Trace-based Deductive Verification

KeY Symposium 2023

Richard Bubel, Dilian Gurov, Reiner Hähnle, Marco Scaletta

Bergen, 08.08.2023

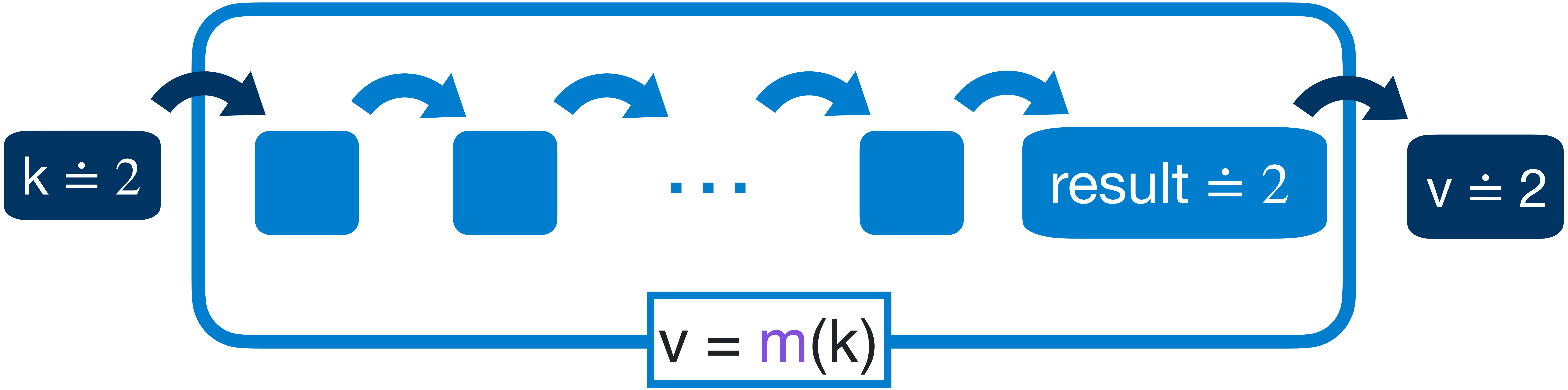
Part I

State-based Contracts

And their limitations

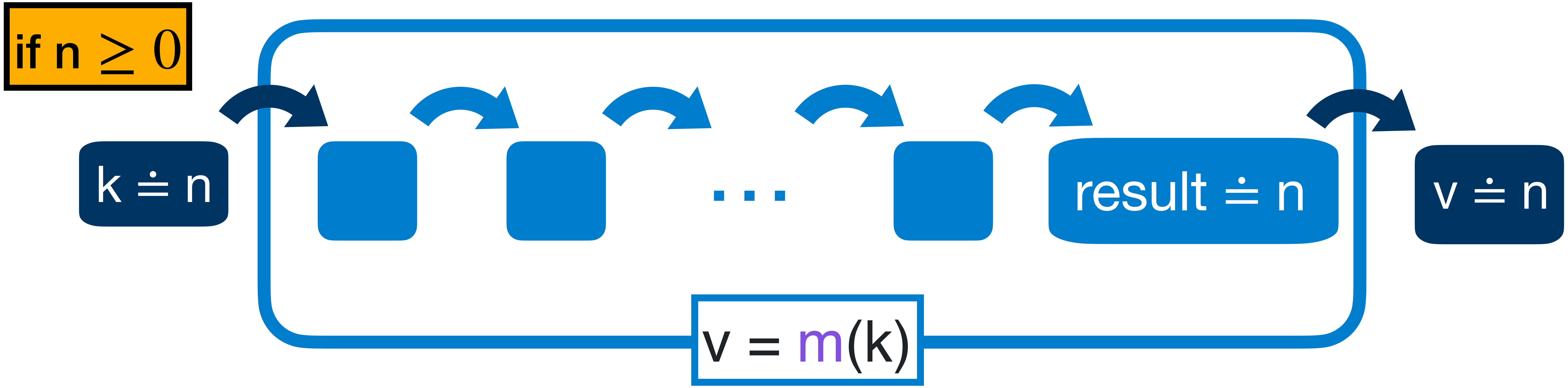
Example

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



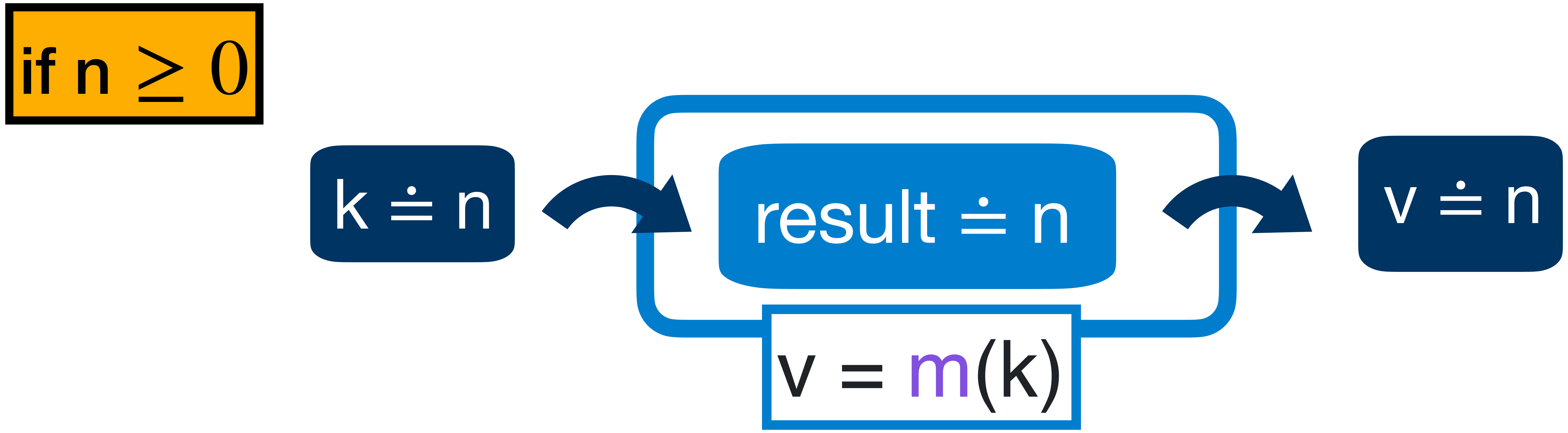
Example

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



Example

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



State-based Contracts

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

pre : $n \geq 0$

↓

v = m(n)

↓

post : $v \doteq n$

State-based Contracts

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

pre : $k-1 \geq 0$

v = m(k-1)

post : $v \doteq k-1$

State-based Contracts

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = k-1 ;
        r = r + 1
    };
    return r
}
```

pre : k-1 ≥ 0

v = m(k-1)

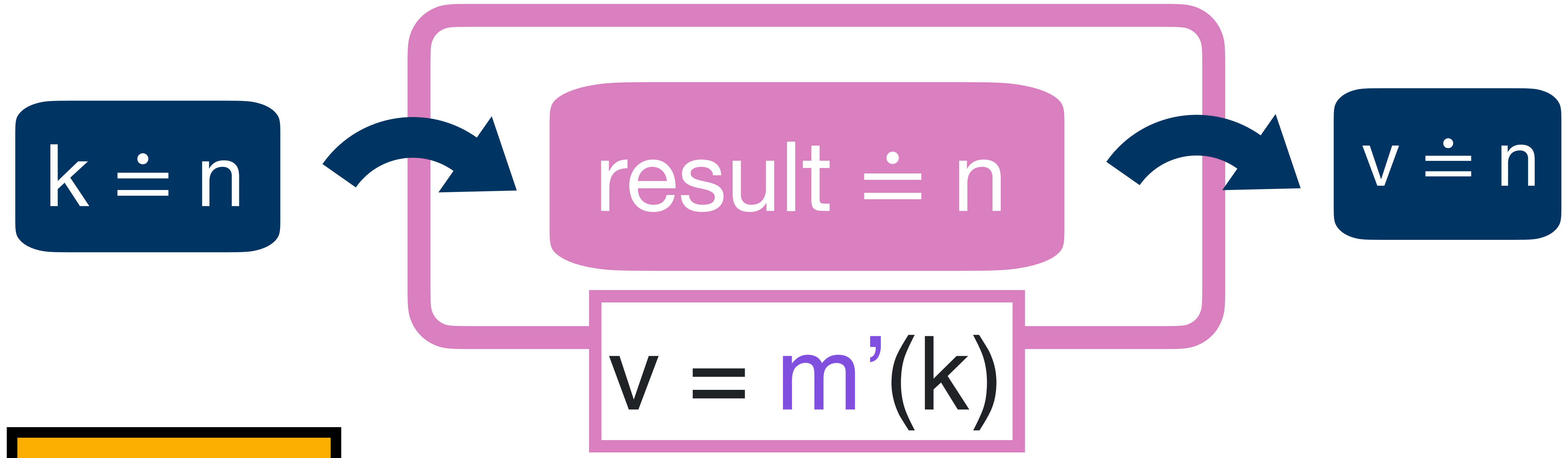
post : v $\doteq$ k-1

Modular Verification of Recursive Procedure!

Limitations

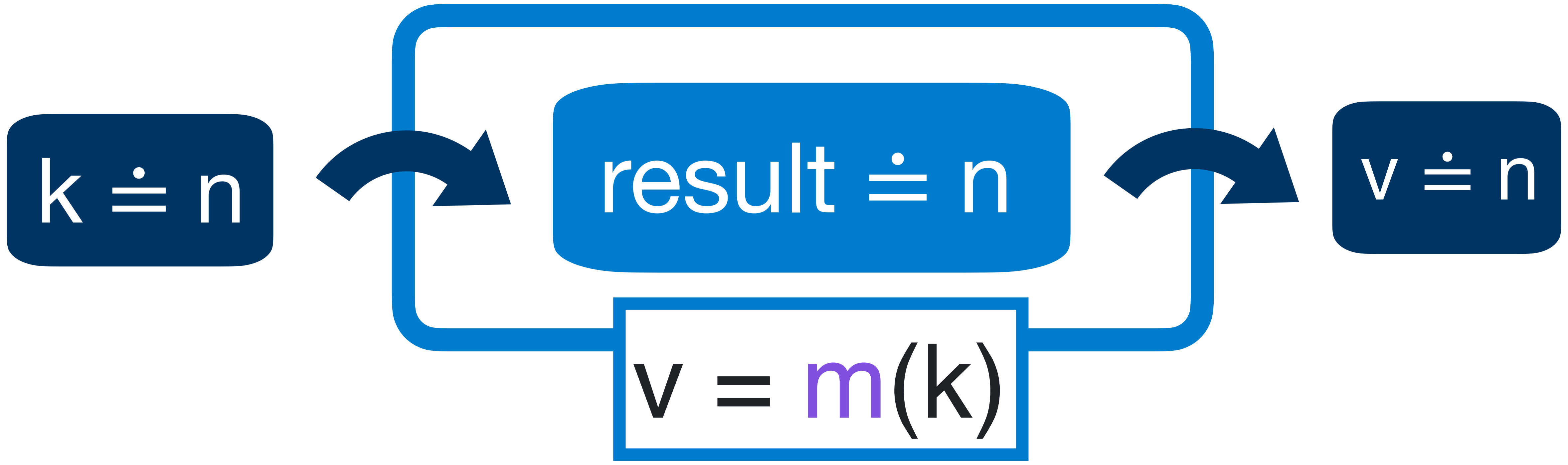
Aren't we abstracting too much?

```
m'(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = k-1;
        r = r+1
    };
    return r
}
```

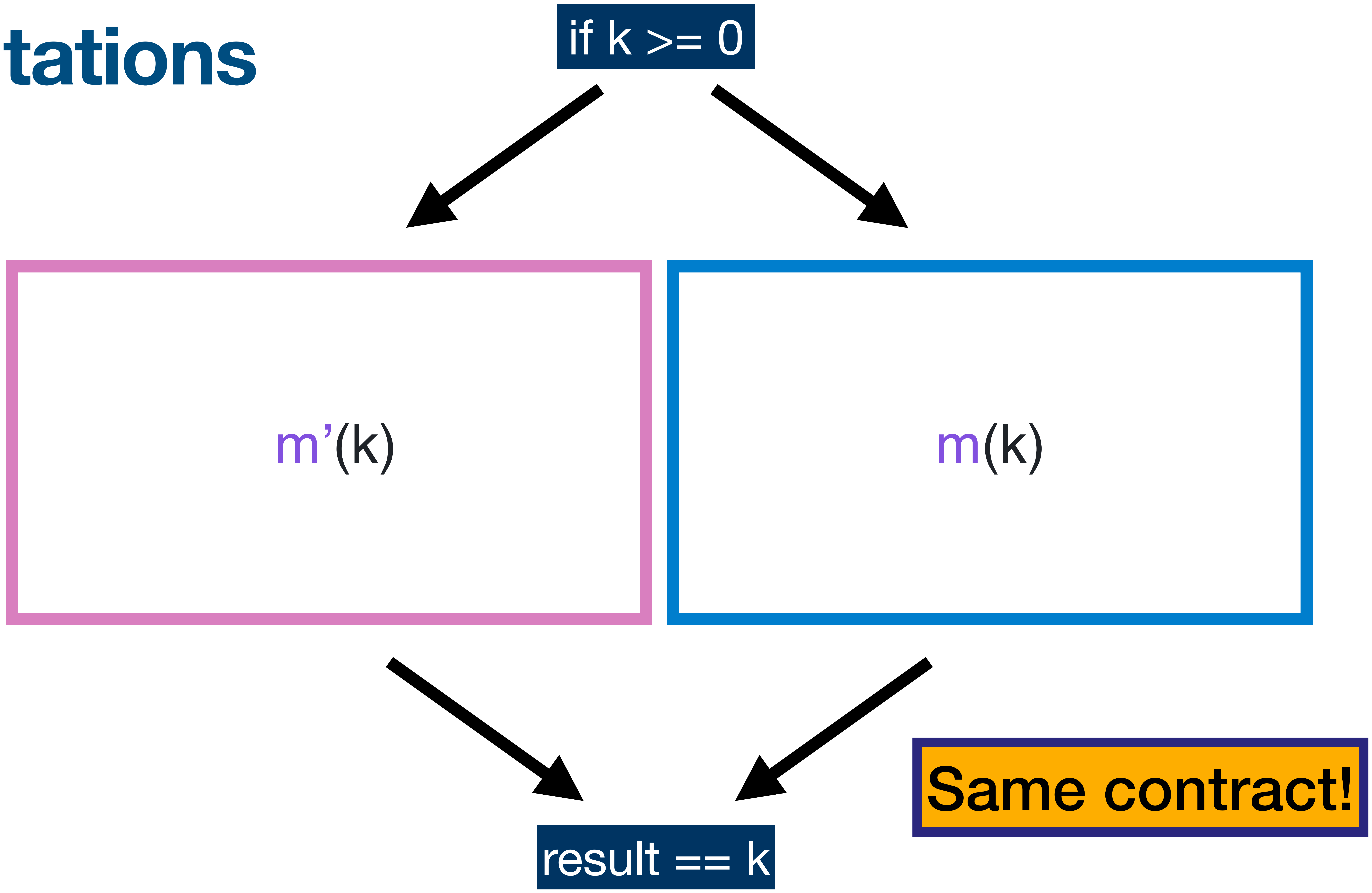


if $n \geq 0$

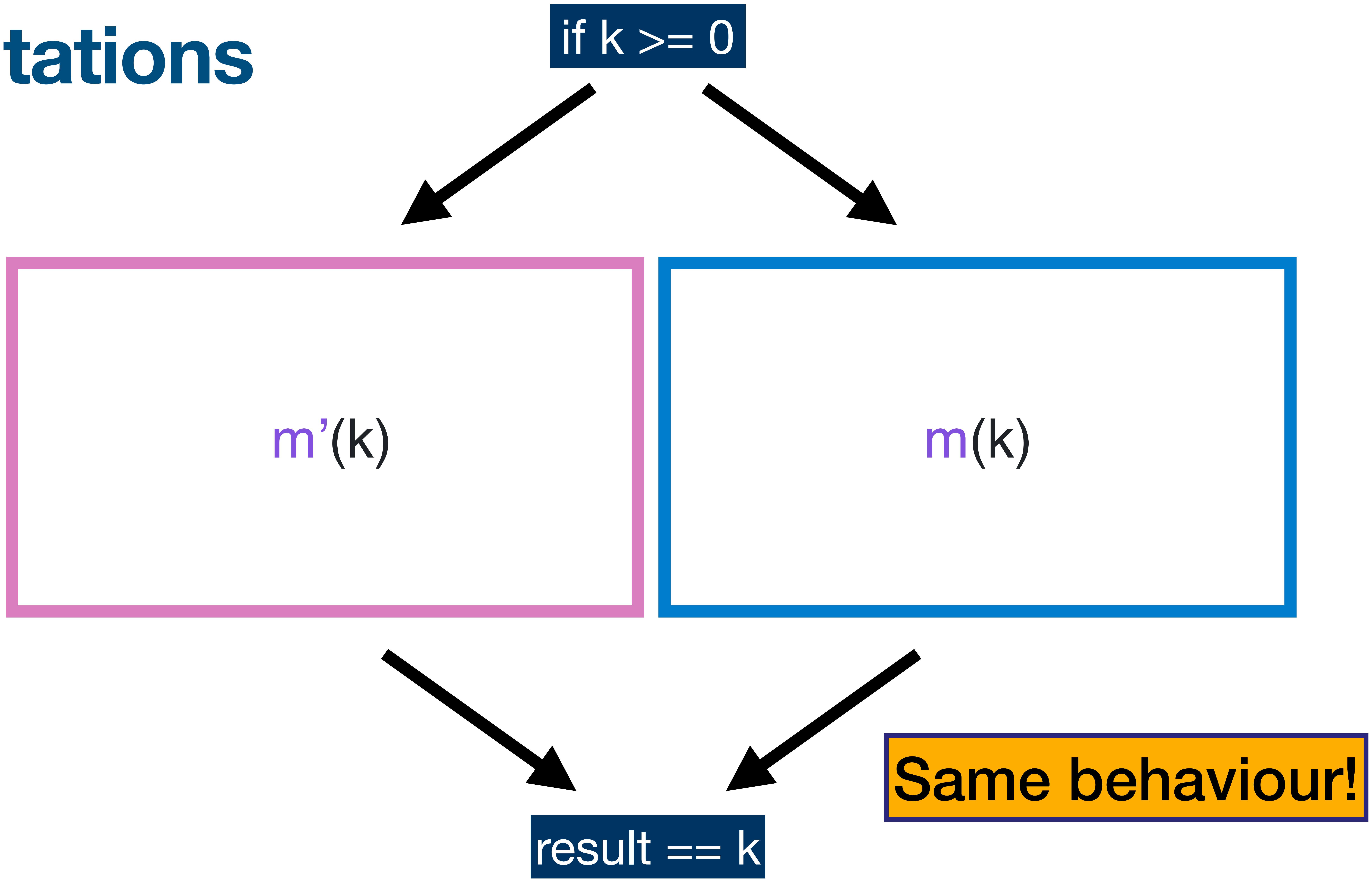
```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r+1
    };
    return r
}
```



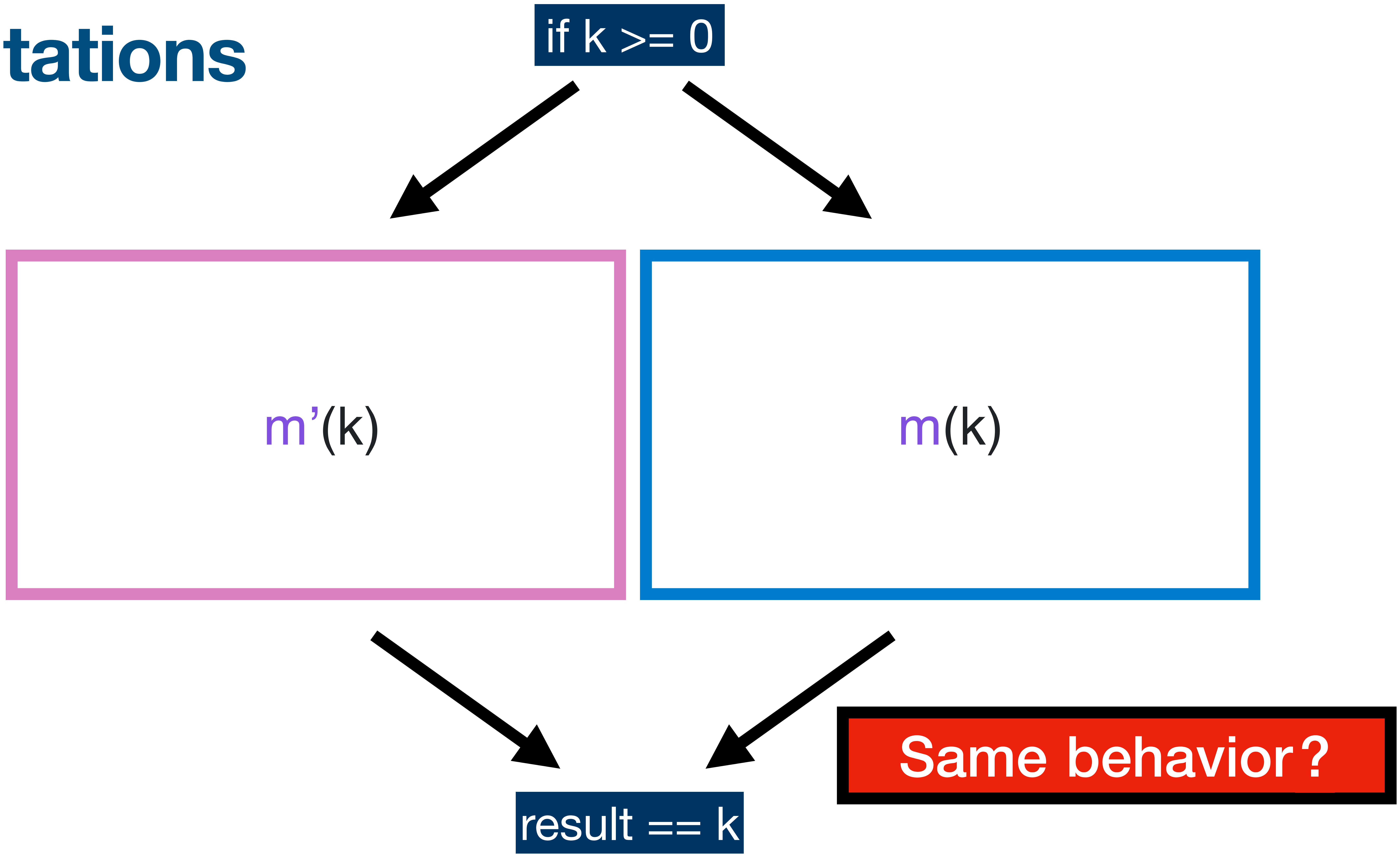
Limitations



Limitations



Limitations



Limitations

Limitations

What Happens During Execution?

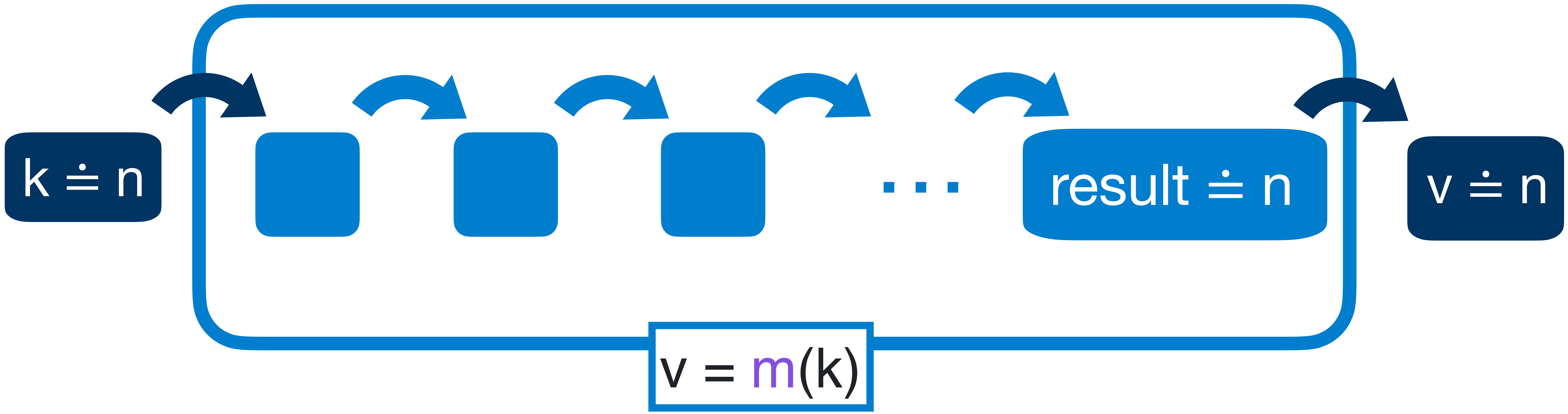
```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
Intermediate annotations?
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

Low readability!

No independence of the code!

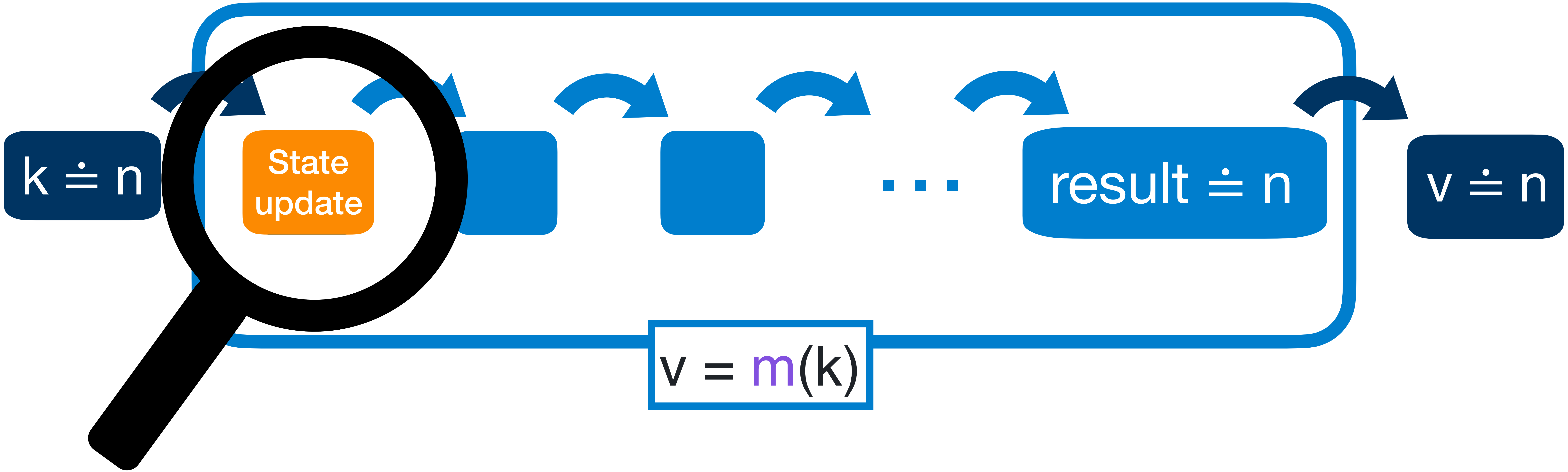
What Happens During Execution?

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



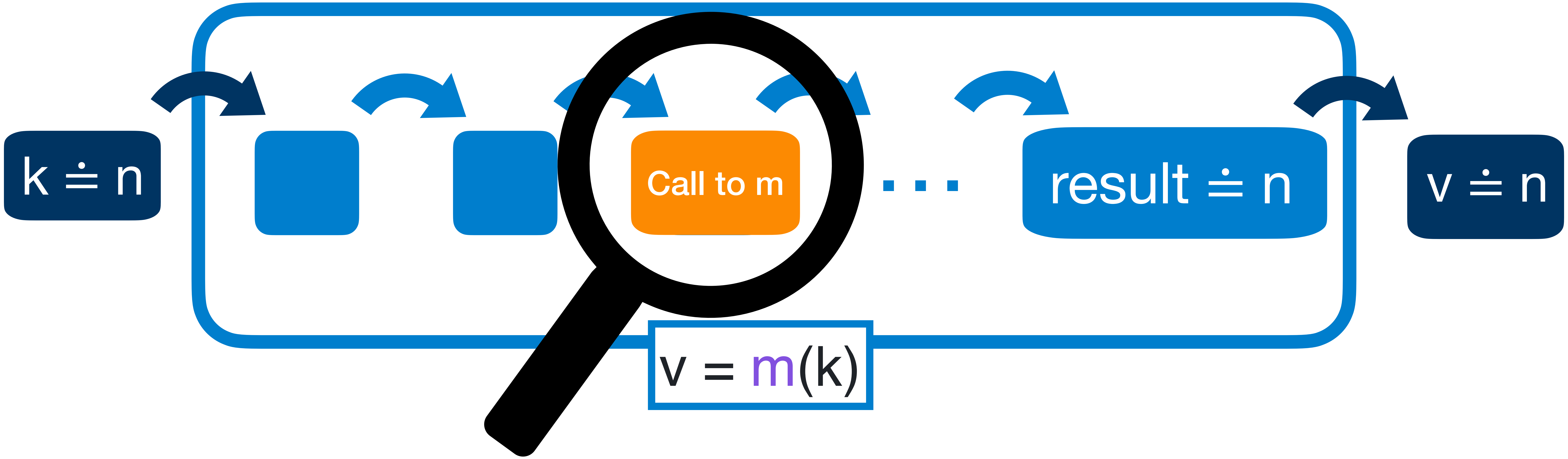
What Happens During Execution?

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



What Happens During Execution?

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



Part II

A Trace Semantics

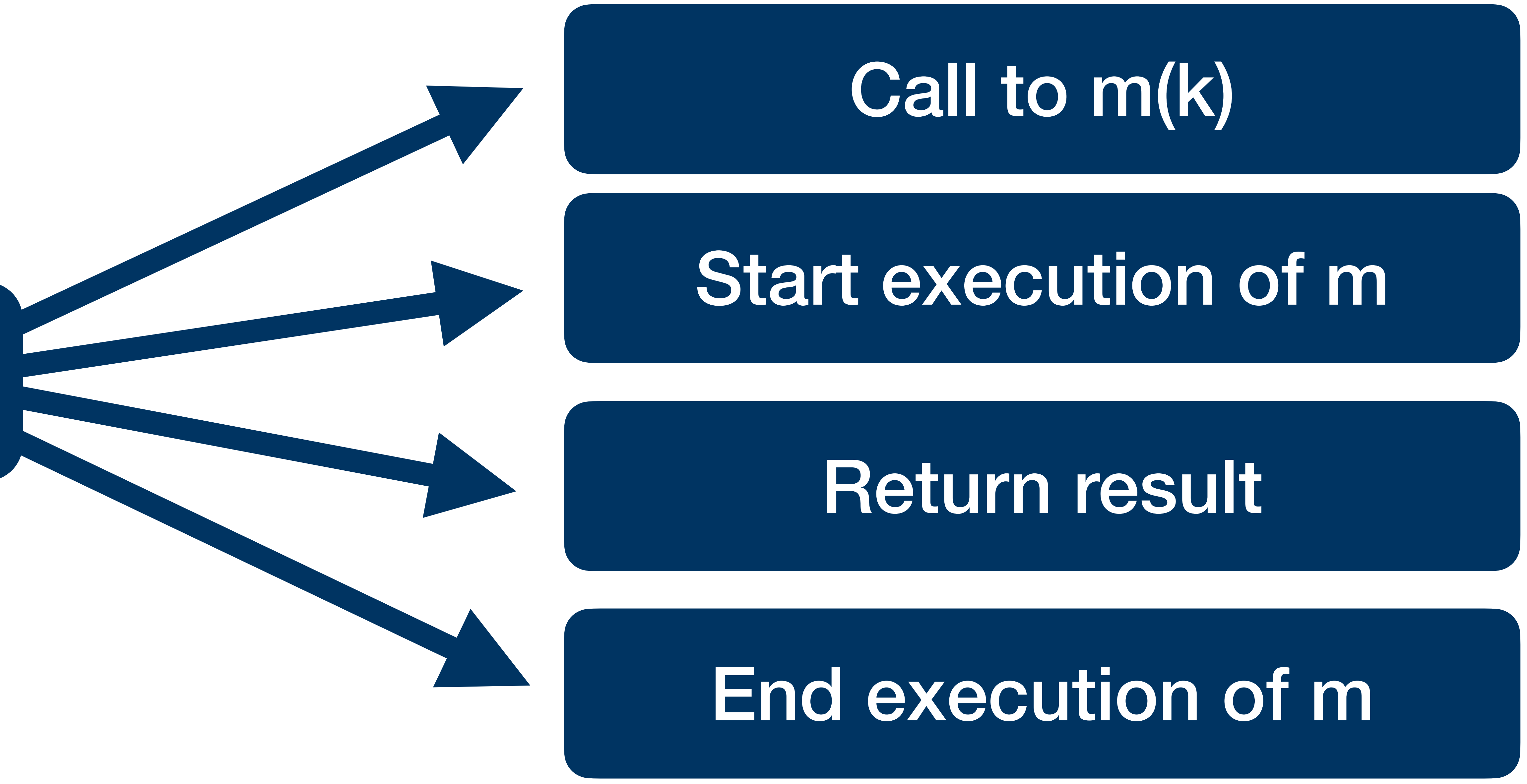
Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$
$$t ::= \sigma \mid \mathsf{ev}(\overline{\mathsf{e}})$$

Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$

$$t ::= \sigma \mid \mathrm{ev}(\bar{\mathrm{e}})$$



Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$

$$t ::= \sigma \mid \text{ev}(\overline{\text{e}})$$

- $\text{call}(m, k, callId)$ — Call to m(k)
- Start execution of m
- Return result
- End execution of m

Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$

$$t ::= \sigma \mid \mathsf{ev}(\bar{\mathsf{e}})$$

$\mathsf{call}(m, k, callId)$ — Call to m(k)

$\mathsf{push}((m, callId))$ — Start execution of m

Return result

End execution of m

Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$

$$t ::= \sigma \mid \mathsf{ev}(\overline{\mathsf{e}})$$

$\mathsf{call}(m, k, callId)$ — Call to m(k)

$\mathsf{push}((m, callId))$ — Start execution of m

$\mathsf{ret}(res)$ — Return result

End execution of m

Traces

$$\tau ::= \varepsilon \mid \tau \curvearrowright t$$

$$t ::= \sigma \mid \text{ev}(\bar{\text{e}})$$

call$(m, k, callId)$	Call to m(k)
push$((m, callId))$	Start execution of m
ret(res)	Return result
pop$((m, callId))$	End execution of m

Trace Semantics

```
{
    x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

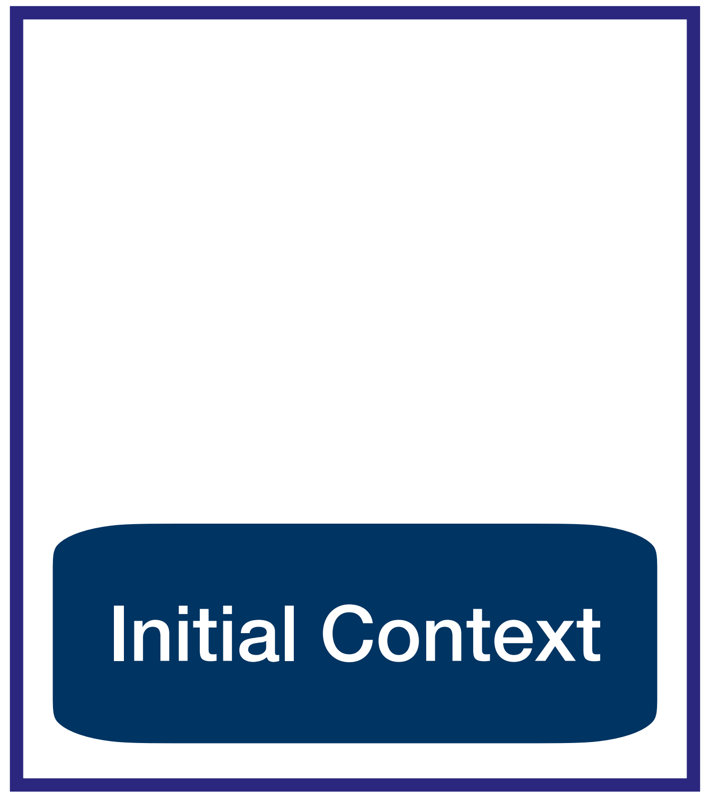
σ_0

$[x \mapsto 0]$

Initial Context

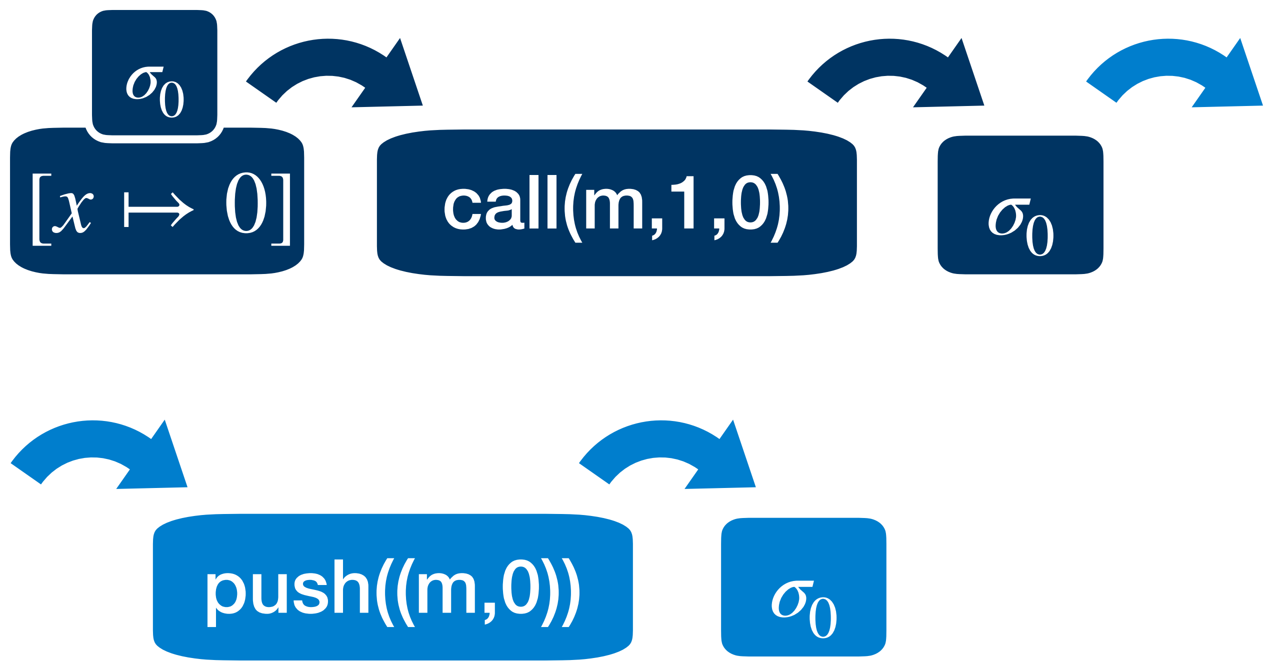
Trace Semantics

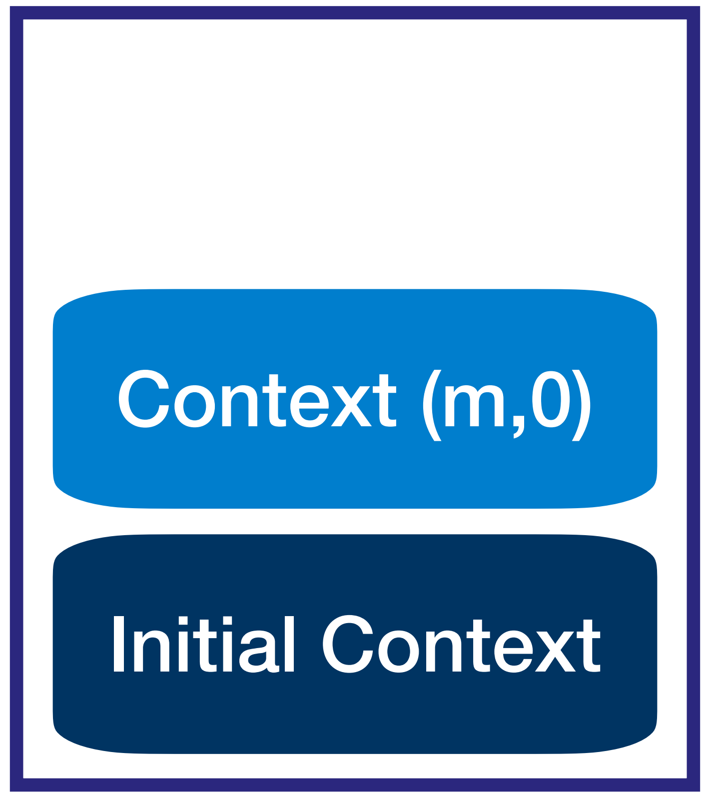
```
{
  x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



Trace Semantics

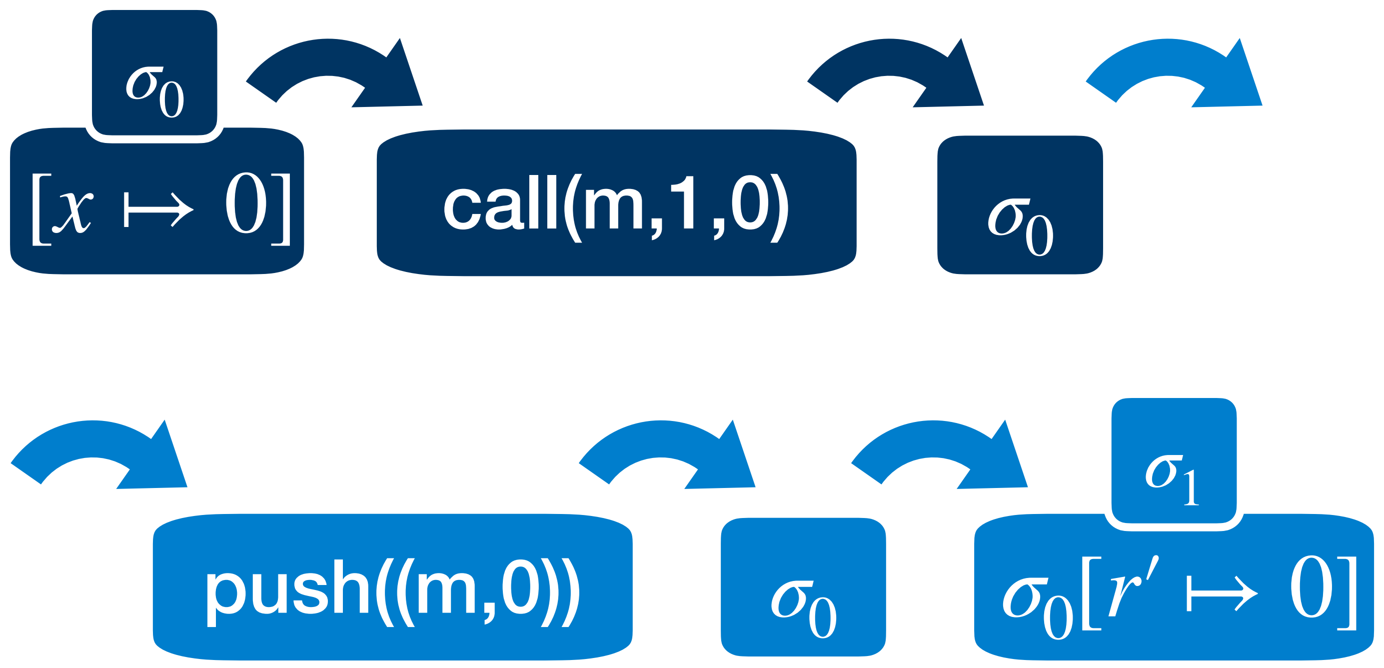
```
{
    x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

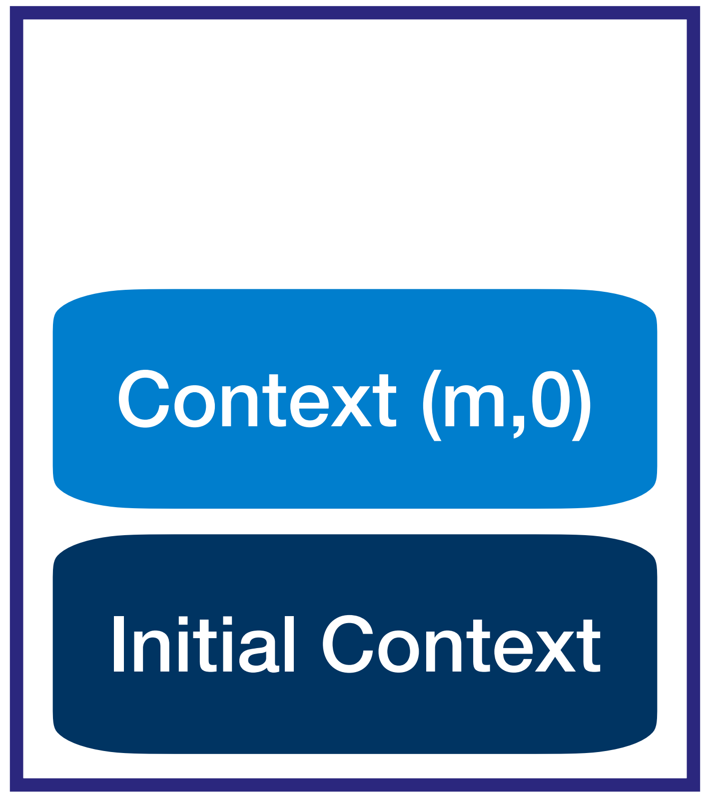




Trace Semantics

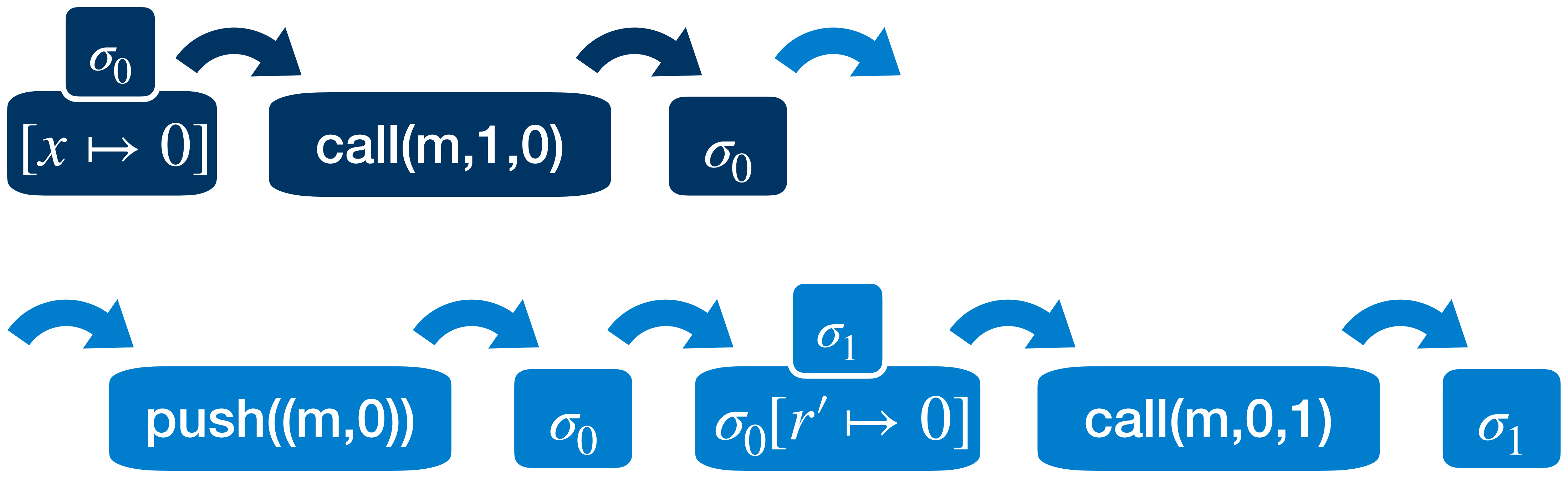
```
{
  x = m(1)
}

m(k) {
  r ; // initialized to 0
  if (k != 0) {
    r = m(k-1);
    r = r + 1
  };
  return r
}
```

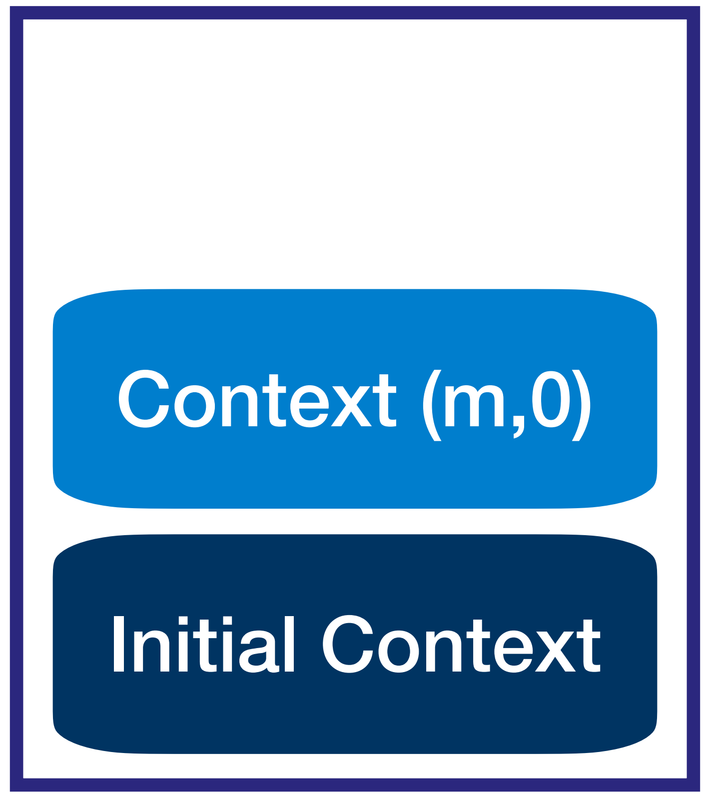




Trace Semantics

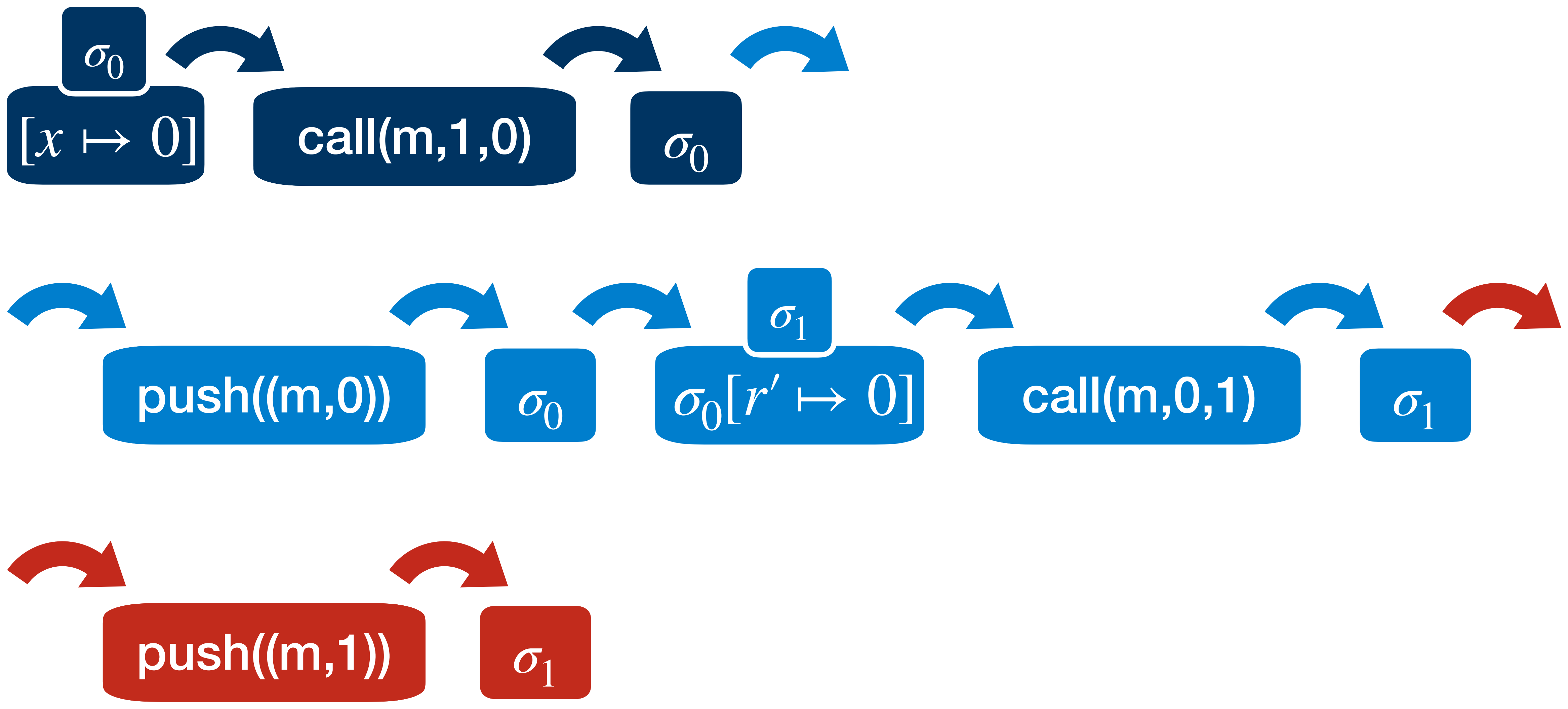
```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
      r = m(k-1);
      r = r + 1
   };
   return r
}
```

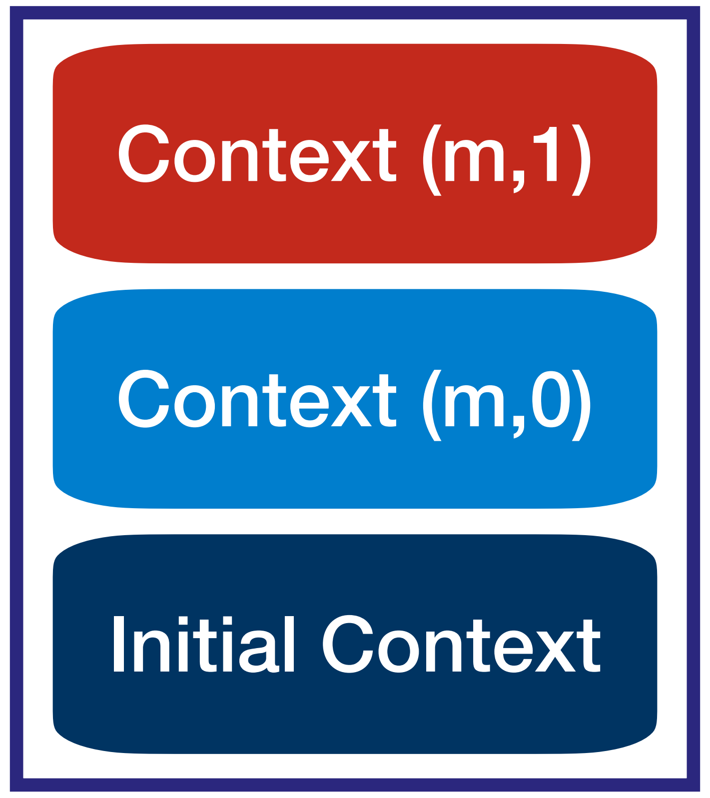




Trace Semantics

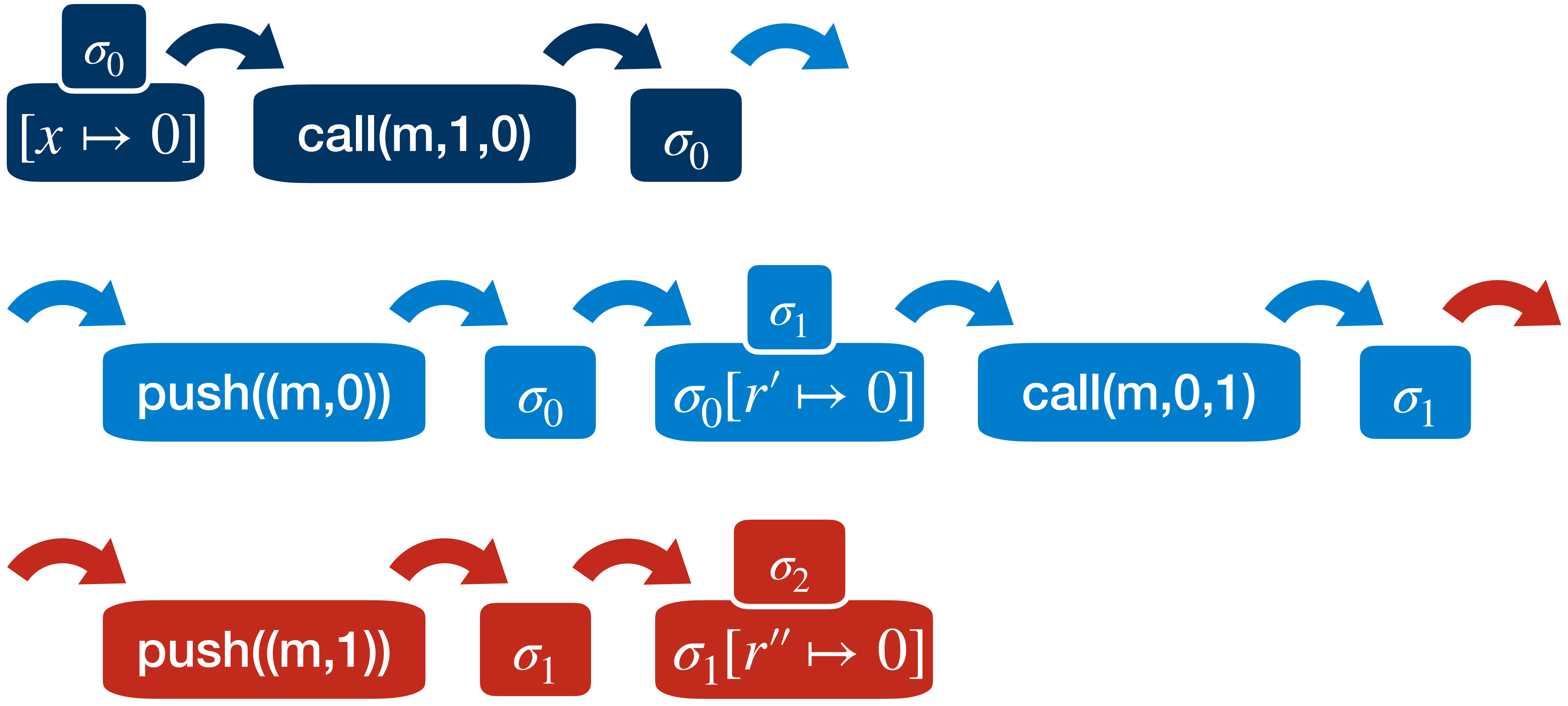
```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
      r = m(k-1);
      r = r + 1
   };
   return r
}
```

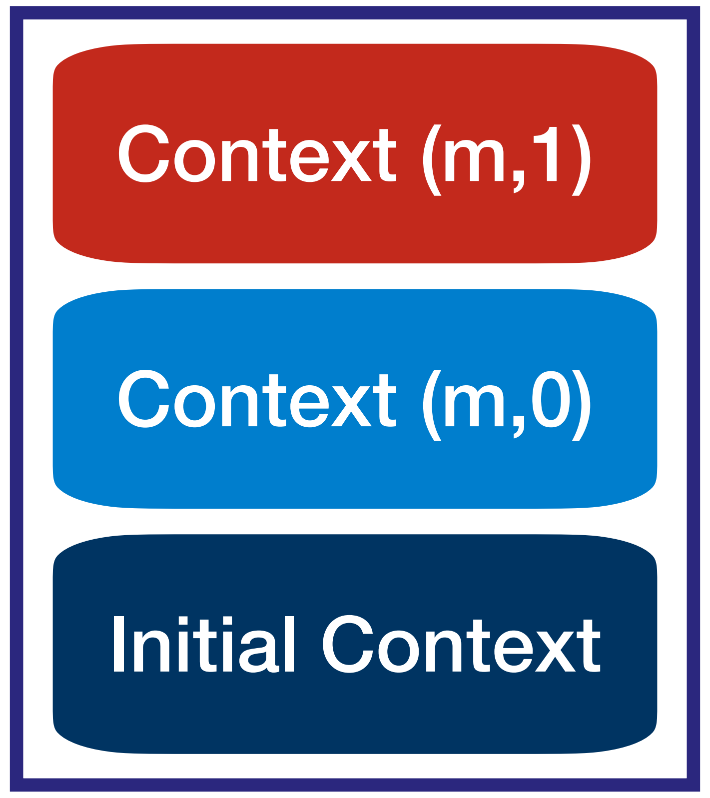




Trace Semantics

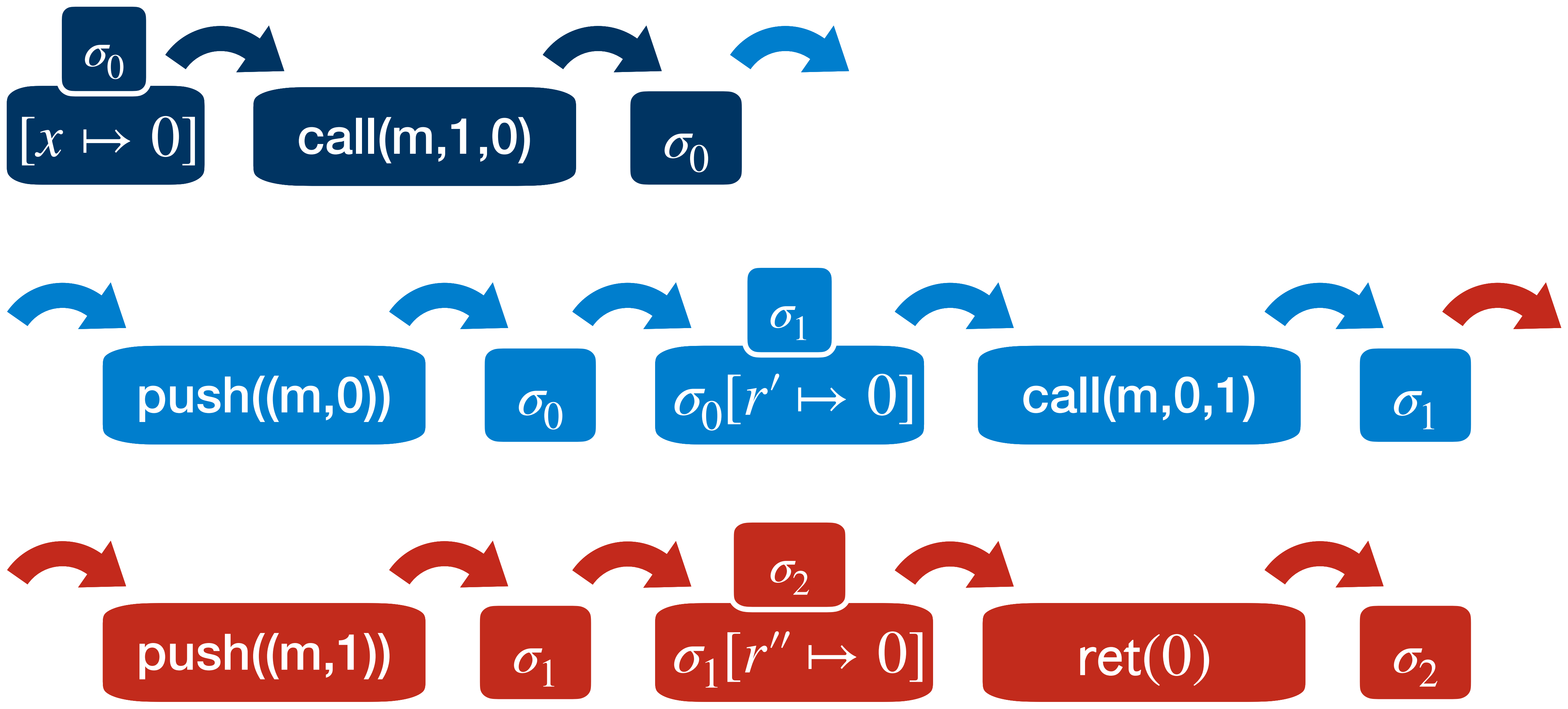
```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
      r = m(k-1);
      r = r + 1
   };
   return r
}
```

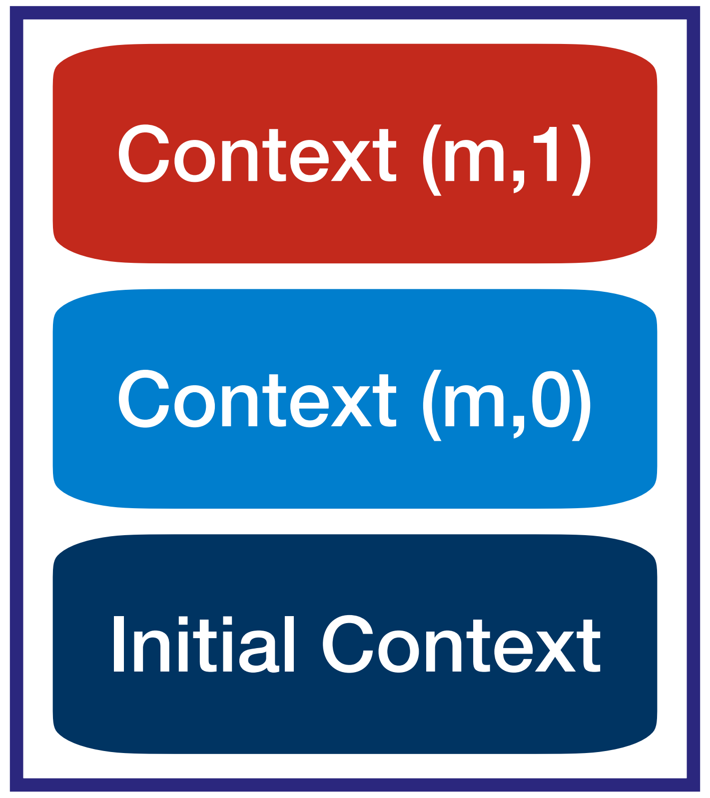




Trace Semantics

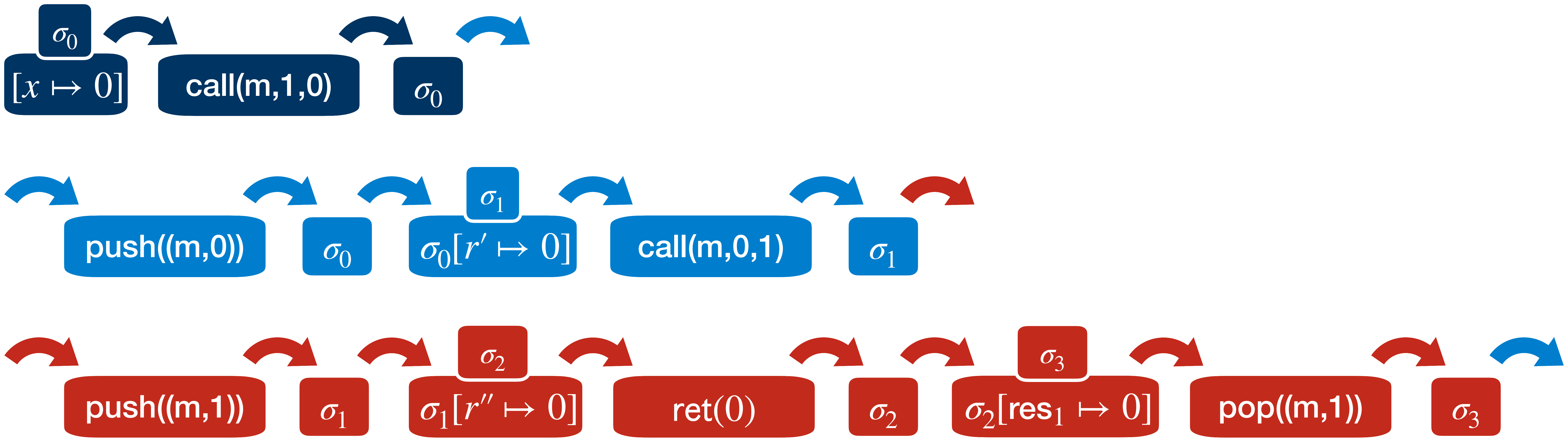
```
{
    x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

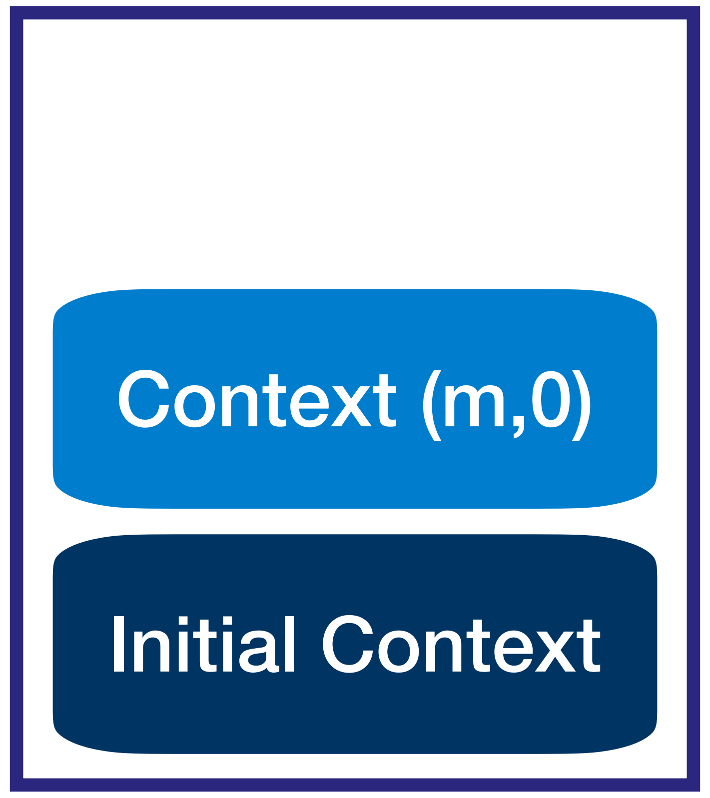




Trace Semantics

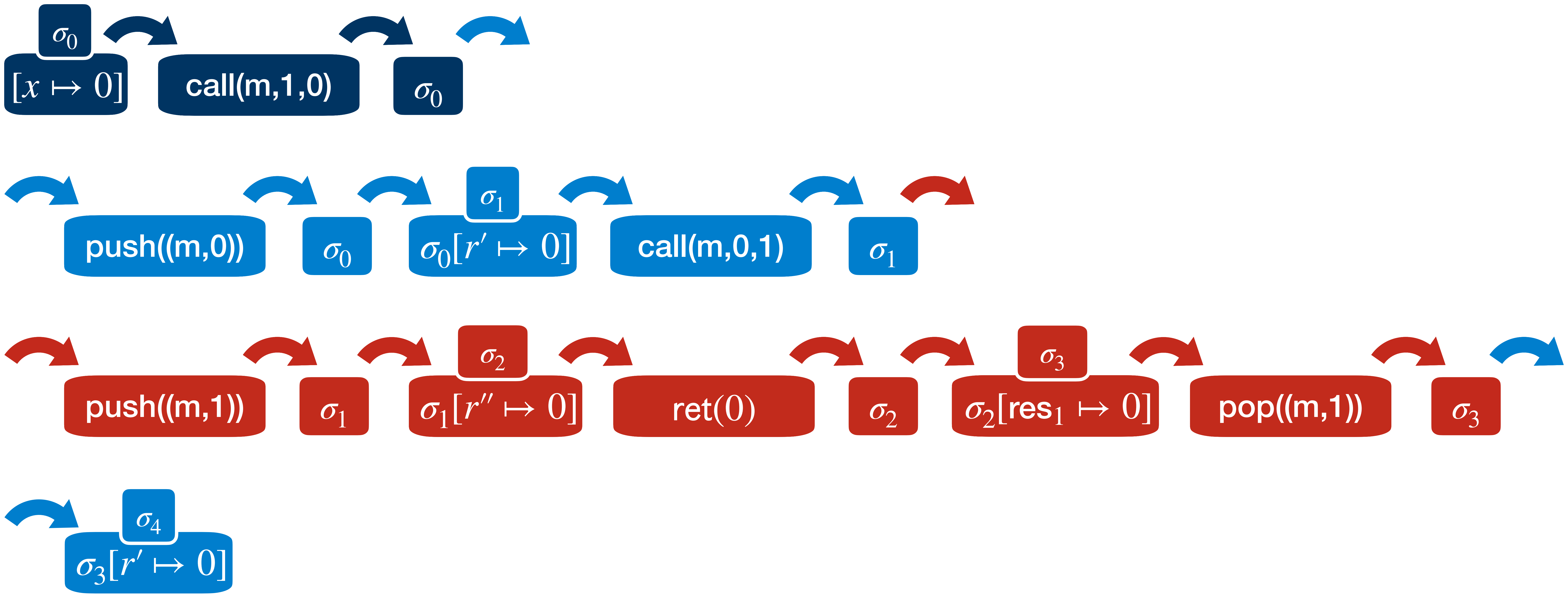
```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
      r = m(k-1);
      r = r + 1
   };
   return r
}
```

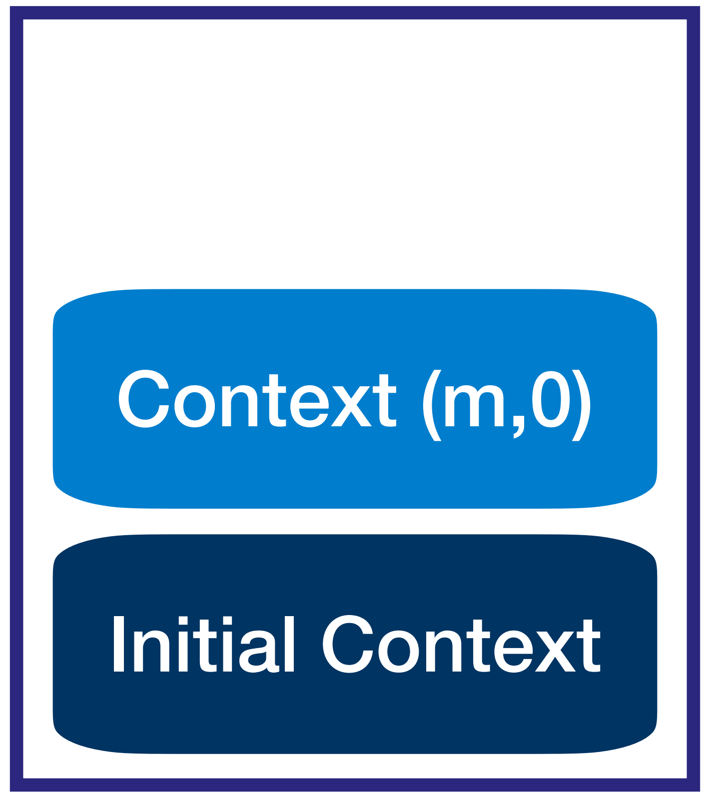




Trace Semantics

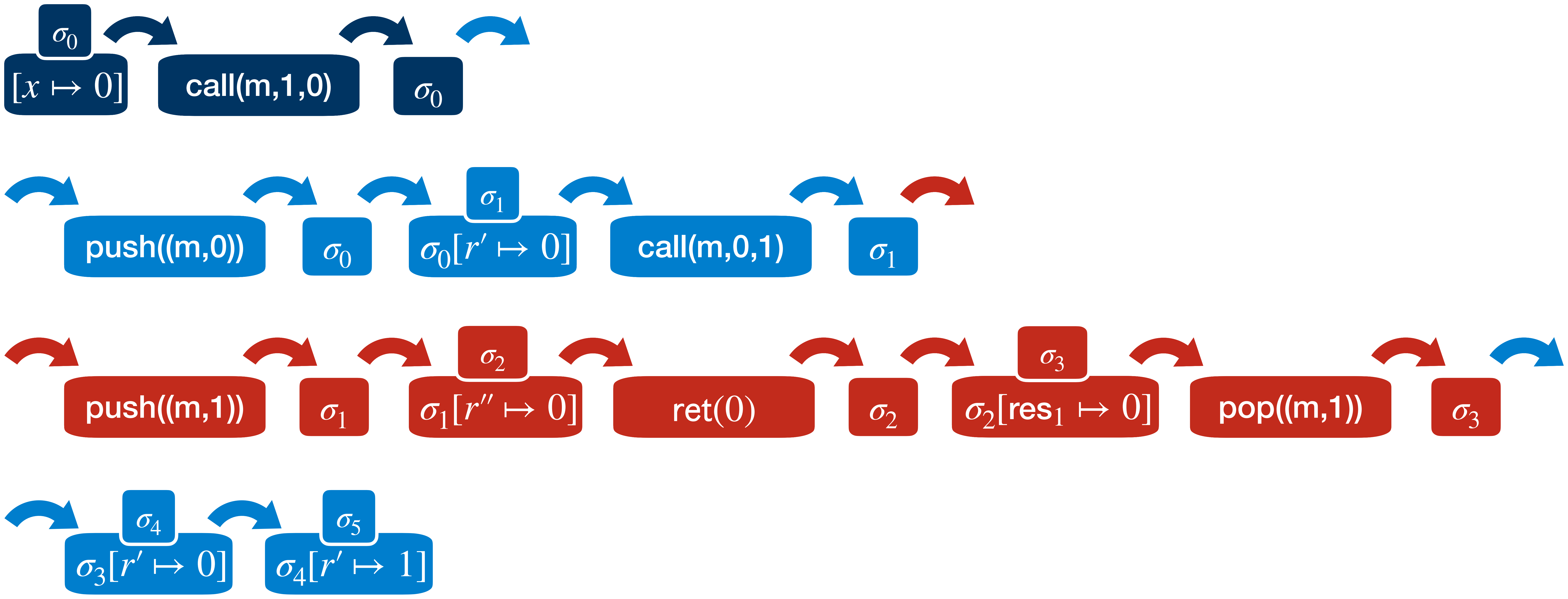
```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
  →  r = m(k-1);
     r = r + 1
   };
   return r
}
```

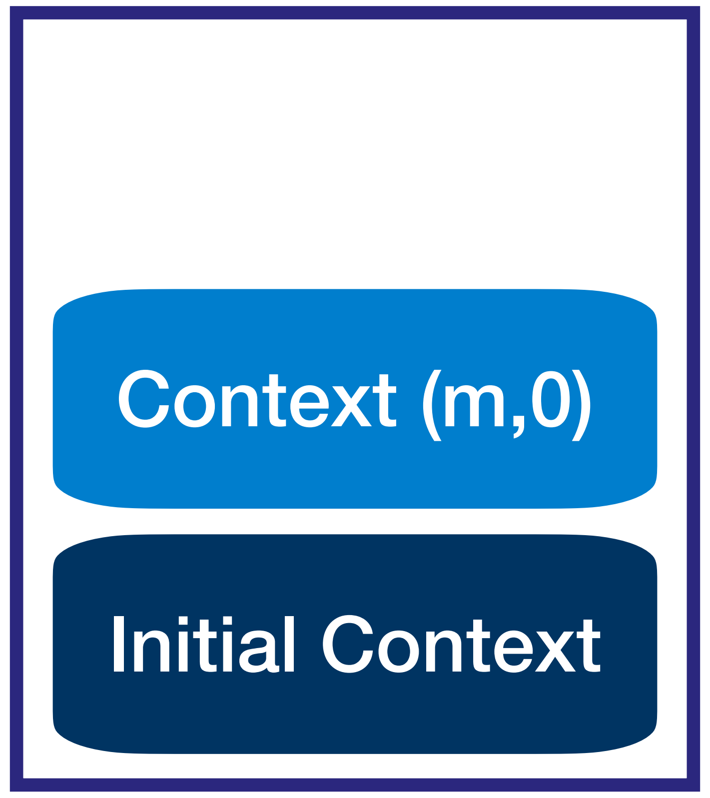




Trace Semantics

```
{
    x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



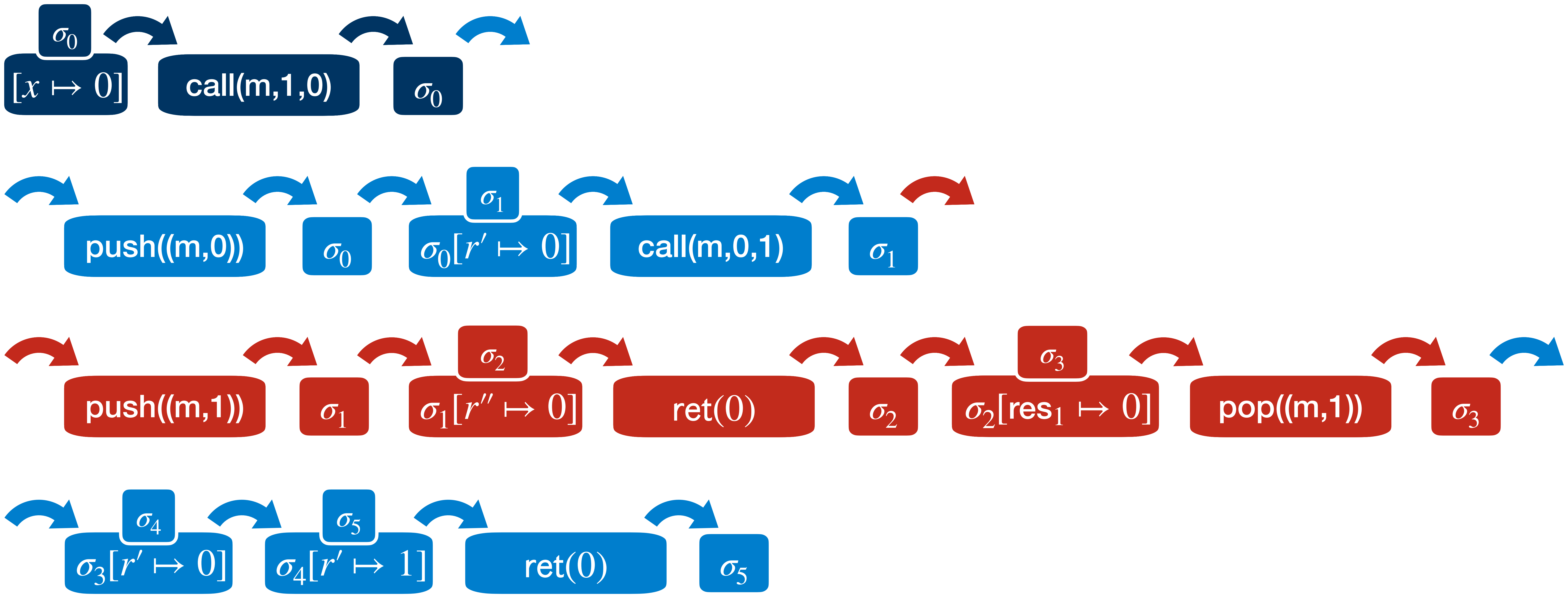


Trace Semantics

```
{
    x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

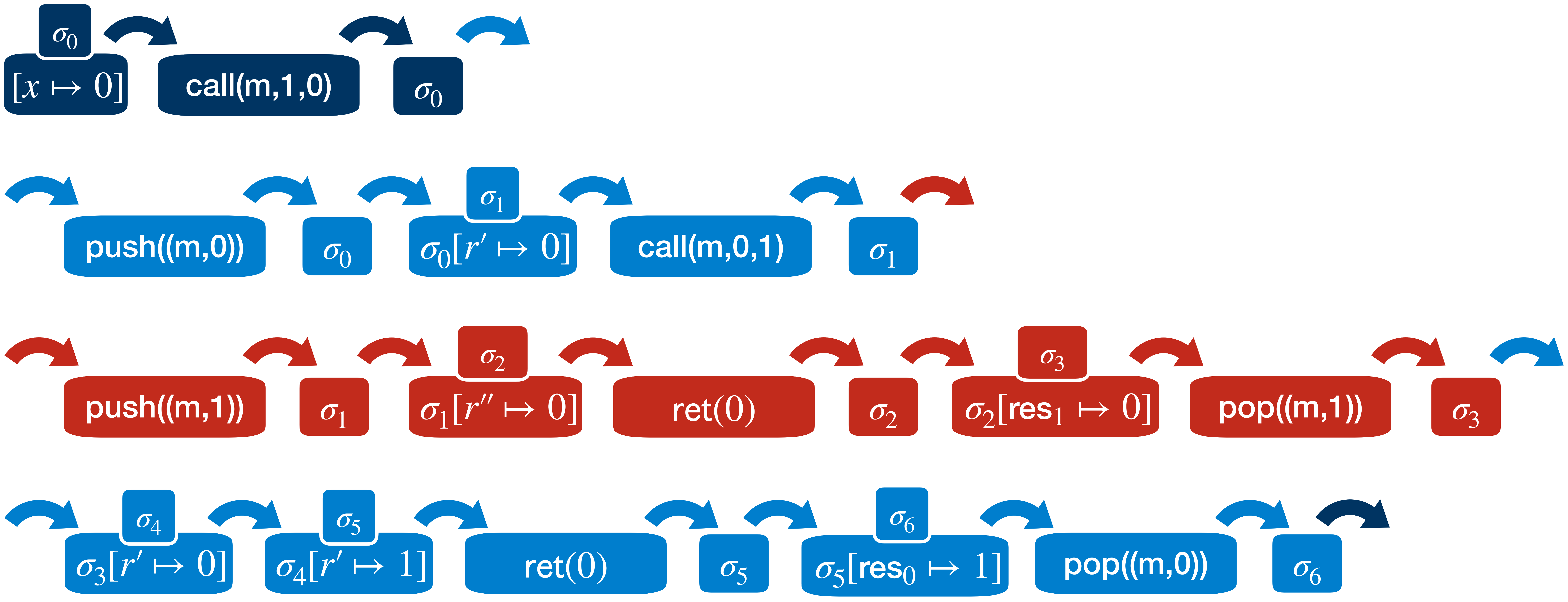
Context (m,0)

Initial Context



Trace Semantics

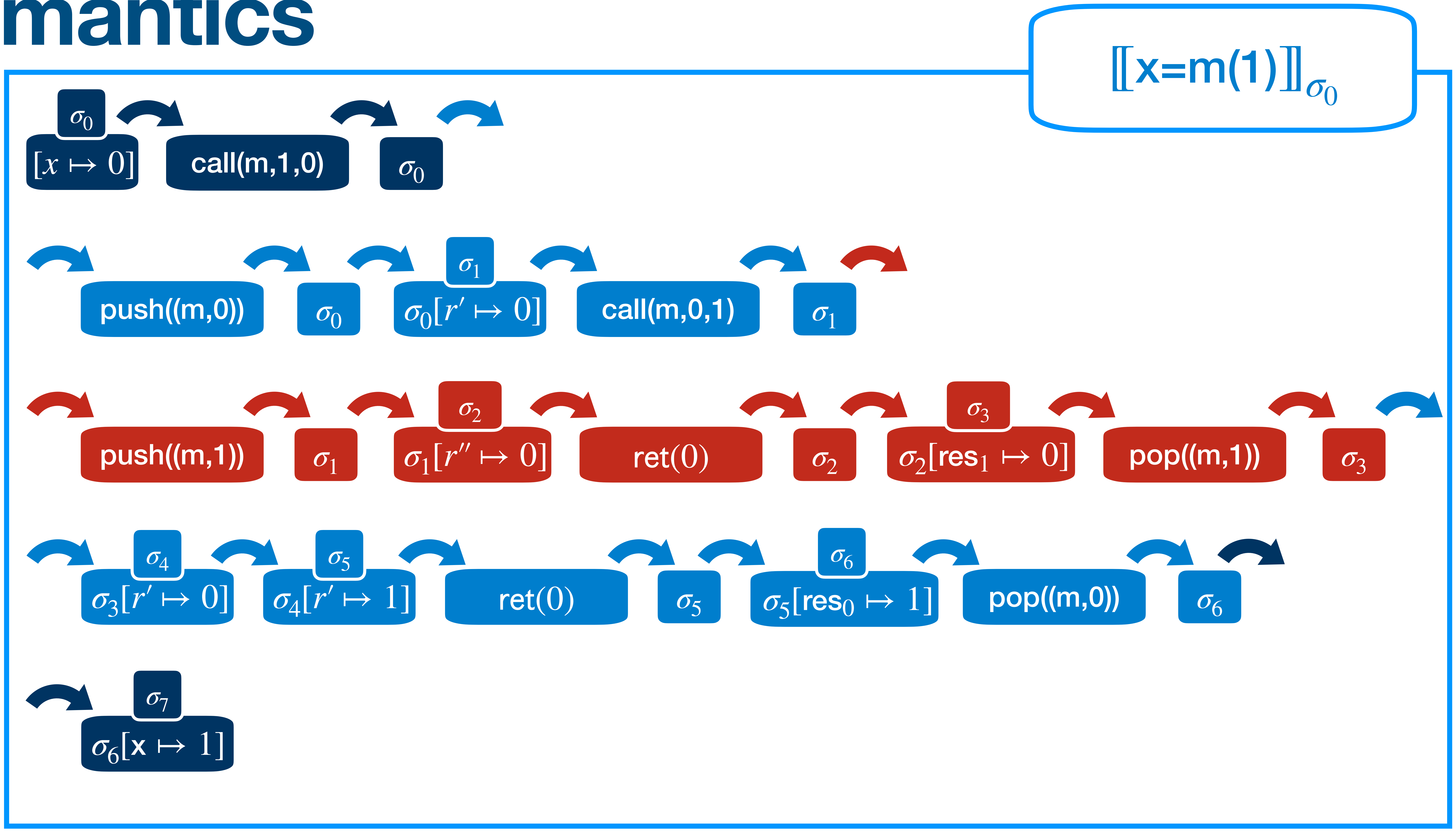
```
{
  x = m(1)
}

m(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```



Trace Semantics

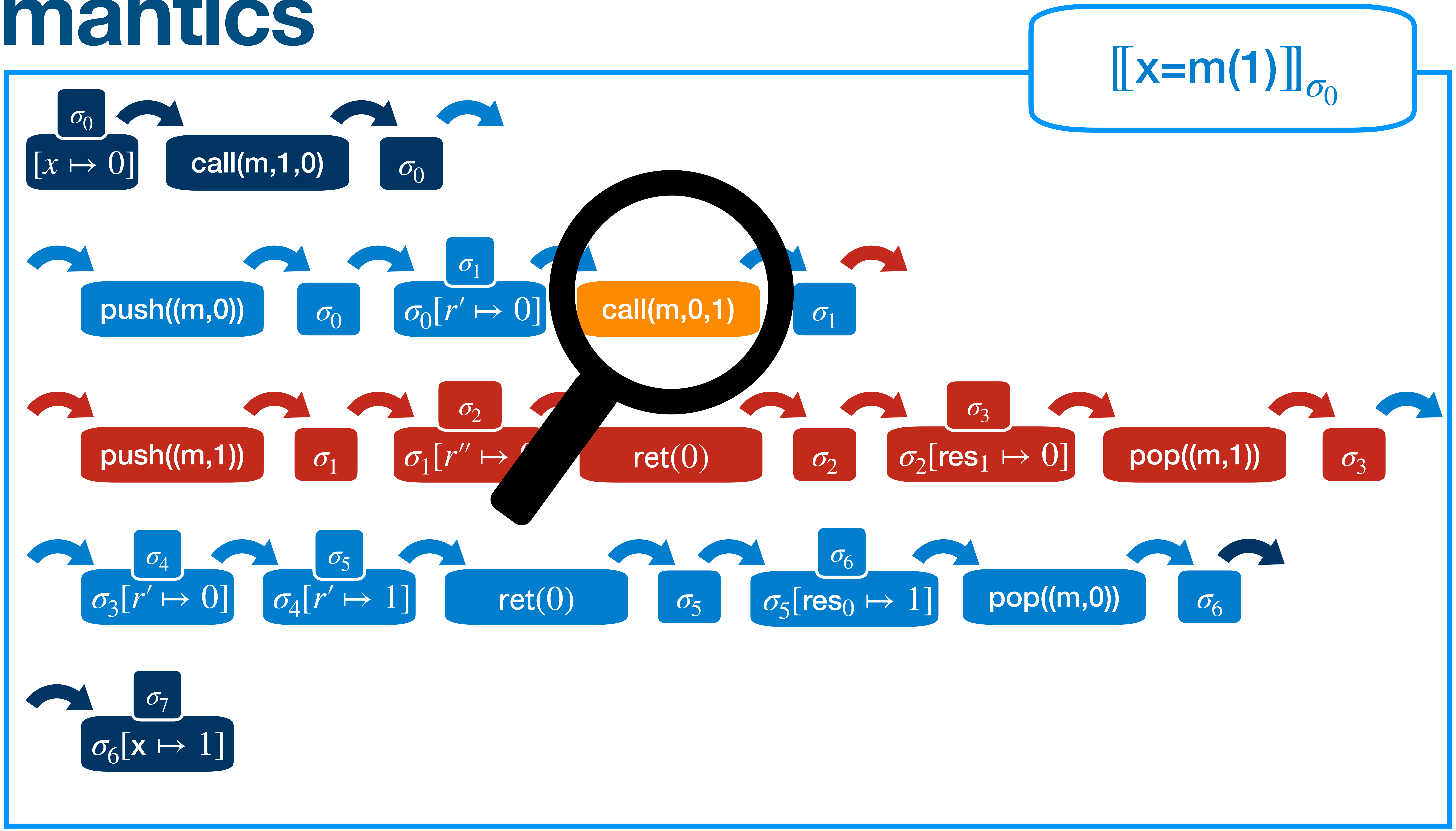
```
{
  x = m(1)
}

m(k) {
  r ; // initialized to 0
  if (k != 0) {
    r = m(k-1);
    r = r + 1
  };
  return r
}
```

Initial Context



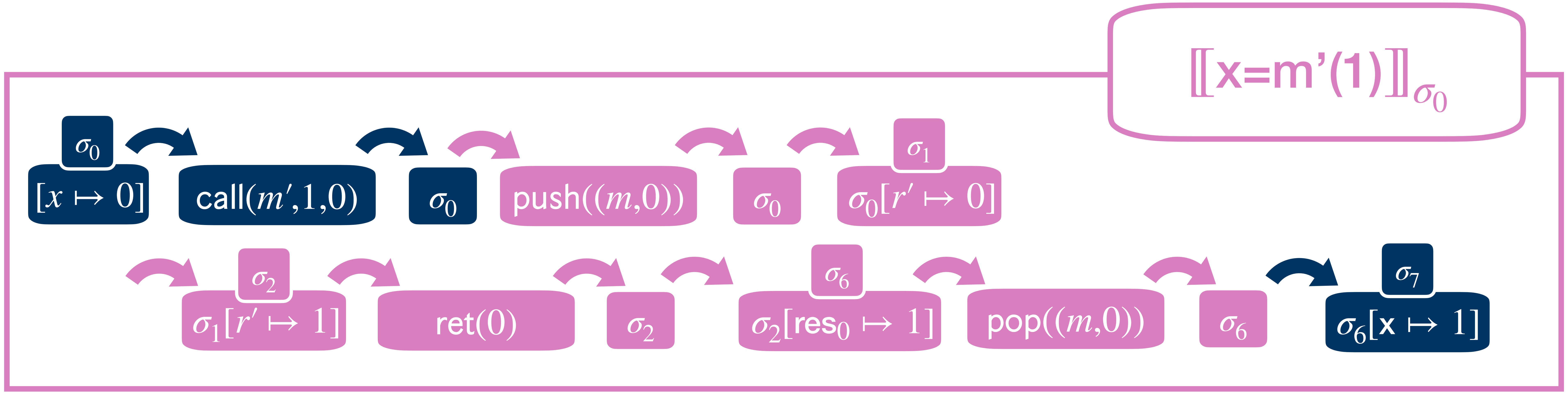
Trace Semantics

```
{
   x = m(1)
}

m(k) {
   r ; // initialized to 0
   if (k != 0) {
      r = m(k-1);
      r = r + 1
   };
   return r
}
```



Trace Semantics

```
{
    x = m'(1)
}

m'(k) {
    r ; // initialized to 0
    if (k != 0) {
        r = k-1;
        r = r + 1
    };
    return r
}
```



No procedure calls in m'

Part III

A Logic for Trace Contracts

A Logic for Trace Contracts

Syntax

$$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi ** \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$$

Trace formula

$$\Phi_1 = \lceil \mathsf{n} \doteq 0 \rceil$$

Semantics (set of traces)

$[\![\Phi_1]\!]$ = set of traces of length 1 with $\mathsf{n} \doteq 0$

Trace formula

$$\Phi_2 = \lceil \mathsf{n} \doteq 0 \rceil \cdot \lceil \mathsf{n} \doteq 1 \rceil = \Phi_1 \cdot \lceil \mathsf{n} \doteq 1 \rceil$$

Semantics (set of traces)

$$[\![\Phi_2]\!] = [\![\lceil \mathsf{n} \doteq 0 \rceil \cdot \lceil \mathsf{n} \doteq 1 \rceil]\!] = [\![\Phi_1]\!] \cdot [\![\lceil \mathsf{n} \doteq 1 \rceil]\!]$$

A Logic for Trace Contracts

Syntax

$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi{*}{*}\Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$

startEv(m,par), finish(m,res)

$\lceil \mathsf{true} \rceil \vee \mathsf{Ev}(\overline{\mathsf{m}})$

Trace formula

$\Psi = \mu X(\mathsf{Anything} \vee \mathsf{Anything} \cdot X)$

$\Psi^0 = \mathsf{Anything}$ — 0th unfold

$\Psi^1 = \mathsf{Anything} \cdot \Psi$ — 1st unfold

$\Psi^2 = \mathsf{Anything} \cdot \mathsf{Anything} \cdot \Psi$ — 2nd unfold

A Logic for Trace Contracts

Syntax

$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi ** \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$

$\lceil \mathsf{true} \rceil \vee \mathsf{Ev}(\overline{\mathsf{m}})$

Trace formula

$\Psi = \mu X(\mathsf{Anything} \vee \mathsf{Anything} \cdot X)$

Semantics
(set of traces)

$[\![\Psi]\!]$ = set of all finite traces

Any terminating program is <u>conforming to</u> Ψ

Its traces are in $[\![\Psi]\!]$

A Logic for Trace Contracts

Syntax

$$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi ** \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$$

$\lceil \mathsf{true} \rceil \vee \mathsf{Ev}(\overline{\mathsf{m}})$

Trace formula

$$\Psi = \mu X(\mathsf{Anything} \vee \mathsf{Anything} \cdot X)$$

Semantics (set of traces)

$[\![\Psi]\!]$ = set of all finite traces

$[\![\mathsf{v = m(n)}]\!]_\sigma \in [\![\Psi]\!]$

$[\![\mathsf{v = m'(n)}]\!]_\sigma \in [\![\Psi]\!]$

With $\sigma \vDash \mathsf{n} \geq 0$

Assignments with m and m' are conforming to Ψ

A Logic for Trace Contracts

Syntax

$$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi * * \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$$

Trace formula

$$\Psi = \mu X(\mathsf{noEv(m)} \vee \lceil \mathsf{noEv(m)} \rceil \cdot X)$$

Semantics (set of traces)

$[\![\Psi]\!] =$ set of all finite traces with no calls to m

A Logic for Trace Contracts

Syntax

$$\Phi ::= \lceil P \rceil \mid \mathsf{Ev} \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \cdot \Phi \mid \Phi {*}{*} \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$$

Syntactic Sugar

$$\underset{\cdot\,\cdot}{\overline{\mathsf{m}}} \equiv \mu X(\mathsf{noEv}(\overline{\mathsf{m}}) \vee \lceil \mathsf{noEv}(\overline{\mathsf{m}}) \rceil \cdot X)$$

Semantics (set of traces)

$[\![\underset{\cdot\,\cdot}{\overline{\mathsf{m}}}]\!] =$ set of all finite traces with no calls to $\overline{\mathsf{m}}$

Non conforming

$$[\![\mathsf{v = m(n)}]\!]_{\sigma} \notin [\![\underset{\cdot\,\cdot}{\mathsf{m}}]\!]$$

Calls to m generate events with m!

Conforming

With $\sigma' \models \mathsf{n} \geq 0$

$$[\![\mathsf{v = m'(n)}]\!]_{\sigma'} \in [\![\underset{\cdot\,\cdot}{\mathsf{m}}]\!]$$

m’ does contains calls to m

A Logic for Trace Contracts

Syntax

$$\Phi ::= \lceil P \rceil \mid \text{Ev} \mid \Phi \land \Phi \mid \Phi \lor \Phi \mid \Phi \cdot \Phi \mid \Phi ** \Phi \mid \mu X(\bar{y})(\bar{t}) \mid X(\bar{t})$$

Syntactic Sugar

$$\overset{\overline{\mathsf{m}}}{\because} \equiv \mu X(\text{noEv}(\overline{\mathsf{m}}) \lor \lceil \text{noEv}(\overline{\mathsf{m}}) \rceil \cdot X)$$

With $\sigma \models \mathsf{n} > 0$

$$[\![\mathsf{v} = \mathsf{m}(\mathsf{n})]\!]_\sigma \notin [\![\text{start}(\mathsf{m},\mathsf{n}) \overset{\mathsf{m}}{\because} \text{finish}(\mathsf{m},\mathsf{n})]\!]$$

Non conforming

m contains a call to m!

Judgments

Program — $s : \Phi$ — Trace Formula

s is conforming to Φ

$$\sigma \models s : \Phi \iff [\![s]\!]_\sigma \in [\![\Phi]\!]$$

Judgments

Program — $s : \Phi$ — Trace Formula

s is conforming to Φ

$\sigma \models s : \Phi \iff [\![s]\!]_\sigma \in [\![\Phi]\!]$

$\sigma \models$ v = m(n) : $\cdot\ \cdot$

$\sigma \models$ n ≥ 0

$\sigma \not\models$ v = m(n) : $\overset{\text{m}}{\cdot\ \cdot}$

$\sigma' \models$ n > 0

$\sigma' \not\models$ v = m(n) : start(m,n) $\overset{\text{m}}{\cdot\ \cdot}$ finish(m,n)

$\sigma \models$ v = m'(n) : $\cdot\ \cdot$

$\sigma \models$ v = m'(n) : $\overset{\text{m}}{\cdot\ \cdot}$

$\sigma \models$ n ≥ 0

Part IV

A Calculus for Deductive Verification

Symbolic Execution with Trace Updates

$$\mathcal{U} ::= \epsilon \mid \{\mathsf{v} := \mathsf{e}\}\mathcal{U} \mid \{\mathsf{Ev}(\overline{\mathsf{e}})\}\mathcal{U}$$

Sequent

$$\Gamma \vdash \mathcal{U}\mathsf{s} : \Phi$$

Judgement with leading trace update

$$\Gamma \vdash \mathsf{s} : \Phi$$

Symbolic Execution

$$\Gamma' \vdash \mathcal{U}_{\mathsf{s}} : \Phi$$

Example

No procedure calls:
No need for contracts!

Symbolic Execution of m'

$$\textbf{Return}\ \dfrac{k > 0 \vdash \mathcal{U}\{r := k - 1\}\{r := r{+}1\}\{\text{finishEv}(m', r)\}\{\text{res} := r\} : \Phi}{\textbf{Assign}\ \dfrac{k > 0 \vdash \mathcal{U}\{r := k{-}1\}\{r := r{+}1\}\ \textbf{return}\ r : \Phi}{\textbf{Assign}\ \dfrac{k > 0 \vdash \mathcal{U}\{r := k{-}1\} r{=}r{+}1;\ \textbf{return}\ r : \Phi}{\textbf{Cond}\ \dfrac{k > 0 \vdash \mathcal{U}\ r{=}k{-}1;\ r{=}r{+}1;\ \textbf{return}\ r : \Phi}{k > 0 \vdash \mathcal{U}\ \textbf{if}\ (k!{=}0)\ \{r{=}k{-}1;\ r{=}r{+}1;\}\ \textbf{return}\ r : \Phi}}}}$$

Leading Update
$\mathcal{U} \equiv \{\text{startEv}(m', k)\}\{r := 0\}$

Body of m'

Example

A procedure call:
We need to use the contract of m!

Symbolic Execution of m

Return

$$k > 0 \vdash \mathcal{U}\{r{:=}m(k-1)\}\{r := k+1\}\{finishEv(m, r)\}\{res{:=} r\} : \Phi$$

Assign

$$k > 0 \vdash \mathcal{U}\{r{:=}m(k-1)\}\{r := r+1\}\ \textbf{return}\ r : \Phi$$

Assign

$$k > 0 \vdash \mathcal{U}\{r{:=}m(k-1)\}\ r{=}r{+}1;\ \textbf{return}\ r : \Phi$$

Cond

$$k > 0 \vdash \mathcal{U}\ r{=}m(k-1);\ r{=}r{+}1;\ \textbf{return}\ r : \Phi$$

$$k > 0 \vdash \mathcal{U}\ \textbf{if}\ (k!{=}0)\ \{r{=}m(k-1);\ r{=}r{+}1;\}\ \textbf{return}\ r : \Phi$$

Leading Update
$\mathcal{U} \equiv \{startEv(m, k)\}\{r := 0\}$

Body of m

State- vs Trace-based Contracts

$pre : \mathsf{n} \geq 0$

$post : \mathsf{res} \doteq \mathsf{n}$

State-based

$pre : \mathsf{n} \geq 0$

$post : \mathsf{res} \doteq \mathsf{n}$

$trace : \cdots$

Trace-based

Fulfilled by the same programs!

State- vs Trace-based Contracts

$pre : \mathsf{n} \geq 0$

$post : \mathsf{res} \doteq \mathsf{n}$

State-based

Fulfilled by m and m’

$pre : \mathsf{n} \geq 0$

$post : \mathsf{res} \doteq \mathsf{n}$

$trace : \overset{\mathsf{m}}{\cdot\;\cdot}$

Trace-based

Not fulfilled by m

Trace-based Contracts

pre : $\mathsf{n} \geq 0$

$post$: $\mathsf{res} \doteq \mathsf{n}$

$trace$: Φ_{m}

Contract of m

How does Φ_{m} look like?

Trace-based Contracts

$$\Phi_{\mathrm{m}} = \mu X_{\mathrm{m}}(\mathrm{k}) \,.\, (\lceil \mathrm{k} \doteq 0 \rceil \ast\ast\ \mathrm{start(m,0)} \overset{\mathrm{m}}{\cdot\cdot}\ \mathrm{finish(m,0)} \ast\ast \lceil \mathrm{res} \doteq 0 \rceil) \vee$$

$$(\lceil \mathrm{k} > 0 \rceil \ast\ast\ \mathrm{start(m,k)} \overset{\mathrm{m}}{\cdot\cdot} X_{\mathrm{m}}(\mathrm{k\text{-}1}) \overset{\mathrm{m}}{\cdot\cdot}\ \mathrm{finish(m,k)}\ \lceil \mathrm{res} \doteq \mathrm{k} \rceil)$$

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

pre : n $\geq$ 0

post : res $\doteq$ n

trace : Φ_{m}

Contract of m

Trace-based Contracts

$$\Phi_{m} = \mu X_{m}(k) \,.\, (\lceil k \doteq 0 \rceil ** \ \text{start}(m,0) \overset{m}{\because} \text{finish}(m,0) ** \lceil \text{res} \doteq 0 \rceil) \vee$$
$$(\lceil k > 0 \rceil ** \ \text{start}(m,k) \overset{m}{\because} X_{m}(k\text{-}1) \overset{m}{\because} \text{finish}(m,k) \ \lceil \text{res} \doteq k \rceil)$$

Condition is false

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

pre : $n \geq 0$

post : $\text{res} \doteq n$

trace : Φ_{m}

Contract of m

Trace-based Contracts

$$\Phi_m = \mu X_m(k)\ .\ (\lceil k \doteq 0 \rceil ** \ \mathrm{start(m,0)} \overset{m}{\because} \mathrm{finish(m,0)} \ ** \lceil \mathrm{res} \doteq 0 \rceil) \vee$$

$$(\lceil k > 0 \rceil ** \ \mathrm{start(m,k)} \overset{m}{\because} X_m(k\text{-}1) \overset{m}{\because} \mathrm{finish(m,k)} \ \lceil \mathrm{res} \doteq k \rceil)$$

Condition is false

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

No calls to m.
Contract not needed

pre : n ≥ 0

post : res ≐ n

trace : Φ_m

Contract of m

Trace-based Contracts

$$\Phi_{m} = \mu X_{m}(k) \,.\, (\lceil k \doteq 0 \rceil ** \ start(m,0) \overset{m}{\cdot\,\cdot} finish(m,0) ** \lceil res \doteq 0 \rceil) \vee$$

$$(\lceil k > 0 \rceil ** \ start(m,k) \overset{m}{\cdot\,\cdot} X_{m}(k-1) \overset{m}{\cdot\,\cdot} finish(m,k) \ \lceil res \doteq k \rceil)$$

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

Condition is false

$r \doteq 0$

pre : $n \geq 0$

$post$: $res \doteq n$

$trace$: Φ_{m}

Contract of m

Trace-based Contracts

$$\Phi_m = \mu X_m(k) \,.\, (\lceil k \doteq 0 \rceil ** \text{ start}(m,0) \overset{m}{\cdot\cdot} \text{ finish}(m,0) ** \lceil res \doteq 0 \rceil) \vee$$

$$(\lceil k > 0 \rceil ** \text{ start}(m,k) \overset{m}{\cdot\cdot} X_m(k-1) \overset{m}{\cdot\cdot} \text{ finish}(m,k) \lceil res \doteq k \rceil)$$

Condition is true

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

pre : $n \geq 0$

post : $res \doteq n$

trace : Φ_m

Contract of m

Trace-based Contracts

$$\Phi_m = \mu X_m(k)\,.\,(\lceil k \doteq 0 \rceil ** \text{ start(m,0)} \overset{m}{\cdot\cdot} \text{finish(m,0)} ** \lceil res \doteq 0 \rceil) \vee$$

$$(\lceil k > 0 \rceil ** \text{ start(m,k)} \overset{m}{\cdot\cdot} X_m(k-1) \overset{m}{\cdot\cdot} \text{finish(m,k)}\ \lceil res \doteq k \rceil)$$

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

Condition is true

$r \doteq k\text{-}1$

m(k-1) conforming to Φ_m(k-1)

Matching m(k-1) against X_m(k-1)

pre : $n \geq 0$

post : $res \doteq n$

trace : Φ_m

Contract of m

Trace-based Contracts

$$\Phi_m = \mu X_m(k) \,.\, (\lceil k \doteq 0 \rceil ** \text{ start}(m,0) \overset{m}{\cdot\cdot} \text{ finish}(m,0) ** \lceil res \doteq 0 \rceil) \vee$$

$$(\lceil k > 0 \rceil ** \text{ start}(m,k) \overset{m}{\cdot\cdot} X_m(k-1) \overset{m}{\cdot\cdot} \text{ finish}(m,k) \lceil res \doteq k \rceil)$$

```
m(k) {
    r ; // initialised to 0
    if (k != 0) {
        r = m(k-1);
        r = r + 1
    };
    return r
}
```

Condition is true

$r \doteq k$

$r \doteq k-1$

m(k-1) conforming to Φ_m(k-1)

Matching m(k-1) against X_m(k-1)

pre : $n \geq 0$

post : $res \doteq n$

trace : Φ_m

Contract of m

Trace Abstraction

TrAbs

$$\frac{\Gamma \vdash \mathscr{U}_1 : \Phi_1 \qquad \Gamma \vdash \mathscr{U}_1(\mathrm{pre_m(e)}) : \Phi_1 \qquad \mathbf{C}_m \vdash \{\mathrm{v{:=}f_m}(\mathscr{U}_1(\mathrm{e}))\}\mathscr{U}_2 : \Phi_2}{\Gamma, \mathbf{C}_m \vdash \mathscr{U}_1 \{\mathrm{v{:=}m(e)}\}\ \mathscr{U}_2 : \Phi_1 ** \Phi_m(\mathrm{e}) ** \Phi_2}$$

Trace Abstraction

TrAbs

$$\frac{\Gamma \vdash \mathcal{U}_1 : \Phi_1 \qquad \Gamma \vdash \mathcal{U}_1(\mathrm{pre}_m(e)) : \Phi_1 \qquad \mathbf{C}_m \vdash \{v:=f_m(\mathcal{U}_1(e))\}\mathcal{U}_2 : \Phi_2}{\Gamma, \mathbf{C}_m \vdash \mathcal{U}_1 \boxed{\{v:=m(e)\}}\ \mathcal{U}_2 : \Phi_1 ** \boxed{\Phi_m(e)} ** \Phi_2}$$

Unfold

$$\cfrac{\cfrac{\cfrac{k > 0, \mathbf{C}_m \vdash \mathcal{U}_1 \boxed{\{r:=m(k-1)\}}\ \mathcal{U}_2 : \Phi_1 ** \boxed{\Phi_m(k\text{-}1)} ** \Phi_2}{k > 0, \mathbf{C}_m \vdash \mathcal{U}_1\{r:=m(k-1)\}\ \mathcal{U}_2 : \Phi_m(k)}}{\vdots}}{k > 0 \vdash \mathcal{U}_1\ \textbf{if}\ (k!=0)\ \{r=m(k-1);\ r=r+1;\}\ \textbf{return}\ r : \Phi_m}$$

$\mathcal{U}_1 \equiv \{\mathrm{startEv}(m, k)\}\{r := 0\}\{r := k-1\}$

$\mathcal{U}_2 \equiv \{r := r+1\}\{\mathrm{finish}(m, r)\}\{res := r\}$

$\Phi_1 \equiv \lceil k > 0 \rceil ** \mathrm{start}(m,k) \overset{m}{\cdot\cdot}$

$\Phi_2 \equiv \overset{m}{\cdot\cdot}\ \mathrm{finish}(m,k)\ \lceil res \doteq k \rceil$

Trace Abstraction

$$\textbf{TrAbs}\ \frac{\Gamma \vdash \mathcal{U}_1 : \Phi_1 \quad \Gamma \vdash \mathcal{U}_1(\mathrm{pre}_m(e)) : \Phi_1 \quad \mathbf{C}_m \vdash \{v{:=}f_m(\mathcal{U}_1(e))\}\mathcal{U}_2 : \Phi_2}{\Gamma, \mathbf{C}_m \vdash \mathcal{U}_1\,\{v{:=}m(e)\}\,\mathcal{U}_2 : \Phi_1 ** \Phi_m(e) ** \Phi_2}$$

$$\textbf{TrAbs}\ \frac{k>0 \vdash \mathcal{U}_1 : \Phi_1 \quad k>0 \vdash k \geq 0 \quad \mathbf{C}_m \vdash \{v{:=}k\text{-}1)\}\mathcal{U}_2 : \Phi_2}{\textbf{Unfold}\ \dfrac{k>0, \mathbf{C}_m \vdash \mathcal{U}_1\,\{r{:=}m(k-1)\}\,\mathcal{U}_2 : \Phi_1 ** \Phi_m(k\text{-}1) ** \Phi_2}{\dfrac{k>0, \mathbf{C}_m \vdash \mathcal{U}_1\{r{:=}m(k-1)\}\,\mathcal{U}_2 : \Phi_m(k)}{\dfrac{\vdots}{k>0 \vdash \mathcal{U}_1\ \textbf{if}\ (k!{=}0)\ \{r{=}m(k-1);\ r{=}r{+}1;\}\ \textbf{return}\ r : \Phi_m}}}}$$

$\mathcal{U}_1 \equiv \{\mathrm{startEv}(m, k)\}\{r := 0\}\{r := k-1\}$

$\Phi_1 \equiv \lceil k > 0 \rceil ** \mathrm{start}(m,k) \overset{m}{\because}$

$\mathcal{U}_2 \equiv \{r := r+1\}\{\mathrm{finish}(m, r)\}\{res := r\}$

$\Phi_2 \equiv \overset{m}{\because} \mathrm{finish}(m,k)\ \lceil res \doteq k \rceil$

Closing the Proof

Sound Calculus!

$$\frac{k > 0 \vdash \{startEv(m, k)\}\{r := 0\}\{r := k-1\} : \lceil k > 0 \rceil ** \ start(m,k) \ \overset{m}{\because}}{(a)}$$

$$\frac{\vdash \{r:=k\text{-}1)\}\{r := r+1\}\{finish(m, r)\}\{res := r\} : \overset{m}{\because} \ finish(m,k) \ \lceil res \doteq k \rceil}{(b)}$$

$$\textbf{Unfold}\ \dfrac{\textbf{TrAbs}\ \dfrac{\dfrac{(a)}{k > 0 \vdash \mathcal{U}_1 : \Phi_1} \quad k > 0 \vdash k \geq 0 \quad \dfrac{(b)}{\mathbf{C}_m \vdash \{r:=k\text{-}1)\}\mathcal{U}_2 : \Phi_2}}{k > 0, \mathbf{C}_m \vdash \mathcal{U}_1\{r:=m(k-1)\}\ \mathcal{U}_2 : \Phi_1 ** \Phi_m(k\text{-}1) ** \Phi_2}}{k > 0, \mathbf{C}_m \vdash \mathcal{U}_1\{r:=m(k-1)\}\ \mathcal{U}_2 : \Phi_m(k)}$$

$$\vdots$$

$$k > 0 \vdash \mathcal{U}_1\ \textbf{if}\ (k!=0)\ \{r=m(k-1);\ r=r+1;\}\ \textbf{return}\ r : \Phi_m$$

Conclusion

Part I
State-based Contracts

Part II
Trace Semantics

Part III
A Logic for Trace Contracts

Part IV
A Calculus for Deductive Verification

Conclusion

Part I
State-based Contracts

Modular Verification of Recursive Procedures

Used by state-of-the-art deductive verifiers

What happens during execution? (Intermediate Annotations)

Low readability

No independence of the code

Conclusion

Part II
Trace Semantics

For sequential programs with recursive calls

Events to capture properties of the execution

Conclusion

Part III
A Logic for Trace Contracts

Specifying properties over finite program traces

Trace contracts

Judgments to express correctness of trace contracts

Conclusion

Part IV
Deductive Verification

Sound Calculus: Verifying trace contracts

Symbolic Execution & Trace Abstraction

Modular Verification!

Conclusion

Part I
State-based Contracts
And their limitations

Part II
Trace Semantics
Capturing what happens during execution

Part III
A Logic for Trace Contracts
Specifying accepted behaviours

Part IV
Deductive Verification
Verifying Trace Contracts