Loop Summaries as Horn Clauses

CPA 2020

Gidon Ernst

gidon.ernst@lmu.de
int \ x = x0, y = y0;

while(x != y) {
  if(x > y) x -= y;
  else y -= x;
}

int \ xn = x;
assert(xn == gcd(x0,y0));
Motivation (I)

Initially $x_0 = x^0 \land y_0 = y^0$

```c
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x0, y0));
```
Motivation (I)

\[
\text{initially } x_0 = x^0 \land y_0 = y^0
\]

forward invariant
\[
gcd(x_0, y_0) = gcd(x^0, y^0)
\]

```c
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else    y -= x;
}

int xn = x;
assert(xn == gcd(x0, y0));
```
Motivation (I)

Initially \( x_0 = x_0 \land y_0 = y_0 \)

\[
\text{forward invariant} \\
gcd(x_0, y_0) = gcd(x_0, y_0) \\
\downarrow \\
gcd(x_1, y_1) = gcd(x_0, y_0)
\]

\[
\text{finally } x_n = x_n = y_n \\
\text{conclusion} \\
gcd(x_n, y_n) = \bullet \quad \circ \quad \bullet
\]

\[
def = x_n = gcd(x_0, y_0);
\]
**Motivation (I)**

```c
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x0,y0));
```

**initially** $x_0 = x_0 \land y_0 = y_0$

**forward invariant**

$\begin{align*}
gcd(x_0, y_0) &= gcd(x_0, y_0) \\
gcd(x_1, y_1) &= gcd(x_0, y_0) \\
gcd(x_2, y_2) &= gcd(x_0, y_0)
\end{align*}$

finally $xn = x_n = y_n$

conclusion $gcd(x_n, y_n) = def = xn = gcd(x_0, y_0)$
Motivation (I)

```c
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x0,y0));
```

**initially** \( x_0 = x \theta \land y_0 = y \theta \)

**forward invariant**

\[
gcd(x_0, y_0) = gcd(x_\theta, y_\theta)
\]

\[
downarrow
\]

\[
gcd(x_1, y_1) = gcd(x_\theta, y_\theta)
\]

\[
downarrow
\]

\[
gcd(x_2, y_2) = gcd(x_\theta, y_\theta)
\]

\[
\vdots
\]

\[
gcd(x_n, y_n) = gcd(x_\theta, y_\theta)
\]
Motivation (I)

Initially $x_0 = x^0 \land y_0 = y^0$

Forward invariant

$\gcd(x_0, y_0) = \gcd(x^0, y^0)$

$\downarrow$

$\gcd(x_1, y_1) = \gcd(x^0, y^0)$

$\downarrow$

$\gcd(x_2, y_2) = \gcd(x^0, y^0)$

$\vdots$

$\gcd(x_n, y_n) = \gcd(x^0, y^0)$

Finally $x_n = x_n = y_n$

```
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else    y -= x;
}

int xn = x;
assert(xn == gcd(x0,y0));
```
Motivation (I)

```c
int x = x0, y = y0;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x0,y0));
```

**initially** $x_0 = x^0 \land y_0 = y^0$

**forward invariant**

$\begin{align*}
gcd(x_0, y_0) &= gcd(x^0, y^0) \\
gcd(x_1, y_1) &= gcd(x^0, y^0) \\
gcd(x_2, y_2) &= gcd(x^0, y^0) \\
&\vdots \\
gcd(x_n, y_n) &= gcd(x^0, y^0)
\end{align*}$

**finally** $x_n = x_n = y_n$

**conclusion**

$\underbrace{gcd(x_n, y_n)}_{def=\text{xn}} = gcd(x^0, y^0)$
Invariants $I$

$P(s_0) \Rightarrow I(s_0)$

$s_0 \Rightarrow s_i \Rightarrow s_{i+1} \Rightarrow s_n$
Invariants $I$

\[ P(s_0) \Rightarrow I(s_0) \quad I(s_i) \]

\[ s_0 \rightarrow s_i \rightarrow s_{i+1} \rightarrow \ldots \rightarrow s_n \]
Invariants $I$

$P(s_0) \Rightarrow I(s_0)$  $I(s_i)$  $I(s_{i+1})$

$s_0 \rightarrow s_i \rightarrow s_{i+1} \rightarrow \cdots \rightarrow s_n$
**Invariants** $I$

\[ P(s_0) \Rightarrow I(s_0) \quad I(s_i) \quad I(s_{i+1}) \quad I(s_n) \Rightarrow Q(s_n) \]

\[ s_0 \quad \cdots \quad \Rightarrow \quad s_i \quad \Rightarrow \quad s_{i+1} \quad \cdots \quad \Rightarrow \quad s_n \]
// unroll 1st iteration
if(x0 > y0) x1 = x0 - y0;
else y1 = y0 - x0;

// n-1 iterations
int x = x1, y = y1;

while(x != y) {
    if(x > y)  x -= y;
    else      y -= x;
}

int xn = x;
assert(xn == gcd(x1, y1));
Motivation (II)

// unroll 1st iteration
if(x0 > y0) x1 = x0 - y0;
else y1 = y0 - x0;

// n-1 iterations
int x = x1, y = y1;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x1,y1));

hypothesis \( x_n = \gcd(x_1, y_1) \)
(loop from \( x_1, y_1 \) is correct)
Motivation (II)

// unroll 1st iteration
if(x0 > y0) x1 = x0 - y0;
else y1 = y0 - x0;

// n-1 iterations
int x = x1, y = y1;

while(x != y) {
    if(x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x1,y1));

hypothesis \( x_n = \gcd(x_1, y_1) \)
(loop from \( x_1, y_1 \) is correct)

case \( x_0 > y_0 \)
\[
x_n = \gcd(x_1, y_1) \\
= \gcd(x_0 - y_0, y_0) \\
def = \gcd(x_0, y_0)
\]
Motivation (II)

// unroll 1st iteration
if (x0 > y0) x1 = x0 - y0;
else y1 = y0 - x0;

// n-1 iterations
int x = x1, y = y1;

while (x != y) {
  if (x > y) x -= y;
  else y -= x;
}

int xn = x;
assert(xn == gcd(x1, y1));

**hypothesis** \( x_n = \gcd(x_1, y_1) \)
(loop from \( x_1, y_1 \) is correct)

**case** \( x_0 > y_0 \)
\[
x_n = \gcd(x_1, y_1) \\
= \gcd(x_0 - y_0, y_0) \\
\text{def} = \gcd(x_0, y_0)
\]

**case** \( x_0 \leq y_0 \)
\[
x_n = \gcd(x_1, y_1) \\
= \gcd(x_0, y_0 - x_0) \\
\text{def} = \gcd(x_0, y_0)\]
// unroll 1st iteration
if (x0 > y0) x1 = x0 - y0;
else y1 = y0 - x0;

// n-1 iterations
int x = x1, y = y1;

while (x != y) {
    if (x > y) x -= y;
    else y -= x;
}

int xn = x;
assert(xn == gcd(x1,y1));

\textbf{hypothesis} \ x_n = \text{gcd}(x_1, y_1) \\
(\text{loop from } x_1,y_1 \text{ is correct})

\textbf{case } x_0 > y_0 \ \\
x_n = \text{gcd}(x_1, y_1) \ \\
= \text{gcd}(x_0 - y_0, y_0) \ \\
\text{def } \ = \text{gcd}(x_0, y_0)

\textbf{case } x_0 \leq y_0 \ \\
x_n = \text{gcd}(x_1, y_1) \ \\
= \text{gcd}(x_0, y_0 - x_0) \ \\
\text{def } \ = \text{gcd}(x_0, y_0)

\textbf{conclusion } x_n = \text{gcd}(x_0, y_0) \ \\
\text{loop from } x_0,y_0 \text{ is correct}
Invariants $I$ + Summaries $R$

\[ P(s_0) \Rightarrow I(s_0) \quad I(s_i) \quad I(s_{i+1}) \]

\[ s_0 \longrightarrow s_i \longrightarrow s_{i+1} \longrightarrow \cdots \rightarrow s_n \]
Invariants $I +$ Summaries $R$

$P(s_0) \Rightarrow I(s_0)$

$I(s_i)$

$I(s_{i+1})$

$s_0 \ldots \Rightarrow s_i \Rightarrow s_{i+1} \ldots \Rightarrow s_n$

assume $R(s_{i+1}, s_n)$
Invariants $I$ + Summaries $R$

\[ P(s_0) \Rightarrow I(s_0) \quad I(s_i) \quad I(s_{i+1}) \]

\[ s_0 \rightarrow s_i \rightarrow s_{i+1} \rightarrow \cdots \rightarrow s_n \]

- Assume $R(s_{i+1}, s_n)$
- Prove $R(s_i, s_n)$
Invariants $I + \text{Summaries } R$

$P(s_0) \Rightarrow I(s_0) \quad I(s_i) \quad I(s_{i+1})$

$s_0 \ldots \Rightarrow s_i \Rightarrow s_{i+1} \ldots \Rightarrow s_n$

assume $R(s_{i+1}, s_n)$

prove $R(s_i, s_n)$

finally $R(s_0, s_n) \Rightarrow Q(s_n)$
\[ P(s_0) \Rightarrow I(s_0) \quad I(s_i) \quad I(s_{i+1}) \]

\[ s_0 \rightarrow s_i \rightarrow s_{i+1} \rightarrow s_n \]

\[ \text{assume } R(s_{i+1}, s_n) \]

\[ \text{prove } R(s_i, s_n) \]

\[ \text{finally } R(s_0, s_n) \Rightarrow Q(s_n) \]

- Invariant \( I \) characterizes states reachable from \( P \)
- Relation \( R \) summarizes the effect of remaining iterations (treat loop as tail-recursive procedure)
Comparison

**invariant**
\[ \text{gcd}(x, y) = \text{gcd}(x_0, y_0) \]
- characterizes reachable *states*
- propagates forward over iterations
- \( \times \) non-trivial to find

Goal of this work:
- “accessible” presentation of summary-based techniques
- explore merit/limits of summaries in theory + experiments

Related: [Hehner 05, Tuerk 10, Charguéraud 10, Mraihi et al 13, ...]
### Comparison

<table>
<thead>
<tr>
<th><strong>Invariant</strong></th>
<th><strong>Summary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \gcd(x, y) = \gcd(x^0, y^0) )</td>
<td>( x_n = \gcd(x_i, y_i) )</td>
</tr>
<tr>
<td>▶ characterizes reachable states</td>
<td>▶ summarizes remaining loop iterations as a <code>relation</code></td>
</tr>
<tr>
<td>▶ propagates forward over iterations</td>
<td>▶ propagates starting state backwards over iterations</td>
</tr>
<tr>
<td>✗ non-trivial to find</td>
<td>✓ often similar to postcondition</td>
</tr>
</tbody>
</table>

**Goal of this work:**

- "accessible" presentation of summary-based techniques
- explore merit/limits of summaries in theory + experiments

**Related:** [Hehner 05, Tuerk 10, Charguéraud 10, Mraihi et al 13, ...]
Comparison

**Invariant**
\[ \text{gcd}(x, y) = \text{gcd}(x_0, y_0) \]
- characterizes reachable *states*
- propagates forward over iterations
- non-trivial to find

**Summary**
\[ x_n = \text{gcd}(x_i, y_i) \]
- summarizes remaining loop iterations as a *relation*
- propagates starting state backwards over iterations
- often similar to postcondition

Goal of this work:
- "accessible" presentation of summary-based techniques
- explore merit/limits of summaries in theory + experiments

Related: [Hehner 05, Tuerk 10, Charguéraud 10, Mraihi et al 13, ...]
Verfication Conditions for \{P\} while\(t\) \(B\) \(\{Q\}\)

- Invariant holds initially and propagates forwards:
  \[P(s) = \Rightarrow I(s) \land t(s) \land B(s, s') = \Rightarrow I(s')\]

- Invariant guarantees absence of runtime errors in the body:
  \[I(s) \land t(s) \land B(s, \bot) = \Rightarrow \text{false}\]

- Traditionally: Invariant ensures postcondition:
  \[I(s) \land \neg t(s) = \Rightarrow Q(s)\]

- Alternative: Summary holds finally, propagates backwards, and ensures postcondition:
  \[I(s) \land \neg t(s) = \Rightarrow R(s, s_n) \land I(s) \land t(s) \land B(s, s') \land R(s', s_n) = \Rightarrow R(s, s_n)\]
Verifcation Conditions for \( \{P\} \textbf{ while}(t) \ B \ \{Q\} \)

- Invariant holds initially and propagates forwards
  
  \[
  P(s) \implies I(s)
  \]
  
  \[
  I(s) \land t(s) \land B(s, s') \implies I(s')
  \]

- Invariant guarantees absence of runtime errors in the body
  
  \[
  I(s) \land t(s) \land B(s, \perp) \implies \text{false}
  \]
Invariant holds initially and propagates forwards
\[ P(s) \implies I(s) \]
\[ I(s) \land t(s) \land B(s, s') \implies I(s') \]

Invariant guarantees absence of runtime errors in the body
\[ I(s) \land t(s) \land B(s, \bot) \implies \text{false} \]

Traditionally: Invariant ensures postcondition
\[ I(s) \land \neg t(s) \implies Q(s) \]
Verification Conditions for $\{P\} \textbf{while}(t) \ B \ \{Q\}$

- Invariant holds initially and propagates forwards
  \[
P(s) \implies I(s)
  \]
  \[
  I(s) \land t(s) \land B(s, s') \implies I(s')
  \]

- Invariant guarantees absence of runtime errors in the body
  \[
  I(s) \land t(s) \land B(s, \bot) \implies \text{false}
  \]

- Traditionally: Invariant ensures postcondition
  \[
  I(s) \land \neg t(s) \implies Q(s)
  \]

- Alternative: Summary holds finally, propagates backwards,
  \[
  I(s) \land \neg t(s) \implies R(s, s)
  \]
  \[
  I(s) \land t(s) \land B(s, s') \land R(s', s_n) \implies R(s, s_n)
  \]

  and ensures postcondition
  \[
  P(s_0) \land R(s_0, s_n) \implies Q(s_n)
  \]
Theoretical Results

Theorem (Completeness): Given \( \{P\} \text{ while}(t) B \{Q\} \) holds. If invariant \( I \) proves the body safe, then there a summary exists \( R \) that proves postcondition \( Q \).

\( \rightsquigarrow \) approach to decompose the verification
Theoretical Results

Theorem (Completeness): Given $\{P\}$ while$(t)$ $B$ $\{Q\}$ holds. If invariant $I$ proves the body safe, then there a summary exists $R$ that proves postcondition $Q$.

$\mapsto$ approach to decompose the verification

Proposition: Given invariant $I$ and summary $R$, then the corresponding regular invariant is

$$\hat{I}(s) \equiv I(s) \land (\forall s_n. P(s_0) \land R(s, s_n) \implies R(s_0, s_n))$$

$\mapsto$ encode/validate summaries in existing tools
Theoretical Results

Theorem (Completeness): Given \( \{P\} \text{ while}(t) B \{Q\} \) holds. If invariant \( I \) proves the body safe, then there a summary exists \( R \) that proves postcondition \( Q \).

\( \leadsto \) approach to decompose the verification

Proposition: Given invariant \( I \) and summary \( R \), then the corresponding regular invariant is
\[
\hat{I}(s) \equiv I(s) \land \left( \forall s_n. \ P(s_0) \land R(s, s_n) \implies R(s_0, s_n) \right)
\]

\( \leadsto \) encode/validate summaries in existing tools

Proposition: Given invariant \( I \), there is a canonical summary \( R \)
\[
\hat{R}(s, s_n) \equiv \neg t(s_n) \land Q(s_n)
\]

\( \leadsto \) can represent both approaches in a single tool
Experiments

Preliminary Implementation

▶ translate C to verification conditions (cf. previous slide)
▶ ... which are Horn-clauses
   ⇝ can use existing solvers to find invariants + summaries
▶ current limitation: integers + arrays (partial), no heap

\(^1\) Note: summaries-only will give false answers
Experiments

Preliminary Implementation

- translate C to verification conditions (cf. previous slide)
- ... which are Horn-clauses
  - can use existing solvers to find invariants + summaries
- current limitation: integers + arrays (partial), no heap

Compare two settings

- invariants-only
- invariants + summaries

Hypothesis: invariants + summaries provides “more opportunity” to find solution

- solvers pick easiest approach
- simpler invariants possible with summaries

1 Note: summaries-only will give false answers
Results: ReachSafety-Loops

60s timeout, 289 tasks (55 unsupported)

Z3 4.8.9
Eldarica 2.0.4

Gidon Ernst  
LMU Munich
Discussion

Hypothesis: invariants + summaries provides “more opportunity” to find solution

X solvers pick easiest approach:
  ▶ optimized to finding invariants (loop detection)
  ▶ numeric problems unsuitable (too easy)

? simpler invariants possible with summaries
  ▶ no analysis done so far
  ▶ future work!
Summary

Verification with invariants and summaries known and used in practice

This work:
- Completeness Theorem: approach to decomposition
- Lifting Theorem: transfer between tools
- Preliminary evaluation

Outlook
- Analyze and compare results between two approaches
- Exploit decomposition theorem: infer small invariants first
- “Proper” implementation in CPAchecker