Verification of Recursive Programs

Andreas Podelski

February 8, 2012

$$m(x) = \begin{cases} x - 10 & \text{if } x > 100 \\ m(m(x+11)) & \text{if } x \leq 100 \end{cases}$$

procedure m(x) returns (res)

ℓ_0: if x>100

ℓ_1: res:=x-10

else

ℓ_2: $\mathtt{x}_m$:= x+11

ℓ_3: $\mathtt{res}_m$:= call m($\mathtt{x}_m$)

ℓ_4: $\mathtt{x}_m$:= $\mathtt{res}_m$

ℓ_5: $\mathtt{res}_m$:=call m($\mathtt{x}_m$)

ℓ_6: res := $\mathtt{res}_m$

ℓ_7: assert (x<=101 -> res=91)

return m

restrictions and conventions

- no pointers, no global variables and only call-by-value procedure calls
- forbidden to write to the input variables of a procedure
- procedure p has one input variable x ("formal argument") and one output variable res
- procedure calls appear in the following form:

$$res_p \texttt{ := call p(}x_p\texttt{)}$$

- actual argument (actual parameter) is always x_p (special variable)
- on return, value of output variable res is stored into variable res_p of the caller

procedure m(x) returns (res)

ℓ_0: if x>100

ℓ_1: res:=x-10

else

ℓ_2: $\mathtt{x}_m$:= x+11

ℓ_3: $\mathtt{res}_m$:= call m($\mathtt{x}_m$)

ℓ_4: $\mathtt{x}_m$:= $\mathtt{res}_m$

ℓ_5: $\mathtt{res}_m$:=call m($\mathtt{x}_m$)

ℓ_6: res := $\mathtt{res}_m$

ℓ_7: assert (x<=101 -> res=91)

return m

procedure m(x) returns (res)

ℓ_0: if x>100

ℓ_1: res:=x-10

else

ℓ_2: $\mathtt{x}_m$:= x+11

ℓ_3: call m

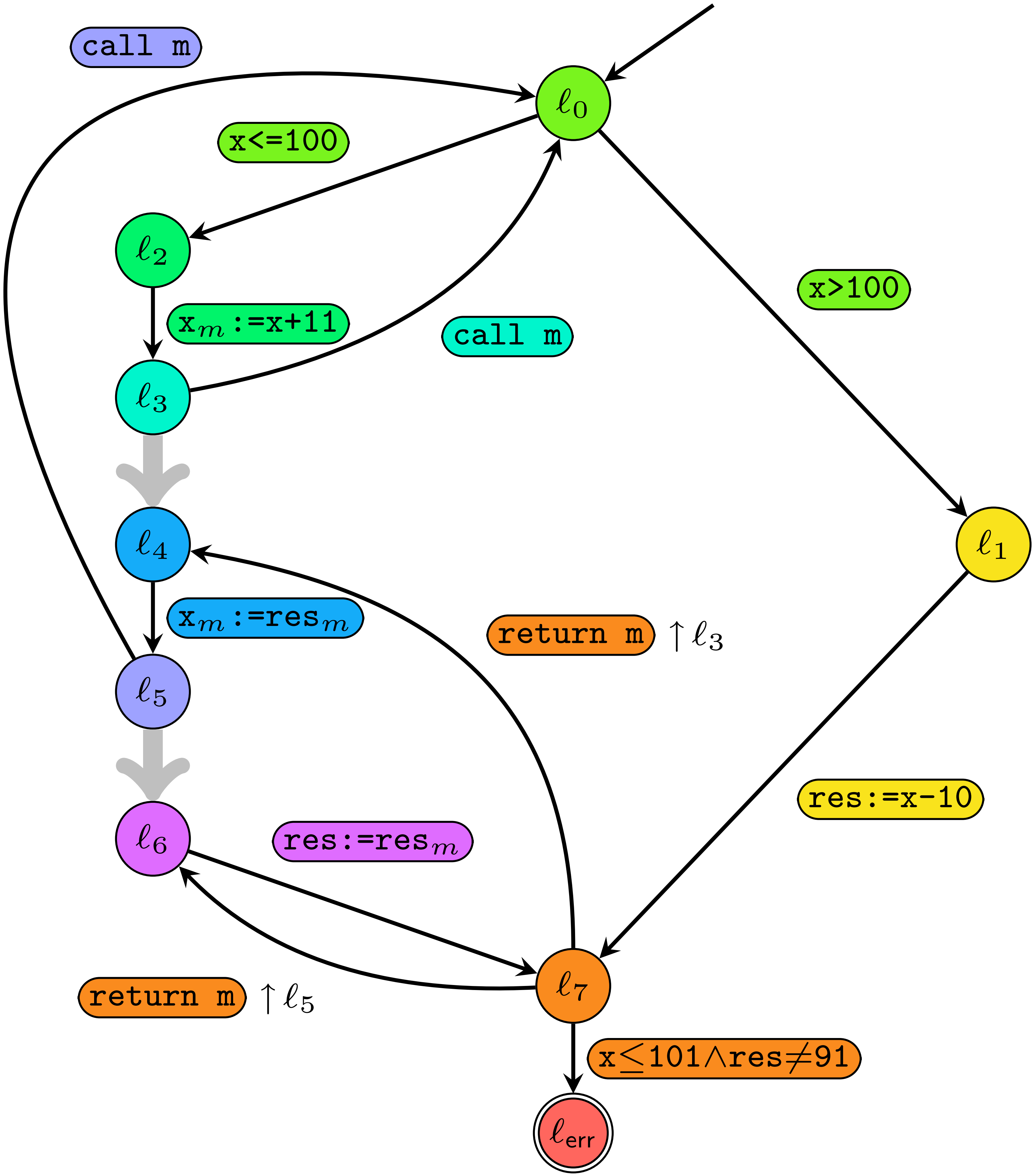
ℓ_4: $\mathtt{x}_m$:= $\mathtt{res}_m$

ℓ_5: call m

ℓ_6: res := $\mathtt{res}_m$

ℓ_7: assert (x<=101 -> res=91)

return m



call m
ℓ_0
x<=100
ℓ_2
x>100
x_m:=x+11
call m
ℓ_3
ℓ_4
ℓ_1
x_m:=res_m
return m $\uparrow \ell_3$
ℓ_5
res:=x-10
ℓ_6
res:=res_m
ℓ_7
return m $\uparrow \ell_5$
x≤101∧res≠91
ℓ_{err}

four kinds of statements

- call statement `call p`
  creates a new calling context where (unique) input variable x gets assigned value of variable x_p from previous calling context
  new calling context is added on top of stack
- return statement `return p`
  result value res is taken from the current calling context
  then current calling context is removed from stack ("pop")
  result value res is written to variable res_p (in context of caller)
- assignment statement `y:=t`
  value of term t is assigned to variable y in top-most calling context
- assume statement ϕ
  only executed if variable valuation of top-most calling context satisfies Boolean expression ϕ

To model that procedure p is called from procedure q at program location ℓ_j^q (in q's control flow graph),

- an edge labeled `call p` goes from ℓ_j^q to ℓ_0^p, the entry location of p,
- an edge labeled `return p` $\uparrow \ell_j^q$ goes from ℓ_n^p, the exit location of p, to the location ℓ_{j+1}^q in q's control flow graph, where ℓ_{j+1}^q is the successor location of ℓ_j^q with respect to the call of procedure p.

semantics

- *valuation* ν = function that maps program variables to values
- (local) state of a procedure = pair (ℓ, ν) of program location and valuation
- (global) state S of the program = stack of local states

$$S = (\ell_0, \nu_0).(\ell_1, \nu_1) \dots (\ell_n, \nu_n)$$

- stack element = called (and not yet returned) procedure
- topmost/rightmost element represents current calling context

state	label of edge (ℓ, ℓ')	successor state	side condition
$S.(\ell, \nu)$	`y:=t`	$S.(\ell', \nu')$	$\nu' = \nu \oplus \{y \mapsto \nu(t)\}$
$S.(\ell, \nu)$	ϕ	$S.(\ell', \nu)$	$\nu \models \phi$
$S.(\ell, \nu)$	`call p`	$S.(\ell, \nu).(\ell', \nu')$	$\nu'(x) = \nu(x_p)$
$S.(\ell_<, \nu_<).(\ell, \nu)$	`return p` $\uparrow \ell_<$	$S.(\ell', \nu')$	$\nu' = \nu_< \oplus \{res_p \mapsto \nu(res)\}$

transition from state S to successor state S' under statement st

$$S \xrightarrow{st} S'$$

transition under sequence of statements $\pi = st_0 \dots st_{n-1}$

$$S \xrightarrow{\pi} S'$$

Hoare rule for recursion

$$\frac{\{\phi(x, res)\}\ \mathsf{body}_p\ \{\theta(x, res)\}}{\{\phi(x_p, res_p)\}\ res_p := p(x_p)\ \{\exists res_p.\phi(x_p, res_p) \wedge \theta(x_p, res_p)\}}$$

procedure m(x) returns (res)
if x>100
  res:=x-10
else
  $\mathtt{x}_m$:= x+11
  $\mathtt{res}_m$:=call m($\mathtt{x}_m$)
  $\mathtt{x}_m$:= $\mathtt{res}_m$
  $\mathtt{res}_m$:=call m($\mathtt{x}_m$)
  res := $\mathtt{res}_m$

```
procedure m(x) returns (res)
```
$\{true\}$
```
if x>100
```
  $\{x \geq 101\}$
```
  res:=x-10
else
```
  $\{x \leq 100\}$

  $\mathtt{x}_m$ `:= x+11`

  $\{x_m \leq 111\}$

  $\mathtt{res}_m$ `:=call m(`$\mathtt{x}_m$`)`

  $\{\mathit{res}_m \leq 101\}$

  $\mathtt{x}_m$ `:=` $\mathtt{res}_m$

  $\{x_m \leq 101\}$

  $\mathtt{res}_m$ `:=call m(`$\mathtt{x}_m$`)`

  $\{\mathit{res}_m = 91\}$

  `res :=` $\mathtt{res}_m$

$\{\mathit{res} = 91 \;\vee\; (x \geq 101 \wedge \mathit{res} = x - 10)\}$

Hoare rule for recursion

$$\frac{\{\phi(x, res)\}\ \mathsf{body}_p\ \{\theta(x, res)\}}{\{\phi(x_p, res_p)\}\ res_p := p(x_p)\ \{\exists res_p.\phi(x_p, res_p) \wedge \theta(x_p, res_p)\}}$$

```
        procedure m(x) returns (res)
        {⊤}
ℓ0:     if x>100
          {x ≥ 101}
ℓ1:       res:=x-10
        else
          {x ≤ 100}
ℓ2:       x_m := x+11
          {x_m ≤ 111}
ℓ3:       call m
          {res_m ≤ 101}
ℓ4:       x_m := res_m
          {x_m ≤ 101}
ℓ5:       call m
          {res_m = 91}
ℓ6:       res := res_m
        {res = 91 ∨ (x ≥ 101 ∧ res = x − 10)}
ℓ7:     assert (x<=101 -> res=91)
        return m
```