

## **Wikiprint Book**

**Title:** projects/jpf-mango/examples/ltsAWrap

**Subject:** Java Path Finder - projects/jpf-mango/examples/ltsAWrap

**Version:** 28

**Date:** 03/01/2013 10:35:45 AM

## Table of Contents

ItsAWrap example, special considerations for loops

3

## ItsAWrap example, special considerations for loops

This example focusses on issues that arise in the consideration of loops. In particular, the introduction of "shadow rules" to imply the truth of automatically generated loop-exit conjectures. This example assumes you have already worked through the [Hello World](#) and [CarRecall](#) examples. If not, you might find the steps a little hard to follow. However, links have been provided to parallel steps in the earlier examples for your convenience.

Start by [loading](#) the ItsAWrap project, located inside the "rbk" folder.

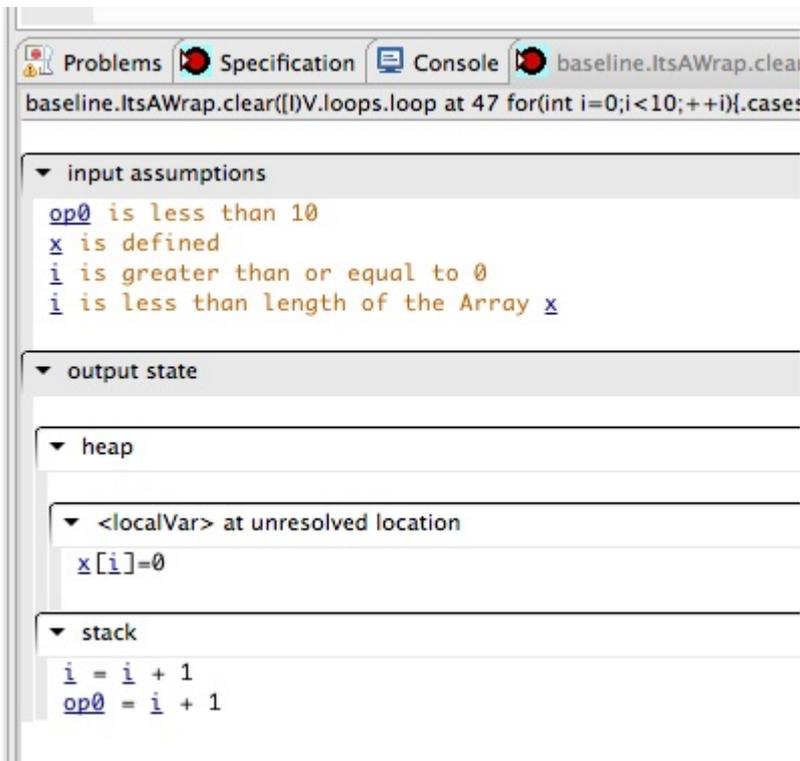
[Restore the default preferences](#), then **check** preferences "Suppress advice dialogs" and "Do not pause for case-splits". Now **uncheck** "Suppress structural hypotheses".

[Open](#) the ItsAWrap class in an editor window.

[Populate and run](#), using the ItsAWrap class as command target.

Observe that three [specifications are generated](#), for the loop body of the clear method, for the clear method, and for the test method. We shall consider each of these in turn.

[Open](#) the loop specification (see figure).



The specification describes the state transition of one pass through the loop. In order to remain in the loop, "i" must be within bounds. The output state sets  $x[i]=0$  and increments "i". The rest of the specification concerns "op0". The point is that "i" is pushed onto the operand stack for a compare against 10, just prior to the pass through the loop. Likewise, the loop output state pushes "i+1" onto the operand stack.

Now consider the specification for the clear method.

```

baseline.ItsAWrap.clear([I]V.cases.1
└─ input assumptions
  x is defined
  length of the Array x is greater than or equal to 10
  i^ is greater than or equal to 10
└─ output state
  └─ heap
    heap^
    No return value.

```

The "<sup>^</sup>" link indicates a quantity that is derived from the output state of a loop. Clicking on the "<sup>^</sup>" reveals a new page with link to the loop and a separate link to the loop input state. Following the loop link reveals the specification for the loop in the Mango Explorer, whereas following the state link reveals a new page with a description of the loop input state. Notice you can go back and forth between pages of a view using the familiar "forwards arrow", "backwards array" and "home" buttons in the view menu bar.

Now observe that two of the assumptions are green while the third is blue. A blue assumption is actually a conjecture generated by Mango. In this case, the conjecture is the criterion for exiting the loop. A priori, any loop has the potential to hang, so conditions are always generated which imply loop termination. In this case, the loop exit condition states that the loop output value of "i" is greater than or equal to ten. Because this assumption is stated in terms of loop output, it is a conjecture. Mango just doesn't know if this statement is true or false, or possibly true for some input states and false for others. Of course, Mango **should** know, but for now, this functionality is missing, see [bug #7](#).

It is indeed the case that the loop termination conjecture is sometimes true and sometimes false. For if "x" is not defined, then a NullPointerException will be thrown, and if "i" is out of bounds, then an IndexOutOfBoundsException exception will be thrown. Since the "All exceptions are bad local exits" preference is checked, these exit criteria are considered invalid. This brings us to the crux of the matter. Currently the burden is on the user to generate constraints on the input state which imply the turn of the termination conjecture. The two assumptions in green were added by the user for precisely this reason. A moments reflection is enough to see that these assumptions deny the possibility of a thrown exception, and hence the truth of the conjecture. What is required is a mechanism to generate the green assumptions **automatically** in as general a setting as possible. Currently, the green assumptions are created by the attentive user by writing shadow rules (see figure). Unfortunately, the editing facility for these shadow rules is rather buggy, see [bug #2](#). For information on creating rules, see how to [write a rule](#).

The screenshot shows the Mango Explorer interface with two shadow rule configuration pages open. The left page is titled "length of x >= 10" and the right page is titled "length of x". Both pages have a "Pattern" field, an "Action" field, a "Substitution" field, and a "Variables" field. The "Pattern" field for "length of x >= 10" contains the text "( addHypothesis &<modulePath> 'root' )". The "Action" field contains "AddHypothesis". The "Substitution" field contains "( not ( < &"length of x" 10 ) )". The "Variables" field is empty. The right page has a "Pattern" field containing "( parameterMap ( valueH ( loc &x ^arrayLength ) @inputHeap ) )", an "Action" field containing "ShadowParameter", and an empty "Substitution" and "Variables" field.

Problems Specification Console baseline.ItsAWrap.clear([I]V.lo

baseline.ItsAWrap.test([I]Z.cases.1

input assumptions

- x is defined
- length of the Array x is greater than or equal to 10
- x[5]^ equals x[6]^

output state

- heap
  - heap^

Returns boolean: true

main good case

main nonexistent case

length of x >= 10 length of x

Problems Specification Console baseline.ItsAWrap.clear([I] cle

baseline.ItsAWrap.test([I]Z.cases.2

input assumptions

- x is defined
- length of the Array x is greater than or equal to 10
- x[5]^ does not equal x[6]^

output state

- heap
  - heap^

Returns boolean: false