SMT-Based Verification of Parameterized Systems

Arie Gurfinkel, Sharon Shoham, and Yuri Mesham
Parameterized Verification

System(s)
$P_1, P_2, \ldots, P_N$

Property
$\phi$

Parameterized Model Checker

$\forall i. \; P_i \not\models \phi$

$\neg (\forall i. \; P_i \models \phi)$
Our Contributions

Verification Condition for Safety Properties of Parameterized Systems
• expressed as Constrained Horn Clauses (CHC)
• infinite state transition relation expressed in SMT (arithmetic + arrays)
• invariants/models expressed as \((\text{universally})\) quantified first order formulae

Proof-rules for inferring Universally Quantified Invariants
• generalizes Owicki-Gries proof scheme to \(k\) processes
• systematically derived from the VC above

Decision procedure for specialized finite-data sub-classes

Prototype implementation in Python
• verification of small (in code) infinite state protocols with no finite state abs.
  – bakery, ticket, dining philosophers,…
SYMBOLIC REACHABILITY
Symbolic Reachability Problem

\( P = (V, \text{Init}, \text{Tr}, \text{Bad}) \)

\( P \) is UNSAFE if and only if there exists a number \( N \) s.t.

\[
\text{Init}(X_0) \land \left( \bigwedge_{i=0}^{N-1} \text{Tr}(X_i, X_{i+1}) \right) \land \text{Bad}(X_N) \not\models \bot
\]

\( P \) is SAFE if and only if there exists a safe inductive invariant \( \text{Inv} \) s.t.

\[
\begin{align*}
\text{Init} & \Rightarrow \text{Inv} \\
\text{Inv}(X) \land \text{Tr}(X, X') & \Rightarrow \text{Inv}(X') \\
\text{Inv} & \Rightarrow \neg \text{Bad}
\end{align*}
\]
Constrained Horn Clauses (CHC)

A Constrained Horn Clause (CHC) is a FOL formula of the forms

\[ \forall V . (\phi \land p_1[X_1] \land \ldots \land p_n[X_n] \rightarrow p_{n+1}[X]) \]

\[ \forall V . (\phi \land p_1[X_1] \land \ldots \land p_n[X_n] \rightarrow \text{false}) \]

where

- \( \phi \) is a constraint in a background theory \( A \)
  - usually, \( A \) is the combined theory of Linear Arithmetic, Arrays, Bit-Vectors, ...
- \( p_1, \ldots, p_{n+1} \) are n-ary predicates
- \( p_i[X] \) is an application of a predicate to first-order terms
Spacer: Solving SMT-constrained CHC

Spacer: a solver for SMT-constrained Horn Clauses
- stand-alone implementation in a fork of Z3
- [http://bitbucket.org/spacer/code](http://bitbucket.org/spacer/code)

Support for Non-Linear CHC
- model procedure summaries in inter-procedural verification conditions
- model assume-guarantee reasoning
- uses MBP to under-approximate models for finite unfoldings of predicates
- uses MAX-SAT to decide on an unfolding strategy

Supported SMT-Theories
- Best-effort support for arbitrary SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic
- Full support for Linear arithmetic (rational and integer)
- Quantifier-free theory of arrays
  - only quantifier free models with limited applications of array equality
PARAMETRIZED SYMBOLIC REACHABILITY
Motivation: Collision Avoidance Protocol

local

\( pc : \{ \text{CHOOSE, TRY, WAIT, MOVE}\} \);  
\( curr, next, desired : \text{Location} \)

def proc(i):
  do
  \( pc[i] = \text{CHOOSE} : desired[i] \leftarrow * ; pc[i] \leftarrow \text{TRY} \);
  \( pc[i] = \text{TRY} \land \forall j . i < j \Rightarrow curr[j] \neq desired[i] \land next[j] \neq desired[i] \):
    next[i] \leftarrow desired[i] ; pc[i] \leftarrow \text{WAIT} ;
  \( pc[i] = \text{WAIT} \land \forall j . j < i \Rightarrow next[i] \neq curr[j] \land next[i] \neq next[j] \):
    pc[i] \leftarrow \text{MOVE} ;
  \( pc[i] = \text{MOVE} : \)
    curr[i] \leftarrow next[i] ; pc[i] \leftarrow \text{CHOOSE} ;

def init(i, j):
  \( pc[i] = \text{CHOOSE} \land curr[i] = next[i] \land (i \neq j \Rightarrow curr[i] \neq curr[j]) \)

def bad(i, j):
  \( i \neq j \land curr[i] = curr[j] \)
Parameterized Symbolic Reachability Problem

\[ T = ( \mathbf{v}, \text{Init}(N, \mathbf{v}), \text{Tr}(i, N, \mathbf{v}, \mathbf{v}'), \text{Bad}(N, \mathbf{v}) ) \]

- \( \mathbf{v} \) is a set of state variables
  - each \( v_k \in \mathbf{v} \) is a map \( \text{Nat} \to \text{Rat} \)
  - \( \mathbf{v} \) is partitioned into \( \text{Local}(\mathbf{v}) \) and \( \text{Global}(\mathbf{v}) \)
- \( \text{Init}(N, \mathbf{v}) \) and \( \text{Bad}(N, \mathbf{v}) \) are initial and bad states, respectively
- \( \text{Tr}(i, N, \mathbf{v}, \mathbf{v}') \) is a transition relation, parameterized by a process identifier \( i \) and total number of processes \( N \)

All formulas are over the combined theories of arrays and LRA

\( \text{Init}(N, \mathbf{v}) \) and \( \text{Bad}(N, \mathbf{v}) \) contain at most 2 quantifiers

- \( \text{Init}(N, \mathbf{v}) = \forall x, y . \varphi_{\text{Init}}(N, x, y, \mathbf{v}) \), where \( \varphi_{\text{Init}} \) is quantifier free (QF)
- \( \text{Bad}(N, \mathbf{v}) = \forall x, y . \varphi_{\text{Bad}}(N, x, y, \mathbf{v}) \), where \( \varphi_{\text{Bad}} \) is QF

\( \text{Tr} \) contains at most 1 quantifier

- \( \text{Tr}(i, N, \mathbf{v}, \mathbf{v}') = \forall j . \rho(i, j, N, \mathbf{v}, \mathbf{v}') \)
A State of a Parameterized System

<table>
<thead>
<tr>
<th>Global</th>
<th>PID</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0</td>
<td>V4</td>
<td>V5</td>
</tr>
<tr>
<td>V1</td>
<td>V6</td>
<td>V7</td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Global variables* $v_0$ to $v_3$ and *local variables* $v_4$ to $v_7$ are shown for each PID level from 0 to $N$. The table demonstrates the state variables across the system.
Parameterized Symbolic Reachability

\[ T = (v, \text{Init}, Tr, Bad) \]

\( T \) is UNSAFE if and only if there exists a number \( K \) s.t.

\[ \text{Init}(v_0) \land \left( \bigwedge_{s \in [0, K]} Tr(i_s, N, v_s, v_{s+1}) \right) \land \text{Bad}(v_K) \not\rightarrow \bot \]

\( T \) is SAFE if and only if there exists a safe inductive invariant \( Inv \) s.t.

\[ \begin{align*}
\text{Init}(v) & \Rightarrow Inv(v) \\
Inv(v) \land Tr(i, N, v, v') & \Rightarrow Inv(v') \\
Inv(v) & \Rightarrow \neg Bad(v)
\end{align*} \]  

\[ \text{VC}(T) \]
Parameterized vs Non-Parameterized Reachability

\[ \text{Init}(\mathbf{v}) \Rightarrow \text{Inv}(\mathbf{v}) \]
\[ \text{Inv}(\mathbf{v}) \land \text{Tr}(i, N, \mathbf{v}, \mathbf{v'}) \Rightarrow \text{Inv}(\mathbf{v'}) \]
\[ \text{Inv}(\mathbf{v}) \Rightarrow \neg \text{Bad}(\mathbf{v}) \]

Init, Bad, and Tr might contain quantifiers
- e.g., “ALL processes start in unique locations”
- e.g., “only make a step if ALL other processes are ok”
- e.g., “EXIST two distinct process in a critical section”

Inv cannot be assumed to be quantifier free
- QF Inv is either non-parametric or trivial

Decide existence of quantified solution for CHC
- stratify search by the number of quantifiers
- solutions with 1 quantifier, 2 quantifiers, 3 quantifiers, etc…
ONE QUANTIFIER
TWO QUANTIFIER

One fish by Dr. Seuss

Two fish

Red fish

Blue fish
One Quantifier (Solution)

\[
\begin{align*}
\text{Init}(i, i, v) & \implies \text{Inv}_1(i, v) \\
\text{Inv}_1(i, v) \land \text{Tr}(i, v, v') & \implies \text{Inv}_1(i, v') \\
\frac{}{j \neq i \land \text{Inv}_1(i, v) \land \text{Inv}_1(j, v) \land \text{Tr}(j, v, v') & \implies \text{Inv}_1(i, v')} \\
\text{Inv}_1(i, v) \land \text{Inv}_1(j, v) & \implies \neg \text{Bad}(i, j, v)
\end{align*}
\]

Theorem

- If VC$_1$(T) is QF-SAT then VC(T) is SAT
- If Tr does not contain functions that range over PID$s$, then VC$_1$(T) is QF-SAT only if VC(T) admits a solution definable by a simple single quantifier formula
  - simple $\iff$ quantified id variables do not appear as arguments to functions

VC$_1$(T) is essentially Owicki-Gries for 2 processes $i$ and $j$

If there are no global variables then (3) is unnecessary

- VC$_1$(T) is linear
How do we get it

1. Restrict Inv to a fixed number of quantifiers
   • e.g., replace Inv(N, v) with $\forall k.\text{Inv}_1(k, N, v)$

2. Case split consecution Horn clause based on the process that makes the move
   • $w+1$ cases for $w$-quantifiers
     – one for each quantified id variable
     – one for interference by “other” process (only for global variables)

3. Instantiate the universal quantifier in $\forall k.\text{Inv}_1(k, N, v)$
   • use symmetry to reduce the space of instantiations

4. Other instantiations might be needed for quantifiers if
   • id variables appear as arguments to functions
How do we get it

\[ Inv(v) \land Tr(j, v, v') \implies Inv(v') \]

\[(\forall k \cdot Inv_1(k, v)) \land Tr(j, v, v') \implies Inv_1(i, v') \]

\[(\forall k \cdot Inv_1(k, v)) \land Tr(i, v, v') \implies Inv_1(i, v') \]

\[(\forall k \cdot Inv_1(k, v)) \land j \neq i \land Tr(j, v, v') \implies Inv_1(i, v') \]

\[ Inv_1(i, v) \land Tr(i, v, v') \implies Inv_1(i, v') \]

\[ Inv_1(i, v) \land Inv_1(j, v) \land j \neq i \land Tr(j, v, v') \implies Inv_1(i, v') \]
Two Quantifier Solution

\[ Init(i, j, v) \land Init(j, i, v) \land Init(i, i, v) \land Init(j, j, v) \Rightarrow I_2(i, j, v) \]
\[ I_2(i, j, v) \land Tr(i, v, v') \Rightarrow I_2(i, j, v') \]
\[ I_2(i, j, v) \land Tr(j, v, v') \Rightarrow I_2(i, j, v') \]
\[ I_2(i, j, v) \land I_2(i, z, v) \land I_2(j, z, v) \land Tr(z, v, v') \land z \neq i \land z \neq j \Rightarrow I_2(i, j, v') \]
\[ I_2(i, j, v) \Rightarrow \neg Bad(i, j, v) \]

Theorem

• If \( VC_2(T) \) is QF-SAT then \( VC(T) \) is SAT
• If \( Tr \) does not contain functions that range over PIDs, then \( VC_2(T) \) is QF-SAT only if \( VC(T) \) admits a solution definable by a \textit{simple} two quantifier formula

• At least 2 quantifiers are “needed” for systems with global guards

Extends to \( k \)-quantifiers
Ticket Protocol

INIT: pc=Thinking, ticket=0, serving=0
BAD: self.pc=Eating, other.pc=Eating
(define-fun Inv ((id0 Int) (id1 Int) (pc (Array Int Int)) (t (Array Int Int)) (serving Int) (ticket Int)) Bool
(let ((a!1 (<= (+ (select t id1) (* (- 1) (select t id0))) (- 1))))
  (a!2 (<= (+ (select t id0) (* (- 1) (select t id1))) (- 1))))
  (a!3 (or (<= (select pc id1) 1) (<= (+ serving (* (- 1) ticket)) (- 1))))
  (a!4 (or (<= (select pc id0) 0)
    (<= (+ (select t id0) (* (- 1) ticket)) (- 1))))))

∀i, j, i ≠ j ⇒ (i.pc ≠ E) ∨ (j.pc ≠ E)
∀i, i.pc = E ⇒ serving < ticket
∀i, i.pc ∈ {H, E} ⇒ i.t < ticket
∀i, j, i ≠ j ⇒ (i.pc ∈ {H, E} ∧ j.pc ∈ {H, E} ⇒ i.t ≠ j.t)
∀i, j, i ≠ j ⇒ (i.pc = E ∧ j.pc ∈ {H, E} ⇒ j.t ≠ serving)

(or (<= (select pc id1) 0) (<= (select pc id0) 0) a!1 a!2)
a!3
a!4
a!5
a!6
(or a!7 (<= (select pc id1) 1) a!8 (<= (select pc id0) 0)) a!10))
Putting it all together

$k := 1$

while true do

$Inv_k(i_1, \ldots, i_k, v) := \text{Solve}(U^k(VC^{\omega}(T)))$

if $Inv_k(i_1, \ldots, i_k, v) \neq \text{null}$ then

return "inductive invariant found: $\forall i_1, \ldots, i_k . Inv(i_1, \ldots, i_k, v)$"

res := ModelCheck($T_k$)

if res = cex then

return "counterexample found for $k$ processes"

$k := k + 1$
Finite vs Infinite Number of Processes

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 \land pc[j] \neq I \land b[j] \neq b[i]) : pc[i] := E;
    def init(i, j) : pc[i] = I;
    def bad(i, j) : i \neq j \land pc[i] = E \land pc[j] = E;

Tr does not depend on N (number of processes)
Safe for infinitely many processes. Invariant is:

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

Unsafe for N = 2!
Evaluation and Implementation

Python-based Implementation

- Simple language for specifying concurrent protocols
- Local and Universally guarded transitions
- Constraints over arrays and integer arithmetic
- Reduce to CHC using the rules and solve using Spacer

Evaluated on Simple/Tricky Well-Know Protocols

- Dining philosophers, bakery1, bakery2, collision avoidance, ticket
- Models are pretty close to an implementation
  - limit abstraction in modeling, try to make verification hard
- Safe inductive invariants computed within seconds
Related Work

Kedar Namjoshi et al.
- Local Proofs for Global Safety Properties, and many other papers
- systematic derivation of proof rules for concurrent systems
- finite state and fixed number of processes

Andrey Rybalchenko et al.
- Compositional Verification of Multi-Threaded Programs, and others
- compositional proof rules for concurrent systems are CHC
- infinite state and fixed number of processes

Lenore Zuck et al.
- Invisible Invariants
- finite state and parametric number of processes
- finite model theorem for special classes of parametric systems

Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko
- On Solving Universally Quantified Horn Clauses. SAS 2013:
Conclusion

Parameterized Verification == Quantified solutions for CHC

Quantifier instantiation to *systematically* derive proof rules for verification of safety properties of parameterized systems

- Parameterized systems definable with SMT-LIB syntax

Lazy vs Eager Quantifier Instantiation

- eager instantiation in this talk
- would be good to extend to lazy / dynamic / model-based instantiation

Connections with other work in parameterized verification

- complete instantiation = decidability ?
- relative completeness
- …
Did I Ever Tell You How Lucky You Are?

By Dr. Seuss