Practical Mathematical Proof of Correctness

The Journey to the Workbench

Frank Rimlinger
SNAC (System & Network Analysis Center)
Information Assurance Directorate, National Security Agency

A little history

- Currently an open-source RCP application
- Part of JPF (JavaPathfinder) project of NASA/ AMES
- Original technology transferred to NASA Software Release Authority in 2007.
- Port to Workbench, more usable, wider audience, scale up to approx 10,000 lines.

Mango

- Mango plugin for the Eclipse workbench
  - case enumeration and specification, proof artifact generation
  - Javadoc level of detail
- Organize by parallel cases
  - Fun with graphs
- Understanding the results
  - Exposing Mango model with Workbench debugger
- Goal: tool to provide high-assurance techniques to Eclipse java programmers

Examples

- Hello world: assumptions and output state
- ItsAWrap: conjectures and proofs
- CarRecall: case splits and variables
- Tic tac toe: deeply nested
- Mango plugin prototype

Hello World

```
package hello;

public class HelloWorld {

  /**
   * @param args
   */
  public static void main(String[] args) {
    System.out.println("Hello World!");

  }

}
```

assumptions and conjectures

```
static void clear(int[] x){
  for(int i=0;i<10;++i){
    x[i]=0;
  }
}
```

ItsAWrap, proving x[5]==x[6]

```
public class ItsAWrap {

  public static boolean main (int[] x) {
    clear(x);
    if(x[5]==x[6]){
      return true;
    }
    return false;
  }

  static void clear(int[] x){
    for(int i=0;i<10;++i){
      x[i]=0;
    }
  }
}
```

CarRecall case splits and variables

```
public class CarRecall {
   public static void main(String[] args){
      int modelNum;
      Scanner input = new Scanner (System.in);
      do{
         System.out.println("Enter the car's model number or 0 to quit: ");
         modelNum=input.nextInt();

         //test if car is defective
         if ((modelNum==119)||(modelNum==179)
              ||((modelNum>=189)&&(modelNum<=195))
              ||(modelNum==221)||(modelNum==780))
         {
            System.out.println("Your car is defective. It must be repaired.");
         }
         else if (modelNum==0)
         {
            System.out.println ("Program terminated.");
         }
         else
         {
            System.out.println ("Your car is not defective.");
         }
      }while (modelNum!=0);
   }
}
```

CASE 1 (modelNum==0)
CASE 2 (modelNum== 780)
CASE 3 (modelNum== 221)

```
Scanner input = new Scanner (System.in);
do{
   System.out.println("Enter the car's model number or 0 to quit: ");
   modelNum=input.nextInt();

   //test if car is defective
   if ((modelNum==119)||(modelNum==179)
        ||((modelNum>=189)&&(modelNum<=195))
        ||(modelNum==221)||(modelNum==780))
   {
      System.out.println("Your car is defective. It must be repaired.");
   }
   else if (modelNum==0)
   {
      System.out.println ("Program terminated.");
   }
   else
   {
      System.out.println ("Your car is not defective.");
   }
}while (modelNum!=0);
```

3

2

1

Tic tac toe, deeply nested

```
1   package firstYearCode;
2   import java.util.Scanner;
3   public class tictactoe
4   {
5       public static void mai
6       {
7           int x;
8           int y;
9           String winner="";
10          double moves=0;
11          boolean gameover=f
12          int row;
13          int col;
14          String[]names=new
15          String [][]board=n
16          Scanner input=new
17          row=board.length;
18          col=board[0].lengt
19          for (x=0;x<row;x++
20          {
21              for (y=0;y<col
22              {
23                  if(y<row-1
24                      board[
25                  else
26                      board[
27              }
28          }
29          //reading in playe
30          System.out.println
31          names[0]=input.nex
32          System.out.println
33          names[1]=input.nex
34          //instructions
35          System.out.println
36          System.out.println
```

```
51          while (!gameover)
52          {
53              moves++;
54              if (moves/2!=(int)moves/
55              {
56                  do
57                  {
58                      do
59                      {
60                          System.out.p
61                          x=input.next
62                      }while(x>2||x<0)
63                      do
64                      {
65                          System.out.p
66                          y=input.next
67                      }while(y>2||y<0)
68                  }while (!board[x][y]
69                  if (y<row-1)
70                      board[x][y]="  X
71                  else
72                      board[x][y]="  X
73                  for(x=0;x<row;x++)//
74                  {
75                      for (y=0;y<col;y
76                          System.out.p
77                      if (x<row-1)
78                      {
79                          System.out.p
80                          System.out.p
81                          System.out.p
82                      }
83                  }
84                  System.out.println (
85
86          }
```

```
118         }
119
120         if (moves==9)
121         {
122             gameover=true;
123         }
124         //checks if Player 1 won
125         if (board [1][1].equals("  X  |"
126         {
127             if ((board[0][0].equals("  X
128                 (board[0][2].equals("  X
129                 (board[0][1].equals("  X
130                 (board[1][0].equals("  X
131             {
132                 gameover=true;
133                 winner=names[0].concat("
134             }
135         }
136         if (board [0][0].equals("  X  |"
137         {
138             if ((board[0][1].equals("  X
139                 (board[1][0].equals("  X
140             {
141                 gameover=true;
142                 winner=names[0].concat("
143             }
144         }
145         if (board[2][2].equals("  X  "))
146         {
147             if ((board[2][0].equals("  X
148                 (board[0][2].equals("  X
149             {
150                 gameover=true;
151                 winner=names[0].concat("
152             }
153         }
```

Mango plugin prototype

- Call chain becomes subdivision hierarchy
- Source code view is the same
- Control buttons and breakpoints for mango model build (symbolic simulation)
- Assumptions and conjectures in peer views to “variables”, which now exhibit symbolic state
- Full support for hyper-linking, navigation of nested state is just object inspection.

Conclusions

- Mango creates a good model for explaining code.
- Merely rendering this model as html is totally inadequate for an end user.
- Eclipse provides an excellent framework for realizing the potential of the tool.
- Hiding/revealing detail, hyper-linking, using debugger semantics to control creation and inspection of specification, managing persistence and update.

References

- Mango is an opensource project, part of JPF of NASA/AMES. The JPF website is
- http://babelfish.arc.nasa.gov/trac/jpf/
- Email address for the speaker, Frank Rimlinger is
- frankrimlinger@gmail.com
- The Mango wiki is
- http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/jpf-mango
- The Mango plugin update site (under development)
- http://babelfish.arc.nasa.gov/trac/jpf/raw-attachment/wiki/projects/jpf-mango/update/
- The open source repository for the code is
- http://babelfish.arc.nasa.gov/hg/jpf/jpf-mango
- The original submission for EclipseCon, is at
- http://www.slideshare.net/frankrimlinger/
- The ACL2 website is
- http://www.cs.utexas.edu/users/moore/acl2/
- The website for ACL2s, an Eclipse plugin for ACL2, can be found at
- http://acl2s.ccs.neu.edu/acl2s/doc/