Extending Sample with non-relational numerical domains and heap analyses

Pietro Ferrara
ETH Zurich
pietro.ferrara@inf.ethz.ch

February 25, 2010

Abstract

Sample can be plugged with different (relational and non-relational) numerical domains and heap analyses. This document presents and explains the interface that has to be implemented in order to develop a new non-relational numerical domain and heap analysis in Java and plug them in Sample.

1 Lattice

```
package ch.ethz.inf.pm.AbstractInterpreter.AbstractDomain;

public interface Lattice {
    public boolean lessEqual(Lattice);
    public Lattice widening(Lattice, Lattice);
    public Lattice glb(Lattice, Lattice);
    public Lattice lub(Lattice, Lattice);
    public Lattice bottom();
    public Lattice top();
    public Lattice factory();
}
```

The methods of interface Lattice have the common meaning of operators on lattices. Formally,

- **this**.lessEqual(a) returns **true** $\Leftrightarrow$ **this** $\leq$ a
- widening(a, b) returns a ∇ b
- glb(a, b) returns a $\sqcap$ b
- lub(a, b) returns a $\sqcup$ b

- top() returns $\top$
- bottom() returns $\bot$

In addition, factory () returns a new instance of the current domain.

2 NonRelationalNumericalDomain

```
package ch.ethz.inf.pm.AbstractInterpreter.AbstractDomain.NumericalDomain;

public interface NonRelationalNumericalDomain extends Lattice {
    public NonRelationalNumericalDomain valueLEQ(NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain valueGEQ(NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain
            divide(NonRelationalNumericalDomain, NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain
            multiply(NonRelationalNumericalDomain, NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain
            subtract(NonRelationalNumericalDomain, NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain
            sum(NonRelationalNumericalDomain, NonRelationalNumericalDomain);
    public NonRelationalNumericalDomain evalConstant(int);
}
```

Interface NonRelationalNumericalDomain requires to implement all the common arithmetic operators on non-relational numerical domains. In particular,

- valueLEQ(a) returns the abstract value approximating the numerical values that are less or equal than a, for instance, valueLEQ($[0..2]$)= $[-\infty..2]$
- valueGEQ(a) returns the abstract value approximating the numerical values that are greater or equal than a, for instance, valueGEQ($[0..2]$)= $[0.. + \infty]$
- divide(a, b) returns the abstract result of the division a/b, for instance, divide($[2..4], [2..2]$)= $[1..2]$
- multiply(a, b) returns the abstract result of the multiplication a$*$b, for instance, multiply($[2..4], [2..2]$)= $[4..8]$
- subtract(a, b) returns the abstract result of the subtraction a$-$b, for instance, subtract($[2..4], [2..2]$)= $[0..2]$
- sum(a, b) returns the abstract result of the addition a+b, for instance, sum($[2..4], [2..2]$)= $[4..6]$

- evalConstant(i) returns the abstract representation of the numerical value i, for instance, evalConstant(1)= [1..1]

3 HeapIdentifier

```
package ch.ethz.inf.pm.AbstractInterpreter.AbstractDomain

public abstract class HeapIdentifier extends Identifier {
    public HeapIdentifier(Type);
    public abstract HeapIdentifier factory();
    public abstract HeapIdentifier extractField(HeapIdentifier, String, Type);
    public abstract HeapIdentifier createAddress(Type, ProgramPoint);
    public HeapIdentifier accessStaticObject(Type t);
}

public abstract class Identifier extends Expression {
    public Identifier(Type);
    public abstract boolean representSingleVariable();
    public abstract String getName();
}
```

Abstract class HeapIdentifier requires to implement all the operators required to run the heap analysis. In particular,

- factory() returns a new instance of the HeapIdentifier class
- extractField(HeapIdentifier h, String f, Type t) returns the heap identifier representing the access of field f of type t on the object identifier by h
- createAddress(Type t, ProgramPoint p) returns the heap identifier representing a new instance of type t created at program point p
- acessStaticObject(t) returns the identifier of accessing the static object of type t
- representSingleVariable() returns **true** if and only if the current abstract heap identifier represents exactly one concrete reference
- getName() returns the name of the current heap identifier

4 How to run the analysis

```
NonRelationalNumericalDomainAndHeapAnalysis.analyze(
                                                    "<ClassName>",
                                                    "<MethodName>",
```

```
                                        "<FilePath>",
                                        <NumericalDomain>,
                                        <HeapAnalysis>
    );
```

In order to run the analysis in Java, the user has to call method analyze on the static object NonRelationalNumericalDomainAndHeapAnalysis passing

1. "<ClassName>": the string containing the name of the class to be analyzed
2. "<MethodName>": the string containing the name of the method to be analyzed
3. "<FilePath>": the complete path of the file to be analyzed
4. <NumericalDomain>: an instance of the non relational domain that has to be used during the analysis. Such object has to be instance of interface NonRelationalNumericalDomain
5. <HeapAnalysis>: an instance of the heap analysis that has to be used during the analysis. Such object has to be instance of abstract class HeapIdentifier