Loop Summaries as Horn Clauses

CPA 2020

Gidon Ernst

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

SoSy-Lab Software Systems

gidon.ernst@lmu.de

Motivation (I)

```
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 $\mathsf{x}_0 = \mathtt{x0} \wedge \mathsf{y}_0 = \mathtt{y0}$

```
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 $\mathsf{x}_0 = \mathtt{x0} \wedge \mathsf{y}_0 = \mathtt{y0}$

forward invariant

$$\mathrm{gcd}(\mathsf{x}_0, \mathsf{y}_0) = \mathrm{gcd}(\mathtt{x0}, \mathtt{y0})$$

```
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)

```
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 $\mathsf{x}_0 = \mathtt{x0} \wedge \mathsf{y}_0 = \mathtt{y0}$

forward invariant

$$\gcd(\mathsf{x}_0, \mathsf{y}_0) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\downarrow$$
$$\gcd(\mathsf{x}_1, \mathsf{y}_1) = \gcd(\mathtt{x0}, \mathtt{y0})$$

Motivation (I)

```
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 $\mathtt{x}_0 = \mathtt{x0} \land \mathtt{y}_0 = \mathtt{y0}$

forward invariant

$$\gcd(\mathtt{x}_0, \mathtt{y}_0) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\downarrow$$
$$\gcd(\mathtt{x}_1, \mathtt{y}_1) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\downarrow$$
$$\gcd(\mathtt{x}_2, \mathtt{y}_2) = \gcd(\mathtt{x0}, \mathtt{y0})$$

Motivation (I)

```
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 $x_0 = \mathtt{x0} \land y_0 = \mathtt{y0}$

forward invariant

$$\begin{array}{c} \gcd(x_0, y_0) = \gcd(\mathtt{x0}, \mathtt{y0}) \\ \downarrow \\ \gcd(x_1, y_1) = \gcd(\mathtt{x0}, \mathtt{y0}) \\ \downarrow \\ \gcd(x_2, y_2) = \gcd(\mathtt{x0}, \mathtt{y0}) \\ \vdots \\ \gcd(x_n, y_n) = \gcd(\mathtt{x0}, \mathtt{y0}) \end{array}$$

Motivation (I)

```
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 $\mathsf{x}_0 = \mathtt{x0} \wedge \mathsf{y}_0 = \mathtt{y0}$

forward invariant

$$\begin{gathered}
\gcd(\mathsf{x}_0, \mathsf{y}_0) = \gcd(\mathtt{x0}, \mathtt{y0}) \\
\downarrow \\
\gcd(\mathsf{x}_1, \mathsf{y}_1) = \gcd(\mathtt{x0}, \mathtt{y0}) \\
\downarrow \\
\gcd(\mathsf{x}_2, \mathsf{y}_2) = \gcd(\mathtt{x0}, \mathtt{y0}) \\
\vdots \\
\gcd(\mathsf{x}_n, \mathsf{y}_n) = \gcd(\mathtt{x0}, \mathtt{y0})
\end{gathered}$$

finally $\mathtt{xn} = \mathsf{x}_n = \mathsf{y}_n$

Motivation (I)

```
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 $\mathsf{x}_0 = \mathtt{x0} \wedge \mathsf{y}_0 = \mathtt{y0}$

forward invariant

$$\gcd(\mathsf{x}_0, \mathsf{y}_0) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\downarrow$$
$$\gcd(\mathsf{x}_1, \mathsf{y}_1) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\downarrow$$
$$\gcd(\mathsf{x}_2, \mathsf{y}_2) = \gcd(\mathtt{x0}, \mathtt{y0})$$
$$\vdots$$
$$\gcd(\mathsf{x}_n, \mathsf{y}_n) = \gcd(\mathtt{x0}, \mathtt{y0})$$

finally $\mathtt{xn} = \mathsf{x}_n = \mathsf{y}_n$

conclusion

$$\underbrace{\gcd(\mathsf{x}_n, \mathsf{y}_n)}_{\overset{\text{def}}{=}\mathtt{xn}} = \gcd(\mathtt{x0}, \mathtt{y0})$$

Invariants I

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

Invariants I

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

Invariants I

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

Invariants I

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

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));
```

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 $\mathsf{x}_n = \gcd(\mathsf{x}_1, \mathsf{y}_1)$
(loop from x1,y1 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 $\mathsf{x}_n = \gcd(\mathsf{x}_1, \mathsf{y}_1)$
(loop from x1,y1 is correct)

case $\mathsf{x}_0 > \mathsf{y}_0$

$$\begin{aligned} \mathsf{x}_n &= \gcd(\mathsf{x}_1, \mathsf{y}_1) \\ &= \gcd(\mathsf{x}_0 - \mathsf{y}_0, \mathsf{y}_0) \\ &\overset{\text{def}}{=} \gcd(\mathsf{x}_0, \mathsf{y}_0) \end{aligned}$$

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 x1,y1 is correct)

case $x_0 > y_0$

$$\begin{aligned} x_n &= \gcd(x_1, y_1) \\ &= \gcd(x_0 - y_0, y_0) \\ &\overset{\text{def}}{=} \gcd(x_0, y_0) \end{aligned}$$

case $x_0 \leq y_0$

$$\begin{aligned} x_n &= \gcd(x_1, y_1) \\ &= \gcd(x_0, y_0 - x_0) \\ &\overset{\text{def}}{=} \gcd(x_0, y_0) \end{aligned}$$

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 `x1`,`y1` is correct)

case $x_0 > y_0$

$$\begin{aligned} x_n &= \gcd(x_1, y_1) \\ &= \gcd(x_0 - y_0, y_0) \\ &\overset{\text{def}}{=} \gcd(x_0, y_0) \end{aligned}$$

case $x_0 \leq y_0$

$$\begin{aligned} x_n &= \gcd(x_1, y_1) \\ &= \gcd(x_0, y_0 - x_0) \\ &\overset{\text{def}}{=} \gcd(x_0, y_0) \end{aligned}$$

conclusion $x_n = \gcd(x_0, y_0)$
loop from `x0`,`y0` is correct

Invariants I + Summaries R

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

Invariants I + Summaries R

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

$$s_0 \dashrightarrow s_i \longrightarrow \underbrace{s_{i+1} \dashrightarrow s_n}_{\text{assume } R(s_{i+1}, s_n)}$$

Invariants I + Summaries R

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

$$s_0 \dashrightarrow \underbrace{s_i \longrightarrow \underbrace{s_{i+1} \dashrightarrow s_n}_{\text{assume } R(s_{i+1}, s_n)}}_{\text{prove } R(s_i, s_n)}$$

Invariants I + Summaries R

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

$$s_0 \dashrightarrow s_i \longrightarrow s_{i+1} \dashrightarrow s_n$$

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

prove $R(s_i, s_n)$

finally $R(s_0, s_n) \Rightarrow Q(s_n)$

Invariants I + Summaries R

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

$$\underbrace{s_0 \dashrightarrow \underbrace{s_i \longrightarrow \underbrace{s_{i+1} \dashrightarrow s_n}_{\text{assume } R(s_{i+1}, s_n)}}_{\text{prove } R(s_i, s_n)}}_{\text{finally } R(s_0, s_n) \Rightarrow Q(s_n)}$$

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

Comparison

invariant

$\gcd(\mathsf{x}, \mathsf{y}) = \gcd(\mathtt{x0}, \mathtt{y0})$

- characterizes reachable *states*
- propagates forward over iterations
- ✗ non-trivial to find

Comparison

invariant

$\mathrm{gcd}(\mathsf{x}, \mathsf{y}) = \mathrm{gcd}(\mathtt{x0}, \mathtt{y0})$

- characterizes reachable *states*
- propagates forward over iterations
- ✗ non-trivial to find

summary

$\mathsf{x}_n = \mathrm{gcd}(\mathsf{x}_i, \mathsf{y}_i)$

- summarizes remaining loop iterations as a *relation*
- propagates starting state backwards over iterations
- ✓ often similar to postcondition

Comparison

invariant	**summary**
$\gcd(\mathtt{x}, \mathtt{y}) = \gcd(\mathtt{x0}, \mathtt{y0})$	$\mathtt{x}_n = \gcd(\mathtt{x}_i, \mathtt{y}_i)$
▶ characterizes reachable *states*	▶ summarizes remaining loop iterations as a *relation*
▶ propagates forward over iterations	▶ propagates starting state backwards over iterations
✗ non-trivial to find	✓ 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 $\{P\}\ \mathbf{while}(t)\ B\ \{Q\}$

Verfication 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
  $$I(s) \land t(s) \land B(s, \bot) \implies \texttt{false}$$

Verfication Conditions for $\{P\}\ \mathbf{while}(t)\ B\ \{Q\}$

- 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
  $$I(s) \land t(s) \land B(s, \bot) \implies \texttt{false}$$
- Traditionally: Invariant ensures postcondition
  $$I(s) \land \neg t(s) \implies Q(s)$$

Verfication Conditions for $\{P\}$ $\mathbf{while}(t)$ 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
  $$I(s) \land t(s) \land B(s, \bot) \implies \texttt{false}$$
- 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 $\{P\}\ \mathbf{while}(t)\ B\ \{Q\}$ 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 $\{P\}\ \mathbf{while}(t)\ B\ \{Q\}$ holds.
If invariant I proves the body safe, then there a summary exists R that proves postcondition Q.

$\rightsquigarrow$ approach to decompose the verification

Proposition: Given invariant I and summary R,
then the corresponding regular invariant is
$\widehat{I}(s) \equiv I(s) \wedge \Big(\forall s_n.\ P(s_0) \wedge R(s, s_n) \implies R(s_0, s_n)\Big)$

$\rightsquigarrow$ encode/validate summaries in existing tools

Theoretical Results

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

$\rightsquigarrow$ approach to decompose the verification

Proposition: Given invariant I and summary R,
then the corresponding regular invariant is
$\widehat{I}(s) \equiv I(s) \wedge \Big(\forall s_n.\ P(s_0) \wedge R(s, s_n) \implies R(s_0, s_n)\Big)$

$\rightsquigarrow$ encode/validate summaries in existing tools

Proposition: Given invariant I, there is a canonical summary R
$\widehat{R}(s, s_n) \equiv \neg t(s_n) \wedge Q(s_n)$

$\rightsquigarrow$ can represent both approaches in a single tool

Experiments

Preliminary Implementation

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

Compare two settings

- invariants-only
- invariants + summaries[1]

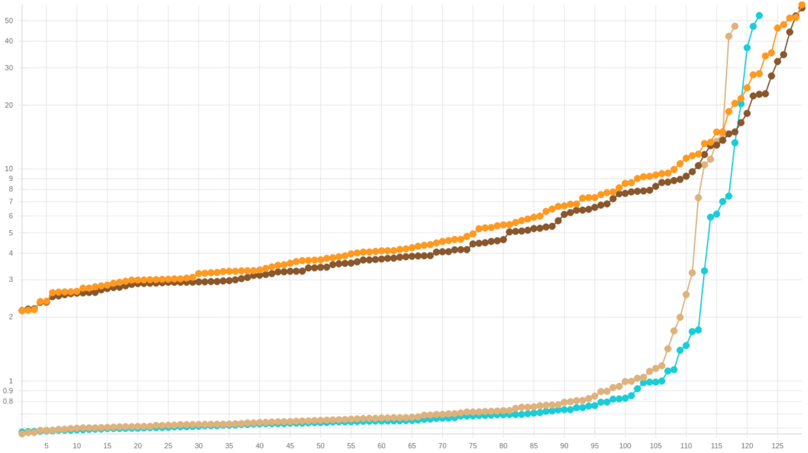
Hypothesis: invariants + summaries provides "more opportunity" to find solution

- 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	● invariants	● invariants+summary
Eldarica 2.0.4	● invariants	● invariants+summary

Discussion

Hypothesis: invariants + summaries provides "more opportunity" to find solution

✗ 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