Software Verification

Bounded Box

Mattia Bottaro - Mauro Carlin
May, 2018

Università degli Studi di Padova

Outline

Project - Domain

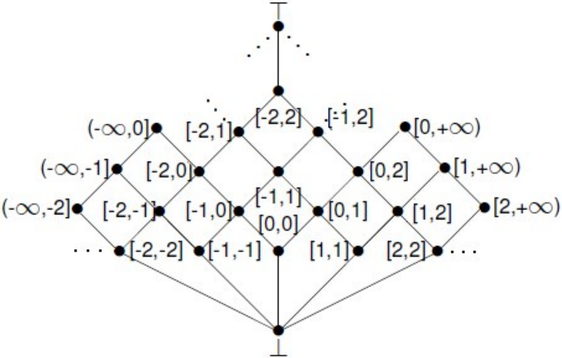
We have chosen the **Bounded Box Domain**, which is a parametric restriction of the interval abstract domain *Int*:

Given $m, n \in \mathbb{Z} \cup \{-\infty, +\infty\}$, then

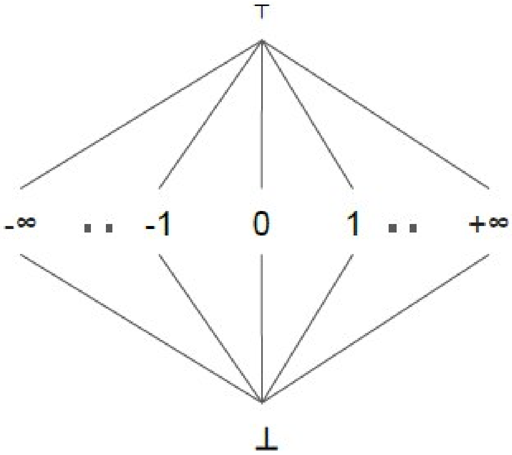
$$\mathrm{Int}_{m,n} := \{\varnothing, \mathbb{Z}\} \cup \{[k,k] \mid k \in \mathbb{Z}\} \cup \{[a,b] \mid a < b,\ [a,b] \subseteq [m,n]\} \cup \{(-\infty, k] \mid k \in [m,n]\} \cup \{[k, +\infty) \mid k \in [m,n]\}$$

Project - Domain

Bounded Box with **m = -2**, **n = 2**



Bounded Box with **m** > **n**



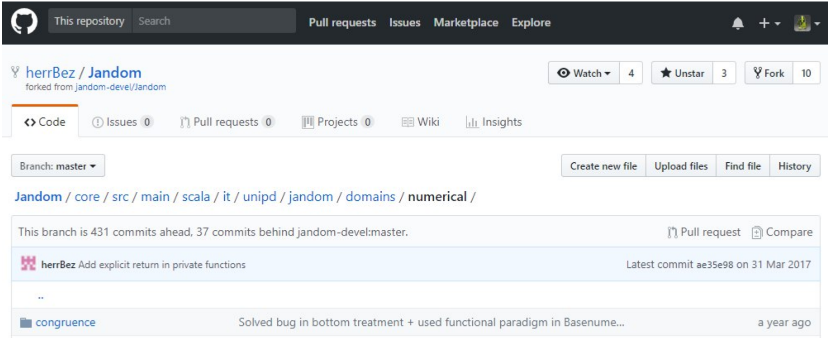
Project - Jandom

What is **Jandom**?

1. Static Analyzer for **Numerical** Domains
2. It was created at the University of Chieti-Pescara
3. It's a buildup of **RANDOM**, which analyzes **R** code
4. It analyzes JVM bytecode using **Soot**
5. It is written in **Scala**

Project - Jandom

We've extended Jandom from this repository created by University of Padua' students



Project - Architecture

In order to create a new domain **X**, you have to implement the following classes:

1. **XElement**: *sealed trait* used to represent an element of the domain.
2. **XDomainCore**: *class* which implements abstract operators.
3. **XDomain**: *class* which represents the domain.

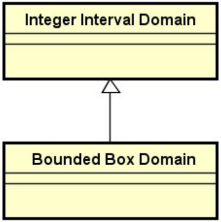
Outline

Our Contribution

We have:

1. Implemented the Integer Interval Domain

2. Implemented Bounded Box Domain specializing the previous domain, because abstract operators of both domains are very similar



Our Contribution

Abstract **sum** operator algorithm in Bounded Box Domain.

1. Execute sum operator of Interval Domain.
   $[a, b] +^{\#}_{b} [c, d] = [a + c, b + d] = [e, f]$
2. $[e, f]$ must be represented as an element of Bounded Box Domain

$$[e, f] = \begin{cases} \top^{\#} & e < m \wedge f > n \\ [n, +\infty) & e \geq n \wedge e \neq f \\ [e, +\infty) & e < n \wedge f > n \\ (-\infty, m] & f \leq m \wedge e \neq f \\ (-\infty, f] & f > m \wedge e < m \\ [e, f] & otherwise \end{cases}$$

Our Contribution

UNIVERSITÀ DEGLI STUDI DI PADOVA

Abstract **reminder** operator algorithm in Box Domain

Definition

$$[a,b]\%^{\#}_{b}[c,d] = \begin{cases} \top^{\#} & [c,d] = \top^{\#} \\ \bot^{\#} & [a,b] = \bot^{\#} \vee [c,d] = \bot^{\#} \vee [c,d] = [0,0] \\ [0,0] & [a,b] = [0,0] \\ [0, d-1] & c \geq 0 \\ [c+1, 0] & d \leq 0 \\ [c+1, d-1] & \textit{otherwise} \end{cases}$$

Our Contribution

We have defined a new type, called *InfInt*, to:

1. Model infinity values with Integer type
2. Overload operations between integer number
3. Simplify further contribution

Example

$(+\infty) + n = +\infty$
$(+\infty) \times (-\infty) = -\infty$
$(+\infty) \div (+\infty) = 0$

Outline

First Example

Let's suppose **m = -10**, **n = 10**

```
public static void constantTest() {
    int x = 0;
    int y = -20;
    int w = x + y;
    if (w - x != -5) {
        y = y * w;
    } else {
        w = x % y;
    }
}
```

(1)

```
public static void constantTest()
  {
     byte b0, b2;
     int $i1, i3, i4, i5;
/*[ [ b0 = ⊤ , $i1 = ⊤ , b2 = ⊤ , i3 = ⊤ , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     b0 = 0;
/*[ [ b0 = [0,0] , $i1 = ⊤ , b2 = ⊤ , i3 = ⊤ , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     b2 = -20;
/*[ [ b0 = [0,0] , $i1 = ⊤ , b2 = [-20,-20] , i3 = ⊤ , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     i3 = b0 + b2;
/*[ [ b0 = [0,0] , $i1 = ⊤ , b2 = [-20,-20] , i3 = [-20,-20] , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     $i1 = i3 - b0;
/*[ [ b0 = [0,0] , $i1 = [-20,-20] , b2 = [-20,-20] , i3 = [-20,-20] , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     if $i1 == -5 goto label1;
/*[ [ b0 = [0,0] , $i1 = [-20,-20] , b2 = [-20,-20] , i3 = [-20,-20] , i4 = ⊤ , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     i4 = b2 * i3;
/*[ [ b0 = [0,0] , $i1 = [-20,-20] , b2 = [-20,-20] , i3 = [-20,-20] , i4 = [400,400] , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     goto label2;
   label1:
/*[ [ empty ]types: byte,int,byte,int,int,int ]*/      Unreachable point
     i5 = b0 % b2;
   label2:
/*[ [ b0 = [0,0] , $i1 = [-20,-20] , b2 = [-20,-20] , i3 = [-20,-20] , i4 = [400,400] , i5 = ⊤ ]types: byte,int,byte,int,int,int ]*/
     return;
  }
/* Output: [ b0 = [0,0] , $i1 = [-20,-20] , b2 = [-20,-20] , i3 = [-20,-20] , i4 = [400,400] , i5 = ⊤ ]types: byte,int,byte,int,int,int */
```

Second Example

Let's suppose **m = -10**, **n = 10**

```java
public static void test1() {
    int x = -5;
    int y = 15;
    while (x <= 5) {
        y = x * y;
        x++;
    }
    if (x >= 7) {
        x = x - 10;
    } else {
        x = x + 10;
    }
}
```

(2)

```
    public static void test1()
    {
        int i0, i1, i2, i3;
/*[ [ i0 = ⊤ , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        i0 = -5;
/*[ [ i0 = [-5,-5] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        i1 = 15;
     label1:
/*[ [ i0 = [-5,6] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/   Loop Invariant
        if i0 > 5 goto label2;
/*[ [ i0 = [-5,5] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        i1 = i0 * i1;
/*[ [ i0 = [-5,5] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        i0 = i0 + 1;
/*[ [ i0 = [-4,6] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        goto label1;
     label2:
/*[ [ i0 = [5,6] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        if i0 < 7 goto label3;
/*[ [ empty ]types: int,int,int,int ]*/
        i2 = i0 - 10;                       Unreachable point
/*[ [ empty ]types: int,int,int,int ]*/
        goto label4;
     label3:
/*[ [ i0 = [5,6] , i1 = ⊤ , i2 = ⊤ , i3 = ⊤ ]types: int,int,int,int ]*/
        i3 = i0 + 10;
     label4:
/*[ [ i0 = [5,6] , i1 = ⊤ , i2 = ⊤ , i3 = [10,+∞] ]types: int,int,int,int ]*/
        return;
    }
/* Output: [ i0 = [5,6] , i1 = ⊤ , i2 = ⊤ , i3 = [10,+∞] ]types: int,int,int,int */
```

Nota Bene

Jandom implements **subtraction** with the following steps:

1. Perform **Inverse** operator on the second operand
2. Perfom **Add** operator

So, when **m** $\neq$ **-n** the following will happen:

Example with $m = 0, n = 10$

$[4,5] -^{\#} [2,3]$ is computed as $[4,5] +^{\#} (-\infty, 0] = (-\infty, 5]$
instead of $[1,3]$, that would be the best possibile approximation