ is not in the range $[0, N)$, then σ satisfies c vacuously. Assume all index variables are in range, and let c' be the corresponding premise from $U^k(VC^\omega)$. We consider two cases: (a) c corresponds to a local or existential guarded command. Then any additional range constraints in its body are conjoined with the body. Since the body is satisfied by σ, they all hold. Therefore, under σ, c simplifies to c' and the claim follows; (b) c corresponds to a universal guarded command (this is the interesting case that was the source of unsoundness before). Here, the body of c also contains conjuncts of the form $(j \in [0, N) \wedge j \bowtie i \Rightarrow \varphi_g(i, j, \overline{v}))$, where the range conditions are antecedents in implications. However, since in each such implication j is an instantiation element (see Definition 6.2), its range condition also appears as a conjunct in the body, and by our assumption that the body is satisfied by σ, so does the range condition. (Further, the instantiations where j is 0 or N do not exist in c'). Therefore, under σ, again, c simplifies to c' and the claim follows. □

Lemma 6.3 implies the following corollary that establishes that VC^ω approximates VC^* under the U^k transformation.

COROLLARY 6.1. *If $U^k(VC^\omega)$ has a quantifier-free solution, then VC^* has a k-universally quantified solution.*

This allows us to verify the example in Figure 1 with VC^ω, even though it is not weakly monotonic.

Relative completeness. While the soundness of our approach does not rely on the restriction to *simple* universal invariants, completeness does.

LEMMA 6.4. *If VC^* has a k-universal simple solution, then $U^k(VC^*)$ has a quantifier-free solution.*

The proof of this lemma follows from the fact that the syntax of processes ensures that *Init*, *Tr* and *Bad* do not use identifier variables as arguments to functions. This and the restriction to simple invariants is sufficient to guarantee that our set of instantiations is complete.

7. SAFETY VERIFICATION

Safety verification of parameterized systems is in general undecidable. In this section, we extend the technique of Section 6 to a semi-algorithm for the problem (i.e., our procedure might not terminate). We show that for certain classes of systems (e.g., Petri nets), our procedure is in fact a decision procedure.

Our procedure is shown in Algorithm 1. It combines a search for a simple k-universal safe inductive invariant with a search for a counterexample in a k-process instance of the system. Initially, k is 1 and it is increased in each iteration. The search for a simple k-universal invariant is done as described in Section 6 based on VC^ω. The search for a counterexample is done by an off-the-shelf model checker. In practice, we use CHC solver for both steps.

Termination. Algorithm 1 might not terminate due to several reasons. First, solving the CHCs generated by U^k might

$k := 1$;
while *true* **do**
  $Inv_k(i_1, \ldots, i_k, \overline{v}) := \text{Solve}(U^k(VC^\omega(T)))$;
  if $Inv_k(i_1, \ldots, i_k, \overline{v}) \neq null$ **then**
    return *"inductive invariant found:* $\forall i_1, \ldots, i_k \,.\, Inv(i_1, \ldots, i_k, \overline{v})$*"*
  $res := \text{ModelCheck}(T_k)$;
  if $res = cex$ **then**
    return *"counterexample found for k processes"*
  $k := k + 1$

Algorithm 1: Procedure for safety verification of parameterized systems.

be undecidable. Second, even safety verification of a single instance of the system might be undecidable. These issues are alleviated if we restrict ourselves to finite state processes.

However, even if both inner steps of Algorithm 1 are decidable, the procedure might not terminate. The reason is that it is possible that a parameterized system is safe (i.e., no counterexample exists in any instance), but it has no simple universal invariant. In this case, both searches performed by Algorithm 1 diverge. Fortunately, there are classes of systems for which this phenomenon is impossible. In such cases, we obtain a decision procedure for safety verification.

An example of such a class of systems is when the parameterized system forms a *monotonic transition system* with respect to the subsequence ordering over configurations [1]. In particular, parameterized systems with local and existential transitions (but no universal transitions) meet the monotonicity condition [2]. An important subclass of such systems is Petri nets.

8. IMPLEMENTATION AND EVALUATION

We have implemented a prototype of our technique in Python. The input is a description of a parametric system as a collection of guarded commands. The output is an inductive invariant or a counterexample. Our input language is similar to the input language of CUBICLE [9] and is more liberal than the syntax of Figure 4 used for the formal presentation in the paper. Given a parametric system, our implementation computes a verification condition as a set of UHCs, reduces it to CHCs using the transformations described in Section 6, and solves them using SPACER [19].

We have experimented with several small but challenging protocols including the collision avoidance protocol from Figure 1, dining philosophers, and several variants of Lamport's bakery mutual exclusion algorithm. In all these cases, we were able to successfully compute a 2-universal inductive invariant in a few seconds. Further work is necessary to tune our implementation for more challenging benchmarks.

9. RELATED WORK

There is a large body of work on verification of concurrent, distributed, and parameterized systems. We refer the reader to [5] for a recent survey. Below, we compare our approach only with the most closely related work.

Our technique can be seen as an adaptation of the *view abstraction* of Abdulla et al. [2] (and, in this, similar to dynamic cut-off of Kroening et al. [17], invisible invariants of Zuck et al. [21], and environment abstraction of Clarke et al. [8]). When our procedure converges, the constructed invariant is a view abstraction of the original system, where the size of the view is determined by the number of quantifiers. However, the details of the technique are very different. In particular, our approach naturally extends to infinite-state processes, such as processes with integer local and shared variables without the need for a separate finite-state abstraction. While such variables might not appear directly in the protocol, they are convenient for verification purposes. For example, it is common to abstract the number of processes in a particular state by an integer counter, which introduces a shared integer variable in the model.

Our reduction from universal Horn clauses to CHC is inspired by the eager quantifier instantiation of Bjørner et al. [4]. Our key insight is in the use of problem-specific restrictions to dramatically reduce the number of the necessary instantiations, while maintaining completeness. In particular, we require exponentially fewer instantiations than [4]. In that, our formulation is similar to that of Hojjat et al. [16].

In the paper, we reduce verification of parameterized systems to CHC satisfiability and leave the choice of the CHC solver open. In the evaluation, the resulting CHCs are solved by a CHC solver SPACER [19, 18]. This combination is similar to the MCMT [14] approach, and, in particular, to the BARB algorithm [10] of CUBICLE [9]. The key similarities and differences are highlighted below. SPACER is a CHC solver based on the generalization of the IC3 model checking algorithm [6]. Similar to BARB, it combines abstract backward and concrete forward reachability computations. Unlike BARB, the backward reachability is under-approximated using Model Based Projection [19] and is generalized using interpolation and over-approximation of forward reachable states [3, 15]. This makes our algorithm less susceptible to the syntactic description of the model under analysis. For example, generalizations are not restricted to the predicates that are obtained by quantifier elimination in the computation of backward reachability. As an added bonus, when the problem is safe, SPACER always produces an easy-to-validate certificate. On the other hand, CUBICLE requires a non-trivial certificate generation procedure [11]. So far, we have used SPACER as a black-box. We leave exploring the many possible optimizations of the SPACER algorithm for this domain for future work.

10. CONCLUSION

In this paper, we present an SMT-based approach for verifying parameterized systems – systems consisting of asynchronous composition of N identical processes. We show that a verification condition of a parameterized system is captured by a first-order formula as a conjunction of constrained Horn clauses with constraints in the combined theory of Linear Integer Arithmetic and (quantified) theory of Arrays. Since satisfiability of such CHCs is undecidable, we develop a technique for inferring universally quantified solutions. Our approach yields a novel procedure for automated verification of parameterized system using existing SMT-based CHC solvers such as SPACER [19]. Interestingly, the constraints, that we derive systematically, match and extend the well-known Owicki-Gries proof rules to the parameterized setting. While the paper is focused on asynchronous composition of identical process, the main ideas extend to process groups and to synchronous composition.

11. REFERENCES

[1] P. A. Abdulla, K. Cerans, B. Jonsson, and Y. Tsay. General decidability theorems for infinite-state systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 313–321, 1996.

[2] P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 476–495, 2013.

[3] N. Bjørner and A. Gurfinkel. Property directed polyhedral abstraction. In *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, pages 263–281, 2015.

[4] N. Bjørner, K. L. McMillan, and A. Rybalchenko. On Solving Universally Quantified Horn Clauses. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, pages 105–125, 2013.

[5] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

[6] A. R. Bradley. SAT-Based Model Checking without Unrolling. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, pages 70–87, 2011.

[7] S. Chaki and J. R. Edmondson. Model-Driven Verifying Compilation of Synchronous Distributed Applications. In *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings*, pages 201–217, 2014.

[8] E. M. Clarke, M. Talupur, and H. Veith. Environment Abstraction for Parameterized Verification. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings*, pages 126–141, 2006.

[9] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A Parallel SMT-Based Model Checker for Parameterized Systems - Tool Paper. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 718–724, 2012.

[10] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Invariants for finite instances and beyond. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 61–68, 2013.

[11] S. Conchon, A. Mebsout, and F. Zaïdi. Certificates for parameterized model checking. In *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, pages 126–142, 2015.

[12] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[13] R. W. Floyd. Assigning meanings to programs. In *Proceedings of Symposium on Applied Mathematics*, number 32, 1967.

[14] S. Ghilardi and S. Ranise. MCMT: A model checker modulo theories. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, pages 22–29, 2010.

[15] K. Hoder and N. Bjørner. Generalized property directed reachability. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 157–171, 2012.

[16] H. Hojjat, P. Rümmer, P. Subotic, and W. Yi. Horn Clauses for Communicating Timed Systems. In *Proceedings First Workshop on Horn Clauses for Verification and Synthesis, HCVS 2014, Vienna, Austria, 17 July 2014.*, pages 39–52, 2014.

[17] A. Kaiser, D. Kroening, and T. Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 645–659, 2010.

[18] A. Komuravelli, N. Bjørner, A. Gurfinkel, and K. L. McMillan. Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, pages 89–96, 2015.

[19] A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-Based Model Checking for Recursive Programs. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 17–34, 2014.

[20] S. Owicki and D. Gries. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM*, 19(5):279–285, May 1976.

[21] A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, pages 82–97, 2001.