Algorithmic Logic-Based Verification

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Arie Gurfinkel
SYMBOLIC REACHABILITY
Symbolic Reachability Problem

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

\[ P \text{ is UNSAFE if and only if there exists a number } N \text{ s.t.} \]
\[ \text{Init}(X_0) \land \left( \bigwedge_{i=0}^{N-1} \text{Tr}(X_i, X_{i+1}) \right) \land \text{Bad}(X_N) \not\Rightarrow \bot \]

\[ P \text{ is SAFE if and only if there exists a safe inductive invariant } \text{Inv} \text{ s.t.} \]
\[ \begin{align*}
    \text{Init} & \Rightarrow \text{Inv} \\
    \text{Inv}(X) \land \text{Tr}(X, X') & \Rightarrow \text{Inv}(X') \\
    \text{Inv} & \Rightarrow \neg \text{Bad}
\end{align*} \]
Constrained Horn Clauses (CHC)

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

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

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

where

- $\phi$ is a constrained in a background theory $A$
  - of 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
- \url{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
IC3, PDR, and Friends (1)

IC3: A SAT-based Hardware Model Checker
• Incremental Construction of Inductive Clauses for Indubitable Correctness
• A. Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011

PDR: Explained and extended the implementation
• Property Directed Reachability
• N. Eén, A. Mishchenko, R. K. Brayton: Efficient implementation of property directed reachability. FMCAD 2011

PDR with Predicate Abstraction (easy extension of IC3/PDR to SMT)
• A. Cimatti, A. Griggio, S. Mover, St. Tonetta: IC3 Modulo Theories via Implicit Predicate Abstraction. TACAS 2014
• J. Birgmeier, A. Bradley, G. Weissenbacher: Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR). CAV 2014
IC3, PDR, and Friends (2)

GPDR: Non-Linear CHC with Arithmetic constraints
• Generalized Property Directed Reachability
• K. Hoder and N. Bjørner: Generalized Property Directed Reachability. SAT 2012

SPACER: Non-Linear CHC with Arithmetic
• fixes an incompleteness issue in GPDR and extends it with under-approximate summaries
• A. Komuravelli, A. Gurfinkel, S. Chaki: SMT-Based Model Checking for Recursive Programs. CAV 2014

PolyPDR: Convex models for Linear CHC
• simulating Numeric Abstract Interpretation with PDR
• N. Bjørner and A. Gurfinkel: Property Directed Polyhedral Abstraction. VMCAI 2015

ArrayPDR: CHC with constraints over Arithmetic + Arrays
• Required to model heap manipulating programs
• A. Komuravelli, N. Bjørner, A. Gurfinkel, K. L. McMillan: Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays. FMCAD 2015
Spacer In Pictures

\[ x = 3, y = 0 \]
\[ x = 1, y = 0 \]
\[ x \neq 3 \lor y \neq 0 \]
\[ x > y \]
Logic-based Algorithmic Verification

- Simulink
- Java
- C/C++
- concurrent/distributed systems

Spacer

CoCoSim

Lustre

Zustre

Termination for C

T2

JayHorn

SeaHorn

CPR

Software Engineering Institute | Carnegie Mellon University
SeaHorn

A fully automated verification framework for LLVM-based languages.

http://seahorn.github.io
SeaHorn Usage

Example: in test.c, check that \textit{x is always greater than or equal to y}.

```c
extern int nd();
extern void __VERIFIER_error() __attribute__((noreturn));
void assert (int cond) { if (!cond) __VERIFIER_error (); } 
int main(){
    int x,y;
    x=1; y=0;
    while (nd ())
    {
        x=x+y;
        y++;
    }
    \textbf{assert (x>=y)};
    return 0;
}
```

SeaHorn command:
```
-> sea_pf test.c
```

SeaHorn result:
```
-------------
SEAHORN
-------------
PROPERTY (line 12) | TRUE
-------------
TIME(ms) | 0.06
---------
```
SeaHorn Encoding of Verification Conditions

int x = 1;
int y = 0;
while (*) {
    x = x + y;
y = y + 1;
}
assert(x ≥ y);

⟨1⟩ p₀.
⟨2⟩ p₁(x, y) ← p₀, x = 1, y = 0.
⟨3⟩ p₂(x, y) ← p₁(x, y).
⟨4⟩ p₃(x, y) ← p₁(x, y).
⟨5⟩ p₁(x', y') ← p₂(x, y),
    x' = x + y,
y' = y + 1.
⟨6⟩ p₄ ← (x ≥ y), p₃(x, y).
⟨7⟩ p₆ ← (x < y), p₃(x, y).
⟨8⟩ p₄ ← p₄.
⟨9⟩ ⊥ ← p₆.
PARAMETRIZED SYMBOLIC REACHABILITY

joint work with Sharon Shoham
What we want to do …

local
\[ pc : \{ \text{CHOOSE}, \text{TRY}, \text{WAIT}, \text{MOVE} \} ; \]
\[ curr, next, desired : \text{Location} \]
def proc(i):
    do
        \[ pc[i] = \text{CHOOSE} : \text{desired}[i] \leftarrow * ; pc[i] \leftarrow \text{TRY}; \]
        \[ pc[i] = \text{TRY} \land \forall j. i < j \Rightarrow \text{curr}[j] \neq \text{desired}[i] \land \text{next}[j] \neq \text{desired}[i] \]
        \[ next[i] \leftarrow \text{desired}[i] ; pc[i] \leftarrow \text{WAIT} ; \]
        \[ pc[i] = \text{WAIT} \land \forall j. j < i \Rightarrow \text{next}[i] \neq \text{curr}[j] \land \text{next}[i] \neq \text{next}[j] \]
        \[ pc[i] \leftarrow \text{MOVE} ; \]
        \[ pc[i] = \text{MOVE} : \]
        \[ \text{curr}[i] \leftarrow \text{next}[i] ; pc[i] \leftarrow \text{CHOOSE}; \]
def init(i, j):
    \[ pc[i] = \text{CHOOSE} \land \text{curr}[i] = \text{next}[i] \land (i \neq j \Rightarrow \text{curr}[i] \neq \text{curr}[j]) \]
def bad(i, j):
    \[ i \neq j \land \text{curr}[i] = \text{curr}[j] \]
Parameterized Symbolic Reachability Problem

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

- \( \mathbf{v} \) is a set of state variables
  - each \( v_k \in \mathbf{v} \) is a map \( \text{Nat} \rightarrow \text{Rat} \)
  - \( \mathbf{v} \) 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
- \( Tr(i, N, \mathbf{v}, \mathbf{v}') \) 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
- \( \text{Init}(N,\mathbf{v}) = \forall x,y. \phi_{\text{Init}}(N, x, y, \mathbf{v}) \), where \( \phi_{\text{Init}} \) is quantifier free (QF)
- \( \text{Bad}(N,\mathbf{v}) = \forall x,y. \phi_{\text{Bad}}(N, x, y, \mathbf{v}) \), where \( \phi_{\text{Bad}} \) is QF

\( Tr \) contains at most 1 quantifier
- \( Tr(i, N, \mathbf{v}, \mathbf{v}') = \forall j. \rho(i, j, N, \mathbf{v}, \mathbf{v}') \)
A State of a Parameterized System

<table>
<thead>
<tr>
<th>Global</th>
<th>PID</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V₄</td>
<td>V₅</td>
</tr>
<tr>
<td>V₀</td>
<td>V₆</td>
<td>V₇</td>
</tr>
<tr>
<td>V₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V₃</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameterized Symbolic Reachability

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

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

\[
\text{Init}(\nu_0) \land \left( \bigwedge_{s \in [0,K]} \text{Tr}(i_s, N, \nu_s, \nu_{s+1}) \right) \land \text{Bad}(\nu_K) \not\Rightarrow \bot
\]

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

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

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

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

\[\{ VC(T) \}\]

\(Init, Bad, \) 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 Quantifier (Solution)

\[ \text{Init}(i, i, v) \implies \text{Inv}_1(i, v) \]
\[ \text{Inv}_1(i, v) \land \text{Tr}(i, v, v') \implies \text{Inv}_1(i, v') \]
\[ j \neq i \land \text{Inv}_1(i, v) \land \text{Inv}_1(j, v) \land \text{Tr}(j, v, v') \implies \text{Inv}_1(i, v') \]
\[ \text{Inv}_1(i, v) \land \text{Inv}_1(j, v) \implies \neg \text{Bad}(i, j, v) \]

Claim

- 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 \textit{simple} single quantifier formula
  – simple == 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

$$Inv(v) \land Tr(j, v, v') \implies Inv(v')$$

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

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

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

Restrict

Cases

Instantiate

$$Inv_1(i, v) \land Tr(i, v, v') \implies Inv_1(i, v')$$

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

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

Claim

- 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 simple two quantifier formula
- At least 2 quantifiers are “needed” for systems with global guards

Extends to K-quantifiers
Putting it all together

\[ k := 1 \; \]
\[ \textbf{while} \; \text{true} \; \textbf{do} \]
\[ \quad Inv_k(i_1, \ldots, i_k, v) := \text{Solve}(U^k(V C^\omega(T))) \; ; \]
\[ \quad \text{if} \; Inv_k(i_1, \ldots, i_k, v) \neq \text{null} \; \text{then} \]
\[ \quad \quad \text{return} \; \text{“inductive invariant found:} \]
\[ \quad \quad \quad \forall i_1, \ldots, i_k . Inv(i_1, \ldots, i_k, v)’’ \]
\[ \quad \; \text{res} := \text{ModelCheck}(T_k) \; ; \]
\[ \quad \text{if} \; \text{res} = \text{cex} \; \text{then} \]
\[ \quad \quad \text{return} \; \text{“counterexample found for} \; k \; \text{processes}’’ \]
\[ \quad k := k + 1 \]
Finite vs Infinite Number of Processes

```python
def proc(i):
    do
    b[i] = 0 : b[i] := 1 ;
    b[i] = 1 : b[i] := 0 ;
    (∀j ≠ i . b[j] ≠ b[i]) : pc[i] := E ;
    def init(i, j):
        pc[i] = I ∧ b[i] = 0 ;
    def bad(i, j):
        pc[i] = E ;
```

Tr does not depend on N (number of processes)
Safe for infinitely many processes

\[ Inv ≡ (∀i . b[i] ∈ [0, 1] ∧ pc[i] = I) ∧ \]
\[ (∀i, j, k . distinct(i, j, k) ⇒ ¬distinct(b[i], b[j], b[k])) \]

Cex 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
- …
Contact Information

Arie Gurfinkel, Ph. D.
Principle Researcher
CSC/SSD
Telephone: +1 412-268-5800
Email: info@sei.cmu.edu

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257