# An Abstract Domain to Infer Symbolic Ranges over Nonnegative Parameters

Xueguang Wu    Liqian Chen    Ji Wang

National University of Defense Technology, Changsha, China

10/09/2014 – NSAD 2014

# Overview

# Motivation

# Value range analysis

Range: $x \in [a, b]$ (denoting $a \leq x \leq b$)

- the lower & upper bound of the values that a variable may take
- applications: compiler optimization, automatic parallelization, bug detection, etc.

Numeric range

- bounds: **numeric** constants
- E.g., $1 \leq x \leq 3$

Symbolic range

- bounds: symbolic **expressions** over program variables except $x$
- E.g., $n \leq x \leq 2n + 3m$

# Value range analysis by abstract interpretation

Value range analysis :

- **goal**: to automatically infer a range $[a, b]$ for each program variable $x$ at compile time

Theoretical framework: **abstract interpretation**

- to design static analyses that are sound by construction
  (no behavior is omitted)
  $\rightsquigarrow$ over-approximate ranges

Example: the interval abstract domain [Cousot Cousot 76]

- infer the numeric range information of variables

$$x \in [a, b] \quad \text{where } a, b \in \mathbb{R}$$

# Motivation

## Programs with parameters

- parameters
  - inputs from I/O devices
  - formal parameters of program procedures
  - global variables that are only read but never written by the considered program procedure
  - ...
- nonnegative parameters
  - size, length, starting address of a memory region, ...

## Symbolic ranges are desired

- there exist relations among program variables and parameters
- numeric ranges are not precise enough

# Motivation

```
void foo(unsigned int n) {
    unsigned int x;
    x := n;
① while (x ≤ 2n) do {
②     if (?) then   x := x + 2;
       else          x := 2 * x + 1;
③ } od }
```

| Loc | Intervals | Polyhedra |
|---|---|---|
| ① | $x \in [0, +\infty]$ | $x \in [n, 4n+2]$ |
| ② | $x \in [0, +\infty]$ | $x \in [n, 2n]$ |
| ③ | $x \in [1, +\infty]$ | $x \in [n+1, 4n+2]$ |

Using the polyhedra abstract domain [Cousot Halbwachs 78]

- to infer linear relations among variables $x_i$ and parameters $p_j$

  $$\bigwedge \Sigma_i a_i x_i + \Sigma_j b_j p_j \leq c$$

  where $a_i, b_j, c \in \mathbb{R}$
- **drawback**: computational cost is too high

# Motivation

Our goal

- infer the **symoblic** lower and upper bounds for each program variable where each bound is a **linear expression over nonnegative parameters**
  - E.g., $x \in [p_1 + 1.5p_2, 2p_1 + 2p_2 + 3]$ where $p_1$ and $p_2$ are nonnegative parameters
  - expressiveness: between intervals and polyhedra
- be lightweight: $O(nm)$
  - $n$: the number of program variables
  - $m$: the number of nonnegative parameters

# An Abstract Domain to Infer Symbolic Ranges over Nonnegative Parameters

# The Parametric Range (PaRa) abstract domain

A program with

- $n$ program variables: $x_1, \ldots, x_n$
- $m$ nonnegative parameters: $p_1, \ldots, p_m$

Domain representation for PaRa domain

- representation: a linear expression over nonnegative parameters

$$x_j \in [\Sigma_{i=1}^{m} a_i p_i + c, \Sigma_{i=1}^{m} b_i p_i + d]$$

where $a_i, b_i \in \mathbb{R}, c \in \mathbb{R} \cup \{-\infty\}, d \in \mathbb{R} \cup \{+\infty\}$

- semantics:

$$\gamma([\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d]) = \{x_j \in \mathbb{R} \mid \Sigma_i a_i p_i + c \leq x_j \leq \Sigma_i b_i p_i + d\}$$

# The PaRa abstract domain (representation)

## Relaxation of the non-negativity restriction

- for a parameter $p_i$ that may take negative values, if we know its numeric lower bound $c$ or upper bound $d$
- introduce a new auxiliary nonnegative parameter $p_i'$

$$p_i' \stackrel{\text{def}}{=} p_i - c \qquad \text{or} \qquad p_i' \stackrel{\text{def}}{=} d - p_i$$

- replace all the appearances of $p_i$ by $p_i' + c$ (or $d - p_i'$) in the program

**Example (If $n \in [-5, +\infty]$ then introduce $n'$ s.t. $n' = n + 5$)**

```
void foo(int n) {
    int x;
    x := n;
    while (x ≤ 2n) do {
        if (?)  then   x := x + 2;
        else           x := 2 * x + 1;
    } od }
```

$\xRightarrow{n'=n+5}$

```
void foo(unsigned int n') {
    int x;
    x := n' − 5;
    while (x ≤ 2 * (n' − 5)) do {
        if (?)  then   x := x + 2;
        else           x := 2 * x + 1;
    } od }
```

# The PaRa abstract domain (operations)

## Domain operations

1. lattice operations
   - ordering $\sqsubseteq_e$: on linear expressions over nonnegative parameters

$$\Sigma_i a_i p_i + c \sqsubseteq_e \Sigma_i b_i p_i + d \overset{\text{def}}{\Leftrightarrow} \forall p \in [\underline{p}, \overline{p}], \Sigma_i (b_i - a_i) p_i + (d - c) \geq 0$$

     - where $[\underline{p}, \overline{p}]$ denotes numerical ranges for parameters $p$
     - in practice, we check

$$\Sigma_i (b_i - a_i) p'_i + (d - c) \geq 0 \text{ where } p'_i = \begin{cases} \overline{p_i} & \text{if } a_i \geq b_i \\ \underline{p_i} & \text{otherwise} \end{cases}$$

   - inclusion test $\sqsubseteq_p$: between two parametric ranges for the same variable

$$\begin{aligned} & [\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d] \sqsubseteq_p [\Sigma_i a'_i p_i + c', \Sigma_i b'_i p_i + d'] \\ \overset{\text{def}}{=} \ & \Sigma_i a'_i p_i + c' \sqsubseteq_e \Sigma_i a_i p_i + c \ \wedge \ \Sigma_i b_i p_i + d \sqsubseteq_e \Sigma_i b'_i p_i + d' \end{aligned}$$

**Example**

$[p_1 + p_2, 2p_1 + p_2] \sqsubseteq_p [p_2, 2p_1 + 2p_2]$, since $p_2 \sqsubseteq_e p_1 + p_2$ and $2p_1 + p_2 \sqsubseteq_e 2p_1 + 2p_2$

# The PaRa abstract domain (operations)

1. lattice operations (cont)
   - meet $\sqcap_p$: intersection of two parametric ranges for the same variable

$$[\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d] \sqcap_p [\Sigma_i a'_i p_i + c', \Sigma_i b'_i p_i + d']$$

$$\stackrel{\text{def}}{=} \begin{cases} \bot_p & \text{if } \Sigma_i b_i p_i + d \sqsubset_e \Sigma_i a'_i p_i + c' \vee \Sigma_i b'_i p_i + d' \sqsubset_e \Sigma_i a_i p_i + c \\ [lexp, lexp'] & \text{otherwise} \end{cases}$$

$$\text{where} \quad lexp \stackrel{\text{def}}{=} \begin{cases} \Sigma_i a_i p_i + c & \text{if } \Sigma_i a'_i p_i + c' \sqsubseteq_e \Sigma_i a_i p_i + c \\ \Sigma_i a'_i p_i + c' & \text{else if } \Sigma_i a_i p_i + c \sqsubseteq_e \Sigma_i a'_i p_i + c' \\ \Sigma_i a_i p_i + c & \text{else if } \Sigma_i a_i + c \geq \Sigma_i a'_i + c' \\ \Sigma_i a'_i p_i + c' & \text{otherwise} \end{cases}$$

$$lexp' \stackrel{\text{def}}{=} \begin{cases} \Sigma_i b_i p_i + d & \text{if } \Sigma_i b_i p_i + d \sqsubseteq_e \Sigma_i b'_i p_i + d' \\ \Sigma_i b'_i p_i + d' & \text{else if } \Sigma_i b'_i p_i + d' \sqsubseteq_e \Sigma_i b_i p_i + d \\ \Sigma_i b_i p_i + d & \text{else if } \Sigma_i b_i + d \leq \Sigma_i b'_i + d' \\ \Sigma_i b'_i p_i + d' & \text{otherwise} \end{cases}$$

**Example**

Comparable: $[p_1, 2p_1 + p_2] \sqcap_p [p_1 + p_2, 2p_1 + 2p_2] = [p_1 + p_2, 2p_1 + p_2]$

Incomparable: $[2p_1 + p_2, 2p_1 + 3p_2] \sqcap_p [p_1 + 3p_2, p_1 + 5p_2] = [p_1 + 3p_2, 2p_1 + 3p_2]$

# The PaRa abstract domain (operations)

1. lattice operations (cont)
   - join $\sqcup_p$: over-approximation of the union of two parametric ranges:

$$[\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d] \sqcup_p [\Sigma_i a'_i p_i + c', \Sigma_i b'_i p_i + d'] \stackrel{\text{def}}{=} [lexp, lexp'] \text{ where}$$

$$lexp \stackrel{\text{def}}{=} \begin{cases} \Sigma_i a_i p_i + c & \text{if } \Sigma_i a_i p_i + c \sqsubseteq_e \Sigma_i a'_i p_i + c' \\ \Sigma_i a'_i p_i + c' & \text{else if } \Sigma_i a'_i p_i + c' \sqsubseteq_e \Sigma_i a_i p_i + c \\ \Sigma_i \min(a_i, a'_i) p_i + \min(c, c') & \text{otherwise} \end{cases}$$

$$lexp' \stackrel{\text{def}}{=} \begin{cases} \Sigma_i b_i p_i + d & \text{if } \Sigma_i b'_i p_i + d' \sqsubseteq_e \Sigma_i b_i p_i + d \\ \Sigma_i b'_i p_i + d' & \text{else if } \Sigma_i b_i p_i + d \sqsubseteq_e \Sigma_i b'_i p_i + d' \\ \Sigma_i \max(b_i, b'_i) p_i + \max(d, d') & \text{otherwise} \end{cases}$$

**Example**

```
void foo(unsigned int n) {
   unsigned int x;
   x := n;
   while (x ≤ 2n) do {
①     if (?)  then    x := x + 2;  ②
      else          x := 2 * x + 1;  ③
④ } od   }
```

①: $\rho_x = [n, 2n]$

②: $\rho'_x = [n+2,\ 2n+2]$

③: $\rho''_x = [2n+1, 4n+1]$

④: $\rho'_x \sqcup_p \rho''_x = [n+1,\ 4n+2]$

# The PaRa abstract domain (operations)

2. transfer functions
   - test transfer function $[\![x_j \leq \Sigma_i a_i p_i + c]\!]^{\#}(\rho_{x_j})$:

$$[\![x_j \leq \Sigma_i a_i p_i + c]\!]^{\#}(\rho_{x_j}) \stackrel{\text{def}}{=} \rho_{x_j} \sqcap_p [-\infty, \Sigma_i a_i p_i + c]$$

$$[\![x_j \geq \Sigma_i b_i p_i + d]\!]^{\#}(\rho_{x_j}) \stackrel{\text{def}}{=} \rho_{x_j} \sqcap_p [\Sigma_i b_i p_i + d, +\infty]$$

   - assignment transfer function $[\![x_j \leq \Sigma_i a_i p_i + c]\!]^{\#}(\rho_{x_j})$:

$$[\![x_j := [\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d]]\!]^{\#}(\rho_{x_j}) \stackrel{\text{def}}{=} [\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d]$$

# The PaRa abstract domain (operations)

3. Widening with Thresholds $\nabla_p^T$: a widening parameterized by a finite set of threshold values $T$ including $-\infty$ and $+\infty$

$$[\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d] \nabla_p^T [\Sigma_i a'_i p_i + c', \Sigma_i b'_i p_i + d']$$

$$\stackrel{\text{def}}{=} [\Sigma_i a''_i p_i + c'', \Sigma_i b''_i p_i + d'']$$

where

$$\begin{cases} a''_i \stackrel{\text{def}}{=} a_i \leq a'_i \ ? \ a_i : \max\{\ell \in T \mid \ell \leq a'_i\} \\ c'' \stackrel{\text{def}}{=} c \leq c' \ ? \ c : \max\{\ell \in T \mid \ell \leq c'\} \\ b''_i \stackrel{\text{def}}{=} b_i \geq b'_i \ ? \ b_i : \min\{h \in T \mid h \geq b'_i\} \\ d'' \stackrel{\text{def}}{=} d \geq d' \ ? \ d : \min\{h \in T \mid h \geq d'\} \end{cases}$$

# The PaRa abstract domain (operations)

## Example (Widening with Thresholds)

```
void foowiden(unsigned int n){
   real x;
   x := 0.75 * n + 1;
   while (true) do {
   ①  if (?)
        then  x := n + 1;
        else  x := 0.25 * x + 0.5 * n + 1;
   } od
}
```

$$T = \{0, 0.5, 1, 1.5, -\infty, +\infty\}$$

$$\rho_x = [0.75n + 1, \quad 0.75n + 1]$$

$$\rho'_x = [0.6875n + 1, n + 1.25]$$

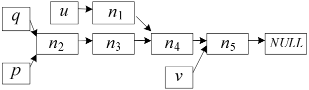
$$\rho_x \nabla_p^T \rho'_x = [0.5n + 1, \quad n + 1.5]$$

# Application to Infer Symbolic Ranges of List Segment Sizes

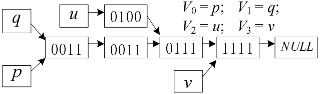
# Inferring symbolic ranges of list segment sizes

Analyzing programs manipulating singly linked lists (SLL) [Chen et al. 2013]

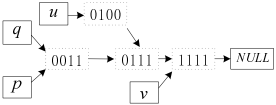
- divide a SLL into a set of non-overlapping list segments according to reachability of pointer variables to list nodes
- introduce a nonnegative integer variable $t^{bitvec}$ to track the size for each list segment (denoted by *bitvec*)
- nonnegative parameters: the initial lengths of input lists




(a)




(b)




(c)

| vuqp | num |
|---|---|
| 0011 | 2 |
| 0100 | 1 |
| 0111 | 1 |
| 1111 | 1 |

(d)

# Example of deriving numeric programs from SLL programs

```
void copy_and_delete(List* x, uint n){
    List* y, p, q;
    assume \length(x)==n;
    y := x;
    q := p := null;
    while (y != null) do {
        y := y → next;
        q := malloc();
        q → next := pList;
        p := q;
    } od
    y := x;
    while (y != null) do {
        y := y → next;
        q := q → next;
        free(x);
        free(p);
        x := y;
        p := q;   } od }
```

```
void copy_and_delete_num(uint n){
  uint t^x, t^y, t^xy, t^p, t^q, t^pq;
  t^x := n;
  t^xy := t^x; t^x := 0;
  t^p := 0; t^q := 0; t^pq := 0;
  while (t^xy ≥ 1) do {
     t^x := t^x+1; t^xy := t^xy -1;
     t^p := t^pq; t^pq := 0; t^q := 1;
     t^pq := t^p; t^p := 0;
     t^pq := t^q+t^pq; t^q := 0;
  } od
  t^xy := t^x; t^x := 0;
  while (t^xy ≥ 1) do {
     t^x := t^x+1; t^xy := t^xy -1;
     t^p := t^p+1; t^pq := t^pq -1;
     t^y := t^xy; t^xy := 0; t^x := 0;
     t^q := t^pq; t^pq := 0; t^p := 0;
     t^xy := t^y; t^y := 0;
     t^pq := t^q; t^q := 0;   } od }
```

# Inferring symbolic ranges of list segment sizes

## Combine PaRa and affine equalities

- there often exist affine equality relations between program variables and parameters in the derived numeric programs
  - parametric ranges: to track symbolic ranges of each program variable
  - affine equalities: to track the affine equality relations among program variables and parameters

## Representation

$$A\ [x\ \ p]^T = b' \quad \text{affine equalities}$$

$$x_j \in [\Sigma_i a_i p_i + c, \Sigma_i b_i p_i + d] \quad \text{parametric ranges}$$

$$p_i \in [c', d'] \quad \text{numeric ranges}$$

# Example of inferring symbolic ranges of list segment sizes

```
void copy_and_delete(List* x, uint n){
    List* y, p, q;
    assume \length(x)==n;
    y := x;
    q := p := null;
    while (y != null) do {
      y := y → next;
      q := malloc();
      q → next := pList;
      p := q;
    } od
     y := x;
     while (y != null) do {


        y := y→ next;
        q := q→ next;
        free(x);
        free(p);
        x := y;
        p := q;   } od }
```

```
void copy_and_delete_num(uint n){
  uint t^x, t^y, t^xy, t^p, t^q, t^pq;
   t^x := n;
   t^xy := t^x; t^x := 0;
   t^p := 0; t^q := 0; t^pq := 0;
   while (t^xy ≥ 1) do {
      t^x := t^x+1; t^xy := t^xy -1;
      t^p := t^pq; t^pq := 0; t^q := 1;
      t^pq := t^p; t^p := 0;
      t^pq := t^q+t^pq; t^q := 0;
   } od
  t^xy := t^x; t^x := 0;
  while (t^xy ≥ 1) do {
   /* t^xy − t^pq == 0,
       t^xy ∈ [1, n], t^pq ∈ [1, n], n ∈ [1, +∞] */
     t^x := t^x+1; t^xy := t^xy -1;
     t^p := t^p+1; t^pq := t^pq -1;
     t^y := t^xy; t^xy := 0; t^x := 0;
     t^q := t^pq; t^pq := 0; t^p := 0;
     t^xy := t^y; t^y := 0;
     t^pq := t^q; t^q := 0;   } od }
```

# Implementation and Experiments

# Prototype

Prototype implementation PARA

- using GMP (the GNU Multiple Precision arithmetic library)
  - to guarantee the soundness of the implementation

Interface:

- plugged into the APRON library [Jeannet Miné]
- programs analyzed with INTERPROC [Jeannet et al.]

Comparison with

- Box: intervals
- NewPolka: polyhedra

# Example analyses

```
void foo(unsigned int n) {
      unsigned int x;
      x := n;
   ① while (x ≤ 2n) do {
   ②      if (?)  then    x := x + 2;
          else        x := 2 * x + 1;
   ③ } od
}
```

| Loc | Intervals | Polyhedra | Parametric Ranges |
|---|---|---|---|
| ① | $x \in [0, +\infty]$ | $x \in [n, 4n+2]$ | $x \in [n, 4n+2]$ |
| ② | $x \in [0, +\infty]$ | $x \in [n, 2n]$ | $x \in [n, 2n]$ |
| ③ | $x \in [1, +\infty]$ | $x \in [n+1, 4n+2]$ | $x \in [n+1, 4n+2]$ |

# Experimental results on numeric programs

| Program | | | Analysis Results | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | #Vars | #Pars | Box | Inv. | PaRa | Inv. | PaRa + Affine | Inv. | NewPolka |
| foo | 1 | 1 | 6ms | $<$ | 7ms | $=$ | 8ms | $=$ | 12ms |
| foowiden | 1 | 1 | 6ms | $<$ | 7ms | $=$ | 8ms | $>$ | 12ms |
| ex_ipps95 | 1 | 1 | 4ms | $<$ | 6ms | $=$ | 7ms | $=$ | 11ms |
| ex_ippms95 | 1 | 1 | 4ms | $<$ | 6ms | $=$ | 7ms | $=$ | 11ms |
| ex_sas07 | 2 | 2 | 5ms | $<$ | 6ms | $<$ | 6ms | $=$ | 12ms |
| ex_toplas05 | 2 | 1 | 6ms | $<$ | 8ms | $<$ | 10ms | $=$ | 16ms |
| ex_cav09_1 | 3 | 2 | 7ms | $<$ | 10ms | $<$ | 15ms | $=$ | 21ms |
| ex_cav09_2 | 2 | 2 | 7ms | $<$ | 7ms | $<$ | 10ms | $=$ | 17ms |
| ex_cav09_3 | 4 | 1 | 8ms | $<$ | 11ms | $<$ | 17ms | $=$ | 21ms |
| ex_cav12_1 | 2 | 1 | 3ms | $<$ | 4ms | $=$ | 4ms | $<$ | 10ms |
| ex_cav12_2 | 2 | 0 | 1ms | $=$ | 2ms | $<$ | 4ms | $=$ | 6ms |
| all_above | 20 | 12 | 23ms | $<$ | 53ms | $<$ | 92ms | $\neq$ | 335ms |

Most "Para+AffineEqs" results are better than Box and as precise as Newpolka

# Experimental results on SLL programs

| Program | | | Analysis Results | | | | |
|---|---|---|---|---|---|---|---|
| Name | #Vars | #Pars | Box | Inv. | PaRa+ AffineEqs | Inv. | NewPolka |
| list_create | 4 | 1 | 8ms | $<$ | 11ms | $=$ | 18ms |
| list_traverse | 3 | 1 | 6ms | $<$ | 8ms | $=$ | 16ms |
| list_reverse | 5 | 1 | 9ms | $<$ | 15ms | $=$ | 25ms |
| list_length_equal | 4 | 1 | 7ms | $<$ | 10ms | $=$ | 17ms |
| list_merge | 5 | 2 | 9ms | $<$ | 18ms | $=$ | 27ms |
| list_copy_and_delete | 6 | 1 | 7ms | $<$ | 24ms | $=$ | 32ms |
| list_dispatch | 7 | 1 | 11ms | $<$ | 29ms | $=$ | 40ms |
| list_all_above | 34 | 8 | 36ms | $<$ | 181ms | $=$ | 826ms |

- On these programs, "PaRa+AffineEqs" gives as precise invariants as those by NewPolka
- These invariants are precise enough to prove the memory safety of the original list-manipulating programs

# Conclusion

Summary:

- **goal**: infer symbolic ranges of program variables efficiently
- **idea**: using linear expressions over nonnegative parameters as symbolic ranges
  - a new abstract domain: parametric ranges (PaRa)
    - time and space complexity of this domain: $O(nm)$
  - applications to infer symbolic ranges of list segment sizes
    - combining parametric ranges (PaRa) and affine equalities

Future Work

- consider the usage of parametric ranges in more applications
- use nonlinear expressions over nonnegative parameters as symbolic ranges