FAKULTÄT
FÜR INFORMATIK

Faculty of Informatics

Bound Analysis of Imperative Programs with the Size-change Abstraction

Florian Zuleger, TU Vienna

SAS, Venice, 16.09.2011

Joint work with
Sumit Gulwani, Microsoft Research
Moritz Sinn, Helmut Veith, TU Vienna

Resource Bounds

Programs consume a variety of resources:

- CPU time, memory, network bandwidth, power

Bounding the use of such resources is important:

- Economic incentives
- Better user experience
- Hard constraints on availability of resources

Program correctness depends on bounding quantitative properties of data:

- Information leakage, Propagation of numerical errors

The Reachability-bound Problem

(Gulwani, Zuleger, PLDI 2010)

Given a control location l inside a program P.

How often can l be visited inside P?

```
void main (int n, C[] temp) {
int i := 0;
while (i < n) {
   int j := i+1;
   while (j < n) {
      if (nondet()) {
       l:  temp[n] := new C();
           j--; n--; }
      j++; }
   i++; }
}
```

Goal: A symbolic bound Bound(l) in terms of the inputs of P.

Bound Computation and Termination

A bound for a loop implies the termination of the loop.

$\Rightarrow$ Computing bounds is more difficult than proving termination!

Can successfull techniques for termination analysis be extended to bound analysis?

What about

- Size-change Abstraction (Ben-Amram, Lee, Jones, 2001)

Yes!

- Transition Invariants (Podelski, Rybalchenko, 2004)?

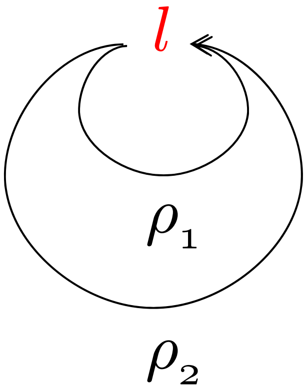
Not so easy…

Outline

1. Introduction
2. Comparing SCA with Transition Invariants
3. SCA solves Technical Challanges
4. How to apply SCA on Imperative Programs

Size-change Abstraction (SCA)

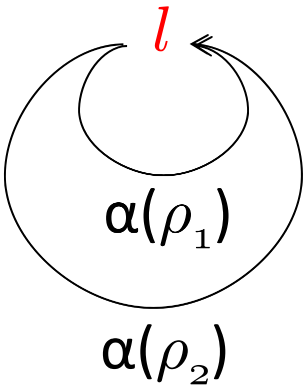
```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else
        y--;
   }
```



$$\rho_1 \equiv x > 0 \wedge y > 0 \wedge x\text{-}1 = x' \wedge y = y'$$
$$\rho_2 \equiv x > 0 \wedge y > 0 \wedge x = x' \wedge y\text{-}1 = y'$$

Size-change Abstraction (SCA)

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else
        y--;
   }
```

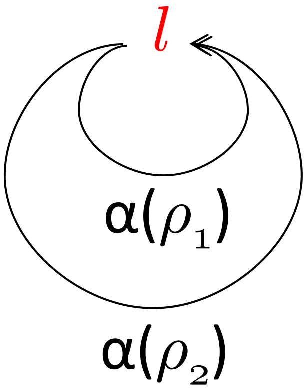


$$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge y = y'$$
$$\alpha(\rho_2) \equiv x > 0 \wedge y > 0 \wedge x = x' \wedge y > y'$$

Predicate abstract domain consisting of inequalities between integer variables (primed and unprimed)

Size-change Abstraction (SCA)

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
      if (nondet())
         x--;
      else
         y--;
   }
```

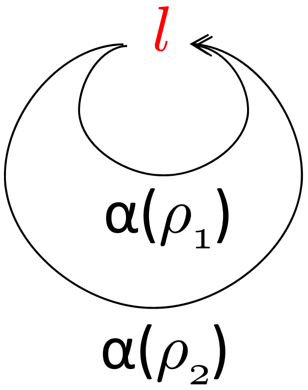


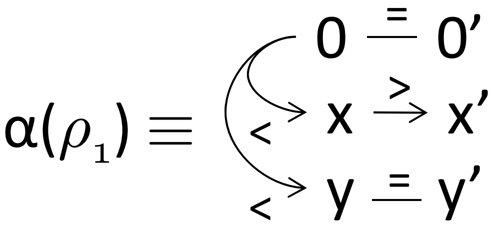
$$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge y = y'$$
$$\alpha(\rho_2) \equiv x > 0 \wedge y > 0 \wedge x = x' \wedge y > y'$$

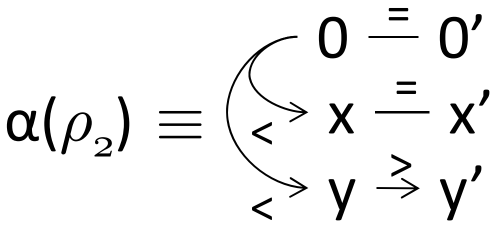
Finite powerset abstract domain whose base elements are conjuncts of inequalities between integer variables (primed and unprimed)

Size-change Abstraction (SCA)

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
       if (nondet())
          x--;
       else
          y--;
   }
```







Control flow graph whose edges are labeled by size-change graphs

SCA is a Success Story

- Termination is decidable in PSPACE (Ben-Amram, Lee, Jones, 2001; Ben-Amram,2011)
- Complete method for extracting ranking functions on terminating instances (possibly exponentially large)

- SCA based termination analysis is implemented in widely-used systems such as ACL2, Isabelle, AProVE
- The industrial-strength tool ACL2 can automatically prove the termination of 98% of the functions in its database

Good Computational Properties

- Enjoys built-in disjunction.
- Transitive hulls can be computed without overapproximation techniques such as widening.
- Transitive hulls preserve termination.
- Abstraction can be done by SMT solver calls.

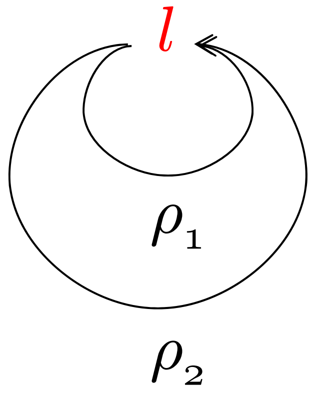
$\Rightarrow$ **Potential for automation**

Transition Invariants

- Have been developed as adaption of SCA to imperative programs
- Experimentally proven useful on device drivers in the Terminator tool; see Cook et al. 2006
- More general than SCA; for formal comparison see Heizmann et al. 2011

Transition Invariants

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else
        y--;
   }
```



$$\rho_1 \equiv x > 0 \wedge y > 0 \wedge x\text{-}1 = x' \wedge y = y'$$
$$\rho_2 \equiv x > 0 \wedge y > 0 \wedge x = x' \wedge y\text{-}1 = y'$$

Termination proof: $(\rho_1 \cup \rho_2)^+ \subseteq T_1 \cup T_2$,

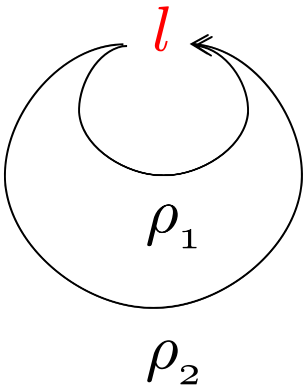
where $T_1 \equiv x > 0 \wedge x' = x\text{-}1$

and $T_2 \equiv y > 0 \wedge y' = y\text{-}1$

Transition Invariants

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
```

Well-founded relations



$\rho_1 \equiv x > 0 \wedge y > 0 \wedge x\text{-}1 = x' \wedge y = y'$

$\rho_2 \equiv x > 0 \wedge y > 0 \wedge x = x' \wedge y\text{-}1 = y'$

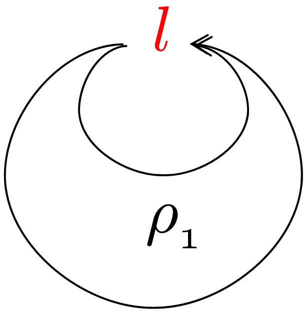
Terminat n proof: $(\rho_1 \cup \rho_2)^+ \subseteq T_1 \cup T_2$,

where $T_1 \equiv x > 0 \wedge x' = x\text{-}1$

and $T_2 \equiv y > 0 \wedge y' = y\text{-}1$

Transition Invariants

```
void main (int n) {
  int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
```



Well-founded relations

x and y are local ranking functions

$\rho_1 \equiv x > 0 \wedge y > 0$

$\rho_2 \equiv x > 0 \wedge y > 0$

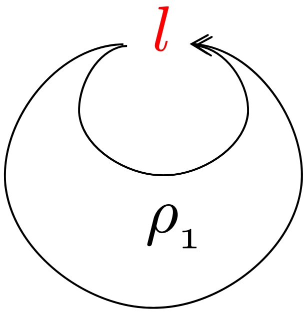
Terminat n proof: $(\rho_1 \cup \rho_2)^+ \subseteq {}_1 \cup T_2$,

where $T_1 \equiv x > 0 \wedge x' = x-1$

and $T_2 \equiv y > 0 \wedge y' = y-1$

Transition Invariants

```
void main (int n) {
  int x=n; int y
l: while (x>0 ∧
     if (nondet
       x--;
```



Transitive hull in the concrete

Well-founded relations

x and y are local ranking functions

$\rho_1 \equiv x$ 0 $\wedge$ y > 0

$\rho_2 \equiv x >$ 0 $\wedge$ y > 0

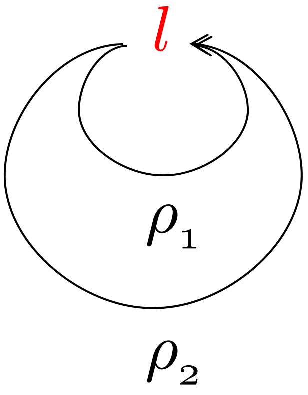
Terminat n proof: $(\rho_1 \cup \rho_2)^+ \subseteq {}_1 \cup T_2$,

where $T_1 \equiv x > 0 \wedge x' = x-1$

and $T_2 \equiv y > 0 \wedge y' = y-1$

Transition Invariants

```
void main (int n) {
   int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else {
        x := n;
        y--; } }
```



$$\rho_1 \equiv x > 0 \land y > 0 \land x\text{-}1 = x' \land y = y'$$

$$\rho_2 \equiv x > 0 \land y > 0 \land \mathbf{n = x'} \land y\text{-}1 = y'$$

Termination proof: $(\rho_1 \cup \rho_2)^+ \subseteq T_1 \cup T_2$,

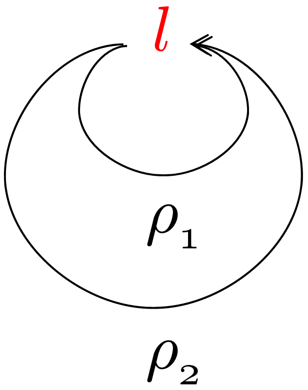
where $T_1 \equiv x > 0 \land x' = x\text{-}1$

and $T_2 \equiv y > 0 \land y' = y\text{-}1$

Transition Invariants

```
void main (int n) {
    int
l: whil
        i      (    det())
           x-  ;
        else  {
           x := n;
           y--; } }
```

We reset x to n!



$\rho_1 \equiv x > 0 \wedge y > 0 \wedge x\text{-}1 = x' \wedge y = y'$

$\rho_2 \equiv x > 0 \wedge y > 0 \wedge \mathbf{n = x'} \wedge y\text{-}1 = y'$

Termination proof: $(\rho_1 \cup \rho_2)^+ \subseteq T_1 \cup T_2$,

where $T_1 \equiv x > 0 \wedge x' = x\text{-}1$

and $T_2 \equiv y > 0 \wedge y' = y\text{-}1$

Transition Invariants

```
void main (int n) {
    int
l: whil
       i     (     det())
          x   ;
       else   {
          x := n;
          y--; } }
```

We reset x to n!

Same termination proof!

$_1 \equiv x > 0 \wedge y > 0 \wedge x\text{-}1 = x' \wedge y = y'$

$\rho_2 \equiv x > 0 \wedge y > 0 \wedge n = x' \wedge y\text{-}1 = y'$

Termination proof: $(\rho_1 \cup \rho_2)^+ \subseteq T_1 \cup T_2$,

where $T_1 \equiv x > 0 \wedge x' = x\text{-}1$

and $T_2 \equiv y > 0 \wedge y' = y\text{-}1$

Transition Invariants

```
void main (int n) {
    int
l: whil
      i   (    det())
        x-  ;
      else  {
        x := n;
        y--; } }
```

We reset x to n!

Same termination proof!

$\rho_1 \equiv$ x > 0 $\wedge$ y > 0 $\wedge$ x-1 … y = y‘

$\rho_2 \equiv$ x > 0 $\wedge$ … = y‘

Termination … $\rho_2)^+ \subseteq T_1 \cup T_2$,

$T_1 \equiv$ x > 0 $\wedge$ x‘ = x-1

and $T_2 \equiv$ y > 0 $\wedge$ y‘ = y-1

Transition Invariants are too imprecise!

Size-change Abstraction (SCA)

```
void main (int n) {
   int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else
        y--;
   }
```

$$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge y = y'$$

$$\alpha(\rho_2) \equiv x > 0 \wedge y > 0 \wedge \mathbf{x = x'} \wedge y > y'$$

```
void main (int n) {
   int x=n; int y=n;
l: while (x>0 ∧ y>0) {
     if (nondet())
        x--;
     else {
        x := n;
        y--; } }
```

$$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge y = y'$$

$$\alpha(\rho_2) \equiv x > 0 \wedge y > 0 \wedge \mathbf{n = x'} \wedge y > y'$$

Size-change Abstraction (SCA)

```
void main (int n) {
    int x=n; int y=n;
l: while (x>0 ∧ y>0) {
      if (nondet())
         x--;
      else
```

```
void main (int n) {
    int x=n; int y=n
l: while (
         y--; } }
```

SCA keeps more information in the abstract than

$T_1 \equiv x > 0 \wedge x' = x\text{-}1$

$T_2 \equiv y > 0 \wedge y' = y\text{-}1$!

$x > x' \wedge y = y'$

$\alpha(\rho_2) \equiv x > 0 \wedge y > 0$

$\wedge\ \mathbf{x = x'} \wedge y > y'$

$\alpha(\rho_1) \equiv x > 0 \wedge y > 0$

$\wedge\ x > x' \wedge y = y'$

$\alpha(\rho_2) \equiv x > 0 \wedge y > 0$

$\wedge\ \mathbf{n = x'} \wedge y > y'$

Bounds by SCA

$$\begin{aligned}\alpha(\rho_1) &\equiv x > 0 \land y > 0 \\ &\quad \land x > x' \land y = y' \\ \alpha(\rho_2) &\equiv x > 0 \land y > 0 \\ &\quad \land \mathbf{x = x'} \land y > y'\end{aligned}$$

$$\begin{aligned}\alpha(\rho_1) &\equiv x > 0 \land y > 0 \\ &\quad \land x > x' \land y = y' \\ \alpha(\rho_2) &\equiv x > 0 \land y > 0 \\ &\quad \land \mathbf{n = x'} \land y > y'\end{aligned}$$

Our bound algorithm for SCA:

- uses only the abstracted transitions
- discovers x and y as norms by heuristics

x and y stay constant on the respective other transition

Our tool computes the ranking function x+y, which results in Bound(l) = 2n

only x is increased on the other transition

Our tool computes the ranking function (x,y), which results in Bound(l) = n^2

Outline

1. Introduction
2. Comparing SCA with Transition Invariants
3. SCA solves Technical Challanges
4. How to apply SCA on Imperative Programs

SCA solves Technical Challanges

We do not use SCA because we like the formalism, but because we believe that

SCA is the right abstraction for the bound analysis of imperative programs.

Technical Challenges

I. Bounds are often non-linear expressions

II. Proving a bound often requires disjunctive invariants

III. Bounds cannot be predicted by templates

IV. How to exploit program structure for bound computation is unclear

Bounds by SCA

$\alpha(\rho_1) \equiv x > 0 \land y > 0$
$\land\ x > x' \land\ y = y'$
$\alpha(\rho_2) \equiv x > 0 \land y > 0$
$\land\ \mathbf{x = x'} \land\ y > y'$

Our bound alg

- U

es the
g function x+y, which
results in Bound(l) = 2n

$\alpha(\rho_1) \equiv x > 0 \land y > 0$
$\land\ x > x' \land\ y = y'$
$\alpha(\rho_2) \equiv x > 0 \land$

increased on
the other transition

Our tool computes the
ranking function (x,y), which
results in Bound(l) = n^2

III. Bounds cannot be predicted by templates
Extract norms locally and compose them to a global bound

Bounds by SCA

$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge y = y'$

$\alpha(\rho_2) \equiv x > 0 \wedge y > 0 \wedge \mathbf{x = x'} \wedge y > y'$

Our boun

on x+y, which

lts in Bound(l) = 2n

$\alpha(\rho_1) \equiv x > 0 \wedge y > 0 \wedge x > x' \wedge$

$\alpha(\rho_2) \equiv$

ed on

her transition

Our tool computes the ranking function (x,y), which results in Bound(l) = n^2

I. Bounds are often non-linear expressions

Upper bounds on bound constituents derived from global invariants computed by standard abstract domains

Bounds by SCA

$\alpha(\rho_1) \equiv x > 0 \wedge y > 0$
$\wedge\ x > x' \wedge\ y = y'$
$\alpha(\rho_2) \equiv x > 0 \wedge y > 0$
$\wedge\ \mathbf{x = x'} \wedge\ y > y'$

Our bound algorit

- use

C e the
ra unction x+y, which
results in Bound(l) = 2n

$\alpha(\rho_1) \equiv x > 0 \wedge y > 0$
$\wedge\ x > x' \wedge\ y = y'$
$\alpha(\rho_2) \equiv x > 0 \wedge$

creased on
e other transition

Our tool computes the
ranking function (x,y), which
results in Bound(l) = n^2

II. Proving a bound often requires disjunctive invariants

Disjunctive analysis by path enumeration

Outline

1. Introduction
2. Comparing SCA with Transition Invariants
3. SCA solves Technical Challanges
4. How to apply SCA on Imperative Programs

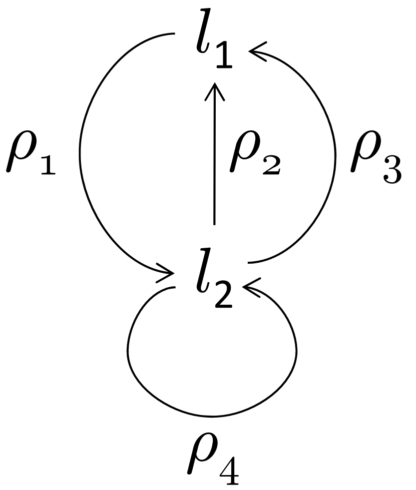
How to apply SCA on Imperative Programs

I. Transition System Generation by Pathwise Analysis

II. Heuristics for Extracting Norms

III. Dealing with Control Structure of Loops by Contextualization

Transition System Generation by Pathwise Analysis



$\rho_1 \equiv i < n \wedge i' = i + 1 \wedge j' = 0$

$\rho_2 \equiv j > 0 \wedge i' = i - 1$

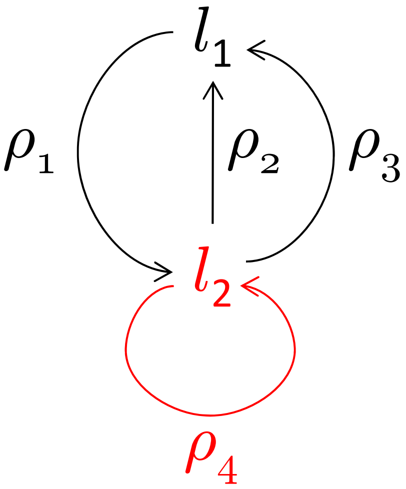
$\rho_3 \equiv j \leq 0$

$\rho_4 \equiv i < n \wedge i' = i + 1 \wedge j' = j + 1$

Goal: Transition System for l_1

Idea: Enumerating all paths from l_1 to l_1

Transition System Generation by Pathwise Analysis



$\rho_1 \equiv i < n \wedge i' = i + 1 \wedge j' = 0$

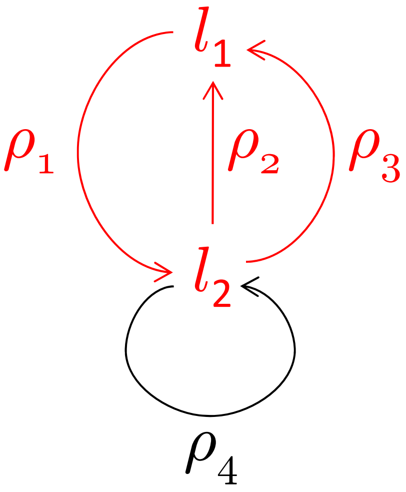
$\rho_2 \equiv j > 0 \wedge i' = i - 1$

$\rho_3 \equiv j \leq 0$

$\rho_4 \equiv i < n \wedge i' = i + 1 \wedge j' = j + 1$

We first summarize the inner loop.

Transition System Generation by Pathwise Analysis



$\rho_1 \equiv i < n \wedge i' = i + 1 \wedge j' = 0$

$\rho_2 \equiv j > 0 \wedge i' = i - 1$

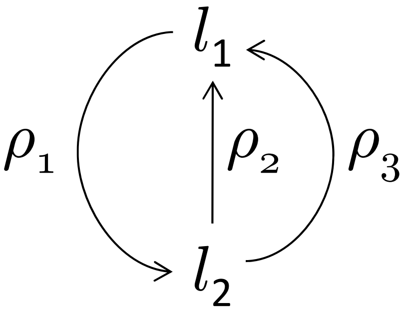
$\rho_3 \equiv j \leq 0$

$\rho_4 \equiv i < n \wedge i' = i + 1 \wedge j' = j + 1$

We first summarize the inner loop.

Using this summary we compute a transitition system for the outer loop.

Transition System of the Outer Loop



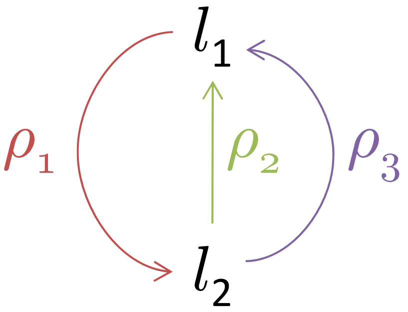
Summary[l_2] = $\{\rho_{4a}, \rho_{4a}\}$

$$\rho_1 \equiv i < n \wedge i' = i + 1 \wedge j' = 0$$
$$\rho_2 \equiv j > 0 \wedge i' = i - 1$$
$$\rho_3 \equiv j \leq 0$$
$$\rho_{4a} \equiv i' = i \wedge j' = j$$
$$\rho_{4b} \equiv i < n \wedge i' > i \wedge j' > j$$

We obtain Summary[l_2] by:

1. Recursively computing a transition system for the inner loop and size-change abstracting it.
2. Computing the transitive hull using SCA.

Transition System of the Outer Loop



Summary[l_2] = $\{\rho_{4a}, \rho_{4a}\}$

$\rho_1 \equiv i < n \land i' = i + 1 \land j' = 0$

$\rho_2 \equiv j > 0 \land i' = i - 1$

$\rho_3 \equiv j \leq 0$

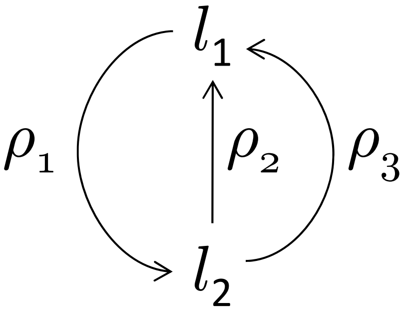
$\rho_{4a} \equiv i' = i \land j' = j$

$\rho_{4b} \equiv i < n \land i' > i \land j' > j$

We obtain a transition system for l_1 by enumerating all paths using the different disjuncts of the summary:

$\{\rho_1 \circ \rho_{4a} \circ \rho_2,$ $\rho_1 \circ \rho_{4a} \circ \rho_3$,
$\rho_1 \circ \rho_{4b} \circ \rho_2,$ $\rho_1 \circ \rho_{4b} \circ \rho_3\}$

Transition System Generation by Pathwise Analysis



Summary$[l_2] = \{\rho_{4a}, \rho_{4a}\}$

$$\rho_1 \equiv i < n \land i' = i + 1 \land j' = 0$$
$$\rho_2 \equiv j > 0 \land i' = i - 1$$
$$\rho_3 \equiv j \leq 0$$
$$\rho_{4a} \equiv i' = i \land j' = j$$
$$\rho_{4b} \equiv i < n \land i' > i \land j' > j$$

We obtain a transition system for l_1 by enumerating all paths using the different disjuncts of the summary:

$\{\text{false}, \quad i < n \land i' = i + 1 \land j' = 0,$
$\;i < n \land i' > i \land j' > 0, \quad \text{false}\}$
$= \{i < n \land i' = i + 1 \land j' = 0, i < n \land i' > i \land j' > 0\}$

Discussion of Pathwise Analysis

- Pathprecise reasoning: abstraction or infeasibility analysis of *complete paths*
- Leverages the progress in SMT solver technology to static analysis
- Generalization of classical SCA
- *More precise than blockwise analysis*

Discussion of Pathwise Analysis

- Pathprecise reasoning: abstra
  infeasibility analysi
- Lev
-
- *M* *than blockwise analysis*

IV. How to exploit program structure for bound computation is unclear

Pathwsie analysis exploits the loop structure of imperative programs

How to apply SCA on Imperative Programs

I. Transition System Generation by Pathwise Analysis
II. Heuristics for Extracting Norms
III. Dealing with Control Structure of Loops by Contextualization

Norms

Given some transition system, its set of norms is the union of the norms of all its transitions.

Let ρ be the formula of some transition.

If the inequality $e_1 \geq e_2$ syntactically appears in ρ, then $e_1 - e_2$ is a **candidate** for an arithmetic norm.

We check with an SMT solver for each candidate e:

If $\rho \Rightarrow e[X'/X] \leq e-1$, then e is a **norm**.

Example

$i' \geq i+1 \wedge i<n$

$n-i$

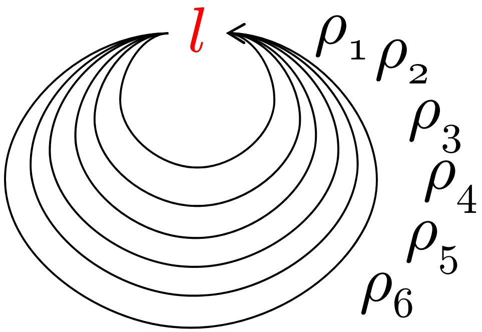
This pattern-based technique readily extends to non-arithmetic norms:
Proof rules for bitvectors and data-structures can be found in Gulwani, Zuleger, 2010.

How to apply SCA on Imperative Programs

I. Transition System Generation by Pathwise Analysis
II. Heuristics for Extracting Norms
III. Dealing with Control Structure of Loops by Contextualization

Contextualization

Computed transition system:



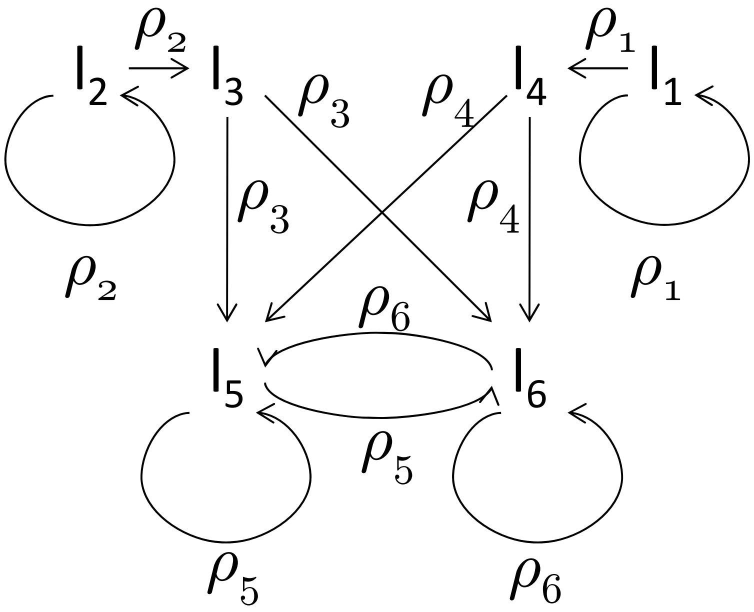
The first step in bound analysis is the construction of a program such that

- every location stores the information what transition is executed next, and
- only feasible transitions are added.

Construction is done by SMT solver queries.

Contextualization

Contextualized transition system:

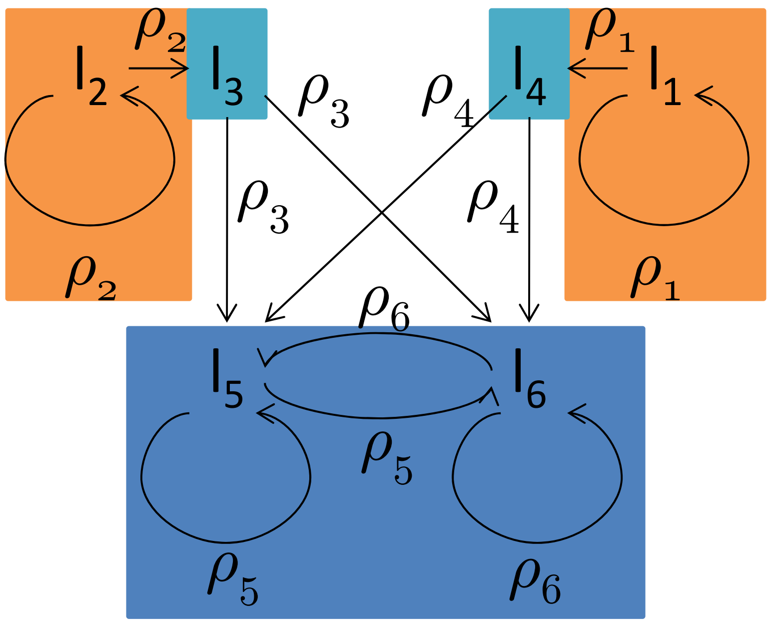


The first step in bound analysis is the construction of a program such that

- every location stores the information what transition is executed next, and
- only feasible transitions are added.

Construction is done by SMT solver queries.

DAG of SCCs



The CFG can be decomposed into its DAG of SCCs.

$\Rightarrow$ Uncovers the control structure of the loop.

Bounds are computed in two steps:

1. Bounds are computed for every SCC in isolation
2. These bounds are composed to an overall bound using the DAG structure.

Loopus

- Built over LLVM Compiler Framework, inputs C source code
- Uses Yices solver as the logical reasoning engine.
- Aliasing was handled using optimistic assumptions.
- 4090 of 4302 loops of the cBench benchmark handled in less than 1000 seconds (3923 loops in less than 4 seconds)
- Success ratio of 75% for computing loop bounds.
- Representative failure cases:
  - Insufficient invariant analysis
  - Memory updates and pointer arithmetic
  - Irreducible CFGs not implemented
  - Loops that are not meant to terminate
  - Complex invariants would be needed

Conclusion

Size-change Abstraction is the right abstraction for bound analyis of imperative programs:

- We have given the first algorithm for computing bounds with SCA
- We have shown how to apply SCA to imperative programs

Questions?