Wikiprint Book

Title: projects/jpf-mango/examples/ItsAWrap

Subject: Java Path Finder - projects/jpf-mango/examples/ItsAWrap

Version: 28

Date: 02/27/2013 05:36:24 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.

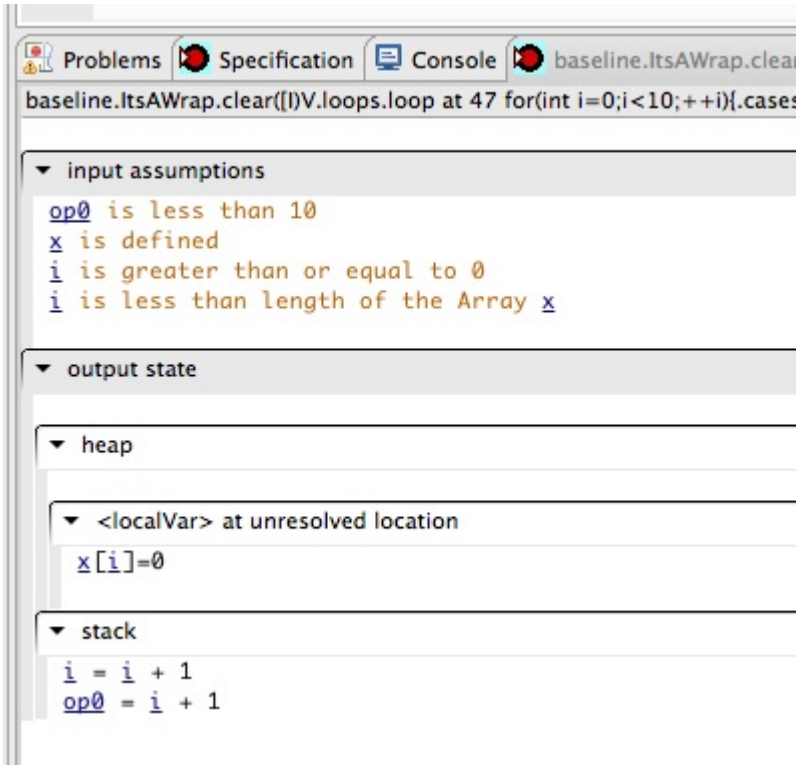
Restore the default preferences, then **check** preferences "Suppress advice dialogs" and "Do not pause for case-splits".

Open the ItsAWrap class in an editor window.

Populate and run, using the ItsAWrap class as command target.

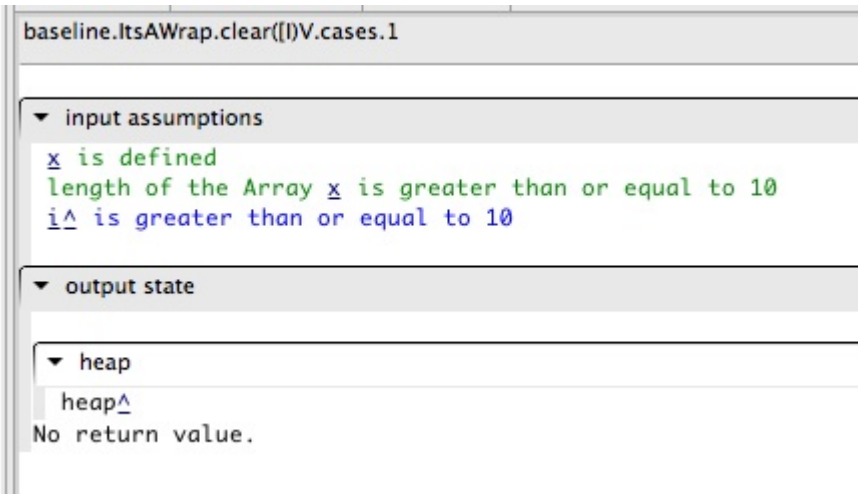
Observe that three specifications are generated,for the loop body of the clear method, for the clear method, and for the test method. We shall consider each of these in turn.

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.

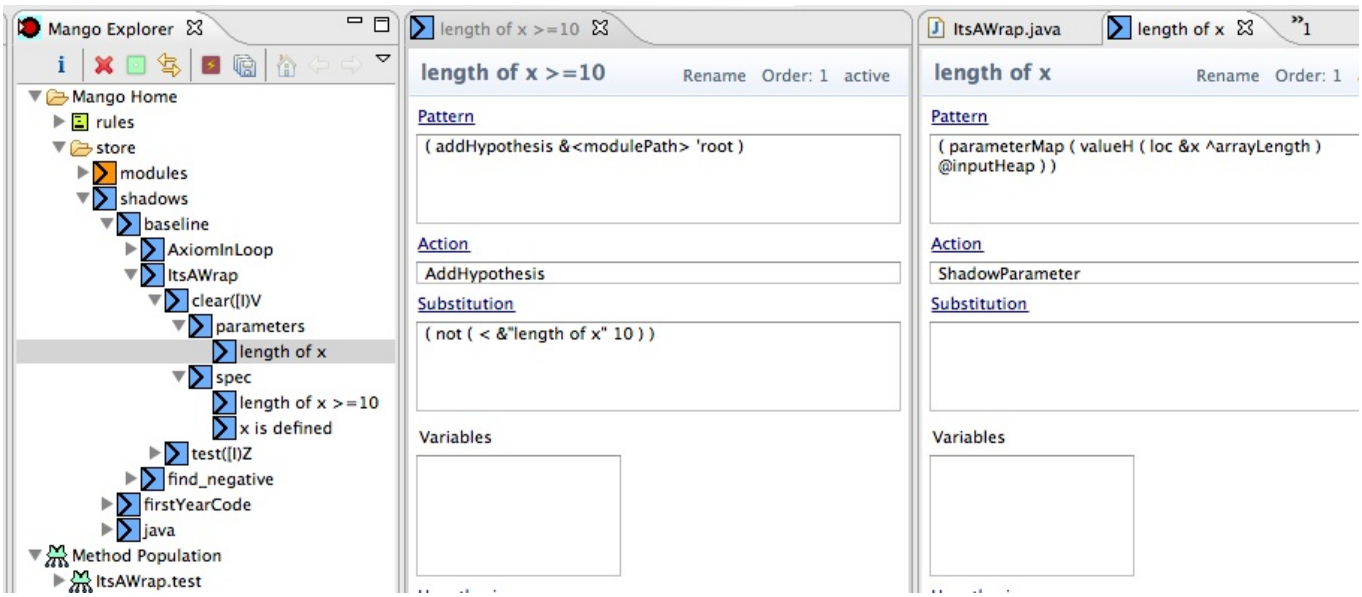
Now consider the specification for the clear method.



The "^" link indicates a quantity that is derived from the output state of a loop. Clicking on the "^" 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.

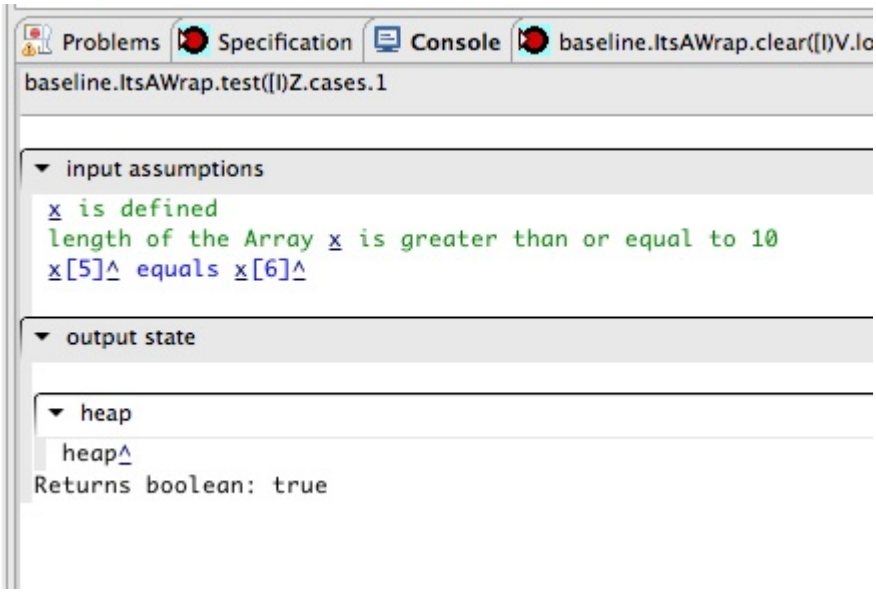
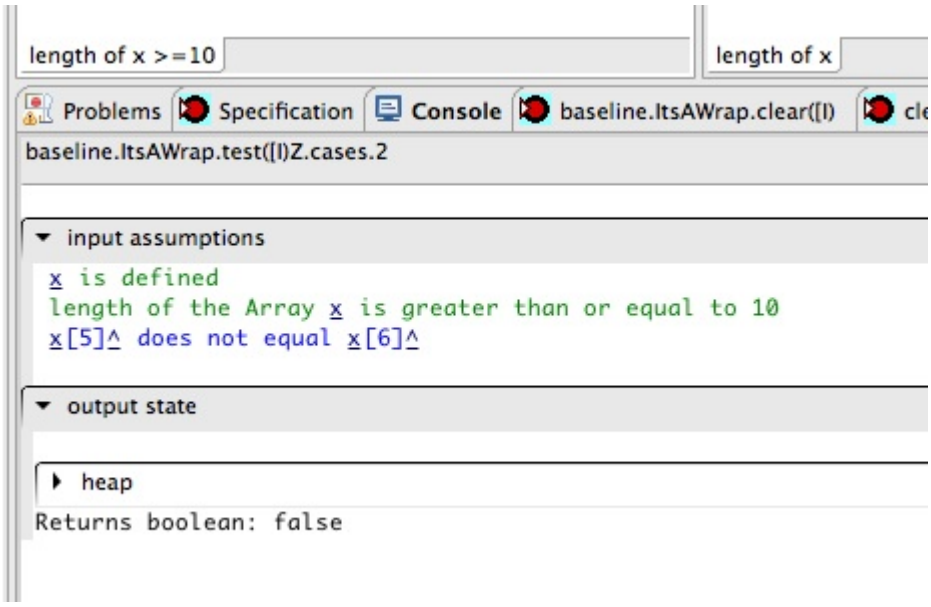
Now observe that two of the assumptions are green while the third is blue. A blue assumption is actually a conjecture generated by Mango. In this case, the conjecture is the criterion for exiting the loop. A priori, any loop has the potential to hang, so conditions are always generated which imply loop 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 IndexOutOfBounds 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 below). 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 specification of the main routine breaks out as two cases, depending on whether or not x[5] equals x[6]. The case x[5] equals x[6] is show below. Observe that the "^" signs are once again present, indicating that these values depend on loop output state, specification on the content of x that was loaded by the loop execution. Technically, the assumptions are deduced by Mango, but since they derive from assumptions generated by the user, they are still shown in green. The assumptions "x[5] equals x[6]" is in blue, since it depends on loop output state and therefore is a conjecture. In the past, Mango generated proof artifacts for this conjecture, and a proof was generated by the ACL2 theorem prover. For some ideas that are still relevant, even

through the output is obsolete, see Local Forward Flow and Prototype Deductive System Currently, the pipeline to ACL2 is broken, see bug #6. This is an area where contributions would be most welcome, as automated theorem proving is the long game.



Finally, observe that Mango generates the vacuous case x[5] does not equal x[6], even though it is impossible (see figure below). A conscientious could introduce a shadow rule to falsify this case, but again, once automatic generation of loop termination hypotheses and automated theorem proving are hooked up, this case will disappear automatically.