Automated Verification of Recursive Programs

Andreas Podelski

February 9, 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

`procedure m(x) returns (res)`

ℓ_0: `if x>100`

ℓ_1: `  res:=x-10`

`else`

ℓ_2: `  x`$_m$ `:= x+11`

ℓ_3: `  call m`

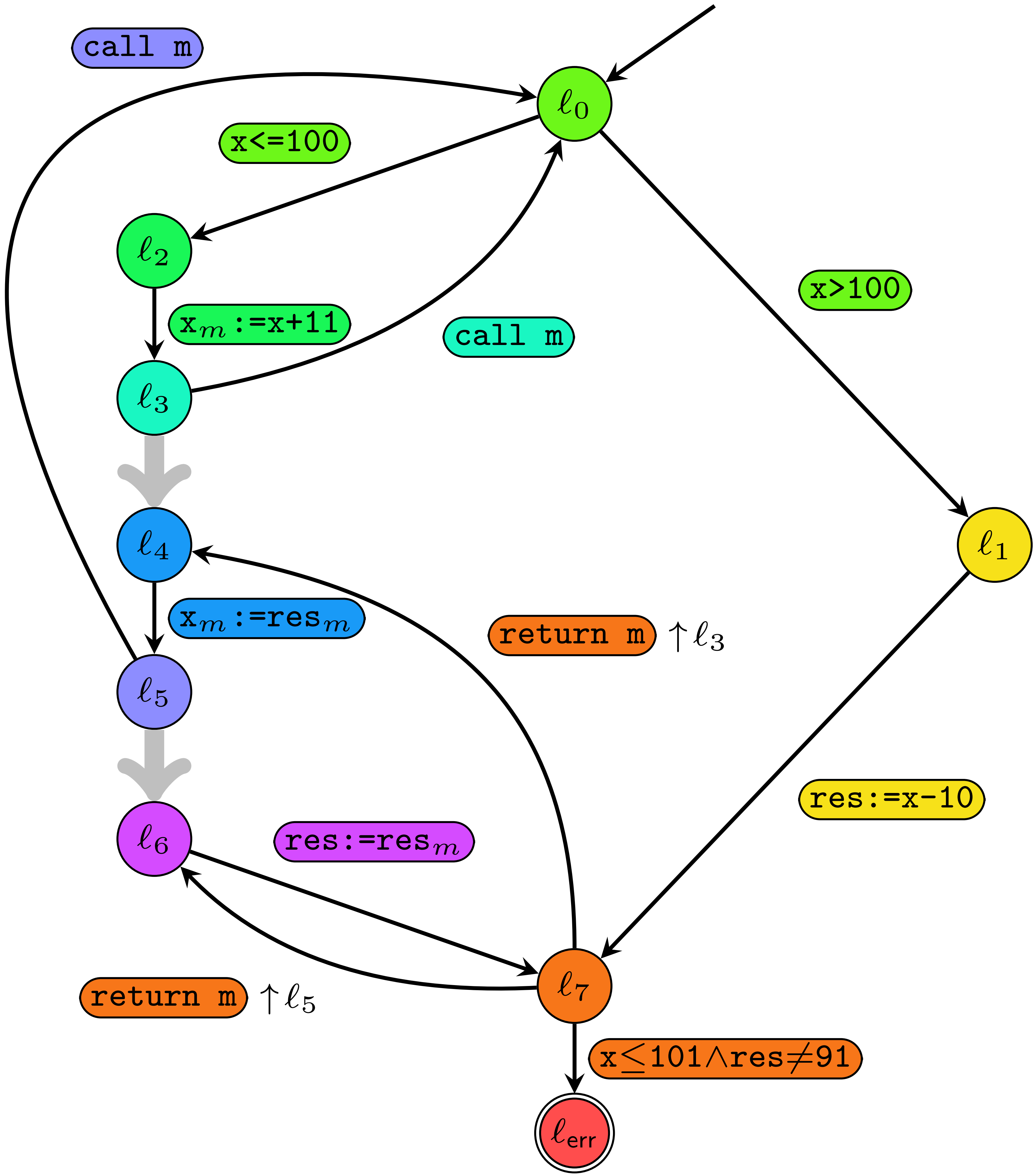
ℓ_4: `  x`$_m$ `:= res`$_m$

ℓ_5: `  call m`

ℓ_6: `  res := 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}

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

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

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

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

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

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

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

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` $\uparrow \ell_<$	$S.(\ell', \nu')$	$\nu' = \nu_< \oplus \{res_p \mapsto \nu(res)\}$

Hoare rule for recursion

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

```
procedure m(x) returns (res)
if x>100
  res:=x-10
else
  x_m := x+11
  res_m :=call m(x_m)
  x_m := res_m
  res_m :=call m(x_m)
  res := res_m
```

```
procedure m(x) returns (res)
```
$\{true\}$
```
if x>100
```
$\{x \geq 101\}$
```
  res:=x-10
else
```
$\{x \leq 100\}$
```
  x_m := x+11
```
$\{x_m \leq 111\}$
```
  res_m :=call m(x_m)
```
$\{res_m \leq 101\}$
```
  x_m := res_m
```
$\{x_m \leq 101\}$
```
  res_m :=call m(x_m)
```
$\{res_m = 91\}$
```
  res := res_m
```
$\{res = 91 \lor (x \geq 101 \land res = x - 10)\}$

Question:

How does one automate
the verification of recursive programs?

procedure m(x) returns (res)

ℓ_0: if x>100

ℓ_1: res:=x-10

else

ℓ_2: x_m := x+11

ℓ_3: res_m := call m(x_m)

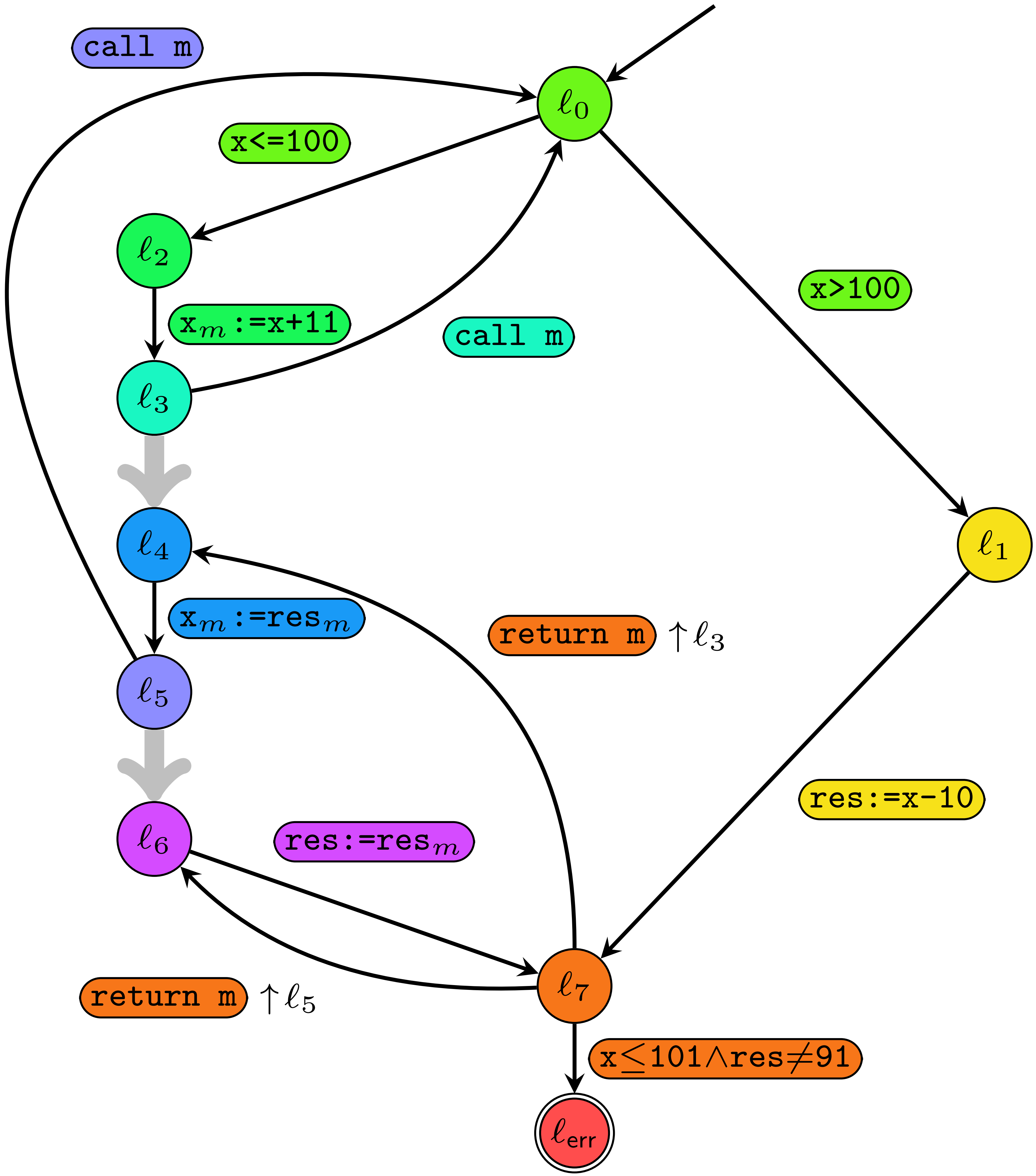
ℓ_4: x_m := res_m

ℓ_5: res_m :=call m(x_m)

ℓ_6: res := 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}

trace π = sequence of statements, $\pi = st_0 \ldots st_{n-1}$

error trace = trace π that labels a path in the recursive control flow graph from the initial location to the error location

feasible trace = trace π which has an execution, i.e.,

$$(\ell_0, \nu_0) \xrightarrow{\pi} S.(\ell_{\mathsf{err}}, \nu_n)$$

for some initial valuation ν_0, stack of local states S, and final valuation ν_n

program $\mathcal{P}$ is *correct* if it has no feasible error trace

from traces to nested traces

- trace = sequence of statements
- feasibility of trace = existence of sequence of global states, i.e., stacks of local states
- stack needed to record correspondence between call position *i* and return position *j*
- nested trace = sequence of statements + edges between positions of in the sequence
    - edge expresses correspondence between call and return positions
- feasibility of nested trace = existence of sequence of local states

nested word = pair $(w, \rightsquigarrow)$ of

- word $w = a_0 \ldots a_{n-1}$ and *nesting relation* $\rightsquigarrow$
- $i \rightsquigarrow j$ expresses correspondence between position i of a call and position j of a matching return
- formally:

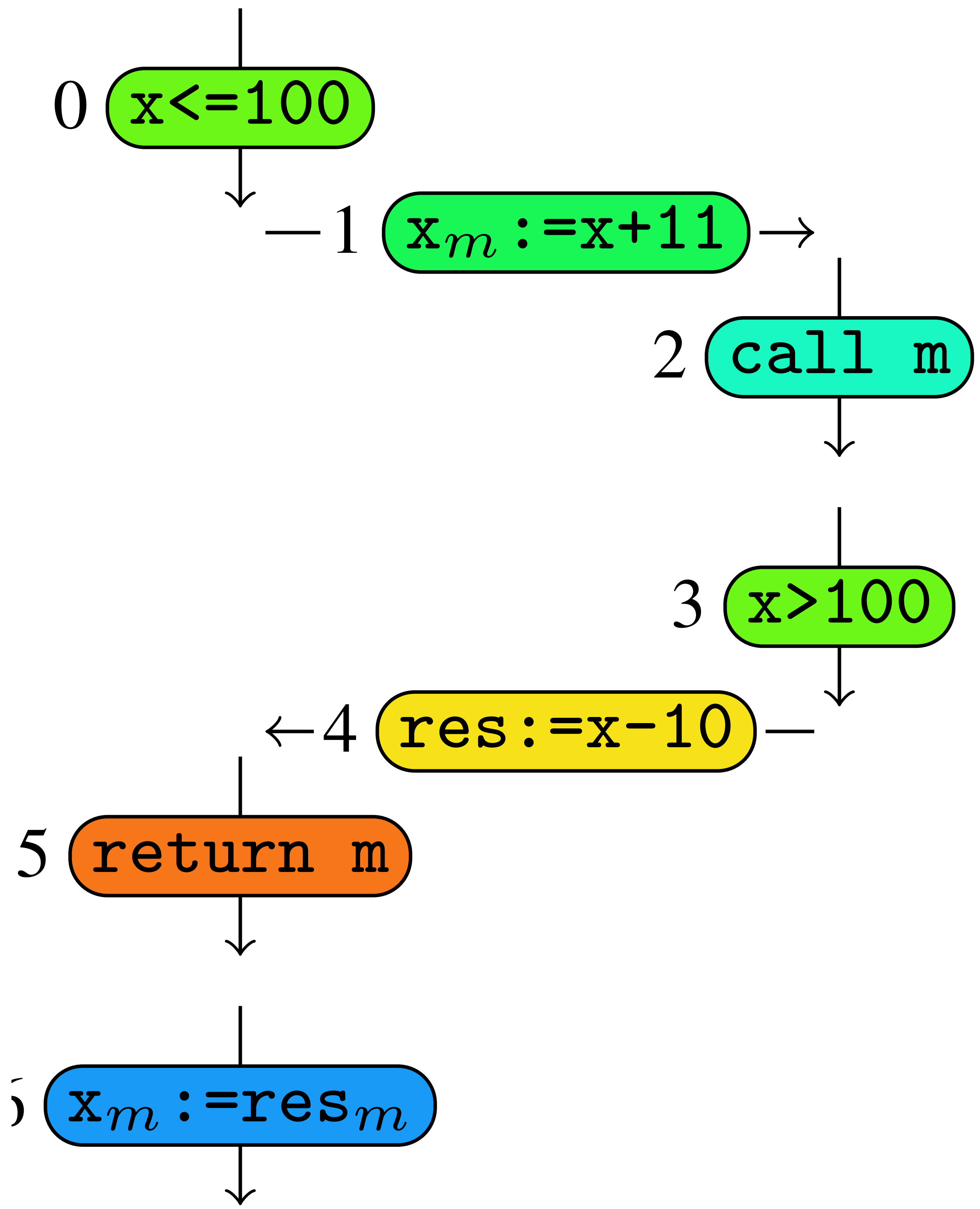
$$\rightsquigarrow \subseteq \{0, \ldots, n-1\} \times \{0, \ldots, n-1, \infty\}$$

- index ∞ = return position for all unfinished call
- relation $\rightsquigarrow$ is left-unique, right-unique, and properly nested

$$i_1 \rightsquigarrow j,\ i_2 \rightsquigarrow j,\ j \neq \infty \ \text{ implies } \ i_1 = i_2$$

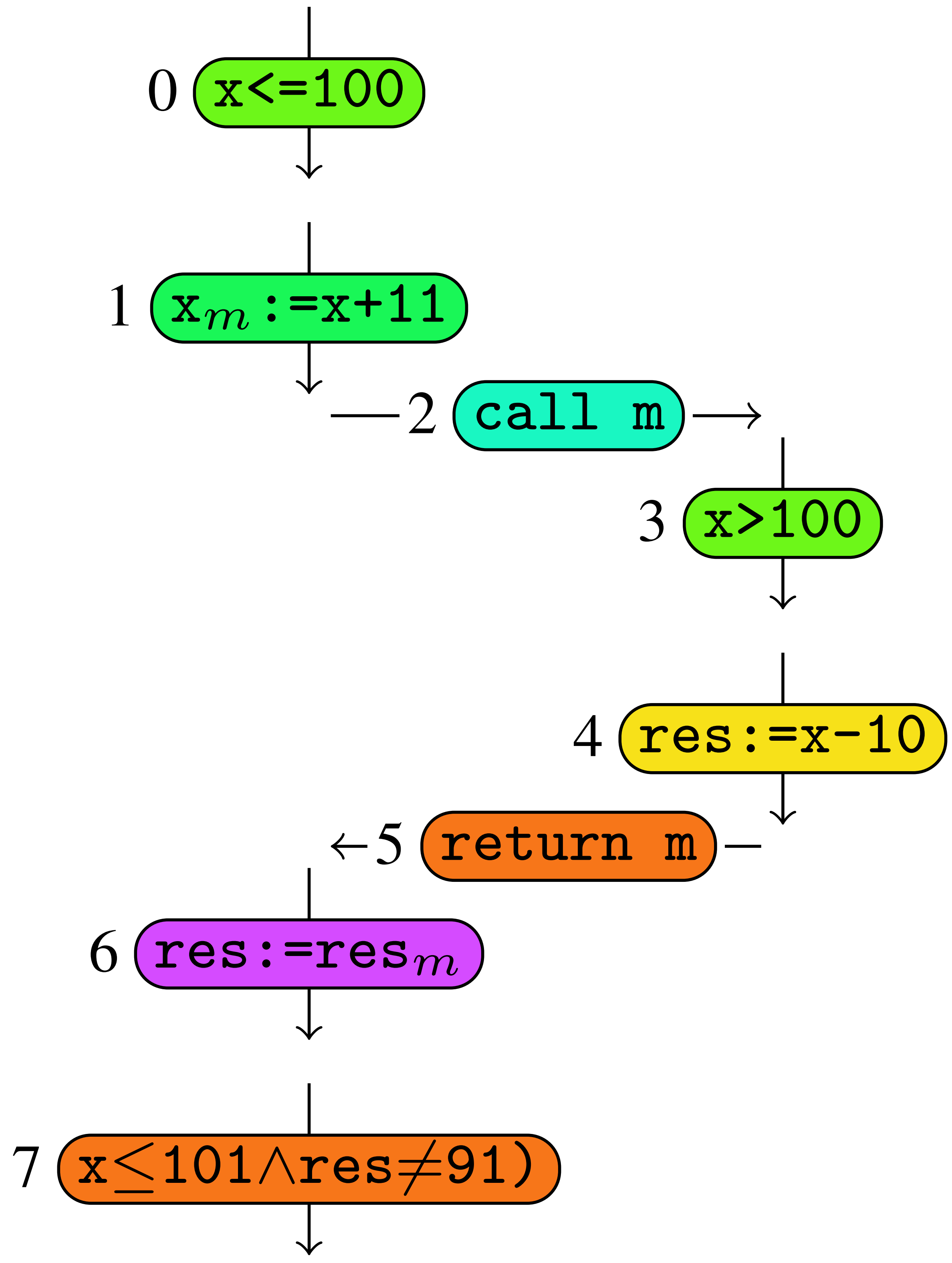
$$i \rightsquigarrow j_1,\ i \rightsquigarrow j_2 \ \text{ implies } \ j_1 = j_2$$

$$i_1 \rightsquigarrow j_1,\ i_2 \rightsquigarrow j_2,\ i_1 \leq i_2 \ \text{ implies } \ \begin{cases} i_1 < j_1 < i_2 < j_2 \\ \text{or} \\ i_1 \leq i_2 < j_2 \leq j_1 \end{cases}$$



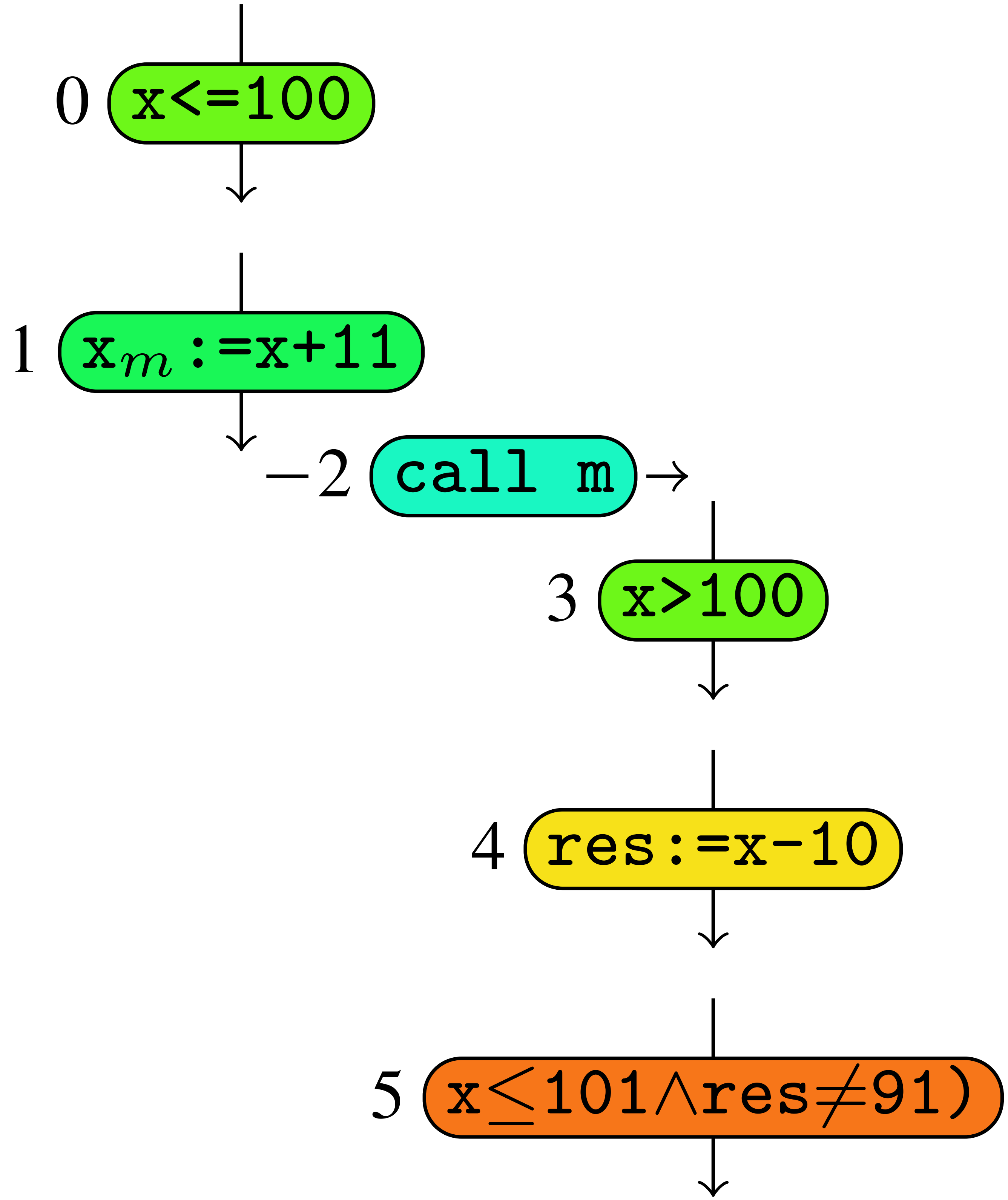
nested trace π_1

$1 \rightsquigarrow_1 4$



nested trace π_2

$2 \rightsquigarrow_2 5$



nested trace π_3

$2 \rightsquigarrow_3 \infty$

nested word automaton

$$\mathcal{A} = (Q, \langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle, Q^{\mathsf{init}}, Q^{\mathsf{fin}})$$

- set of states Q,
- triple $\langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle$ of transition relations for internal, call, and return positions

$$\begin{aligned} \delta_{\mathsf{in}} &\subseteq \phantom{Q \times{}} Q \times \Sigma \times Q \\ \delta_{\mathsf{ca}} &\subseteq \phantom{Q \times{}} Q \times \Sigma \times Q \\ \delta_{\mathsf{re}} &\subseteq Q \times Q \times \Sigma \times Q \end{aligned}$$

- set of initial states $Q^{\mathsf{init}} \subseteq Q$,
- set of final state $Q^{\mathsf{fin}} \subseteq Q$.

run of a nested word automaton $\mathcal{A}$ over the nested word $(a_0 \ldots a_{n-1}, \rightsquigarrow)$ = sequence $q_0, \ldots, q_n$ of states that
starts in an initial state, i.e., $q_0 \in Q^{\mathsf{init}}$, and is consecutive, i.e., for each $i = 0, \ldots, n-1$,

$$\begin{aligned} (q_i, a_i, q_{i+1}) &\in \delta_{\mathsf{in}} && \text{if } i \text{ is an internal position,} \\ (q_i, a_i, q_{i+1}) &\in \delta_{\mathsf{ca}} && \text{if } i \text{ is a call position,} \\ (q_i, q_k, a_i, q_{i+1}) &\in \delta_{\mathsf{re}} && \text{if } i \text{ is a return position and } k \rightsquigarrow i \end{aligned}$$

run *accepting* if it ends in a final state, i.e., $q_n \in Q^{\mathsf{fin}}$

regular languages of nested words

- nested word automaton $\mathcal{A}$ *accepts* nested word $(w, \rightsquigarrow)$ if it has an accepting run over $(w, \rightsquigarrow)$
- language of nested words recognized by $\mathcal{A}$ = set $\mathcal{L}(\mathcal{A})$ consisting of the nested words accepted by $\mathcal{A}$
- language of nested words is *regular* if it is recognized by a *finite* nested word automaton
- standard properties of finite automata
  - closure under intersection and complement
  - decidability of emptiness

program correctness defined via nested word automata

- alphabet = set of statements
  - nested trace = nested word over statements
- two nested word automata:
  - control automaton: nested error traces
  - data automaton: feasible nested traces
- proof rule based on nested word automata
- automate proof rule by constructing finite abstraction of data automaton

control automaton $\mathcal{A}_{\mathcal{P}}$

$$\mathcal{A}_{\mathcal{P}} = (Q, \langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle, Q^{\mathsf{init}}, Q^{\mathsf{fin}})$$

- *set of states* Q = *set of program locations*
- *three transition relations* δ_{in}, δ_{ca}, δ_{re} *according to edges in recursive control flow graph; i.e., if the edge* (ℓ, ℓ') *is labeled with:*
  - *assignment or assume statement* st, *then*

    $$(\ell, st, \ell') \in \delta_{\mathsf{in}}$$

  - *call statement* `call p`, *then*

    $$(\ell, \texttt{call p}, \ell') \in \delta_{\mathsf{ca}}$$

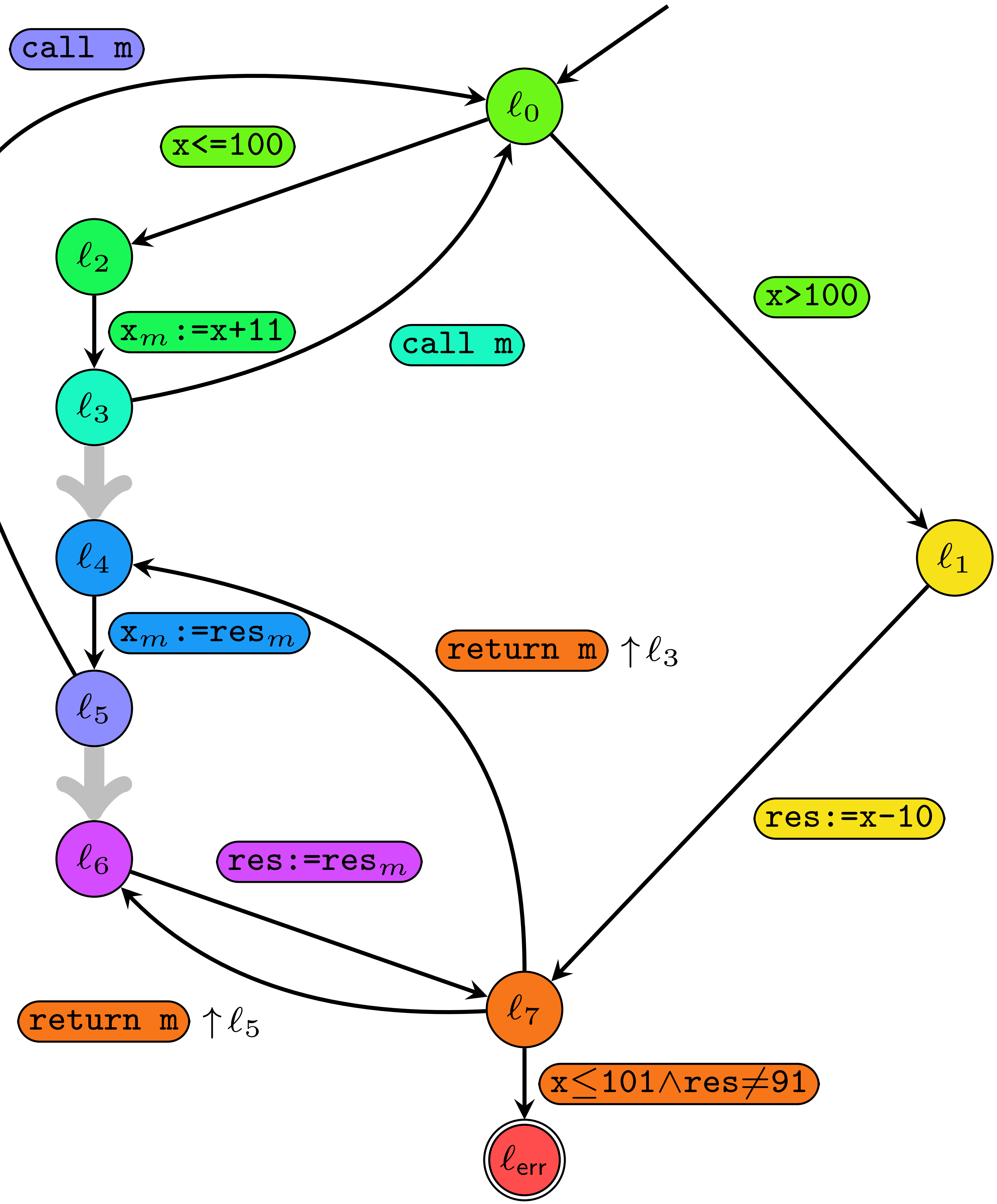
  - *return statement (with call location* $\ell_<$*)* `return p` $\uparrow \ell_<$, *then*

    $$(\ell, \ell_<, \texttt{return p}, \ell') \in \delta_{\mathsf{re}}$$

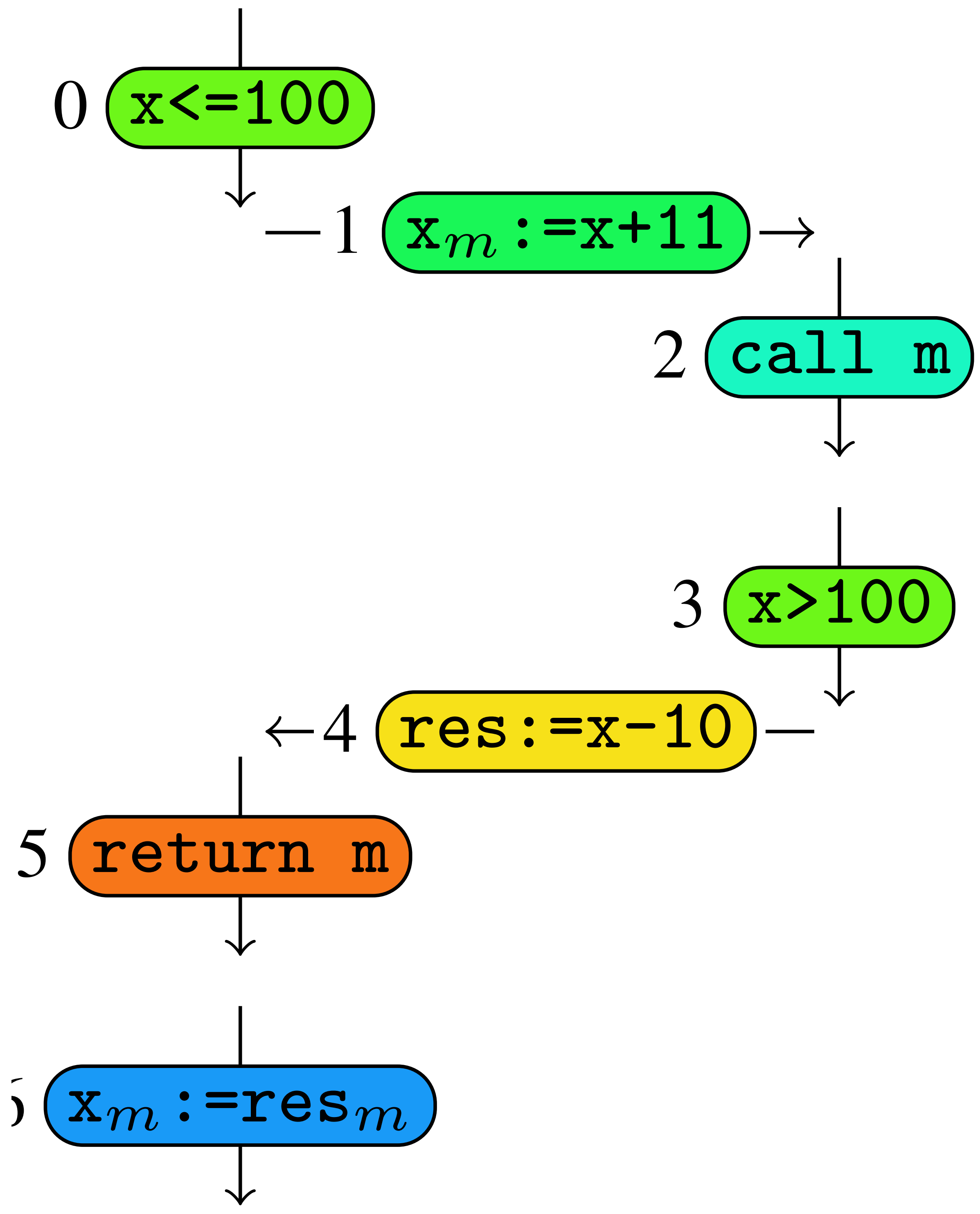
- *initial/final states*

  $$Q^{\mathsf{init}} = \{\ell_0^{main}\}, \qquad Q^{\mathsf{fin}} = \{\ell_{\mathsf{err}}\}$$

nested error trace = *nested word accepted by* $\mathcal{A}_{\mathcal{P}}$

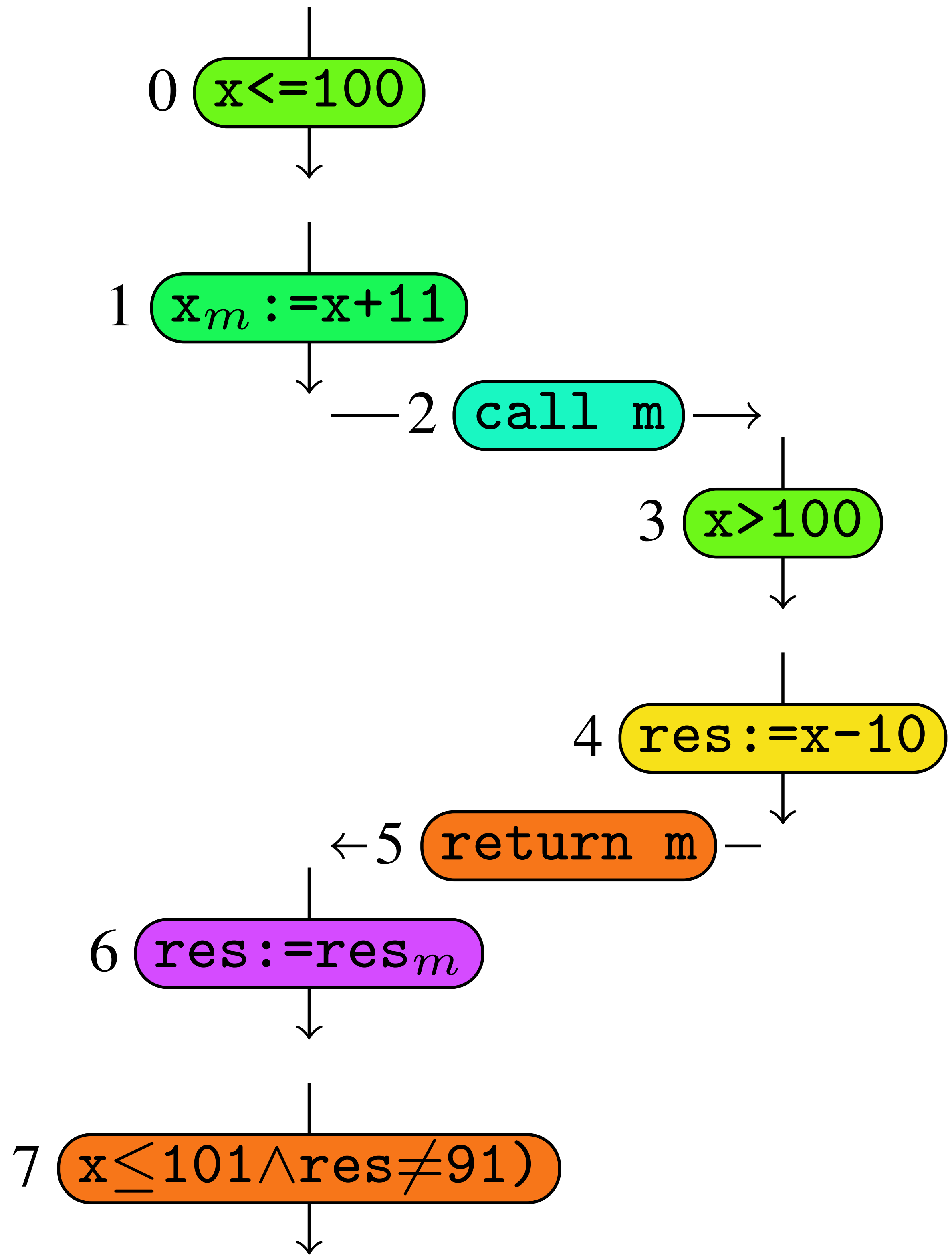


call m
ℓ_0
x<=100
ℓ_2
x>100
x_m:=x+11
call m
ℓ_3
ℓ_4
ℓ_1
x_m:=res_m
return m $\uparrow \ell_3$
ℓ_5
res:=x-10
ℓ_6
res:=res_m
ℓ_7
return m $\uparrow \ell_5$
x≤101∧res≠91
ℓ_{err}

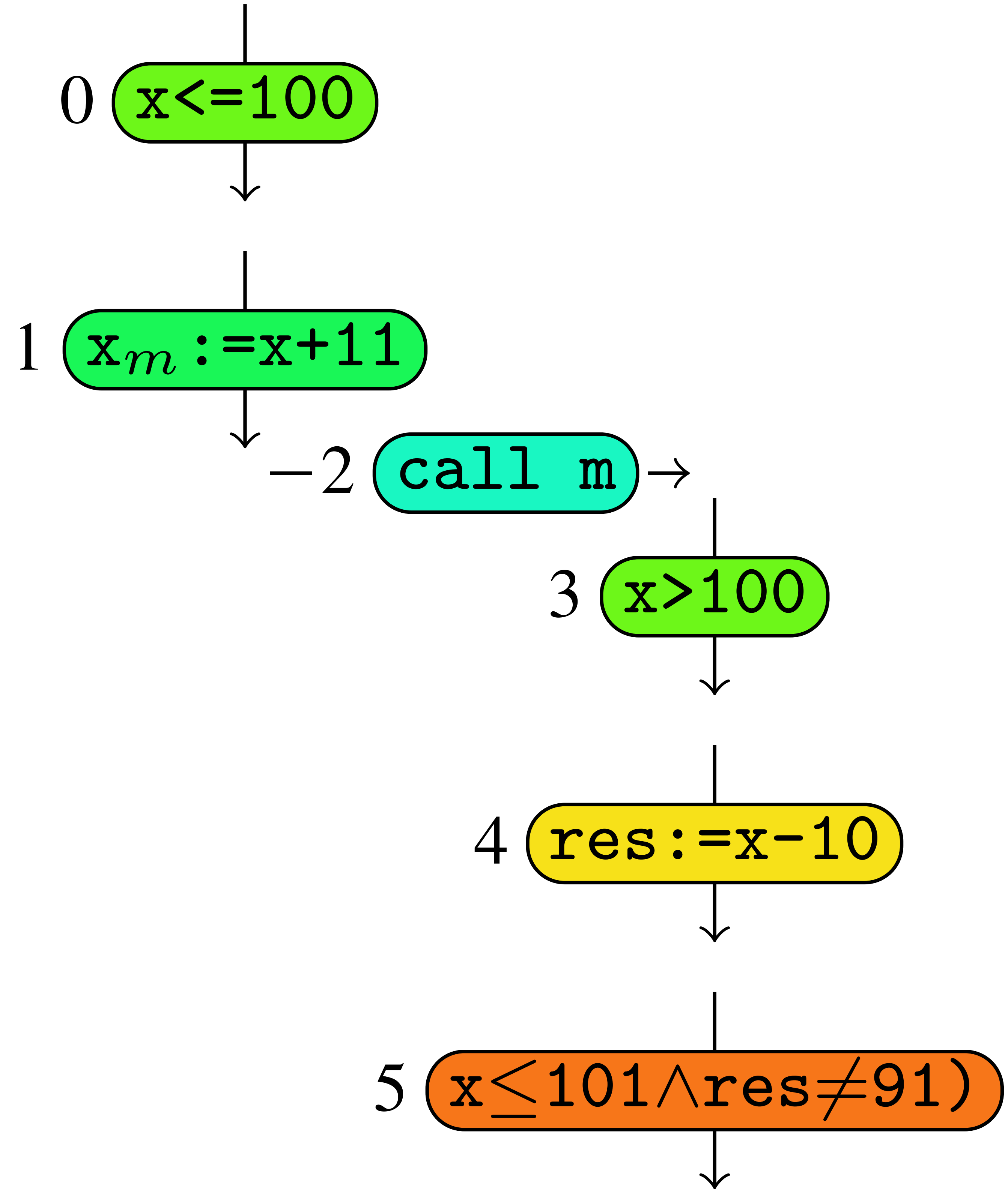


nested trace π_1

$1 \rightsquigarrow_1 4$



nested trace π_2
$2 \rightsquigarrow_2 5$



nested trace π_3

$2 \rightsquigarrow_3 \infty$

data automaton *for set of statements* Σ

$$\mathcal{A}_\Sigma = (Q, \langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle, Q^{\mathsf{init}}, Q^{\mathsf{fin}})$$

- Q = *(in general infinite) set of valuations* ν
- *three transition relations* δ_{in}, δ_{ca}, δ_{re} *are the transition relations induced by the statements in* Σ*; i.e.,*
  if the statement is:
  - *assignment statement* `y:=t`, *then*

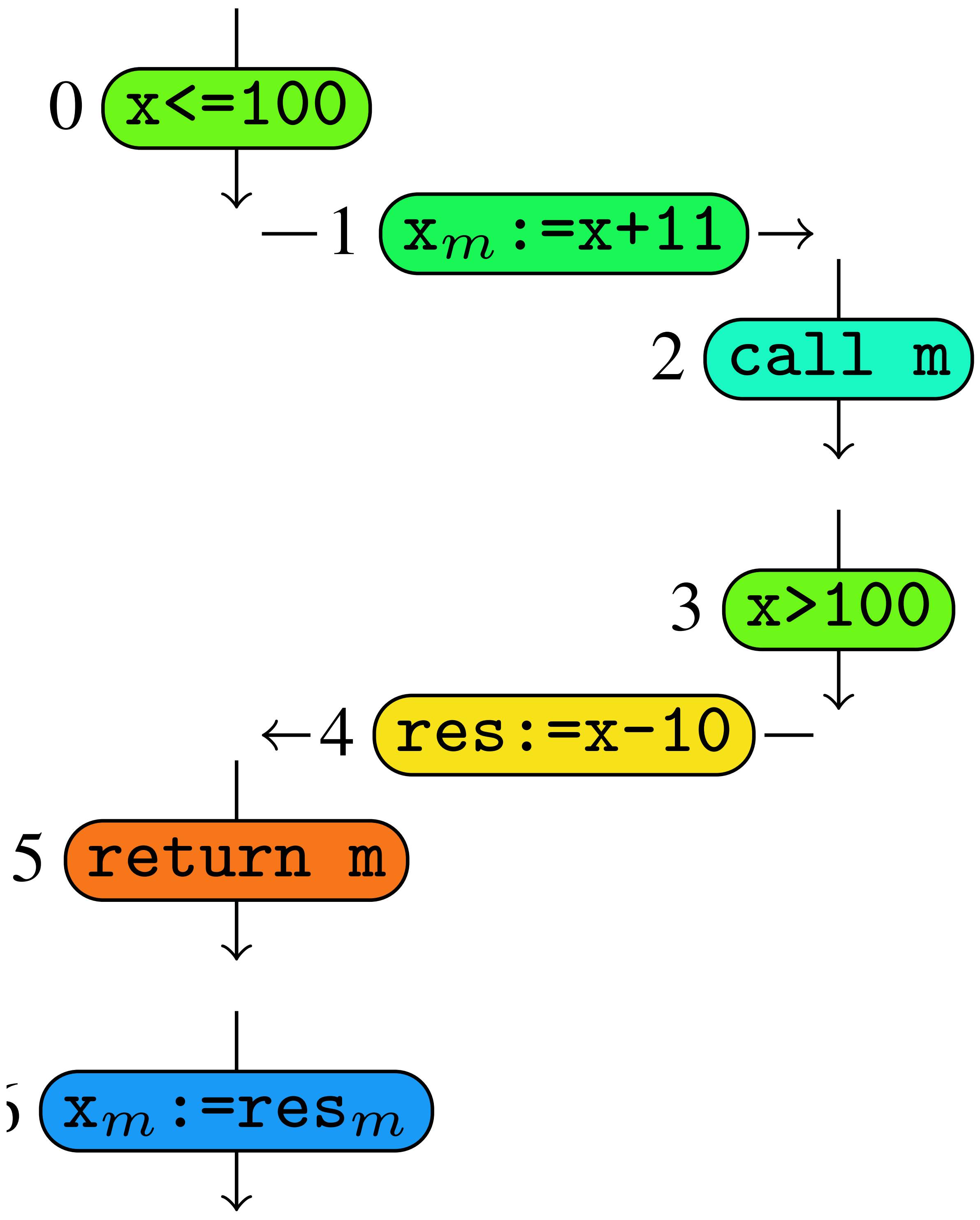
    $$(\nu, \texttt{y:=t}, \nu \oplus \{y \mapsto \nu(t)\}) \in \delta_{\mathsf{in}}$$
  - *assume statement* ϕ, *then*

    $$(\nu, \phi, \nu) \in \delta_{\mathsf{in}} \;\; \textit{if} \;\; \nu \models \phi$$
  - *call statement* `call p`, *then*

    $$(\nu, \texttt{call p}, \nu') \in \delta_{\mathsf{ca}} \;\; \textit{if} \;\; \nu'(x) = \nu(x_p)$$
  - *return statement* `return p`, *then*

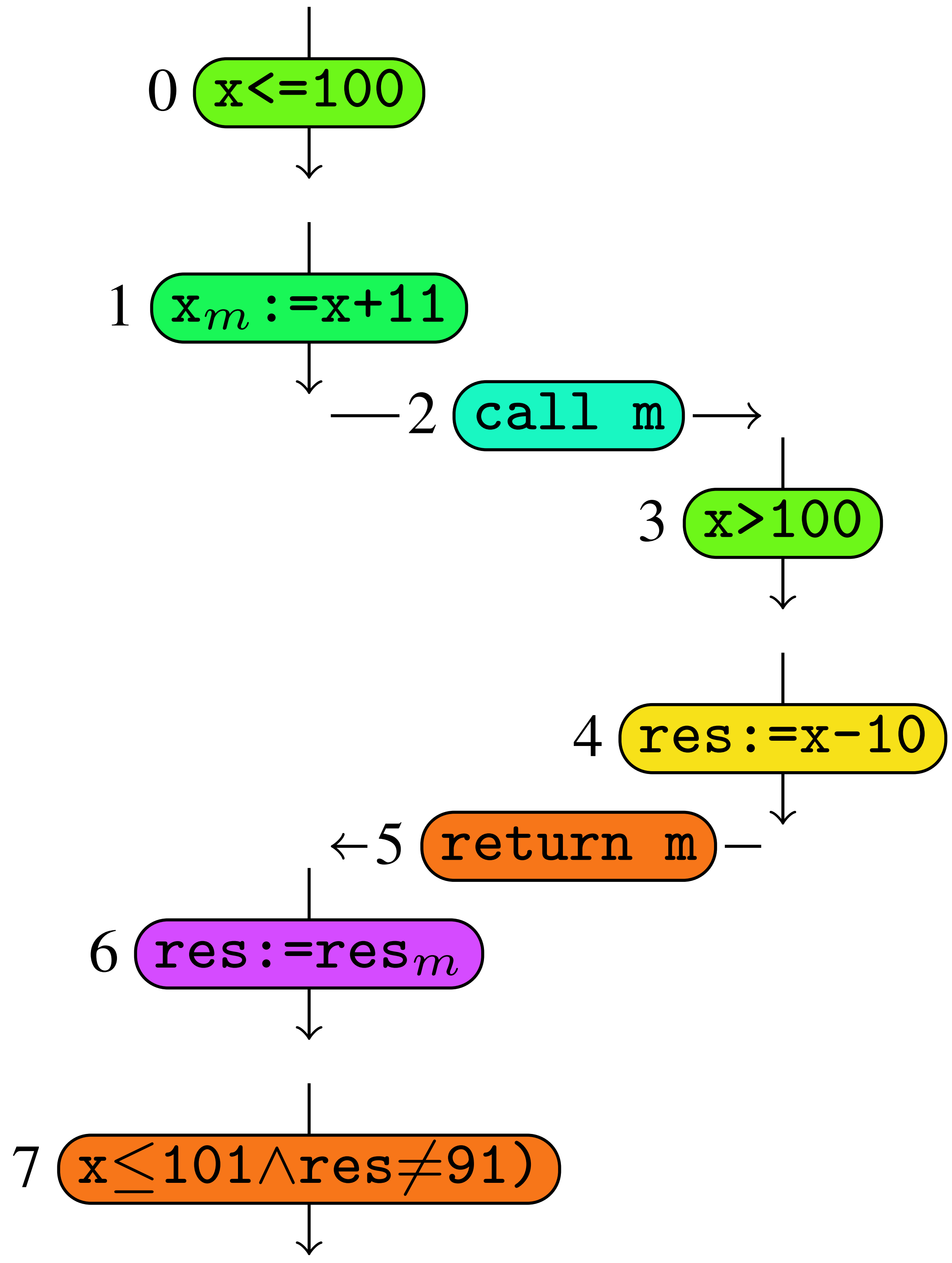
    $$(\nu, \nu_<, \texttt{return p}, \nu_< \oplus \{res_p \mapsto \nu(res)\}) \in \delta_{\mathsf{re}}\ .$$
- *every state is an initial state (i.e.* $Q^{\mathsf{init}} = Q$*),*
- *every state is a final state (i.e.* $Q^{\mathsf{fin}} = Q$*).*

feasible nested trace = *nested word accepted* $\mathcal{A}_\Sigma$



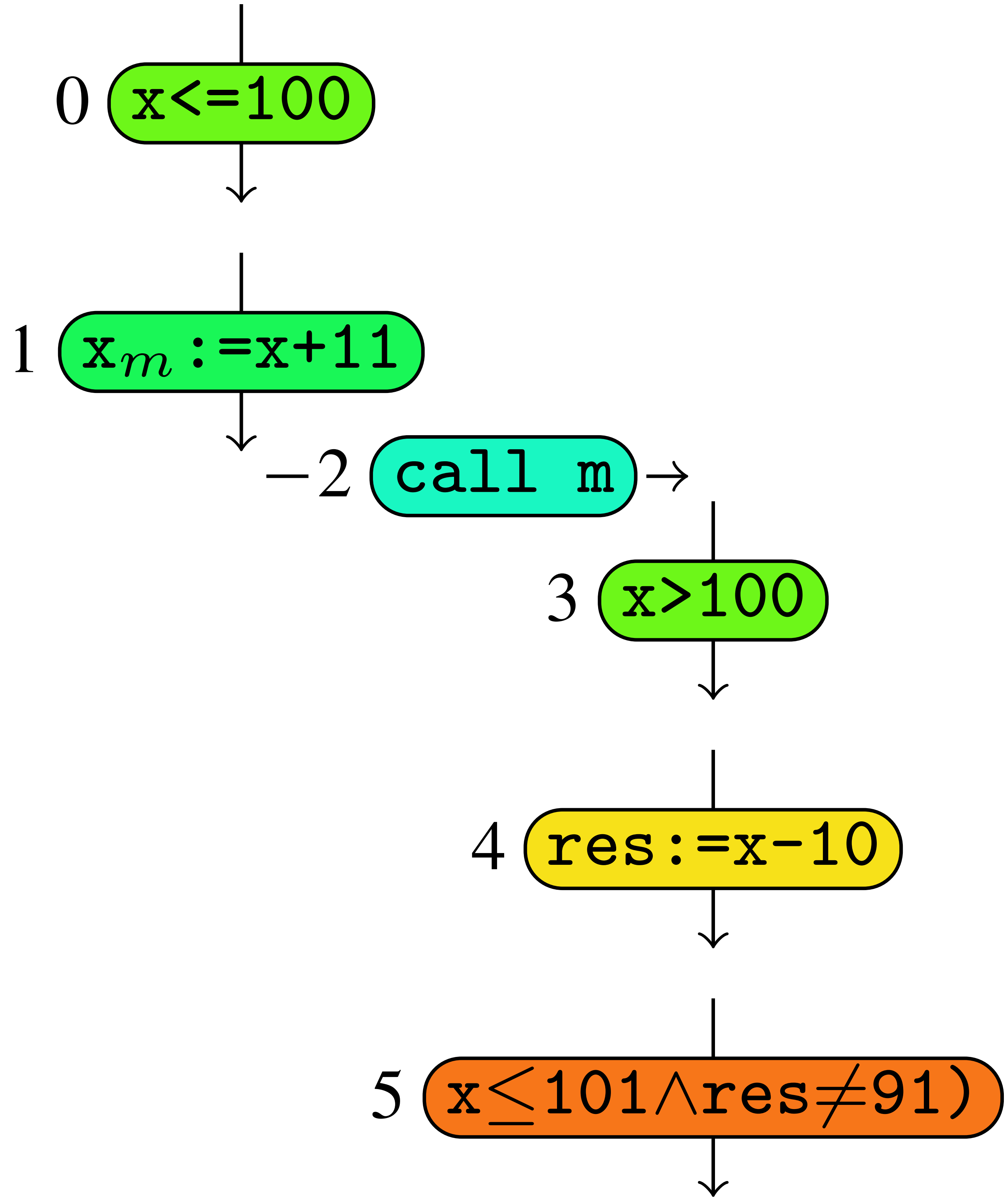
nested trace π_1

$1 \rightsquigarrow_1 4$



nested trace π_2

$2 \rightsquigarrow_2 5$



nested trace π_3

$2 \rightsquigarrow_3 \infty$

characterizing program correctness by NWA's

- program is correct iff intersection between control automaton and data automaton is empty

- equivalent: $$\mathcal{L}(\mathcal{A}_{\mathcal{P}}) \cap \mathcal{L}(\mathcal{A}_{\Sigma}) = \emptyset.$$

    - The sequence of locations $\ell_0, \ldots, \ell_n$ is a run of $\mathcal{A}_{\mathcal{P}}$ for $(st_0 \ldots st_{n-1}, \rightsquigarrow)$ and the sequence of valuations $\nu_0, \ldots, \nu_n$ is a run of $\mathcal{A}_{\Sigma}$ for $(st_0 \ldots st_{n-1}, \rightsquigarrow)$.

    - The nested trace $(st_0 \ldots st_{n-1}, \rightsquigarrow)$ is well-nested and there is a sequence of stacks
    $$(\ell_0, \nu_0) \xrightarrow{st_0} S_1.(\ell_1, \nu_1) \xrightarrow{st_1} \cdots \xrightarrow{st_{n-1}} S_n.(\ell_n, \nu_n)$$
    according to the transition relation of $\mathcal{P}$. In that case $S_i = (\ell_{k_0}, \nu_{k_0}) \ldots (\ell_{k_m}, \nu_{k_m})$ such that the nesting relation contains $k_0 \rightsquigarrow k'_0, \ldots, k_m \rightsquigarrow k'_m$ and $k_0 < \cdots < k_m < i \leq k'_m \leq \cdots \leq k'_0$ holds. $\square$

characterizing program correctness by nwa's

- program $\mathcal{P}$ is correct if and only if the intersection between the control automaton and the data automaton is empty

$$\mathcal{L}(\mathcal{A}_\mathcal{P}) \cap \mathcal{L}(\mathcal{A}_\Sigma) = \emptyset$$

- trace $st_0 \dots st_{n-1}$ is a feasible error trace of $\mathcal{P}$ if and only if there exists a nesting relation $\rightsquigarrow$ such that the nested trace $(st_0 \dots st_{n-1}, \rightsquigarrow)$ is at the same time a nested error trace and a feasible nested trace.

- the following two statements are equivalent:
  - sequence of locations $\ell_0, \dots, \ell_n$ is a run of $\mathcal{A}_\mathcal{P}$ for $(st_0 \dots st_{n-1}, \rightsquigarrow)$ and the sequence of valuations $\nu_0, \dots, \nu_n$ is a run of $\mathcal{A}_\Sigma$ for $(st_0 \dots st_{n-1}, \rightsquigarrow)$
  - there is a sequence of global states $S_1, \dots, S_n$ such that

$$(\ell_0, \nu_0) \xrightarrow{st_0} S_1.(\ell_1, \nu_1) \xrightarrow{st_1} \cdots \xrightarrow{st_{n-1}} S_n.(\ell_n, \nu_n)$$

according to the transition relation of $\mathcal{P}$, which means:
$S_i = (\ell_{k_0}, \nu_{k_0}) \dots (\ell_{k_m}, \nu_{k_m})$,
$k_0 \rightsquigarrow k'_0, \dots, k_m \rightsquigarrow k'_m$, and
$k_0 < \cdots < k_m < i \leq k'_m \leq \cdots \leq k'_0$

proof rule

(sound and complete)

$$\mathcal{L}(\mathcal{A}) \supseteq \mathcal{L}(\mathcal{A}_\Sigma),\ \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\mathcal{P}) = \emptyset \implies \mathcal{P} \text{ is correct}$$

next ...

- construct abstraction of data automaton via predicate abstraction of post operator
  - transition relation between local states <=> post operator over sets of states
  - abstraction of post operator <=> transition relation between abstract local states

Predicates. In our context, a *predicate* is a set of valuations. Such a set may be defined by an assertion, e.g., $x \geq 101$.

Bitvectors. Given a finite set of predicates, say

$$\mathsf{Pred} = \{p_1, \ldots, p_m\}$$

we call an m-tuple $\boldsymbol{b} \in \{0,1\}^m$ a *bitvector*. Assuming a fixed order on the predicates, a bitvector $\boldsymbol{b} = \langle b_1, \ldots, b_m \rangle$ has a meaning defined by

$$[\![\langle b_1, \ldots, b_m \rangle]\!] = \{\nu \mid \forall j \in \{1, \ldots, m\}.\, \nu \in p_j \Leftrightarrow b_j = 1\}.$$

$$
\begin{aligned}
post_{\mathsf{in}}(\mathcal{V}, \texttt{y:=t}) &= \{\nu \oplus \{y \mapsto \nu(t)\} \mid \nu \in \mathcal{V}\} \\
post_{\mathsf{in}}(\mathcal{V}, \phi) &= \{\nu \mid \nu \in \mathcal{V}, \nu \models \phi\} \\
post_{\mathsf{ca}}(\mathcal{V}, \texttt{call p}) &= \{\nu' \mid \{x \mapsto \nu(x_p)\} \in \nu', \nu \in \mathcal{V}\} \\
post_{\mathsf{re}}(\mathcal{V}, \mathcal{V}_<, \texttt{return p}) &= \{\nu_< \oplus \{res_p \mapsto \nu(res)\} \mid \\
&\qquad \nu(x) = \nu_<(x_f), \\
&\qquad \nu_< \in \mathcal{V}_<, \nu \in \mathcal{V}\}
\end{aligned}
$$

$$
\begin{aligned}
post_{\mathsf{in}}(\mathcal{V}, \texttt{y:=t}) &= \{\nu \oplus \{y \mapsto \nu(t)\} \mid \nu \in \mathcal{V}\} \\
post_{\mathsf{in}}(\mathcal{V}, \phi) &= \{\nu \mid \nu \in \mathcal{V}, \nu \models \phi\} \\
post_{\mathsf{ca}}(\mathcal{V}, \texttt{call p}) &= \{\nu' \mid \{x \mapsto \nu(x_p)\} \in \nu', \nu \in \mathcal{V}\} \\
post_{\mathsf{re}}(\mathcal{V}, \mathcal{V}_<, \texttt{return p}) &= \{\nu_< \oplus \{res_p \mapsto \nu(res)\} \mid \\
&\qquad \nu(x) = \nu_<(x_f), \\
&\qquad \nu_< \in \mathcal{V}_<, \nu \in \mathcal{V}\}
\end{aligned}
$$

$$
\begin{aligned}
post_{\mathsf{in}}(\mathcal{V}, st) &= \{\nu' \mid \exists\, \nu \in \mathcal{V} \;:\; (\nu, st, \nu') \in \delta_{\mathsf{in}}\} \\
post_{\mathsf{ca}}(\mathcal{V}, st) &= \{\nu' \mid \exists\, \nu \in \mathcal{V} \;:\; (\nu, st, \nu') \in \delta_{\mathsf{ca}}\} \\
post_{\mathsf{re}}(\mathcal{V}, \mathcal{V}_<, st) &= \{\nu' \mid \exists\, \nu \in \mathcal{V}\; \exists\, \nu_< \in \mathcal{V}_< \;: \\
&\qquad \nu(x) = \nu_<(x_p), \\
&\qquad (\nu, \nu_<, st, \nu') \in \delta_{\mathsf{re}}\}
\end{aligned}
$$

$$
\begin{aligned}
post_{\mathsf{in}}(\mathcal{V}, \boxed{\texttt{y:=t}}) &= \{\nu \oplus \{y \mapsto \nu(t)\} \mid \nu \in \mathcal{V}\} \\
post_{\mathsf{in}}(\mathcal{V}, \boxed{\phi}) &= \{\nu \mid \nu \in \mathcal{V}, \nu \models \phi\} \\
post_{\mathsf{ca}}(\mathcal{V}, \boxed{\texttt{call p}}) &= \{\nu' \mid \{x \mapsto \nu(x_p)\} \in \nu', \nu \in \mathcal{V}\} \\
post_{\mathsf{re}}(\mathcal{V}, \mathcal{V}_<, \boxed{\texttt{return p}}) &= \{\nu_< \oplus \{res_p \mapsto \nu(res)\} \mid \\
&\qquad \nu(x) = \nu_<(x_f), \\
&\qquad \nu_< \in \mathcal{V}_<, \nu \in \mathcal{V}\}
\end{aligned}
$$

$$
\begin{aligned}
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \boxed{\texttt{y:=t}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \boxed{\texttt{y:=t}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \boxed{\phi}) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \boxed{\phi}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{ca}}(\boldsymbol{b}, \boxed{\texttt{call p}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{ca}}([\![\boldsymbol{b}]\!], \boxed{\texttt{call p}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{re}}(\boldsymbol{b}, \boldsymbol{b}_<, \boxed{\texttt{return p}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{re}}([\![\boldsymbol{b}]\!], [\![\boldsymbol{b}_<]\!], \boxed{\texttt{return p}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\}
\end{aligned}
$$

$$
\begin{aligned}
post_{\mathsf{in}}(\mathcal{V}, \texttt{y:=t}) &= \{\nu \oplus \{y \mapsto \nu(t)\} \mid \nu \in \mathcal{V}\} \\
post_{\mathsf{in}}(\mathcal{V}, \phi) &= \{\nu \mid \nu \in \mathcal{V}, \nu \models \phi\} \\
post_{\mathsf{ca}}(\mathcal{V}, \texttt{call p}) &= \{\nu' \mid \{x \mapsto \nu(x_p)\} \in \nu', \nu \in \mathcal{V}\} \\
post_{\mathsf{re}}(\mathcal{V}, \mathcal{V}_<, \texttt{return p}) &= \{\nu_< \oplus \{res_p \mapsto \nu(res)\} \mid \\
&\qquad\qquad \nu(x) = \nu_<(x_f), \\
&\qquad\qquad \nu_< \in \mathcal{V}_<, \nu \in \mathcal{V}\}
\end{aligned}
$$

$$
\begin{aligned}
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \texttt{y:=t}) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \texttt{y:=t}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \phi) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \phi) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{ca}}(\boldsymbol{b}, \texttt{call p}) &= \{\boldsymbol{b}' \mid post_{\mathsf{ca}}([\![\boldsymbol{b}]\!], \texttt{call p}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{re}}(\boldsymbol{b}, \boldsymbol{b}_<, \texttt{return p}) &= \{\boldsymbol{b}' \mid post_{\mathsf{re}}([\![\boldsymbol{b}]\!], [\![\boldsymbol{b}_<]\!], \texttt{return p}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\}
\end{aligned}
$$

predicate automaton

$$\mathcal{A}_{\mathsf{Pred}} = (Q, \langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle, Q^{\mathsf{init}}, Q^{\mathsf{fin}})$$

- *the set of states Q consists of all bitvectors,*
- *every state is initial, i.e., $Q^{\mathsf{init}} = Q$,*
- *every state is final, i.e., $Q^{\mathsf{fin}} = Q$,*
- *the triple of transition relations $\langle \delta_{\mathsf{in}}, \delta_{\mathsf{ca}}, \delta_{\mathsf{re}} \rangle$ corresponds to the triple $\langle post^{\#}_{\mathsf{in}}, post^{\#}_{\mathsf{ca}}, post^{\#}_{\mathsf{re}} \rangle$ of abstract nested post operators*
  - *if st is an assignment or assume statement, then*

    $$(\boldsymbol{b}, st, \boldsymbol{b}') \in \delta_{\mathsf{in}} \quad \text{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{in}}(\boldsymbol{b}, st),$$

  - *if st is a call statement, then*

    $$(\boldsymbol{b}, st, \boldsymbol{b}') \in \delta_{\mathsf{ca}} \quad \text{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{ca}}(\boldsymbol{b}, st),$$

  - *if st is a return statement, then*

    $$(\boldsymbol{b}, \boldsymbol{b}_{<}, st, \boldsymbol{b}') \in \delta_{\mathsf{re}} \quad \text{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{re}}(\boldsymbol{b}, \boldsymbol{b}_{<}, st).$$

$$
\begin{aligned}
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \boxed{\texttt{y:=t}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \boxed{\texttt{y:=t}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{in}}(\boldsymbol{b}, \boxed{\phi}) &= \{\boldsymbol{b}' \mid post_{\mathsf{in}}([\![\boldsymbol{b}]\!], \boxed{\phi}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{ca}}(\boldsymbol{b}, \boxed{\texttt{call p}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{ca}}([\![\boldsymbol{b}]\!], \boxed{\texttt{call p}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\} \\
post^{\#}_{\mathsf{re}}(\boldsymbol{b}, \boldsymbol{b}_{<}, \boxed{\texttt{return p}}) &= \{\boldsymbol{b}' \mid post_{\mathsf{re}}([\![\boldsymbol{b}]\!], [\![\boldsymbol{b}_{<}]\!], \boxed{\texttt{return p}}) \cap [\![\boldsymbol{b}']\!] \neq \emptyset\}
\end{aligned}
$$

- *if st is an assignment or assume statement, then*

  $(\boldsymbol{b}, st, \boldsymbol{b}') \in \delta_{\mathsf{in}} \quad \textit{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{in}}(\boldsymbol{b}, st),$
- *if st is a call statement, then*

  $(\boldsymbol{b}, st, \boldsymbol{b}') \in \delta_{\mathsf{ca}} \quad \textit{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{ca}}(\boldsymbol{b}, st),$
- *if st is a return statement, then*

  $(\boldsymbol{b}, \boldsymbol{b}_{<}, st, \boldsymbol{b}') \in \delta_{\mathsf{re}} \quad \textit{if} \quad \boldsymbol{b}' \in post^{\#}_{\mathsf{re}}(\boldsymbol{b}, \boldsymbol{b}_{<}, st).$

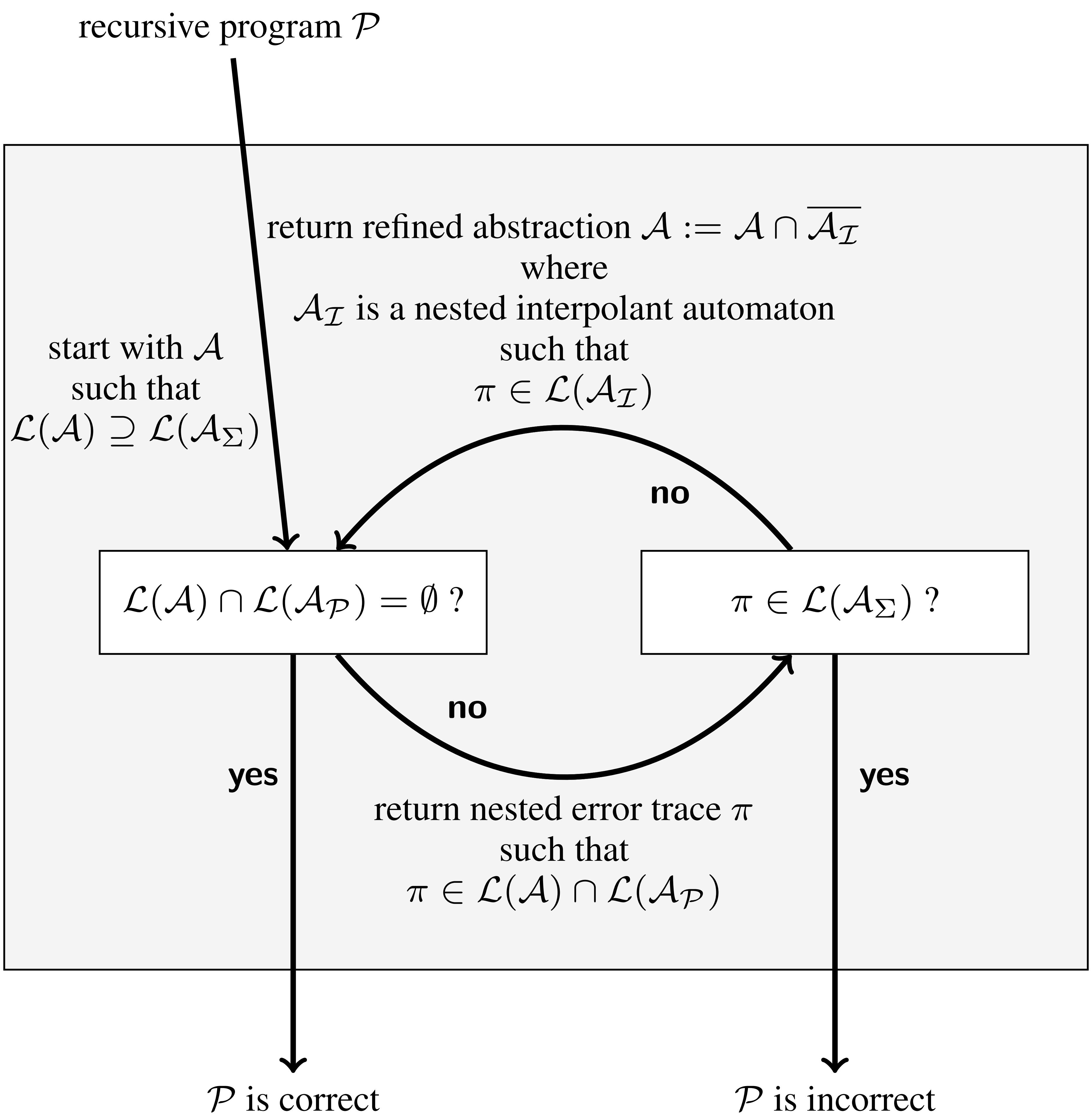
$$\mathcal{L}(\mathcal{A}_{\mathsf{Pred}}) \supseteq \mathcal{L}(\mathcal{A}_\Sigma)$$

- an accepting run $\nu_0, \ldots, \nu_n$ of the data automaton $\mathcal{A}_\Sigma$ on a nested trace π gives rise to an accepting run $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_n$ of the predicate automaton $\mathcal{A}_{\mathsf{Pred}}$ on π
- i-th bitvector contains i-th valuation

$$\nu_i \in [\![\boldsymbol{b}_i]\!]$$

soundness of predicate abstraction for recursive programs

$$\mathcal{L}(\mathcal{A}_{\mathsf{Pred}}) \cap \mathcal{L}(\mathcal{A}_{\mathcal{P}}) = \emptyset \implies \mathcal{P}$$


recursive program $\mathcal{P}$
return refined abstraction $\mathcal{A} := \mathcal{A} \cap \overline{\mathcal{A}_\mathcal{I}}$
where
$\mathcal{A}_\mathcal{I}$ is a nested interpolant automaton
such that
$\pi \in \mathcal{L}(\mathcal{A}_\mathcal{I})$
start with $\mathcal{A}$
such that
$\mathcal{L}(\mathcal{A}) \supseteq \mathcal{L}(\mathcal{A}_\Sigma)$
no
$\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\mathcal{P}) = \emptyset$?
$\pi \in \mathcal{L}(\mathcal{A}_\Sigma)$?
no
yes
yes
return nested error trace π
such that
$\pi \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\mathcal{P})$
$\mathcal{P}$ is correct
$\mathcal{P}$ is incorrect

problem

- how can we generalize an infeasible nested trace to a nested interpolant automaton?
  - what are interpolants for an infeasible nested trace?
    - what is the single static assignment for a nested trace?