

Constraint-based Invariant Inference

Invariants

- **Dictionary Meaning:** A function, quantity, or **property** which remains unchanged
- Property (in our context): a predicate that holds for some, all, or no states
- Invariant is a property of a program
 - at a specific program location
 - that holds for **every** program state that **reaches** the program point
- Specifications are invariants at exit points of programs or procedures
- Also called reachability properties.

Invariants

```
x = 0
```

```
y = n
```

```
while (y > 0) {
```

```
    x = x + 1
```

```
    y = y - 1
```

```
}
```

```
//invariant: x+y = n
```

```
//invariant: y>=0 => x<=n
```

Inductive Invariants

```
x = 0
```

```
y = n
```

```
//x+y = n
```

- Invariant holds initially

```
while (y > 0) {
```

```
    //x+y = n  $\wedge$  y > 0
```

```
    x = x + 1
```

```
    //x+y = n+1
```

- Invariant holds at the start of the loop

=>

```
    y = y - 1
```

invariant holds at the end of the loop

```
    //x+y = n
```

```
}
```

```
//invariant: x+y = n
```

Not all Invariants are Inductive

```
x = 0
```

```
y = n
```

```
//y>=0 => x<=n
```

```
while (y > 0) {
```

```
    //x <= n  $\wedge$  y > 0
```

```
    x = x + 1
```

```
    //x <= n+1  $\wedge$  y > 0
```

```
    y = y - 1
```

```
    //x <= n+1  $\wedge$  y >= 0
```

```
}
```

```
//invariant: y>=0 => x <= n
```



Invariant cannot be proved by induction

Inductive Strengthening

```
x = 0
y = n
// (y >= 0 => x <= n) ∧ x + y = n
while (y > 0) {
  // x < n ∧ y > 0 ∧ x + y = n
  x = x + 1
  // x <= n ∧ y > 0 ∧ x + y = n + 1
  y = y - 1
  // x <= n ∧ y >= 0 ∧ x + y = n
}
//invariant: y >= 0 => x <= n
```

Implied by the
stronger inductive
invariant



Formulating Inductiveness

```
x = 0
y = n
while (y > 0) {
  x = x + 1
  y = y - 1
} //invariant: y >= 0 => x <= n
```

Generally referred to as the verification condition (VC)

$$(x = 0 \wedge y = n) \Rightarrow (y < 0 \vee x \leq n)$$

$$\begin{array}{c} \text{I} \qquad \text{Guard} \qquad \text{Transition} \\ \underbrace{\hspace{10em}} \quad \underbrace{\hspace{3em}} \quad \underbrace{\hspace{15em}} \\ ((y < 0 \vee x \leq n) \wedge y > 0 \wedge x' = x + 1 \wedge y' = y - 1) \\ \Rightarrow (y' < 0 \vee x' \leq n) \end{array}$$

Formulating Inductive Strengthening

```
x = 0
y = n
while (y > 0) {
  x = x + 1
  y = y - 1
} //invariant: y >= 0 => x <= n
```

$$(x = 0 \wedge y = n) \Rightarrow (y < 0 \vee x \leq n) \wedge \mathbf{S}$$

The diagram illustrates the decomposition of the inductive strengthening formula into three parts: **I** (Invariant), **Guard**, and **Transition**. Brackets above the formula group the terms as follows: **I** covers $(y < 0 \vee x \leq n) \wedge \mathbf{S}$; **Guard** covers $y > 0$; and **Transition** covers $x' = x + 1 \wedge y' = y - 1$.

$$\begin{aligned} & \underbrace{((y < 0 \vee x \leq n) \wedge \mathbf{S})}_{\mathbf{I}} \wedge \underbrace{y > 0}_{\text{Guard}} \wedge \underbrace{x' = x + 1 \wedge y' = y - 1}_{\text{Transition}} \\ & \Rightarrow (y' < 0 \vee x' \leq n) \wedge \mathbf{S}' \end{aligned}$$

Finding Linear Invariants

[Colon et al. CAV '03]

```
x = 0
y = n
while (y > 0) {
  x = x + 1
  y = y - 1
} //invariant: y >= 0 => x <= n
```

Perhaps could be
called a parametric
VC

$$(x = 0 \wedge y = n) \Rightarrow (y < 0 \vee x \leq n) \wedge \mathbf{ax} + \mathbf{by} + \mathbf{c} \leq \mathbf{0}$$

$$\begin{array}{c} \mathbf{I} \qquad \qquad \qquad \mathbf{Guard} \qquad \qquad \mathbf{Transition} \\ \underbrace{\hspace{15em}} \qquad \underbrace{\hspace{5em}} \qquad \underbrace{\hspace{35em}} \\ (y < 0 \vee x \leq n) \wedge \mathbf{ax} + \mathbf{by} + \mathbf{c} \leq \mathbf{0} \wedge y > 0 \wedge x' = x + 1 \wedge y' = y - 1 \\ \Rightarrow (y' < 0 \vee x' \leq n) \wedge \mathbf{ax}' + \mathbf{by}' + \mathbf{c} \leq \mathbf{0} \end{array}$$

Finding Template Coefficients

$$(x \geq 0 \wedge y \geq n) \Rightarrow ax + by + c < 0$$

Find values for a,b,c
s.t. the formula
becomes valid

$$A \Rightarrow B \equiv \neg(A \wedge \neg B)$$



$$x \geq 0 \wedge y \geq n \wedge ax + by + c \geq 0$$

Find values for a,b,c
s.t. the formula
becomes unsatisfiable

Farkas' Lemma: A conjunction of linear inequalities is unsatisfiable iff we can derive $\mathbf{1} \leq \mathbf{0}$ by performing the following operations:

- Multiplying the inequalities by a non-negative constant
- Adding two inequalities
- Adding (or subtracting) a non-negative constant to one side

Farkas' Lemma Example

$$x \geq 0 \wedge y \geq n \wedge 2x + 2y - 2n + 3 \leq 0$$

$$x + 0y + 0n + 0 \geq 0$$

$$0x + y - n + 0 \geq 0$$

$$-2x - 2y + 2n - 3 \geq 0$$

Multiply first and second equations by 2,

Add 2 to RHS of last equation

and add them

$$-1 \geq 0$$

Farkas' Lemma: A conjunction of linear inequalities (over reals) is unsatisfiable iff we can derive $\mathbf{1} \leq \mathbf{0}$ by performing the following operations:

- Multiplying the inequalities by a non-negative constant
- Adding two inequalities
- Adding (or subtracting) a non-negative constant to one side

Automating Coefficient Finding

$$x \geq 0 \wedge y - n \geq 0 \wedge 2x + 2y - 2n + 3 \leq 0 \quad \leftarrow \text{Prove unsat}$$



$$\lambda_1 x \geq 0$$

Multiplying by unknown non-negative values

$$\lambda_2 y - \lambda_2 n \geq 0$$

$$-2\lambda_3 x - 2\lambda_3 y + 2\lambda_3 n - 3\lambda_3 \geq 0$$



Adding the inequalities

$$(\lambda_1 - 2\lambda_3)x + (\lambda_2 - 2\lambda_3)y + (2\lambda_3 - \lambda_2)n - 3\lambda_3 \geq 0$$



Adding an unknown non-neg value

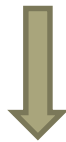
$$(\lambda_1 - 2\lambda_3)x + (\lambda_2 - 2\lambda_3)y + (2\lambda_3 - \lambda_2)n - 3\lambda_3 + \lambda \geq 0$$

$$\equiv -1 \geq 0$$

Equate to $1 \leq 0$

Automating Coefficient Finding [Cont.]

$$(\lambda_1 - 2\lambda_3)x + (\lambda_2 - 2\lambda_3)y + (2\lambda_3 - \lambda_2)n - 3\lambda_3 + \lambda \geq 0$$
$$\equiv -1 \geq 0$$



$$\begin{aligned}\lambda_1 - 2\lambda_3 &= 0 \\ \lambda_2 - 2\lambda_3 &= 0 \\ 2\lambda_3 - \lambda_2 &= 0 \\ -3\lambda_3 + \lambda &= -1\end{aligned}$$



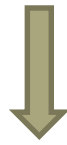
Every solution for
the constraints will
make the inequalities
unsatisfiable

$$\begin{aligned}\lambda_1 &= 2, \lambda_2 = 2, \\ \lambda_3 &= 1, \lambda = 2\end{aligned}$$

Template-based Invariant Inference

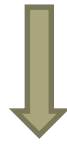
$$x \geq 0 \wedge y - n \geq 0 \wedge ax + by + c \geq 0$$

Find values for a,b,c
s.t. the formula
becomes unsatisfiable



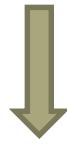
$$\begin{aligned}\lambda_1 x &\geq 0 \\ \lambda_2 y - \lambda_2 n &\geq 0 \\ \lambda_3 ax + \lambda_3 by + \lambda_3 c &\geq 0\end{aligned}$$

Multiplying by unknown non-negative values



$$(\lambda_1 + \lambda_3 a)x + (\lambda_2 + \lambda_3 b)y - \lambda_2 n + \lambda_3 c \geq 0$$

Adding the inequalities



$$(\lambda_1 + \lambda_3 a)x + (\lambda_2 + \lambda_3 b)y - \lambda_2 n + \lambda_3 c + \lambda_4 \geq 0$$

Adding an unknown non-neg value

$$\equiv -1 \geq 0$$

Equate to $1 \leq 0$

Farkas' Constraints [Cont.]

$$(\lambda_1 + \lambda_3 a)x + (\lambda_2 + \lambda_3 b)y - \lambda_2 n + \lambda_3 c + \lambda_4 \geq 0$$
$$\equiv -1 \geq 0$$



$$\begin{aligned}\lambda_1 + \lambda_3 a &= 0 \\ \lambda_2 + \lambda_3 b &= 0 \\ -\lambda_2 &= 0 \\ \lambda_3 c + \lambda_4 &= -1\end{aligned}$$



Every solution for the constraints will make the inequalities unsatisfiable

$$\begin{aligned}b &= 0, a = -1, c = -1, \\ \lambda_1 &= 1, \lambda_2 = 0, \\ \lambda_3 &= 1, \lambda_4 = 0\end{aligned}$$

In summary

- We had a formula of the form: $A[\mathbf{x}] \wedge B[\mathbf{a}, \mathbf{x}] \Rightarrow C[\mathbf{a}, \mathbf{x}]$
- We wanted to find a value for \mathbf{a} that will make the implication hold for all \mathbf{x}
- In other words, we are trying to find a satisfiable assignment for a quantified formula.
- Farkas' Lemma converts it to satisfiability of quantifier-free non-linear real constraints

Limitations

The Farkas' Lemma approach provides a way to find linear invariants for programs that

- do not have many disjunctions
- do not have functions
- do not have data structures
- do not have nonlinear arithmetic

Further Reading and Software

We developed an approach that addresses some of these limitations.

For more details see:

“Symbolic Resource Bounds Inference For Functional Programs”, CAV 2014: [pdf](#), [slides](#)

An extension of Leon (a slightly old version) that supports templates:

Orb : <http://lara.epfl.ch/w/rbound>

- More Related Works
 - “Linear invariant generation using non-linear constraint solving.”, Colon et al., CAV 2003
 - “Program analysis as constraint solving.”, S. Gulwani et al., PLDI 2008
 - “Constraint solving for interpolation.”, A.Rybalchenko et al., VMCAI 2007
 - “Non-linear loop invariant generation using grobner bases.” Sankaranarayanan et al., POPL 2004