

Guida all'analizzatore

Pietro Ferrara - pietro.ferrara@inf.ethz.ch

July 31, 2009

1 Dominio numerico

Per implementare il dominio numerico è necessario estendere la classe astratta `AbstractNumericalValue` del package principale. In particolare, dovrete implementare i seguenti metodi:

```
public abstract NumericalValue lub(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;
public abstract NumericalValue widening(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;
public abstract NumericalValue evalConstant(long value, boolean firstType)
                                   throws SemanticException;
public abstract NumericalValue add(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;
public abstract NumericalValue divide(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;
public abstract NumericalValue multiply(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;
public abstract NumericalValue subtract(NumericalValue v1, NumericalValue v2)
                                   throws SemanticException;

public abstract BooleanDomain testTrue
    (NumericalValue v1, NumericalValue v2, ComparisonOperator op) throws SemanticException;
public abstract BooleanDomain testFalse
    (NumericalValue v1, NumericalValue v2, ComparisonOperator op) throws SemanticException;
public abstract boolean lessEqual(NumericalValue v) throws SemanticException;
public abstract NumericalValue top(boolean firstType);

public abstract String toString();
public abstract boolean equals(Object o);
public abstract int hashCode();

public abstract NumericalValue clone();
```

Il significato delle operazioni di tali metodi è il seguente:

- $\text{lub}(v1, v2) = v1 \sqcup v2$;
- $\text{widening}(v1, v2) = v1 \nabla v2$ (per i domini di altezza finita il widening è uguale al least upper bound);
- $\text{evalConstant}(\text{value})$ ritorna la rappresentazione del valore numerico `value` nel dominio astratto (ad esempio, la costante 1 è rappresentata nel dominio degli intervalli da `[1..1]`);
- $\text{add}(v1, v2) = v1 + v2$;
- $\text{divide}(v1, v2) = v1/v2$;
- $\text{multiply}(v1, v2) = v1 * v2$;

- `subtract(v1, v2) = v1 - v2`;
- `testTrue(v1, v2, op)` ritorna `BooleanDomain.True` se l'espressione $v1 < op > v2$ è valutata a vero, `BooleanDomain.False` se l'espressione $v1 < op > v2$ è valutata a false, `BooleanDomain.Top` se non si riesce a valutare precisamente l'espressione $v1 < op > v2$. L'operatore è di tipo `ComparisonOperator` (tipo `enum` nel package `AbstractDomain`);
- `testFalse(v1, v2, op)`;
- `lessEqual(v) = true` se e solo se $this \leq v$;
- `toString()` ritorna una rappresentazione SINTETICA (su una sola riga di pochi caratteri) del valore astratto;
- `equals(o) = true` se e solo se $this == o$;
- `hashCode()` deve ritornare lo stesso valore intero se due valori astratti potrebbero essere uguali (ovvero se $v1 = v2$ allora $v1.hashCode() = v2.hashCode()$), ad esempio potrebbe ritornare nel dominio degli intervalli il valore minimo dell'intervallo);
- `top()` deve ritornare il valore top del dominio (ad esempio $[-\infty.. +\infty]$).

`ComparisonOperator` contiene tutti i casi possibili di operatore condizionale tra valori. Voi dovete implementare i casi per `equal`, `notequal`, `greater`, `greaterequal`, `less` e `lessequal`.

2 Come eseguire l'analisi

Una volta implementato il dominio numerico potete testare la vostra analisi. Per fare ciò dovete:

1. creare un'istanza della classe `MonoThreadAnalysis`. Per fare ciò, dovete usare il costruttore `public MonoThreadAnalysis(NumericalValue d)`. Basta passare un qualsiasi valore numerico astratto;
2. invocare il metodo `analyze(String directory, String class_name)` della classe `MonoThreadAnalysis`. Il primo parametro `directory` deve contenere il path completo in cui avete messo i files `.class` da analizzare, mentre il secondo parametro `class_name` deve contenere il nome della classe di cui si vuole analizzare il metodo `main`.

Una volta terminata l'analisi apparirà una finestra (come quelle mostrate a lezione) con la rappresentazione del grafo di controllo astratto. In caso di errore invece verrà stampato sullo standard output l'eccezione lanciata.

3 Non supportato

L'analizzatore supporta praticamente tutto il linguaggio bytecode. Tuttavia ci sono una serie di metodi `native` (ovvero implementati in linguaggi diversi di Java - in genere in C - e interfacciati con il sistema) che ovviamente non possono essere analizzati da Java. In generale vi consiglio di evitare di invocare metodi delle librerie (ad esempio `System.out.println`) all'interno dei programmi che analizzate.

4 Compilazione ed esecuzione

Il file `Checkmate.jar` contiene tutti i file dell'analizzatore. Vi consiglio di implementare tutte le vostre classi nel package principale e di mettere il file `jar` nella directory da cui compilate la vostra implementazione. Tra tali classi una deve essere la classe che lancia la vostra analisi. Ad esempio, potete implementarla nella maniera seguente:

```
public class Test {  
  
    public static void main(String[] args) throws Exception {  
        MonoThreadAnalysis Analysis=new MonoThreadAnalysis(new IntervalDomain());  
        Analysis.analyze("C:\\Classi", "Prova321321");  
    }  
  
}
```

A questo punto, per compilare dovete eseguire la linea di comando seguente:

```
javac -classpath .Checkmate.jar;. *.java
```

Una volta compilato, potete eseguire la vostra analisi attraverso la seguente linea di comando (supponendo che la vostra classe che contiene il metodo `main` da eseguire sia `Test`):

```
javac -classpath .Checkmate.jar;. Test
```