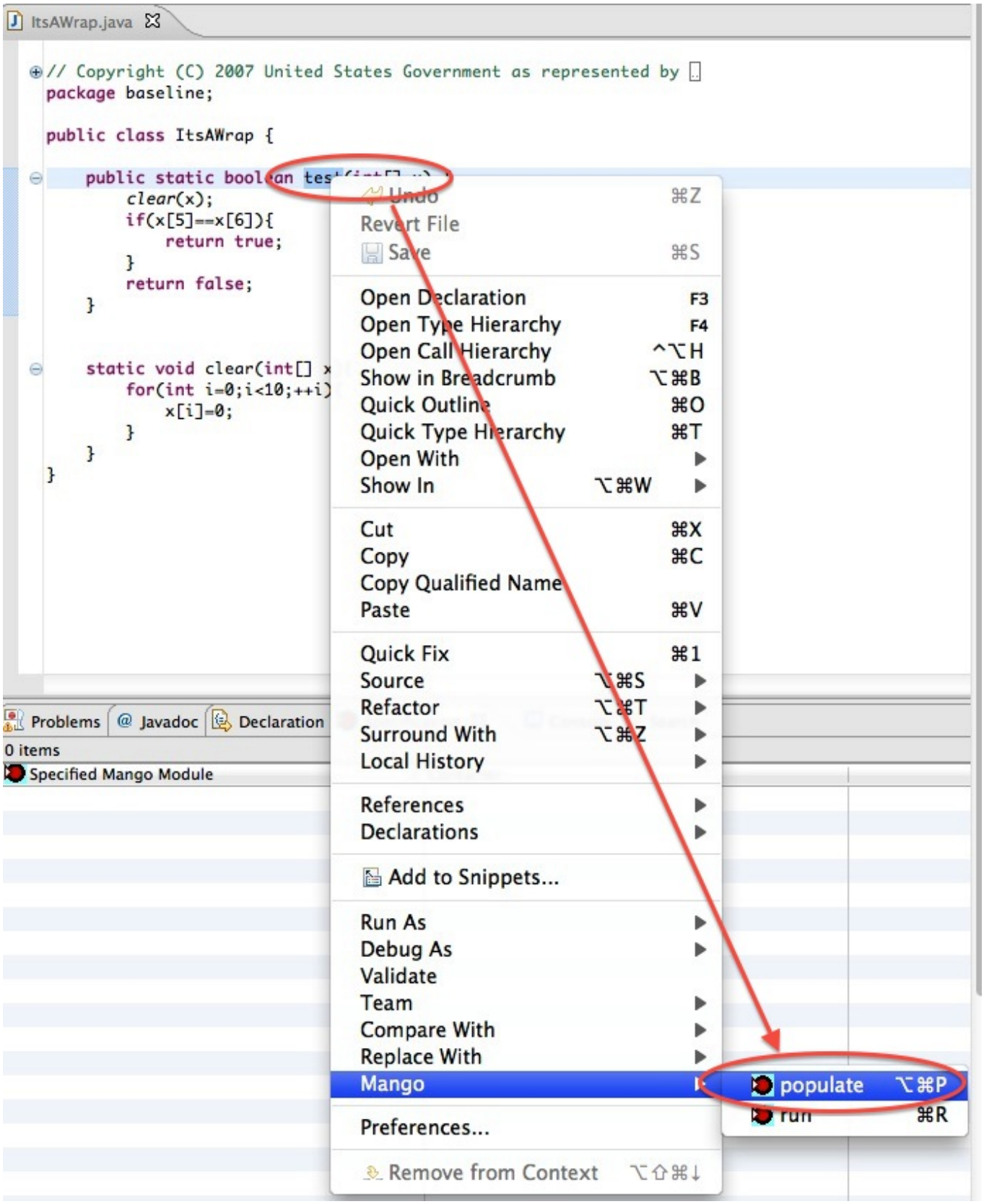
Mango Eclipse plugin introductory demo

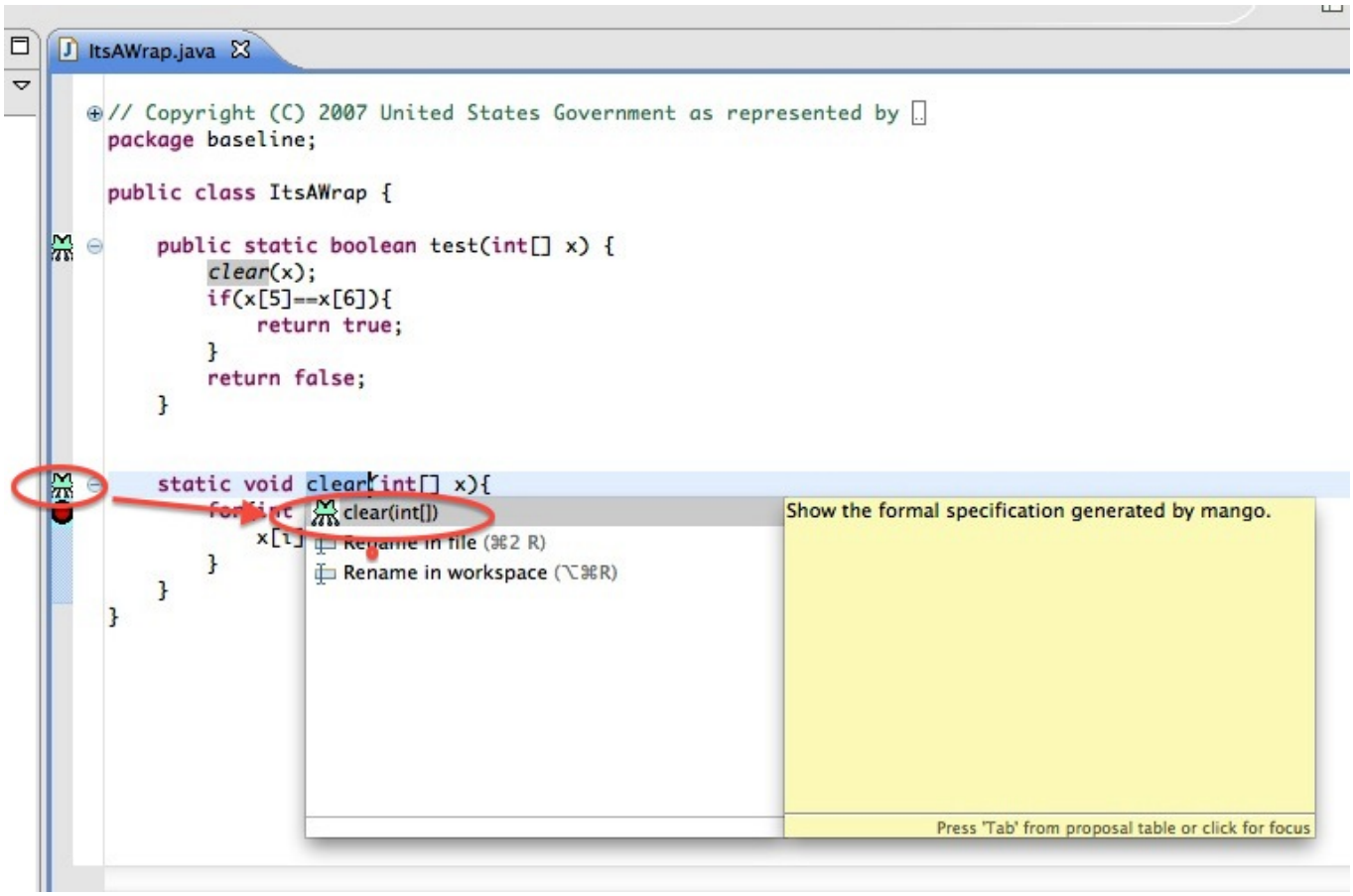
Of course, you have to get the plugin and do some installation chores. Details on this are still TBD, but will be forthcoming before the anticipated release date, around Feb 2012. This demo walks through the Mango specification of the "ItsAWrap" code (see Figure 1). To get the ball rolling, first right-click on a method and build a "Method Population". The population is just the set of all dependencies. For example, the ItsAWrap.test() routine has only one dependency, clear(), which in turn has no dependencies. In general, if a method has a lot of dependencies, the consequent analysis can easily overwhelm Mango, so its best to start small and build specifications bottom up. When the population is ready, (and not too big), right click again and select "Run".

Figure 1: Build and Run



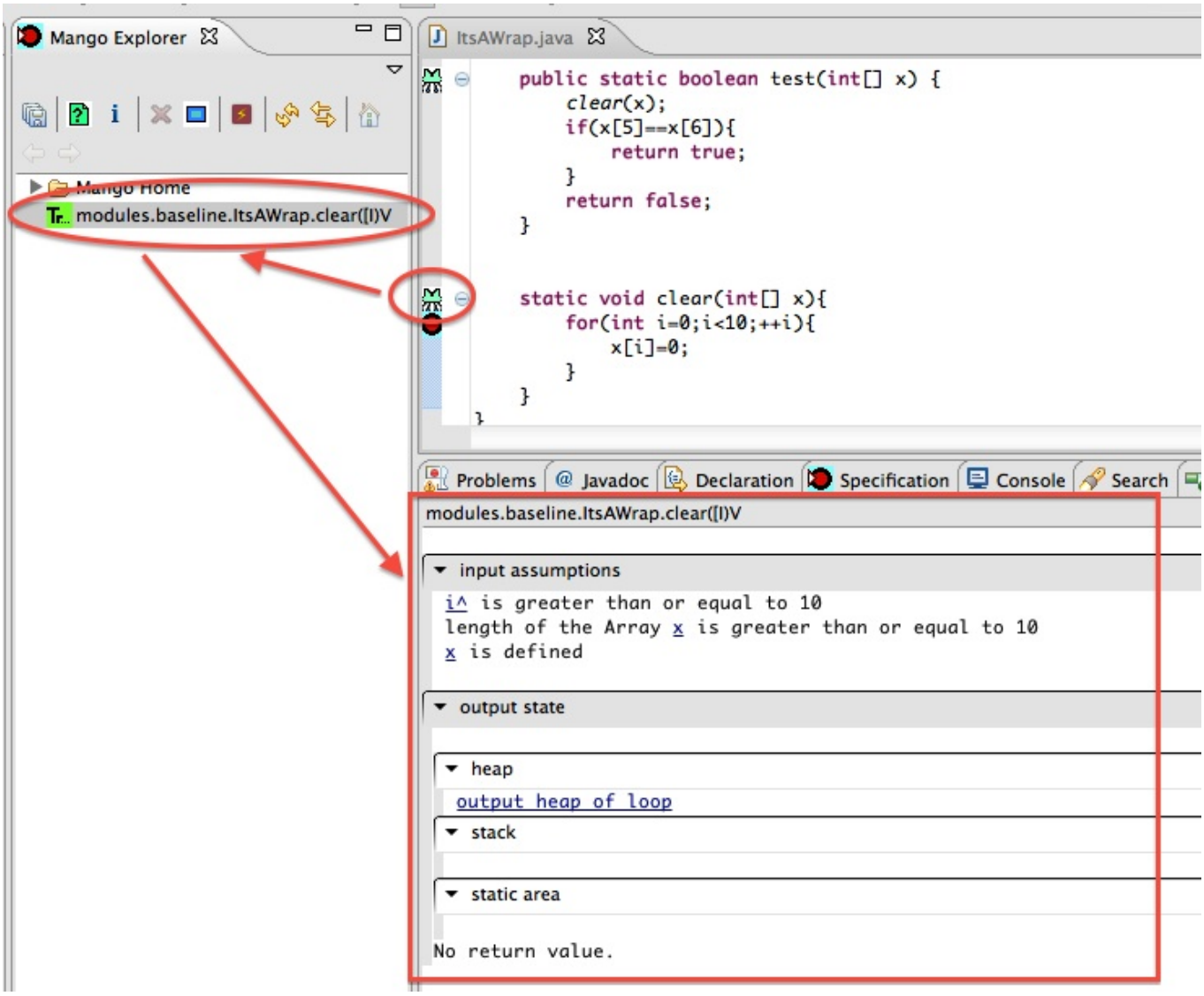
The "Run" command causes icons to appear in the source code window, where the familiar debugging icons appear during compilation (see figure 2). Clicking on the icon for the clear() method will reveal all the activities available at this line. Double-click on the indicated choice to show the formal specification generated by mango

Figure 2: Click on "Module" Icon



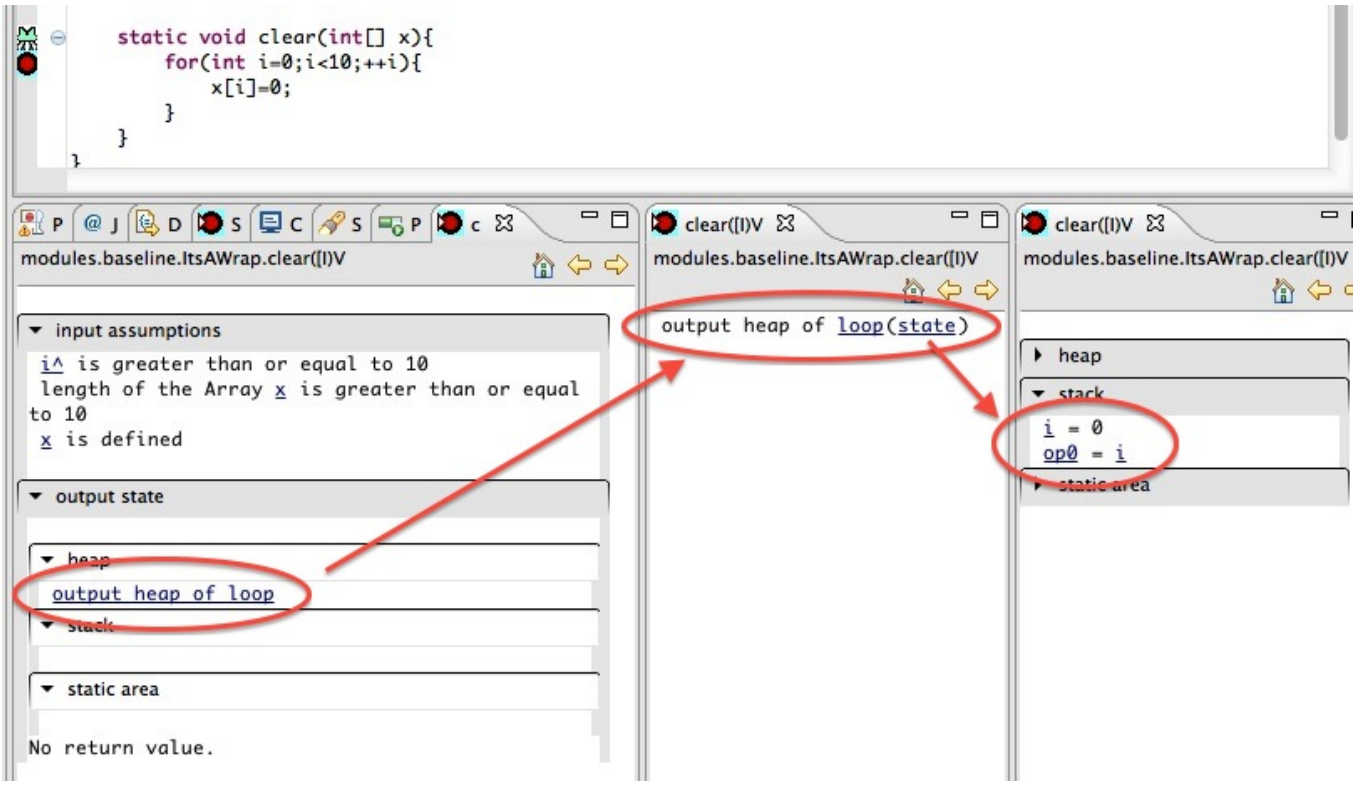
A green "Tr" translation icon will appear in the Mango Explorer (see figure 3). Double click on this to reveal the specification "module" view for the clear() method.

Figure 3: Double-Click on Green Translation Icon



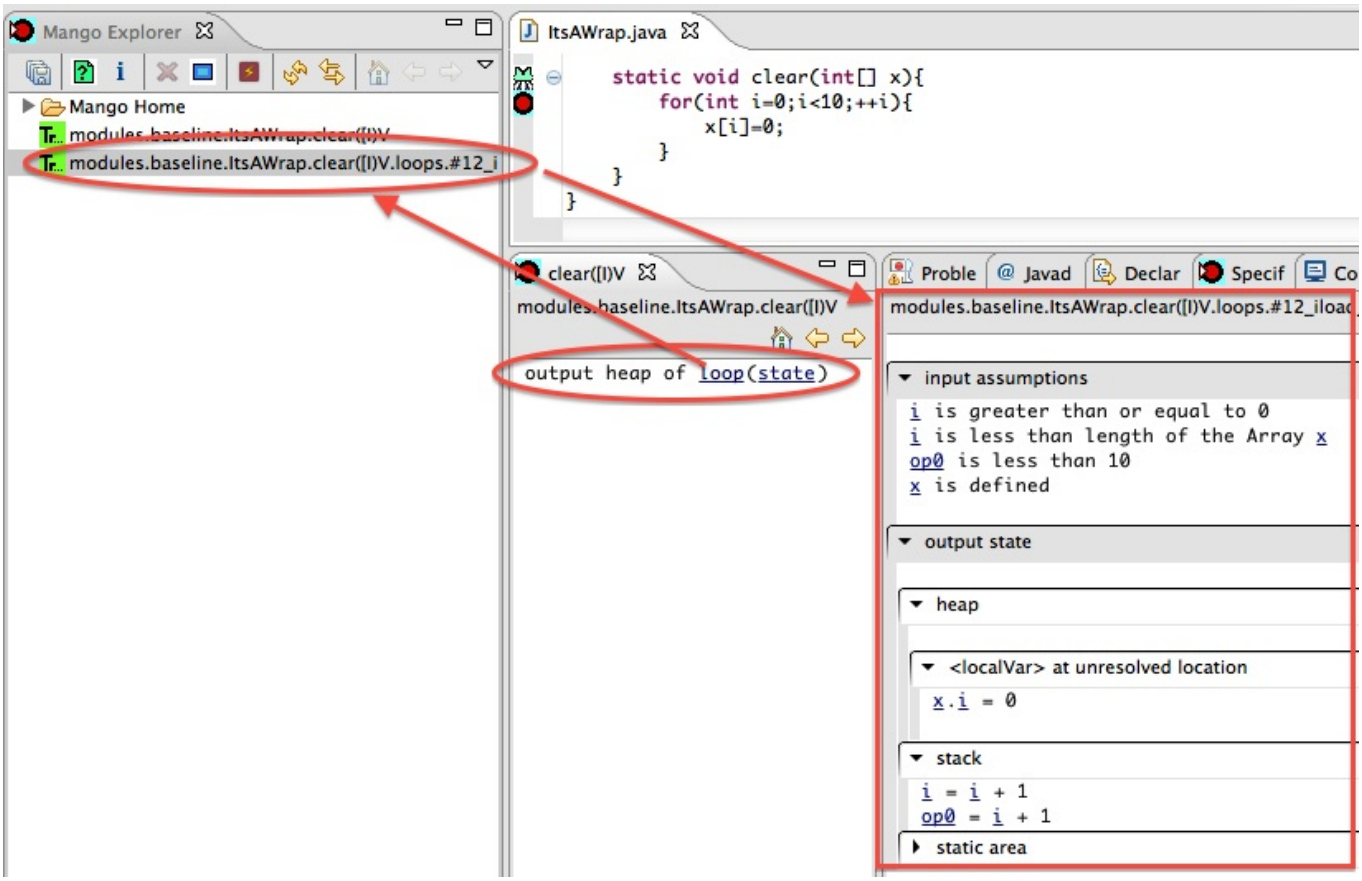
You can move around inside the specification, just like in a browser. For example, clicking on "Output heap of loop" (see figure 4) brings up a new sheet, shown as a separate view here for demo purposes. Now clicking on state will reveal the input state for the loop inside the clear() method.

Figure 4: Click on "Output heap of loop"



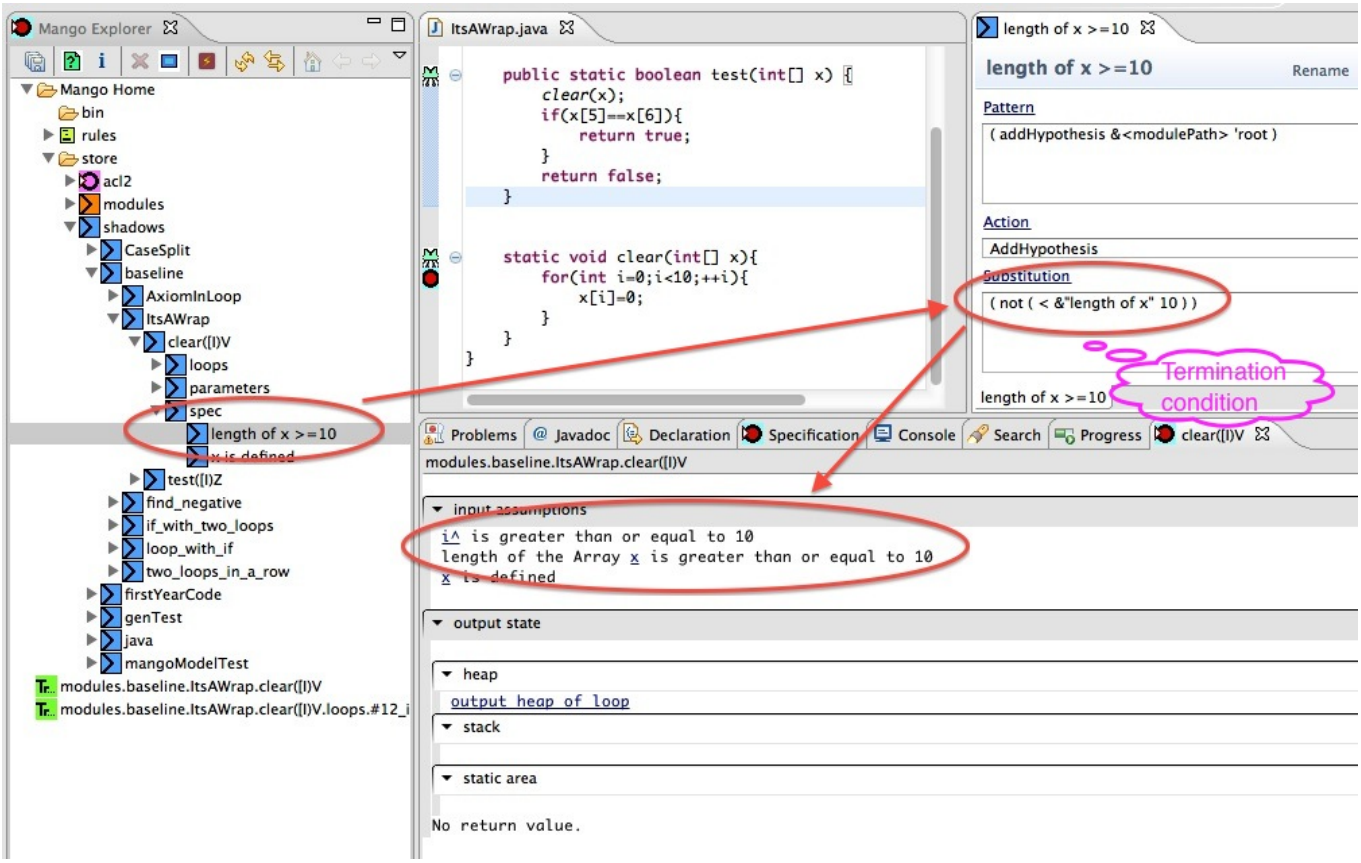
Alternatively, clicking on "loop" (see figure 5) will first bring another green "Tr" for the loop. Double-click on this new icon to get the input assumptions and output state representing a single pass through the loop.

Figure 5: Click on "loop"



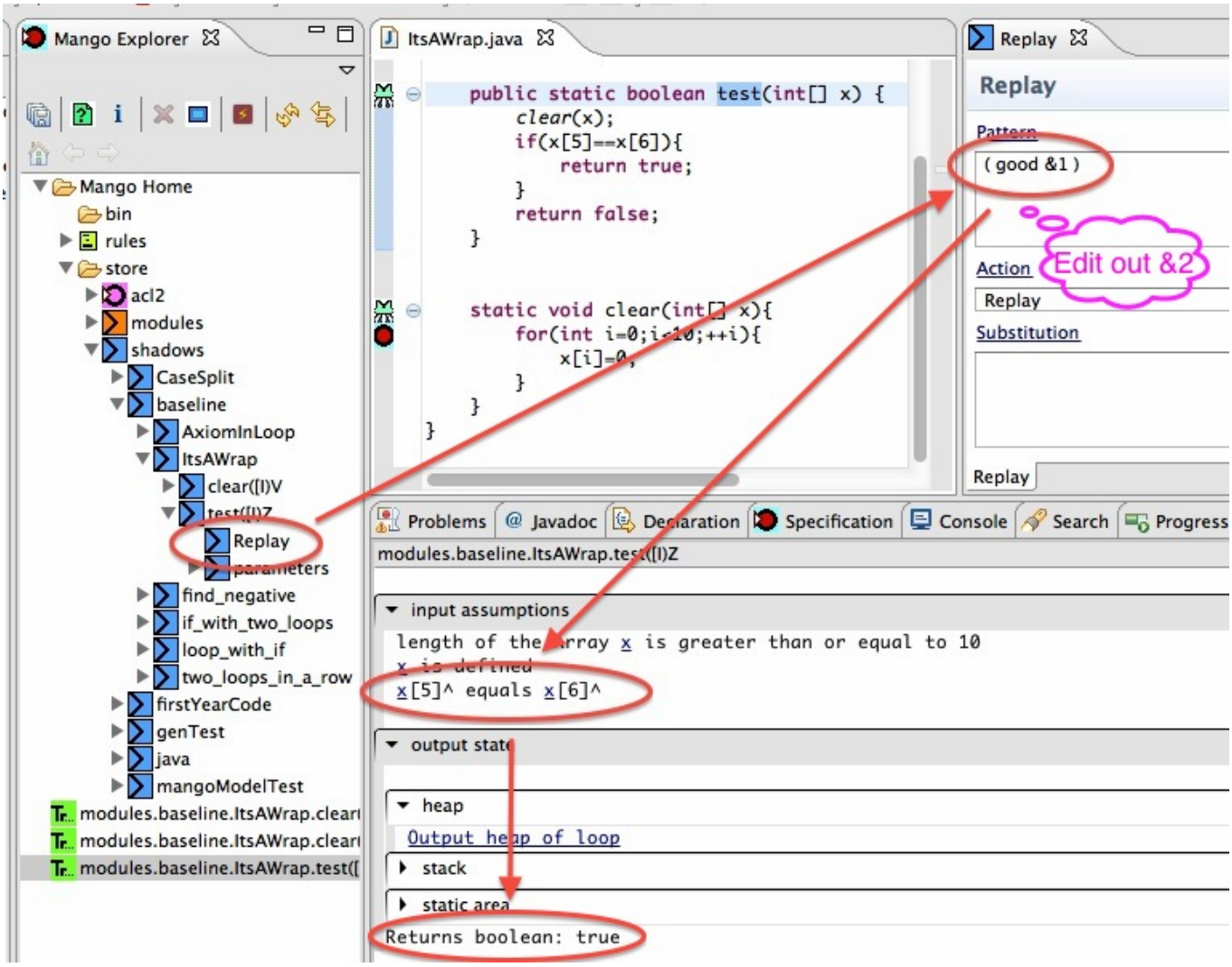
Now going back to the clear method, some quid-pro-quos (see figure 6). First, the "^" symbol generally indicates output from some recursive process. The loop termination conjecture "i^ is greater than or equal to 10" is generated by Mango. In order to guarantee loop termination, the hypothesis "length of x>=10" must currently be entered by hand as a "shadow rule". In the future these shadow rules will be generated automatically for typical loops.

Figure 6: Quid-Pro-Quos



Finally, let's take a look at the specification for test() (see figure 7). Observe that the spec returns true, assuming x[5]^==x[6]^. This can in fact be proven from the assumption on the length of the array x, but Mango is not a theorem-prover. However, Mango can be used to generate input for the ACL2 theorem-prover, which can produce such proofs. It is an interesting research problem to make this proof pipeline as automated as possible. Meanwhile, to manually suppress the unwanted case x[5]!=x[6], the user edits the "Replay" generated by mango. In this case "(good &1 &2) was edited to become (good &1). That's all for the first example, ItsAWrap.

Figure 7: Specification for test()


Mango Explorer
Mango Home
bin
rules
store
acl2
modules
shadows
CaseSplit
baseline
AxiomInLoop
ItsAWrap
clear([I)V
test([I)Z
Replay
parameters
find_negative
if_with_two_loops
loop_with_if
two_loops_in_a_row
firstYearCode
genTest
java
mangoModelTest
modules.baseline.ItsAWrap.clear
modules.baseline.ItsAWrap.clear
modules.baseline.ItsAWrap.test([
ItsAWrap.java
public static boolean test(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;
}
}
}
Replay
Replay
Pattern
(good &1)
Edit out &2
Action
Replay
Substitution
Replay
Problems
Javadoc
Declaration
Specification
Console
Search
Progress
modules.baseline.ItsAWrap.test([I)Z
input assumptions
length of the array x is greater than or equal to 10
x is defined
x[5]^ equals x[6]^
output state
heap
Output heap of loop
stack
static area
Returns boolean: true