# Inductive Invariant Generation via Abductive Inference

Isil Dillig et al.

December 28, 2020

# The Main Problem

## Definition (Loop Invariant Inference)

Given the precondition $P$ and postcondition $Q$. Finding an inductive invariant $I$, s.t.

$$P \Longrightarrow I, \{I \wedge C\}\pi\{I\} \wedge \neg C \Longrightarrow Q$$

Here "inductive" means:

- Implied by the loop's precondition.
- Preserve in each iteration of the loop's body.

# Overview of This Work

- A novel algorithm based on backtracking search for inferring inductive loop invariant.
  Form of invariants: boolean combination of linear integer constraints.
- Hoare-style program reasoning (VC Generation) + Abduction based on quantifier elimination (QE).
- A tool HOLA and implementation consideration.

# What is Abduction?

A non-deductive inference method.

- Verification condition (VC) in this setting: Validity $\longrightarrow$ Each candidate invariant is (1) inductive (2) implies the postcondition of corresponding loop.
- Abduction inference in invariant generation:
  Given an invalid clause $\chi \Rightarrow \gamma$ of the verification condition (VC).
  1. $(\chi \wedge \psi) \Rightarrow \gamma$
  2. SAT$(\chi \wedge \psi)$

  $\chi$ is the speculated invariant. The inference of $\psi$ as a strengthening of $\chi$ is called abduction inference.

# Running Example of the Algorithm

```
void foo(int flag) {
1.    int i, j, a, b;
2.    a = 0; b = 0; j = 1;
3.    if(flag) i=0; else i=1;
4.    while(*) [φ] {
5.          a++; b+=(j-i); i+=2;
6.          if(i%2 == 0) j+=2; else j++;
7.    }
8.    if(flag) assert(a==b);
}
```

1.1 Start with the weakest $\Delta_1(\phi) = true$. According to the explaination of VC: $\phi^1_{vc} = (true \Rightarrow (\text{flag} \Rightarrow a = b))$ which is not valid ( postcondition not implied).
1.2 Find strengthening $\phi_1$, s.t. $\models true \wedge \phi_1 \Rightarrow (\text{flag} \Rightarrow a = b)$.
1.3. Candidates: $\neg\text{flag}, a = b, \text{flag} \Rightarrow a = b$.

# Running Example of the Algorithm

```
void foo(int flag) {
1.    int i, j, a, b;
2.    a = 0; b = 0; j = 1;
3.    if(flag) i=0; else i=1;
4.    while(*) [φ] {
5.          a++; b+=(j-i); i+=2;
6.          if(i%2 == 0) j+=2; else j++;
7.    }
8.    if(flag) assert(a==b);
}
```

1.4 Suppose $\phi_1 = (\text{flag} \Rightarrow a = b)$.
2.1 Get a new VC:
$\phi_{vc}^2 = ((\text{flag} \Rightarrow a = b) \Rightarrow (\text{flag} \Rightarrow a + 1 = b + j - i))$ which is still not valid (not inductive).
2.2 Find strengthening $\phi_2$, s.t.
$\models ((\text{flag} \Rightarrow a = b) \wedge \phi_2 \Rightarrow (\text{flag} \Rightarrow a + 1 = b + j - i))$.
2.3 Candidates: $\neg\text{flag}, j = i + 1, \text{flag} \Rightarrow j = i + 1$

# Running Example of the Algorithm

```
void foo(int flag) {
1.    int i, j, a, b;
2.    a = 0; b = 0; j = 1;
3.    if(flag) i=0; else i=1;
4.    while(*) [φ] {
5.          a++; b+=(j-i); i+=2;
6.          if(i%2 == 0) j+=2; else j++;
7.    }
8.    if(flag) assert(a==b);
}
```

1.4.1 Suppose $\phi_2 = (j = i + 1)$.
2.1 Now the candidate invariant is $(\text{flag} \Rightarrow a = b) \wedge j = i + 1$.
However this is not correct one since the precondition computed does not hold on $j = i + 1$. **Backtracking**.
1.4.2 Suppose $\phi_2 = \text{flag} \Rightarrow j = i + 1$...
After several iterations:

$$(\text{flag} \Rightarrow (a = b \wedge j = i + 1))$$
$$\Rightarrow$$
$$(\text{flag} \Rightarrow (a + 1 = b + j - i \wedge (i\%2 = 0 \Rightarrow j = i + 1) \wedge (i\%2 \neq 0 \Rightarrow i = j)))$$

# Algorithm: InvGen

procedure INVGEN($\pi$):
  input: program $\pi$
  output: mapping $\Delta$ from each placeholder $\phi_i$
    to a concrete loop invariant $\psi_i$

(1) let $\Delta = [\phi_i \mapsto \text{true} \mid \phi_i \in \text{invs}(\pi)]$
(2) $\Delta' = \text{VERIFY}(\pi, \Delta)$
(3) return $\Delta'$

Here $\pi$ is the program whose syntax is:

$$
\begin{array}{lll}
\text{Program } \pi & := & s \\
\text{Statement } s & := & \text{skip} \mid v := e \\
 & & \mid s_1; s_2 \mid \text{choose } s_1\ s_2 \\
 & & \mid \text{while } C\ [\phi]\ \text{ do } \{s\} \\
 & & \mid \text{ assert } p \mid \text{assume } p \\
\text{Expression } e & := & v \mid \text{int} \mid e_1 + e_2 \\
 & & \mid e * \text{int} \mid e\%\text{int} \\
\text{Conditional } C & := & e_1 \odot e_2 \ \ (\odot \in \{<, >, =\}) \\
 & & \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \neg C
\end{array}
$$

# Algorithm: Verify

procedure VERIFY($\pi, \Delta$):
input: program $\pi$ and mapping $\Delta$
output: new mapping $\Delta'$ from each placeholder $\phi_i$ to a concrete loop invariant $\psi_i$

(4) $(\chi, \varphi) = \text{VCGEN}(\pi, \Delta)$
(5) if $\not\models \chi$ return $\emptyset$
(6) if $\models \text{elim}(\varphi)$ return $\Delta$
(7) let $\varphi_i \in \text{clauses}(\varphi)$ such that $\not\models \text{elim}(\varphi_i)$
(8) $(\phi, S) = \text{ABDUCE}(\varphi_i)$
(9) for each $\psi_i \in S$
(10) $\psi = \Delta(\phi) \wedge \psi_i$
(11) $\Delta' = \text{VERIFY}(\pi, \Delta[\phi \mapsto \psi])$
(12) if $\Delta' \neq \emptyset$ return $\Delta'$
(13) done
(14) return $\emptyset$

elim($\phi$): formula that substitutes all placeholders in $\phi$ to $true$.

# Algorithm: VcGen

Input of the subprocedure VcGen is $\pi$ and $\Delta$, where postcondition $\chi$ can be obtained directly from $\pi$.

$$\Delta, \chi \vdash s : \chi', \phi$$

means if elim($\phi$) is valid, then $\{\chi'\}s\{\chi\}$ is true.

$$(1)\frac{}{\Delta, \chi \vdash \text{skip} : \chi, \text{true}} \qquad (2)\frac{}{\Delta, \chi \vdash v := e : \chi[e/v], \text{true}}$$

$$(3)\frac{}{\Delta, \chi \vdash \text{ assert } C : \chi \wedge C, \text{true}} \qquad (4)\frac{}{\Delta, \chi \vdash \text{assume } C : C \Rightarrow \chi, \text{true}}$$

$$(5)\frac{\begin{array}{c}\Delta, \chi \vdash s_2 : \chi_2, \varphi_2 \\ \Delta, \chi_2 \vdash s_1 : \chi_1, \varphi_1\end{array}}{\Delta, \chi \vdash s_1; s_2 : \chi_1, \varphi_1 \wedge \varphi_2} \qquad (6)\frac{\begin{array}{c}\Delta, \chi \vdash s_1 : \chi_1, \varphi_1 \\ \Delta, \chi \vdash s_2 : \chi_2, \varphi_2\end{array}}{\Delta, \chi \vdash \text{choose } s_1\ s_2 : \chi_1 \wedge \chi_2, \varphi_1 \wedge \varphi_2}$$

$$(7)\frac{\begin{array}{c}\Delta, \Delta(\phi) \vdash s : \chi', \varphi' \\ \varphi_1 = (\neg C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi \\ \varphi_2 = (C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi'\end{array}}{\Delta, \chi \vdash \text{while } C\ [\phi] \text{ do } \{s\} : \Delta(\phi), \varphi_1 \wedge \varphi_2 \wedge \varphi'}$$

# Soundness of VcGen

## Theorem (Soundness)

*if* $\Delta, \chi \vdash s : \chi', \phi$ *is derivable and elim*$(\phi)$ *is valid, then* $\{\chi'\}s\{\chi\}$ *is valid.*

## Proof.

Prove by induction.

$$\frac{\begin{array}{c}\Delta, \Delta(\phi) \vdash s : \chi', \varphi' \\ \varphi_1 = (\neg C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi \\ \varphi_2 = (C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi'\end{array}}{\Delta, \chi \vdash \text{while } C\ [\phi] \text{ do } \{s\} : \Delta(\phi), \varphi_1 \wedge \varphi_2 \wedge \varphi'}$$

□

# Algorithm: Abduce

Recall from previous introduction of abduction:

1. $(\chi \wedge \psi) \Rightarrow \gamma$
2. SAT$(\chi \wedge \psi)$

Usually, the formula is of the form

$$I \wedge C \Rightarrow wp(s, I)$$

where $\chi := I \wedge C$. $\gamma := wp(s, I)$.
A direct try: $\psi := wp(s, I)$.
However this often causes a strengthening that is too weak and cause the divergence of the invariant.

# Example: Strengthen using $wp$ introduce divergence

```
int x = 0; int y = 0;
while(x < n) {
        x = x+1;
        y = y+2;
}
assert(y >= n);
```

$$\frac{\begin{array}{c}\Delta, \Delta(\phi) \vdash s : \chi', \varphi' \\ \varphi_1 = (\neg C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi \\ \varphi_2 = (C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi'\end{array}}{\Delta, \chi \vdash \text{while } C\ [\phi] \text{ do } \{s\} : \Delta(\phi), \varphi_1 \wedge \varphi_2 \wedge \varphi'}$$

1. Initially: $\Delta_0(\phi) = true$.
   VC: $x \geq n \wedge \phi \Rightarrow y \geq n$.
2. Strengthening:
   $\Delta_1(\phi) = (y \geq n)$. VC: $x \geq n \wedge y \geq n \wedge \phi \Rightarrow y \geq n + 2$.
3. Strengthening:
   $\Delta_2(\phi) = (y \geq n)$. VC:
   $x \geq n \wedge y \geq n \wedge y \geq n + 2 \wedge \phi \Rightarrow y \geq n + 4$....

# Abduction via Quantifier Elimination

Observation:

$$\models (\chi \wedge \psi \Rightarrow \gamma)$$

$$\psi \models \chi \Rightarrow \gamma$$

$$\forall V.\chi \Rightarrow \gamma \models \chi \Rightarrow \gamma$$

where $V$ is any subset of the set of free variables $\chi \Rightarrow \gamma$.
**Goal**: try formula that is equivalent to $\forall V.\chi \Rightarrow \gamma$ and does not contradict $\chi$.

## Definition (Universal Subset)

We call a set of variables $V$ a universal subset (US) of $\phi$ with respect ot $\psi$ if $(\forall V.\phi) \wedge \psi$ is satisfiable.

A Universal subset $U$ is *maximum* if for all other US $U'$, $|U| \geq |U'|$.

# Algorithm: Abduction via QE

procedure ABDUCE($\varphi$):

input: formula $\varphi$ of the form $\chi \wedge \phi \Rightarrow \gamma$

output: $(\phi, S)$ such that $\models \chi \wedge \psi_i \Rightarrow \gamma$ for every $\psi_i \in S$

(1) let $\phi = (\chi \wedge \phi) \Rightarrow \gamma$
(2) let $\theta = \{\chi\}$
(3) let $S = [\,]$
(4) while true
(5) $\quad V = \text{MUS}(\chi \Rightarrow \gamma, \theta)$
(6) $\quad$ if $V = \emptyset$ then break
(7) $\quad \psi = \text{QE}(\forall V.\chi \Rightarrow \gamma)$
(8) $\quad S = S :: \psi$
(9) $\quad \theta = \theta \cup \{\neg\psi\}$
(10) done
(11) return $(\phi, S)$

# Example: Abduction via QE

```
int x = 0; int y = 0;
while(x < n) {
      x = x+1;
      y = y+2;
}
assert(y >= n);
```

$$\frac{\begin{array}{c}\Delta, \Delta(\phi) \vdash s : \chi', \varphi' \\ \varphi_1 = (\neg C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi \\ \varphi_2 = (C \wedge \Delta(\phi) \wedge \phi) \Rightarrow \chi'\end{array}}{\Delta, \chi \vdash \text{while } C\ [\phi] \text{ do } \{s\} : \Delta(\phi), \varphi_1 \wedge \varphi_2 \wedge \varphi'}$$

1. Initially: $\Delta_0(\phi) = true$.
   VC: $x \geq n \wedge \phi \Rightarrow y \geq n$.

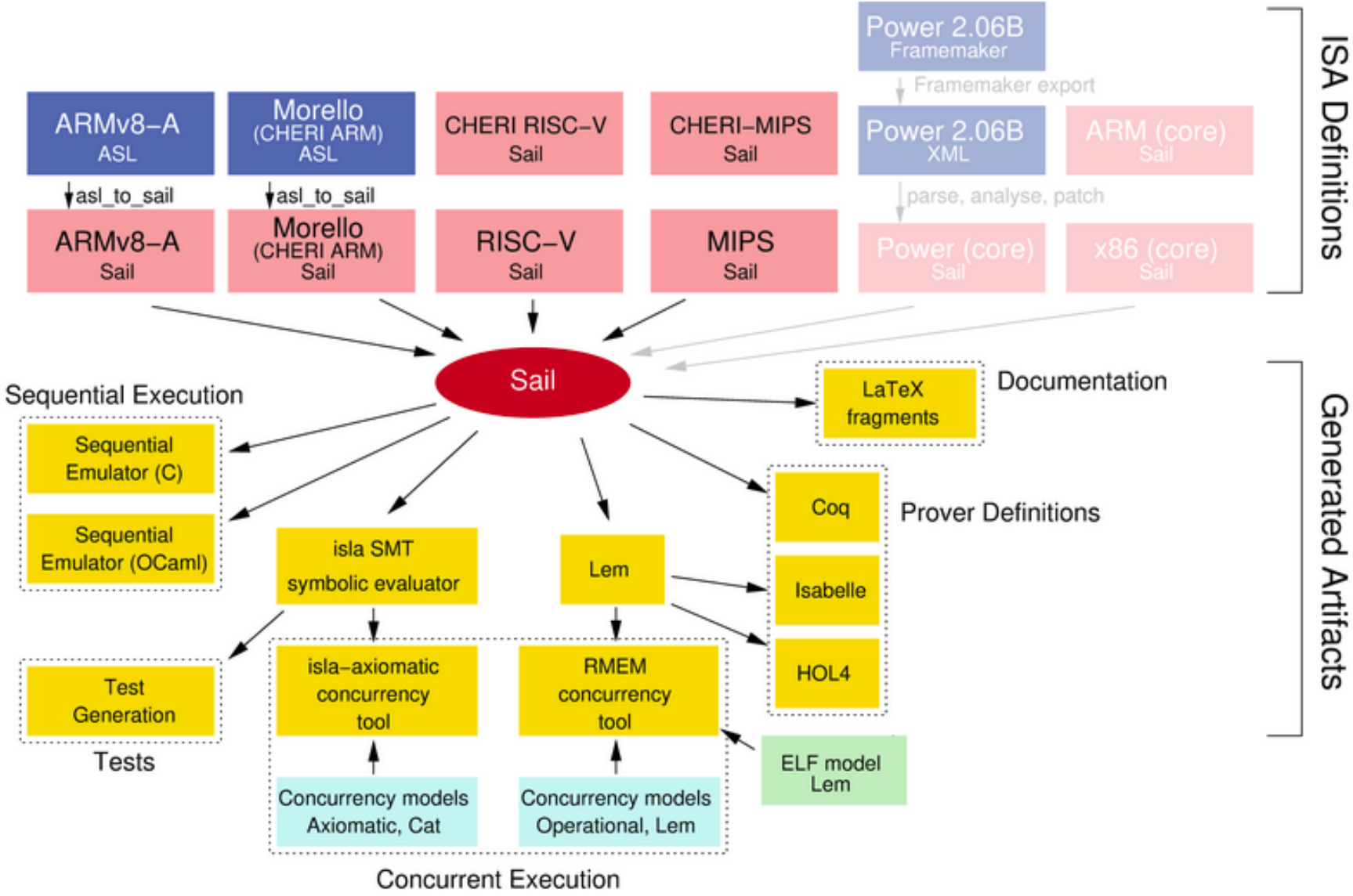
2. Do QE when $V = \{n\}$,
   $\psi := QE(\forall n.x \geq n \Rightarrow y \geq n)$. QE returns a result $y \geq x$, which is exactly the inductive loop invariant.

# Implementation Consideration

The algorithm is implemented as a tool HOLA.
Implemented in C language and use SAIL frontend for converting from C to IR.
Use Mistral SMT solver for satifiability solving and QE.
Features:

- Dual forwards and backwards analysis.
- Forward reasoning allow obtaining stronger inital loop invariants.
- Lazy abduction inference.

# Sail


Power 2.06B
Framemaker
Framemaker export
ARMv8–A
ASL
Morello
(CHERI ARM)
ASL
CHERI RISC–V
Sail
CHERI–MIPS
Sail
Power 2.06B
XML
ARM (core)
Sail
asl_to_sail
asl_to_sail
parse, analyse, patch
ARMv8–A
Sail
Morello
(CHERI ARM)
Sail
RISC–V
Sail
MIPS
Sail
Power (core)
Sail
x86 (core)
Sail
ISA Definitions
Sail
Sequential Execution
LaTeX
fragments
Documentation
Sequential
Emulator (C)
Sequential
Emulator (OCaml)
Coq
Isabelle
HOL4
Prover Definitions
isla SMT
symbolic evaluator
Lem
Test
Generation
Tests
isla–axiomatic
concurrency
tool
RMEM
concurrency
tool
ELF model
Lem
Concurrency models
Axiomatic, Cat
Concurrency models
Operational, Lem
Concurrent Execution
Generated Artifacts

# Experimental Settings

- Comparison with BLAST (CEGAR-based MC which use Craig-interpolation to generate invariants), InvGen (Constraint-based), Interproc (AI).
- Benchmark: 46 cases.
  - 26 are from other sources: inv. gen. papers, InvGen test suite, NECLA benchmarks.
  - 20 cases from their benchmarks.

# Experimental Results

| | HOLA | BLAST | INVGEN | INTERPROC |
|---|---|---|---|---|
| Rate | 93.5% | 43.5% | 47.8% | 37% |

HOLA successfully generate invariants for 13 cases where no other tools can (Divergence on other tools).
HOLA complements the existing techniques by considering more generalized invariant and prevent divergence by QE.
There is also about 2 cases solved by others but not HOLA (QE does not apply).
For HOLA:

- Avg. 22.4 iterations.
- Avg. 19.7 backtracking steps.

Strengthening per iteration and backtracking.

# Conclusion

# Later Work & Direction

- Survey of other inv. gen. tools.
- Survey of the toolchain SAIL.