Range Analysis

Ananya Bahadur

November 16, 2017

Contents

1 Overview

Range Analysis is a forward DFA. The `range` of a variable v at a program point p is the range $[L, R]$ of numeric values that it can assume.

1.1 Scope

We restrict ourselves to integers to avoid issues with the boundaries of ranges, specifically intervals with open end(s) and division. Integer division results are very well defined and are neither implementation nor hardware dependent.

Am I handling integer overflow and wrap around?

1.2 Properties

Inseparability The data flow values of different variables are dependent on each other.

2 Data Flow Equations

2.1 Data Flow values

In_n, Out_n values are mappings $\mathbb{V}\text{ar} \to \hat{L}$. These data flow values are also called *context*, and denoted by X in this document.

$$In_n = \begin{cases} \{\langle\, y \mapsto \texttt{undef} \,\rangle \mid y \in \mathbb{V}\text{ar}\} & \text{if } n \text{ is start} \\ \bigsqcap_{p \in Pred(n)} In_p & \text{otherwise} \end{cases} \tag{1}$$

$$Out_n = f_n(In_n) \tag{2}$$

2.2 Flow Function

We define a Flow Function $f(X) : L \to L$ and a function $\text{eval}(e, X)$ that computes the `range` of expression e in the context X.

$$f_n(X) = \begin{cases} X\big[u \mapsto (-\infty, \infty)\big] & u \in \mathbb{V}\text{ar}, u \text{ is read from file} \\ X\big[u \mapsto [k, k]\big] & u \in \mathbb{V}\text{ar}, u = k, k \text{ is constant} \\ X\big[u \mapsto X[v]\big] & u, v \in \mathbb{V}\text{ar}, u = v \\ X\big[u \mapsto \text{eval}(e, X)\big] & u \in \mathbb{V}\text{ar}, u = e, e \text{ is an expression} \end{cases} \tag{3}$$

$$\text{eval}(e, X) = \begin{cases} X[a] \oplus X[b] & a, b \in \text{Opd}(e) \cap \mathbb{V}\text{ar}, \text{Op} = + \\ X[a] \ominus X[b] & a, b \in \text{Opd}(e) \cap \mathbb{V}\text{ar}, \text{Op} = - \\ X[a] \otimes X[b] & a, b \in \text{Opd}(e) \cap \mathbb{V}\text{ar}, \text{Op} = \times \\ X[a] \oslash X[b] & a, b \in \text{Opd}(e) \cap \mathbb{V}\text{ar}, \text{Op} = \div \end{cases} \tag{4}$$

2.3 Interval Operations

Interval arithmetic is surprisingly tricky, wrt. division and multiplication (9 cases!). 2 discusses ways to implement a fast interval multiplier in modern pipelined superscalar FPUs. Here is a handy web interval arithmetic calculator to verify (my) implementations.

$$[a,b] \oplus [x,y] = [a+x, b+y] \tag{5}$$
$$[a,b] \ominus [x,y] = [a-y, b-x] \tag{6}$$
$$[a,b] \otimes [x,y] = [\min(ax,ay,bx,by),\ \max(ax,ay,bx,by)] \tag{7}$$
$$[a,b] \oslash [x,y] = \begin{cases} [a,b] \otimes [1/y, 1/x] & 0 \notin [x,y] \\ [-\infty,\infty] & 0 \in [a,b] \wedge 0 \in [x,y] \\ [b/x,\infty] & b < 0 \wedge y = 0 \\ [-\infty,b/y] \cup [b/x,\infty] & b < 0 \wedge x < 0 < y \\ [-\infty,b/y] & b < 0 \wedge x = 0 \\ [-\infty,a/x] & a > 0 \wedge y = 0 \\ [-\infty,a/x] \cup [a/y,\infty] & a > 0 \wedge x < 0 < y \\ [a/y,\infty] & a > 0 \wedge x = 0 \\ \emptyset & 0 \notin [a,b] \wedge c = d = 0 \end{cases} \tag{8}$$

All interval arithmetic operations and algorithms are compiled in Hickey et al.. Additional operators can be defined such as exponentiation and logarithm.

2.4 Component Lattice

2.4.1 Range ($\mathcal{R}$)

A range is a tuple,

$$(l,r) \equiv \{a \mid a \in \mathbb{Z},\ l \le a \le r\}, \quad l \le r\ \wedge\ l,r \in \mathbb{Z}$$

We will be *merging* ranges in 1, so we need to define a new *closed* $\hat{\cup}$ operator for our analysis:

$$\begin{aligned} a_1 \,\hat{\cup}\, a_2 &= (l_1,r_1) \,\hat{\cup}\, (l_2,r_2) \\ &= \big(\min(l_1,l_2),\ \max(r_1,r_2)\big) \end{aligned} \tag{9}$$

> This definition results in imprecision upon application of $\sqcap$ in 1, but makes the result computable. If we were to use the natural and *precise* $\cup$ (which gives a union of disjoint sets), the result of 1 can become arbitrarily large.

Some boolean operators are also defined,

$$a_1 \subseteq a_2 = (l_1 \ge l_2\ \wedge\ r_1 \le r_2), \quad a_1,a_2 \in \mathcal{R} \tag{10}$$

2.4.2 Component Set

$\mathcal{A}$ is the set of all ranges in $\mathbb{Z}$, **including** the range $\emptyset$ which acts like an `undef` in our analysis.

2.4.3 The Lattice

$$\hat{L} = (\mathcal{A}, \supseteq) \tag{11}$$

Thus, $\top$ is $\emptyset$ and $\bot$ is $(-\infty, \infty)$.

2.4.4 Meet Operator

$$a \hat{\sqcap} b = \hat{\cup} \quad a, b \in \mathcal{R} \tag{12}$$

> *I have to decide if I'm gonna stay in Z or move to R. This is especially important with my lattice definition, as A may rather be the set of all ranges in* `[-MAX_INT, +MAX_INT]` *in case of Z.*

3 Soot

The best resources to `soot` are packaged with it in the `/tutorial` directory. Run `make` in it and enjoy the material.

3.1 `FlowSet` implementation

Most (simple) analyses compute 1 `bit` information about some code entity (variable or expression, etc). Constant Propagation computes n `bit` information per variable and also has an unbounded flow function like Range Analysis.

For such analyses, a `HashMap` implementation suits best to represent `FlowSets` because we frequently lookup the computed information of other entities – mere `use` or `def` doesn't serve us much. Since Java is weakly typed, I found it hard to implement a generic `HashMapFlowSet` and just went ahead with a much simpler `RangeFlowSet`.

- `clone()`
- `clear()`
- `isEmpty()`
- `copy (FlowSet dest)`
- `union (FlowSet other, FlowSet dest)`
- `intersection (FlowSet other, FlowSet dest)`
- `difference (FlowSet other, FlowSet dest)`

3.2 `Range`

We define a `Range` object. An instance is created at the following locations in code, as defined in 3:

- Assignments (constant, expression, other local)
- Read from `stdin`

References

T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, Sept. 2001. ISSN 0004-5411. doi: 10.1145/502102.502106. URL `http://fab.cba.mit.edu/classes/S62.12/docs/Hickey_interval.pdf`.

E. D. Popova. On the efficiency of interval multiplication algorithms. In *3rd International Conference 'Real Numbers and Computers'*, pages 117–132, Paris, France, Apr. 1998. URL `http://www.math.bas.bg/~epopova/papers/98EPopova-RNC.pdf`.