Simplifying Loop Invariant Generation Using Splitter Predicates

R. Sharma, I. Dillig, T. Dillig, A. Aiken

Presented by Raphael Fuchs

Background

- Context: (Automatic) Program Verification
  - Floyd-Hoare logic {P} S {Q}
  - Often no specification given except for procedure pre-/postcondition
  - Encode program as logical formula, use SMT solvers to check consistency with specification
- Problem: Loops need invariants
  - Programmers might write them
  - **Invariant generation** preferable
  - Many tools and techniques exist
  - Here: Static code analysis

Motivation

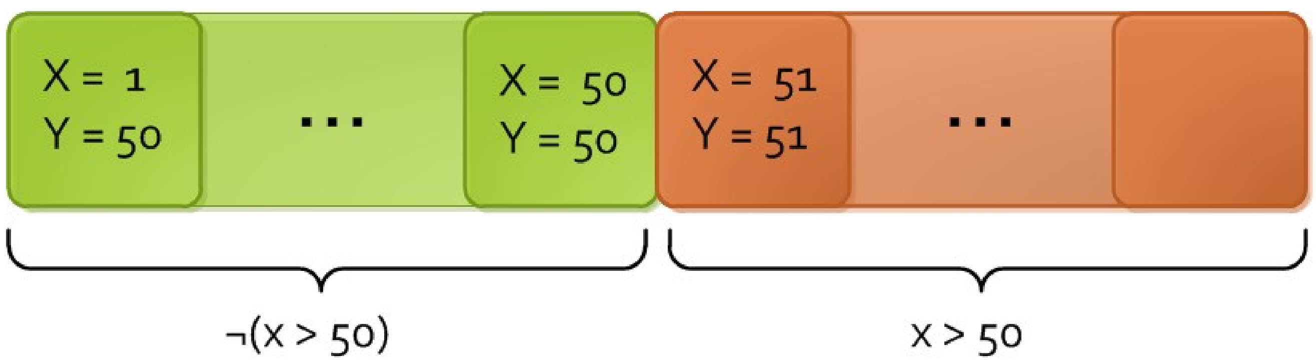
- Disjunctive invariants are difficult to infer!

```
x = 0;
y = 50;
while (x < 100) {
    // (x ≤ y ∧ y = 50) ∨ (50 ≤ x ≤ 100 ∧ y = x)
    x = x + 1;
    if (x > 50)
        y = y + 1;
}
assert (y == 100);
```

- OpenSSH study: ~10% of loops require disjunctive invariants

Multi-phase loops

- Loops with conditions (if-statements)
- Fixed number of **phase transitions**
  - **Phase**: sequence of iterations where condition evaluates to same value
  - Often 2 phases are enough, e.g. special first or last iteration.



- Common cause for disjunctive invariants

Contribution

- Idea: Transform loop to equivalent code with conjunctive invariants only.
- Then apply existing invariant generators

```
x = 0;
y = 50;
while (x < 100) {
    x = x + 1;
    if (x > 50)
        y = y + 1;
}
assert (y == 100);
```

```
x = 0; y = 50;
while (x <= 49) {
  // x ≤ y ∧ y = 50
  x = x + 1;
}
while (x < 100 && x > 49) {
  // 50 ≤ x ≤ 100 ∧ y = x
  x = x + 1;
  y = y + 1;
}
assert (y == 100);
```

(Phase) Splitter Predicates

Technique: We identify phase transitions with a **phase splitter predicate Q** with special properties:

1) Q must split loop into two

$$\texttt{while}(P)\{B\} \equiv$$
$$\texttt{while}(P \wedge \neg Q)\{B\}; \quad \texttt{while}(P \wedge Q)\{B\}$$

2) When Q is *true* (*false*) at entry, conditional C must always be *true* (*false*)

$$\texttt{while}(P)\{E[C]\} \equiv$$
$$\texttt{while}(P \wedge \neg Q)\{E[false]\}; \quad \texttt{while}(P \wedge Q)\{E[true]\}$$

Checking Splitter Predicates

- **Theorem:** Q is a phase splitter predicate for a loop $L = \texttt{while}(P)\{B[C]\}$ if the following holds:

$$\{P \wedge Q\}\, B[C]\, \{Q \vee \neg P\}$$

$$\{Q\}\, \overline{B} \qquad \{C\}$$

$$\{\neg Q\}\, \overline{B} \qquad \{\neg C\}$$

```
while (x < 100) {
    x = x + 1;
    if (x > 50)
        y = y + 1;
}
```

```
while (P) {
    B̅
    if (C)
        y = y + 1;
}
```

B

Splitting Algorithm

1. Find a candidate Q for some conditional C

   Q = WP($\overline{B}$, C) = WP(x=x+1, x > 50) =x > 49

   $$\{Q\}\,\overline{B}\,\{C\}$$

2. Check validity of $(\neg x > 49 \wedge x' = x + 1) \Rightarrow \neg x' > 50$

   $$\{\neg Q\}\,\overline{B}\,\{\neg C\}$$

3. Check $\{P \wedge Q\}\,B[C]\,\{Q \vee \neg P\}$

4. Split loop if successful or try another conditional

Example: Result

$P = x < 100$

$B = x = x + 1$

$C = x > 50$

$Q = WP(\overline{B}, C) = x > 49$

```
x = 0;
y = 50;
while (x < 100) {
    x = x + 1;
    if (x > 50)
        y = y + 1;
}
assert (y == 100);
```

```
x = 0; y = 50;
while (P && !Q) {
  x = x + 1;
}
while (P && Q) {
  x = x + 1;
  y = y + 1;
}
assert (y == 100);
```

Example: Result

P = x < 100

B = x = x + 1

C = x > 50

Q = WP($\overline{B}$, C) = x > 49

```
x = 0;
y = 50;
while (x < 100) {
    x = x + 1;
    if (x > 50)
        y = y + 1;
}
assert (y == 100);
```

```
x = 0; y = 50;
while (~~x < 100 &&~~ x <= 49) {
  x = x + 1;
}
while (x < 100 && x > 49) {
  x = x + 1;
  y = y + 1;
}
assert (y == 100);
```

Implementation

- Prototype using SAIL program analysis front-end, subset of C
- MISTRAL SMT solver: theory of linear arithmetic over integers
- 13 benchmarks from papers+tools run by INTERPROC and INVGEN generators
  - with and without this technique

#Verified	Before	After
INTERPROC	3	12
INVGEN	8	13

Questions?

Limitations

- Disjunctive invariant, no nested "if"

```
x=0;
while(x<n) {
  // n ≥ x ∨ n < 0
  x++;
}
if(n>0)
  assert(x==n);
```

- Not all loops with if-statements are multi-phase
  - But in case the if-condition relates to the iteration they often are!

- Efficiency? Many "C"s may be tried