Game semantics for good general references

Andrzej Murawski
University of Leicester

Nikos Tzevelekos
University of Oxford

What this talk is about

We present a fully abstract and directly presentable denotational model of higher-order references

The model is constructed using nominal game semantics

ML-like references

- Creation

```
ref(M)
```

- Assignment

```
M := N
```

- Dereferencing

```
!M
```

- Equality check

```
M == N
```

OCaml session – ground references

```
# let r0 = ref(());;
val r0 : unit ref = { contents = () }

# let r1 = ref(3);;
val r1 : int ref = { contents = 3 }
```

OCaml session – ground references

```
# let r0 = ref(());;
val r0 : unit ref = { contents = () }

# let r1 = ref(3);;
val r1 : int ref = { contents = 3 }

# let r2 = ref(r1);;
val r2 : int ref ref =
           { contents = { contents = 3 } }
```

More Ocaml – higher-order references

```
# let r3 = ref(fun(x:intref) -> x == r1);;
val r3 : (int ref -> bool) ref =
                        { contents = <fun> }
```

More Ocaml – higher-order references

```
# let r3 = ref(fun(x:intref) -> x == r1);;
val r3 : (int ref -> bool) ref =
                    { contents = <fun> }

# (!r3)(r1);;
- : bool = true

# (!r3)(ref(3));;
- : bool = false
```

RefML: a language with HO-references

θ ::= unit | int | ref θ | $\theta \to \theta$

RefML: a language with HO-references

$$\theta \;::=\; \mathtt{unit} \mid \mathtt{int} \mid \mathtt{ref}\,\theta \mid \theta \to \theta$$

$$\frac{(x:\theta)\in\Gamma}{u,\Gamma\vdash x:\theta} \qquad \frac{}{u,\Gamma\vdash():\mathsf{unit}} \qquad \frac{i\in\mathbb{Z}}{u,\Gamma\vdash i:\mathsf{int}} \qquad \frac{a\in(u\cap\mathcal{N}_\theta)}{u,\Gamma\vdash a:\mathsf{ref}\,\theta}$$

$$\frac{u,\Gamma\vdash M:\mathsf{ref}\,\theta}{u,\Gamma\vdash !M:\theta} \qquad \frac{u,\Gamma\vdash M:\mathsf{ref}\,\theta \quad u,\Gamma\vdash N:\theta}{u,\Gamma\vdash M:=N:\mathsf{unit}}$$

$$\frac{u,\Gamma\vdash M:\theta}{u,\Gamma\vdash \mathsf{ref}(M):\mathsf{ref}\,\theta} \qquad \frac{u,\Gamma\vdash M:\mathsf{ref}\,\theta \quad u,\Gamma\vdash N:\mathsf{ref}\,\theta}{u,\Gamma\vdash M=N:\mathsf{int}}$$

$$\frac{u,\Gamma\vdash M:\theta\to\theta' \quad u,\Gamma\vdash N:\theta}{u,\Gamma\vdash MN:\theta'} \qquad \frac{u,\Gamma\cup\{x:\theta\}\vdash M:\theta'}{u,\Gamma\vdash \lambda x^{\theta}.M:\theta\to\theta'}$$

$$\frac{u,\Gamma\vdash M_1,M_2:\mathsf{int}}{u,\Gamma\vdash M_1\oplus M_2:\mathsf{int}} \qquad \frac{u,\Gamma\vdash M:\mathsf{int} \quad u,\Gamma\vdash N_0,N_1:\theta}{u,\Gamma\vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_0:\theta}$$

Expressivity

- Divergence

```
let x = ref_{unit→unit}(...) in
        x := (λy^unit.(!x)y); (!x)()
```

- Recursion

```
λf^{θ→θ'}. let y = ref_{θ→θ'}(...) in
           y := (λz^θ.f(!y)z); (!y)
```

- Objects, Aspects, ...

Semantics

- Call-by-value evaluation: $M \Downarrow V$
- Terms M_1, M_2 are equivalent if, for all program contexts $C[\_]$, $C[M_1]\Downarrow$ iff $C[M_2]\Downarrow$

Semantics

- Call-by-value evaluation: $M \Downarrow V$
- Terms M_1, M_2 are equivalent if, for all program contexts $C[\_]$, $C[M_1]\Downarrow$ iff $C[M_2]\Downarrow$
- Full abstraction:

$$M_1 \cong M_2 \quad \text{iff} \quad [\![M_1]\!] = [\![M_2]\!]$$

Examples

$$x{:=}V;\ !x \cong x{:=}V;\ V$$

$$x{:=}V;\ x{:=}W \cong x{:=}W$$

$$x{:=}!x \cong ()$$

Examples

$$x{:=}V;\ !x \cong x{:=}V;\ V$$

$$x{:=}V;\ x{:=}W \cong x{:=}W$$

$$x{:=}!x \cong ()$$

$$\texttt{let } x=\texttt{ref}(0) \texttt{ in } \lambda y.(x=y) \not\cong \lambda y.0$$

$$\texttt{let } x=\texttt{ref}(0) \texttt{ in } \lambda y.\texttt{if } !x \texttt{ then } () \texttt{ else } (x{:=}1;\ f())$$

$$\not\cong$$

$$\texttt{let } x=\texttt{ref}(0) \texttt{ in } \lambda y.\texttt{if } !x \texttt{ then } () \texttt{ else } (f();\ x{:=}1)$$

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\mathtt{ref}\theta$
$= (\mathtt{unit} \to \theta) \times (\theta \to \mathtt{unit})$

$\longmapsto \quad (\mathbf{1} \to |[\theta]|) \times (|[\theta]| \to \mathbf{1})$

- Theoretically attractive
- but: *bad variables*, failure of full abstraction, ...

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\mathtt{ref}\theta = (\mathtt{unit} \to \theta) \times (\theta \to \mathtt{unit})$

$\longmapsto (\mathbf{1} \to |[\theta]|) \times (|[\theta]| \to \mathbf{1})$

- Theoretically attractive
- but: *bad variables*, failure of full abstraction, ...

Pairs of ref type are *not* always references

- e.g. `mkvar(λx.3,λx.())`

No notion of *ref-equality*

Spurious non-equivalences:

`x:=!x` vs. `()`

`x:=V; !x` vs. `x:=V; V`

`x:=V; x:=W` vs. `x:=W`

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\mathtt{ref}\theta$
$= (\mathtt{unit} \to \theta) \times (\theta \to \mathtt{unit})$

$\longmapsto (\mathbf{1} \to |[\theta]|) \times (|[\theta]| \to \mathbf{1})$

- Theoretically attractive
- but: *bad variables*, failure of full abstraction, ...

Pitts & Stark
- *nu-calculus (1993)*

References are *names:*

$\mathtt{ref}\ \theta = \mathtt{base\ type}$

$\longmapsto \mathcal{N}_\theta$ (names)

- Notion of *resource (name):*
  - atomic values
  - infinitely many
  - comparable for equality

Full abstraction for HO-references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\mathtt{ref}\theta$
$= (\mathtt{unit} \to \theta) \times (\theta \to \mathtt{unit})$

$\longmapsto (\mathbf{1} \to |[\theta]|) \times (|[\theta]| \to \mathbf{1})$

Abramsky, Honda, McCusker [LICS'98]
- HO-references vs. visibility

Pitts & Stark
- *nu-calculus (1993)*

References are *names:*

$\mathtt{ref}\ \theta = \mathtt{base\ type}$

$\longmapsto \mathcal{N}_\theta$ (names)

Tz. [LICS'07]
- Not directly presentable

$M_1 \cong M_2$ iff $[\![M_1]\!] \cong [\![M_2]\!]$

Our model

Built using game semantics

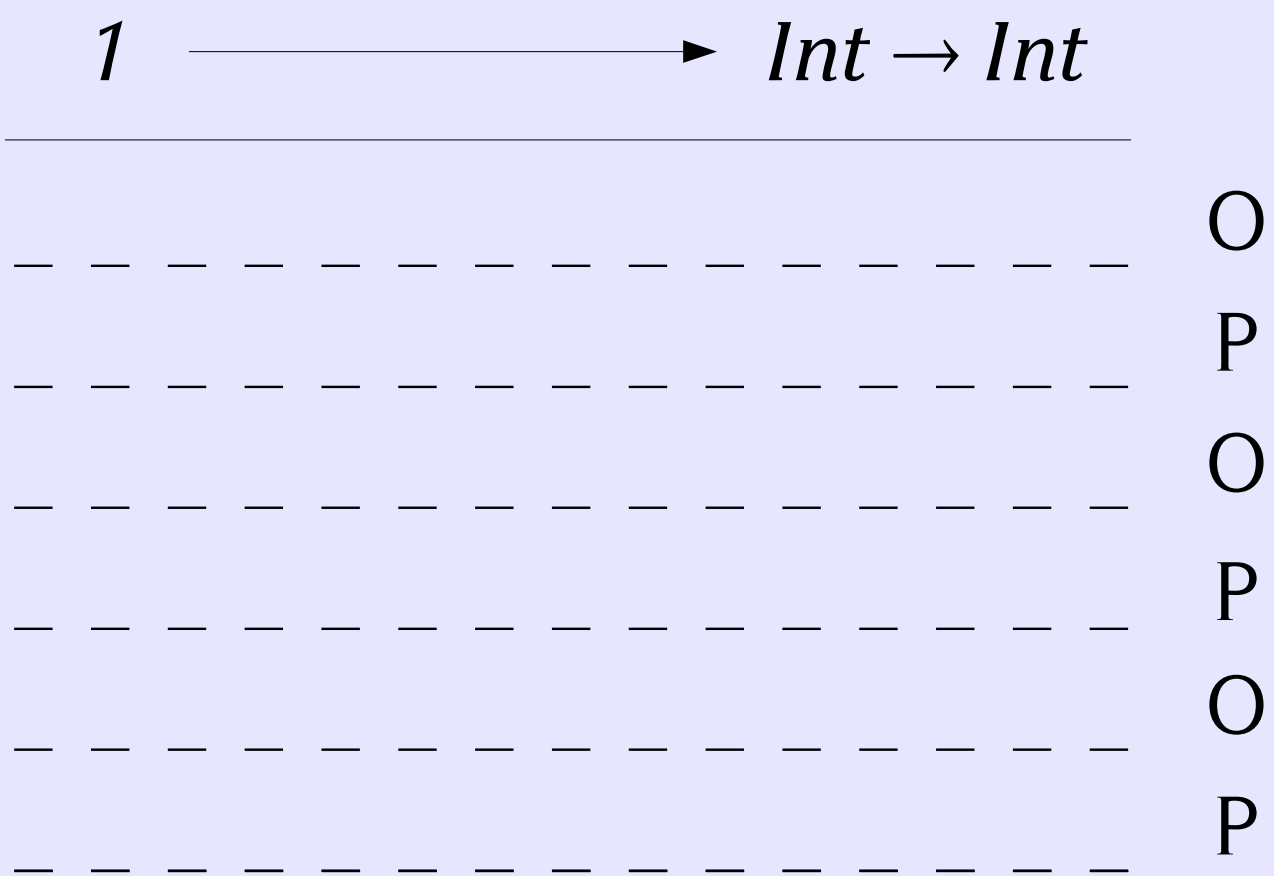
Uses a built-in notion of names and games which carry the store around

Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)

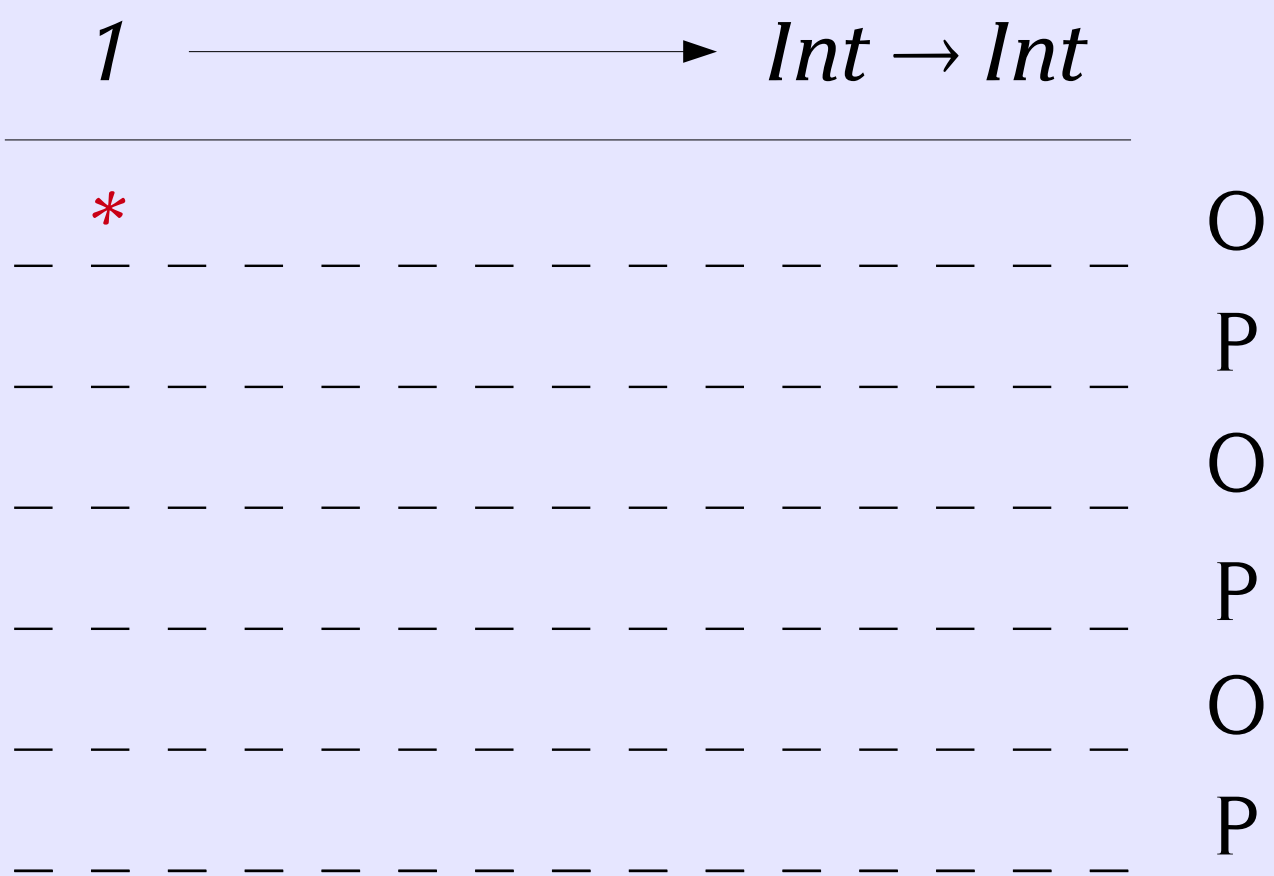
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



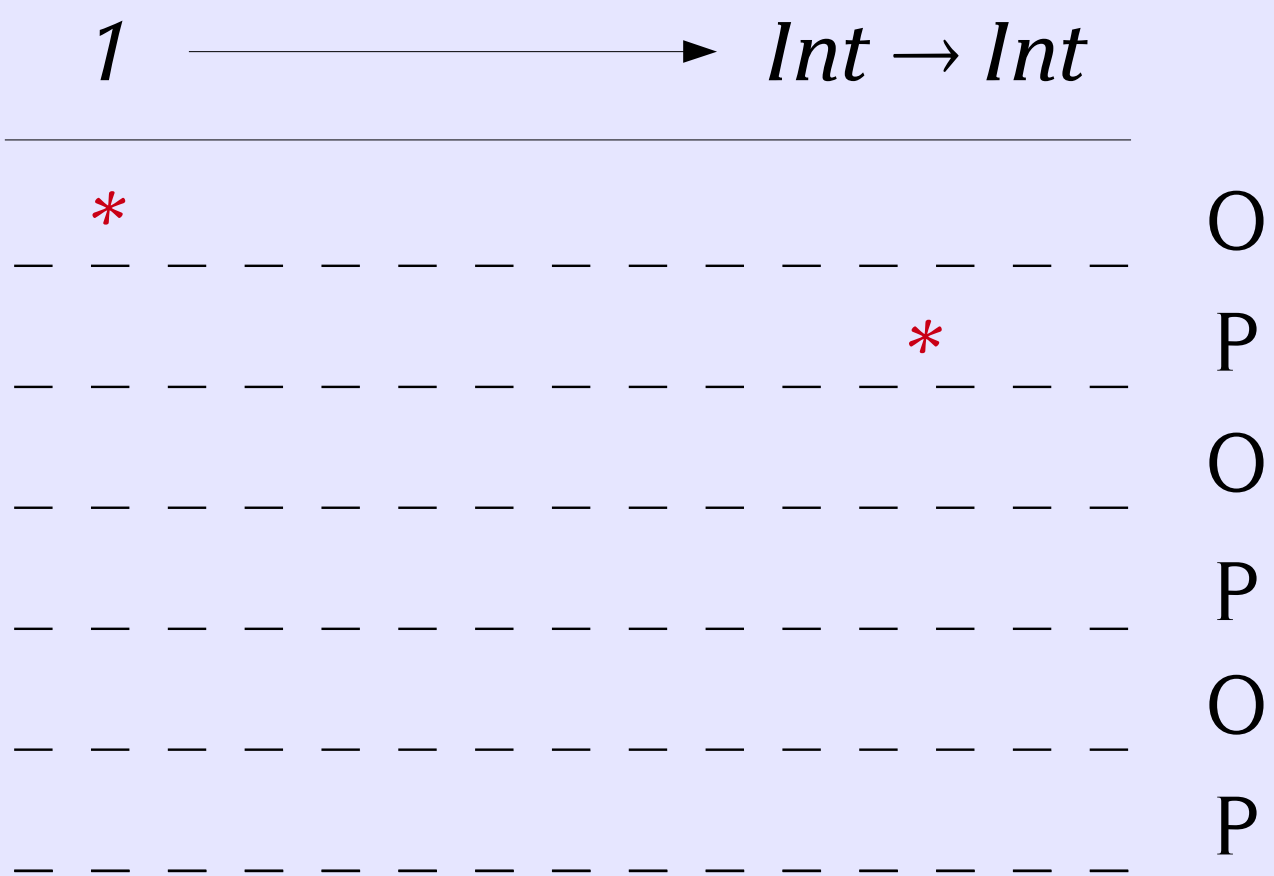
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



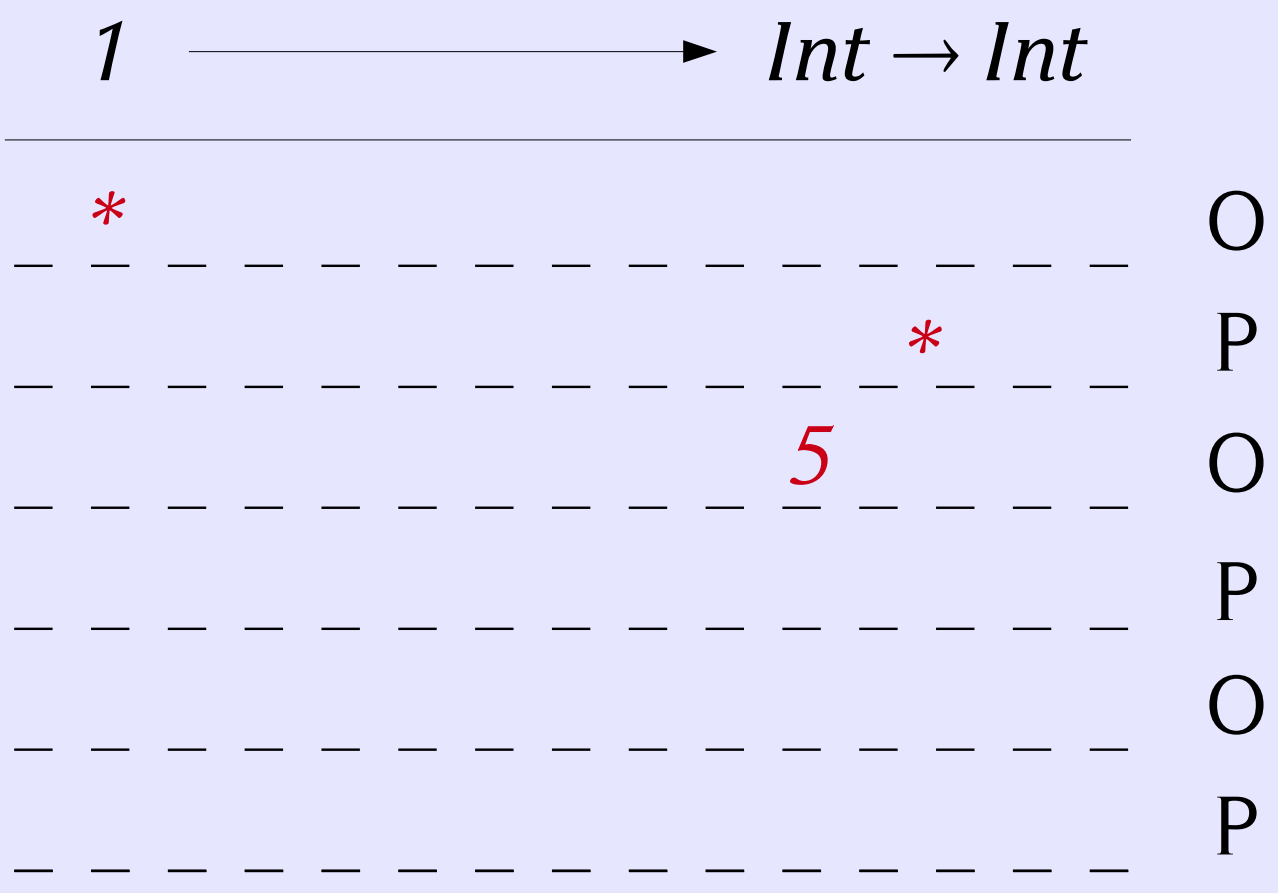
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



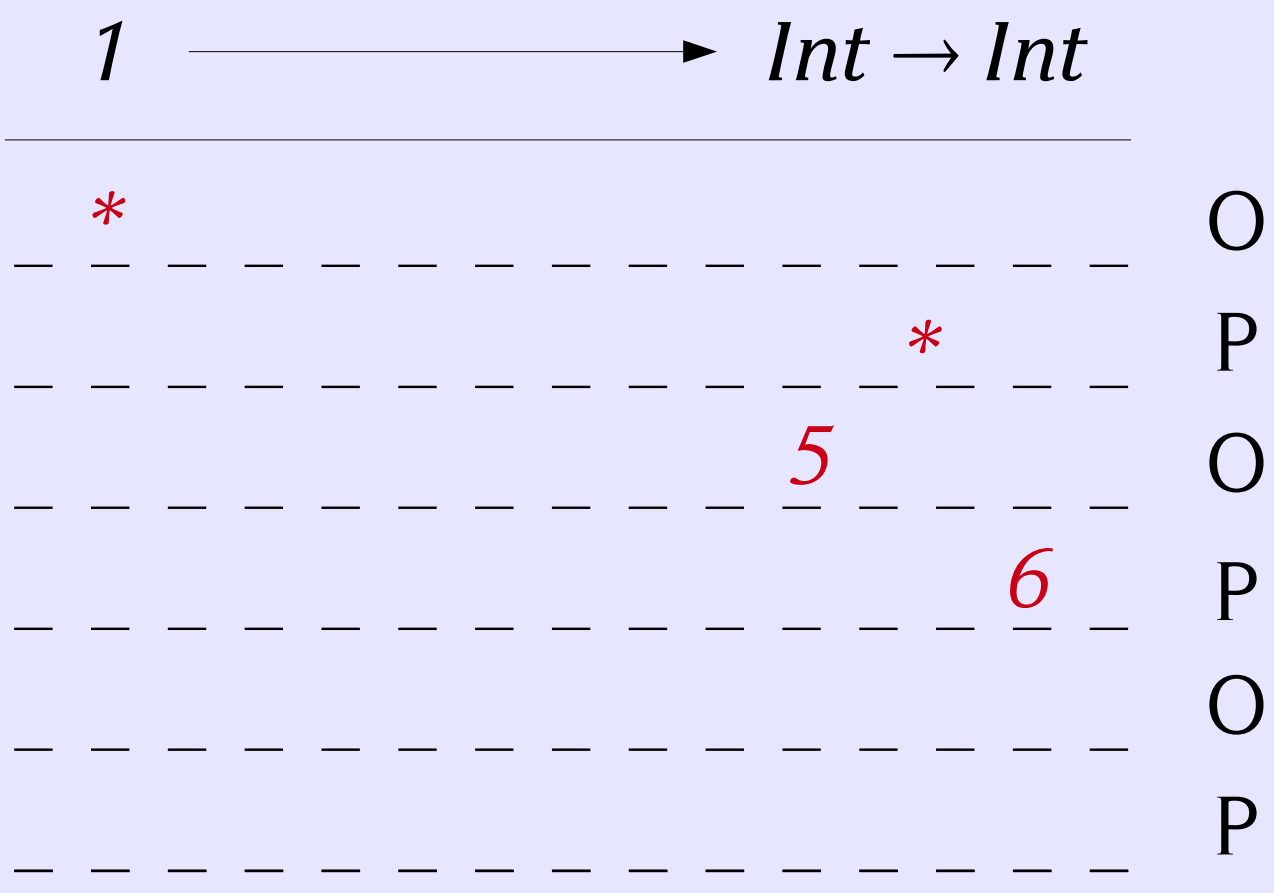
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



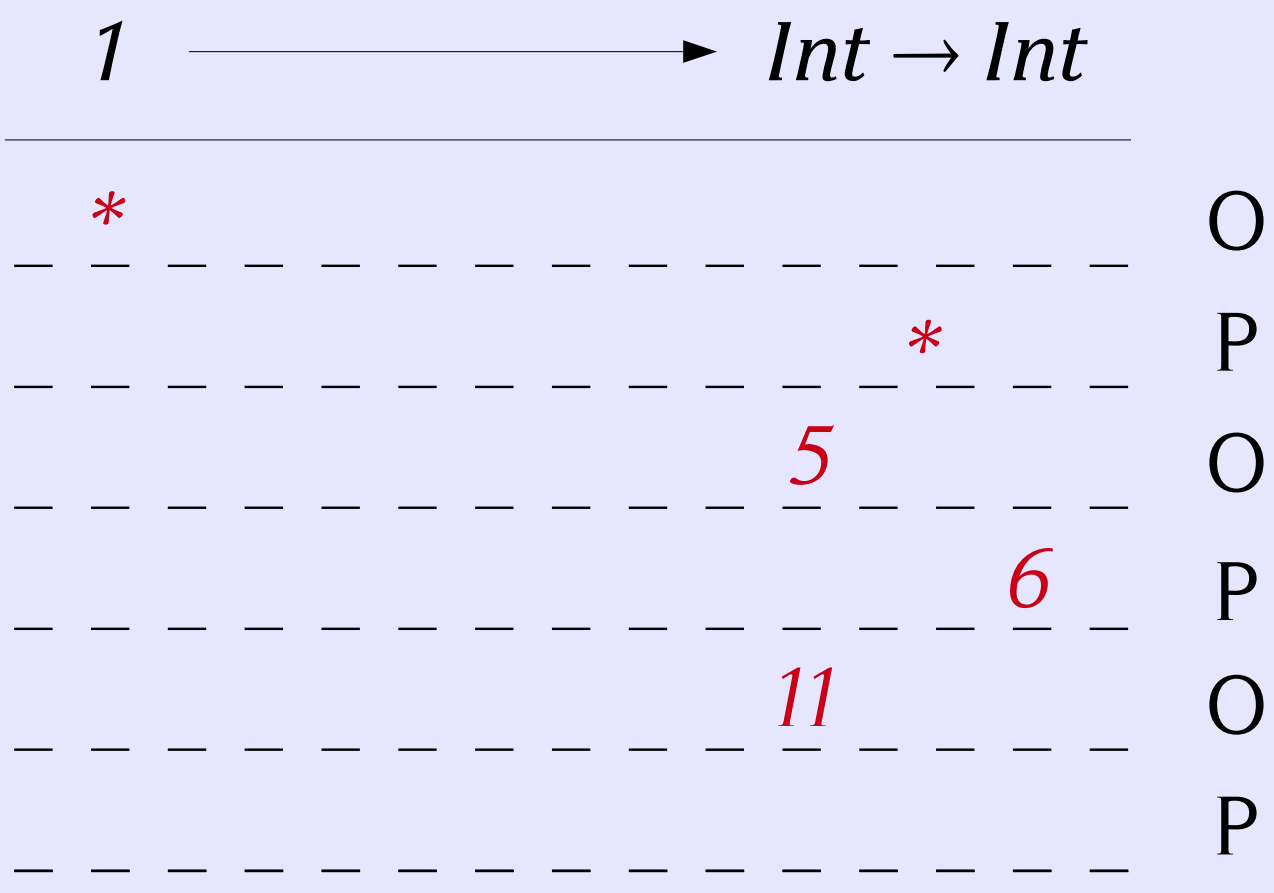
Example

$\vdash \lambda x.x{+}1 : \texttt{int} \to \texttt{int}$



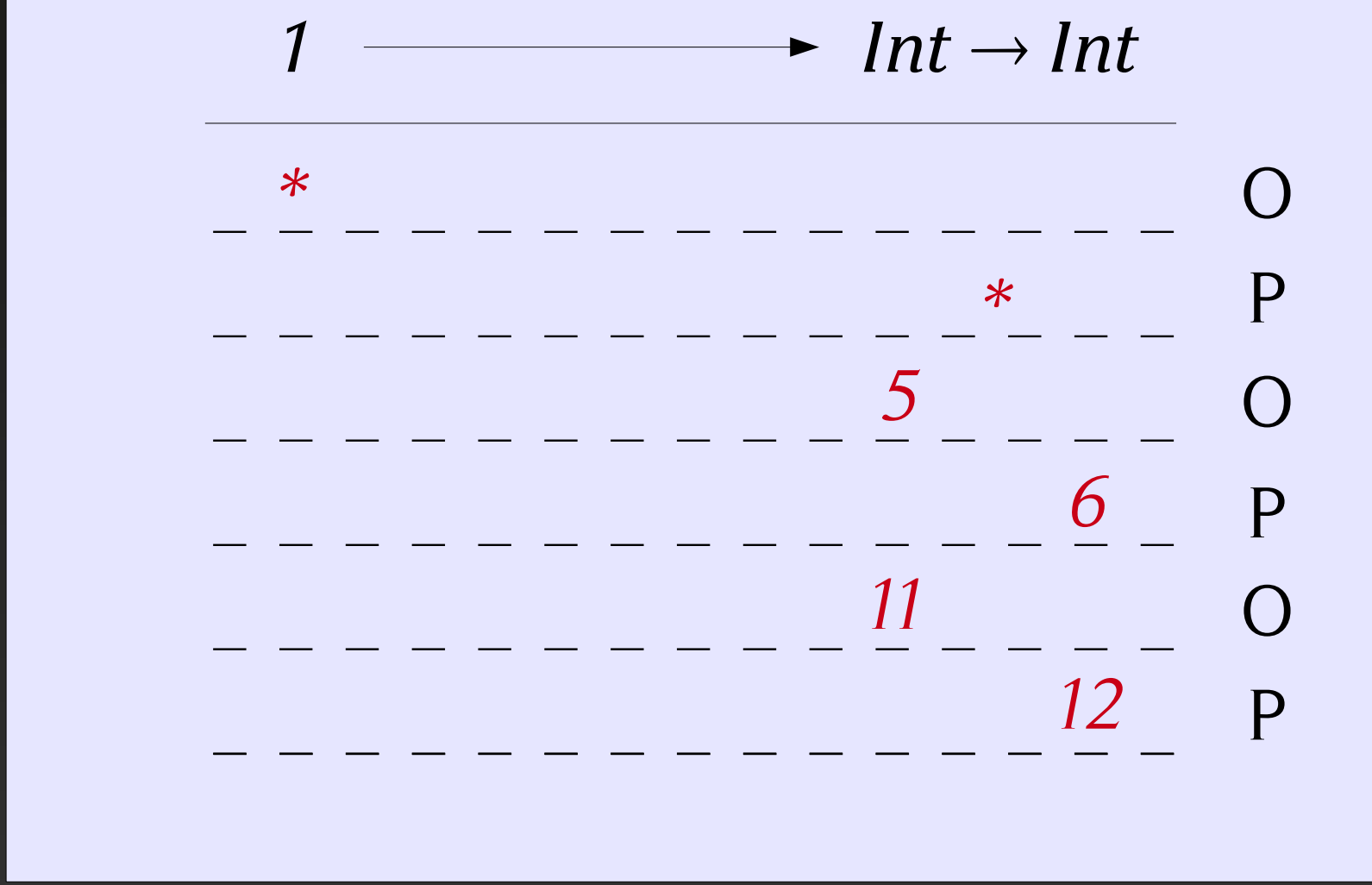
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



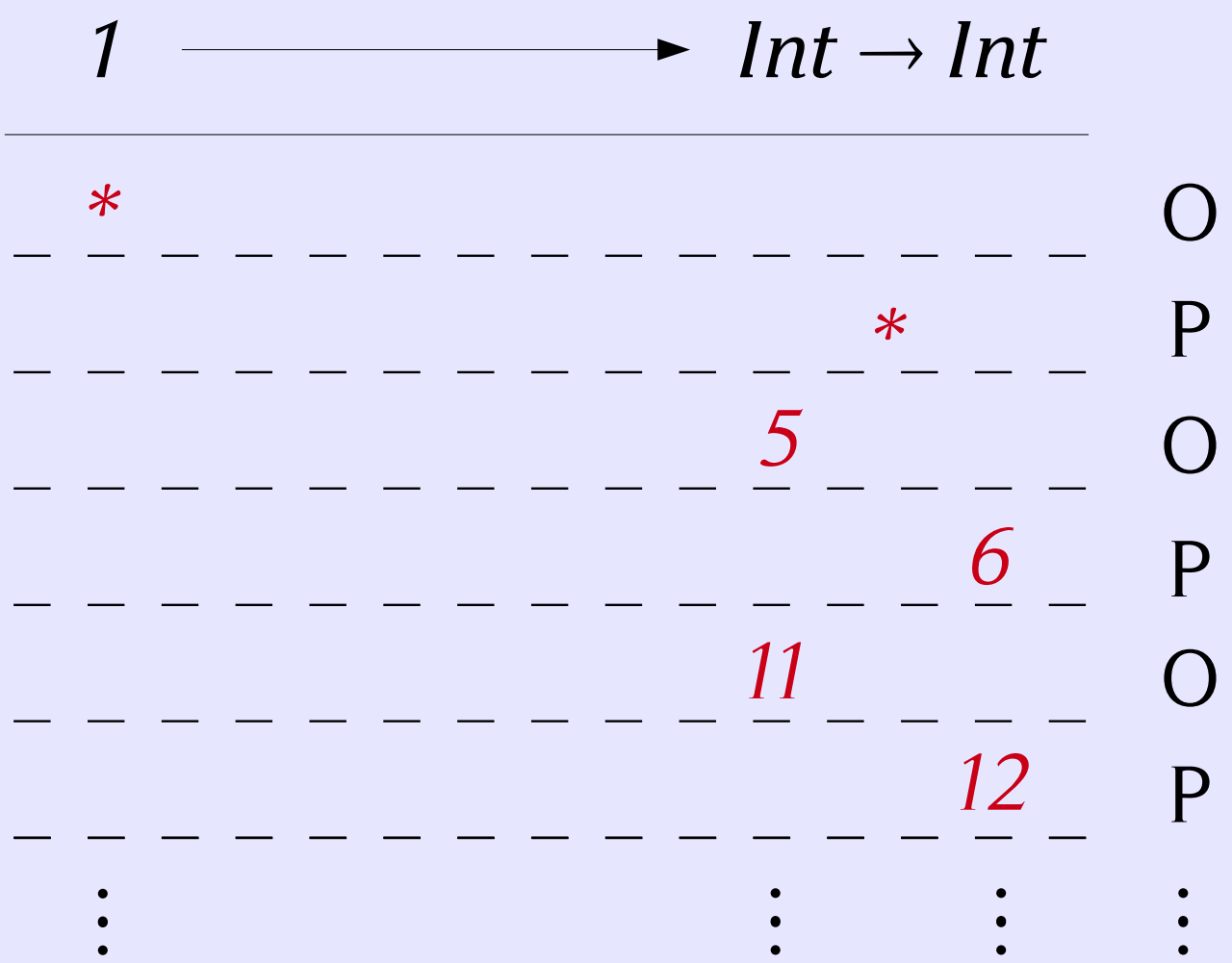
Example

$\vdash \lambda x.x+1 : \texttt{int} \to \texttt{int}$



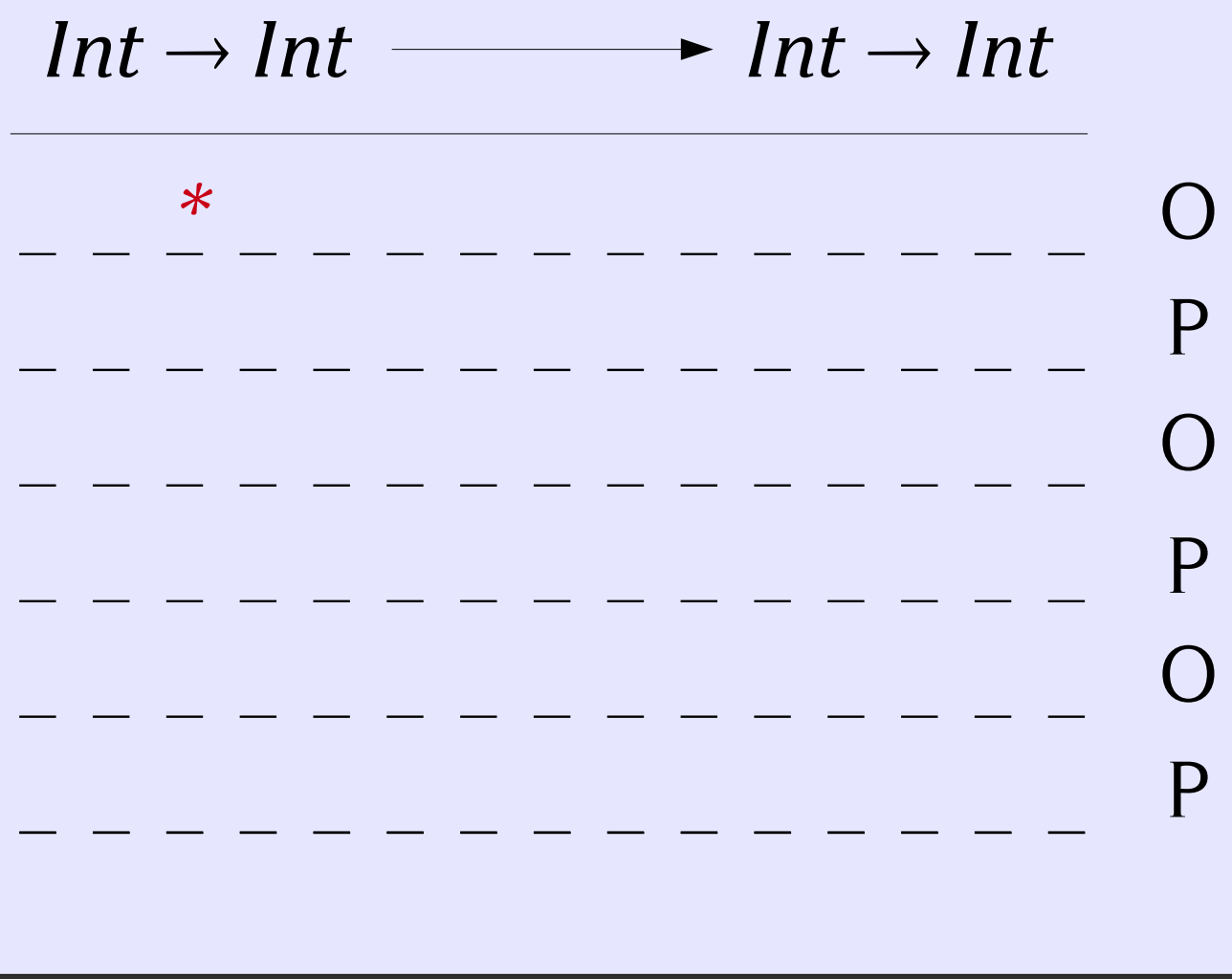
Example

$\vdash \lambda x.x{+}1 : \texttt{int} \to \texttt{int}$



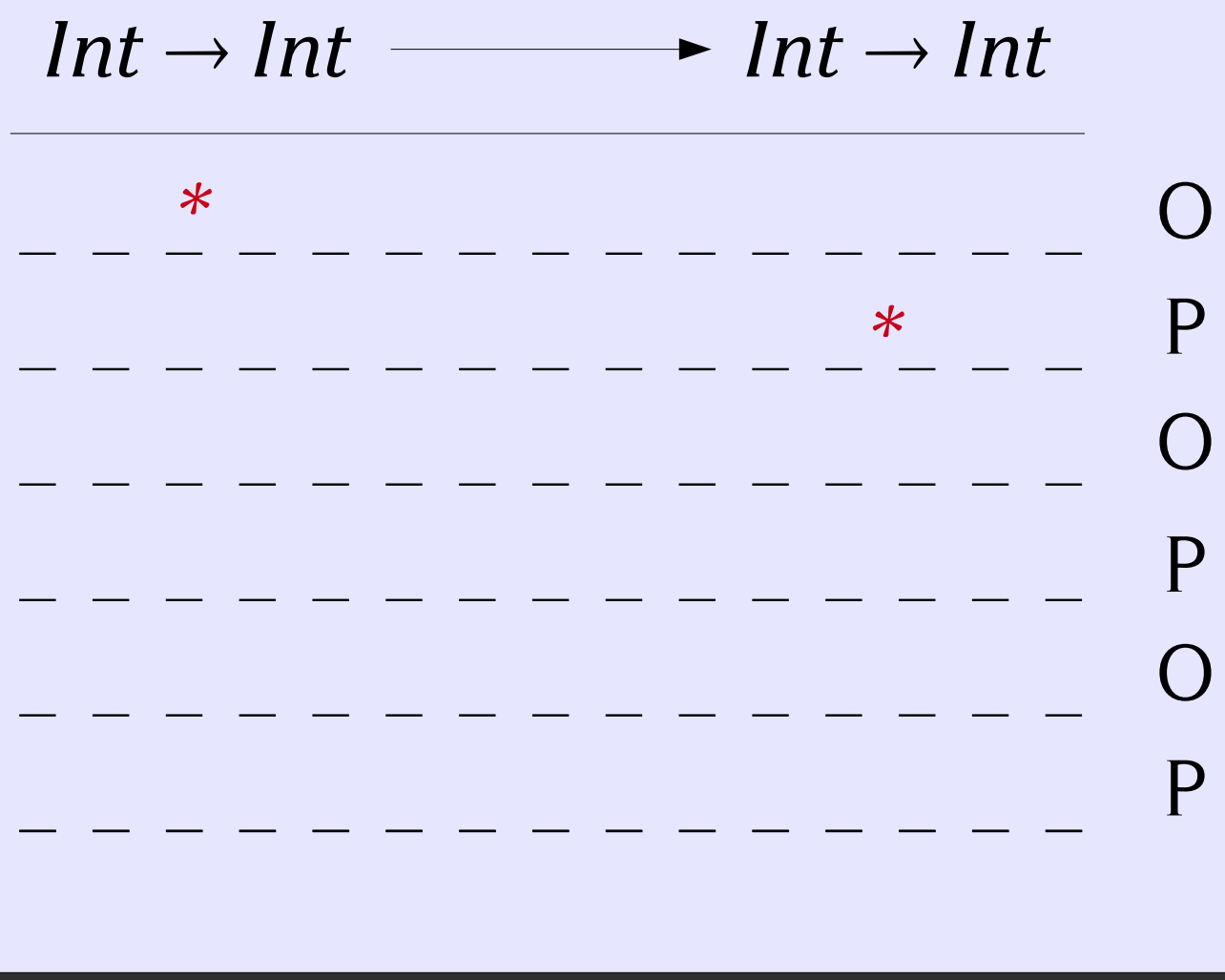
Example

$\texttt{f : int} \rightarrow \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \rightarrow \texttt{int}$



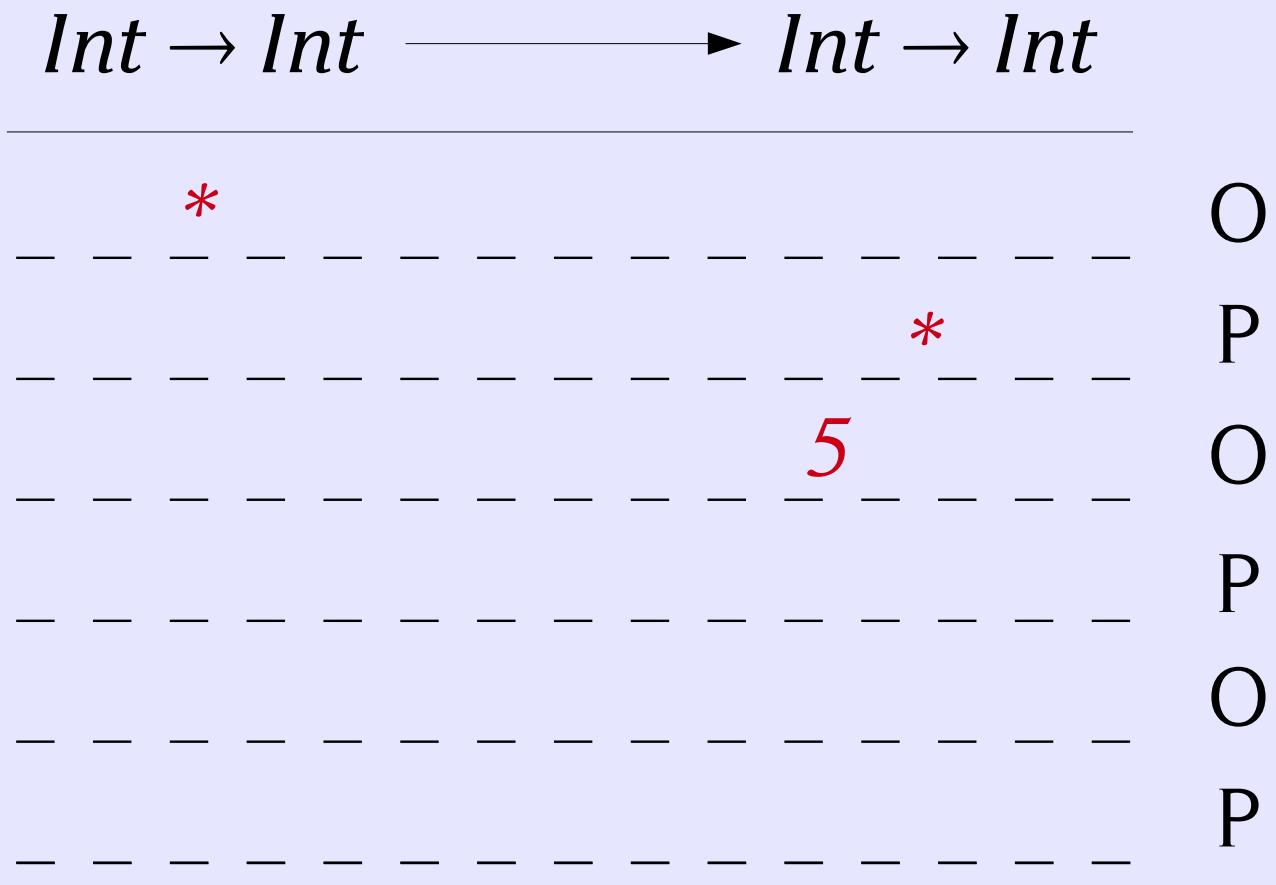
Example

$$\texttt{f : int} \rightarrow \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \rightarrow \texttt{int}$$



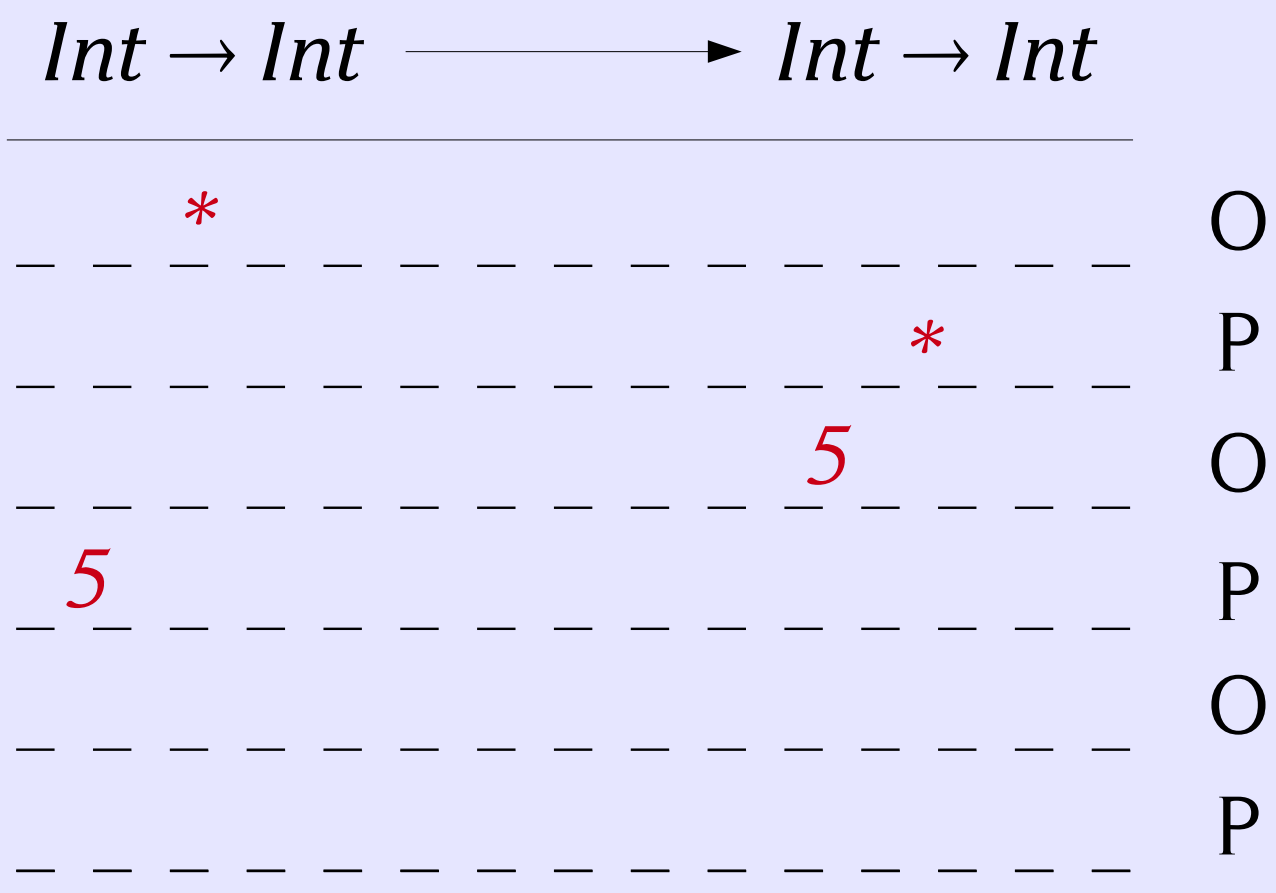
Example

$\texttt{f : int} \to \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \to \texttt{int}$



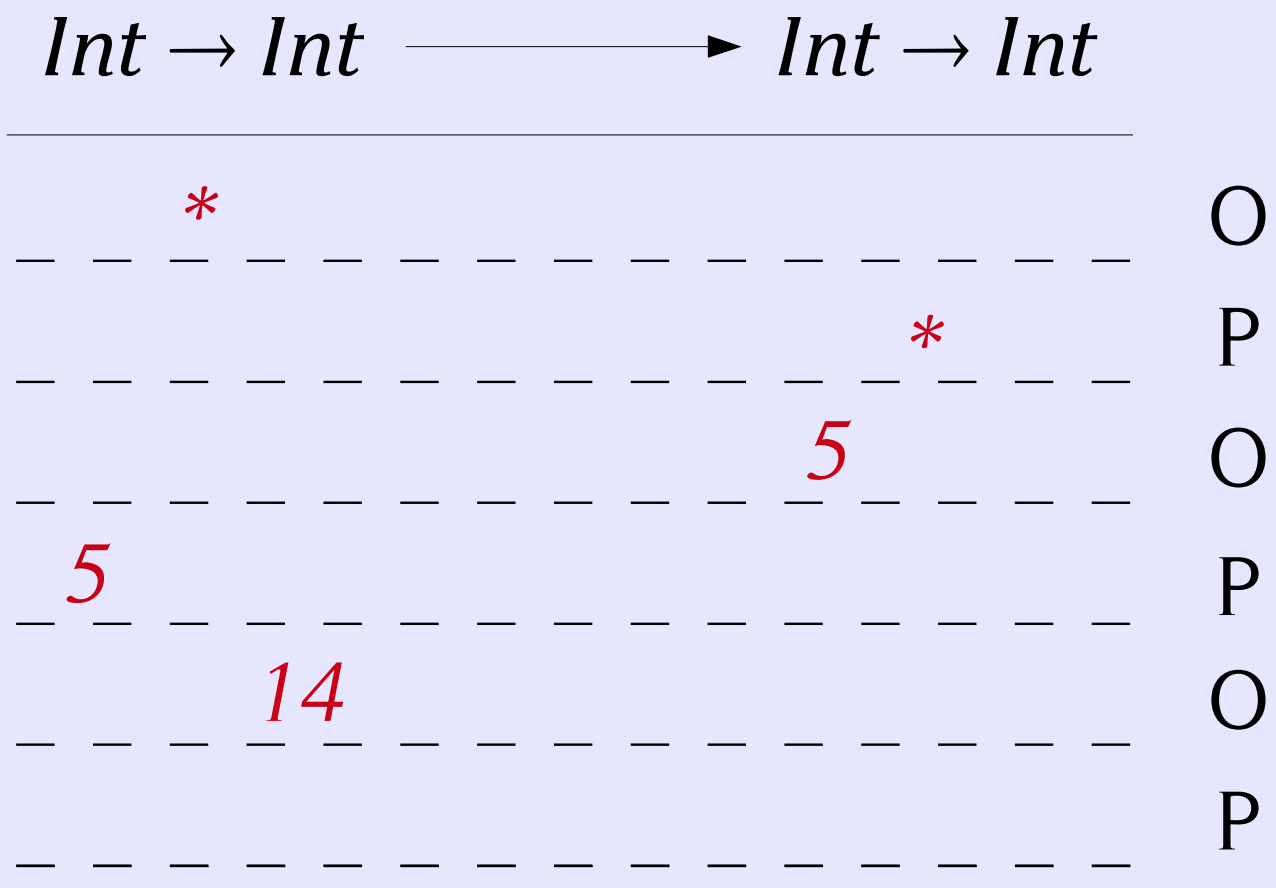
Example

$$\texttt{f : int} \rightarrow \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \rightarrow \texttt{int}$$



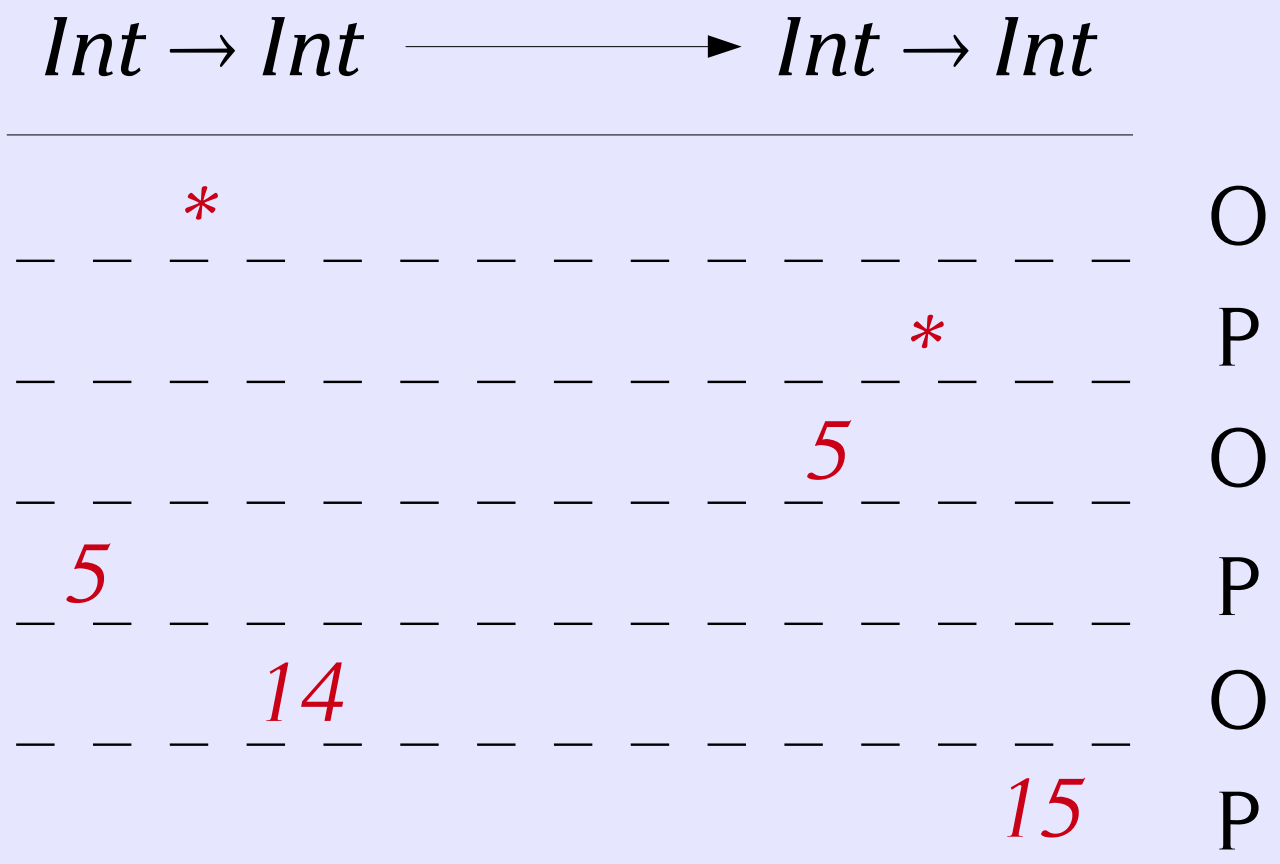
Example

$f : \mathtt{int} \to \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \to \mathtt{int}$



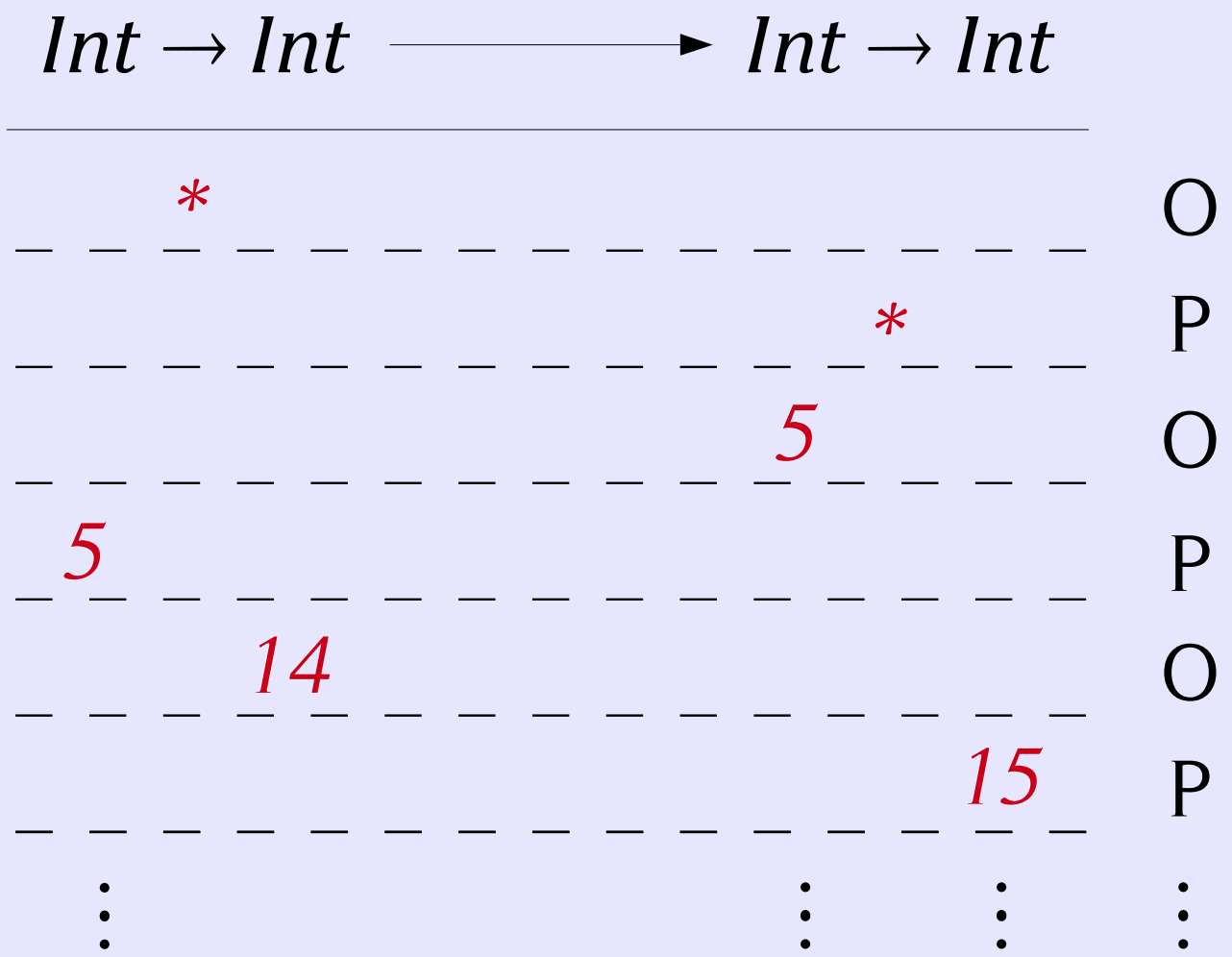
Example

$$\texttt{f : int} \to \texttt{int} \vdash \texttt{λx.f(x)+1 : int} \to \texttt{int}$$

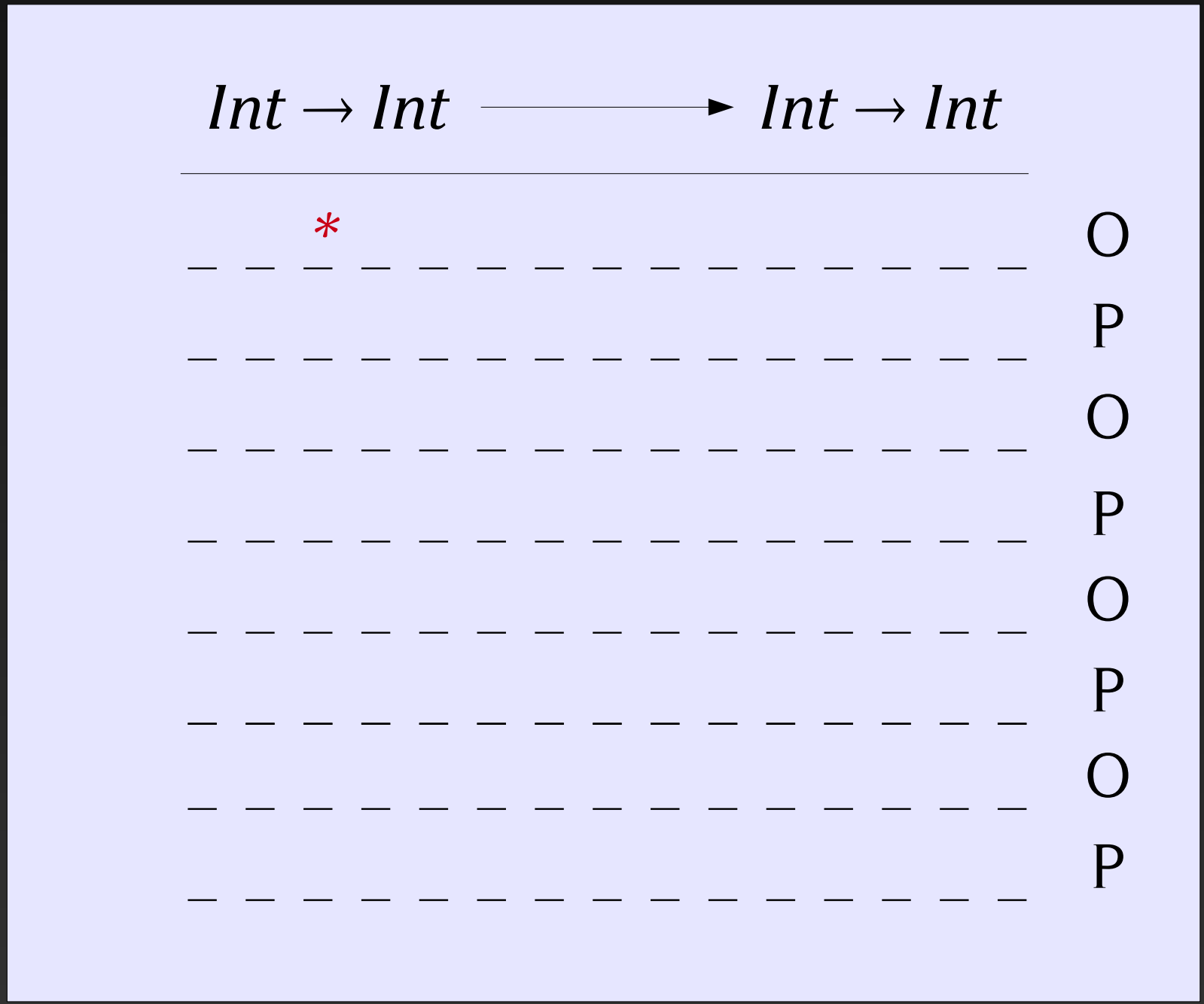


Example

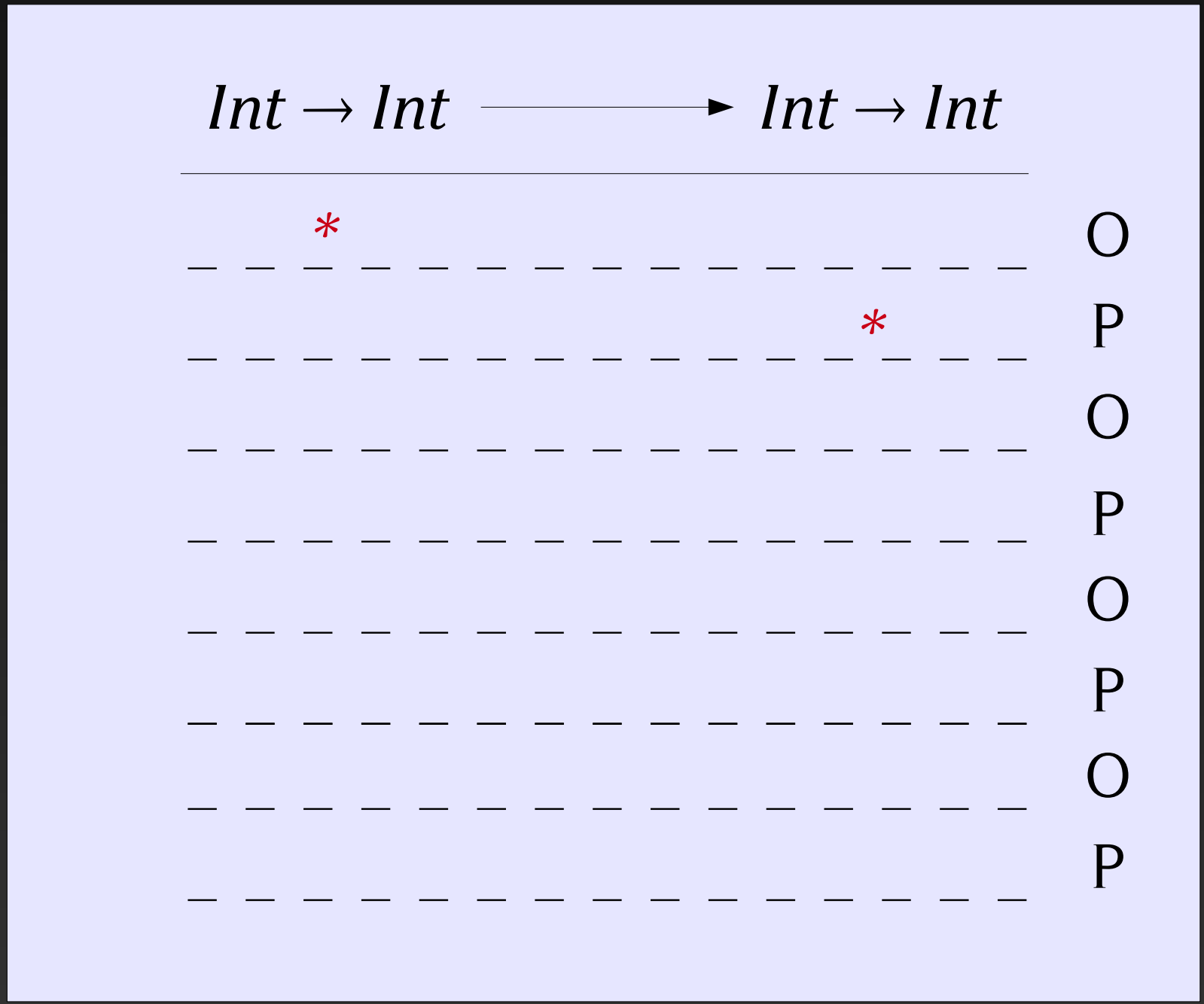
$\texttt{f : int} \to \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \to \texttt{int}$



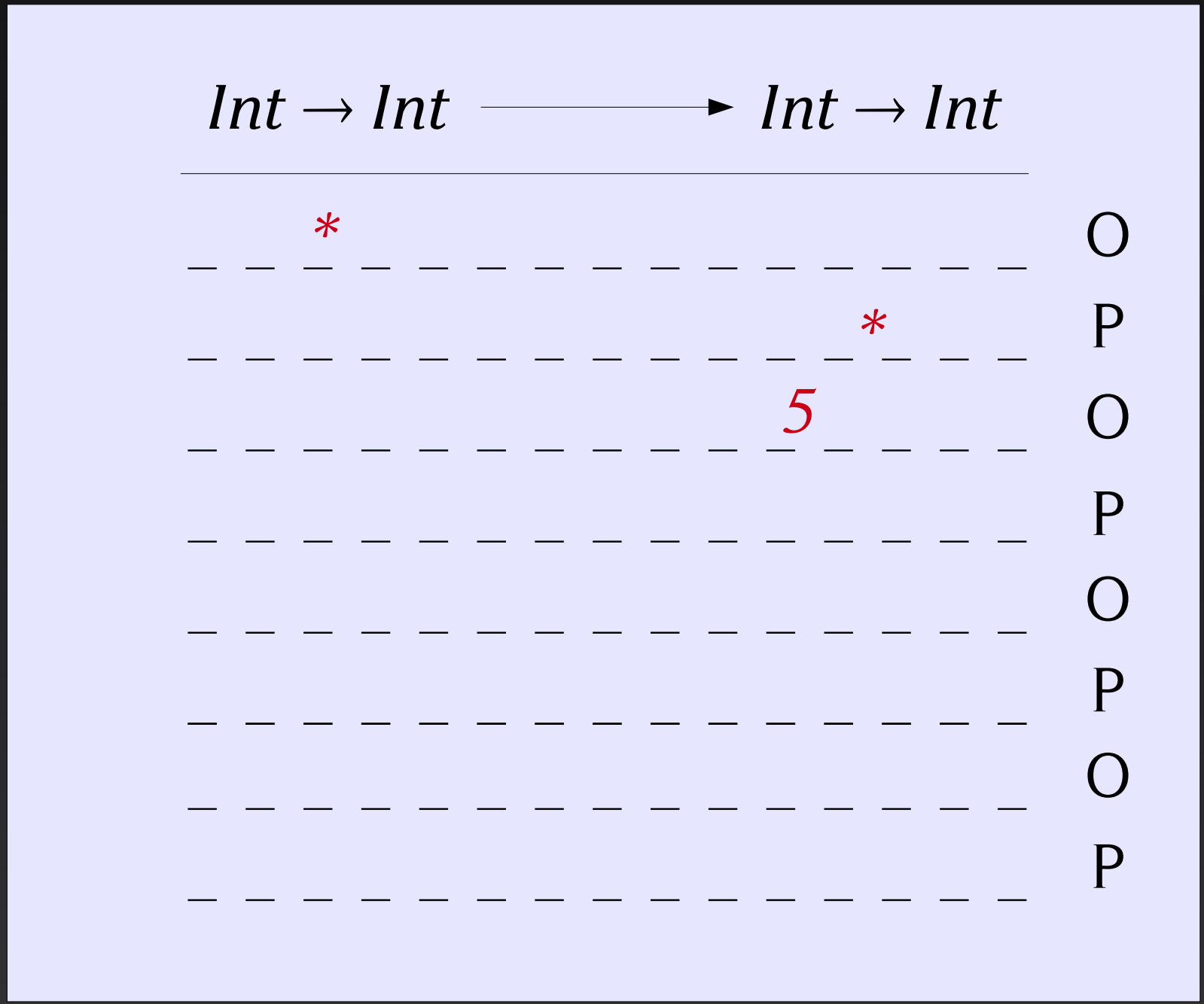
Example



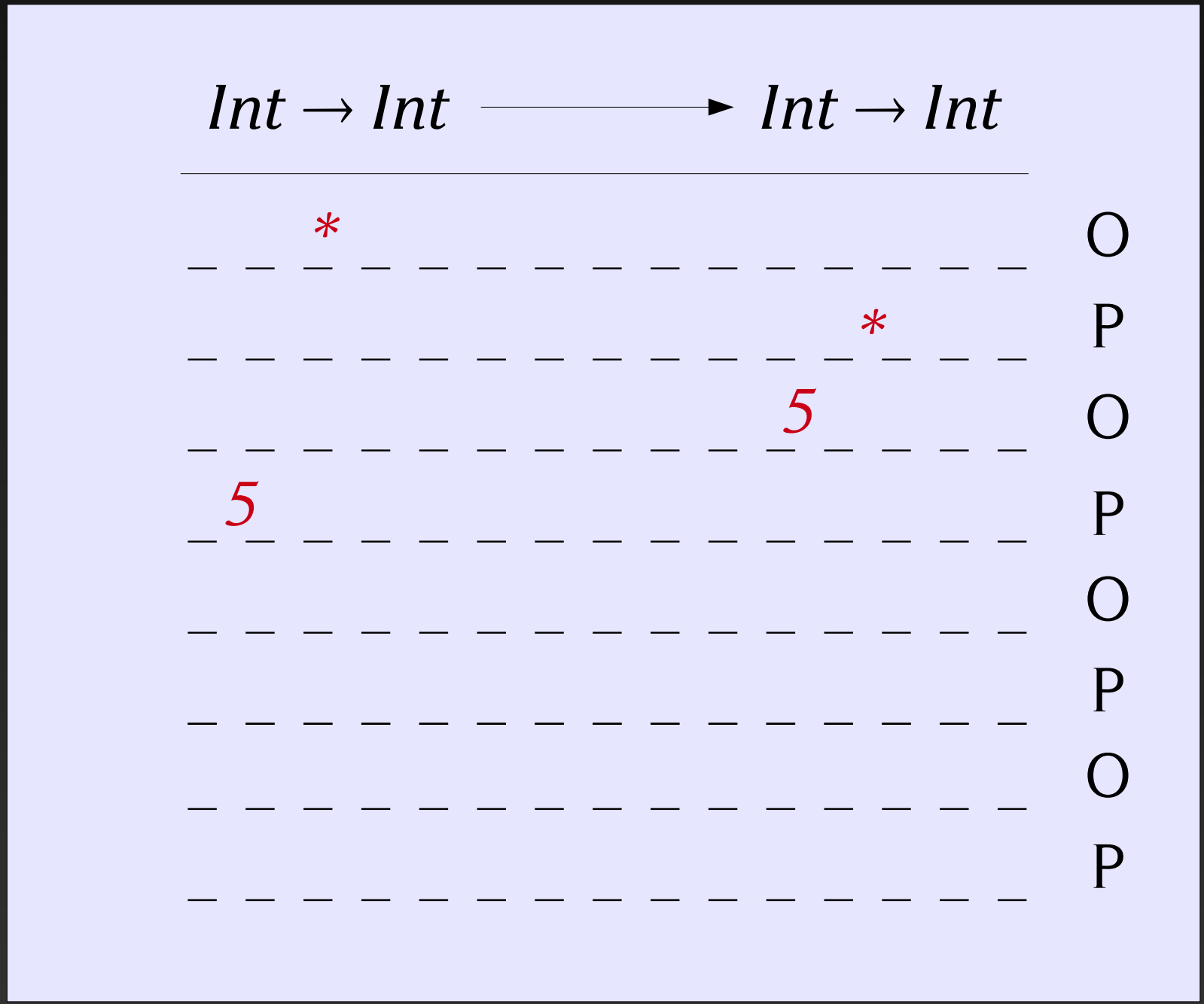
Example



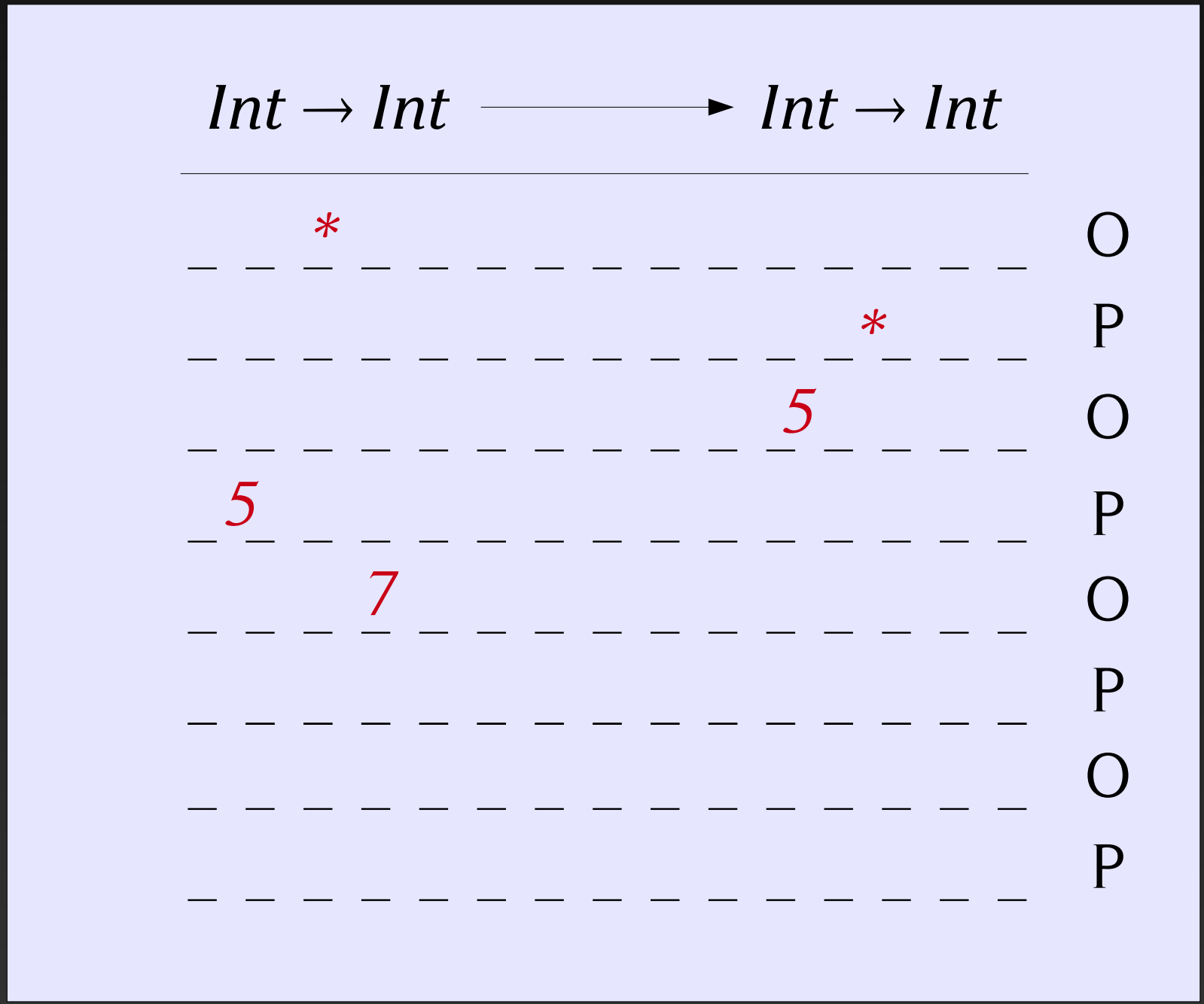
Example



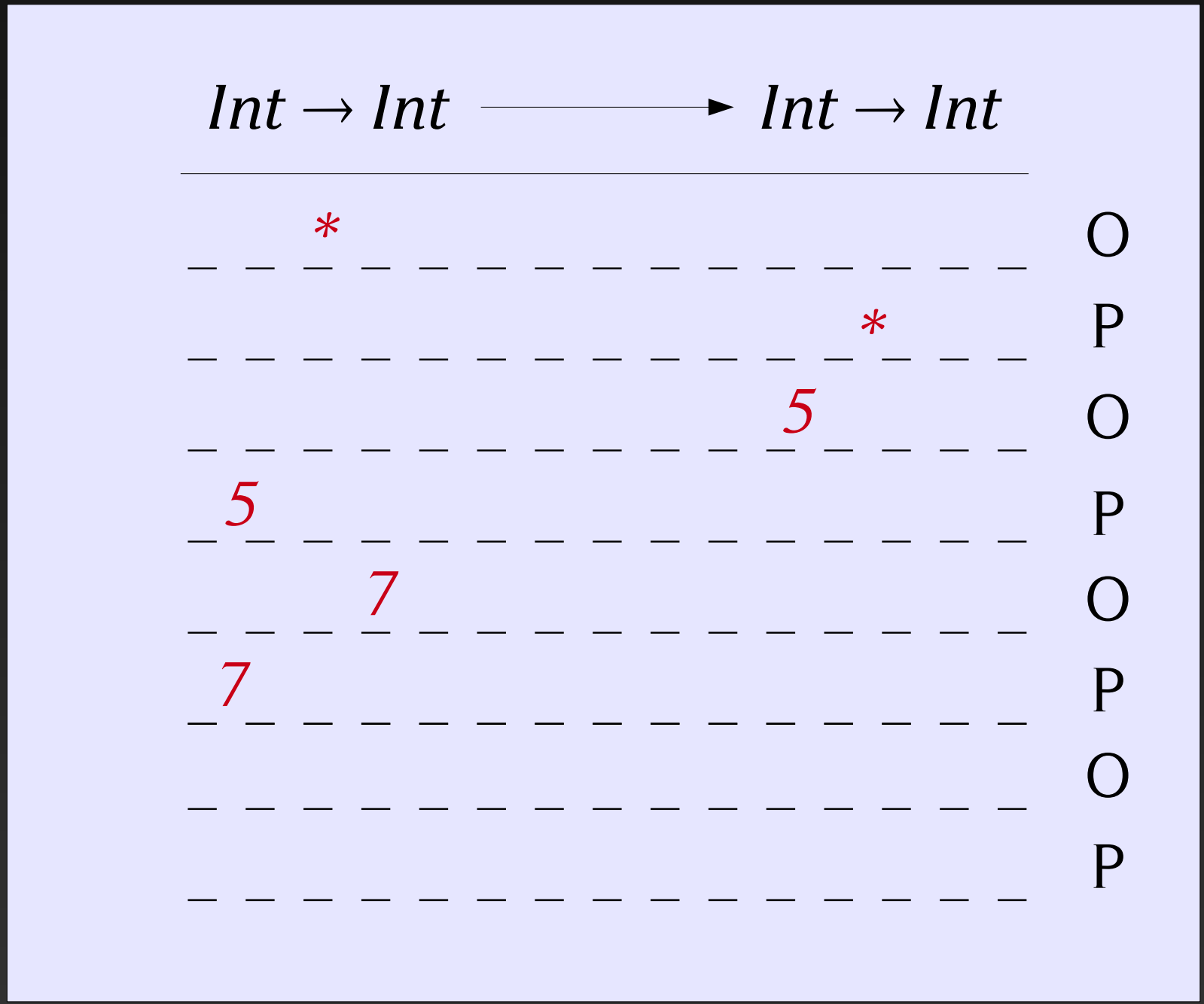
Example



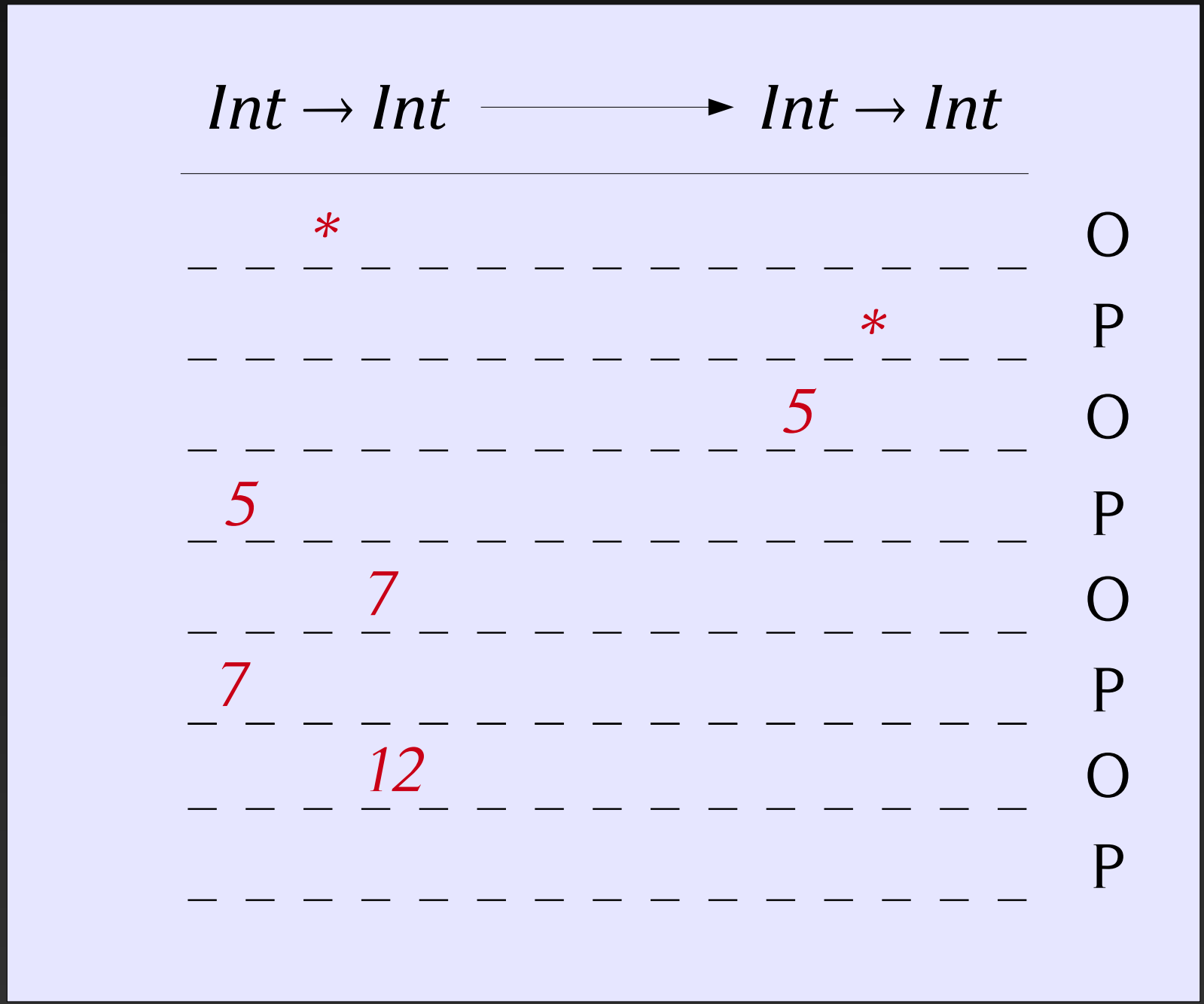
Example



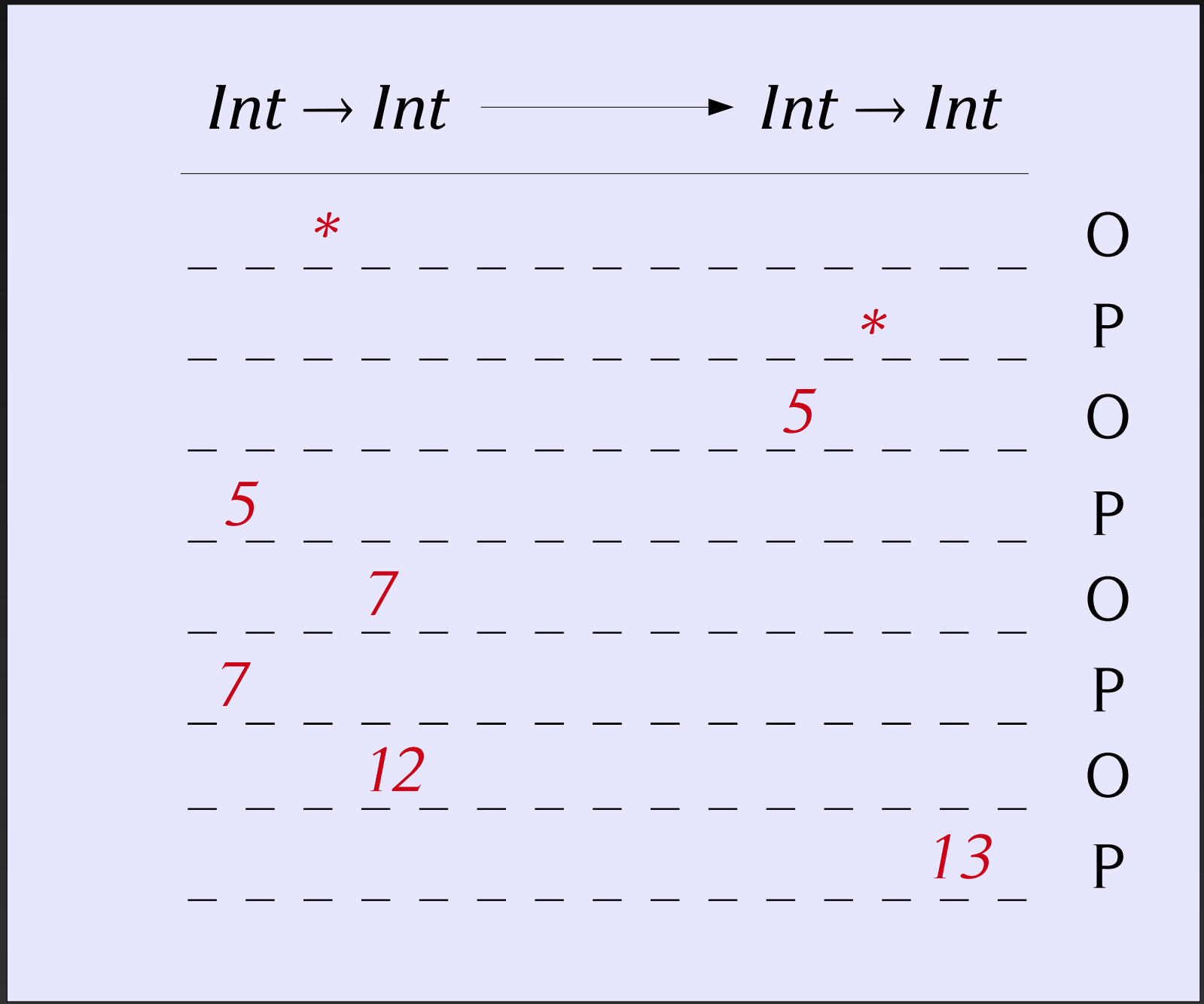
Example



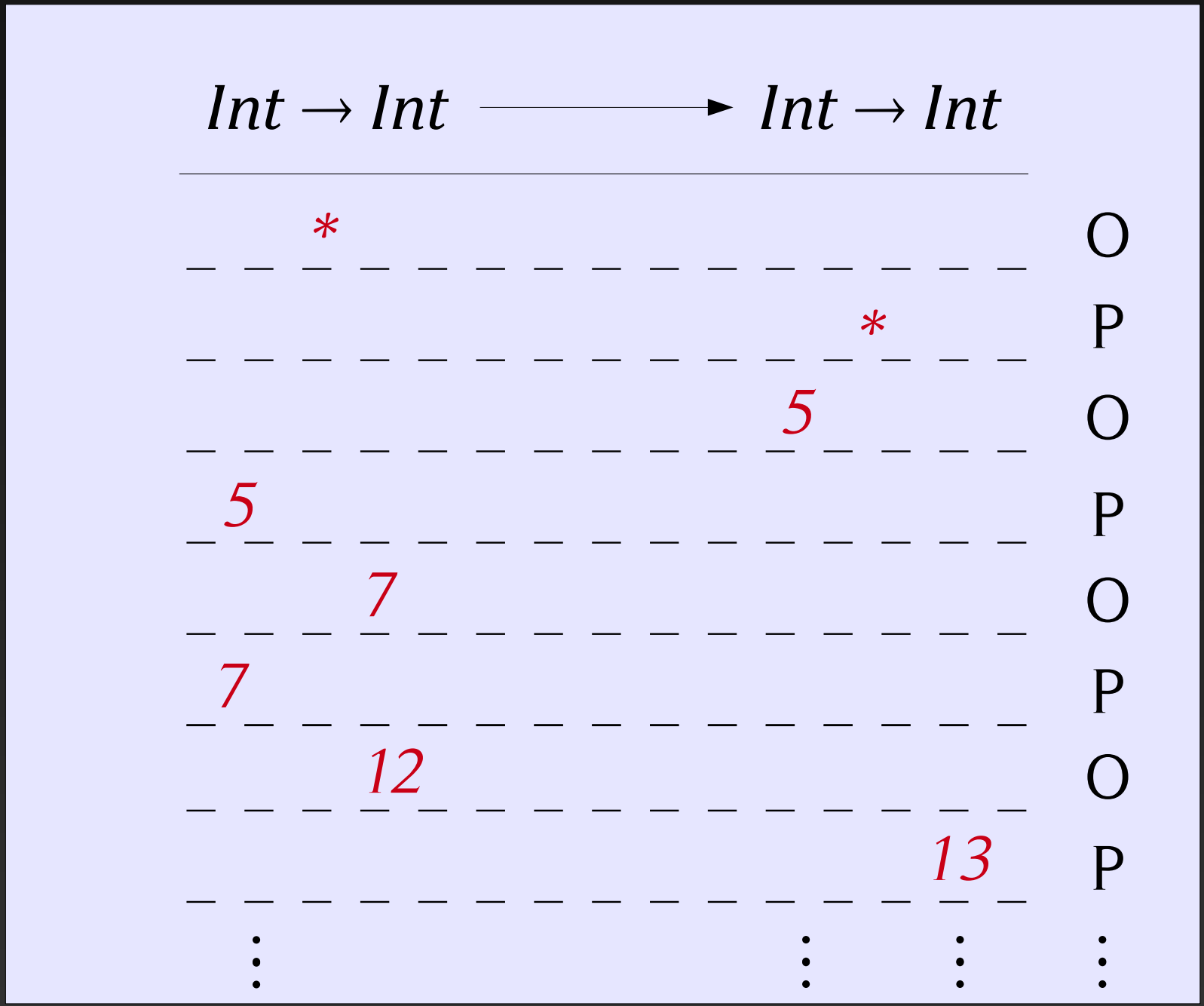
Example



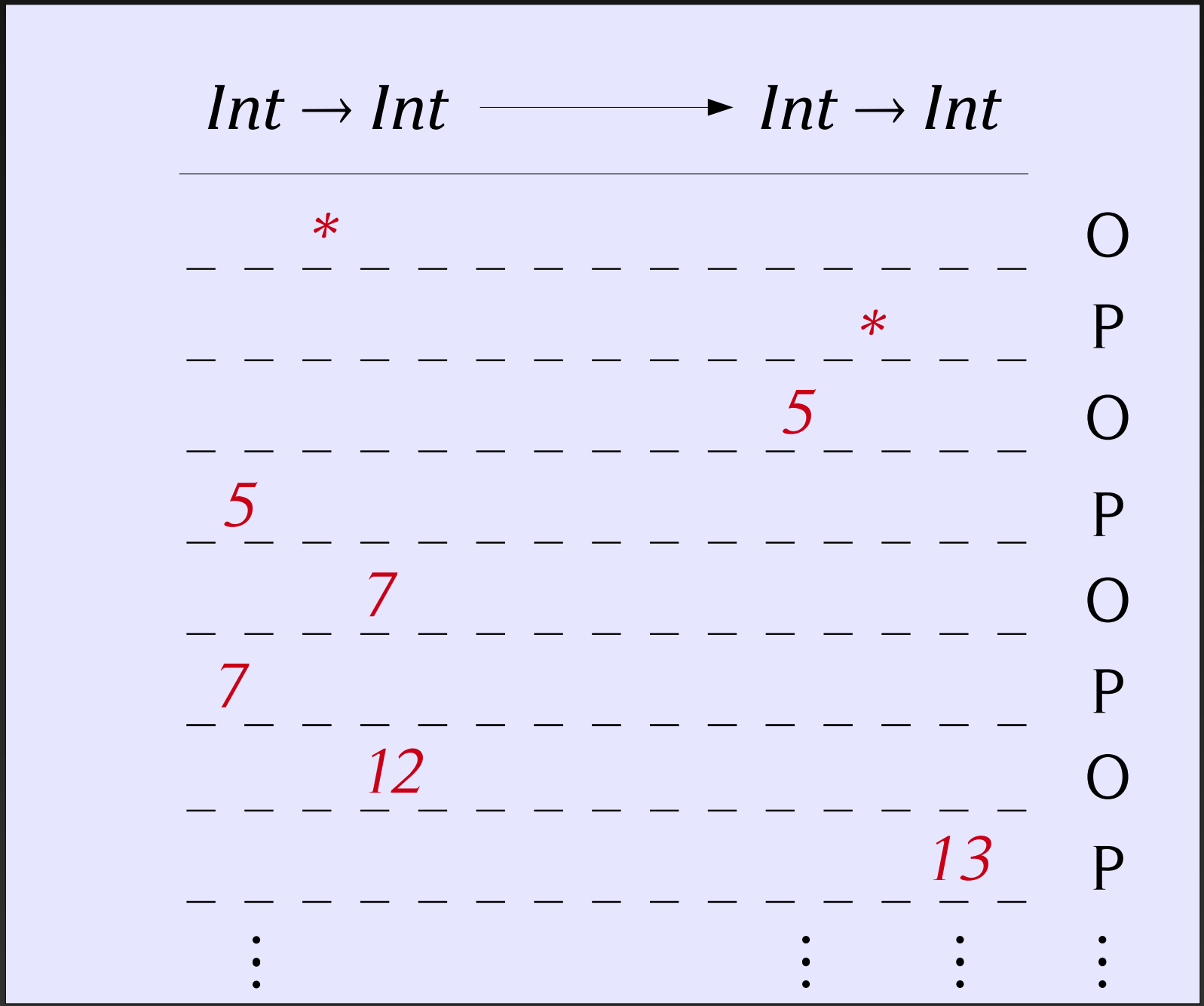
Example



Example



$$f : \texttt{int} \to \texttt{int} \vdash \lambda x.f(f(x))+1 : \texttt{int} \to \texttt{int}$$



Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)
- Qualitative games
- Programs = *strategies* for Proponent

Composition

$\vdash \lambda x.x+1 : \mathtt{int} \longrightarrow \mathtt{int}$

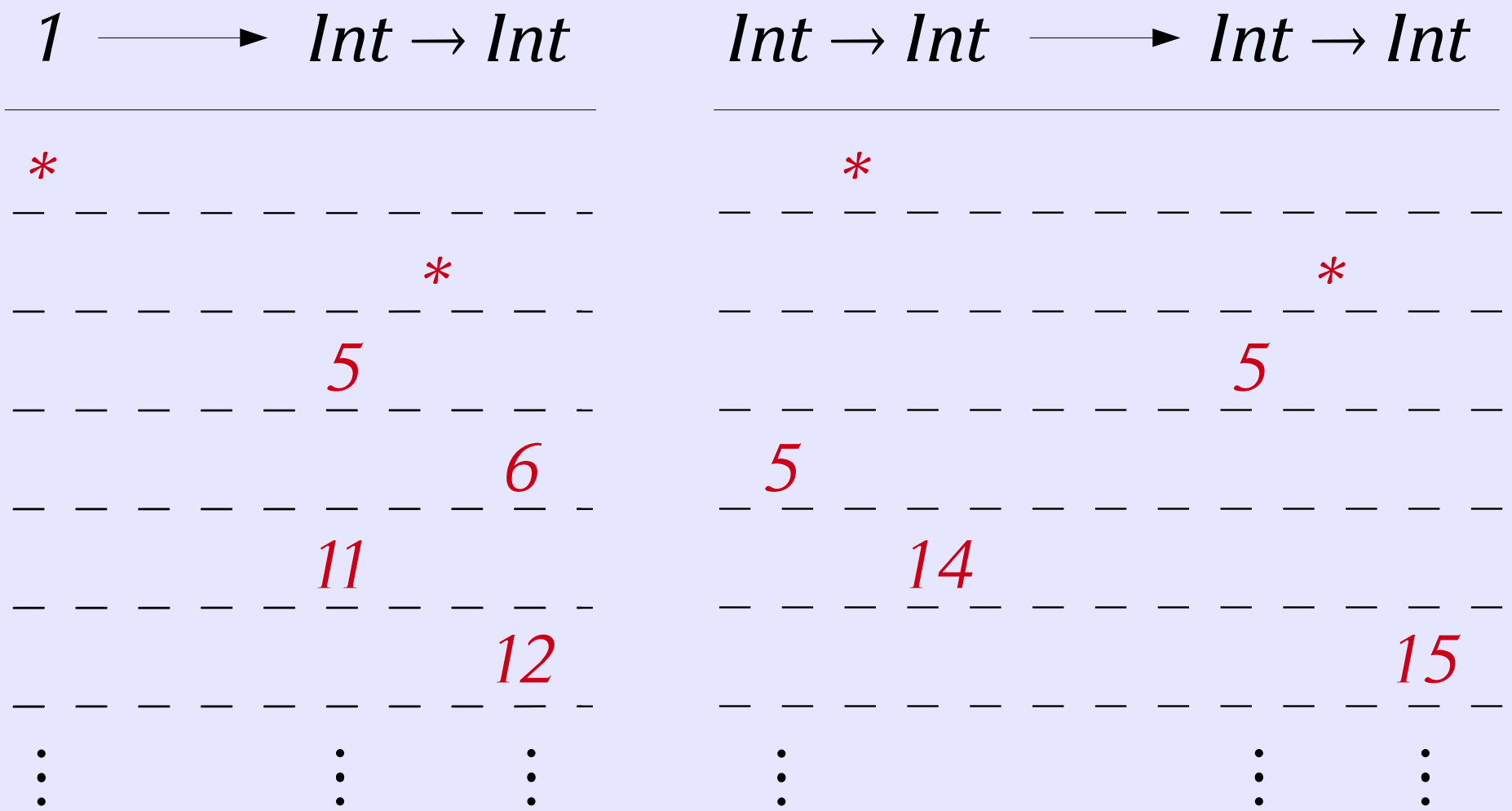
$f : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

Composition

$\vdash \lambda x.x+1 : \mathtt{int} \longrightarrow \mathtt{int}$

$f : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

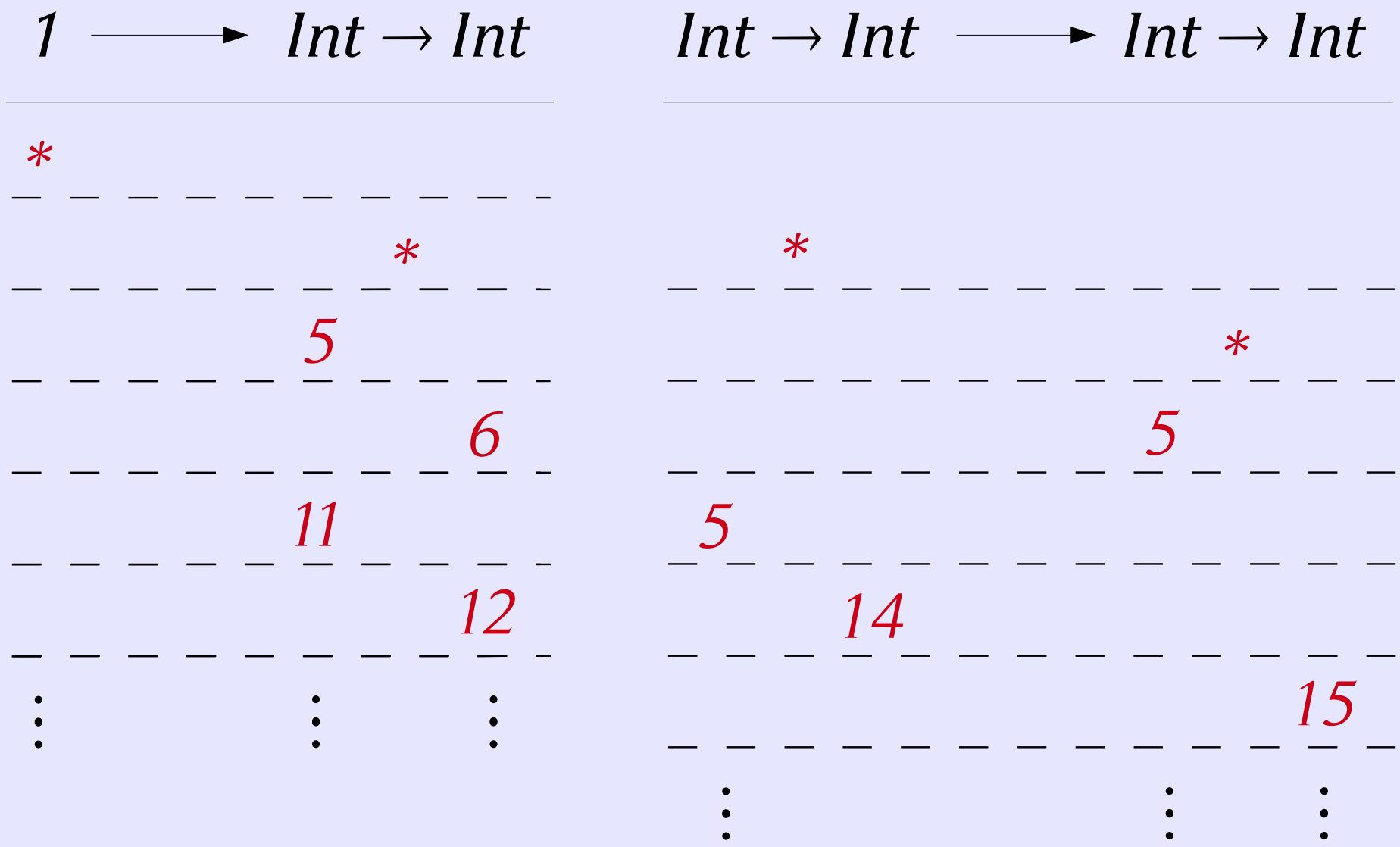
Composition



`⊢ λx.x+1 : int ⟶ int`

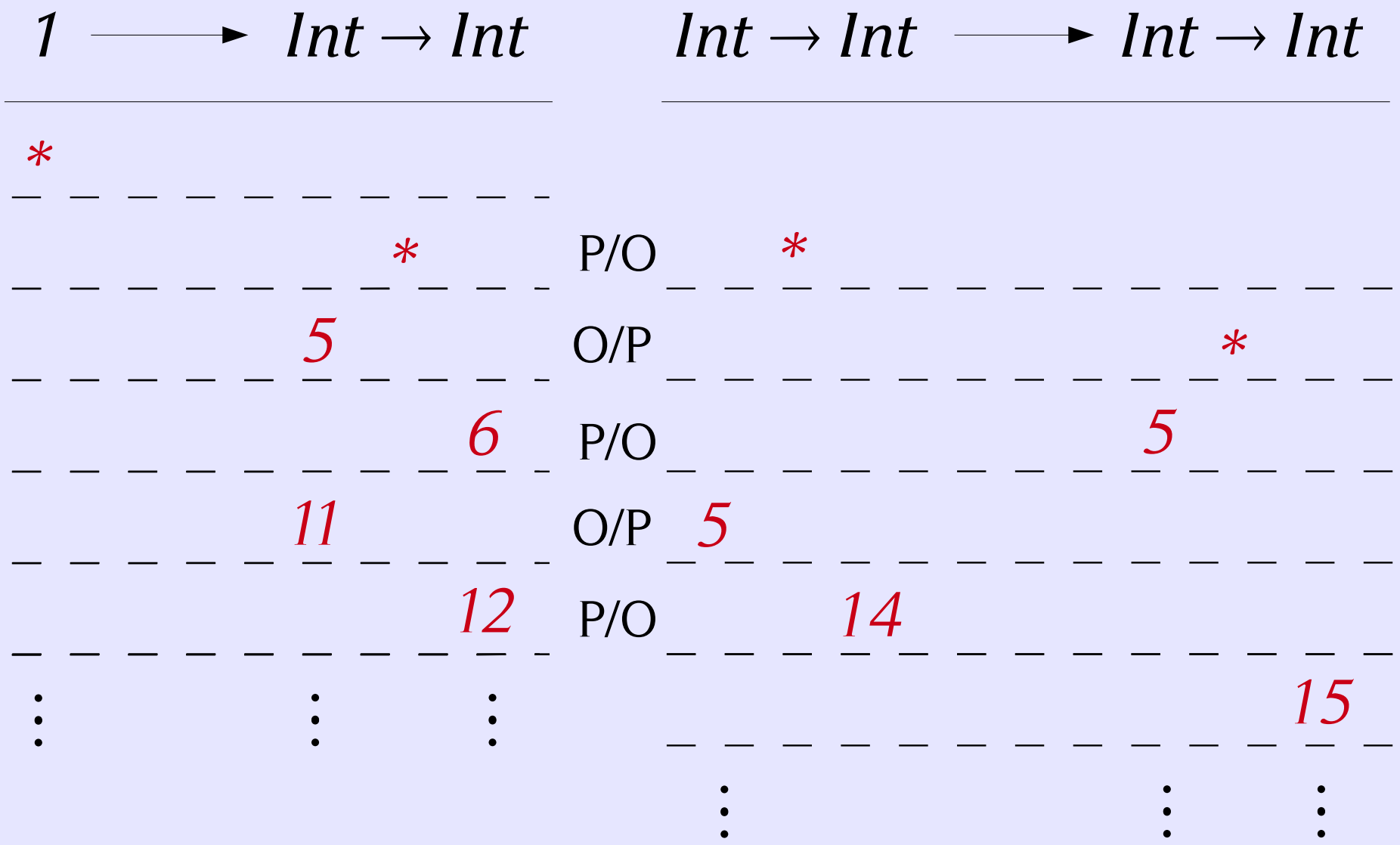
`f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int`

Composition



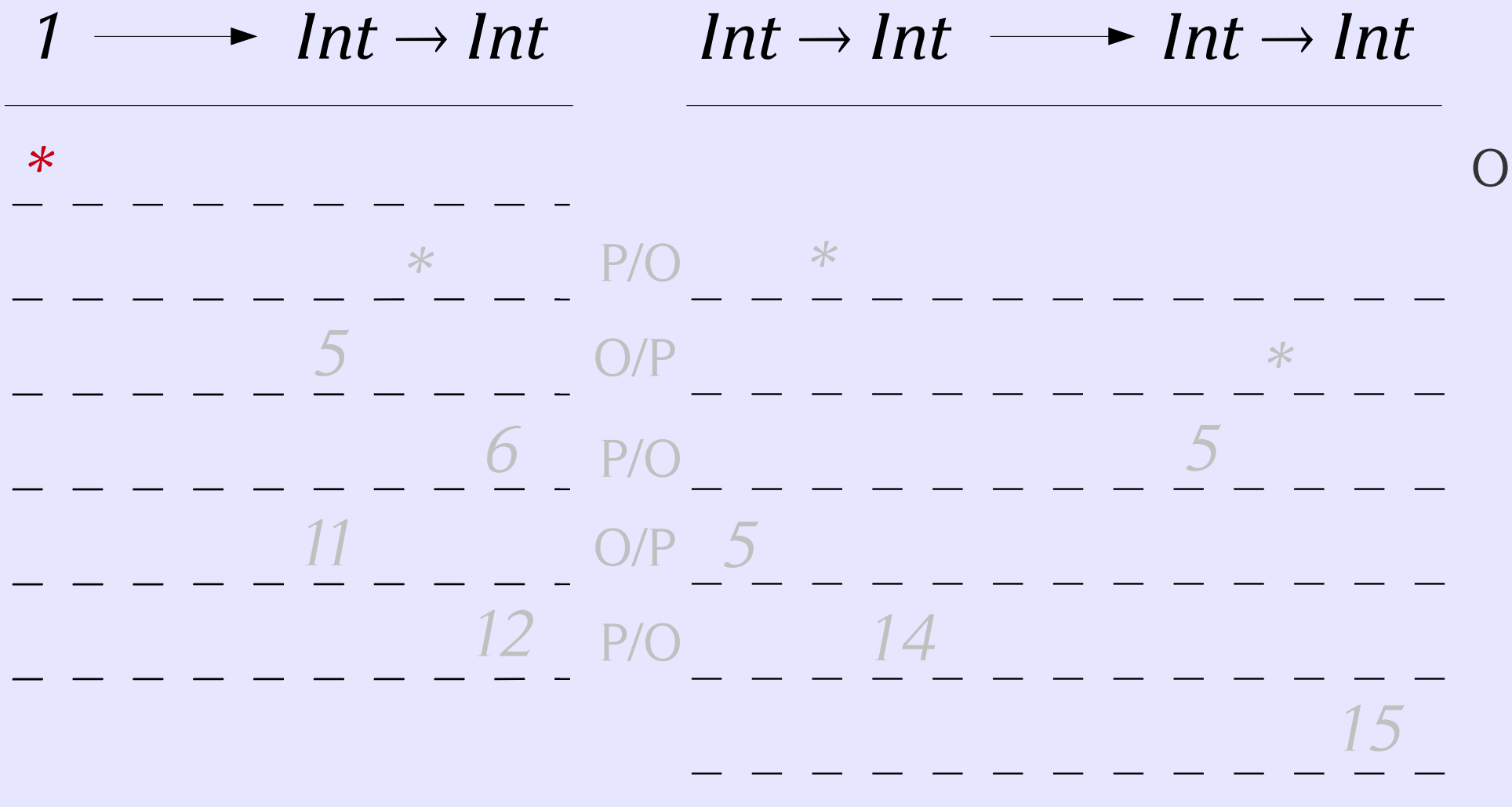
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

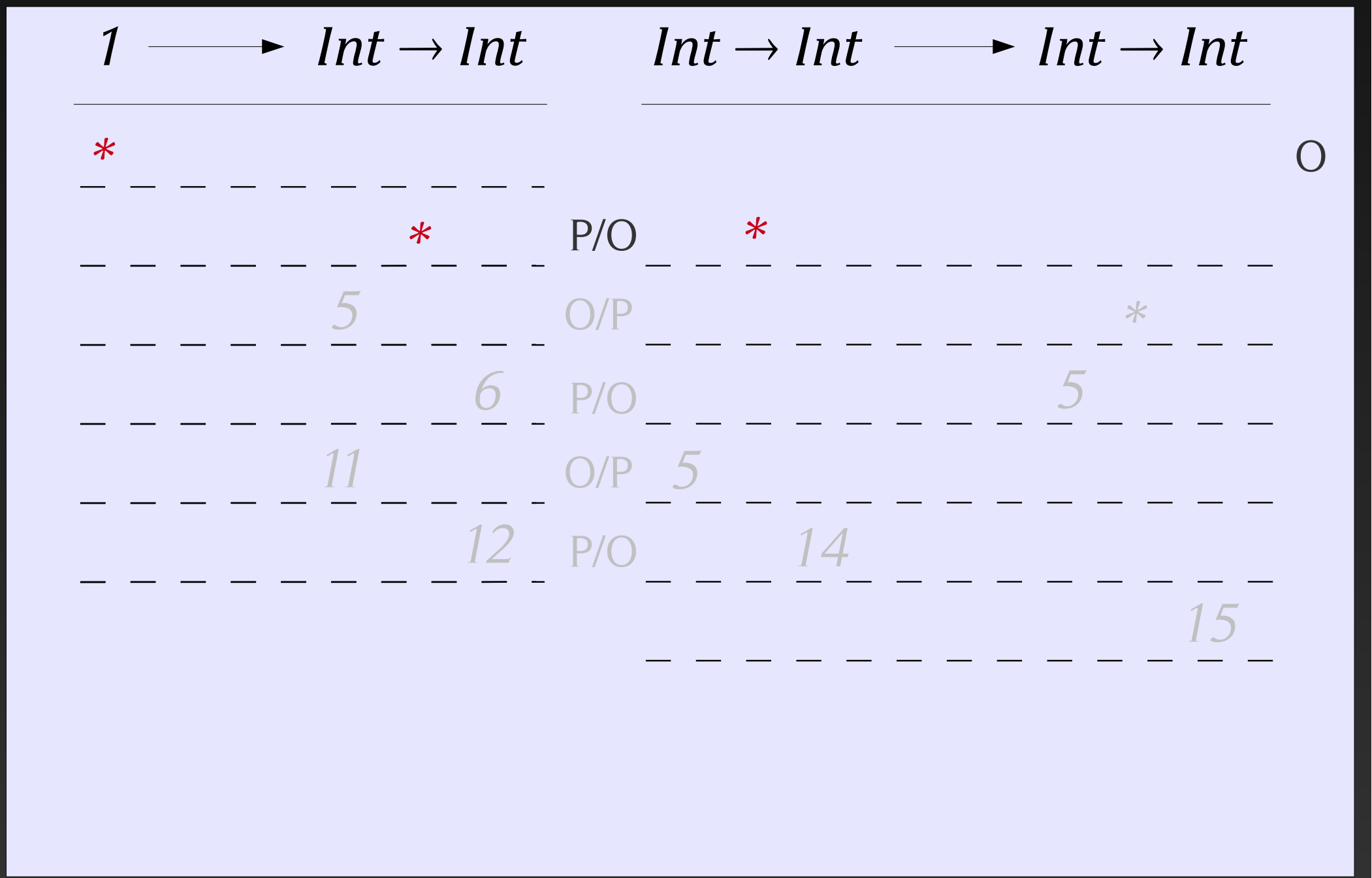
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

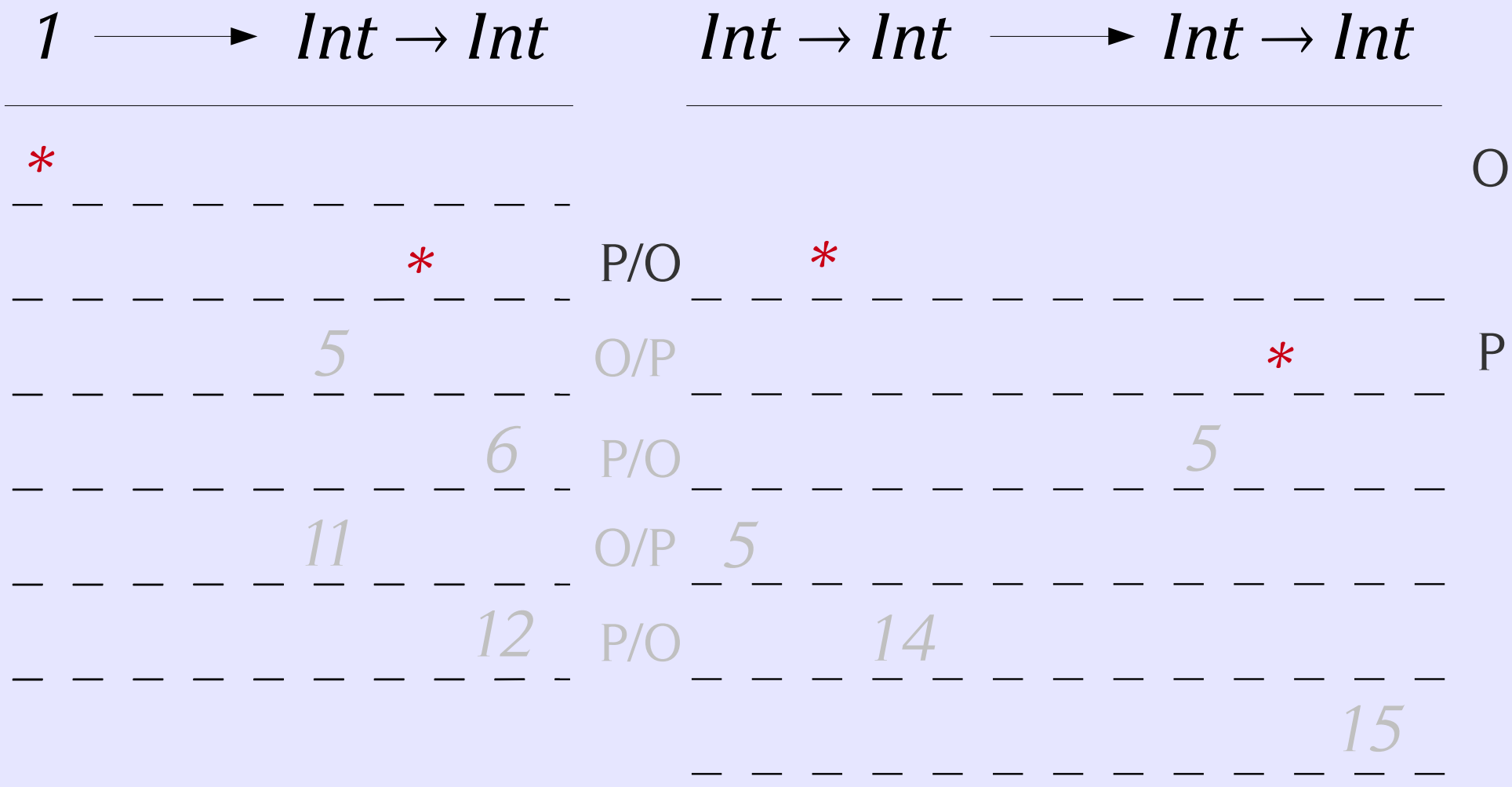
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

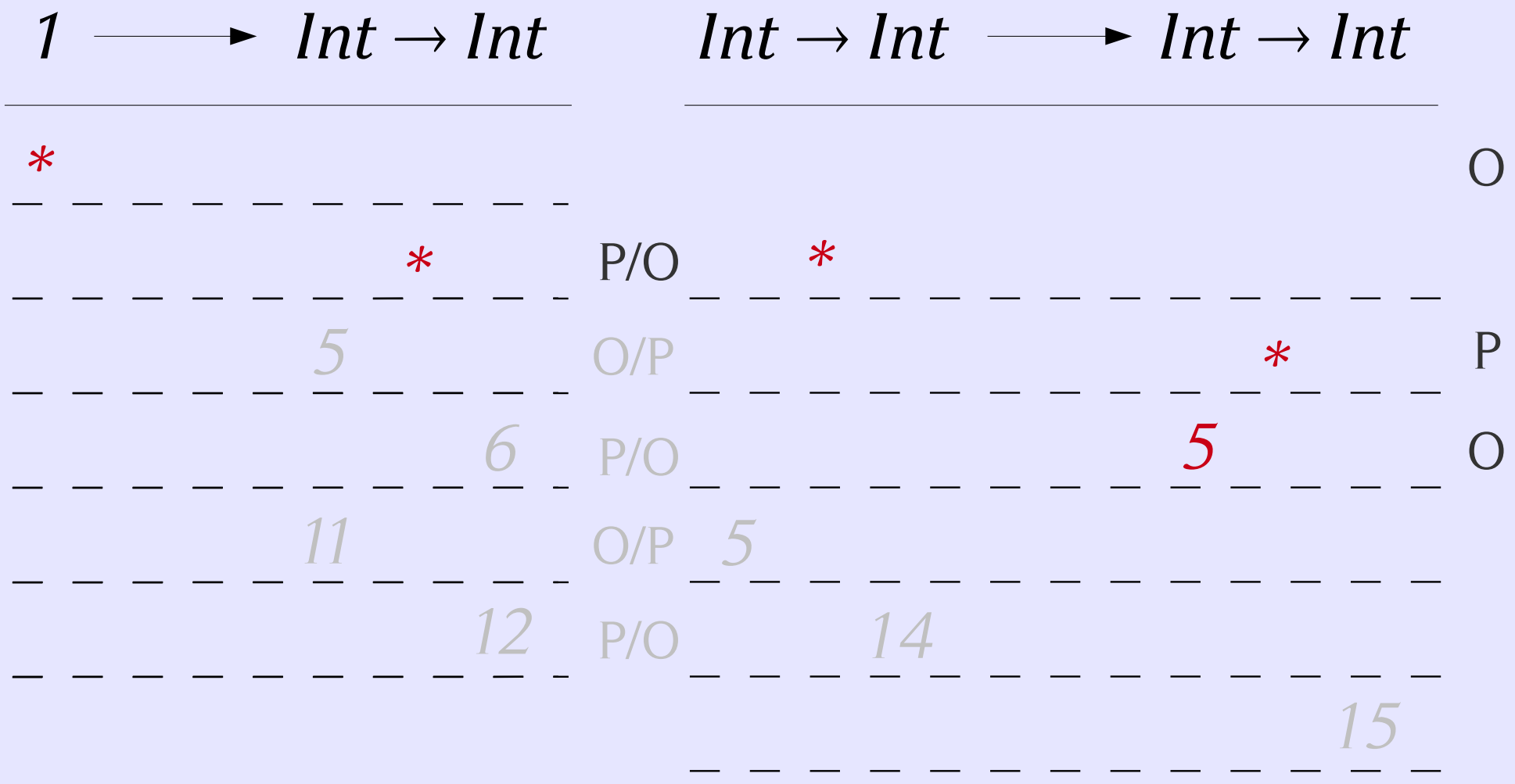
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

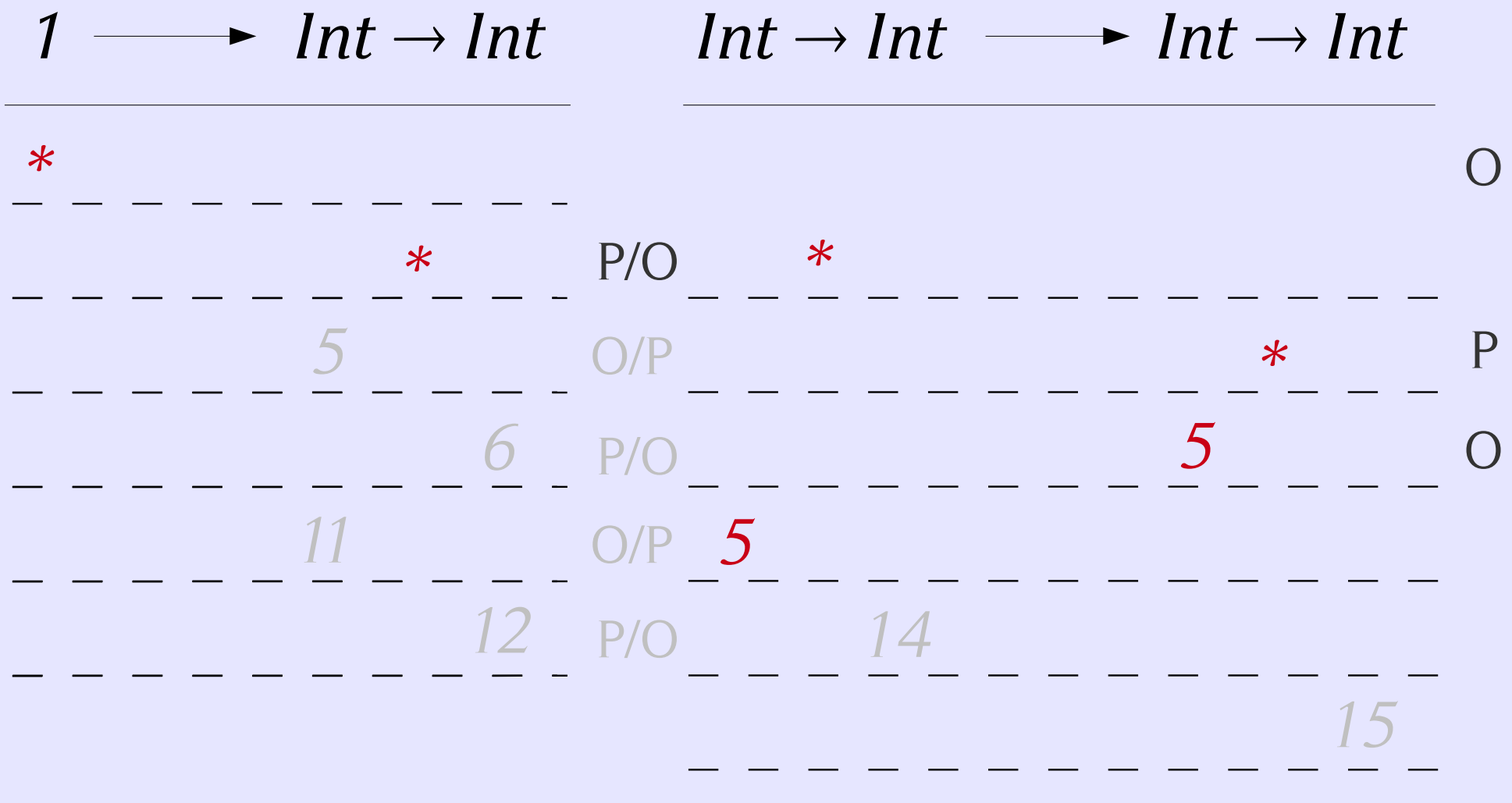
Composition



`⊢ λx.x+1 : int ⟶ int`

`f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int`

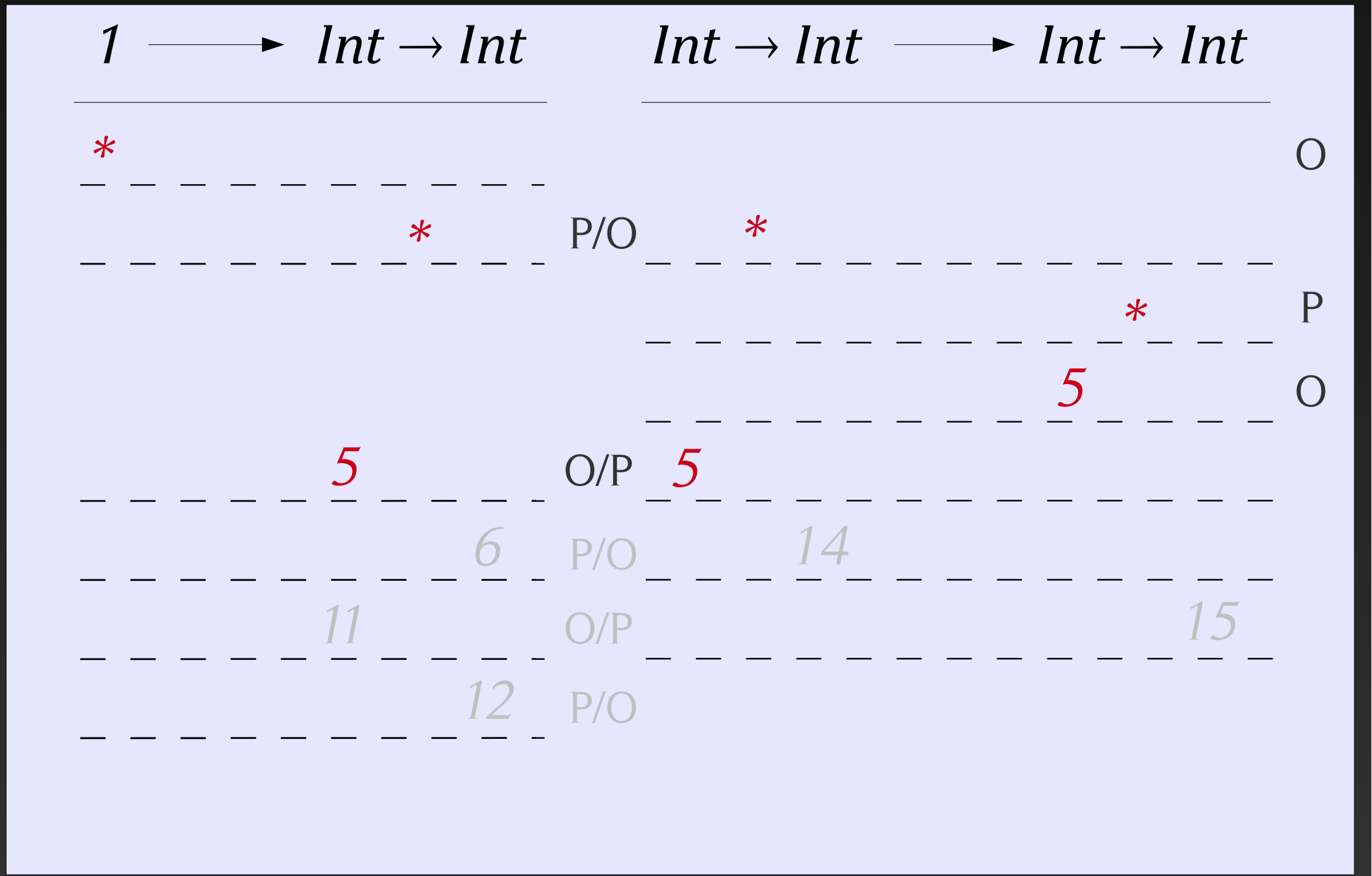
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

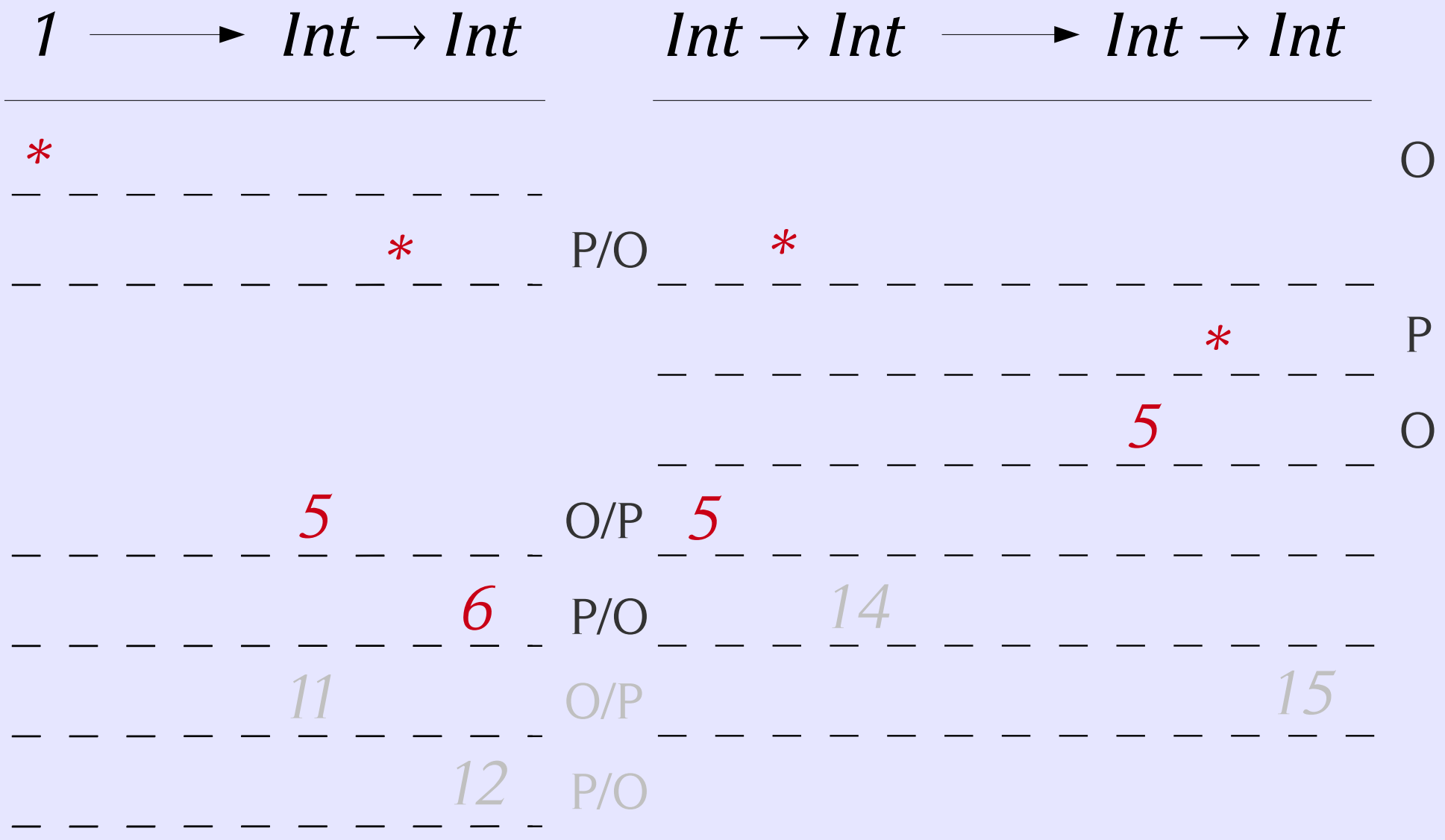
Composition



⊢ λx.x+1 : int ⟶ int

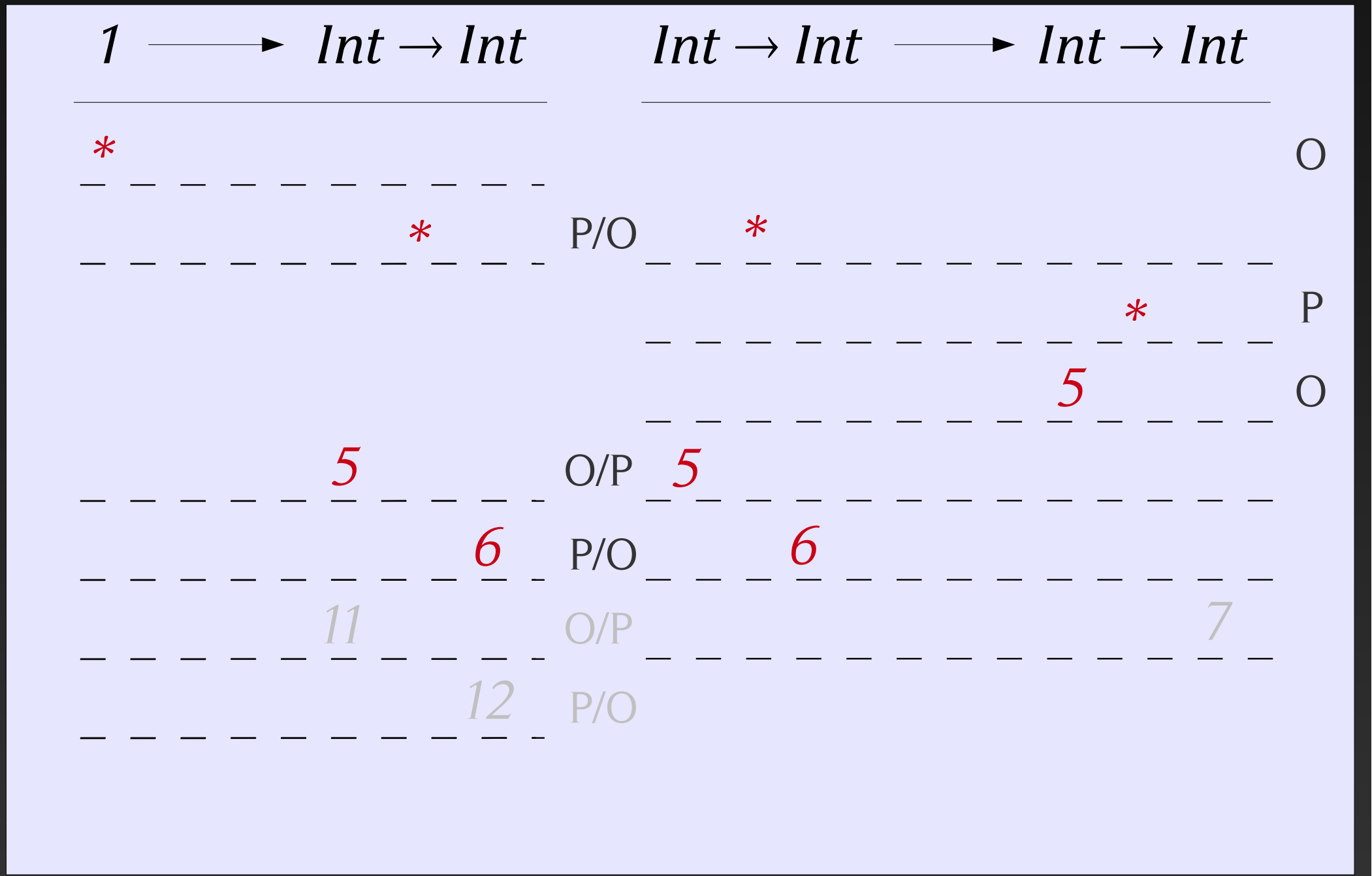
f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

Composition



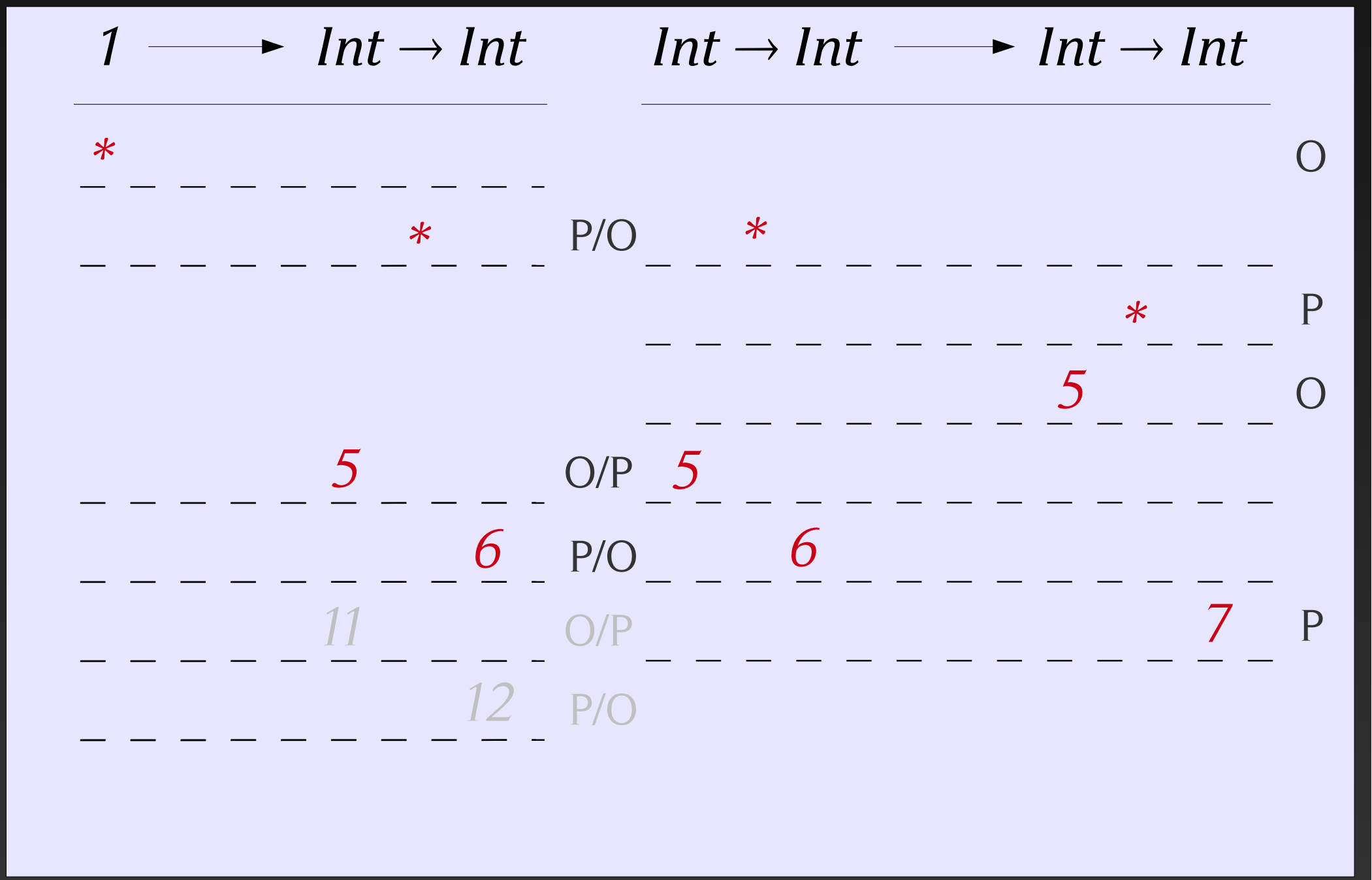
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

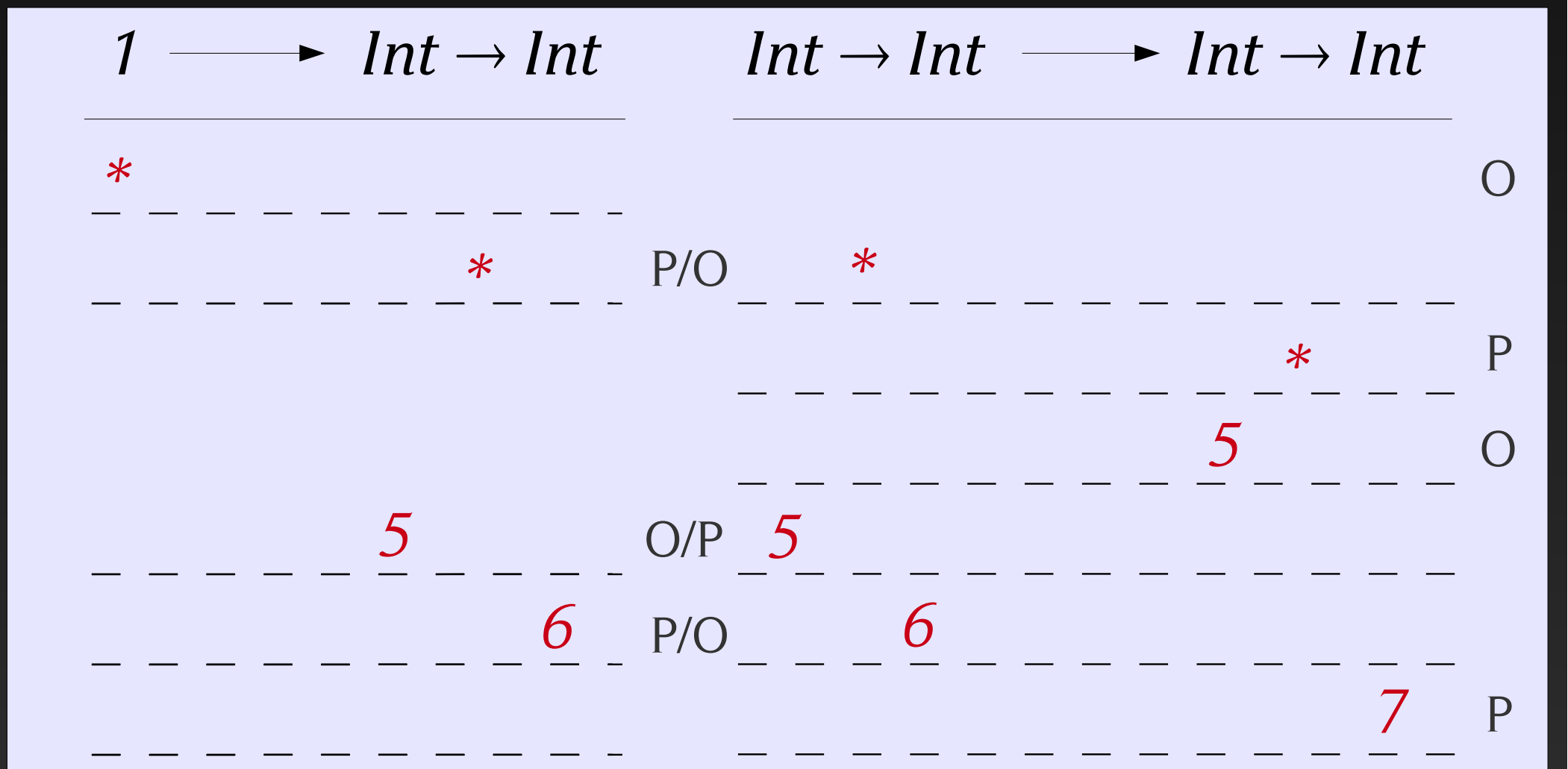
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

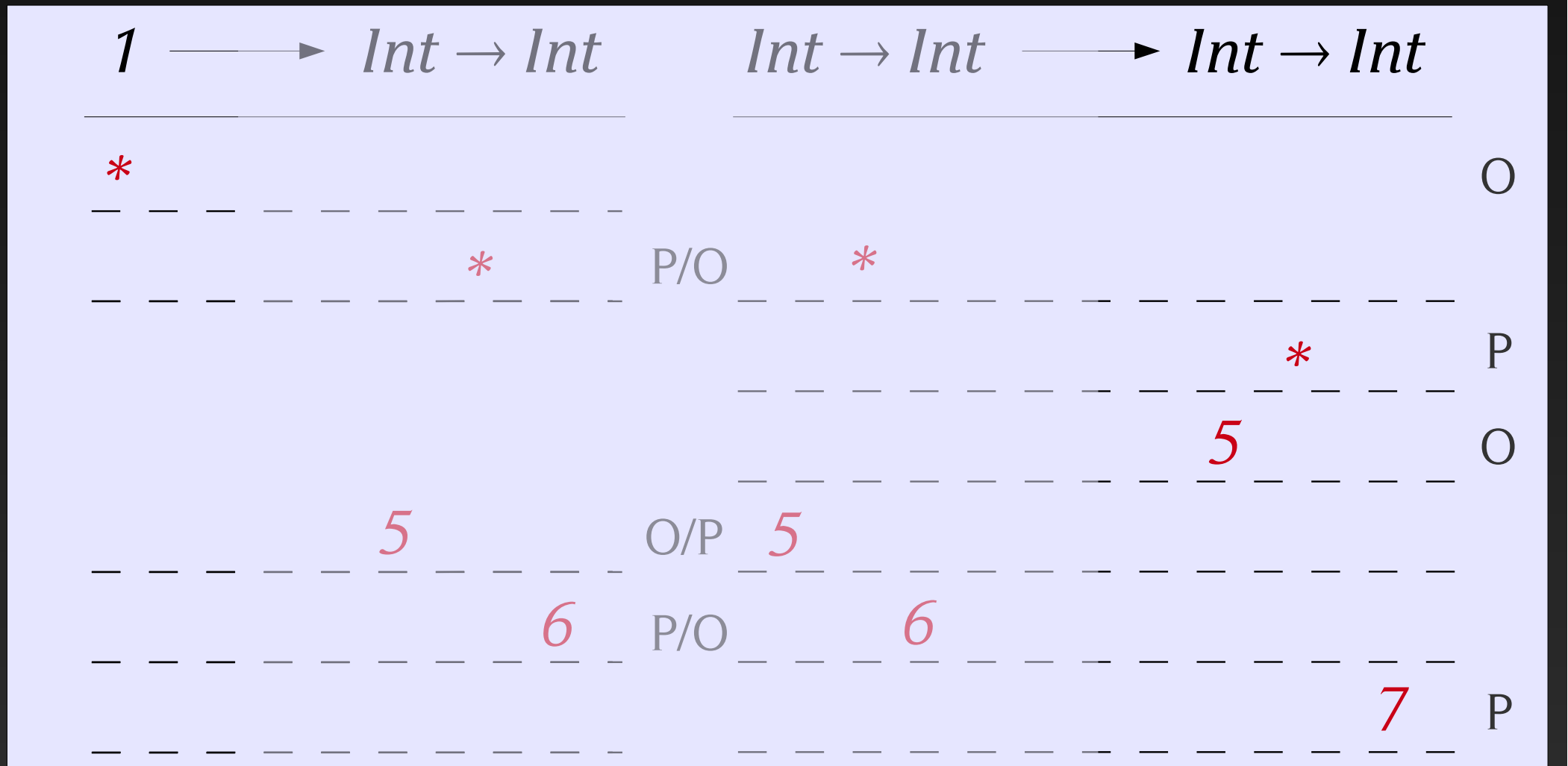
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

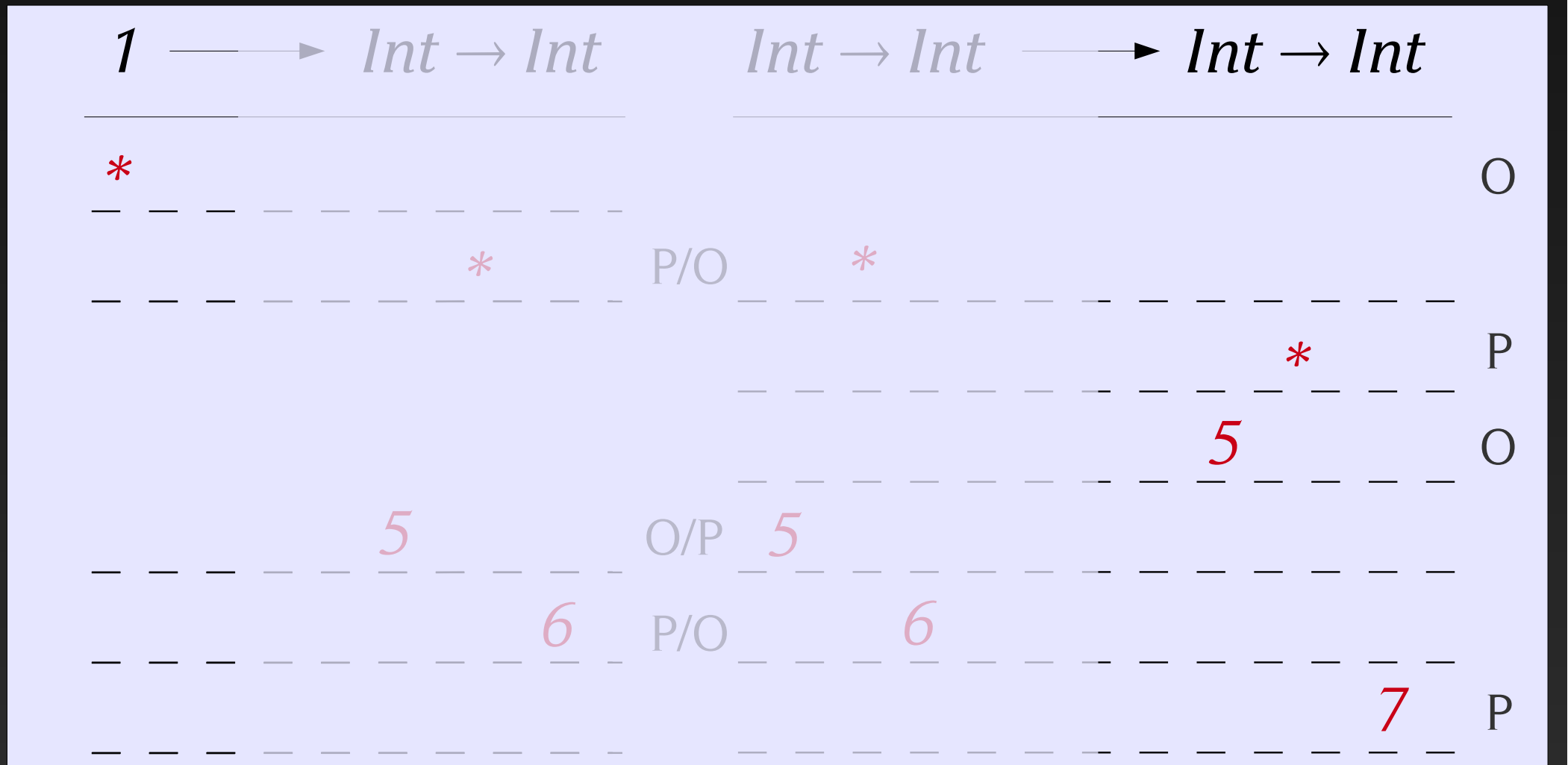
Composition



$\vdash \lambda x.x+1 : \mathtt{int} \longrightarrow \mathtt{int}$

$\mathtt{f} : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

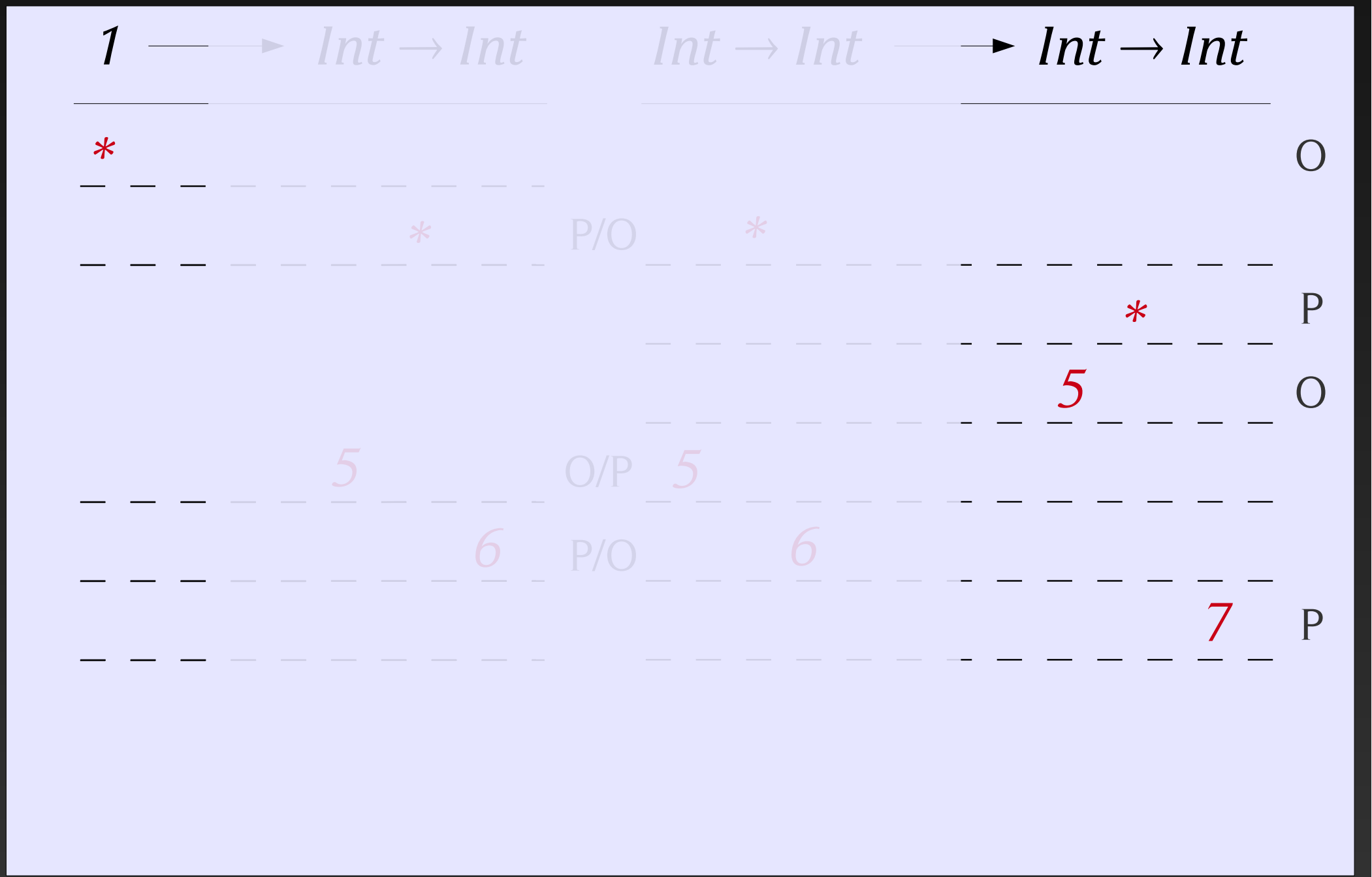
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

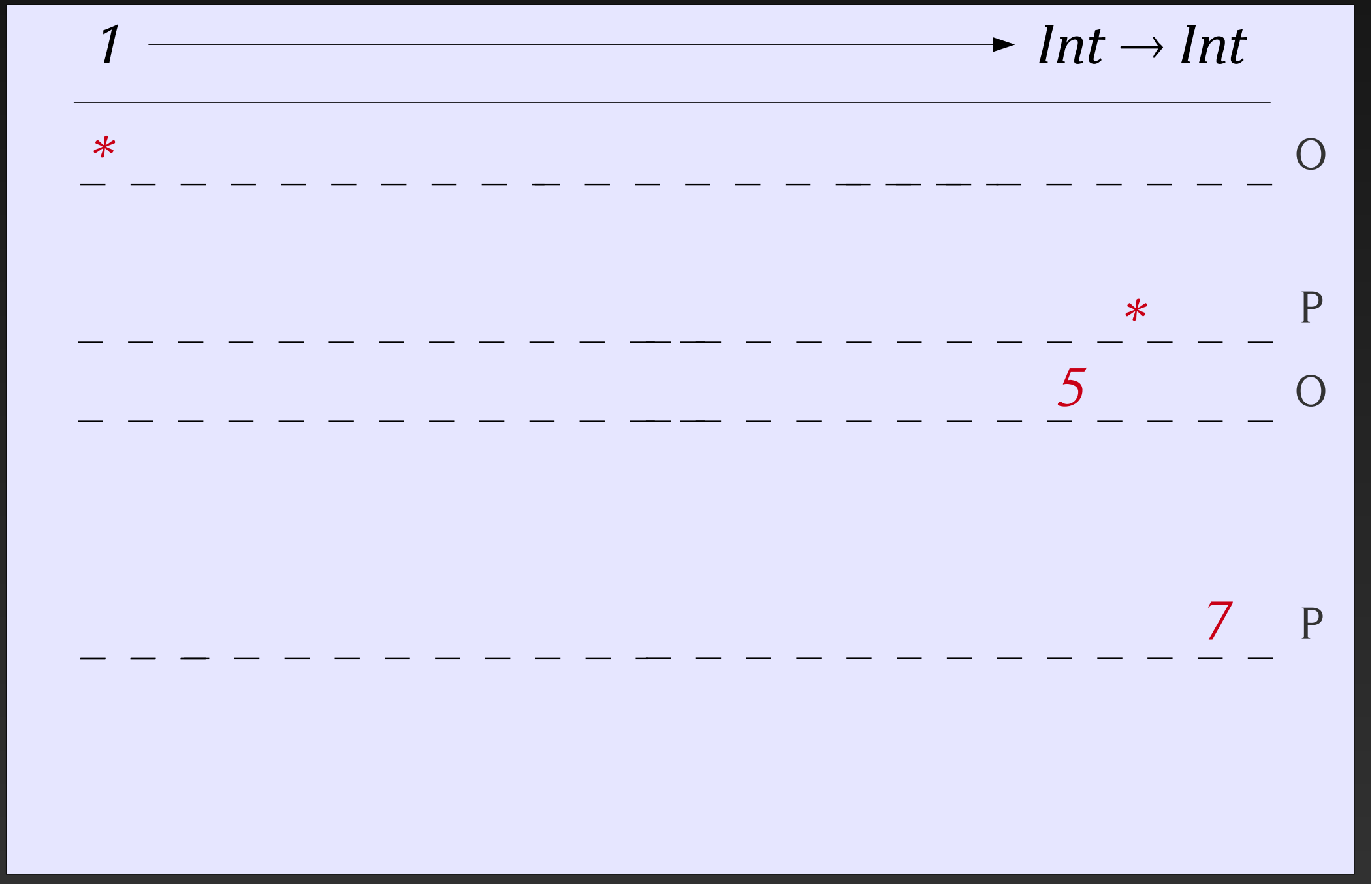
Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

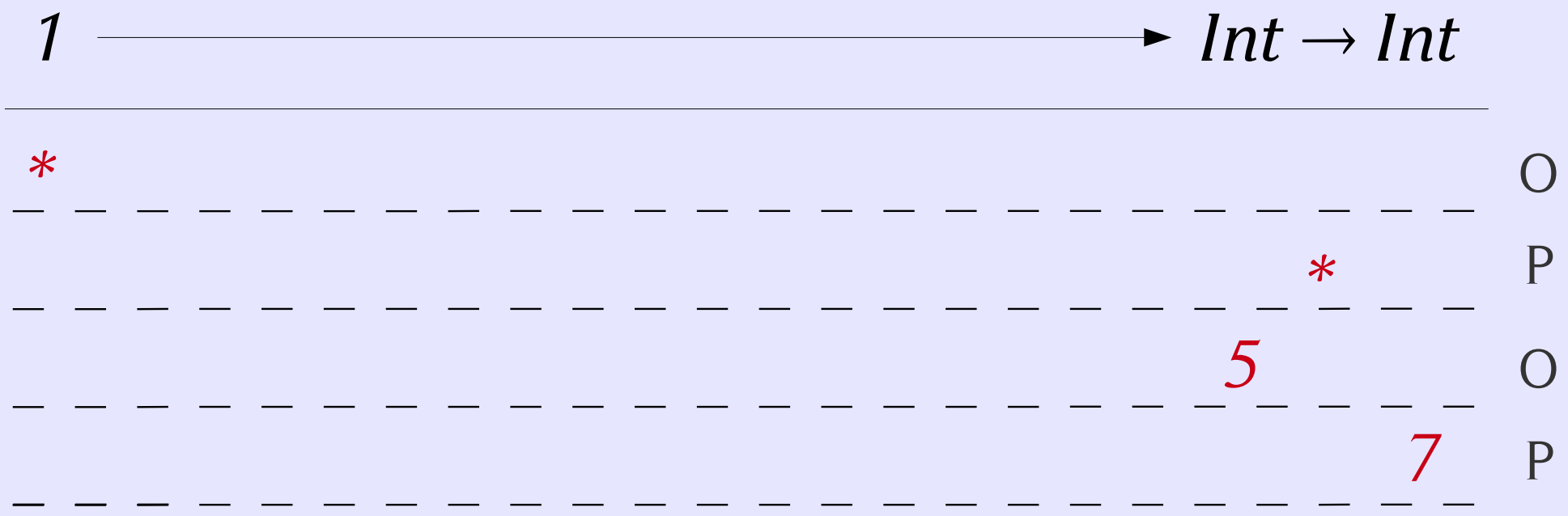
Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$

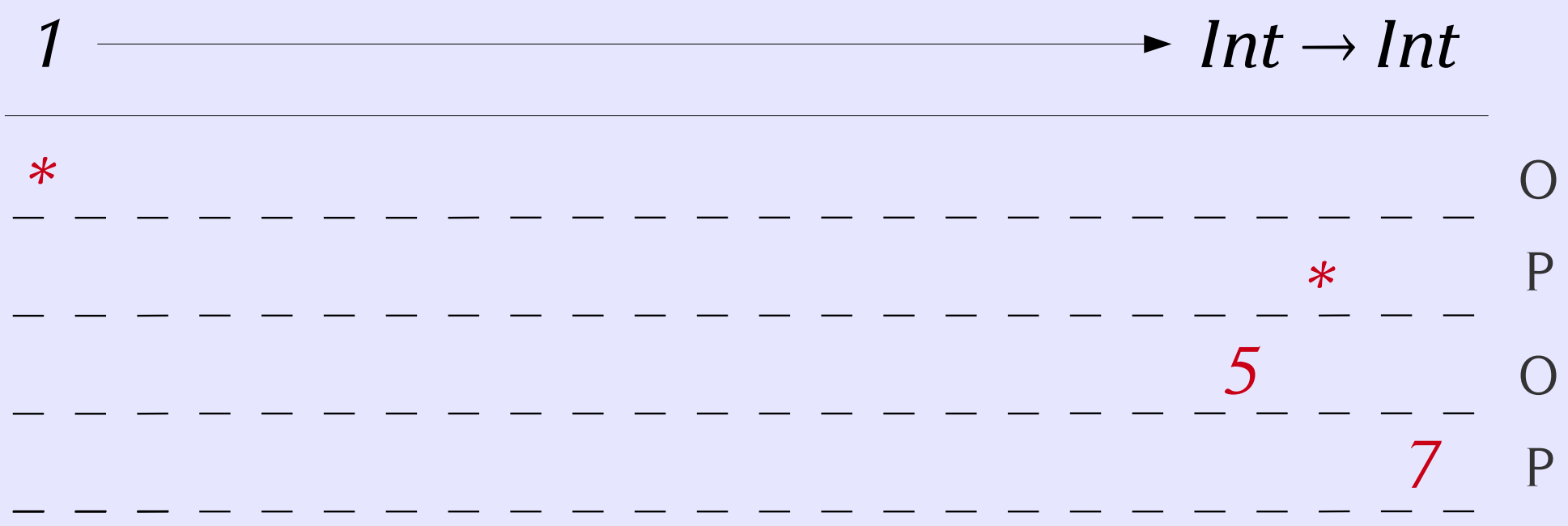
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$ $\quad f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition

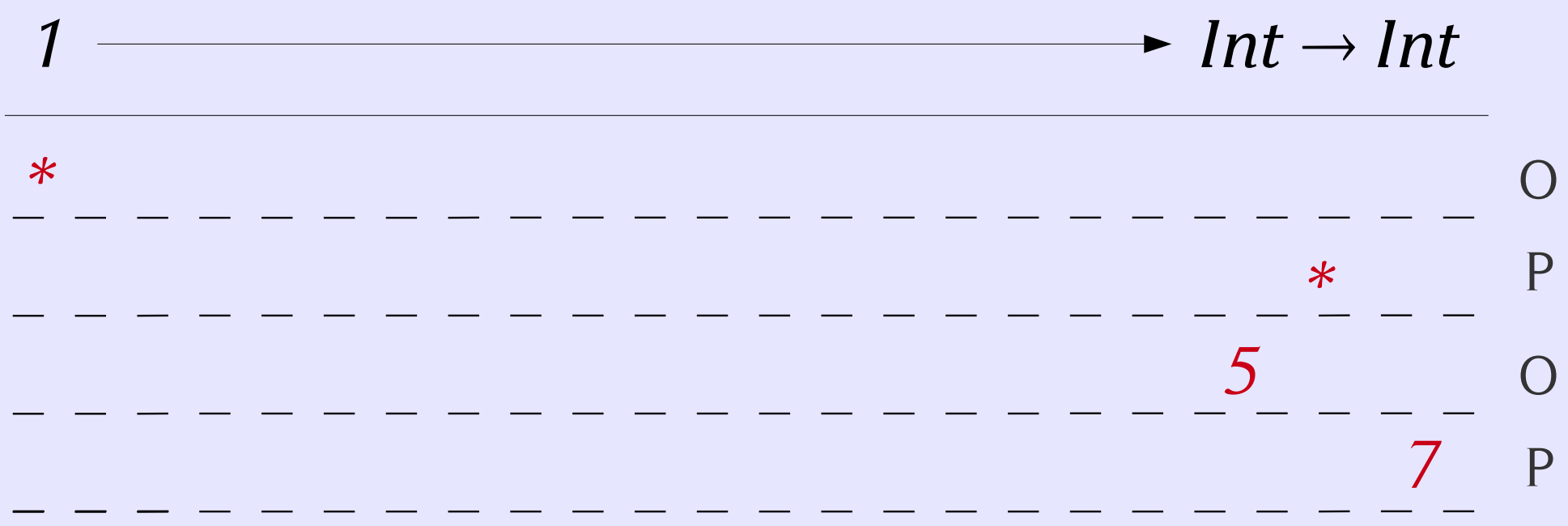


$\vdash \lambda x.x+1 : int \longrightarrow int$; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

$=$ $\vdash \lambda x.x+2 : int \longrightarrow int$

$\vdash \lambda x.x+1 : int \longrightarrow int$ $\quad$ $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



$\vdash \lambda x.x+1 : int \longrightarrow int$; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

$=$ $\vdash \lambda x.x+2 : int \longrightarrow int$

$$A \xrightarrow{\sigma} B \xrightarrow{\tau} C \quad = \quad A \xrightarrow{\sigma;\tau} C$$

Game Semantics

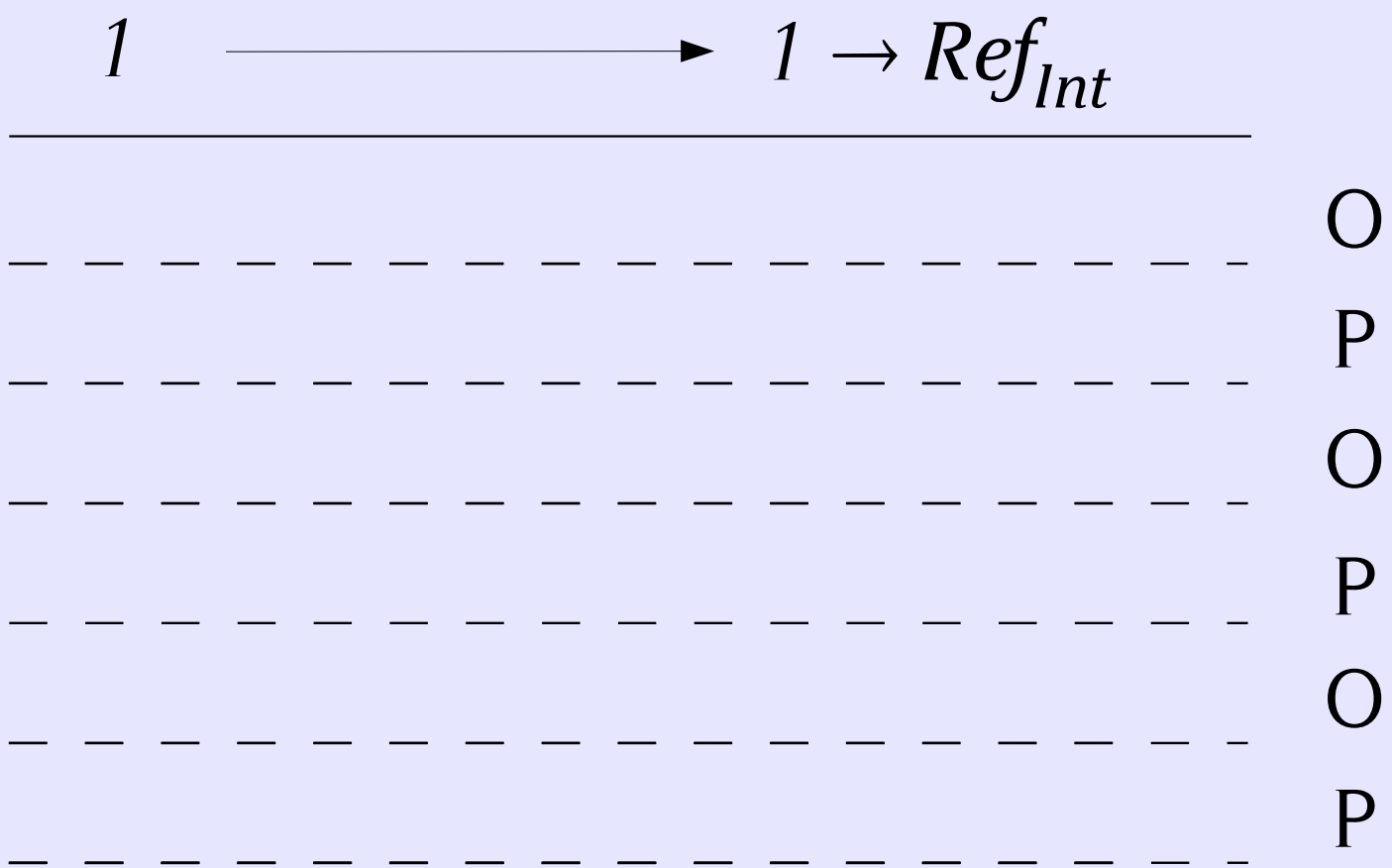
- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)
- Qualitative games
- Programs = *strategies* for Proponent
- Families (i.e. *categories*) of games

Nominal Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)
- Qualitative games
- Programs = *strategies* for Proponent
- Families (i.e. *categories*) of games
- Moves with names and stores

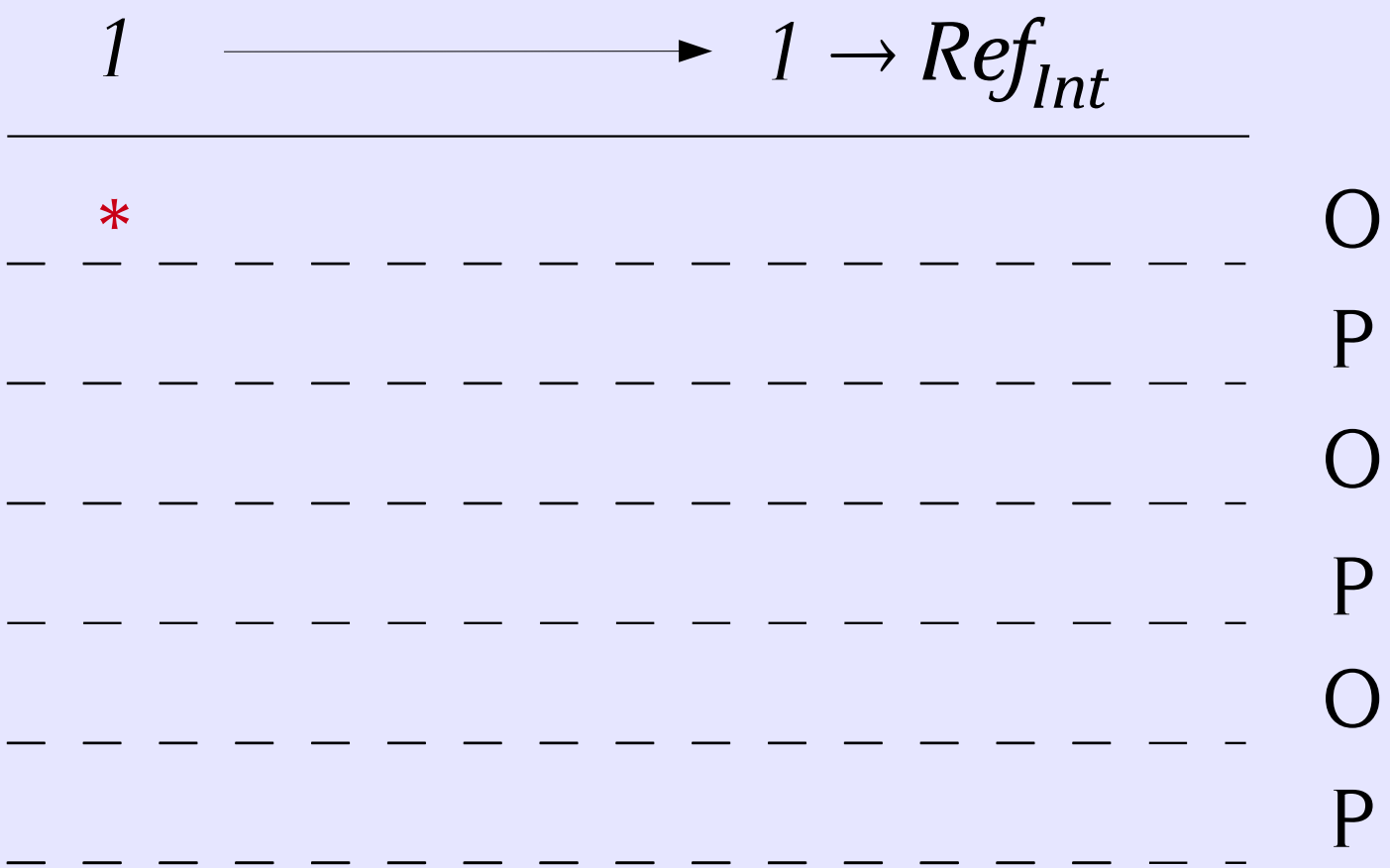
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



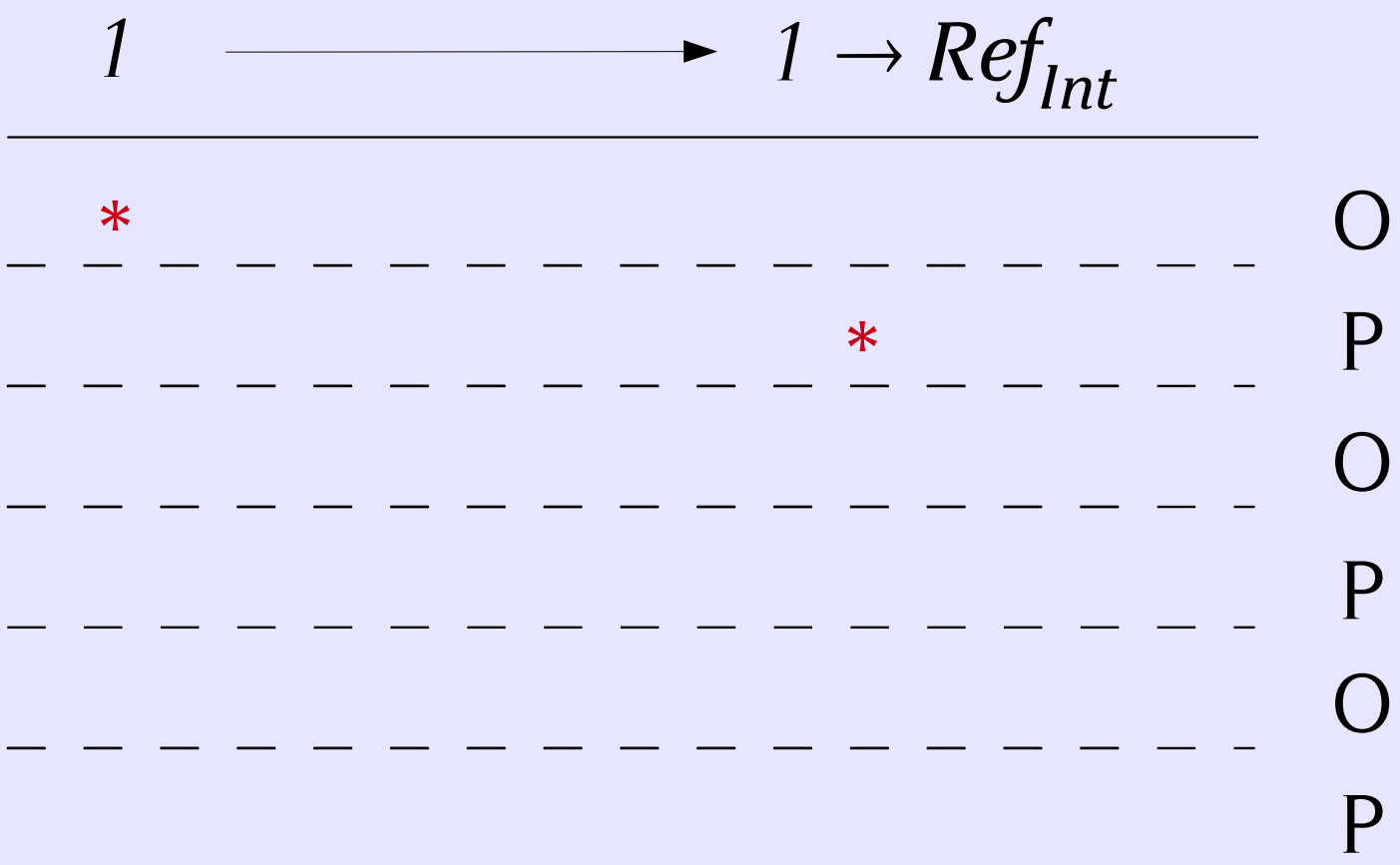
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



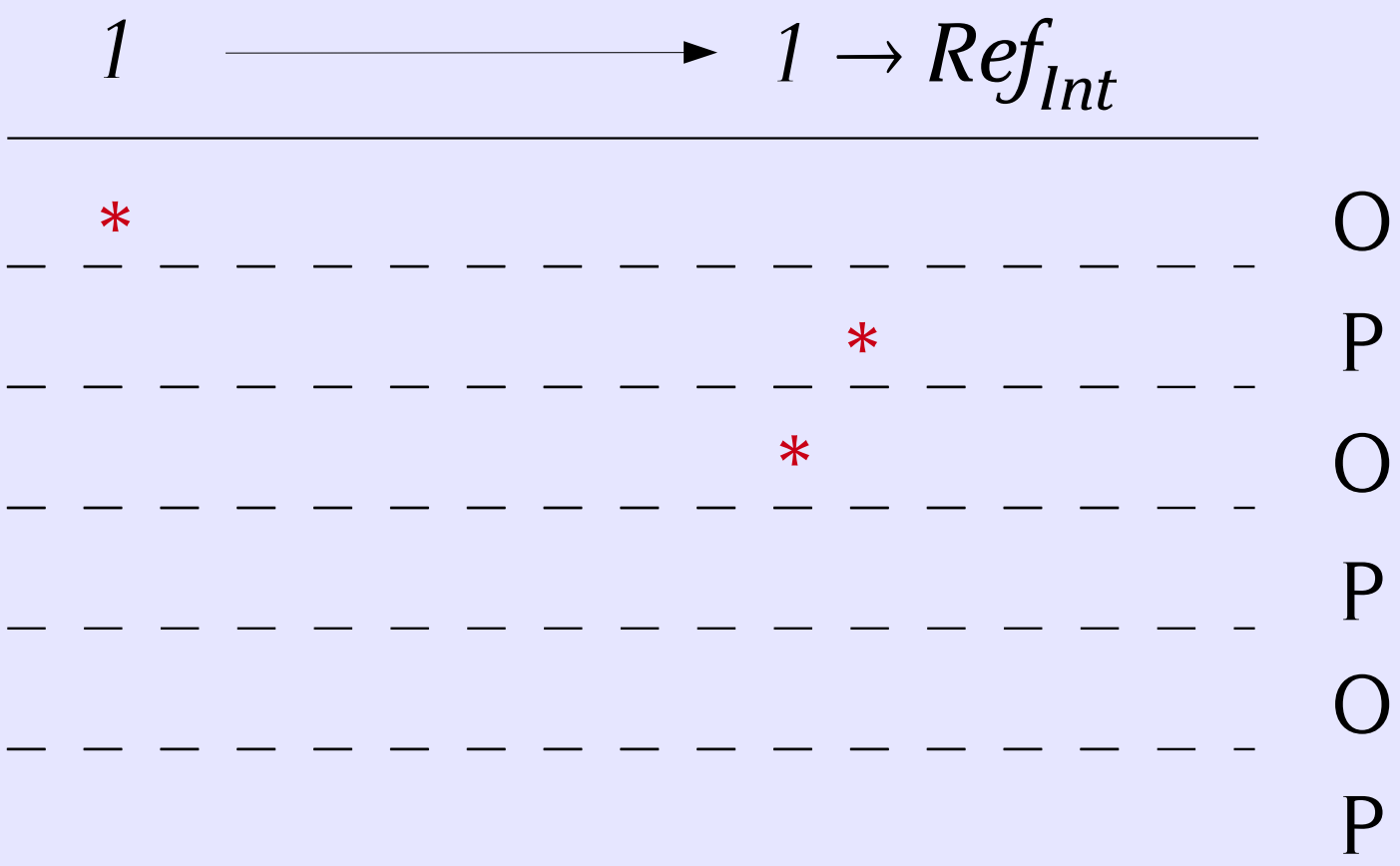
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



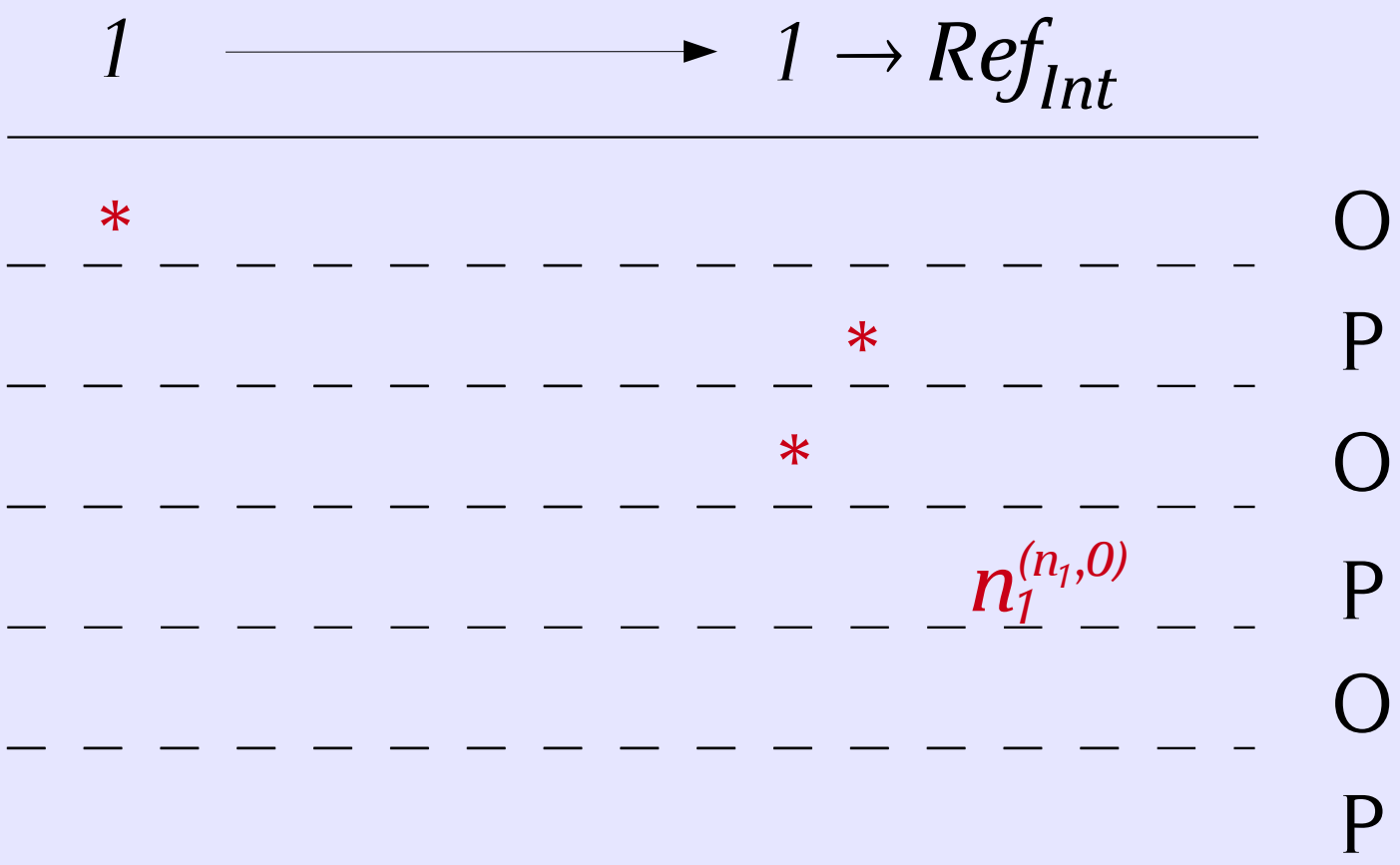
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



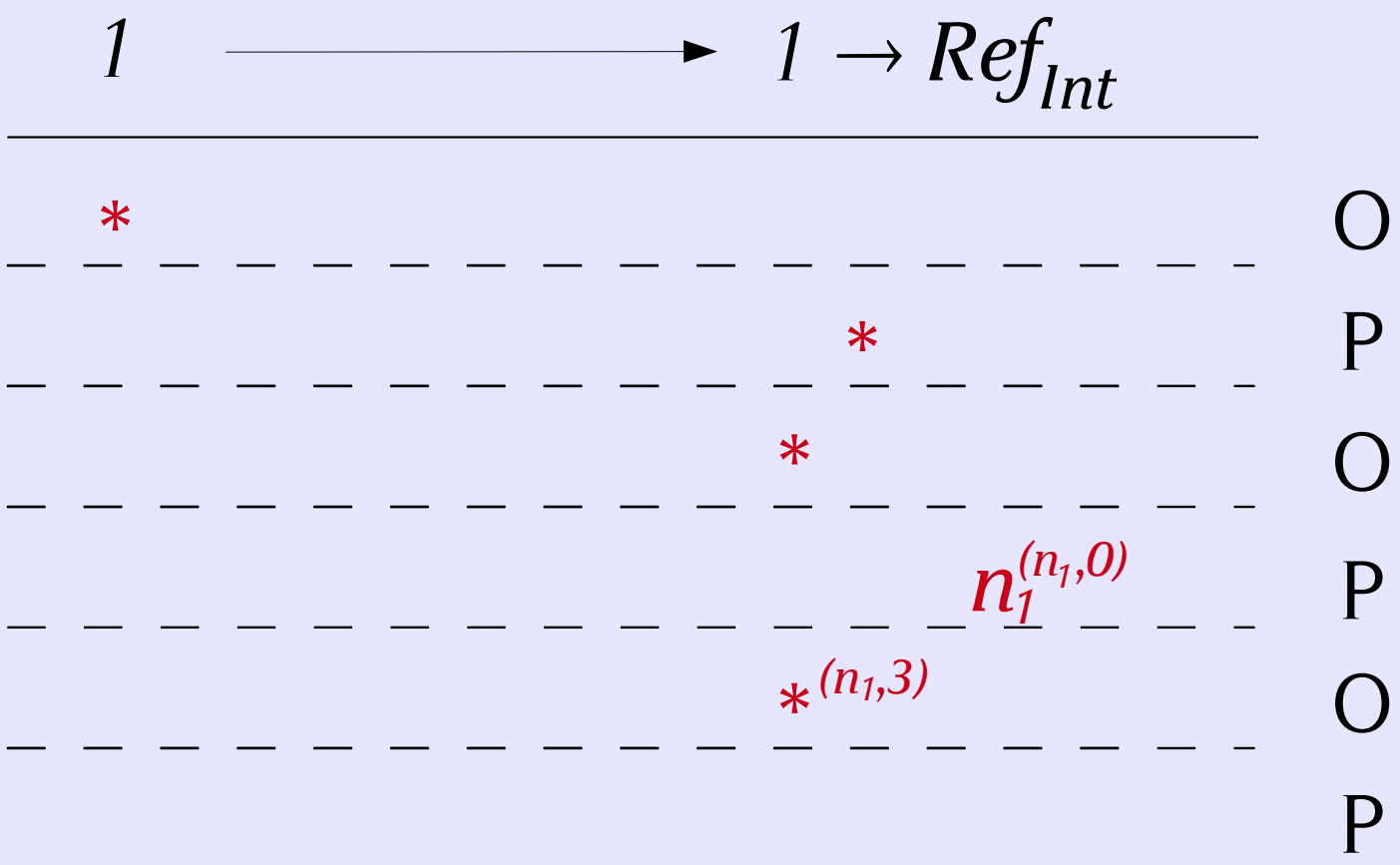
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



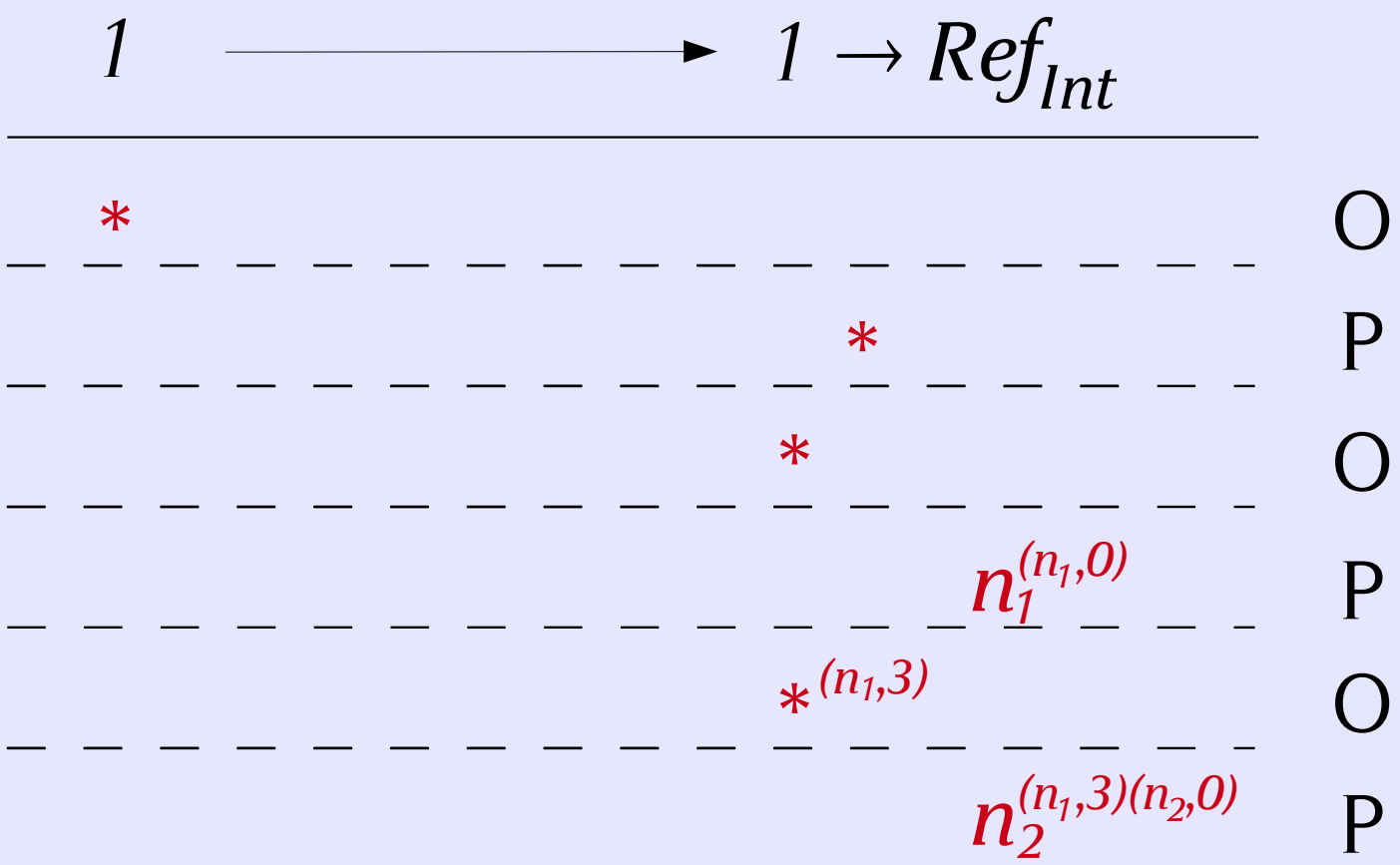
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



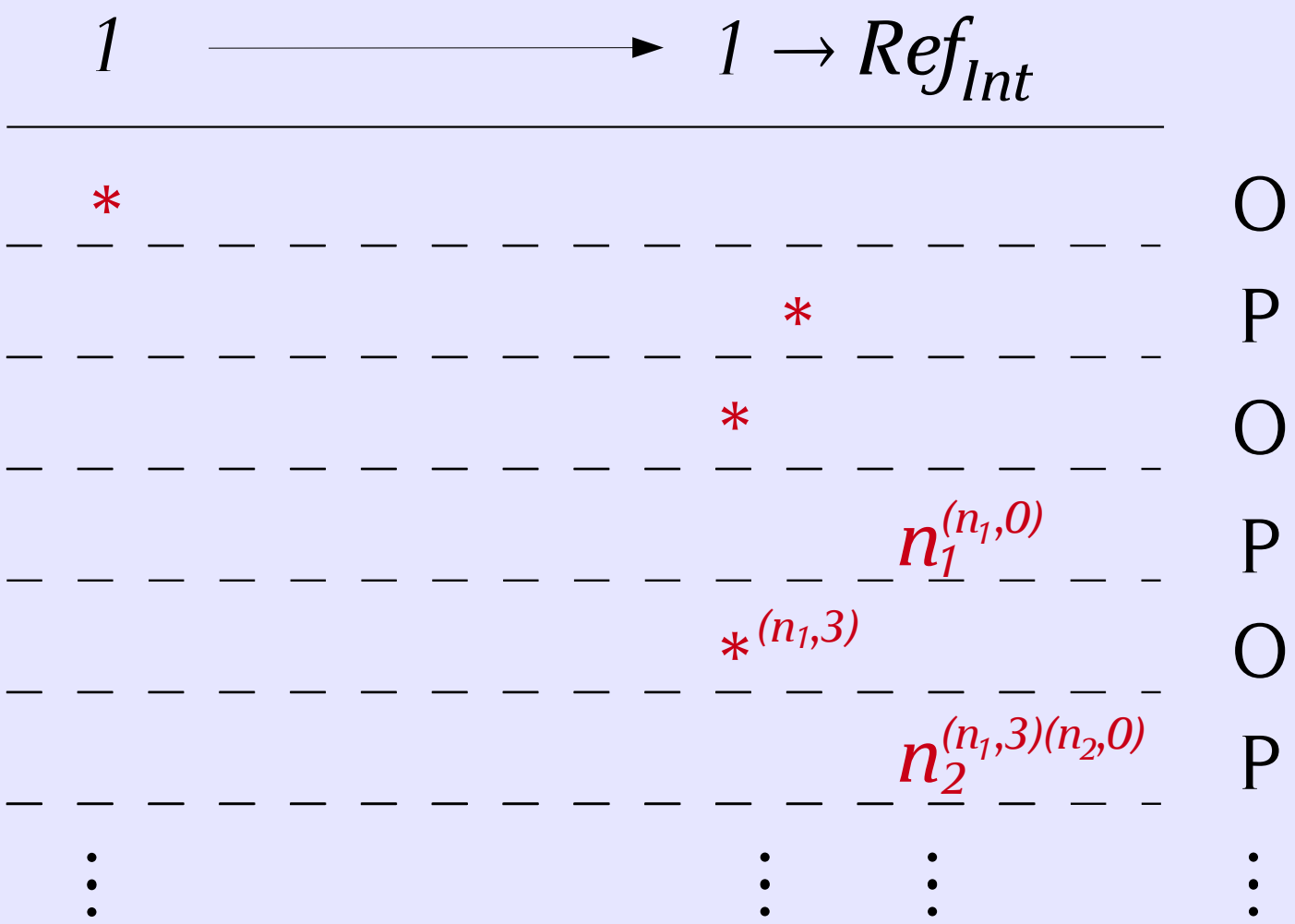
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



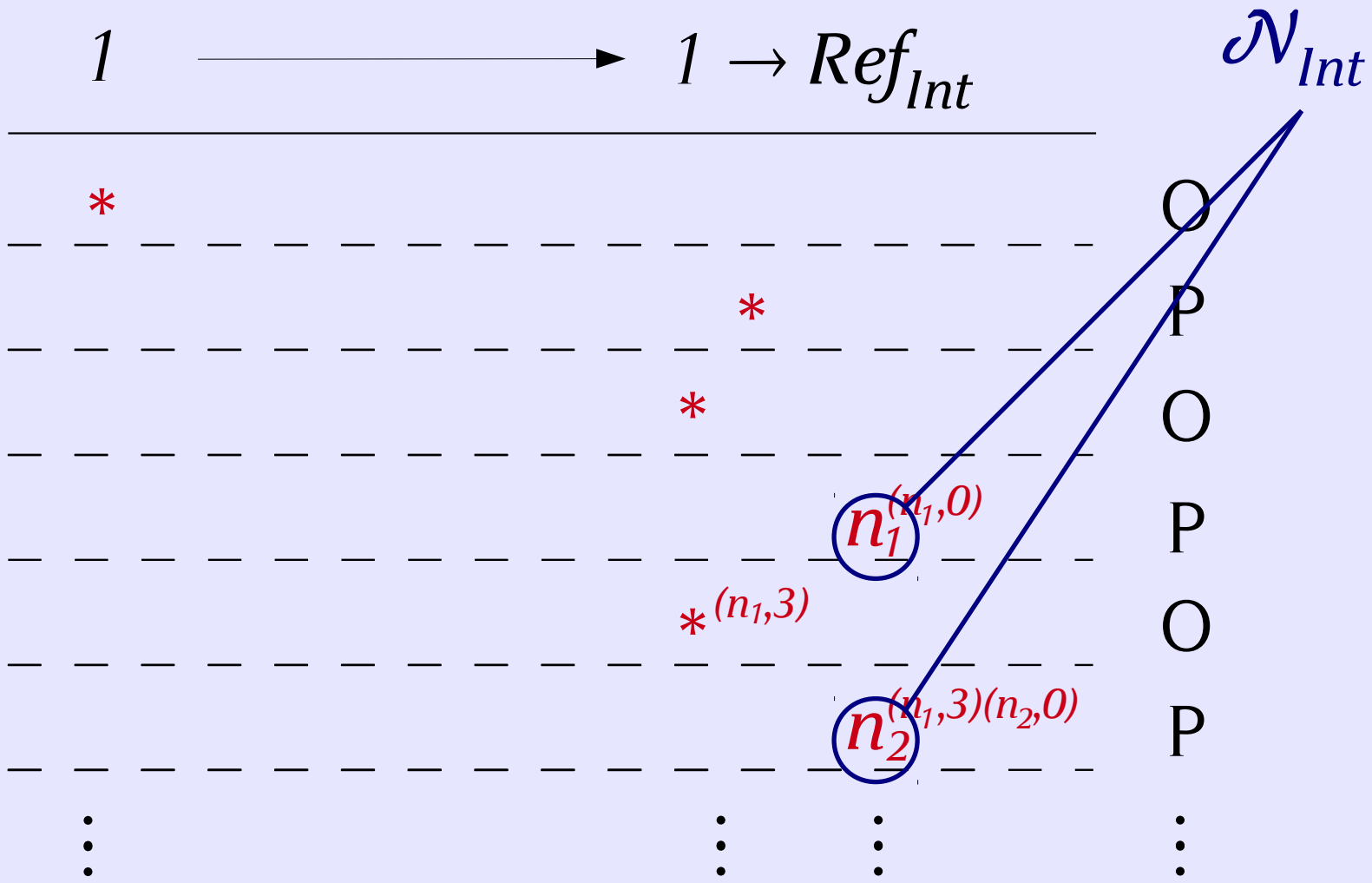
Example

$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



Example

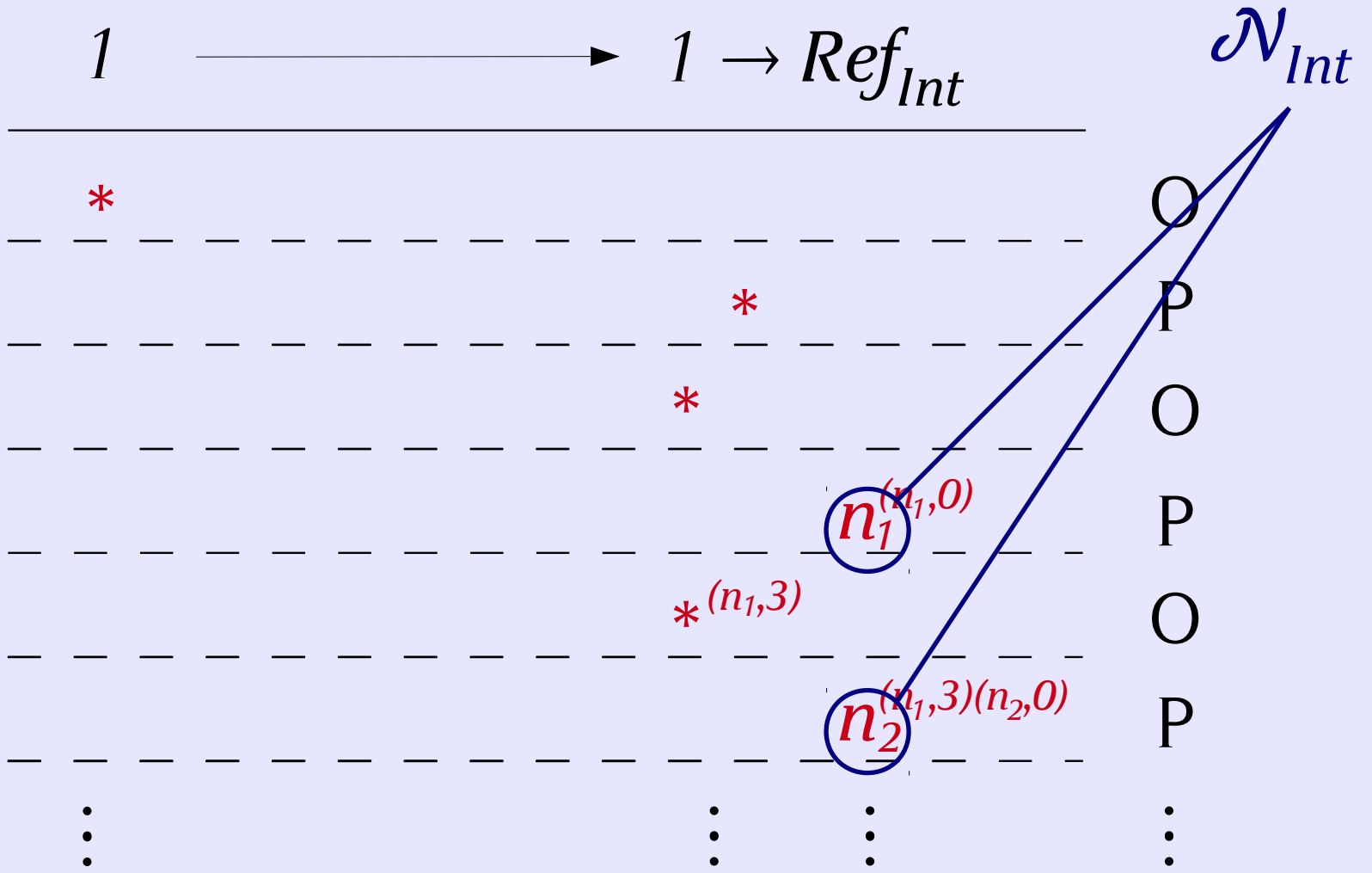
$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$



Example

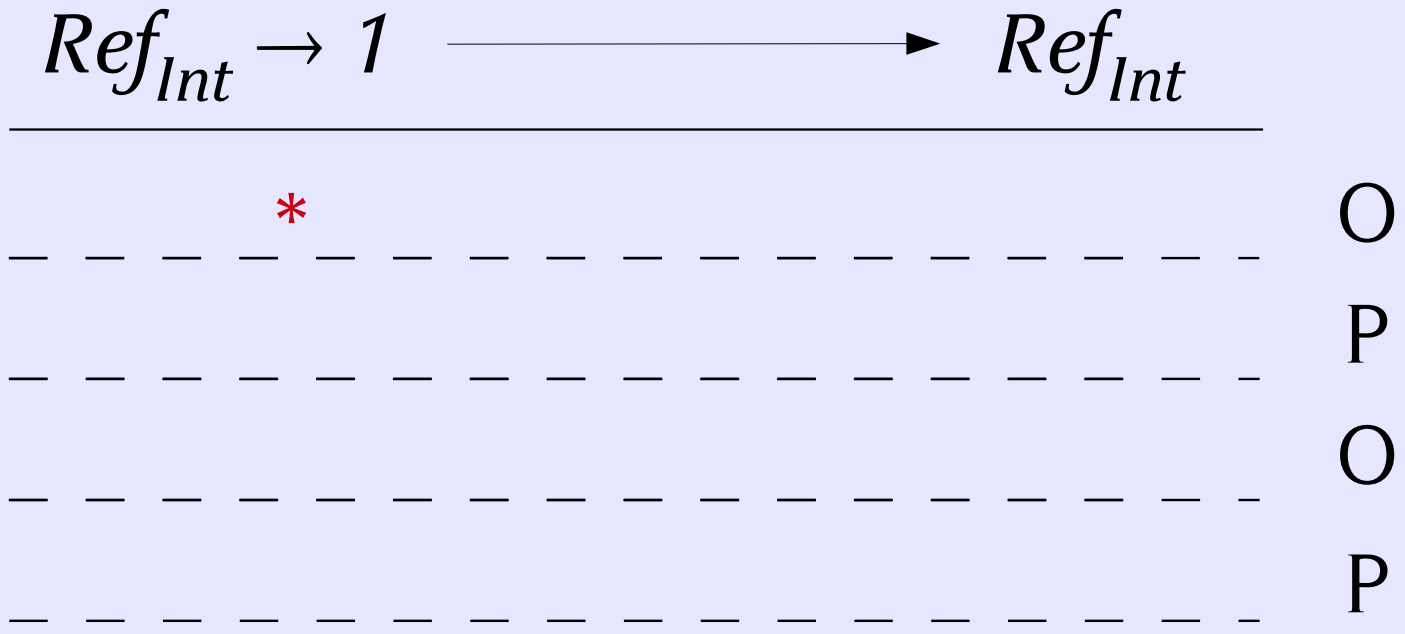
$\vdash \lambda x.\mathtt{ref}(0) : \mathtt{unit} \to \mathtt{refint}$

Use *nominal sets* over $\mathcal{N} = \bigcup_\theta \mathcal{N}_\theta$



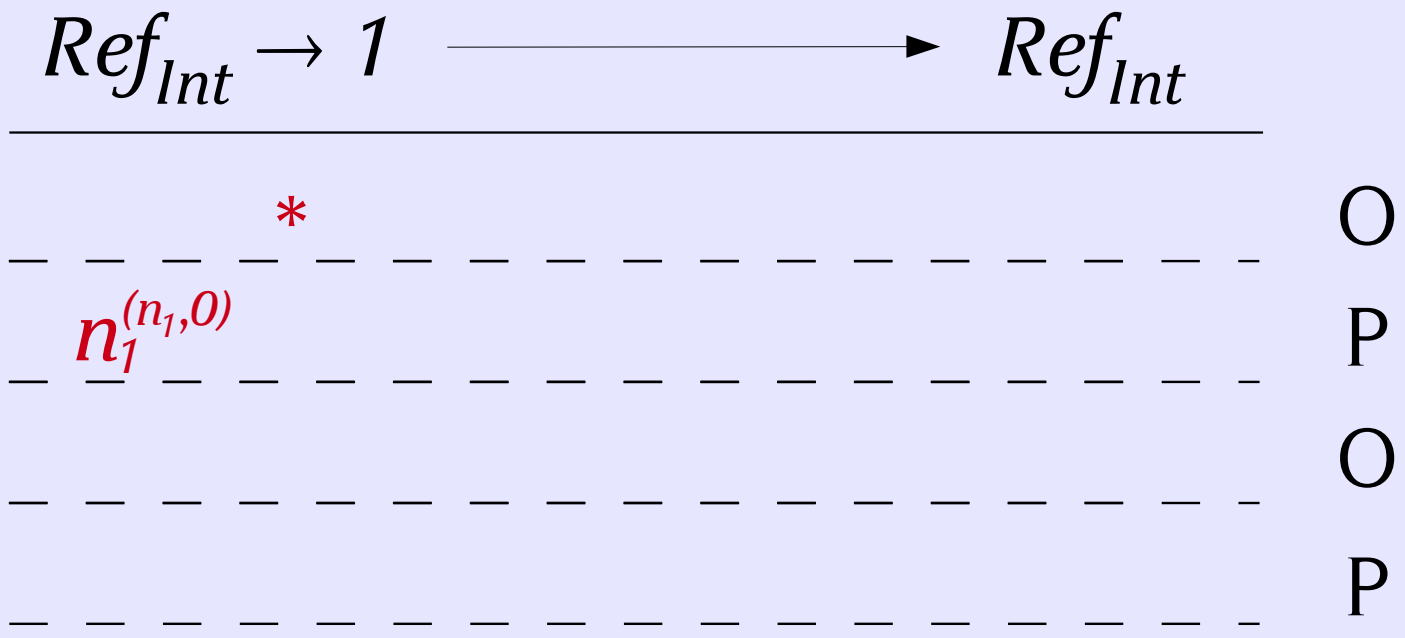
Example

`f:refint → unit ⊢ let x=ref(0) in (fx);x : refint`



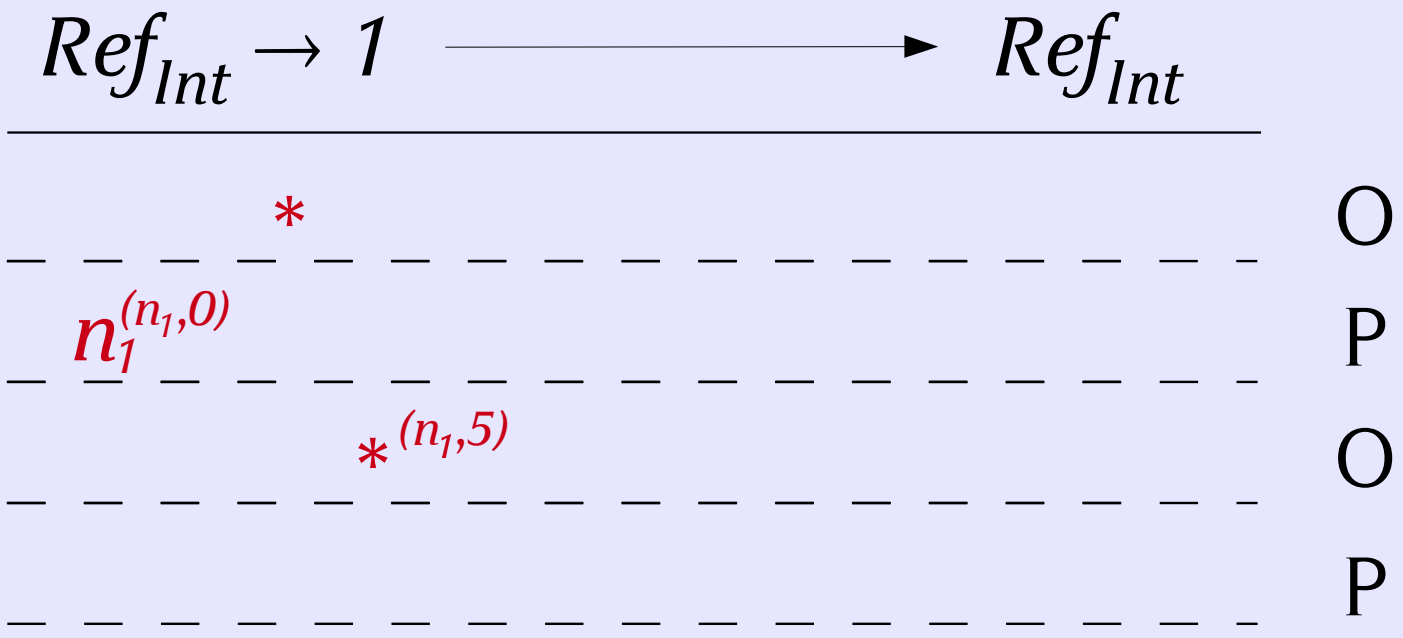
Example

`f:refint → unit ⊢ let x=ref(0) in (fx);x : refint`



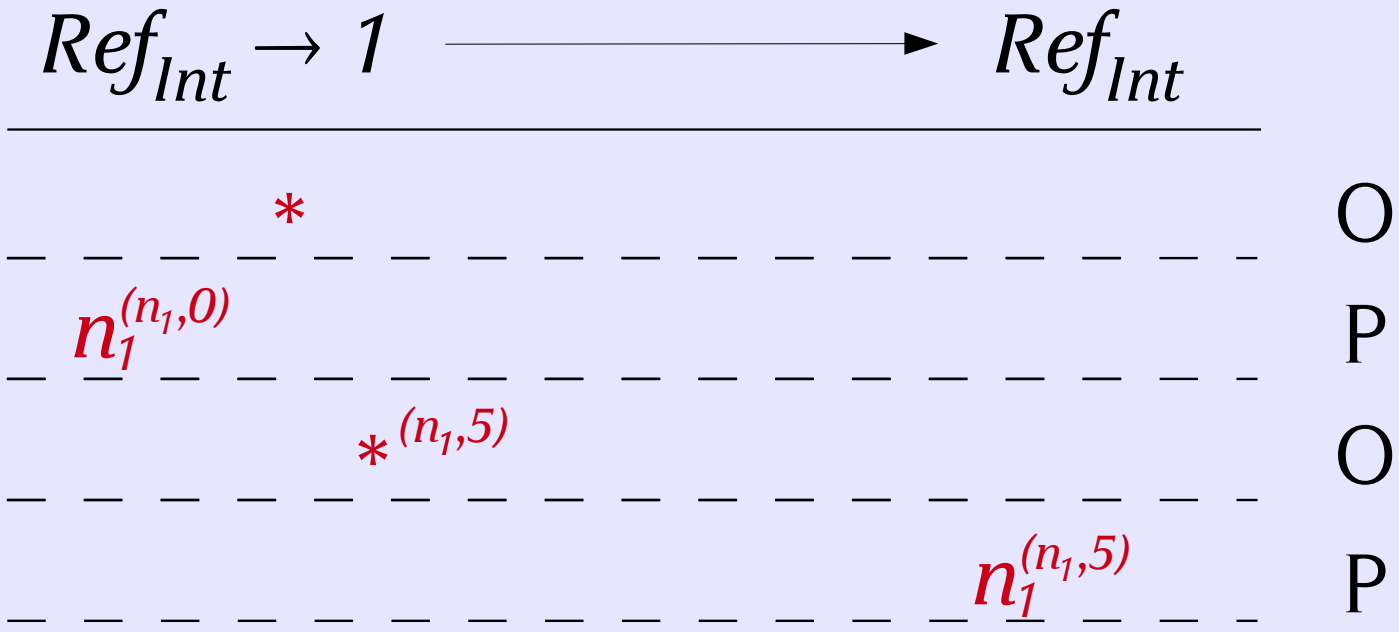
Example

`f:refint → unit ⊢ let x=ref(0) in (fx);x : refint`



Example

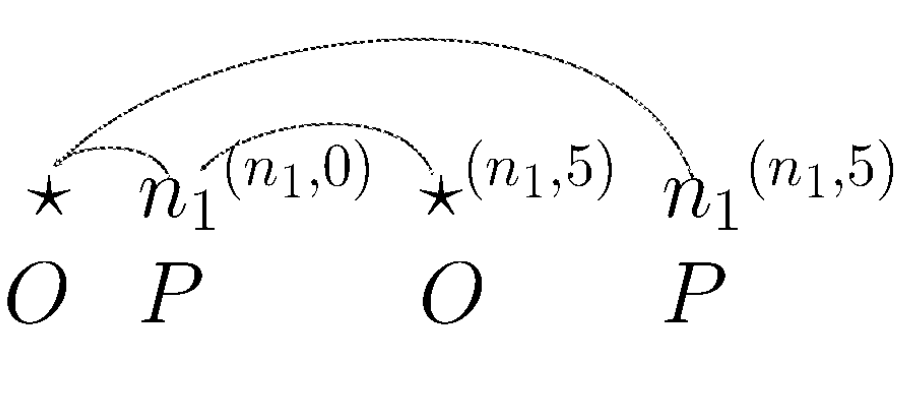
`f:refint → unit ⊢ let x=ref(0) in (fx);x : refint`



Example

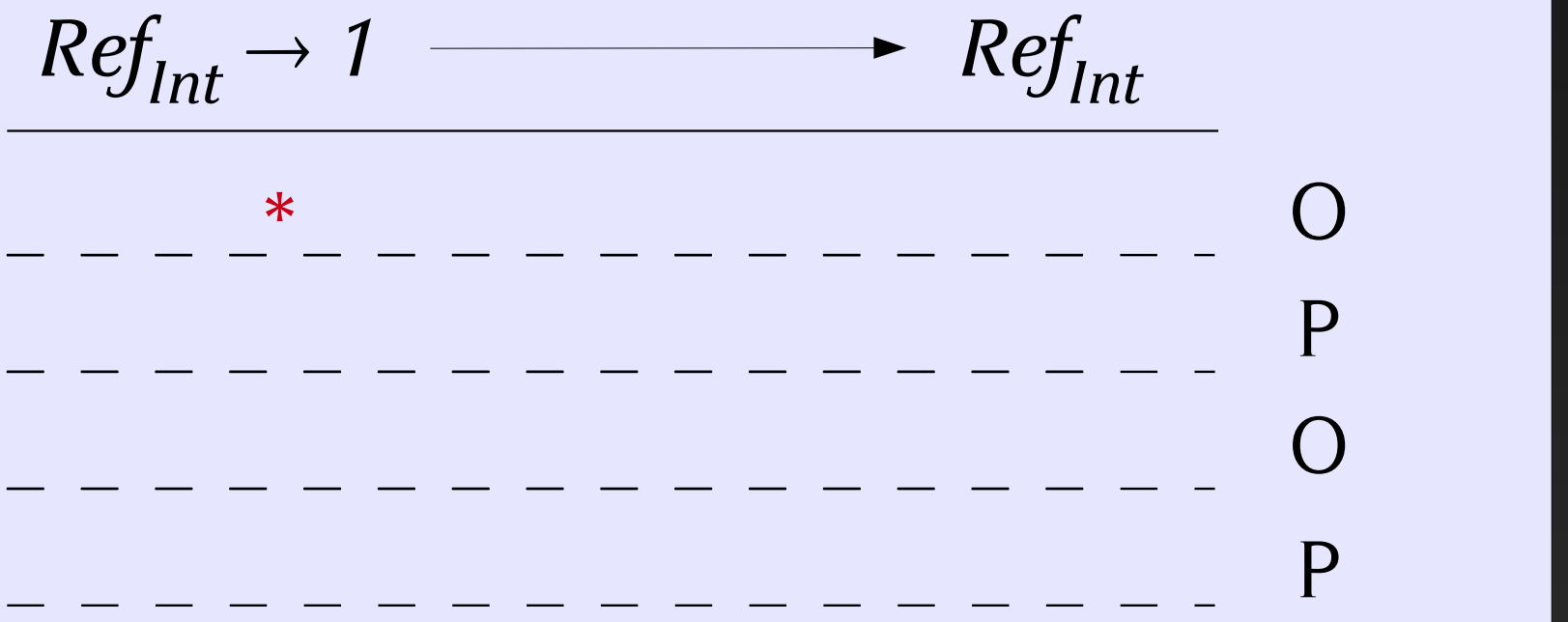
$\texttt{f:refint} \to \texttt{unit} \vdash \texttt{let x=ref(0) in (fx);x : refint}$

$Ref_{Int} \to 1 \longrightarrow Ref_{Int}$



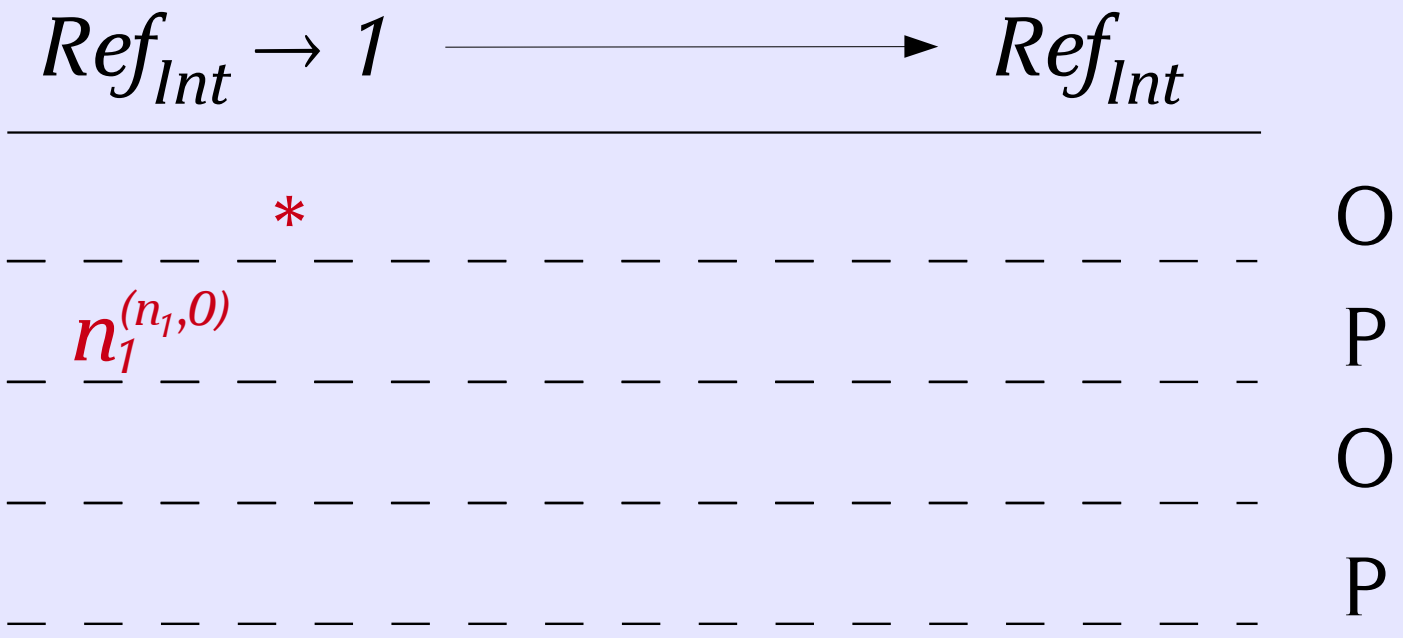
Example

```
f:refint → unit ⊢ let x=ref(0),y=ref(0)
                  in (fy); y:=!x; y : refint
```



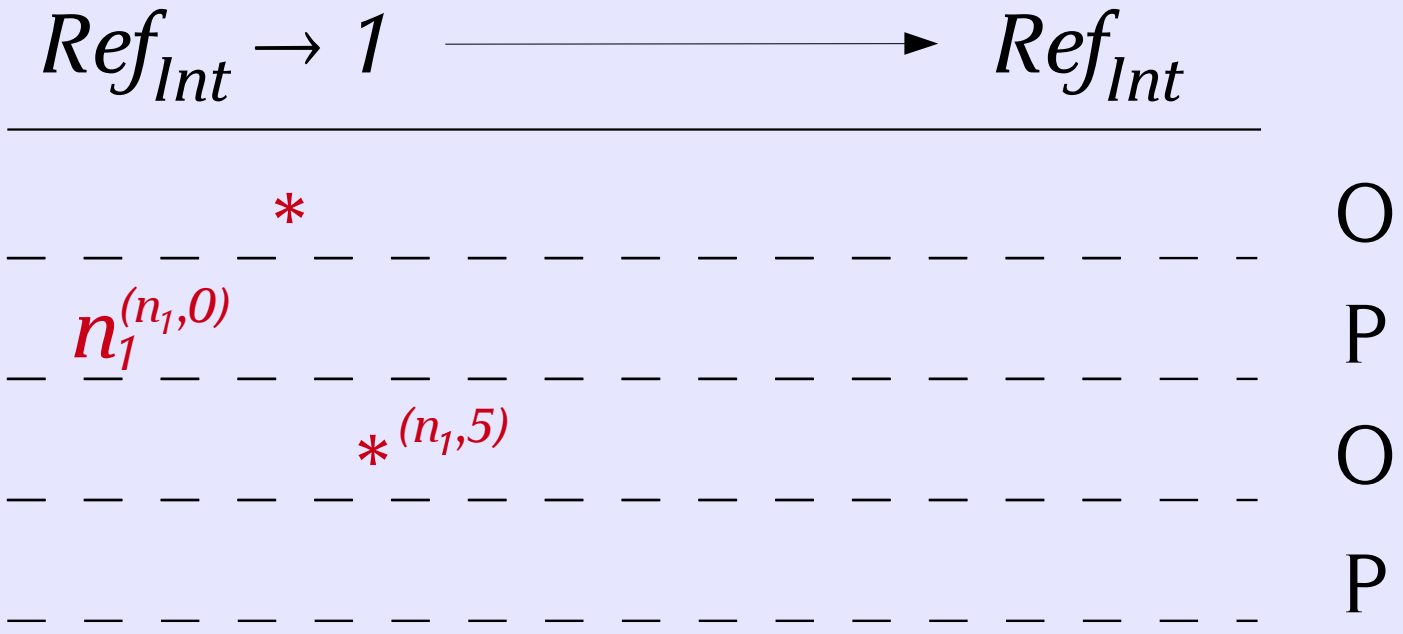
Example

```
f:refint → unit ⊢ let x=ref(0),y=ref(0)
                  in (fy); y:=!x; y : refint
```



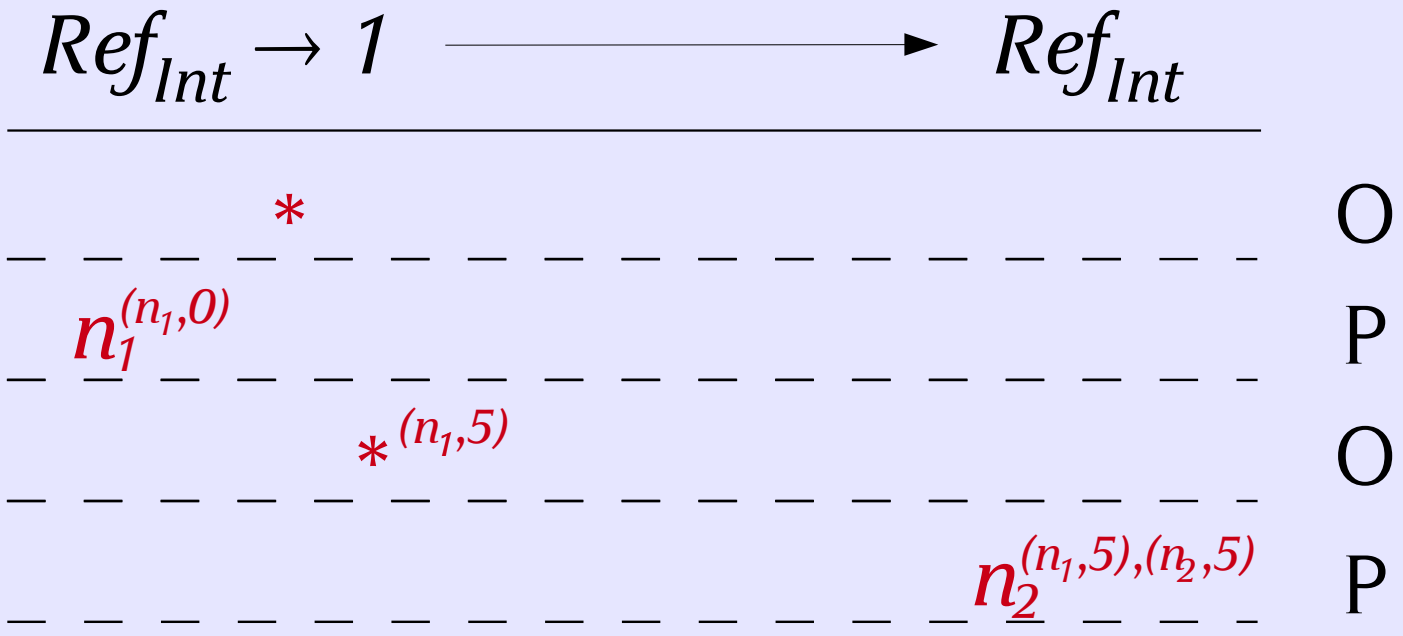
Example

```
f:refint → unit ⊢ let x=ref(0),y=ref(0)
                  in (fy); y:=!x; y : refint
```



Example

```
f:refint → unit ⊢ let x=ref(0),y=ref(0)
                  in (fy); y:=!x; y : refint
```

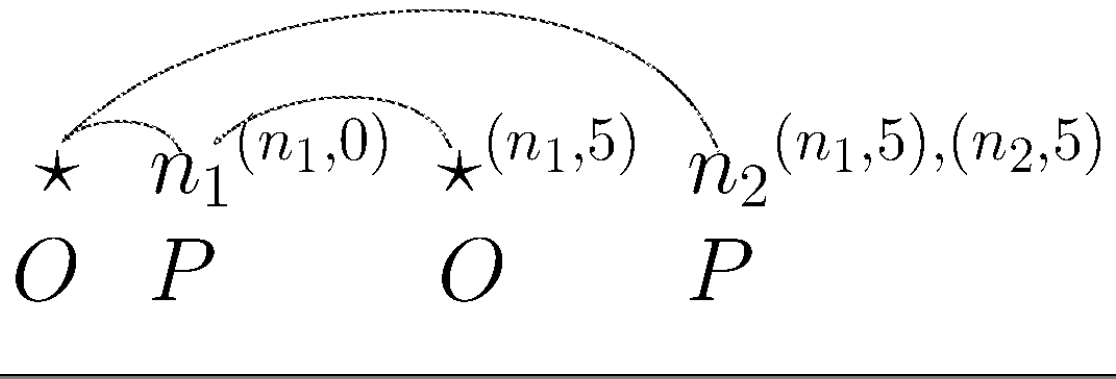


Example

```
f:refint → unit ⊢ let x=ref(0),y=ref(0)
                  in (fy); y:=!x; y : refint
```

$$Ref_{Int} \to 1 \longrightarrow Ref_{Int}$$

$*$	O
$n_1^{(n_1,0)}$	P
$*^{(n_1,5)}$	O
$n_2^{(n_1,5),(n_2,5)}$	P



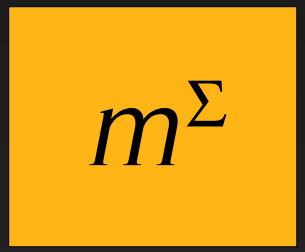
Our model

Built using game semantics

Uses a built-in notion of names and games which carry the store around

Moves-with-stores

Store is passed around in moves



Content of Σ : pairs (*name*,*value*)

Moves-with-stores

Store is passed around in moves

$$m^{\Sigma}$$

Content of Σ : pairs (*name*,*value*)

$$(n_{\texttt{unit}},*) \quad (n_{\texttt{int}},5) \quad (n_{\texttt{ref}\,\theta},n_{\theta}) \quad (n_{\theta\to\theta'},*)$$

Moves-with-stores

Store is passed around in moves

$$m^{\Sigma}$$

Content of Σ : pairs (*name*,*value*)

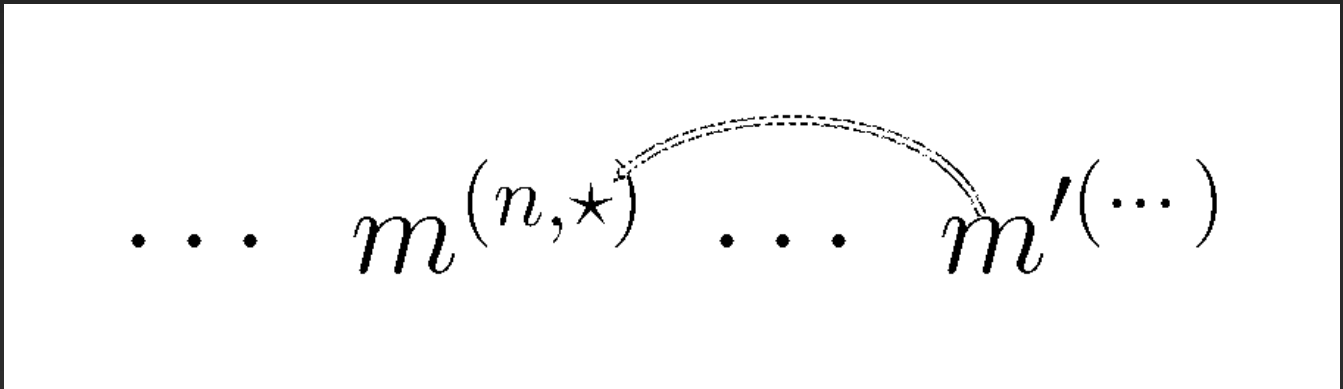
$$(n_{\texttt{unit}},*) \quad (n_{\texttt{int}},5) \quad (n_{\texttt{ref}\theta},n_{\theta}) \quad (n_{\theta\to\theta'},*)$$

```
# let r3 = ref(fun(x:intref) -> x == r1);;
val r3 : (int ref -> bool) ref = { contents = <fun> }
```

Higher-order store

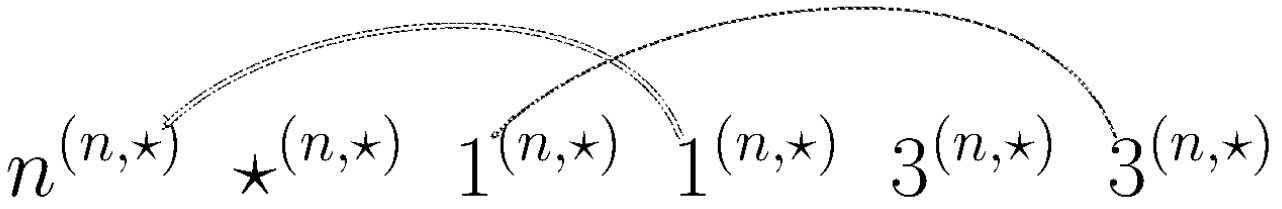
$$(n_{\theta \to \theta'}, *)$$

HO values not revealed for free
but need to be examined



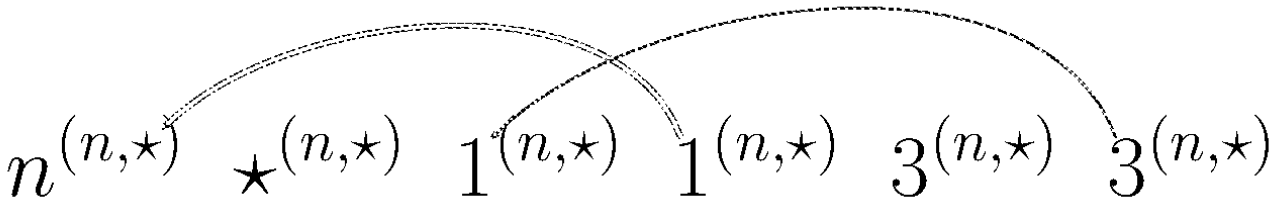
Higher-order store

$$x : \mathsf{ref}(\mathsf{int} \to \mathsf{int}) \vdash !x : \mathsf{int} \to \mathsf{int}$$

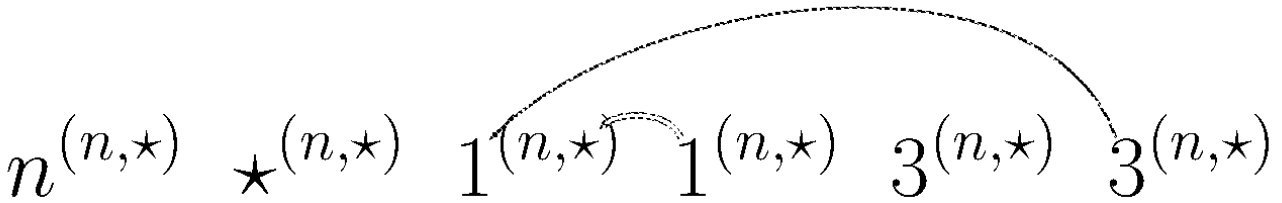


Higher-order store

$$x : \mathsf{ref}(\mathsf{int} \to \mathsf{int}) \vdash !x : \mathsf{int} \to \mathsf{int}$$



$$x : \mathsf{ref}(\mathsf{int} \to \mathsf{int}) \vdash \lambda h^{\mathsf{int}}.(!x)h : \mathsf{int} \to \mathsf{int}$$



Composition refined

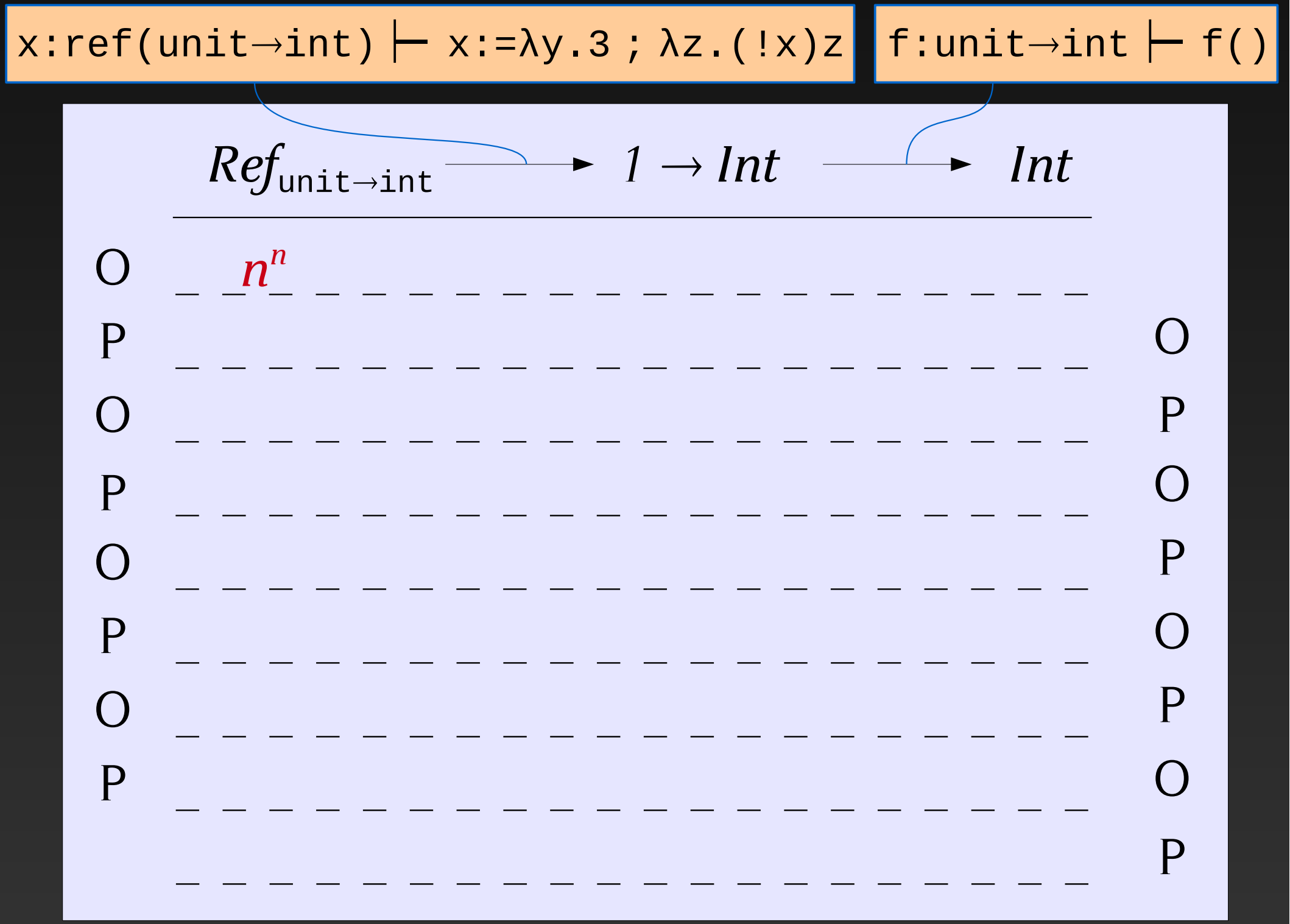
To compose two strategies:

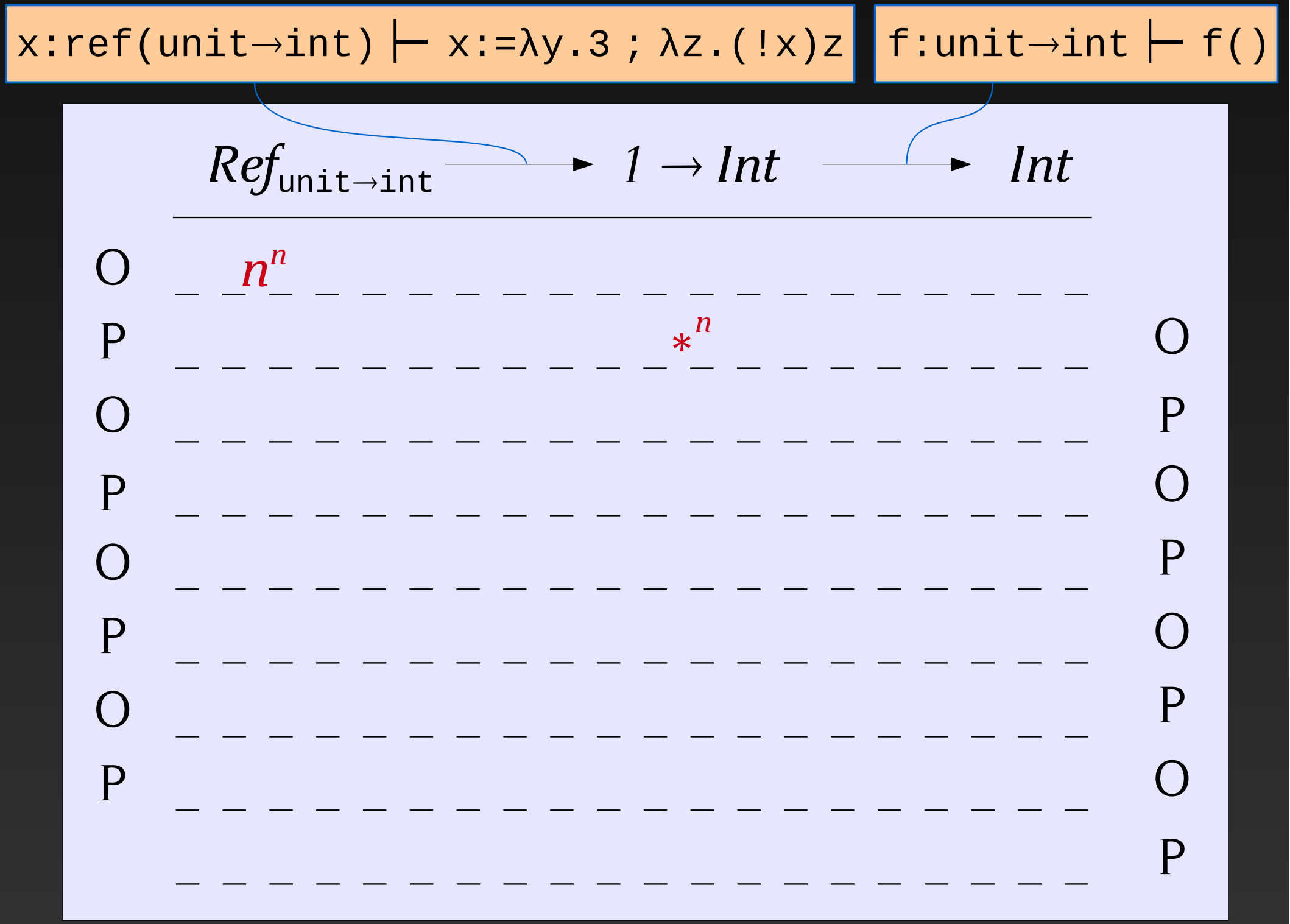
- Synchronise on common component
- Hide moves in common component

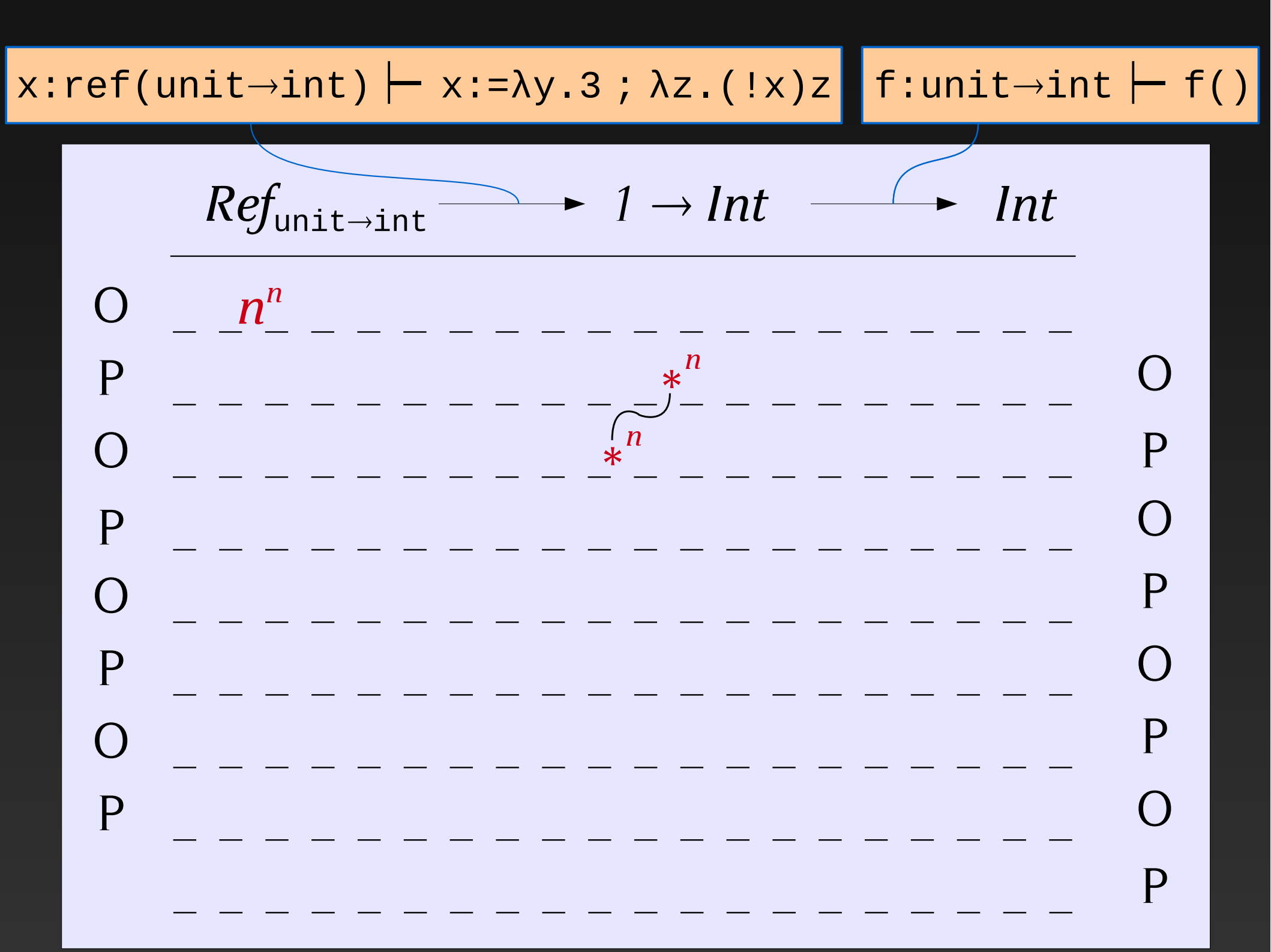
Composition refined

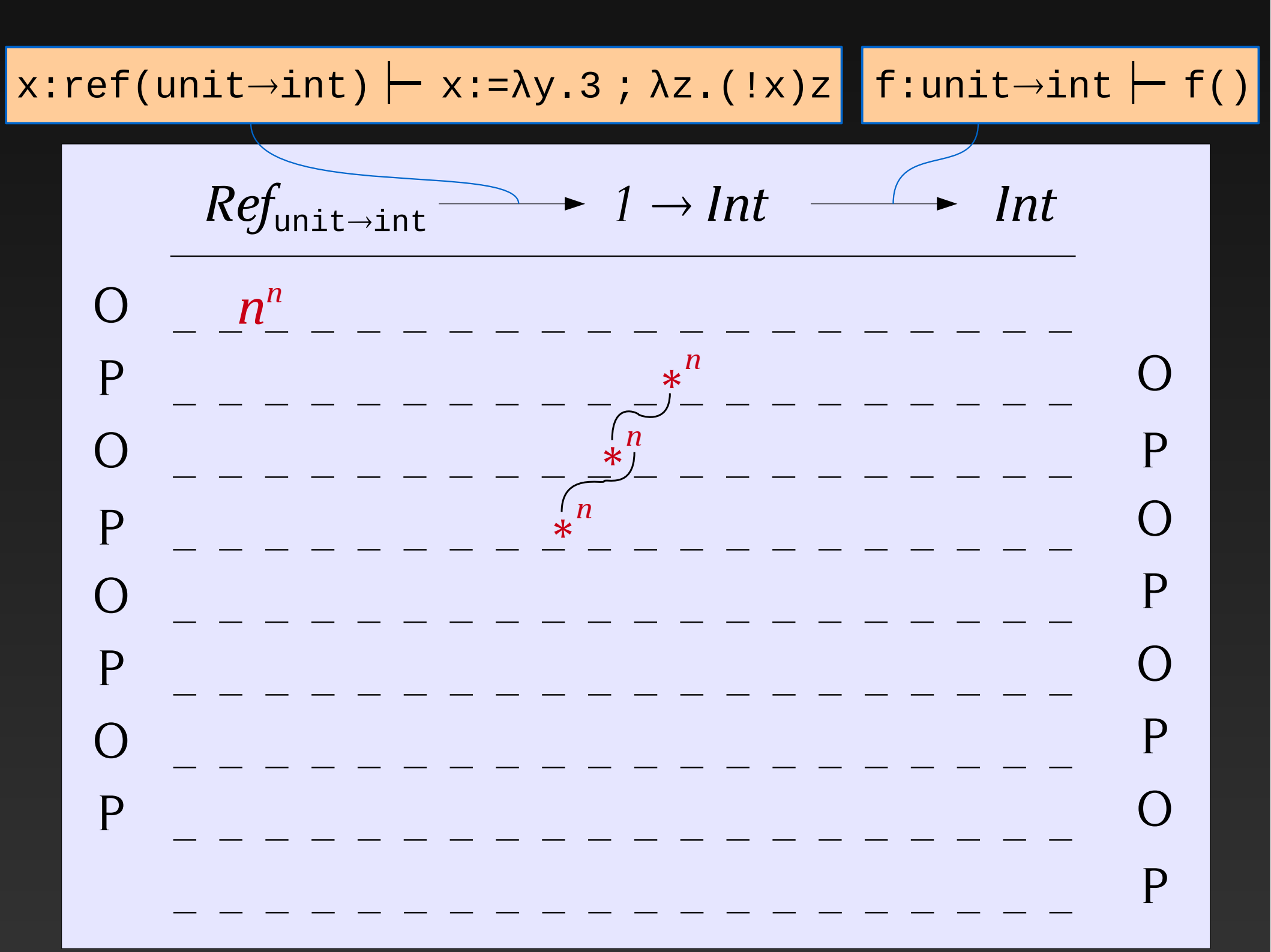
To compose two strategies:

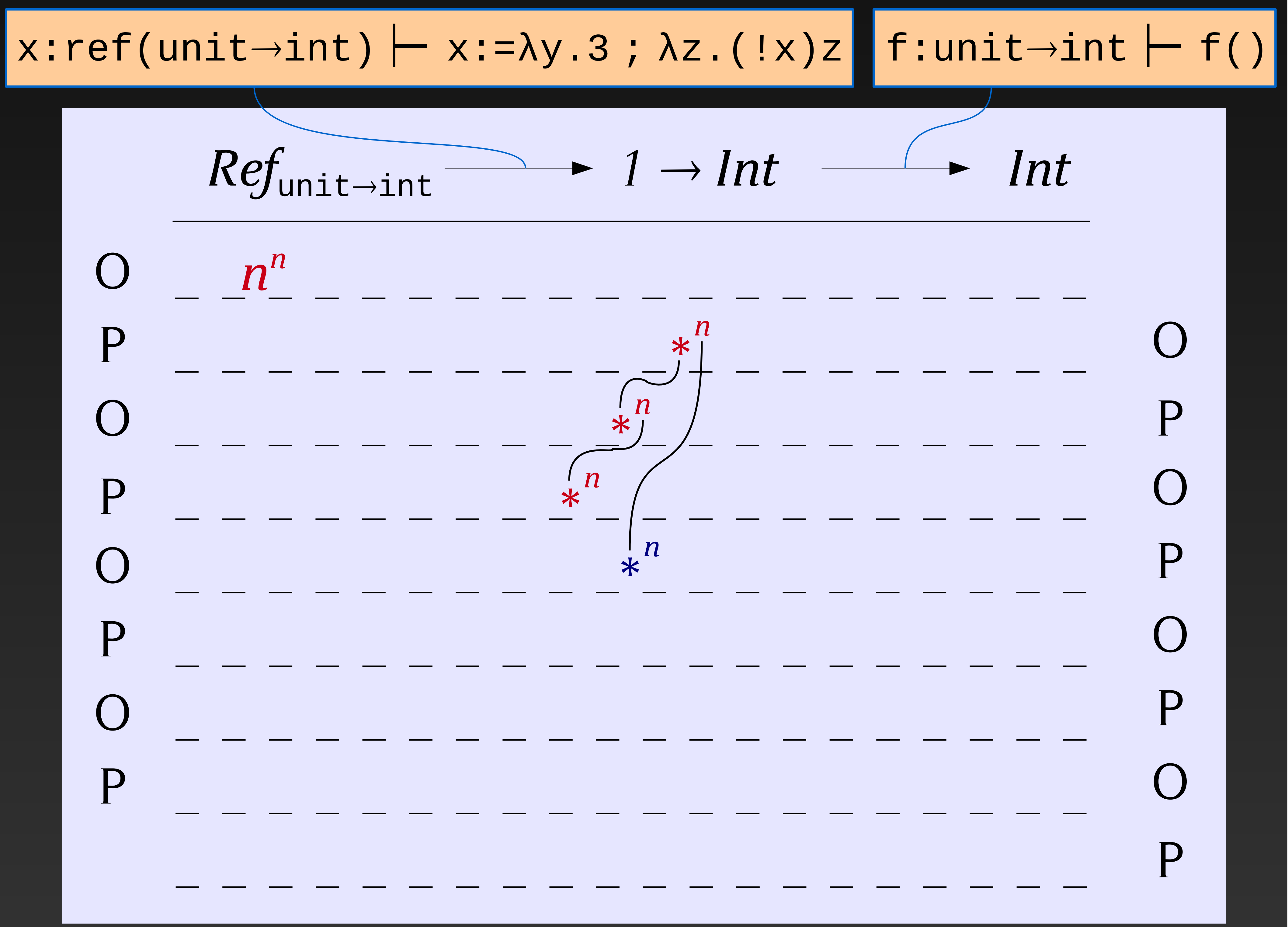
- Synchronise on common component
- Pass along values of private names
- Hide moves in common component
- Garbage collect


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
$Ref_{\mathtt{unit}\to\mathtt{int}}$
$1 \to Int$
Int
n^n
O
P
O
P
O
P
O
P
O
P
O
P

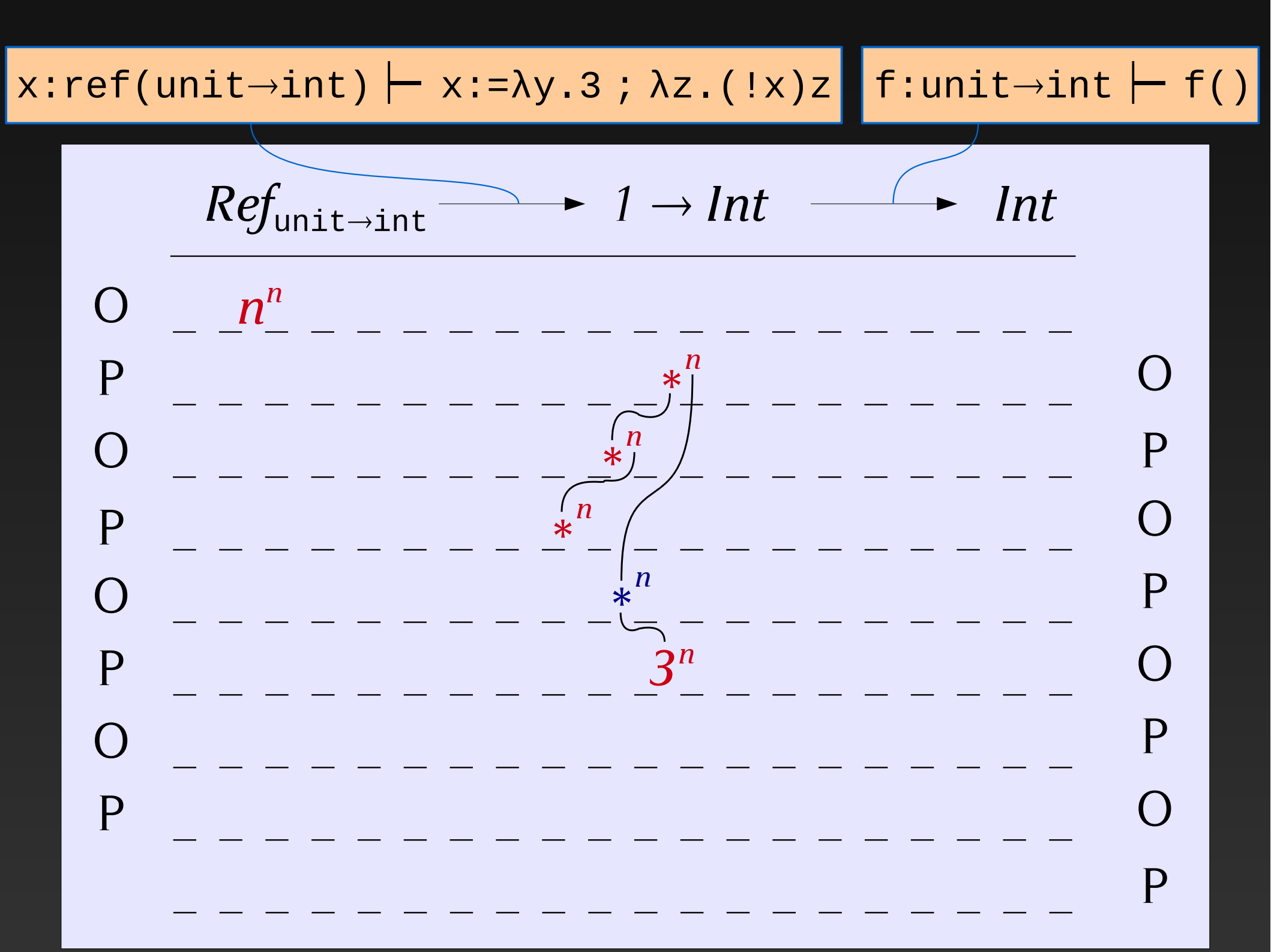

x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
$Ref_{\mathtt{unit}\to\mathtt{int}}$
$1 \to Int$
Int
n^n
$*^n$
O
P
O
P
O
P
O
P
O
P
O
P
O
P

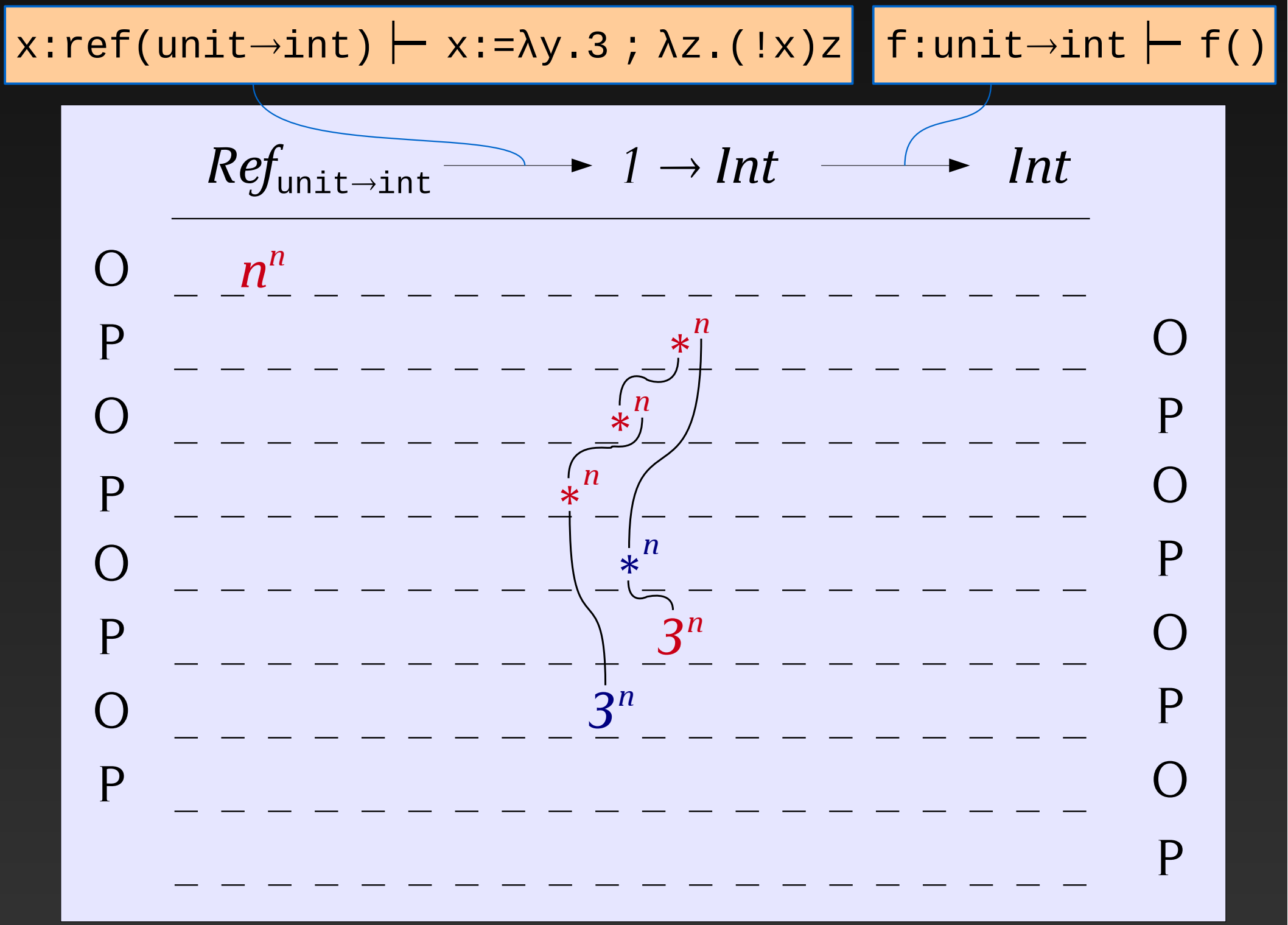

x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
n
n
n
*
*

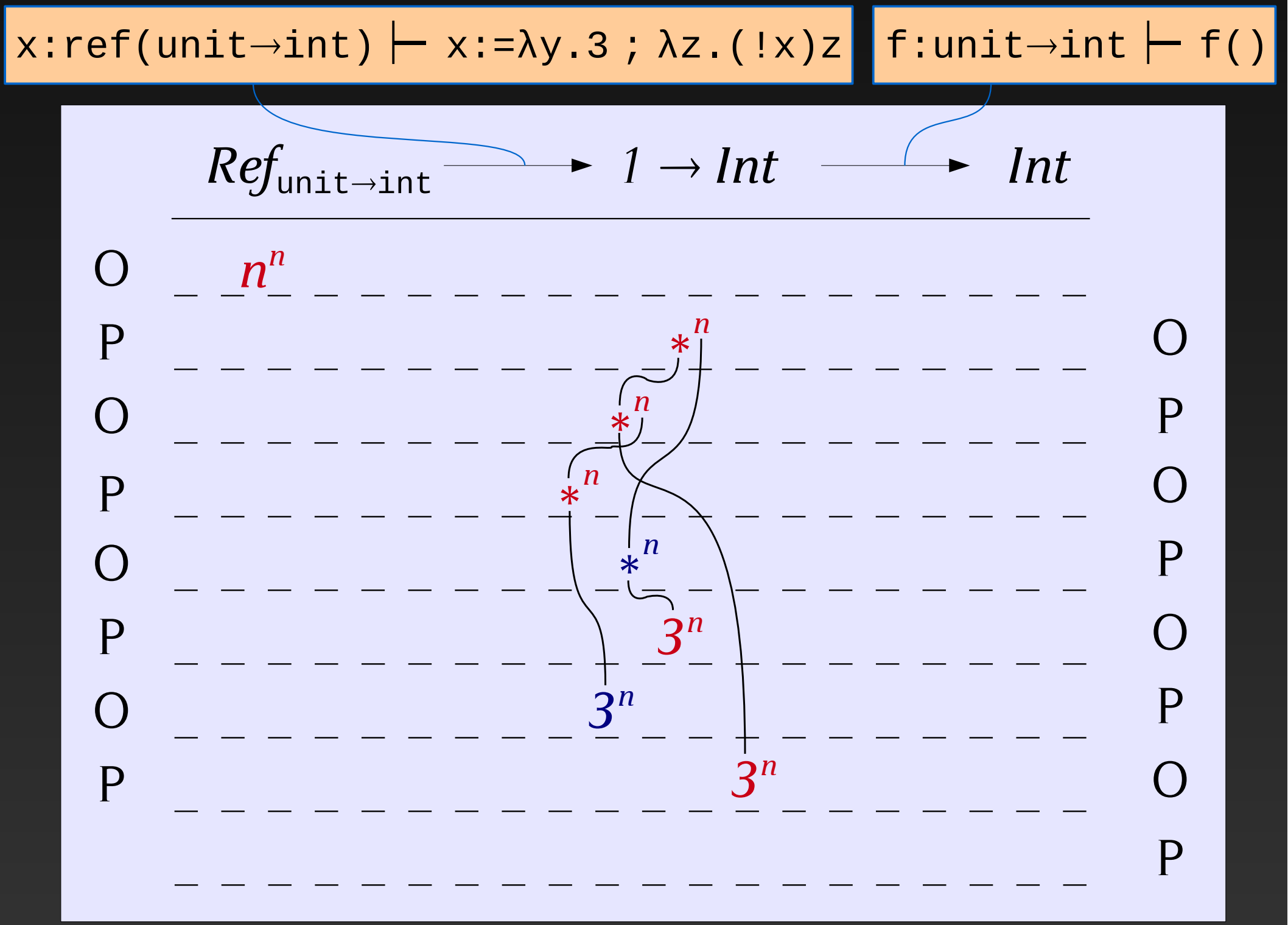

x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
n
n
n
n
O
P
O
P
O
P
O
P
O
P
O
P
O
P

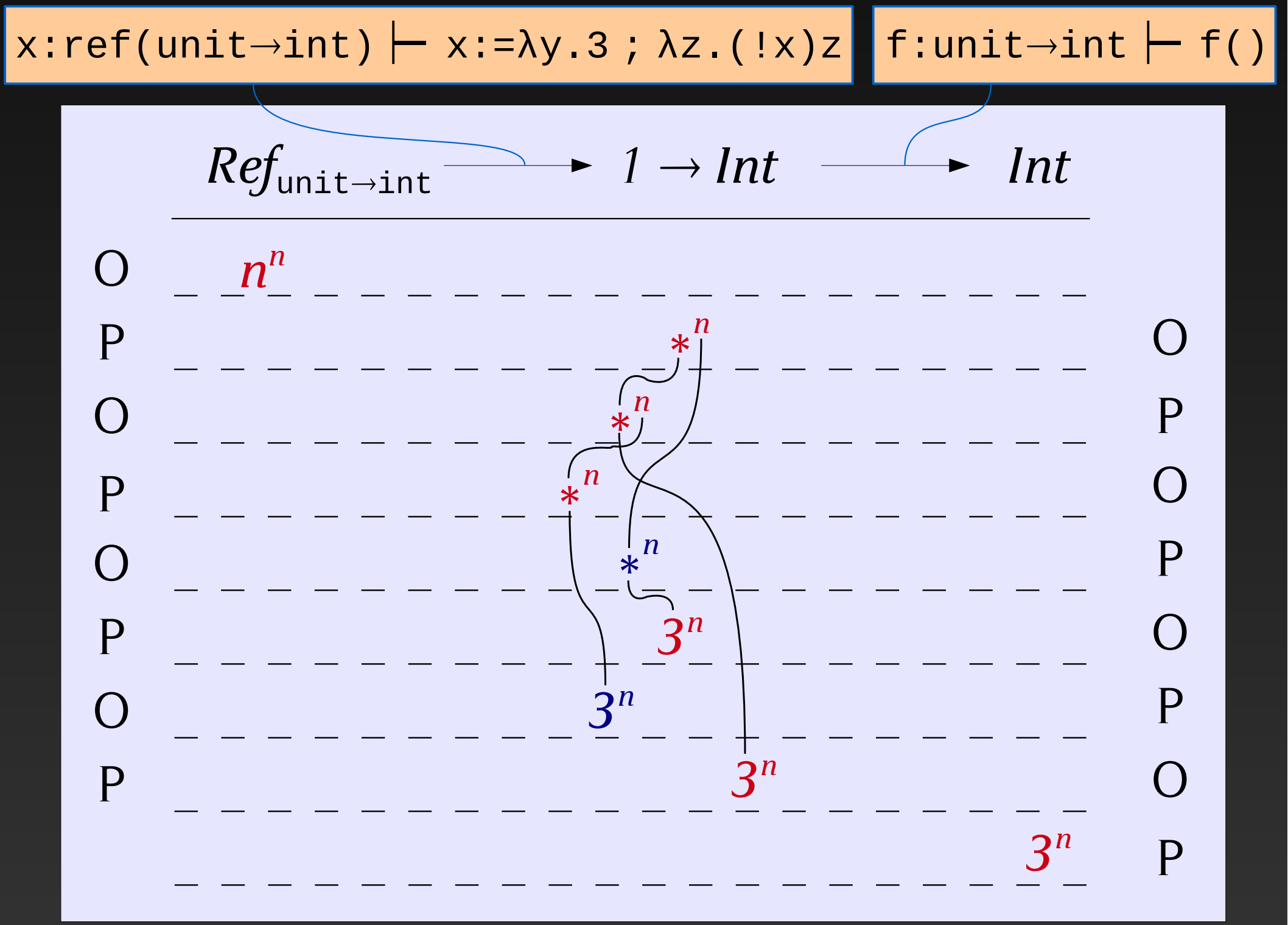


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
O
P
O
P
O
P
n n
* n
* n
* n
* n
O
P
O
P
O
P
O
P


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
O
P
O
P
O
P
O
P
n
n
n
n
n
3
n
*
*
*
*
*


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
n^n
*
n
3^n


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
n^n
*
n
3^n


x:ref(unit→int) ⊢ x:=λy.3 ; λz.(!x)z
f:unit→int ⊢ f()
Ref unit→int
1 → Int
Int
O
P
O
P
O
P
O
P
O
P
n
n
n
n
n
3
n
3
n
3
n
3
n

Our model

Built using game semantics

Uses a built-in notion of names and games which carry the store around

It models RefML fully abstractly

$$M_1 \cong M_2 \iff [\![M_1]\!] = [\![M_2]\!]$$

What's next

- Connections with LTS semantics
  [Jeffrey & Rathke '99], [Laird '07]
- Connections with quotiented model [Tz.'07]
- Algorithmic game semantics
- Polymorphism and recursive types

What's next

thank you!

- Connections with LTS semantics
  [Jeffrey & Rathke '99], [Laird '07]
- Connections with quotiented model [Tz.'07]
- Algorithmic game semantics
- Polymorphism and recursive types