The SeaHorn Verification Framework

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Arie Gurfinkel

with Teme Kahsai, Jorge A. Navas, and Anvesh Komuravelli
April 11th, 2015
Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS Furnished ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0002333
Automated Software Analysis

Program → Automated Analysis

Correct
Incorrect

Software Model Checking with Predicate Abstraction
e.g., Microsoft’s SDV

Abstract Interpretation with Numeric Abstraction
e.g., ASTREE, Polyspace

SeaHorn Verification Framework
Gurfinkel, April 11, 2015
© 2015 Carnegie Mellon University
Turing, 1936: “undecidable”
Turing, 1949

How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Alan M. Turing. “Checking a large routine”, 1949
SeaHorn

A fully automated verification framework for LLVM-based languages.

http://seahorn.github.io
SeaHorn Verification Framework

Distinguishing Features
- LLVM front-end(s)
- Constrained Horn Clauses to represent Verification Conditions
- Comparable to state-of-the-art tools at SV-COMP’15

Goals
- be a state-of-the-art Software Model Checker
- be a framework for experimenting and developing CHC-based verification
Related Tools

CPAChecker
- Custom front-end for C
- Abstract Interpretation-inspired verification engine
- Predicate abstraction, invariant generation, BMC, k-induction

SMACK / Corral
- LLVM-based front-end
- Reduces C verification to Boogie
- Corral / Q verification back-end based on Bounded Model Checking with SMT
SeaHorn Usage

> sea pf FILE.c
Outputs sat for unsafe (has counterexample); unsat for safe

Additional options

- --cex=trace.xml outputs a counter-example in SV-COMP’15 format
- --track={reg,ptr,mem} track registers, pointers, memory content
- --step={large,small} verification condition step-semantics
  - small == basic block, large == loop-free control flow block
- --inline inline all functions in the front-end passes

Additional commands

- sea smt -- generates CHC in extension of SMT-LIB2 format
- sea clp -- generates CHC in CLP format (under development)
- sea lfe-smt -- generates CHC in SMT-LIB2 format using legacy front-end
Verification Pipeline

front-end

clang | pp | ms | opt | horn

compile
pre-process
mixed semantics
optimize
VC gen & solve
Constrained Horn Clauses (CHC)

**Definition:** A Constrained Horn Clause (CHC) is a formula of the form

\[ \forall \, \mathbf{V} \cdot (\phi \land p_1[X_1] \land \ldots \land p_n[X_n] \rightarrow h[X]), \]

where

- \(\phi\) is a constrained in a background theory \(A\) (e.g., arithmetic, arrays, SMT)
- \(p_1, \ldots, p_n, h\) are \(n\)-ary predicates
- \(p_i[X]\) is an application of a predicate to first-order terms

We write clauses as rules, with all variables implicitly quantified

\[ h[X] \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi. \]

A model of a set of clauses \(\Pi\) is an interpretation of each predicate \(p_i\) that makes all clauses in \(\Pi\) valid

A set of clauses is satisfiable if it has a model, and is unsatisfiable otherwise

A model is \(A\)-definable, if each \(p_i\) is definable by a formula \(\psi_i\) in \(A\)
FROM PROGRAMS TO CLAUSES
Horn Clauses by Weakest Liberal Precondition

Prog = def Main(x) { body\textsubscript{M} }, ..., def P(x) { body\textsubscript{P} }

wlp (x=E, Q) = let x=E in Q
wlp (assert (E), Q) = E \land Q
wlp (assume(E), Q) = E \rightarrow Q
wlp (while E do S, Q) = I(w) \land 
  \forall w . ((I(w) \land E) \rightarrow wlp (S, I(w))) \land ((I(w) \land \neg E) \rightarrow Q))

wlp (y = P(E), Q) = p_{\text{pre}}(E) \land (\forall r. p(E, r) \rightarrow Q[r/y])

ToHorn (def P(x) {S}) = wlp (x0=x ; assume (p_{\text{pre}}(x)); S, p(x0, ret))
ToHorn (Prog) = wlp (Main(), true) \land \forall\{P \in Prog\} . ToHorn (P)
Horn Clauses by Dual WLP

Assumptions
• each procedure is represented by a control flow graph
  – i.e., statements of the form $l_i:S; \text{goto } l_j$, where $S$ is loop-free
• program is unsafe iff the last statement of Main() is reachable
  – i.e., no explicit assertions. All assertions are top-level.

For each procedure $P(x)$, create predicates
• $l(w)$ for each label, $p_{en}(x_0,x,w)$ for entry, $p_{ex}(x_0,r)$ for exit

The verification condition is a conjunction of clauses:
\[
p_{en}(x_0,x) \leftarrow x_0 = x \\
l_i(x_0,w') \leftarrow l_j(x_0,w) \land \neg \text{wlp } (S, \neg (w=w')), \text{ for each statement } l_i:S; \text{goto } l_j \\
p(x_0,r) \leftarrow p_{ex}(x_0,r) \\
\text{false } \leftarrow \text{Main}_{ex}(x, \text{ret})
\]
Example Horn Encoding

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

```
\begin{align}
\text{l}_0 : & \quad x = 1 \\
& \quad y = 0 \\
\text{l}_1 : & \quad b_1 = \text{nondet()} \\
\text{l}_2 : & \quad x = x + y \\
& \quad y = y + 1 \\
\text{l}_3 : & \quad b_2 = x ≥ y \\
\text{l}_4 : & \quad \text{err} \\
\text{l}_5 : & \quad p_0. \\
\text{l}_6 : & \quad p_1(x, y) ← p_0, x = 1, y = 0. \\
\text{l}_7 : & \quad p_2(x, y) ← p_1(x, y). \\
\text{l}_8 : & \quad p_3(x, y) ← p_1(x, y). \\
\text{l}_9 : & \quad p_1(x', y') ← p_2(x, y), x' = x + y, y' = y + 1. \\
\text{l}_10 : & \quad p_4 ← (x ≥ y), p_3(x, y). \\
\text{l}_11 : & \quad p_\text{err} ← (x < y), p_3(x, y). \\
\text{l}_12 : & \quad p_4 ← p_4.
\end{align}
```
Large Step Encoding: Single Static Assignment

```c
int x, y, n;
x = 0;
while (x < N) {
    if (y > 0)
        x = x + y;
    else
        x = x - y;
y = -1 * y;
}
```

```
0: goto 1
1: x_0 = PHI(0:0, x_3:5);
y_0 = PHI(y:0, y_1:5);
if (x_0 < N) goto 2 else goto 6
2: if (y_0 > 0) goto 3 else goto 4
3: x_1 = x_0 + y_0; goto 5
4: x_2 = x_0 - y_0; goto 5
5: x_3 = PHI(x_1:3, x_2:4);
y_1 = -1 * y_0;
goto 1
6:
```
Example: Large Step Encoding

0: goto 1
1: x_0 = PHI(0:0, x_3:5);
y_0 = PHI(y:0, y_1:5);
if (x_0 < N) goto 2 else goto 6

2: if (y_0 > 0) goto 3 else goto 4

3: x_1 = x_0 + y_0; goto 5

4: x_2 = x_0 - y_0; goto 5

5: x_3 = PHI(x_1:3, x_2:4);
y_1 = -1 * y_0;
goto 1

6:________
Example: Large Step Encoding

\[ x_1 = x_0 + y_0 \]
\[ x_2 = x_0 - y_0 \]
\[ y_1 = -1 \times y_0 \]

1: \[ x_0 = \text{PHI}(0:0, x_3:5); \]
    \[ y_0 = \text{PHI}(y:0, y_1:5); \]
    if \( x_0 < N \) goto 2 else goto 6

2: if \( y_0 > 0 \) goto 3 else goto 4

3: \[ x_1 = x_0 + y_0 \] goto 5

4: \[ x_2 = x_0 - y_0 \] goto 5

5: \[ x_3 = \text{PHI}(x_1:3, x_2:4); \]
    \[ y_1 = -1 \times y_0; \]
    goto 1
Example: Large Step Encoding

\[ x_1 = x_0 + y_0 \]
\[ x_2 = x_0 - y_0 \]
\[ y_1 = -1 \times y_0 \]

\[ B_2 \rightarrow x_0 < N \]
\[ B_3 \rightarrow B_2 \land y_0 > 0 \]
\[ B_4 \rightarrow B_2 \land y_0 \leq 0 \]
\[ B_5 \rightarrow (B_3 \land x_3=x_1) \lor (B_4 \land x_3=x_2) \]

\[ B_5 \land x'_0=x_3 \land y'_0=y_1 \]

\[ p_1(x'_0, y'_0) \leftarrow p_1(x_0, y_0), \phi. \]

---

1: \[ x_0 = \text{PHI}(0:0, x_3:5); \]
   \[ y_0 = \text{PHI}(y:0, y_1:5); \]
   \[ \text{if} (x_0 < N) \text{ goto 2 else goto 6} \]

2: \[ \text{if} (y_0 > 0) \text{ goto 3 else goto 4} \]

3: \[ x_1 = x_0 + y_0; \text{ goto 5} \]

4: \[ x_2 = x_0 - y_0; \text{ goto 5} \]

5: \[ x_3 = \text{PHI}(x_1:3, x_2:4); \]
   \[ y_1 = -1 \times y_0; \]
   \[ \text{goto 1} \]
Mixed Semantics

PROGRAM TRANSFORMATION
Mixed Semantics

Stack-free program semantics combining:

• operational (or small-step) semantics  
  – i.e., usual execution semantics
• natural (or big-step) semantics: function summary [Sharir-Pnueli 81]  
  – \((\sigma, \sigma') \in ||f||\) iff the execution of \(f\) on input state \(\sigma\) terminates and results in state \(\sigma'\)
• some execution steps are big, some are small

Non-deterministic executions of function calls

• update top activation record using function summary, or
• enter function body, forgetting history records (i.e., no return!)

Preserves reachability and non-termination

Theorem: Let \(K\) be the operational semantics, \(K^m\) the stack-free semantics,  
and \(L\) a program location. Then,
\[
K \vdash EF (pc=L) \iff K^m \vdash EF (pc=L) \quad \text{and} \quad K \vdash EG (pc\neq L) \iff K^m \vdash EG (pc\neq L)
\]
def main()
1: int x = nd();
2: x = x+1;
3: while(x>=0)
4: x=f(x);
5: if(x<0)
6: Error;
7: 
8: END;

def f(int y): return y
9: if(y>=10){
10: y=y+1;
11: y=f(y);
12: else if(y>0)
13: y=y+1;
14: y=y-1
15:

Summary of f(y)
(1≤y≤9 ∧ y’=y)
∨ (y≤0 ∧ y’=y-1)
Mixed Semantics as Program Transformation

```
main ()
    p1 (); p1 ();
    assert (c1);
    p1 ()
    p2 ();
    assert (c2);
    p2 ()
    assert (c3);

main\text{new} ()
    if (*) goto p1_{\text{entry}};
    else p1\text{new} ();
    if (*) goto p1_{\text{entry}};
    else p1\text{new} ();
    if (\neg c1) goto error;
    assume (false);

p1_{\text{entry}} :
    if (*) goto p2_{\text{entry}};
    else p2\text{new} ();
    p2_{\text{entry}} :
    if (\neg c2) goto error;
    assume (false);
    error : assert (false);

p1\text{new} ()
    if (*) goto p2_{\text{entry}};
    else p2\text{new} ();
    assume (c2);
    p2\text{new} ()
    assume (c3);
```
SOLVING CHC WITH SMT
A program \( P = (V, \text{Init}, \rho, \text{Bad}) \)

- Notation: \( \mathcal{F}(X) = \exists u . (X \land \rho) \lor \text{Init} \)

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

\[
\text{Init}(v_0) \land \left( \bigwedge_{i=0}^{N-1} \rho(v_i, v_{i+1}) \right) \land \text{Bad}(v_N) \not= \bot
\]

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

\[
\begin{align*}
\text{Init}(u) & \Rightarrow \text{Inv}(u) \\
\text{Inv}(u) \land \rho(u, v) & \Rightarrow \text{Inv}(v) \\
\text{Inv}(u) & \Rightarrow \neg \text{Bad}(u)
\end{align*}
\]

\{ \textbf{Inductive} \}

\{ \textbf{Safe} \}
IC3/PDR Algorithm Overview

**Input:** Transition system \( T = (Init, Tr, Bad) \)

1. \( F_0 \leftarrow Init \; ; \; N \leftarrow 0 \)

2. repeat

3. \( G \leftarrow \text{PDRMkSAFE}([F_0, \ldots, F_N], Bad) \)

4. if \( G = [\; ] \) then return UNSAFE;

5. \( \forall 0 \leq i \leq N \cdot F_i \leftarrow G[i] \)

6. \( F_0, \ldots, F_N \leftarrow \text{PDRPush}([F_0, \ldots, F_N]) \)

   // \( F_0, \ldots, F_N \) is a safe \( \delta \)-trace

7. if \( \exists 0 \leq i \leq N \cdot F_i = \emptyset \) then return SAFE;

8. \( N \leftarrow N + 1 \; ; \; F_N \leftarrow \emptyset \)

9. until \( \infty \);
IC3/PDR in Pictures

Frame $R_0$  Frame $R_1$  lemma

PdrMkSafe
Cex Queue
Trace

cex

SeaHorn Verification Framework
Gurfinkel, April 11, 2015
© 2015 Carnegie Mellon University
IC3/PDR in Pictures

PdrPush

PDR Invariants

\[ R_i \rightarrow \neg \text{Bad} \]
\[ \text{Init} \rightarrow R_i \]
\[ R_i \rightarrow R_{i+1} \]
\[ R_i \land \rho \rightarrow R_{i+1} \]
IC3/PDR

Data: $Q$ a queue of counter-examples. Initially, $Q = \emptyset$.
Data: $N$ a level indication. Initially, $N = 0$.
Data: $R_0, R_1, \ldots, R_N$ is a trace. Initially, $R_0 = Init$.

repeat

Unreachable If there is an $i < N$ s.t. $R_{i+1} \rightarrow R_i$, return Unreachable.

Reachable If there is an $m$ s.t. $\langle m, 0 \rangle \in Q$ return Reachable.

Unfold If $R_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$, $R_N \leftarrow \top$.

Candidate If for some $m$, $m \rightarrow R_N \wedge Bad$, then add $\langle m, N \rangle$ to $Q$.

Decide If $\langle m, i + 1 \rangle \in Q$ and there are $m_0$ and $m_1$ s.t. $m_1 \rightarrow m$, $m_0 \wedge m'_1$ is satisfiable, and $m_0 \wedge m'_1 \rightarrow \mathcal{F}(R_i) \wedge m'$, then add $\langle m_0, i \rangle$ to $Q$.

Conflict For $0 \leq i < N$: given a candidate model $\langle m, i + 1 \rangle \in Q$ and clause $\varphi$, such that $\neg \varphi \subseteq m$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add $\varphi$ to $R_j$, for $j \leq i + 1$.

Leaf If $\langle m, i \rangle \in Q$, $0 < i < N$ and $\mathcal{F}(R_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i + 1 \rangle$ to $Q$.

Induction For $0 \leq i < N$, a clause $(\varphi \lor \psi) \in R_i$, $\varphi \notin R_{i+1}$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add $\varphi$ to $R_j$, for each $j \leq i + 1$.

until $\infty$;
Data: $Q$ a queue of counter-examples. Initially, $Q = \emptyset$.
Data: $N$ a level indication. Initially, $N = 0$.
Data: $R_0, R_1, \ldots, R_N$ is a trace. Initially, $R_0 = \text{Init}$.

repeat
  Unreachable If there is an $i < N$ s.t. $R_{i+1} \rightarrow R_i$, return Unreachable.

Decide If $\langle m, i+1 \rangle \in Q$ and there are $m_0$ and $m_1$ s.t.
  $m_1 \rightarrow m$, $m_0 \land m'$ is satisfiable, and $m_0 \land m' \rightarrow$
  $F(R_i) \land m'$, then add $\langle m_0, i \rangle$ to $Q$.

Conflict For $0 \leq i < N$: given a candidate model
  $\langle m, i+1 \rangle \in Q$ and clause $\varphi$, such that $\neg \varphi \subseteq m$,
  if $F(R_i \land \varphi) \rightarrow \varphi$, then add $\varphi$ to $R_j$, for $j \leq i+1$.

Induction For $0 \leq i < N$, a clause $(\varphi \lor \psi) \in R_i$, $\varphi \notin R_{i+1}$, if
  $F(R_i \land \varphi) \rightarrow \varphi$, then add $\varphi$ to $R_j$, for each $j \leq i+1$.

until $\infty$;
Extending PDR to Arithmetic: APDR

Decide\textsuperscript{A} If \( \langle P, i + 1 \rangle \in Q \) and there is a model \( m(v, v') \) s.t. \( m \models F(R_i) \land P' \), add \( \langle P_{\downarrow}, i \rangle \) to \( Q \), where \( P_{\downarrow} \in \text{MBP}(v', m, F(R_i) \land P') \).

Conflict\textsuperscript{A} For \( 0 \leq i < N \), given a counterexample \( \langle P, i + 1 \rangle \in Q \) s.t. \( F(R_i) \land P' \) is unsatisfiable, add \( P^\uparrow = \text{ITP}(F(R_i)(v_0, v), P) \) to \( R_j \) for \( j \leq i + 1 \).

Model Based Projection: MBP(\( v, m, F \)) \hspace{1cm} [KGC’14]
- generates an implicant of \( \exists v \cdot F \) that contains the model \( m \)

Counter-examples are monomials (conjunction of inequalities)
Lemmas are clauses (disjunction of inequalities)

APDR computes an (possibly non-convex) QFLRA invariant in CNF
Craig Interpolation Theorem

**Theorem (Craig 1957)**

Let $A$ and $B$ be two First Order (FO) formulae such that $A \implies \neg B$, then there exists a FO formula $I$, denoted $ITP(A, B)$, such that

$$A \implies I \quad I \implies \neg B$$

$\text{atoms}(I) \subseteq \text{atoms}(A) \cap \text{atoms}(B)$

A Craig interpolant $ITP(A, B)$ can be effectively constructed from a resolution proof of unsatisfiability of $A \land B$

In Model Cheching, Craig Interpolation Theorem is used to safely over-approximate the set of (finitely) reachable states
Craig Interpolation for Linear Arithmetic

Useful properties of existing interpolation algorithms [CGS10] [HB12]

- $I \in \text{ITP} (A, B)$ then $\neg I \in \text{ITP} (B, A)$
- if $A$ is syntactically convex (a monomial), then $I$ is convex
- if $B$ is syntactically convex, then $I$ is co-convex (a clause)
- if $A$ and $B$ are syntactically convex, then $I$ is a half-space

$$A = \mathcal{F}(R_i)$$

$$B$$

$I =$ lemma
Model Based Projection

Expensive to find a quantifier-free

\[ \psi(y) \equiv \exists x \cdot \varphi(x, y) \]

1. find

\[ N \models \varphi(x, y) \]

(e.g. specific pre-post pair that needs to be generalized)

2. choose disjunct “covering” \( N \)

using virtual substitution

Lazy Quantifier Elimination!
MBP for Linear Rational Arithmetic

$$\exists \ell \cdot (\ell = e \land \phi_1) \lor (t < \ell \land \ell < u) \lor (\ell < u \land \phi_2)$$

$$\equiv (\phi_1 \lor (t < e \land e < u) \lor (e < u \land \phi_2)) \lor (t < u \lor (t < u \land \phi_2)) \lor \phi_2$$

pick a disjunct that covers a given model

Spacer: Solving CHC in Z3

Spacer: solver for SMT-constrained Horn Clauses
• stand-alone implementation in a fork of Z3
• 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
RESULTS
SV-COMP 2015

4\textsuperscript{th} Competition on Software Verification held (here!) at TACAS 2015

Goals

• Provide a snapshot of the state-of-the-art in software verification to the community.
• Increase the visibility and credits that tool developers receive.
• Establish a set of benchmarks for software verification in the community.

Participants:

• Over 22 participants, including most popular Software Model Checkers and Bounded Model Checkers

Benchmarks:

• C programs with error location (programs include pointers, structures, etc.)
• Over 6,000 files, each 2K – 100K LOC
• Linux Device Drivers, Product Lines, Regressions/Tricky examples
• http://sv-comp.sosy-lab.org/2015/benchmarks.php
Results for DeviceDriver category

![Graph showing the accumulated score over time for different verification tools: BLAST, CBMC, CPAchecker, ESBMC, SeaHorn, SMACKCorral, UAutomizer, UKojak. The x-axis represents the accumulated score, and the y-axis represents the time in seconds. The graph compares the performance of these tools across various scores and times.](image-url)
Conclusion

SeaHorn (http://seahorn.github.io)
• a state-of-the-art Software Model Checker
• LLVM-based front-end
• CHC-based verification engine
• a framework for research in logic-based verification

The future
• making SeaHorn useful to users of verification technology
  – counterexamples, build integration, property specification, proofs, etc.
• targeting many existing CHC engines
  – specialize encoding and transformations to specific engines
  – communicate results between engines
• richer properties
  – termination, liveness, synthesis
Contact Information

Arie Gurfinkel, Ph. D.
Sr. 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