Deriving Invariants in Propositional Logic by Algorithmic Learning, Decision Procedure, and Predicate Abstraction

Yungbum Jung (丁 永範)
(cowork with Soonho Kong, Bow-Yaw Wang*, and Kwangkeun Yi)
Seoul National University
Tsinghua University*

HKUST 2009
Combining Two Techniques for Loop Invariant Generation

Yungbum Jung (丁 永範)
(cowork with Soonho Kong, Bow-Yaw Wang*, and Kwangkeun Yi)
Seoul National University
Tsinghua University*

HKUST 2009
Algorithmic Learning $+$ Predicate Abstraction $?$

$\gamma(\beta) = \beta[p \mapsto \bar{p}]$; and
$\alpha(\theta) = \{\beta \in \text{Bool}_A : \beta$ is a canonical monomial and $\theta \wedge \gamma(\beta)$ is satisfiable}.

\[ \begin{align*}
\text{if } EQ(T) \rightarrow YES \text{ then return } T;
\text{ let } \mu \text{ be such that } EQ(T) \rightarrow \mu;
\text{ a } t = t + 1; \quad (H_t, S_t, a_t) \rightarrow (F, \emptyset, \emptyset);
\text{ if } EQ(\bigwedge_{i=1}^t H_i) \rightarrow YES \text{ then return } \bigwedge_{i=1}^t H_i;
\text{ let } \mu \text{ be such that } EQ(\bigwedge_{i=1}^t H_i) \rightarrow \mu;
I := \{ i : \mu \models H_t \};
\text{ if } I = \emptyset \text{ then goto 0;}
\text{ foreach } i \in I \text{ do}
\mu_i := \mu; \quad \text{ walk from } \mu_i \text{ towards } a_i \text{ while keeping } \mu_i \models \lambda;\nS_i := S_i \cup \{ \mu_i \otimes a_i \};
\text{ end}
H_i := M_{\text{var}}(S_i)[B \mapsto B \oplus a_i] \text{ for } i = 1, \ldots, t;
\end{align*} \]
Static Program Analysis

We want to know program properties without executions

• “Is variable x always greater than ten?”
• “No buffer overruns in the program?”
• “Does this program always terminate?”
Program Semantics

We can define program semantics as compositions

\[
\{P\} S \{Q\}, \ {Q\} T \{R\} \\
\{P\} S; T \{R\}
\]
Program Semantics

We can define program semantics as compositions

\[
\{P\}S\{Q\}, \ {Q}\ T\ \{R\} \\
\{P\}S;T\{R\}
\]

\{y=0\} \ x := y; \ x := x+1; \ \{x=1, \ y=0\}
Program Semantics

We can define program semantics as compositions

\[
\frac{\{P\}S\{Q\}, \{Q\}T\{R\}}{\{P\}S;T\{R\}}
\]

\[
\{y=0\} \ x := y; \ x := x+1; \ \{x=1, \ y=0\}
\]

\[
\{y=0\} \ x := y; \quad \{y=0, \ x=0\} \ x := x+1; \ \{x=1, \ y=0\}
\]
We can define program semantics as compositions

\[
\frac{\{P\}S\{Q\}, \{Q\}T\{R\}}{\{P\}S;T\{R\}}
\]

\[
\{y=0\} \ x := y; \ x := x+1; \ \{x=1, \ y=0\}
\]

\[
\{y=0\} \ x := y; \ \{y=0, \ x=0\} \ x := x+1; \ \{x=1, \ y=0\}
\]

\[
\frac{\{B \land P\}S\{R\}, \{\neg B \land P\}T\{R\},}{\{P\}if \ B \ then \ S \ else \ T\{R\}}
\]
Property Construction
Hurdles

- Loops
  - while, goto, recursive call, ...
- Unknown value from external environment
  - User input, contents of file, random number, ...
- Parallel programs
Loop Invariant

Algorithmic Learning + Predicate Abstraction

Loop Invariant
Loop Invariant?

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ \quad r := \text{random}(); \]
\[ \quad \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]
Loop Invariant?

\[ \text{i := 0;} \]
\[ \text{while i < 10 do} \]
\[ \quad \text{r := random();} \]
\[ \quad \text{if r != 0 then i := i + 1} \]
\[ \text{end} \]
Loop Invariant?

Entry Point

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ r := \text{random}(); \]
\[ \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]
Loop Invariant?

i := 0;
while i < 10 do
  r := random();
  if r != 0 then i := i + 1
end

i is an integer

Entry Point
Loop Invariant?

i := 0;
while i < 10 do
    r := random();
    if r != 0 then i := i + 1
end

i is an integer
i ≥ 0
Loop Invariant?

Entry Point

\[
i := 0;
\]

\[
\text{while } i < 10 \text{ do}
\]

\[
r := \text{random}();
\]

\[
\text{if } r \neq 0 \text{ then } i := i + 1
\]

end

\[i \text{ is an integer}\]
\[i \geq 0\]
\[i \leq 10\]
Loop Invariant?

Entry Point

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ r := \text{random}(); \]
\[ \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]

\[ i \text{ is an integer} \quad i \geq 0 \quad i \leq 10 \quad i = 0 \]
Loop Invariant?

i := 0;
while i < 10 do
    r := random();
    if r != 0 then i := i + 1
end

i is an integer
i ≥ 0  i ≤ 10  i = 0  i > 0
Loop Invariant?

Entry Point

\[ \text{i} := 0; \]
\[ \text{while i} < 10 \text{ do} \]
\[ \text{r} := \text{random}(); \]
\[ \text{if r} \neq 0 \text{ then i} := \text{i} + 1 \]
\[ \text{end} \]

\[ i \text{ is an integer} \quad i \geq 0 \quad i \leq 10 \]
\[ i = 0 \quad i > 0 \]
Loop Invariant?

Entry Point

\[
i := 0; \\
\text{while } i < 10 \text{ do} \\
r := \text{random}() ; \\
\text{if } r \neq 0 \text{ then } i := i + 1 \\
\text{end}
\]

\text{\textbf{i is an integer} } \quad i \geq 0 \quad i \leq 10

\text{\textbf{0} \leq i \leq 10}

\text{\textbf{\times} } i = 0 \quad \text{\textbf{\times} } i > 0
Loop Invariant?

Entry Point

i := 0;

while i < 10 do
    r := random();
    if r != 0 then i := i + 1
end

i is an integer

i ≥ 0  i ≤ 10

i = 0  i > 0

0 ≤ i ≤ 10

0 ≤ i < 10 ∨ (i = 10 ∧ r ≠ 0)
So Why Loop Invariant?
Program Semantics

We can define program semantics as compositions

\[
\frac{\{P\}S\{Q\}, \{Q\}T\{R\}}{\{P\}S;T\{R\}}
\]

\[
\{y=0\} \ x := y; \ x := x+1; \{x=1, y=0\}
\]

\[
\{y=0\} \ x := y; \ \{y=0, x=0\} \ x := x+1; \{x=1, y=0\}
\]

\[
\frac{\{B \land P\}S\{R\}, \{\neg B \land P\}T\{R\}}{\{P\} \text{if } B \text{ then } S \text{ else } T\{R\}}
\]
Program Semantics

Semantics of loop is defined recursively

\[
\{I \land B\} \, S \, \{I\} \\
\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \land I\}
\]
Program Semantics

Semantics of loop is defined recursively

\[
\{ I \land B \} \ S \ \{ I \} \\
\{ I \} \ \text{while} \ B \ \text{do} \ S \ \text{done} \ \{ \neg B \land I \}
\]

Finding loop invariant is challenging!
Post Condition

\begin{verbatim}
  i := 0;
  while i < 10 do
    r := random();
    if r != 0 then i := i + 1
  end
\end{verbatim}
Post Condition

\[
i := 0;\\
\text{while } i < 10 \text{ do}\\
\quad r := \text{random}();\\
\quad \text{if } r \neq 0 \text{ then } i := i + 1\\
\text{end}\\
\]

\[\begin{align*}
\text{Post Condition:} \\
i &\geq 10 \\
0 \leq i < 10 &\lor (i = 10 \land r \neq 0)\\
\neg B &\land I
\end{align*}\]
Post Condition

\begin{align*}
    i &:= 0; \\
    \text{while } i < 10 \text{ do} \\
    & \quad r := \text{random}(); \\
    & \quad \text{if } r \neq 0 \text{ then } i := i + 1 \\
    \text{end}
\end{align*}

\begin{align*}
    i \geq 10 &\quad \text{and} \quad 0 \leq i < 10 \lor (i = 10 \land r \neq 0)
\end{align*}

Postcondition

\begin{align*}
    i = 10 \land r \neq 0
\end{align*}
Loop Invariant

while !(success || give_up) {
    entered_phase := 0;
    if !phase then
        if cutoff = 0 then cutoff := 1;
        else if cutoff = 1 && maxcost > 1 then cutoff := maxcost;
            else phase := 1; entered_phase := 1; cutoff := 1000;
                if cutoff = maxcost && !search then give_up := 1;
            else
                count := count + 1;
                if count > words then give_up := 1;
        if entered_phase then count := 1;
    linkages := random();
    if linkages > 5000 then linkages := 5000;
    canonical := 0; valid := 0;
    if linkages then
        valid := random();
        canonical := 0; valid := 0;
    if valid > 0 then success := 1;
}

Could you find any invariants of this loop?
Could you find any invariants of this loop?
Algorithmic Learning

+ 

Predicate Abstraction

Important

Loop Invariant
Algorithmic Learning

Yes or No

Yes or Counter Example

Membership Query

Equivalence Query
Algorithmic Learning
Algorithmic Learning

Yes for Lee
No for Bush

Membership Query

Equivalence Query

Actor
Algorithmic Learning

Yes for Lee
No for Bush

Actor

Membership Query

Equivalence Query
Algorithmic Learning

Yes for Lee
No for Bush

Actor

for Chinese

Membership Query

Equivalence Query
Algorithmic Learning

- Infers unknown boolean formula $b_1 \lor b_2$
- Membership queries (YES or NO)
  
  \[
  MEM(FT) \rightarrow YES \quad FT \models b_1 \lor b_2 \\
  MEM(FF) \rightarrow NO \quad FF \not\models b_1 \lor b_2
  \]
- Equivalence queries (YES or Counter Example)
  
  \[
  EQ(b_1 \land b_2) \rightarrow FT \quad FT \not\models b_1 \land b_2
  \]
Loop Invariant

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ \quad r := \text{random}(); \]
\[ \quad \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]

\[ 0 \leq i < 10 \lor (i = 10 \land r \neq 0) \]
Loop Invariant

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ \quad r := \text{random}(); \]
\[ \quad \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]

\[ 0 \leq i < 10 \lor (i = 10 \land r \neq 0) \]

Loop invariant is not boolean formula
Predicate Abstraction

Boolean Formula

Algorithmic Learning

Teacher

Answering Queries

+ ?

Predicate Abstraction

Loop Invariant

Important
Predicate Abstraction

• Boolean variable $b_p$ represents atomic proposition $p$

• The found boolean formula is translated into the corresponding propositional formula

$$\alpha(a < 10) = b_{a<10}$$

$$\gamma(b_{a<10} \land b_{a>0} \land \neg b_{a<5}) = a < 10 \land a > 0 \land \neg(a < 5)$$
Predicate Abstraction

We generate finite atomic propositions from program

\[ i := 0; \]
\[ \text{while } i < 10 \text{ do} \]
\[ \quad r := \text{random}(); \]
\[ \quad \text{if } r \neq 0 \text{ then } i := i + 1 \]
\[ \text{end} \]

\{i = 0, i < 10, r = 0, i > 0, \ldots\}
Resolving Queries

Boolean Formula

Algorithmic Learning

+.Predicate Abstraction

Teacher

Answering Queries

Propositions

? Important

Loop Invariant
Algorithmic Learning
Algorithmic Learning

Actor
what we want to know

- Membership Query
- Equivalence Query
Using Approximations
Using Approximations

- Membership Query
- Equivalence Query

Human

Actor

Male Actor
Membership Query

Human

Actor

Male Actor

Jackie Chan
Bruce Lee
Zhang Ziyi

Actor
Membership Query

Human

Actor

Male Actor

Jackie Chan

Bruce Lee

Zhang Ziyi

Actor
Membership Query

Human

Actor

Male Actor

Jackie Chan
Bruce Lee
Zhang Ziyi
Equivalence Query

Human

Actor

Male Actor

Chinese
Equivalence Query

Human

Actor

Male Actor

Chinese

?
Equivalence Query

Human

Actor

Male Actor

Chinese
Equivalence Query

Human

Actor

Male Actors

Jackie Chan
Bruce Lee
Zhang Ziyi
Equivalence Query

Human

Actor

Male Actors
Approximations on Loop Invariant

\[
\{P\} \text{ while } B \text{ do } S \text{ done } \{R\}
\]

\[
\{I \wedge B\} \ S \ \{I\} \\
\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \wedge I\}
\]
Approximations on Loop Invariant

\(\{P\} \text{ while } B \text{ do } S \text{ done } \{R\}\)

\(\{I \land B\} \ S \ \{I\}\)

\(\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \land I\}\)

(a) \(P \Rightarrow I\)
Approximations on Loop Invariant

\[
\{P\} \text{ while } B \text{ do } S \text{ done } \{R\} \\
\{I \land B\} \quad S \quad \{I\} \\
\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \land I\}
\]

(a) \( P \Rightarrow I \) \quad (b) \( I \land B \Rightarrow \text{Pre}(I, S) \)
Approximations on Loop Invariant

\{P\} \text{ while } B \text{ do } S \text{ done } \{R\}

\{I \land B\} \quad S \quad \{I\}

\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \land I\}

(a) \ P \Rightarrow I \quad (b) \ I \land B \Rightarrow \text{Pre}(I, S) \quad (c) \ \neg B \land I \Rightarrow R
Approximations on Loop Invariant

\[
\begin{align*}
\{P\} \text{ while } B \text{ do } S \text{ done } \{R\} \\
\{I \land B\} \quad S \\ \\
I \quad \text{while } B \text{ do } S \text{ done } \{\neg B \land I\}
\end{align*}
\]

\( (a) \ P \Rightarrow I \quad (b) \ I \land B \Rightarrow Pre(I,S) \quad (c) \ \neg B \land I \Rightarrow R \)
Approximations on Loop Invariant

\[
\begin{align*}
\{P\} \text{ while } B \text{ do } S \text{ done } \{R\} \\
\{I \land B\} \quad S \quad \{I\} \\
\{I\} \text{ while } B \text{ do } S \text{ done } \{\neg B \land I\}
\end{align*}
\]

\[(a) \quad P \Rightarrow I \quad (b) \quad I \land B \Rightarrow Pre(I, S) \quad (c) \quad \neg B \land I \Rightarrow R\]

\[\neg(\neg B \land I) \lor R \equiv B \lor \neg I \lor R \equiv I \Rightarrow B \lor R\]
Approximations on Loop Invariant

\[ \{ P \} \text{ while } B \text{ do } S \text{ done } \{ R \} \]

\[ \begin{align*}
\{ I \land B \} & \quad S \\
\{ I \} & \quad \text{ while } B \text{ do } S \text{ done } \{ \neg B \land I \}
\end{align*} \]

\[(a) \quad P \Rightarrow I \quad (b) \quad I \land B \Rightarrow \text{Pre}(I, S) \quad (c) \quad \neg B \land I \Rightarrow R \]

\[ \neg (\neg B \land I) \lor R \equiv B \lor \neg I \lor R \equiv I \Rightarrow B \lor R \]
Loop Invariants

**Precondition**
\[
\{ \text{phase} = 0 \land \text{success} = 0 \land \text{give_up} = 0 \land \text{cutoff} = 0 \land \text{count} = 0 \} \\
\text{while !(success || give_up)} \{ \\
\quad \text{entered_phase} := 0; \\
\quad \text{if !phase then} \\
\quad \quad \text{if cutoff} = 0 \text{ then cutoff} := 1; \\
\quad \quad \text{else if cutoff} = 1 \text{ && maxcost} > 1 \text{ then cutoff} := \text{maxcost}; \\
\quad \quad \text{else phase} := 1; \text{ entered_phase} := 1; \text{ cutoff} := 1000; \\
\quad \text{if cutoff} = \text{maxcost} \text{ && !search then give_up} := 1; \\
\text{else} \\
\quad \text{count} := \text{count} + 1; \\
\quad \text{if count} > \text{words then give_up} := 1; \\
\text{if entered_phase then count} := 1; \\
\text{linkages} := \text{random()}; \\
\text{if linkages} > 5000 \text{ then linkages} := 5000; \\
\text{canonical} := 0; \text{ valid} := 0; \\
\text{if linkages then} \\
\quad \text{valid} := \text{random()}; \\
\quad \text{canonical} := 0; \text{ valid} := 0; \\
\text{if valid} > 0 \text{ then success} := 1; \\
\}\]

**Postcondition**
\[
\{ \text{valid} > 0 \lor \text{count} > \text{words} \lor (\text{cutoff} = \text{maxcost} \land \neg \text{search}) \} \land \\
\text{valid} \leq \text{linkages} \land \text{canonical} = \text{linkages} \land \text{linkages} \leq 5000 \}
\]
Loop Invariants

Precondition

\{ phase = 0 \land success = 0 \land give_up = 0 \land cutoff = 0 \land count = 0 \}

while !(success \lor give_up) {
    entered_phase := 0;
    if !phase then
        if cutoff = 0 then cutoff := 1;
        else if cutoff = 1 && maxcost > 1 then cutoff := maxcost;
    canonical := 0; valid := 0;
    if linkages then
        valid := random();
        canonical := 0; valid := 0;
    if valid > 0 then success := 1;
}

\begin{align*}
success & \Rightarrow (valid \leq linkages \land linkages \leq 5000 \land canonical = linkages) \land \\
success & \Rightarrow (\neg search \lor count > words \lor valid \neq 0) \land \\
success & \Rightarrow (count > words \lor cutoff = maxcost \lor (canonical \neq 0 \land valid \neq 0 \land linkages \neq 0)) \land \\
give_up & \Rightarrow ((valid = 0 \land linkages = 0 \land canonical = linkages) \lor \\
& \quad (canonical \neq 0 \land valid \leq linkages \land linkages \leq 5000 \land canonical = linkages)) \land \\
give_up & \Rightarrow (cutoff = maxcost \lor count > words \lor \\
& \quad (canonical \neq 0 \land valid \neq 0 \land linkages \neq 0)) \land \\
give_up & \Rightarrow (\neg search \lor count > words \lor valid \neq 0)
\end{align*}

Postcondition

\{ (valid > 0 \lor count > words \lor (cutoff = maxcost \land \neg search)) \land \\
valid \leq linkages \land canonical = linkages \land linkages \leq 5000 \}
Loop Invariants

Precondition
\[
\{ \text{phase} = 0 \land \text{success} = 0 \land \text{count} = 0 \land \text{cutoff} = 0 \land \text{linkages} = 0 \}
\]

while !(success || give_up)
\[
\text{entered\_phase} := 0;
\]
if !phase then
\[
\text{if cutoff} = 0 \text{ then cutoff} := 1;
\]
else if cutoff = 1 then cutoff := maxcost;
\[
\text{success} \Rightarrow (\text{valid} \leq \text{linkages} \land \text{linkages} = \text{linkages} \land \text{canonical} = \text{linkages}) \land
\]
\[
\text{success} \Rightarrow (\neg \text{search} \lor \text{count} > \text{words} \lor \text{canonical} = 0 \land \text{valid} = 0 \land \text{cutoff} = 0 \land \text{linkages} = 0 \land \text{linkages} \neq 0) \land
\]
\[
\text{give\_up} \Rightarrow ((\text{valid} = 0 \land \text{linkages} = 0) \land \text{cutoff} = 0 \land \text{linkages}) \lor
\]
\[
(\text{canonical} = 0 \land \text{valid} = 0 \land \text{cutoff} = 0 \land \text{linkages} \leq 5000 \land \text{canonical} = \text{linkages}) \land
\]
\[
\text{give\_up} \Rightarrow (\text{cutoff} = \text{maxcost} \lor \text{count} \leq \text{words} \lor \text{valid} \leq \text{cutoff} = 0) \lor
\]
\[
\text{give\_up} \Rightarrow (\neg \text{search} \lor \text{count} > \text{words} \lor \text{valid} \neq 0)
\]
\[
\text{canonical} := 0; \text{valid} := 0;
\]
if linkages
\[
\text{valid} := 0;
\]
\[
\text{canonical} := 0;
\]
if valid > 0 then cutoff := 1;
\}

Postcondition
\[
\{ (\text{valid} > 0 \lor \text{count} > \text{words} \lor (\text{cutoff} = \text{maxcost} \land \neg \text{search})) \land
\]
\[
\text{valid} \leq \text{linkages} \land \text{canonical} = \text{linkages} \land \text{linkages} \leq 5000 \}
\]
Conclusion

Boolean Formula + Predicate Abstraction

Answering Queries + Propositions

Algorithmic Learning + Important

Teacher Flexible

Approximations Loop Invariant

?
Future Work

• Supporting Quantifiers \( \forall i. 0 \leq i \leq n \Rightarrow a[i] = 0 \)

• Finding polynomial invariants \( x^2 + y^2 = 1 \)
Deriving Invariants by Algorithmic Learning, Decision Procedures, and Predicate Abstraction*

Yungbum Jung¹, Soonho Kong¹, Bow-Yaw Wang², and Kwangkeun Yi¹

¹ School of Computer Science and Engineering, Seoul National University
¹ dreameye,soon,kwang@ropas.snu.ac.kr
² Institute of Information Science, Academia Sinica
² bywang@iis.sinica.edu.tw

Abstract. By combining algorithmic learning, decision procedures, and predicate abstraction, we present an automated technique for finding loop invariants in propositional formulae. Given invariant approximations derived from pre- and post-conditions, our new technique exploits the flexibility in invariants by a simple randomized mechanism. The proposed technique is able to generate invariants for some Linux device drivers and SPEC2000 benchmarks in our experiments.

1 Introduction

Algorithmic learning has been applied to assumption generation in compositional reasoning [9]. In contrast to traditional techniques, the learning approach does not derive assumptions in an off-line manner. It instead finds assumptions by interacting with a model checker progressively. Since assumptions in compositional reasoning are generally not unique, algorithmic learning can exploit the flexibility in assumptions to attain preferable solutions. Applications in formal verification and interface synthesis have also been reported [9, 1, 2, 18, 7].

Finding loop invariants follows a similar pattern. Invariants are often not unique. Indeed, programmers derive invariants incrementally. They usually have their guesses of invariants in mind, and gradually refine their guesses by observing program behavior more. Since in practice there are many invariants for given pre- and post-conditions, programmers have more freedom in deriving invariants. Yet traditional invariant generation techniques do not exploit the flexibility. They have a similar impediment to traditional assumption generation.

This article reports our first findings in applying algorithmic learning to invariant generation. We show that the three technologies (algorithmic learning,