Formal Methods: Lecture 17

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Proving while loops with the ITP

The `javax` command allows a user to enter a Hoare triple goal into the ITP to prove the correctness of the program mentioned in the triple.

However, the compositional approach favored by Hoare logic suggests that we should fist decompose the original Hoare triple into simpler ones by using the Hoare logic inference system.

For this reason, the Java+ITP tool not only does automate the entering of Hoare triples into the ITP, but automates also the application of some Hoare rules. Specifically, this is accomplished for Hoare triples involving while loops by means of the `javax-inv` command.

Proving while loops with the ITP (II)

We consider while loop programs of the general form

`wlp = init loop`

with

`loop= while t do p endo.`

That is, we allow the subprogram `init` to be executed before the while loop proper, since this is a very common situation.

Proving while loops with the ITP (III)

The `javax-inv` command allows the specification of the following information about a while loop program `wlp` of the form just described:

- the precondition P and postcondition Q against which one wants to prove `wlp` correct.

- the invariant A that should be used to decompose the original Hoare triple into simpler ones using the Hoare rules.

Proving while loops with the ITP (IV)

The `javax-inv` command then does the following things:

- it applies the composition rule to:

$$\frac{\{P\}\ \texttt{init}\ \{A\} \qquad \{A\}\ \texttt{loop}\ \{Q\}}{\{P\}\ \texttt{init loop}\ \{Q\}}$$

- it then applies the consequence rule to further decompose the second subgoal

$$\frac{A \Rightarrow A \qquad \{A\}\ \texttt{loop}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\} \qquad (A \wedge (\texttt{evalTst(S,t)} = \texttt{false})) \Rightarrow Q}{\{A\}\ \texttt{loop}\ \{Q\}}$$

- it finally applies the loop rule. (Note that the partial correctness interpretation implicitly includes a termination assumption in the semantics of the Hoare triple.)

$$\frac{\{A \wedge (\texttt{evalTst(S,t)} = \texttt{true})\}\ \texttt{t}\ \texttt{|}\ \texttt{p}\ \{A\} \quad \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}\ \texttt{t}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}}{\{A\}\ \texttt{while t p}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}}$$

As a consequence, the following four subgoals are produced:

1. $\{P\}\ \texttt{init}\ \{A\}$
2. $\{A \wedge (\texttt{evalTst(S,t)} = \texttt{true})\}\ \texttt{t}\ \texttt{|}\ \texttt{p}\ \{A\}$
3. $(A \wedge (\texttt{evalTst(S,t)} = \texttt{false})) \Rightarrow Q$
4. $\{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}\ \texttt{t}\ \{A \wedge (\texttt{evalTst(S,t)} = \texttt{false})\}$.

Proving while loops with the ITP (V)

The implementation of the `javax-inv` command in the Java+ITP tool then implicitly applies the `javax` command to the Hoare triples (1) , (2) and (4) so that we end up with the following four ITP goals:

1. $P \Rightarrow A(S/S\texttt{|init})$

2. $(A \wedge (\texttt{evalTst(S,t)} = \texttt{true})) \Rightarrow A(S/S\texttt{|t|p})$

3. $(A \wedge (\texttt{evalTst(S,t)} = \texttt{false})) \Rightarrow Q$

4. $(A \wedge (\texttt{evalTst(S,t)} = \texttt{false})) \Rightarrow$
   $A(S/S\texttt{|t}) \wedge (\texttt{evalTst}(S/S\texttt{|t},t) = \texttt{false})$

A Factorial Example

We will show the use of the `javax-inv` command with a factorial program defined in `facx.maude`.

```
fmod FACX-JAVAX is including JAVAX .
op facValue : Int -> Int .
var I : Int .
ceq facValue(I) = 1 if 0 < I = false .
ceq facValue(I) = I * facValue(I - 1) if 0 < I = true .
op facx : -> BlockStatements .
op facx-init : -> BlockStatements .
eq facx =
   'C = #i(0) ; 'X = #i(1) ;
   while ('C < 'N) {
      'C = 'C + #i(1) ;
      'X = 'X * 'C ;
      } .
eq facx-init = (int 'C ; int 'X ; int 'N ;) .
endfm
```

The Hoare triple we want to show for this is, with N an integer parameter,

$$\{\texttt{S['N]} = N \wedge 0 \leq N\}\ \texttt{facx-init facx}\ \{\texttt{S['X]} = \texttt{facValue}(N)\}$$

and we use the invariant

$$\texttt{S['X]} = \texttt{facValue(S['C])} \wedge 0 <= \texttt{S['C]} \wedge \texttt{S['C]} <= \texttt{S['N]} \wedge \texttt{S['N]} = N$$

to show that those pre- and postconditions do indeed hold.

A Factorial Example (II)

By using `javax-inv` we can now give our desired Hoare triple directly to Java+ITP. We do that by (1) loading `java-es.maude` and `facx.maude`; (2) loading `itp-tool.maude`; and (3) giving the `javax-inv` command:

```
(javax-inv FACX-JAVAX :
--- specification constants
(N:Int)
--- precondition
((int-val(S:WrappedState['N])) = (N:Int)
& (0 <= N:Int) = (true))
--- program
facx-init
facx
--- postcondition
((int-val(S:WrappedState['X]))
= (facValue(N:Int)))
```

```
--- invariant
((int-val(S:WrappedState['X]))
= (facValue(int-val(S:WrappedState['C])))
&
(0 <= int-val(S:WrappedState['C]))
= (true)
&
(int-val(S:WrappedState['C]) <=
 int-val(S:WrappedState['N]))
= (true)
&
(int-val(S:WrappedState['N]))
= (N:Int))
.)
```

A Factorial Example (III)

Java+ITP responds with four ITP goals that are semantically equivalent to the Hoare triples which are created by the above rule for loops. The first goal states that the invariant holds after executing the initial code started in a state where the precondition holds.

```
==================================
label-sel: facx@1.0
==================================
```

```
A{S:WrappedState ; N:Int}
    (((
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        ['N])
      = N:Int)
    &
      (0 <= N:Int = true))
  ==>
    ((((
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
```

```
          out(noOutput))
    |('C = #i(0); 'X = #i(1);)
    ['X])
  =
    facValue(
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        |('C = #i(0); 'X = #i(1);)
        ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
             out(noOutput))
     |('C = #i(0); 'X = #i(1);)
```

```
      ['N])
   = N:Int))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
             out(noOutput))
     |('C = #i(0); 'X = #i(1);)
     ['C])
   <=
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
                out(noOutput))
        |('C = #i(0); 'X = #i(1);)
        ['N])
   = true))
```

```
&
  (0
   <=
   int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
             out(noOutput))
     |('C = #i(0); 'X = #i(1);)
     ['C])
   = true)))
```

A Factorial Example (IV)

Using `auto` on the first goal results in a goal with three conjuncts. Using `(cnj .)` three times we decompose this into the following four subgoals:

```
==================================
label-sel: facx@1.1.1.1.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C = #i(0); 'X = #i(1);)
  ['X])
=
facValue(
```

```
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C = #i(0); 'X = #i(1);)
    ['C]))

==================================
label: facx@1.1.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C = #i(0); 'X = #i(1);)
  ['N])
=
```

```
  N*Int

==================================
label: facx@1.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C = #i(0); 'X = #i(1);)
  ['C])
<=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C = #i(0); 'X = #i(1);)
```

```
    ['N])
= true

==================================
label: facx@1.2.0
==================================
0 <=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C = #i(0); 'X = #i(1);)
    ['C])
= true
```

These 4 goals are all discharged using `(auto .)` and therefore the first goal is proven.

A Factorial Example (V)

The second goal asserts that the invariant and the negation of the loop condition imply the postcondition.

```
==================================
label: facx@2.0
==================================
A{S:WrappedState ; N:Int}
  ((((((int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        ['X])
      =
        facValue(
          int-val(
```

```
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     ['N])
   = N:Int))
&
  (evalTst(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
```

```
            out(noOutput)),
      'C < 'N)
   = false))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     ['C])
   <=
   int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      ['N])
   = true))
&
```

```
    (0
     <=
     int-val(
       state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
             m([any-loc(0),int(any-int(0))]
               [any-loc(1),int(any-int(1))]
               [any-loc(2),int(any-int(2))]),
             out(noOutput))
       ['C])
     = true))
==>
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     ['X])
   =        facValue(N:Int)))
```

This second goal is immediately discharged by `(auto .)`

A Factorial Example (VI)

The third goal asserts that the invariant actually is an invariant.

```
================================
label: facx@3.0
================================
A{S:WrappedState ; N:Int}
    ((((((
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        ['X])
      =
      facValue(
```

```
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
       ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     ['N])
   = N:Int))
&
  (evalTst(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
```

```
            [any-loc(2),int(any-int(2))]),
          out(noOutput)),
    'C < 'N)
  = true))
&
 (int-val(
   state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
         m([any-loc(0),int(any-int(0))]
           [any-loc(1),int(any-int(1))]
           [any-loc(2),int(any-int(2))]),
         out(noOutput))
   ['C])
  <=
  int-val(
   state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
         m([any-loc(0),int(any-int(0))]
           [any-loc(1),int(any-int(1))]
           [any-loc(2),int(any-int(2))]),
         out(noOutput))
   ['N])
  = true))
```

```
  &
    (0
     <=
     int-val(
       state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
             m([any-loc(0),int(any-int(0))]
               [any-loc(1),int(any-int(1))]
               [any-loc(2),int(any-int(2))]),
             out(noOutput))
       ['C])
     = true))
==>
  ((((int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
             m([any-loc(0),int(any-int(0))]
               [any-loc(1),int(any-int(1))]
               [any-loc(2),int(any-int(2))]),
             out(noOutput))
        |('C < 'N)
        |{'C = 'C + #i(1); 'X = 'X * 'C ;}
        ['X])
```

```
    =
    facValue(
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        |('C < 'N)
        |{'C = 'C + #i(1); 'X = 'X * 'C ;}
        ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     |('C < 'N)
     |{'C = 'C + #i(1); 'X = 'X * 'C ;}
     ['N])
```

```
   = N:Int))
&
   (int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
      |('C < 'N)
      |{'C = 'C + #i(1); 'X = 'X * 'C ;}
      ['C])
    <=
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
      |('C < 'N)
      |{'C = 'C + #i(1); 'X = 'X * 'C ;}
      ['N])
```

```
    = true))
&
  (0
   <=
   int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     |('C < 'N)
     |{'C = 'C + #i(1); 'X = 'X * 'C ;}
     ['C])
   = true)))
```

A Factorial Example (VII)

After one application of `(auto .)` this third goal contains some conjunctions. After three applications of `(cnj .)` we end up with the four subgoals:

```
==================================
label-sel: facx@3.1.1.1.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)
  |{'C = 'C + #i(1); 'X = 'X * 'C ;}
  ['X])
=
```

```
  facValue(
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      |('C < 'N)
      |{'C = 'C + #i(1); 'X = 'X * 'C ;}
      ['C]))
==================================
label: facx@3.1.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)
  |{'C = 'C + #i(1); 'X = 'X * 'C ;}
```

```
  ['N])
=      N*Int
==================================
label: facx@3.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)
  |{'C = 'C + #i(1); 'X = 'X * 'C ;}
  ['C])
<=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
```

```
    |('C < 'N)
    |{'C = 'C + #i(1); 'X = 'X * 'C ;}
    ['N])
= true
==================================
label: facx@3.2.0
==================================
0 <=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C < 'N)
    |{'C = 'C + #i(1); 'X = 'X * 'C ;}
    ['C])
= true
```

These goals can all be discharged with the (auto .) command, concluding the proof of goal 3.

A Factorial Example (VIII)

The fourth goal asserts that the execution of the condition keeps the invariant and negation of the condition true.

```
================================
label: facx@4.0
================================
A{S:WrappedState ; N:Int}
    ((((((
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        ['X])
      =
      facValue(
```

```
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
       ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
      ['N])
   = N:Int))
&
  (evalTst(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
```

```
            [any-loc(2),int(any-int(2))]),
          out(noOutput)),
    'C < 'N)
  = false))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     ['C])
  <=
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      ['N])
  = true))
```

```
  &(0
    <=
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      ['C])
    = true))
==>
  (((((
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      |('C < 'N)
      ['X])
    =
```

```
    facValue(
      int-val(
        state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
              m([any-loc(0),int(any-int(0))]
                [any-loc(1),int(any-int(1))]
                [any-loc(2),int(any-int(2))]),
              out(noOutput))
        |('C < 'N)
        ['C])))
&
  (int-val(
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
             [any-loc(1),int(any-int(1))]
             [any-loc(2),int(any-int(2))]),
           out(noOutput))
     |('C < 'N)
     ['N])
  = N:Int))
&
  (evalTst(
```

```
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C < 'N),
    'C < 'N)
  = false))
&
  (int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C < 'N)
    ['C])
  <=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
```

```
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      |('C < 'N)
      ['N])
    = true))
&
  (0
   <=
   int-val
     state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
           m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
           out(noOutput))
      |('C < 'N)
      ['C])
    = true)))
```

A Factorial Example (IX)

This fourth goal splits into the following subgoals, using `(auto .)` once, and then `(cnj .)` four times:

```
==================================
label-sel: facx@4.1.1.1.1.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)['X])
=
  facValue(
    int-val(
      state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
```

```
            m([any-loc(0),int(any-int(0))]
              [any-loc(1),int(any-int(1))]
              [any-loc(2),int(any-int(2))]),
            out(noOutput))
      |('C < 'N)
      ['C]))

==================================
label: facx@4.1.1.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)
  ['N])
=
  N*Int
```

```
==================================
label: facx@4.1.1.2.0
==================================
evalTst(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N),
  'C < 'N)
=
  false

==================================
label: facx@4.1.2.0
==================================
int-val(
  state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
        m([any-loc(0),int(any-int(0))]
          [any-loc(1),int(any-int(1))]
```

```
          [any-loc(2),int(any-int(2))]),
        out(noOutput))
  |('C < 'N)
  ['C])
<=
  int-val(
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C < 'N)
    ['N])
= true

================================
label: facx@4.2.0
================================
0
<=
  int-val(
```

```
    state(e(['C,any-loc(0)]['N,any-loc(2)]['X,any-loc(1)]),n(0),
          m([any-loc(0),int(any-int(0))]
            [any-loc(1),int(any-int(1))]
            [any-loc(2),int(any-int(2))]),
          out(noOutput))
    |('C < 'N)
    ['C])
= true
++++++++++++++++++++++++++++++++
```

Each of these can be discharged with the `(auto .)` command, finishing the proof of goal 4. The ITP then answers with

```
q.e.d

++++++++++++++++++++++++++++++++
```

showing that the proof is complete.