# Games with names

Nikos Tzevelekos

Queen Mary, University of London

# What this talk is about

Generation of new resources is a pervasive feature in computation (references, objects, channels, etc.)

We see resources as names

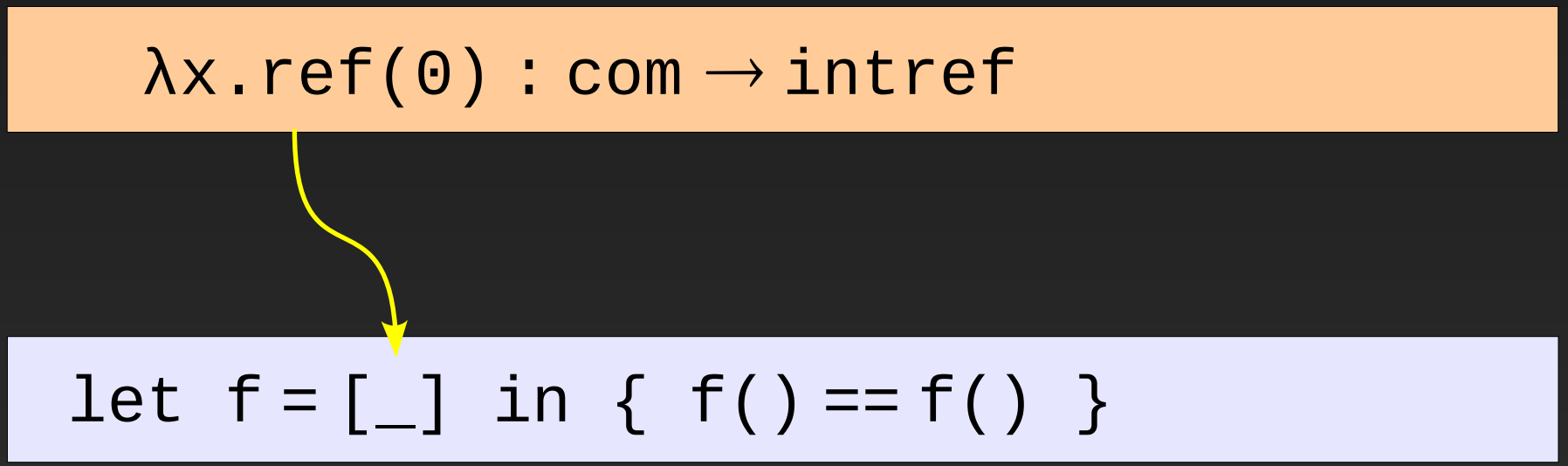
We show how to use game semantics to capture computation with names

# Computation with names

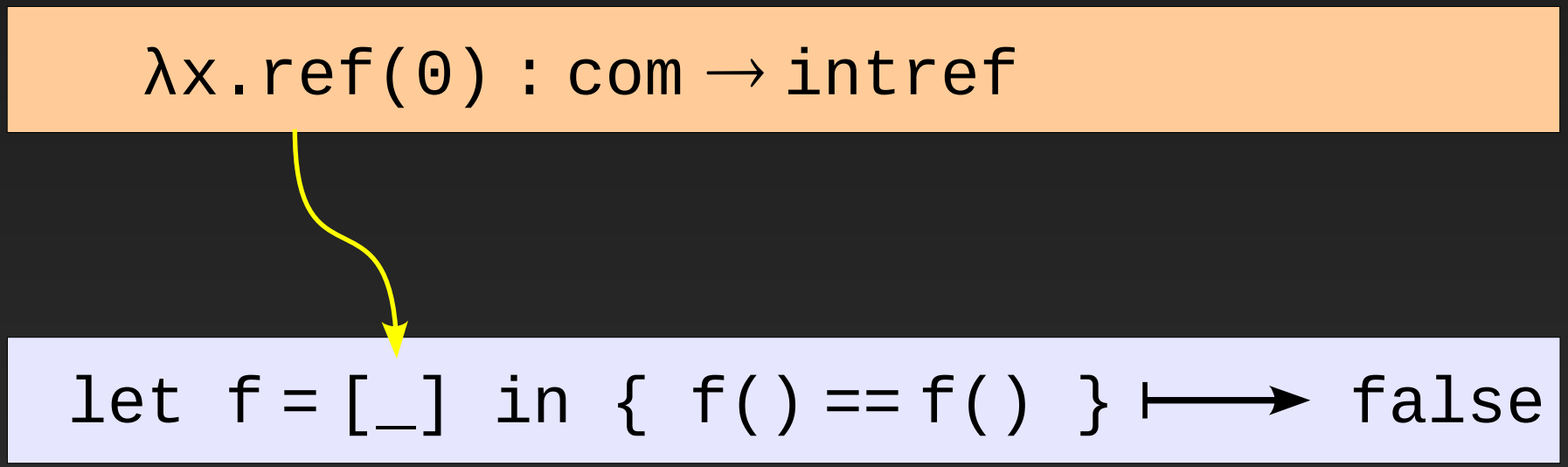
$\lambda x.\mathtt{ref}(0) : \mathtt{com} \to \mathtt{intref}$

```
let f = [_] in { f() == f() }
```

# Computation with names



# Computation with names



# Example: Reduced ML

basic programming language with functions + integer references

$$P \cong P'$$

*equivalent behaviour in every context*

# Example: Reduced ML

basic programming language with functions + integer references

$$\mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\ \lambda y.\, (x == y) \quad \cong \quad \lambda y.\, \mathsf{false}\ : \mathsf{intref} \to \mathsf{int}$$

# Example: Reduced ML

basic programming language with
functions + integer references

$$\mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\ \lambda y.\, (x == y) \quad \cong \quad \lambda y.\, \mathsf{false}\ : \mathsf{intref} \to \mathsf{int}$$

$$\lambda y.\, \mathsf{ref}(0) \quad \ncong \quad \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, (\lambda y.\, x)\ : \mathsf{com} \to \mathsf{intref}$$

# Example: Reduced ML

basic programming language with functions + integer references

$$\mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\ \lambda y.\, (x == y) \quad \cong \quad \lambda y.\, \mathsf{false}\ : \mathsf{intref} \to \mathsf{int}$$

$$\lambda y.\, \mathsf{ref}(0) \quad \ncong \quad \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, (\lambda y.\, x)\ : \mathsf{com} \to \mathsf{intref}$$

$$f : \mathsf{intref} \to \mathsf{int} \ \vdash\ \lambda y.\, f(\mathsf{ref}(0))$$

$$\cong\ \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \lambda y.\, x := 0; f(x)\ : \mathsf{com} \to \mathsf{int}$$

# Two ways to model references

Reynolds

- *Idealized Algol (1978)*

References are *pairs:*

$$\mathtt{intref} = (\mathtt{com} \to \mathtt{int}) \times (\mathtt{int} \to \mathtt{com})$$

$$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$$

# Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$$\texttt{intref} = \texttt{(com} \to \texttt{int)x(int} \to \texttt{com)}$$

$$\longmapsto \texttt{(1} \to \texttt{Z)x(Z} \to \texttt{1)}$$

- Theoretically attractive
- but: `mkvar(λx.3,λx.())` (*bad variables*)

# Two ways to model references

## Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$

$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$

- Theoretically attractive
- but: $\texttt{mkvar}(\lambda x.3, \lambda x.())$ (*bad variables*)

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

References are *names:*

$\texttt{intref} = \texttt{base type}$

$\longmapsto \mathbf{N}$ (names)

# Two ways to model references

## Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$

$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$

- Theoretically attractive
- but: $\texttt{mkvar}(\lambda x.3, \lambda x.())$ (*bad variables*)

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

References are *names:*

$\texttt{intref} = \text{base type}$

$\longmapsto \mathbf{N}$ (names)

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

# Bad variables

- Many pairs of ref type are *not* references
  - e.g. `mkvar(λx.3,λx.())`
- no notion of *reference equality test*
- spurious non-equivalences:

```
x:=0; !x      vs.   x:=0; 0
x:=0; x:=1    vs.   x:=1
```

# Game Semantics

How can we reason about:

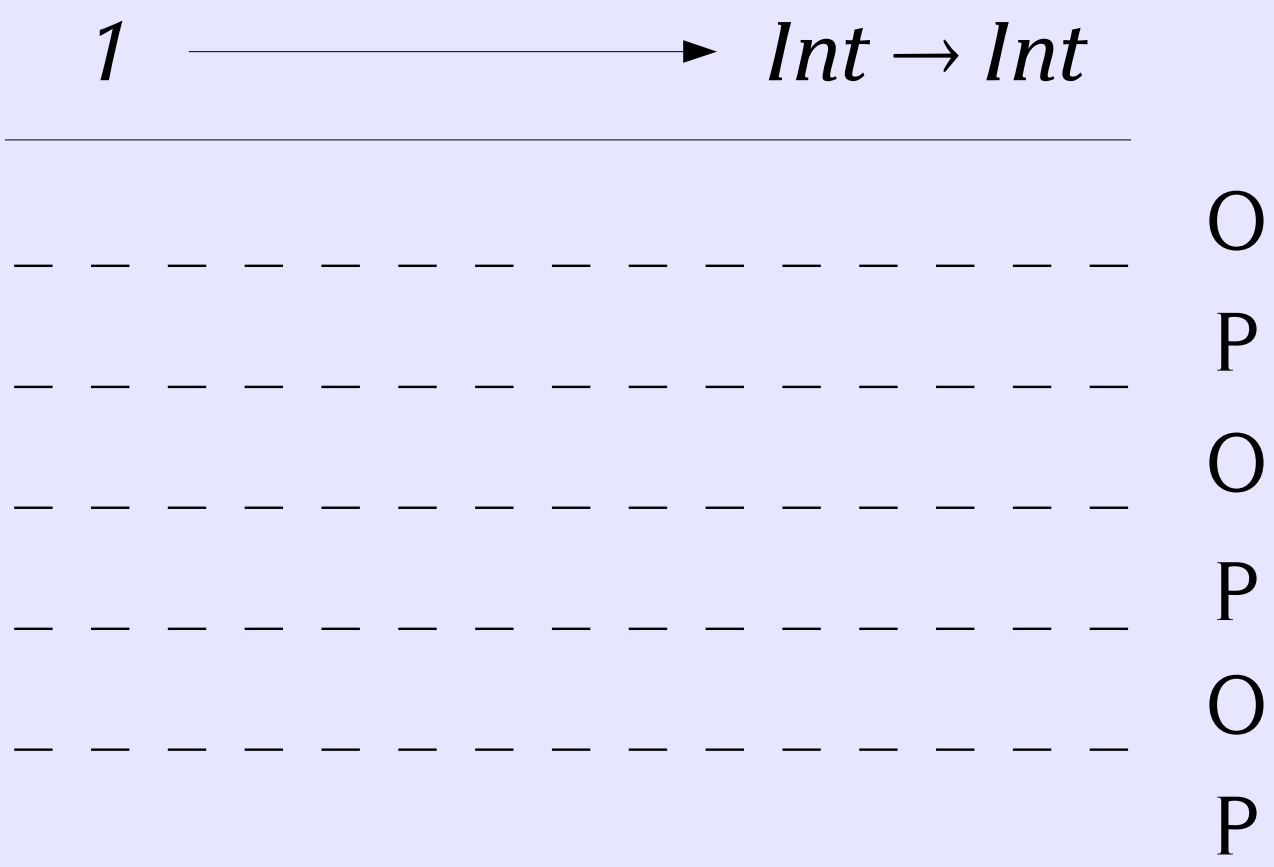
$$P \cong P'$$

# Game Semantics

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

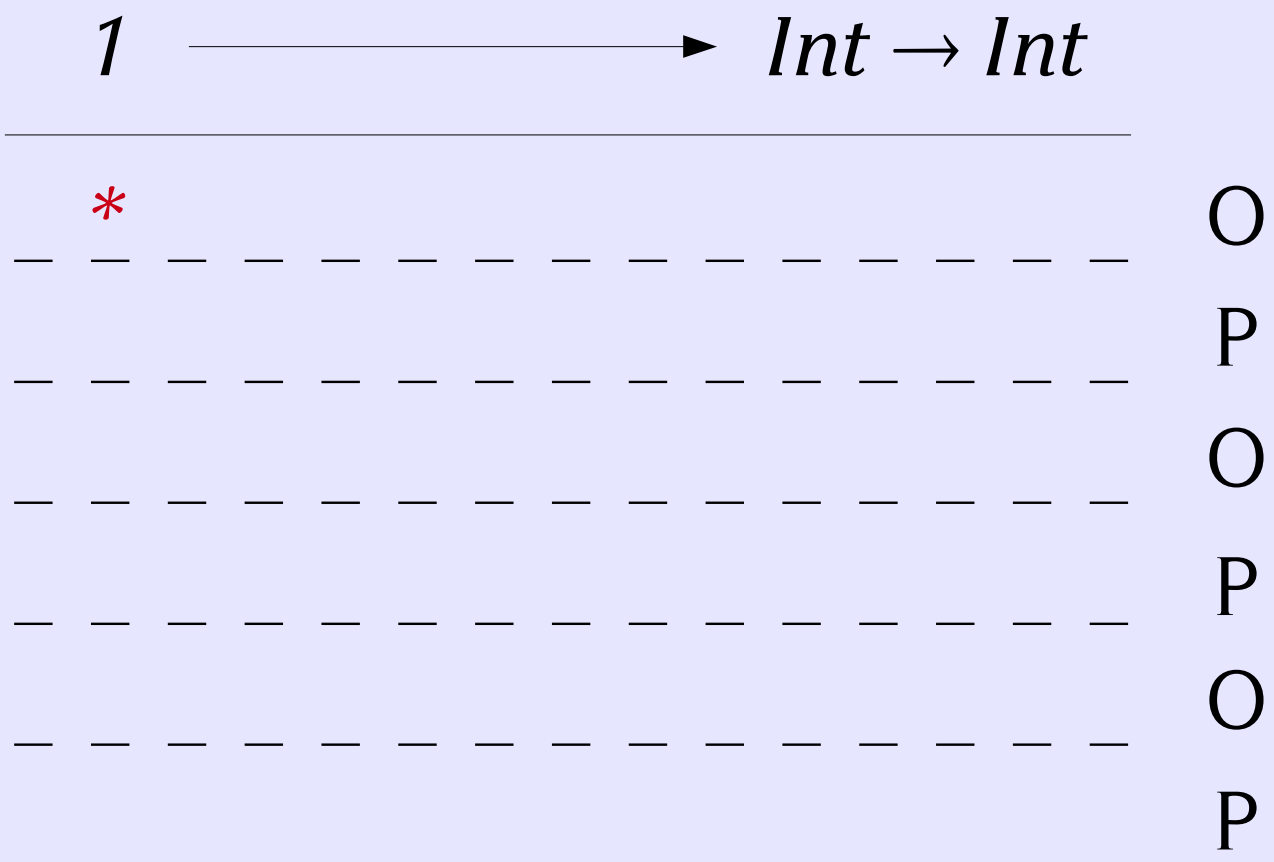
# Example

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



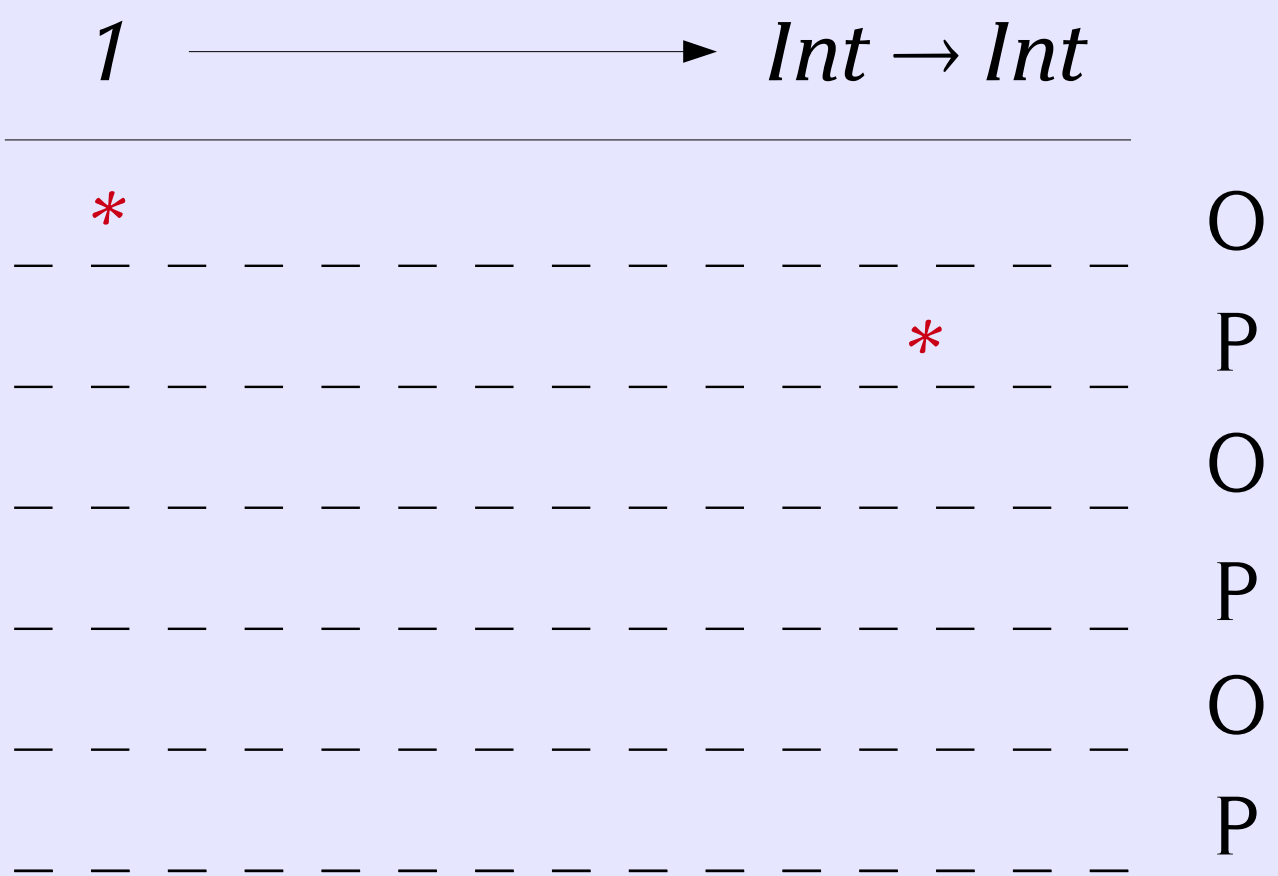
# Example

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



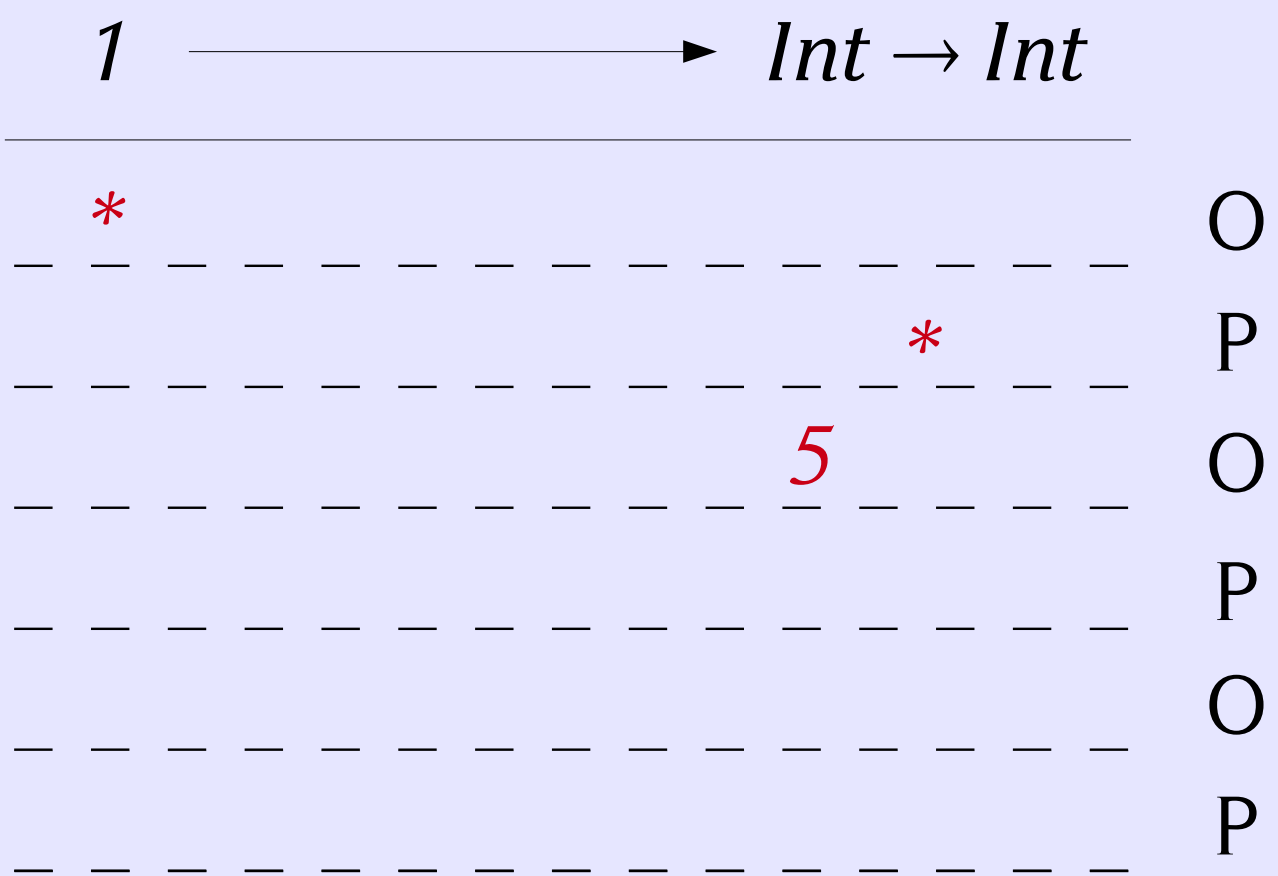
# Example

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



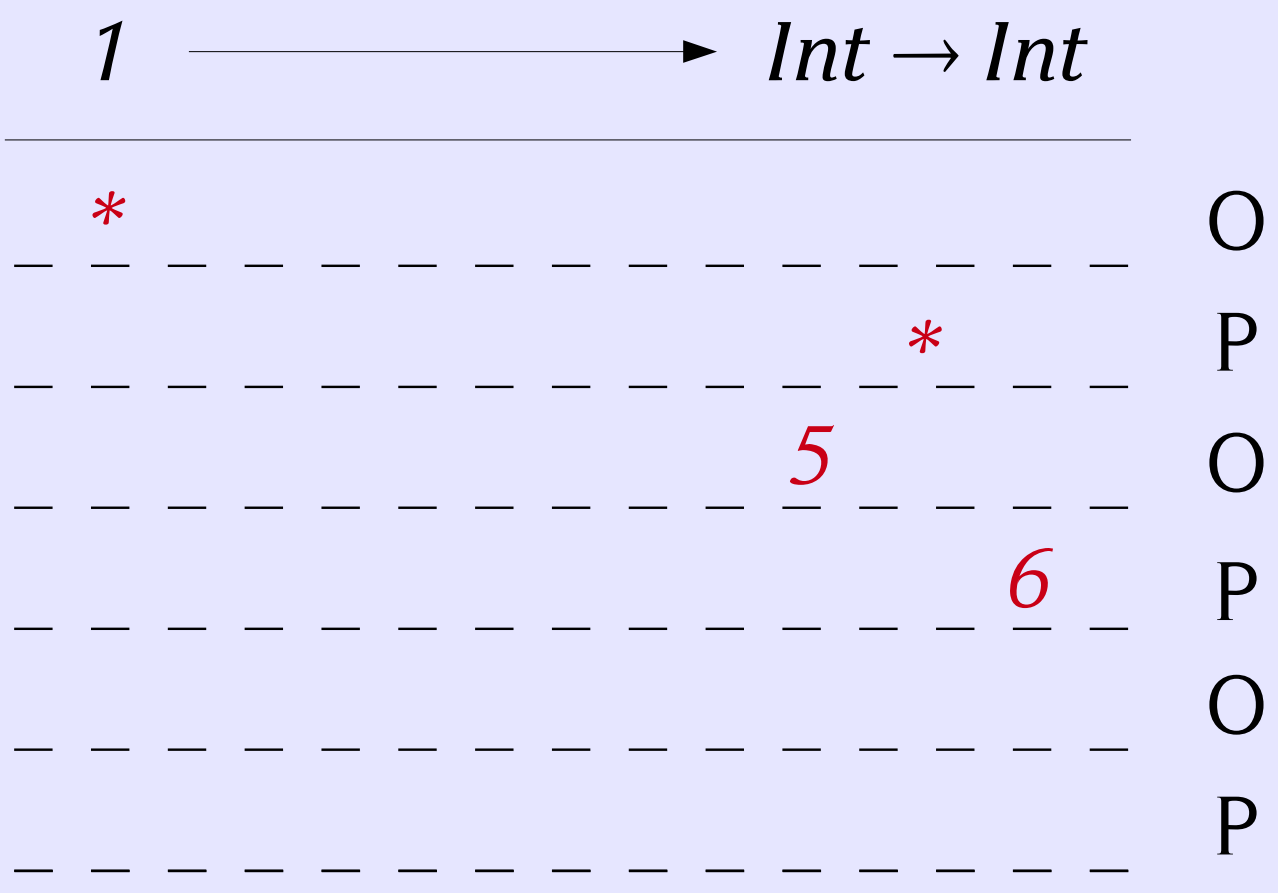
# Example

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



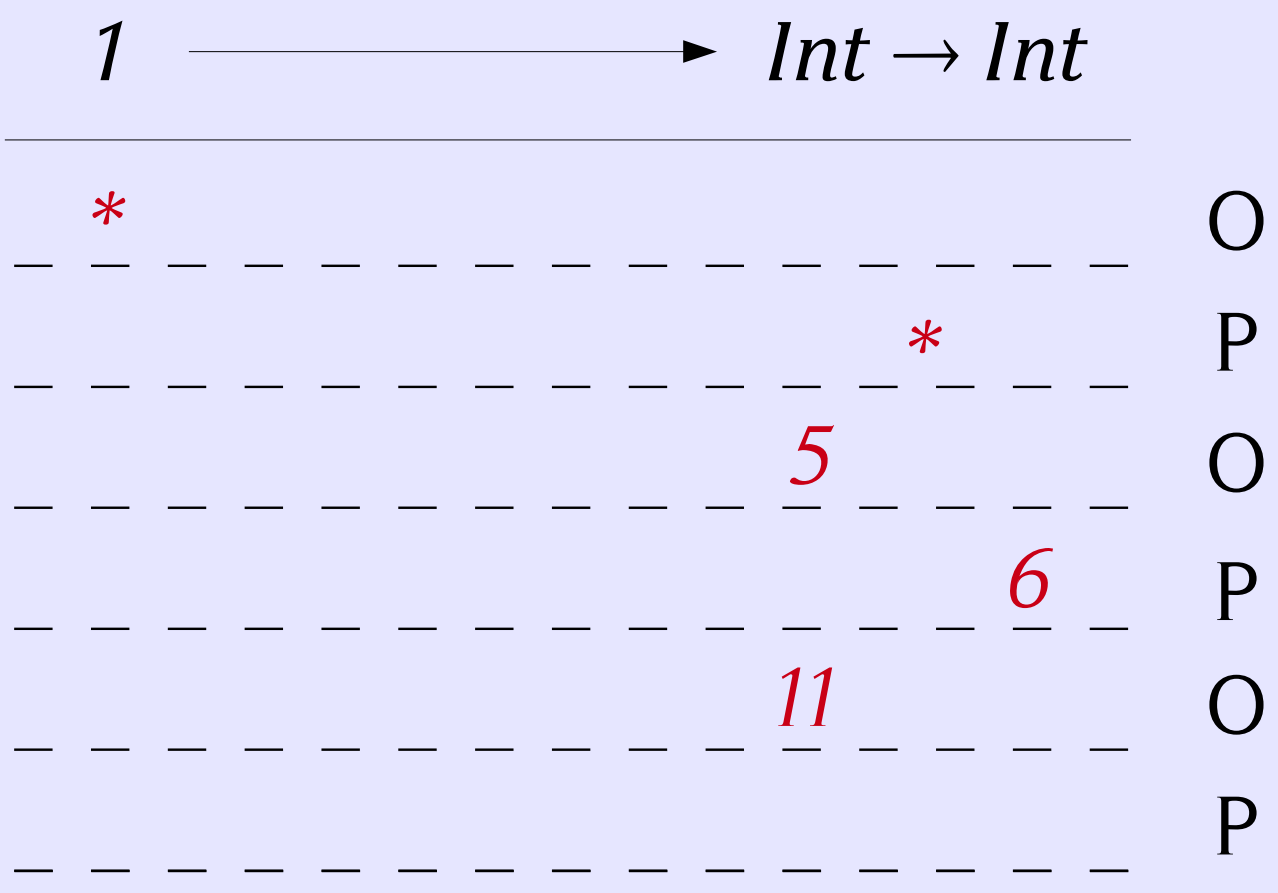
# Example

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



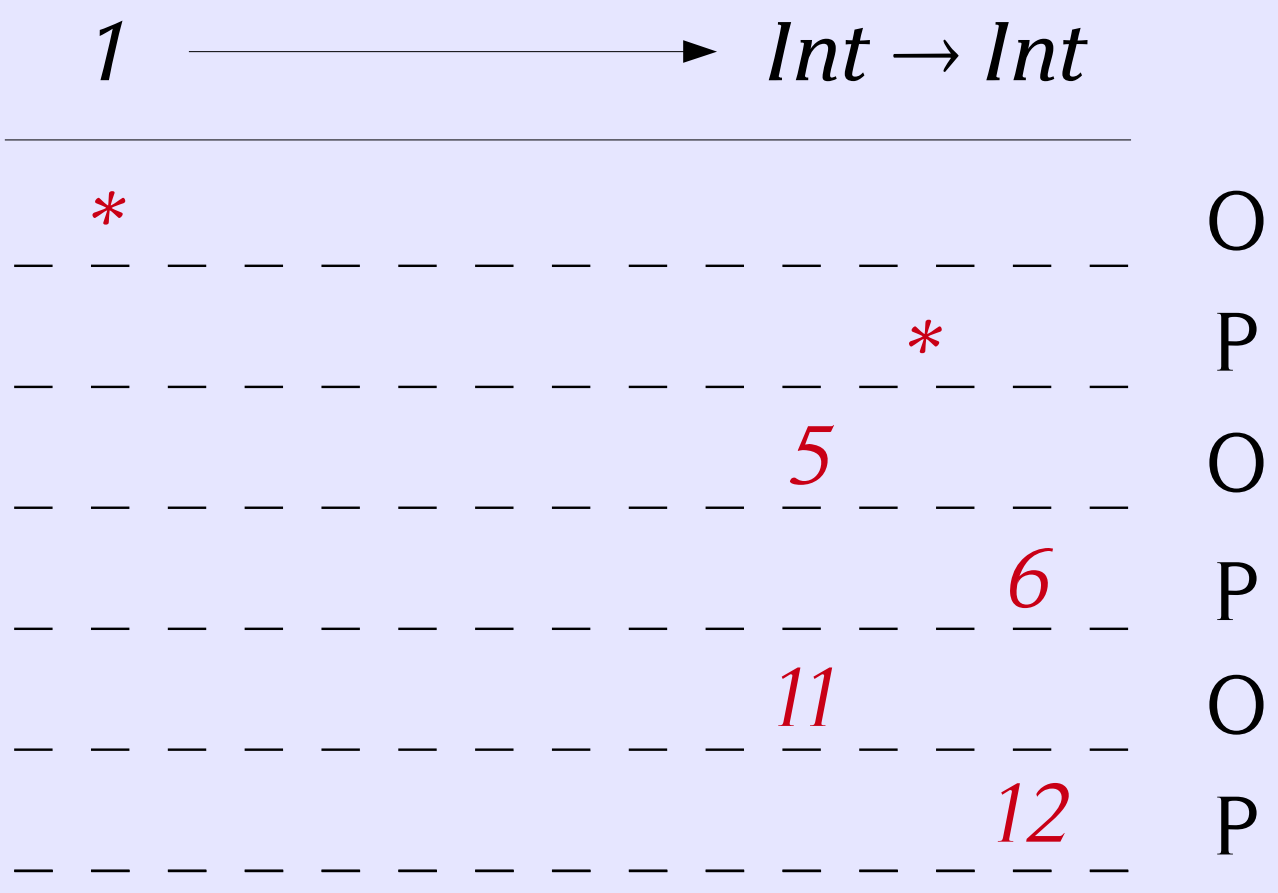
# Example

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



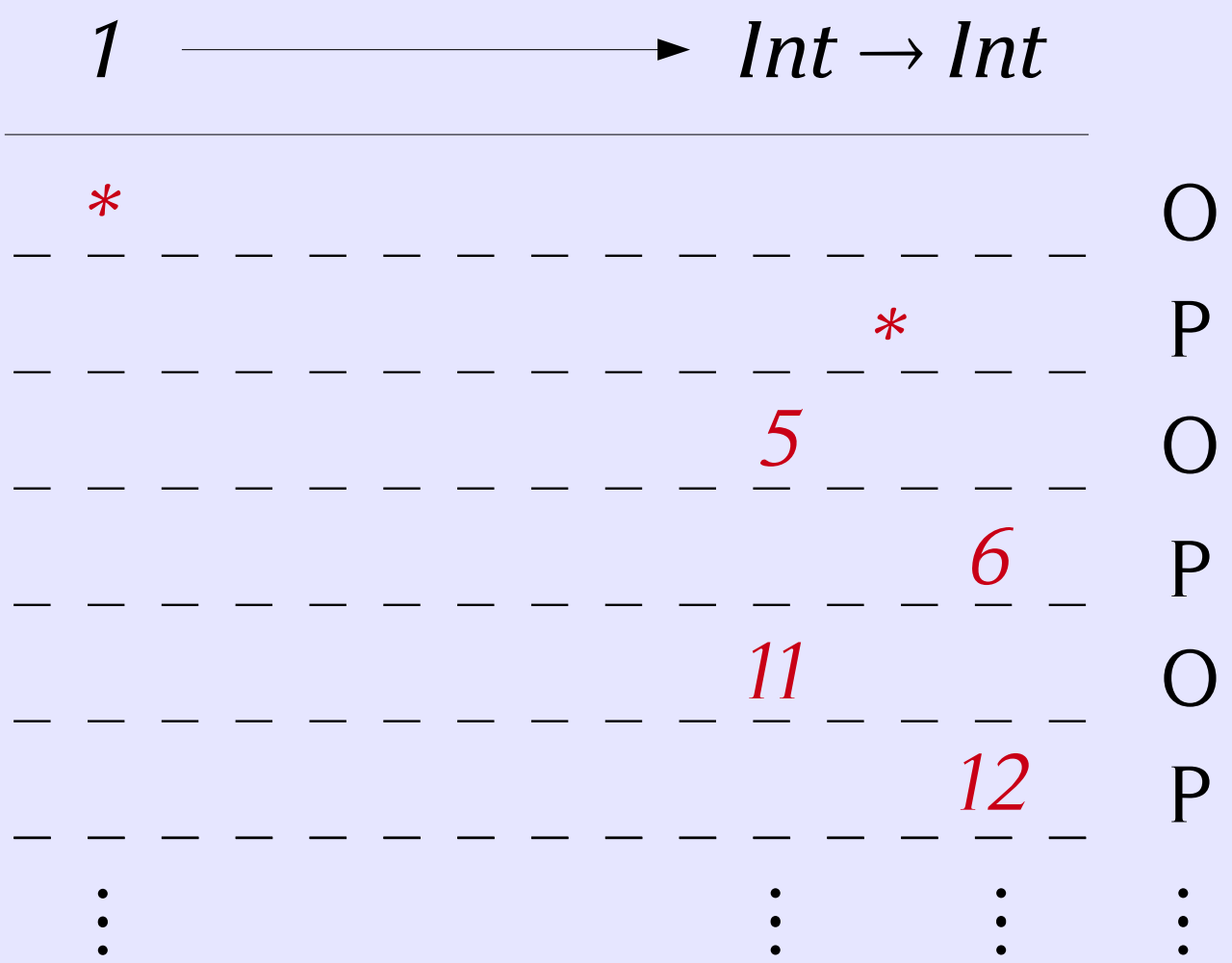
# Example

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



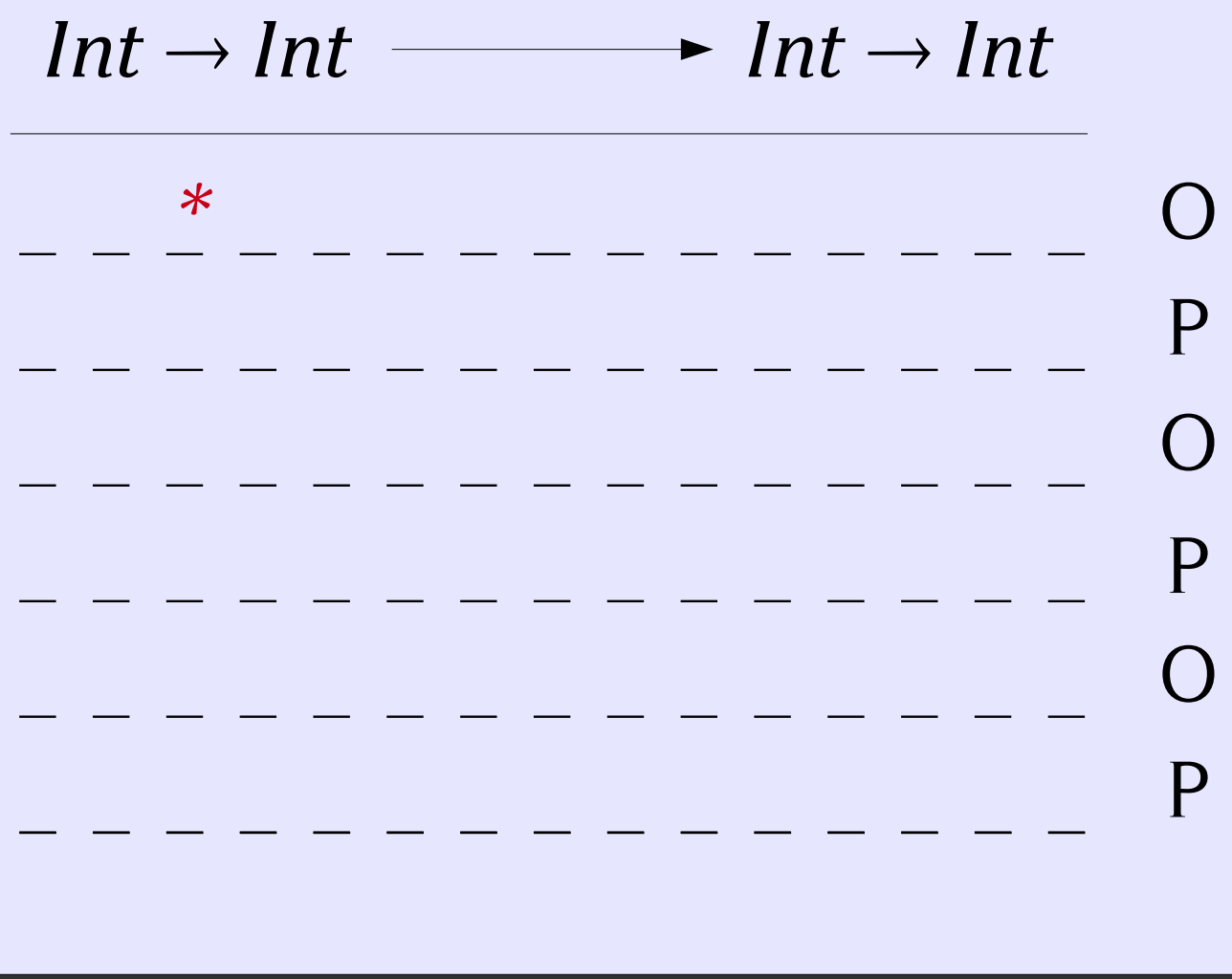
# Example

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



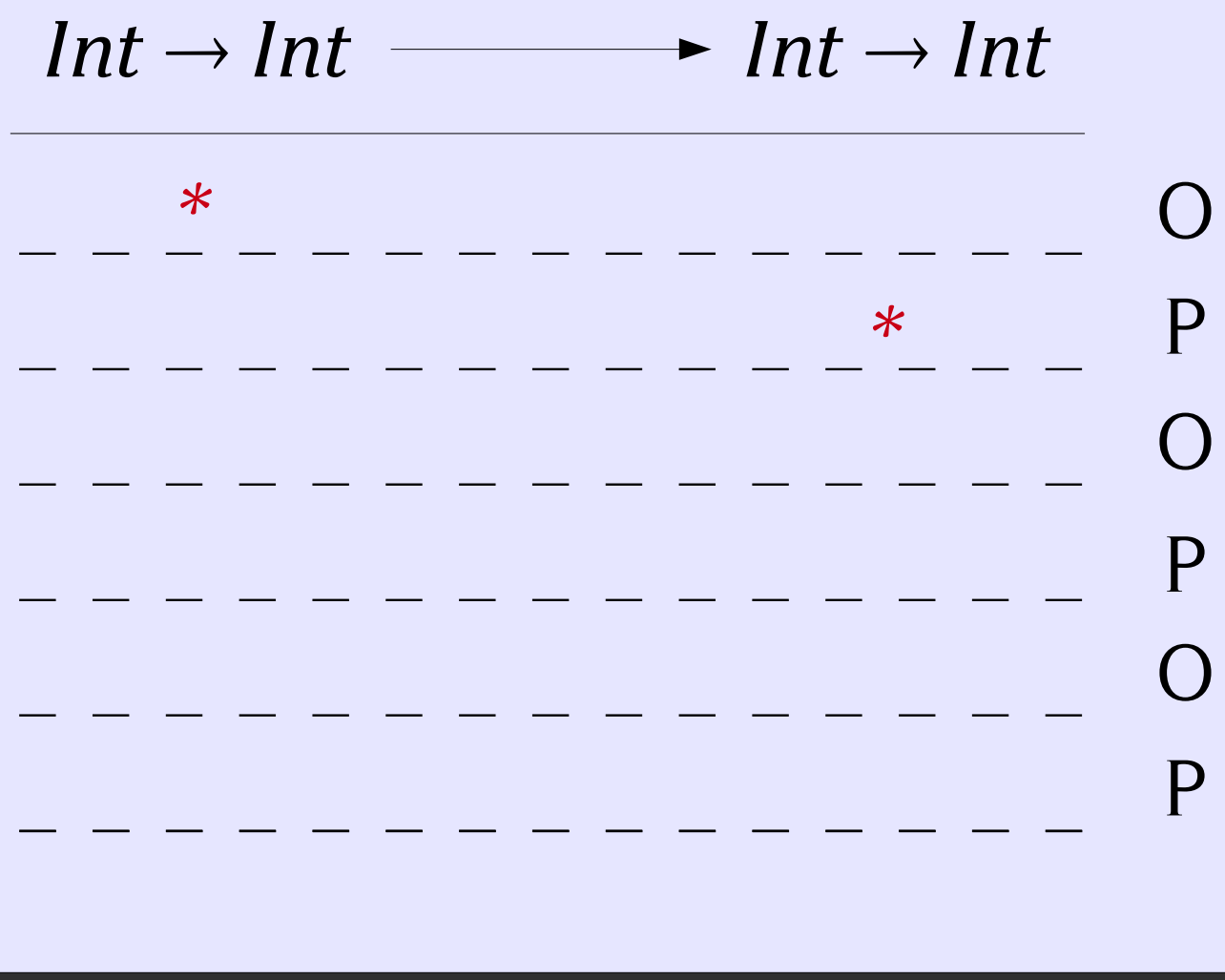
# Example

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



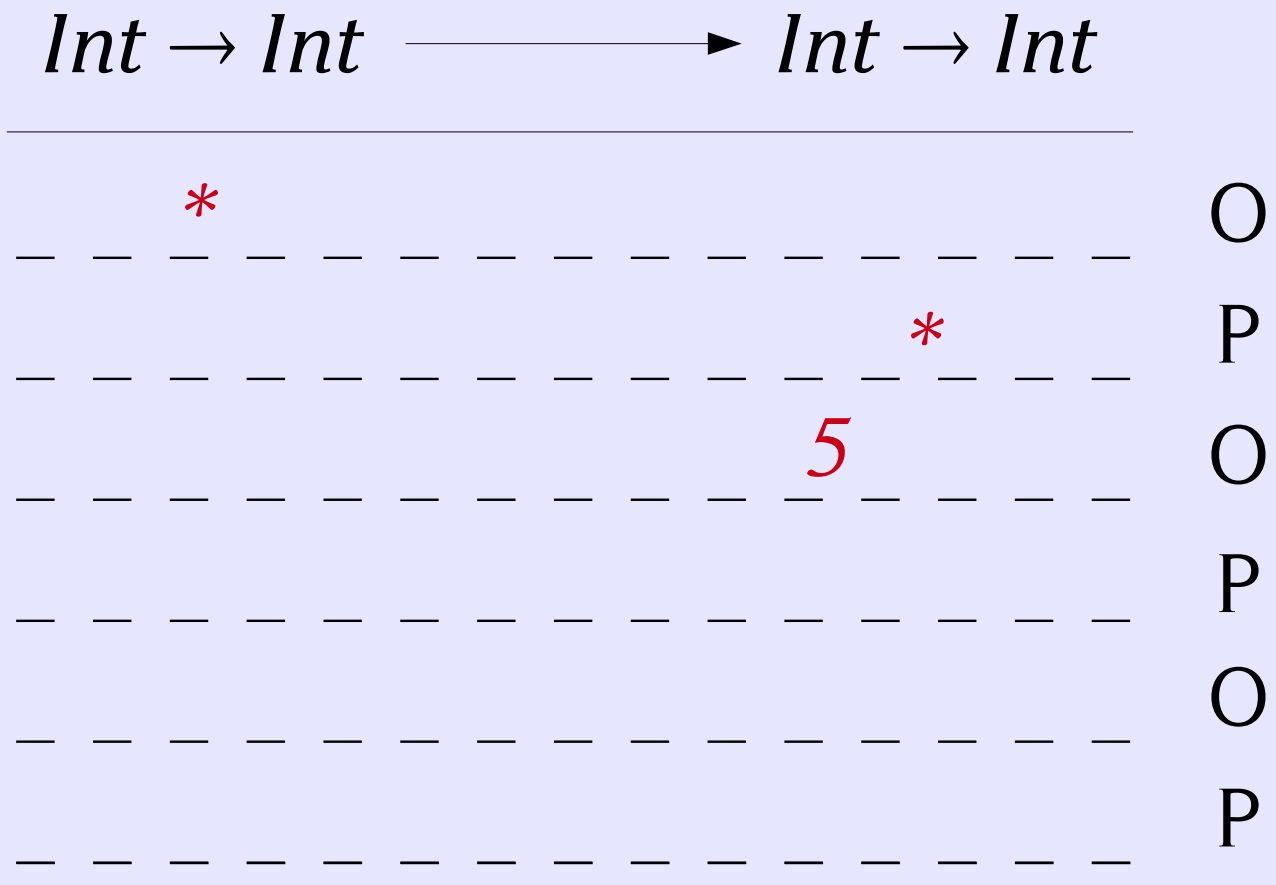
# Example

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



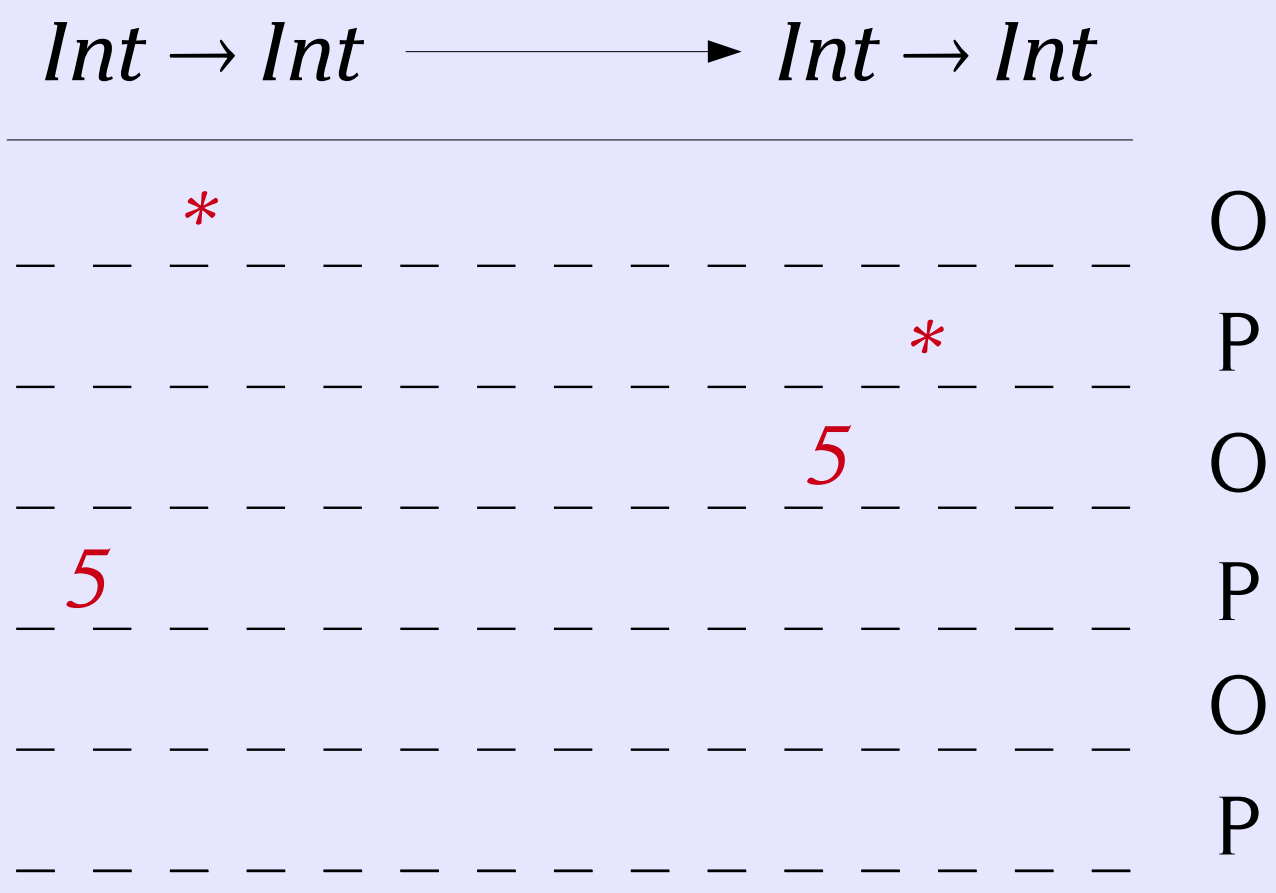
# Example

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



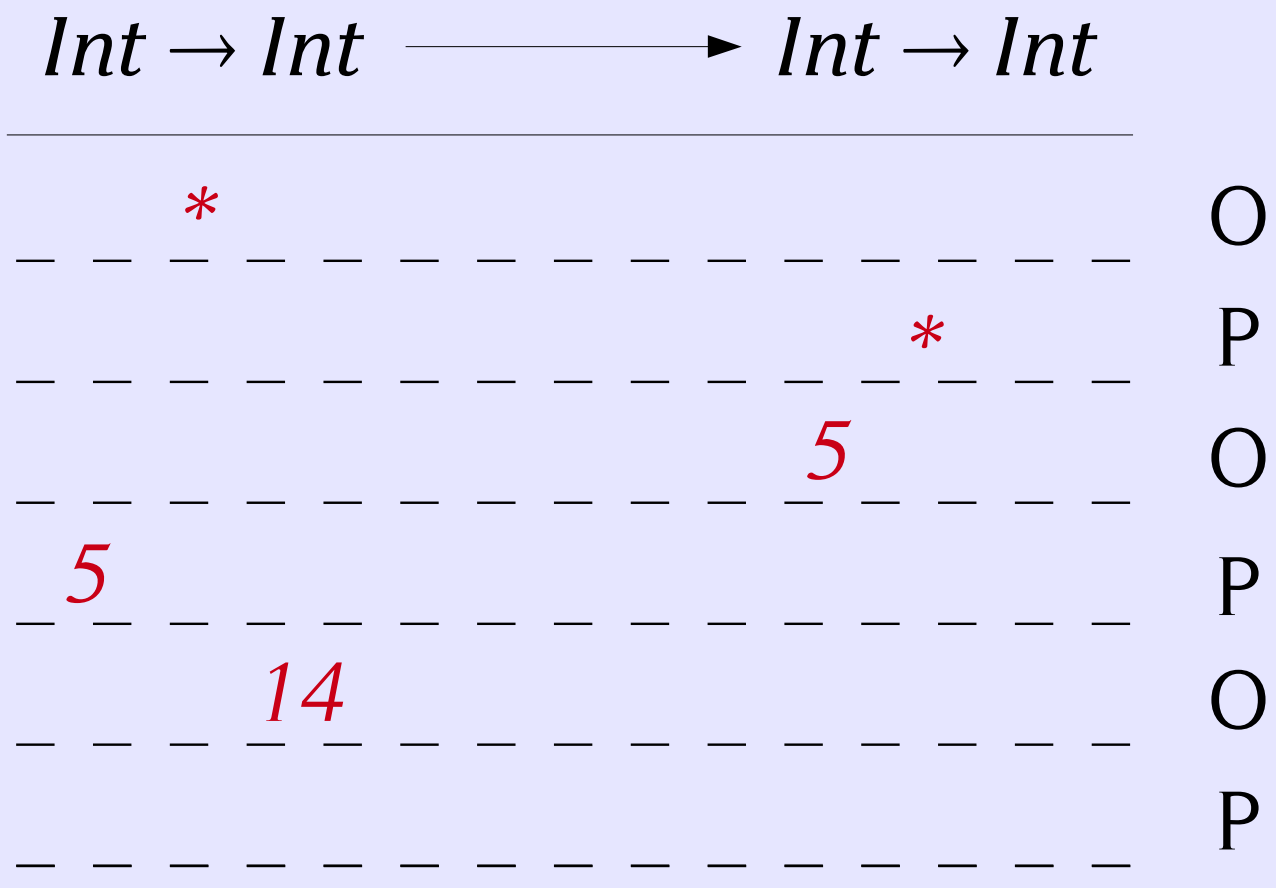
# Example

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



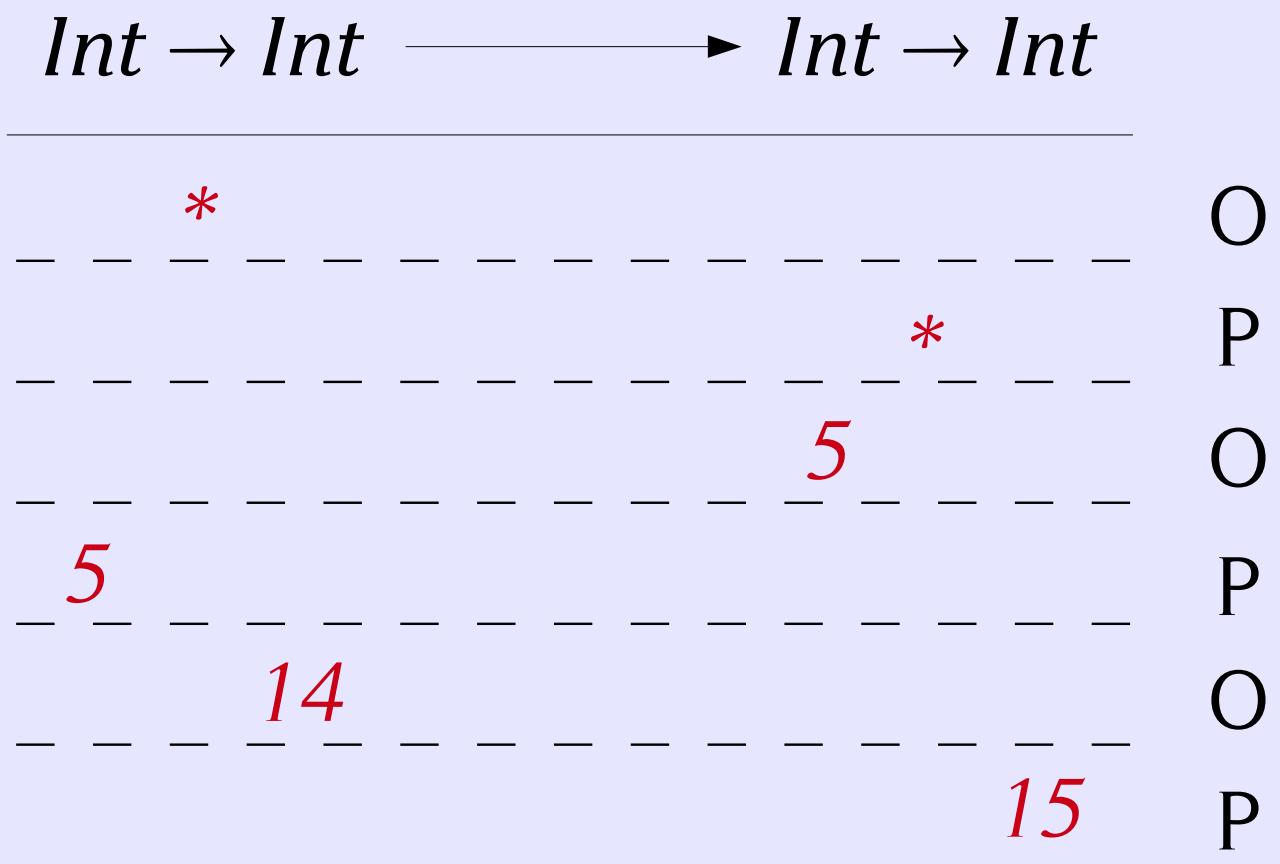
# Example

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



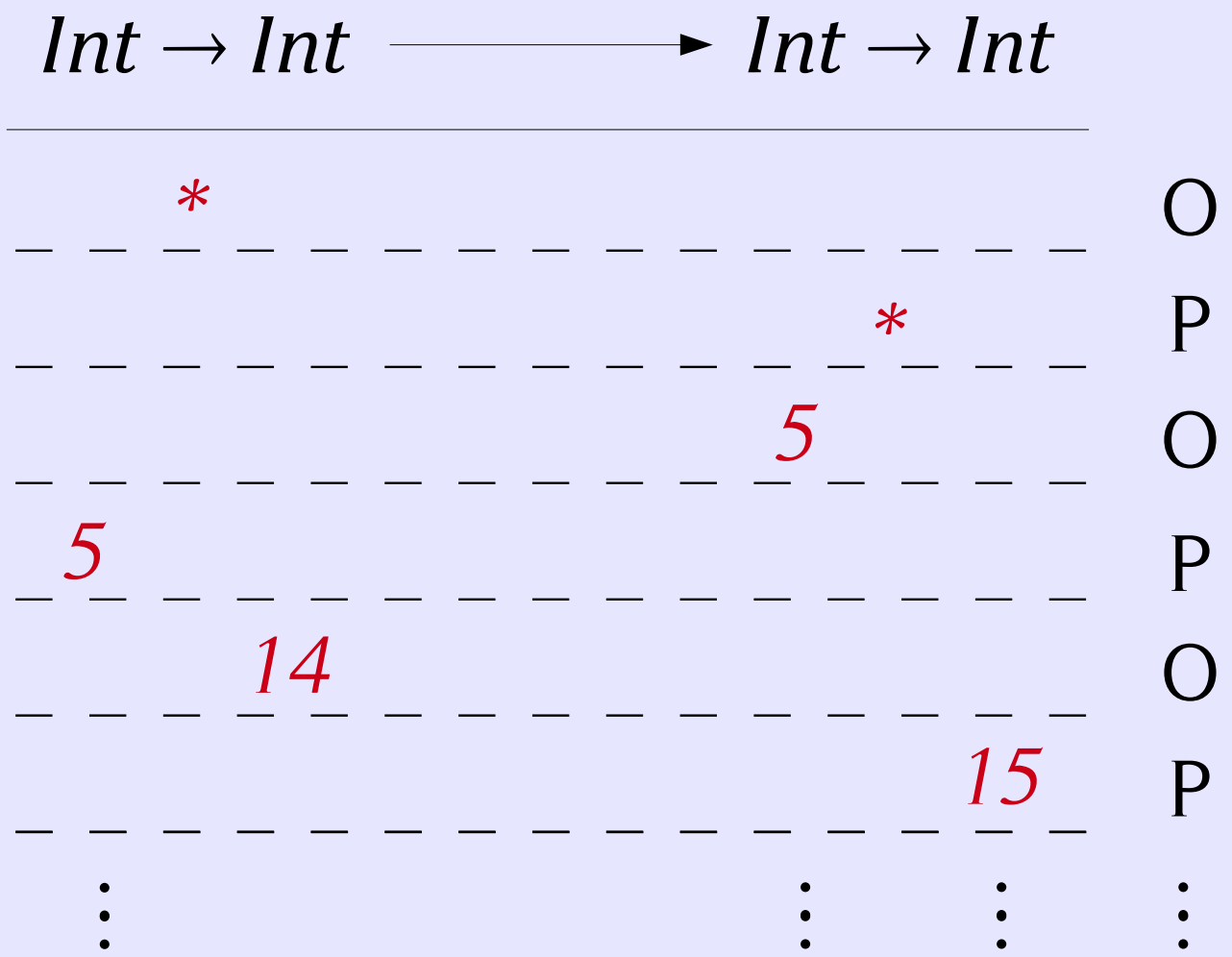
# Example

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

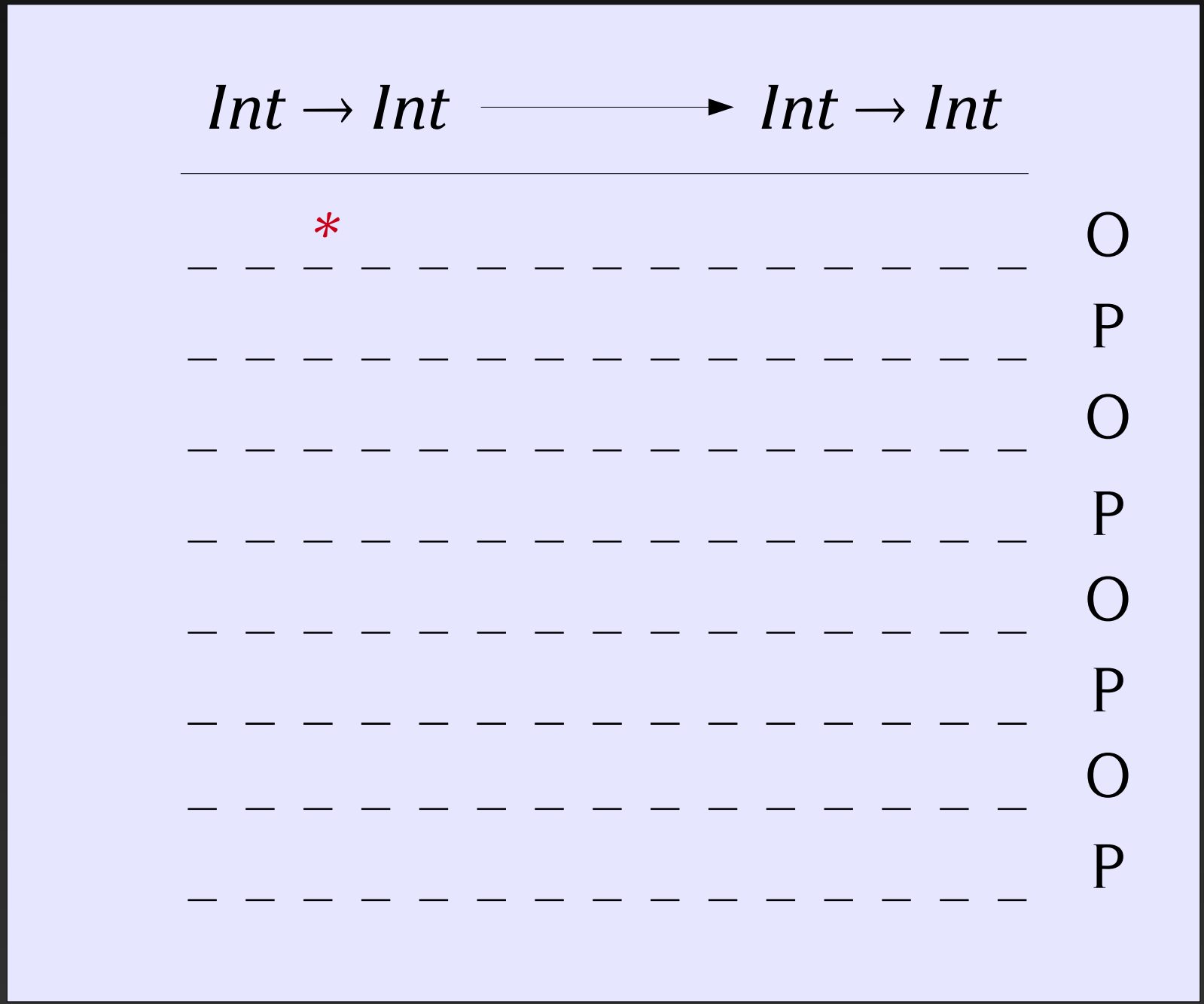


# Example

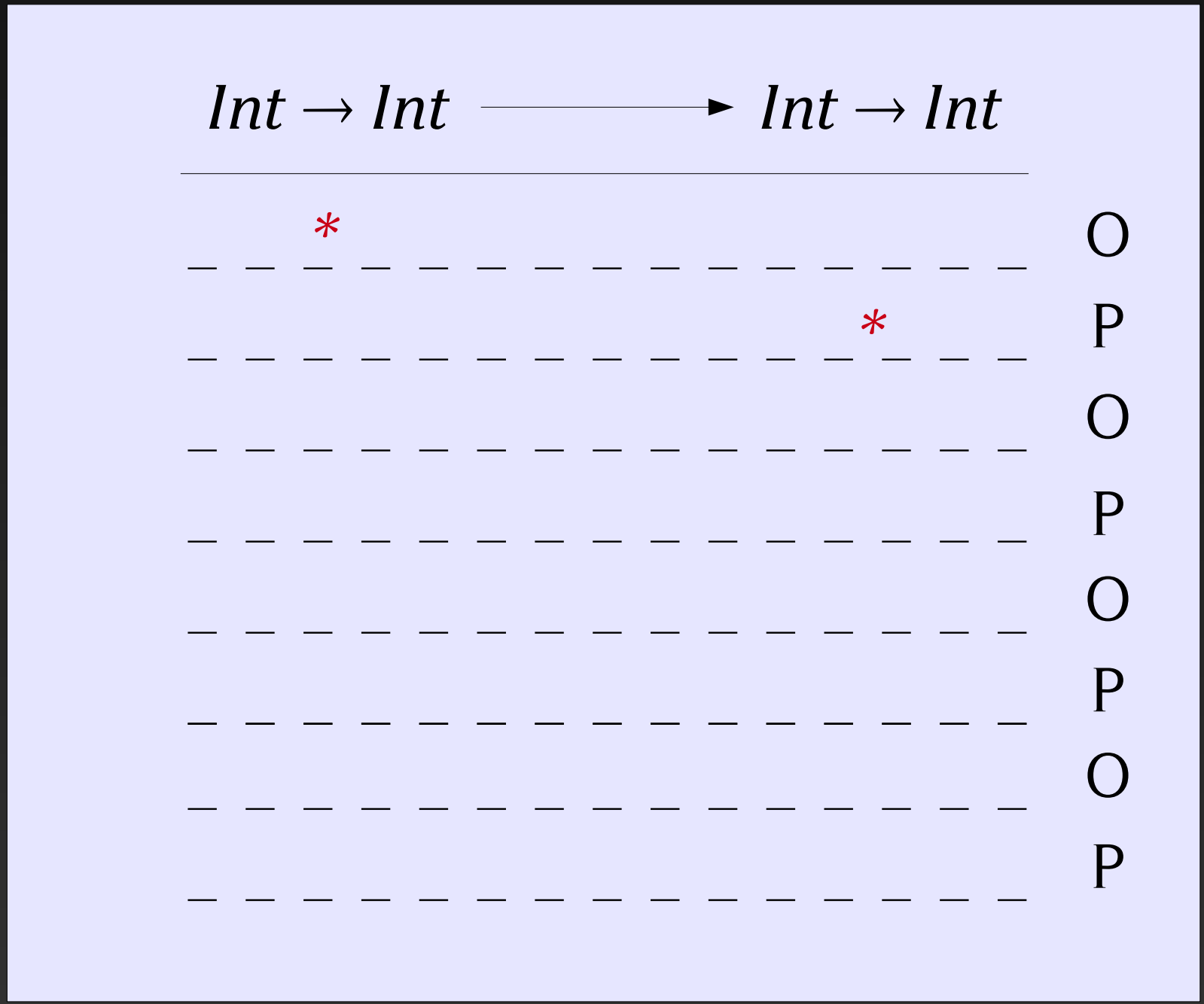
$\texttt{f : int} \to \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \to \texttt{int}$



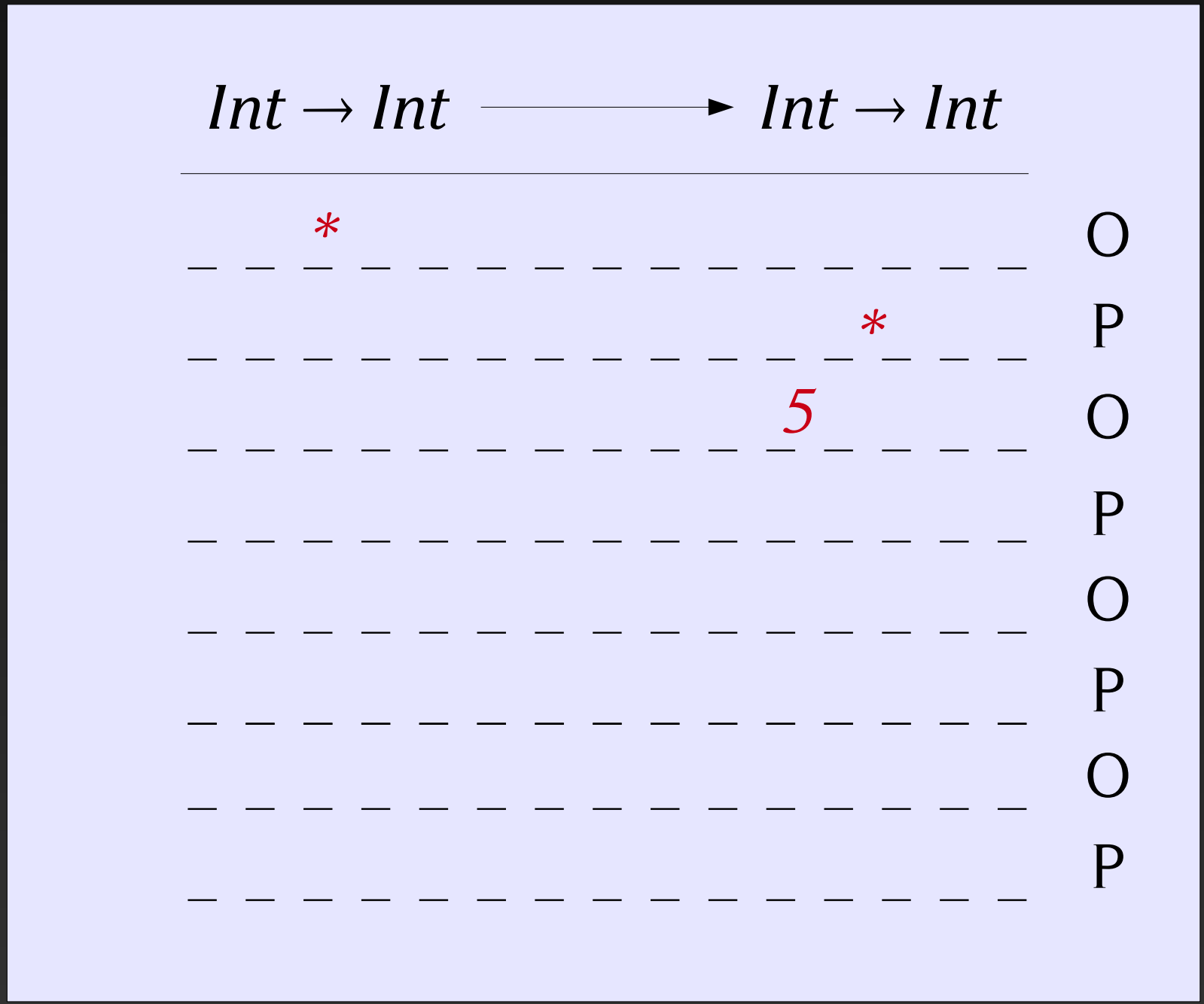
# Example



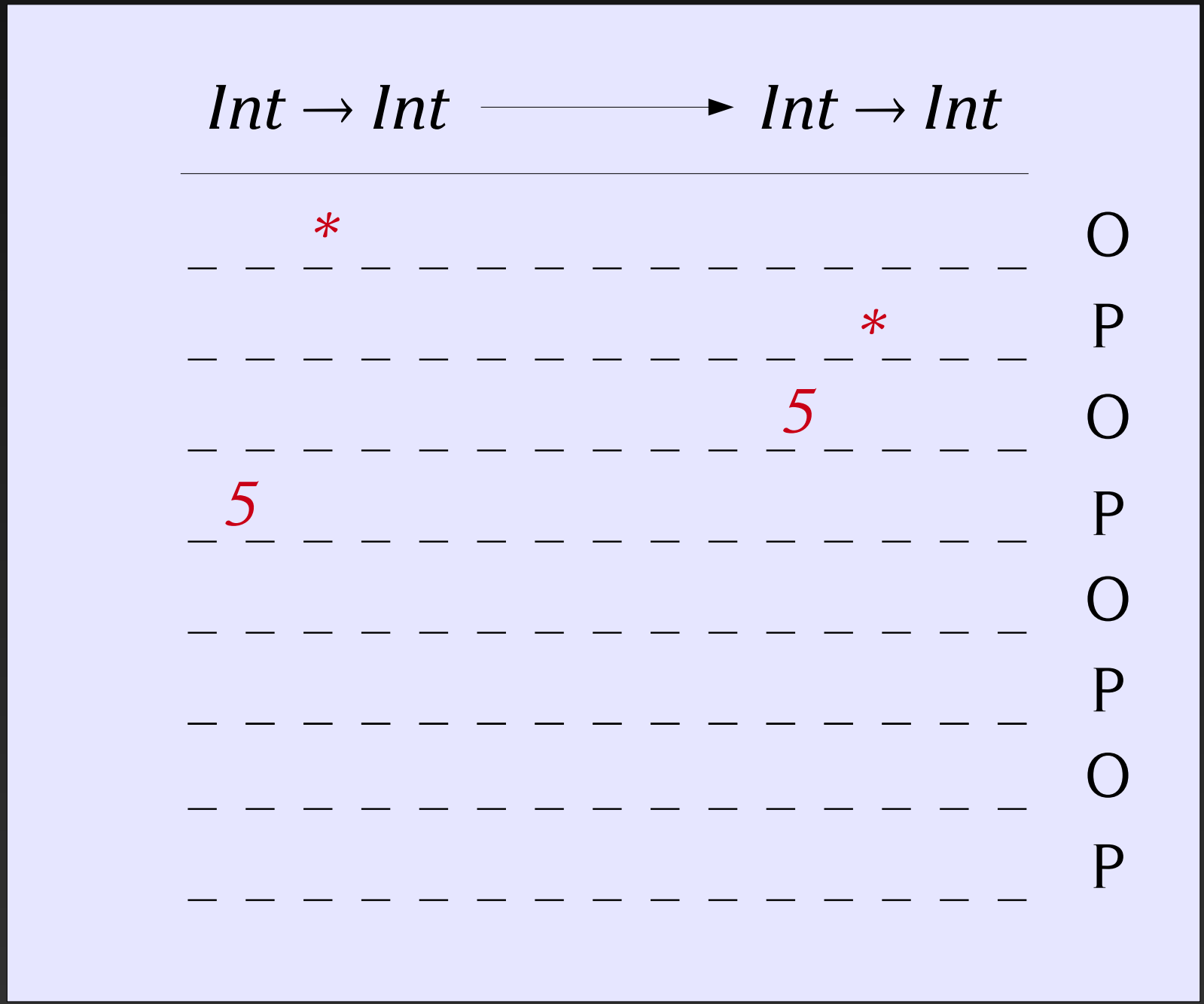
# Example



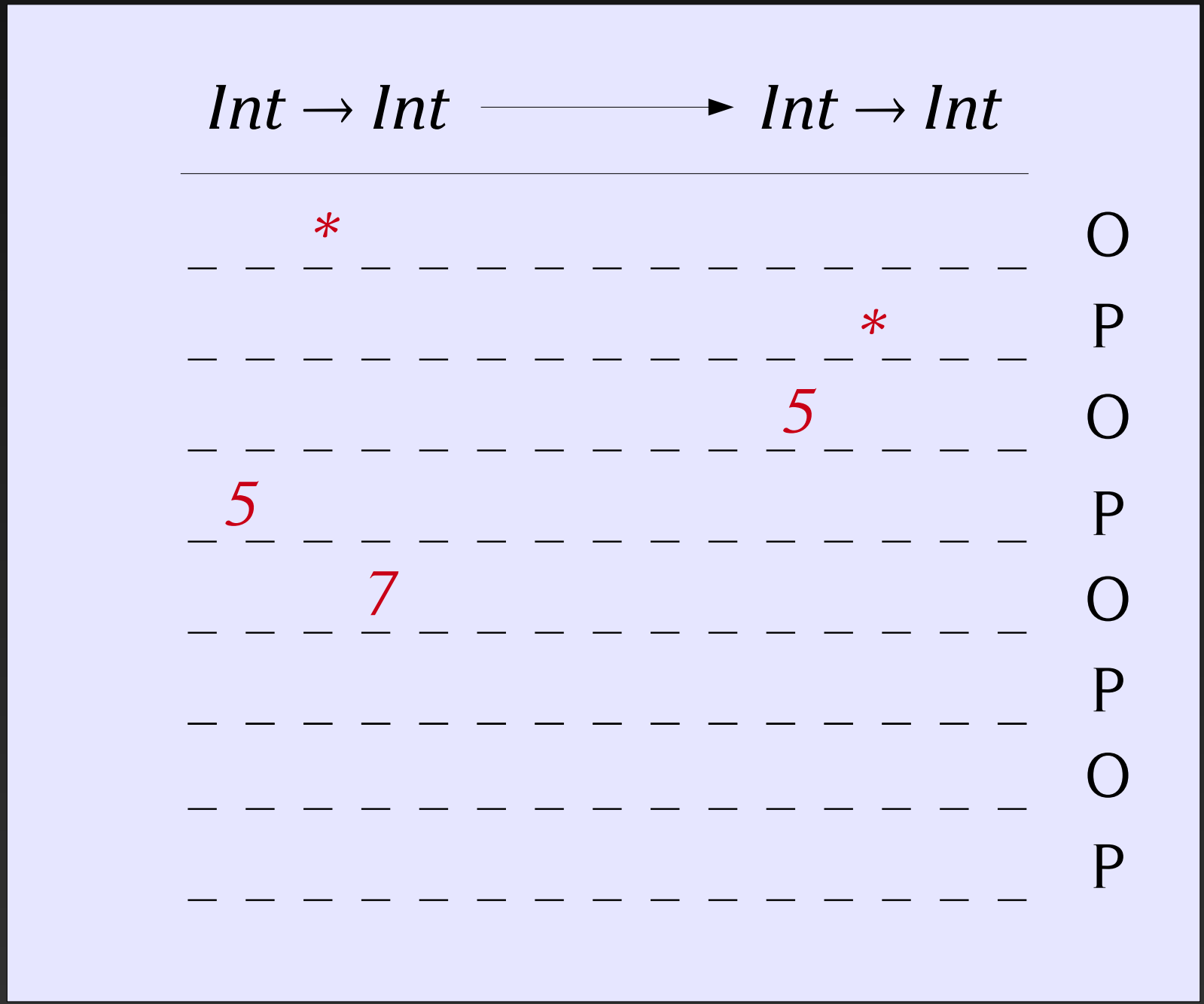
# Example



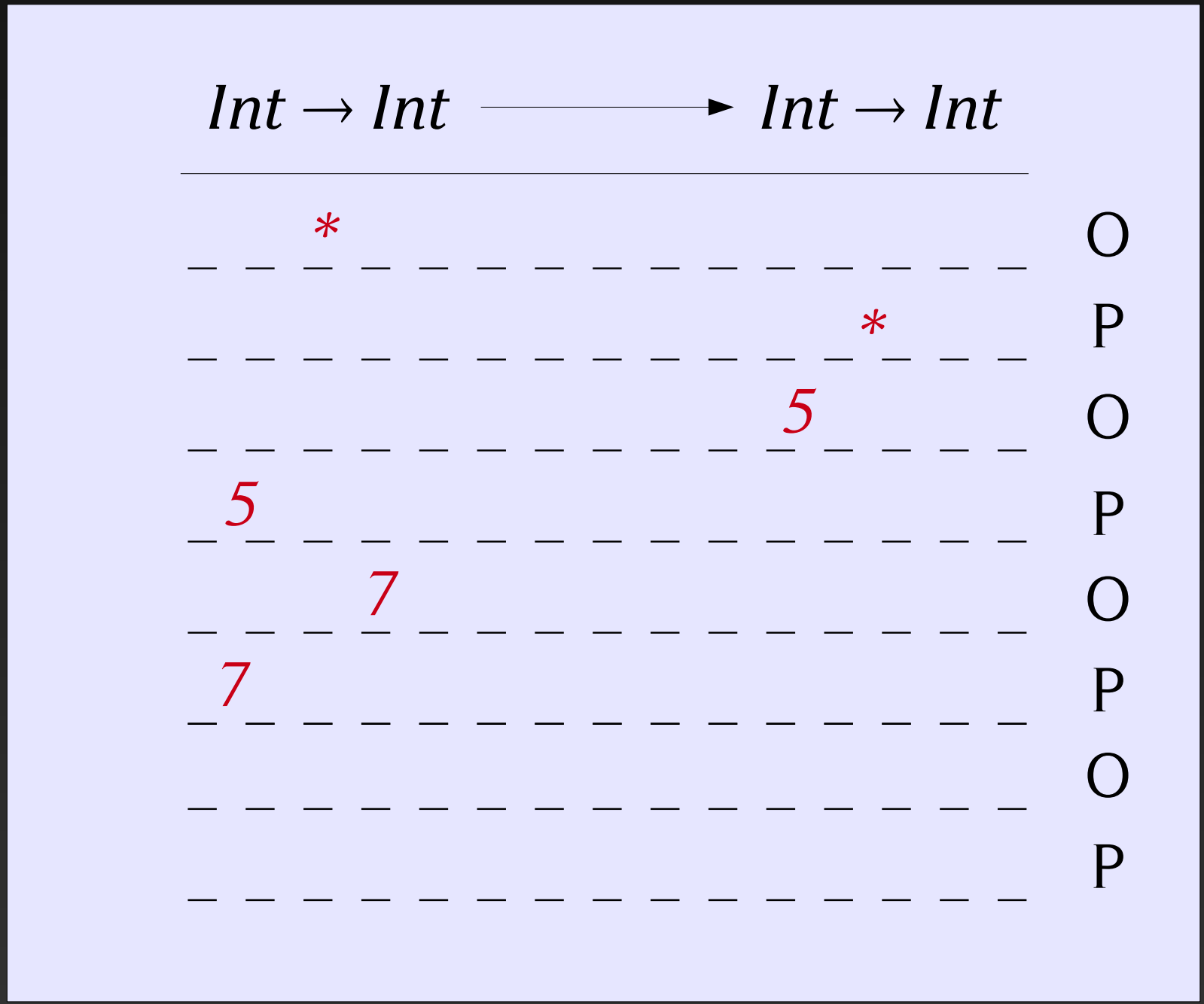
# Example



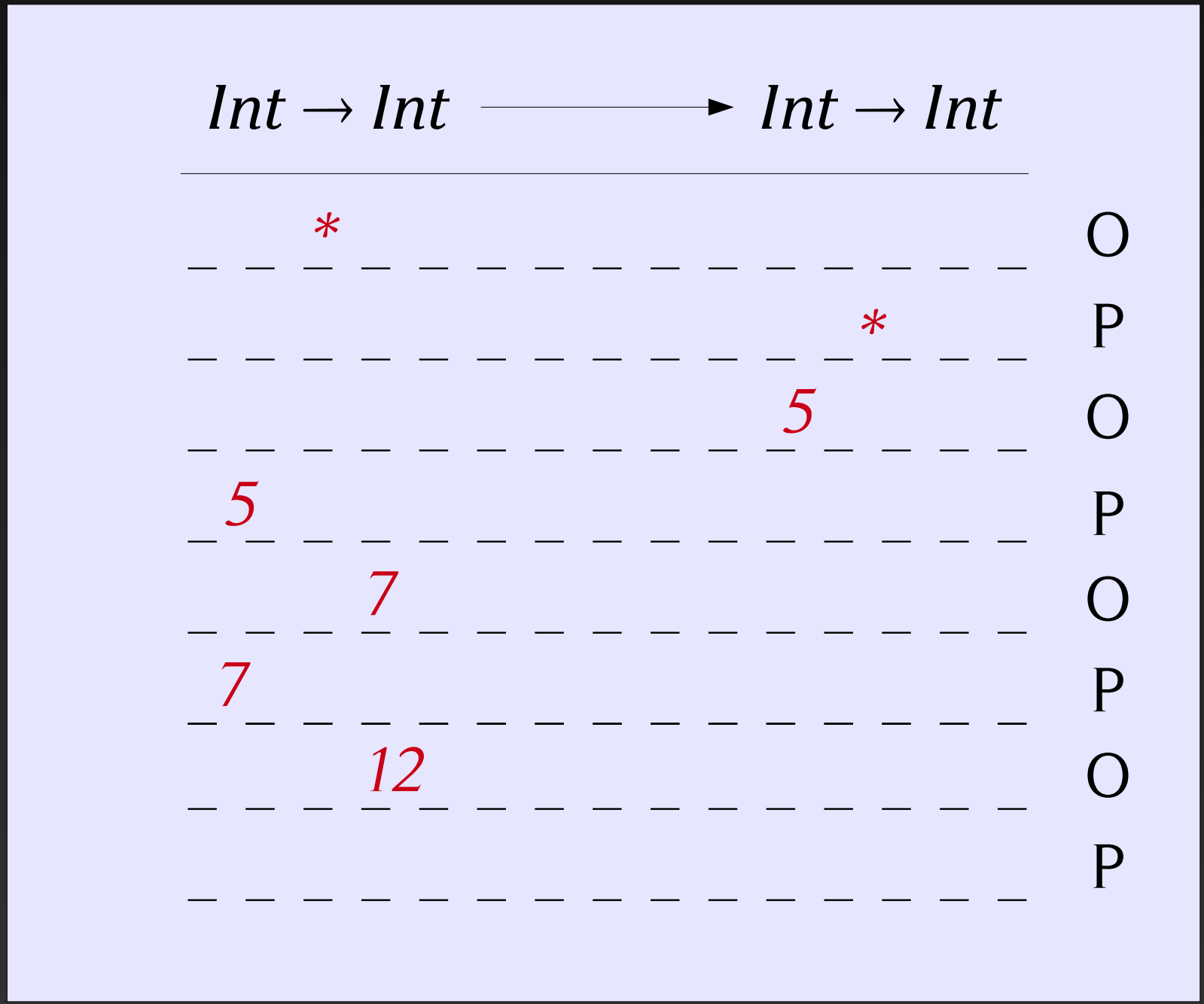
# Example



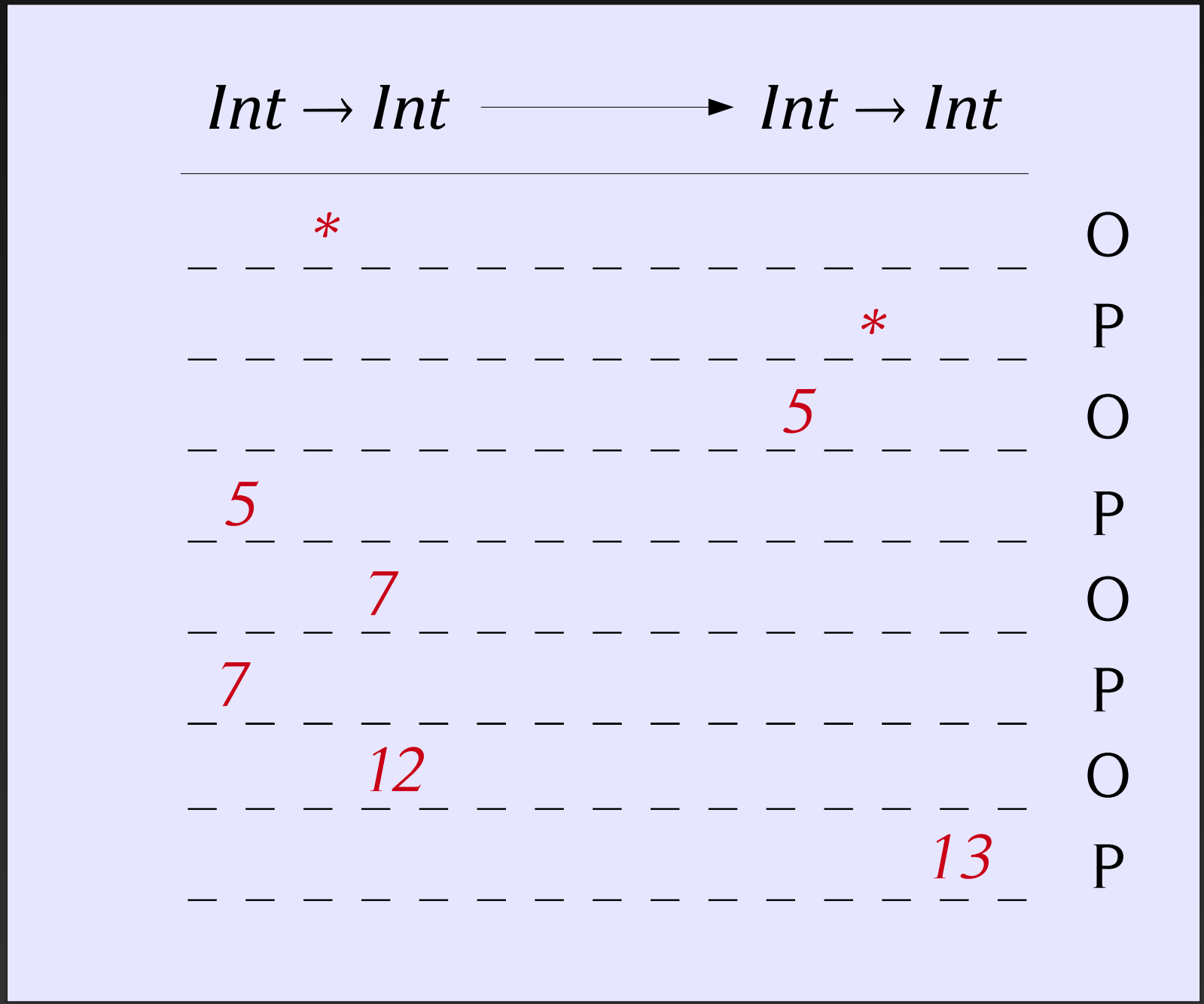
# Example



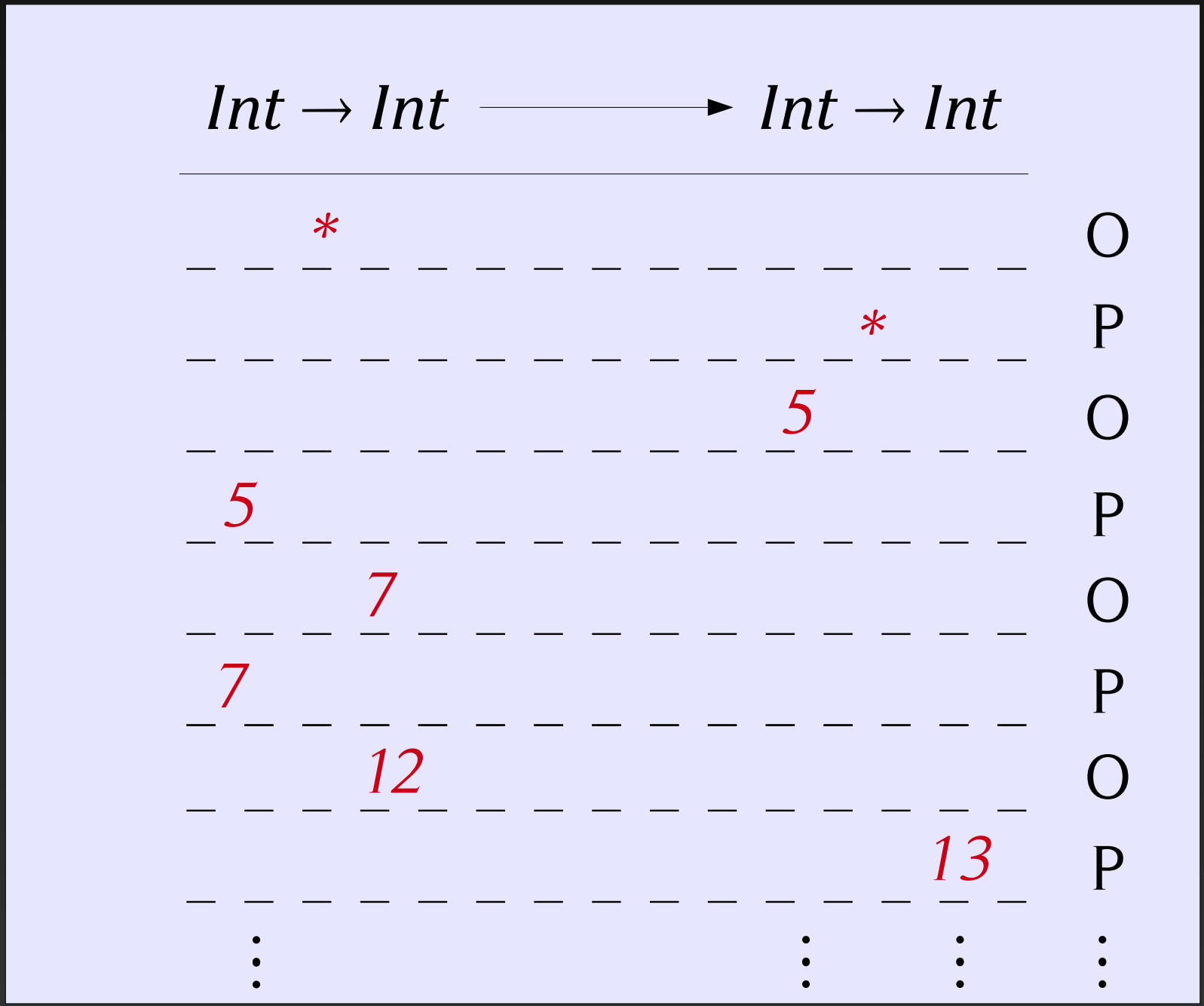
# Example



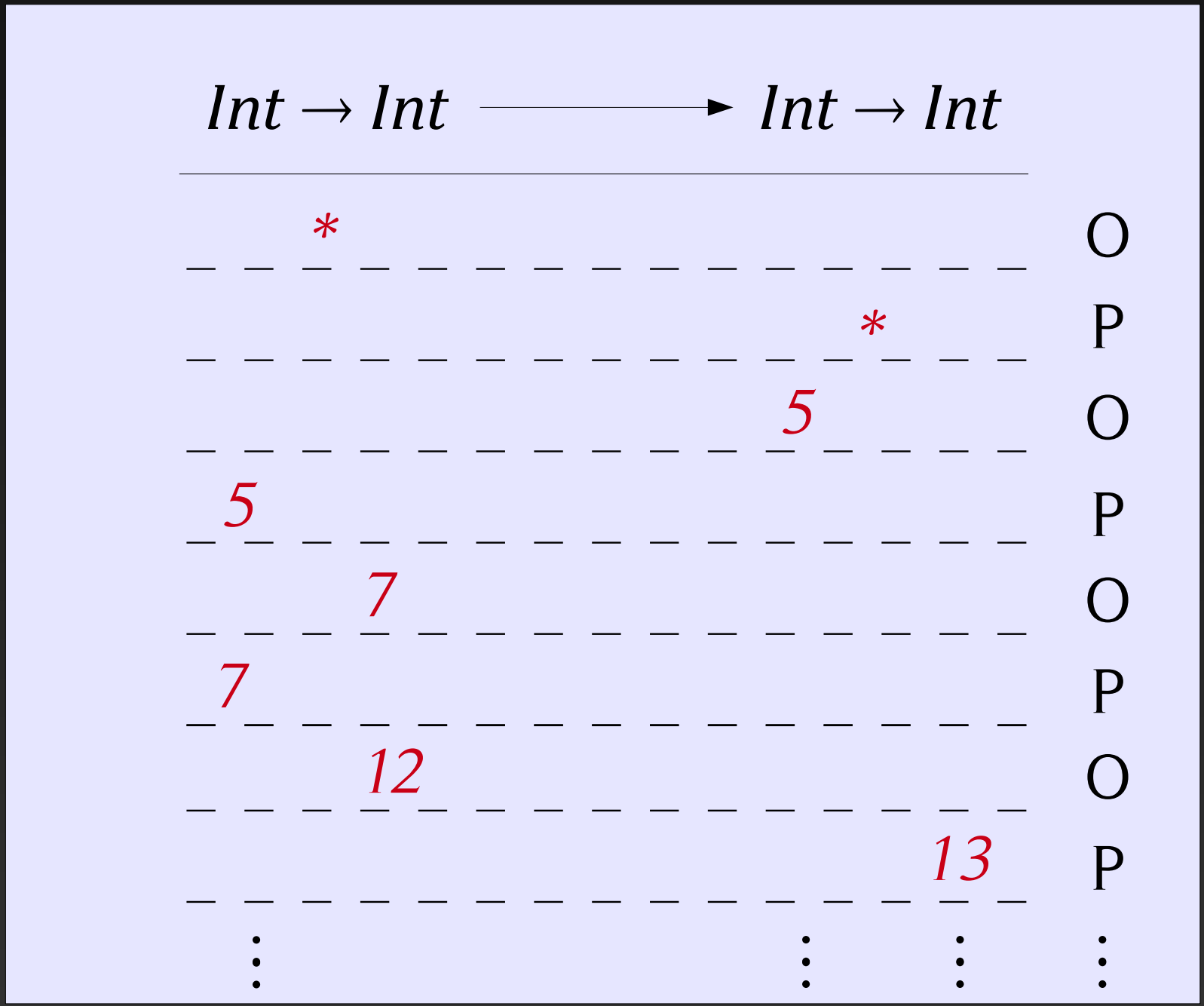
# Example



# 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:
  - *Proponent (the program)*
  - *Opponent* (the environment)
- Qualitative games ( ≠ Game Theory)
- Programs = *strategies* for Proponent

# Composition

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

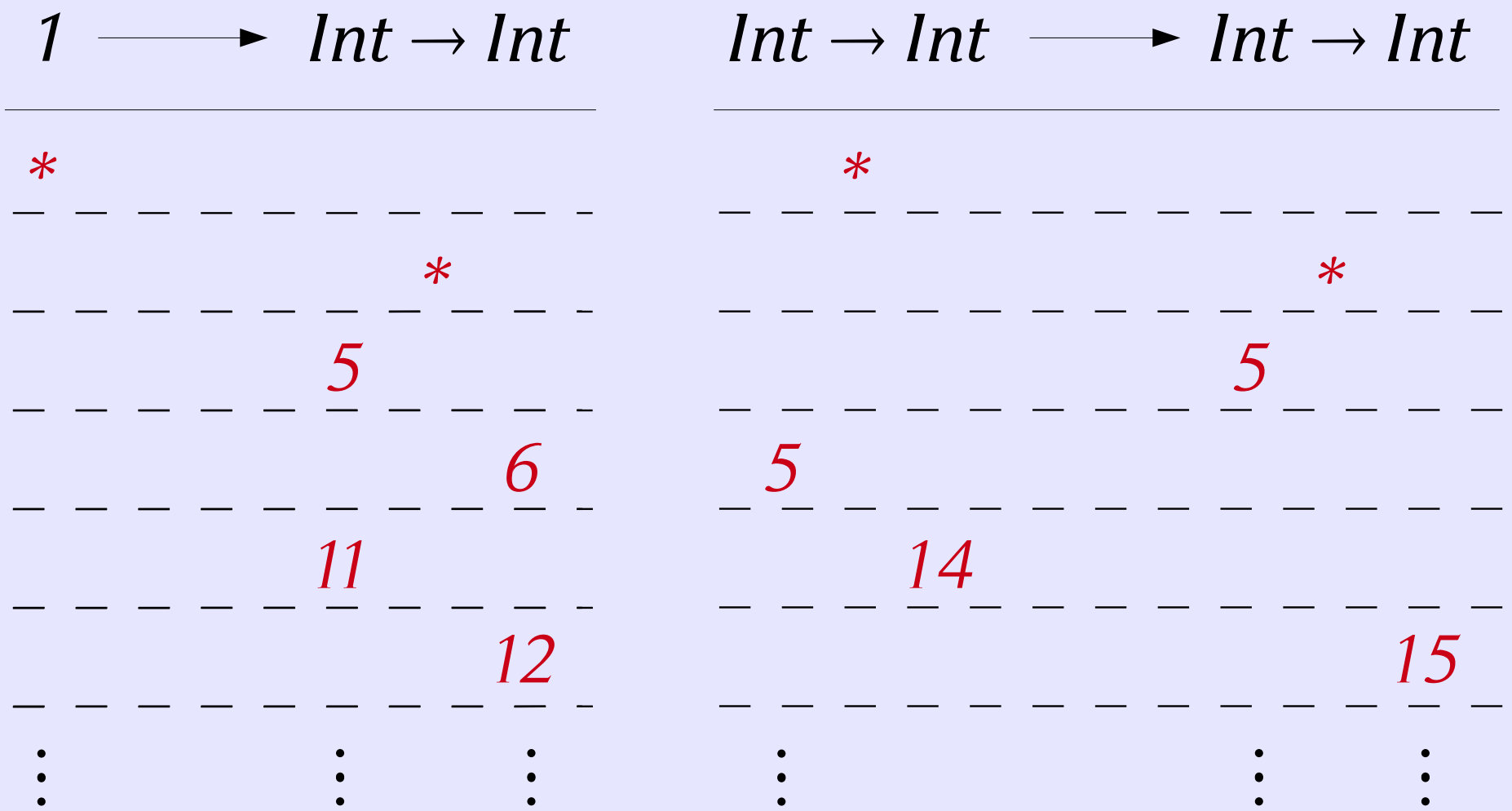
$f : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

# Composition

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

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

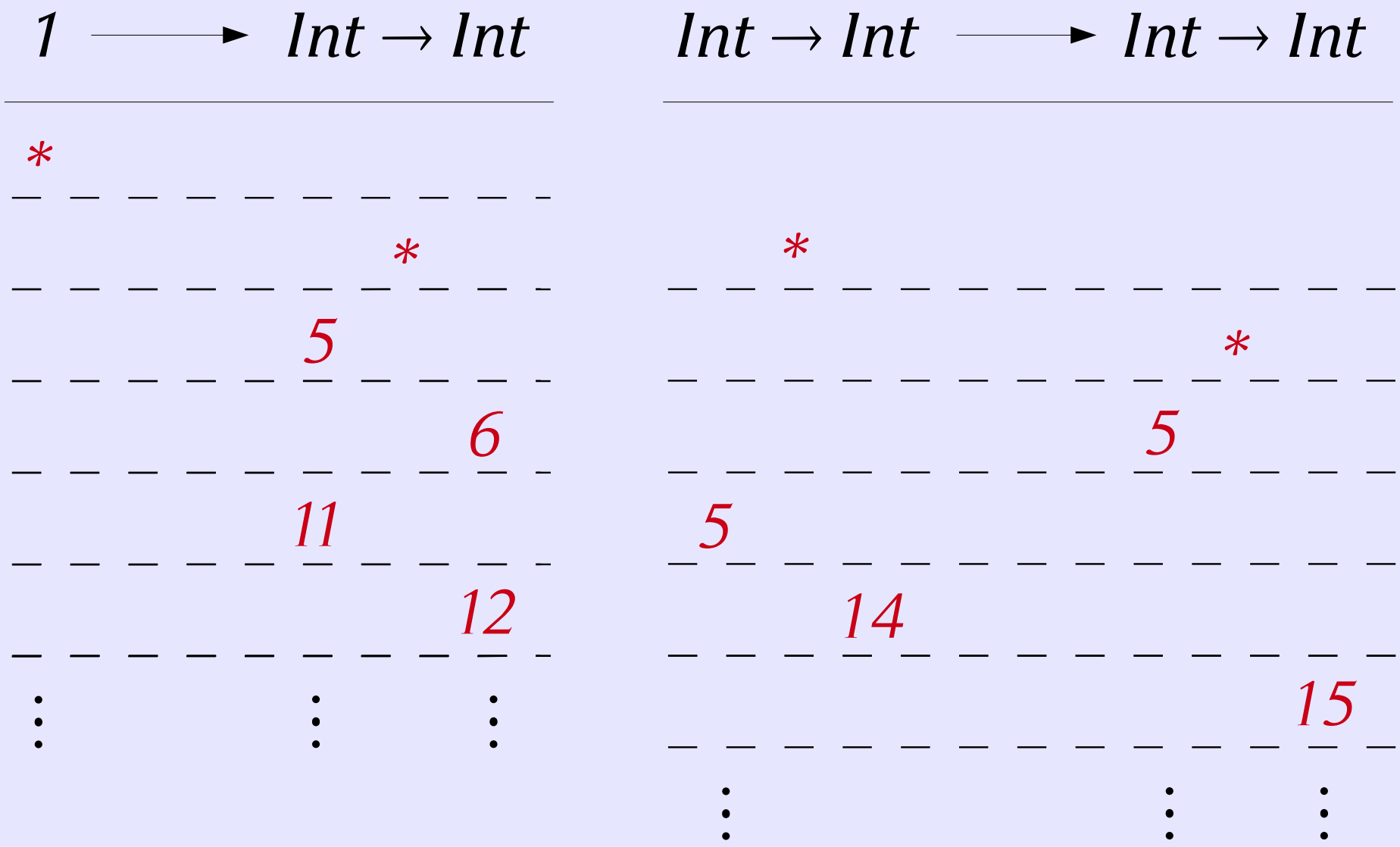
# Composition



`⊢ λx.x+1 : int ⟶ int`

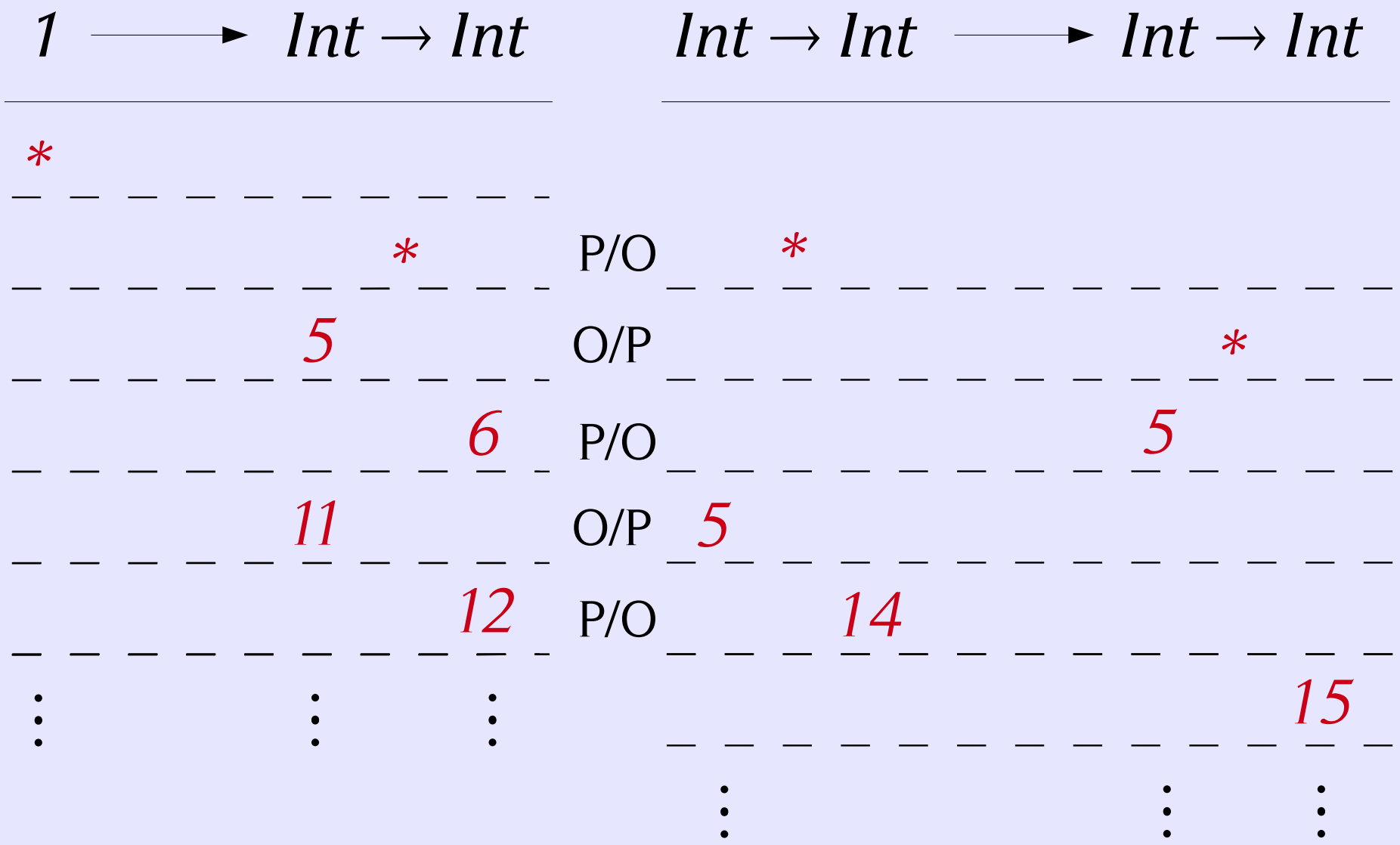
`f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int`

# Composition



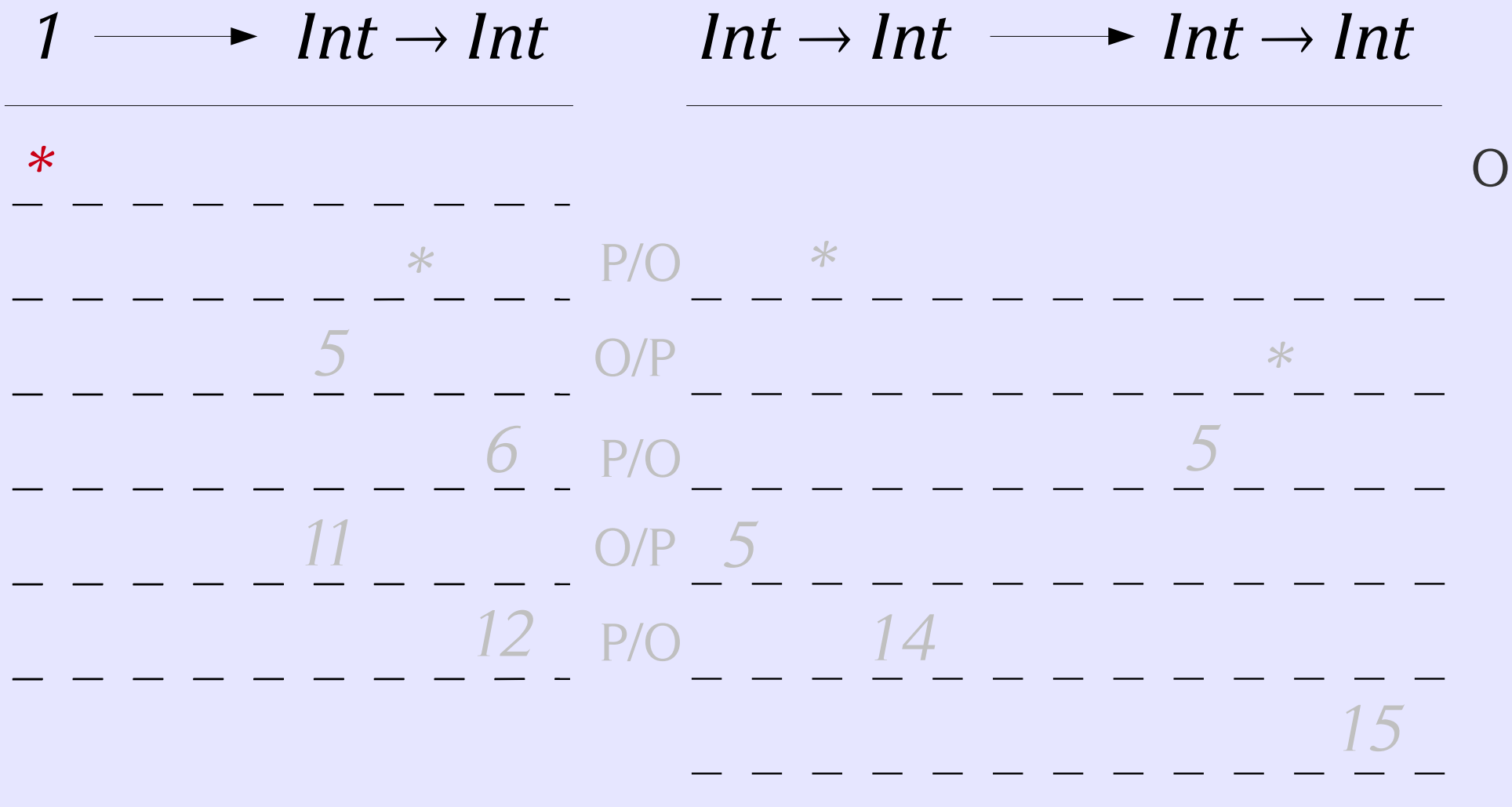
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

# Composition



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

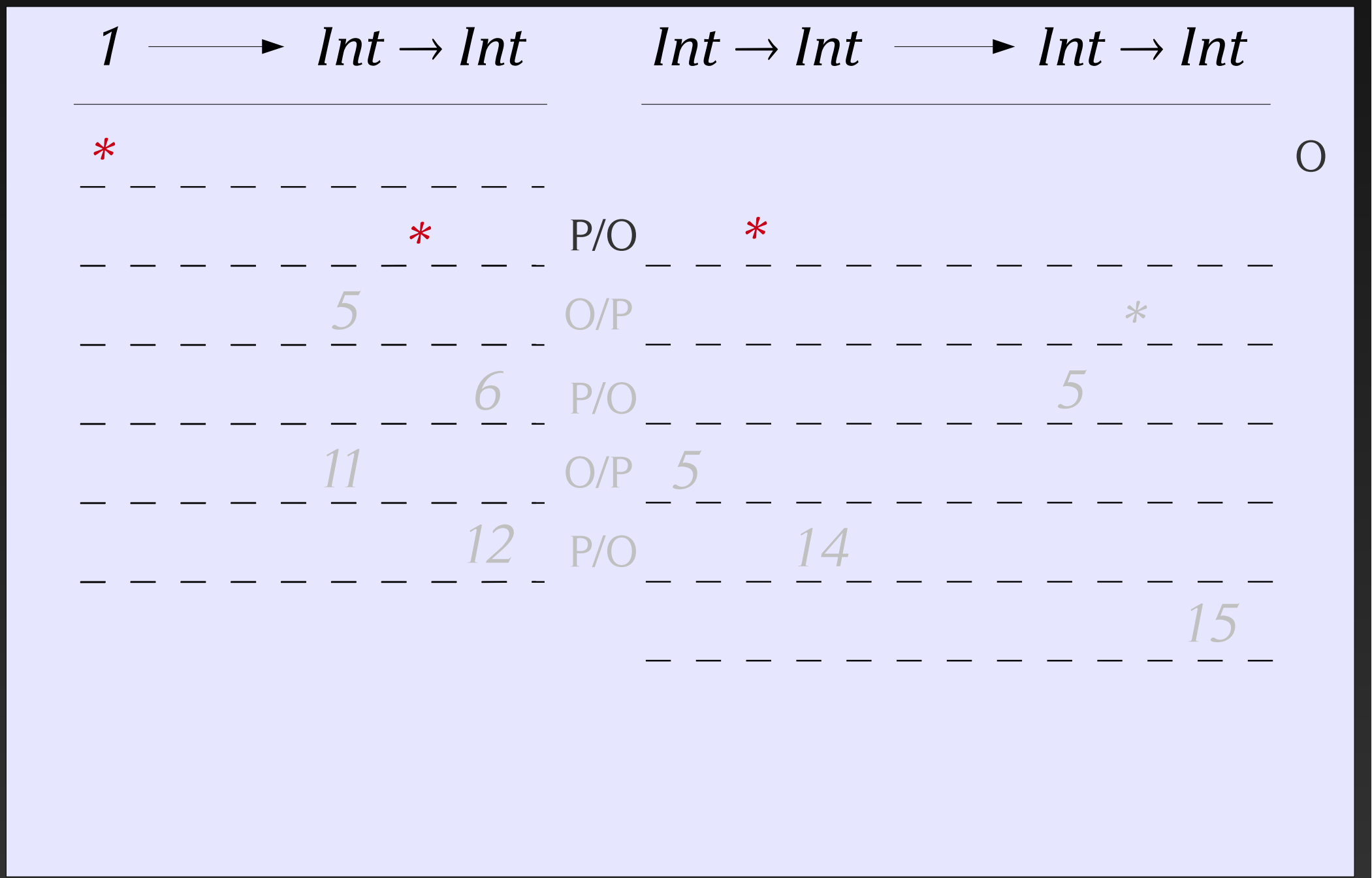
# Composition



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

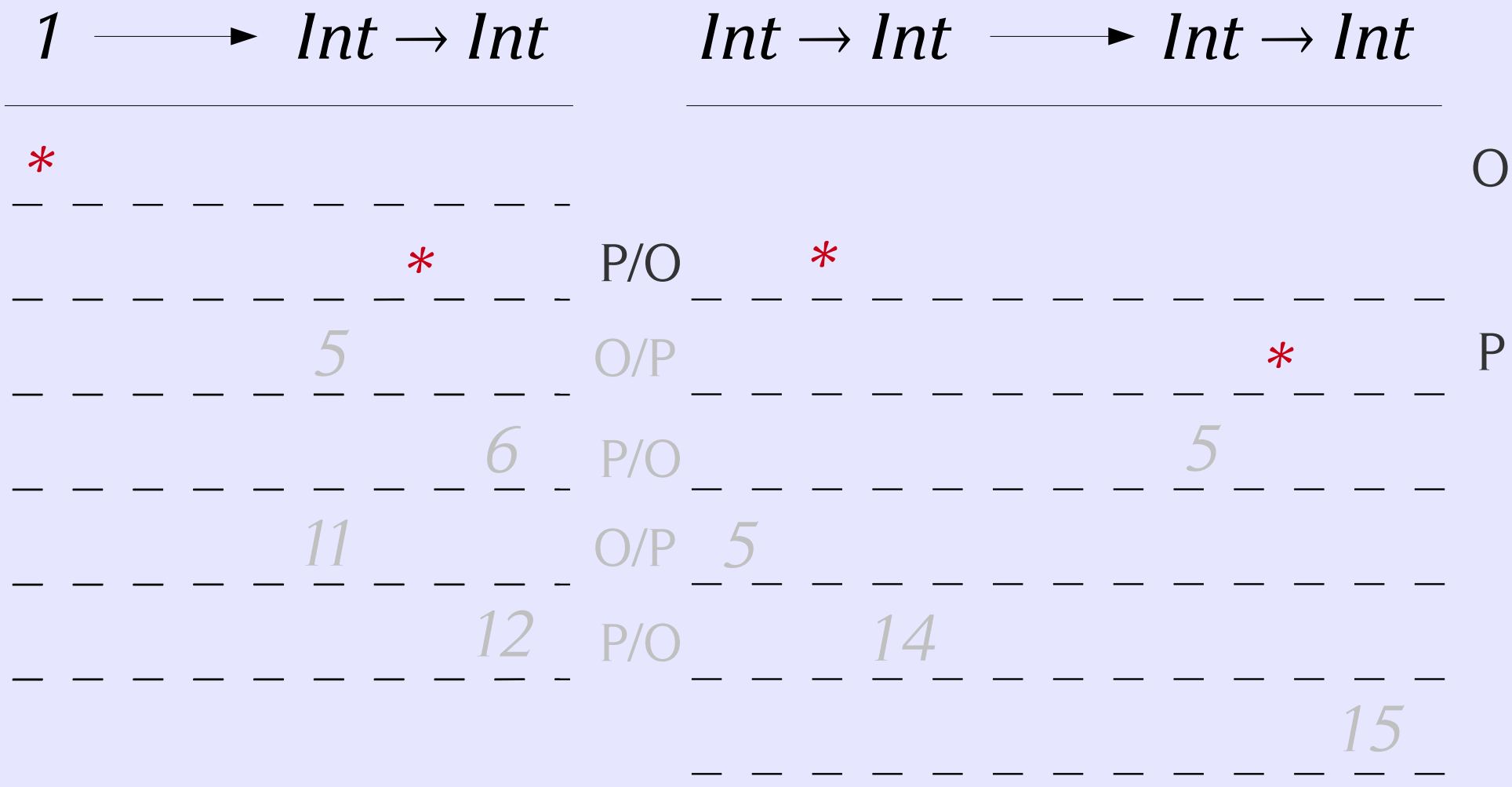
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

# Composition



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

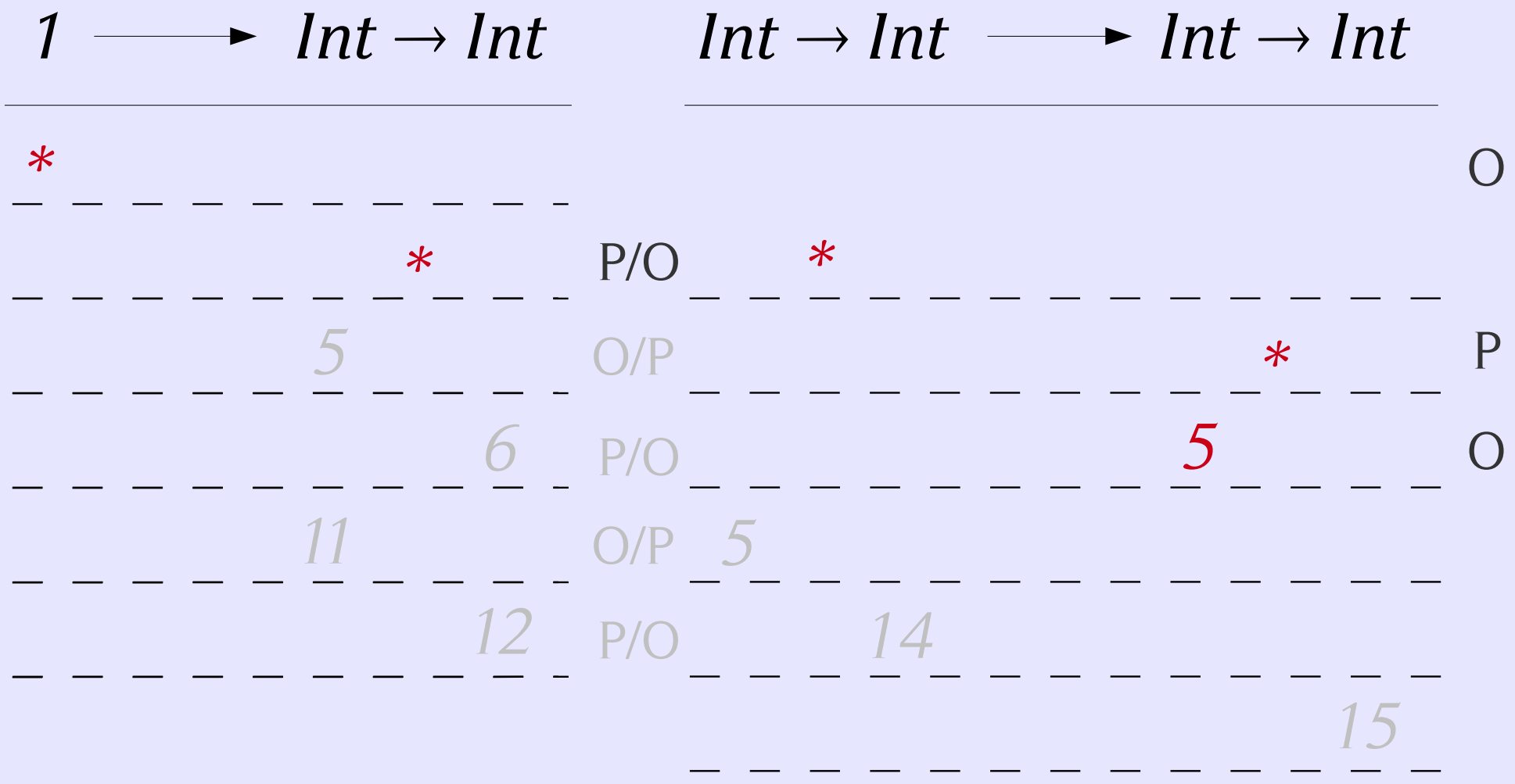
# Composition



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

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

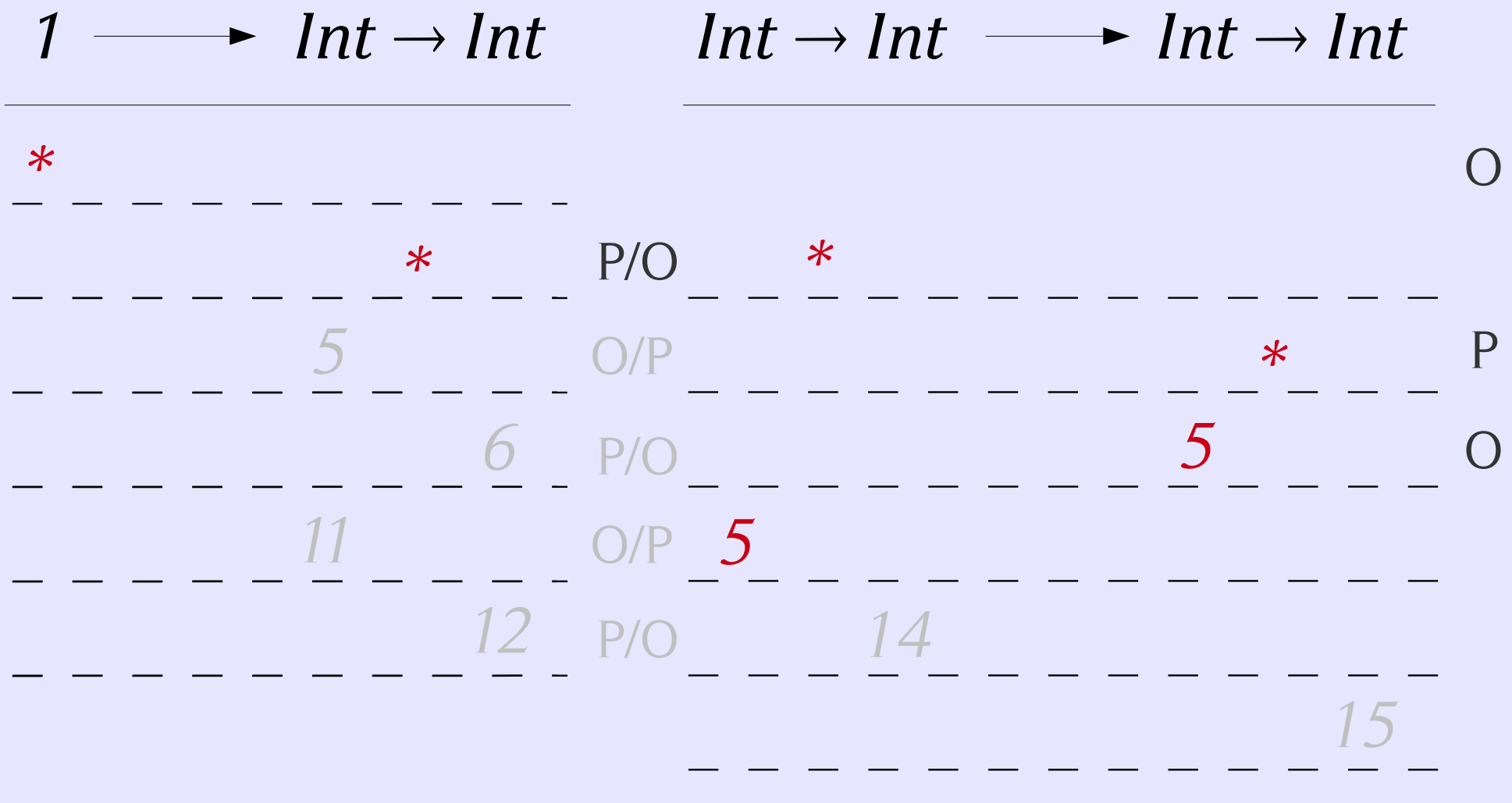
# Composition



`⊢ λx.x+1 : int ⟶ int`

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

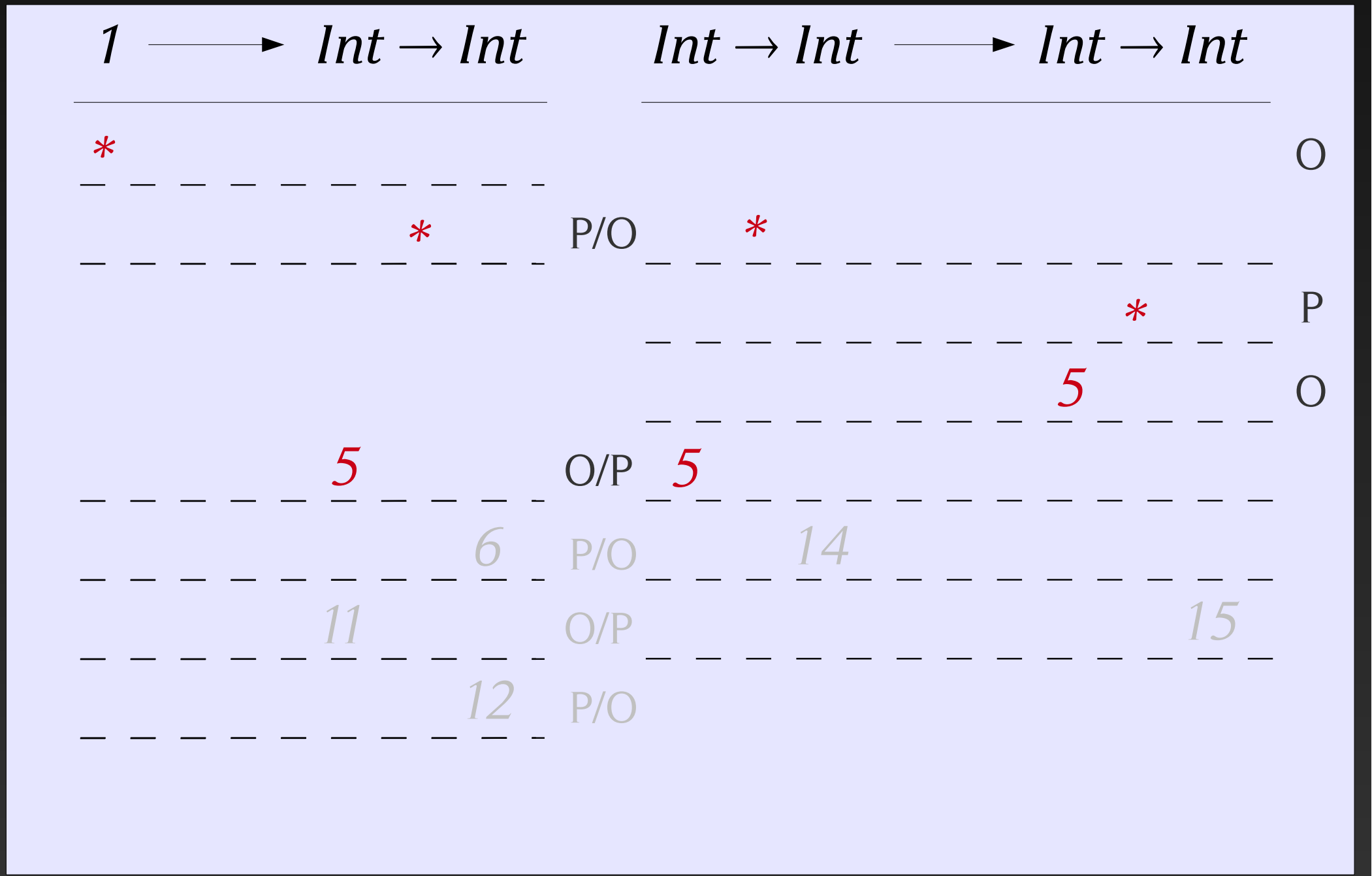
# Composition



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

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

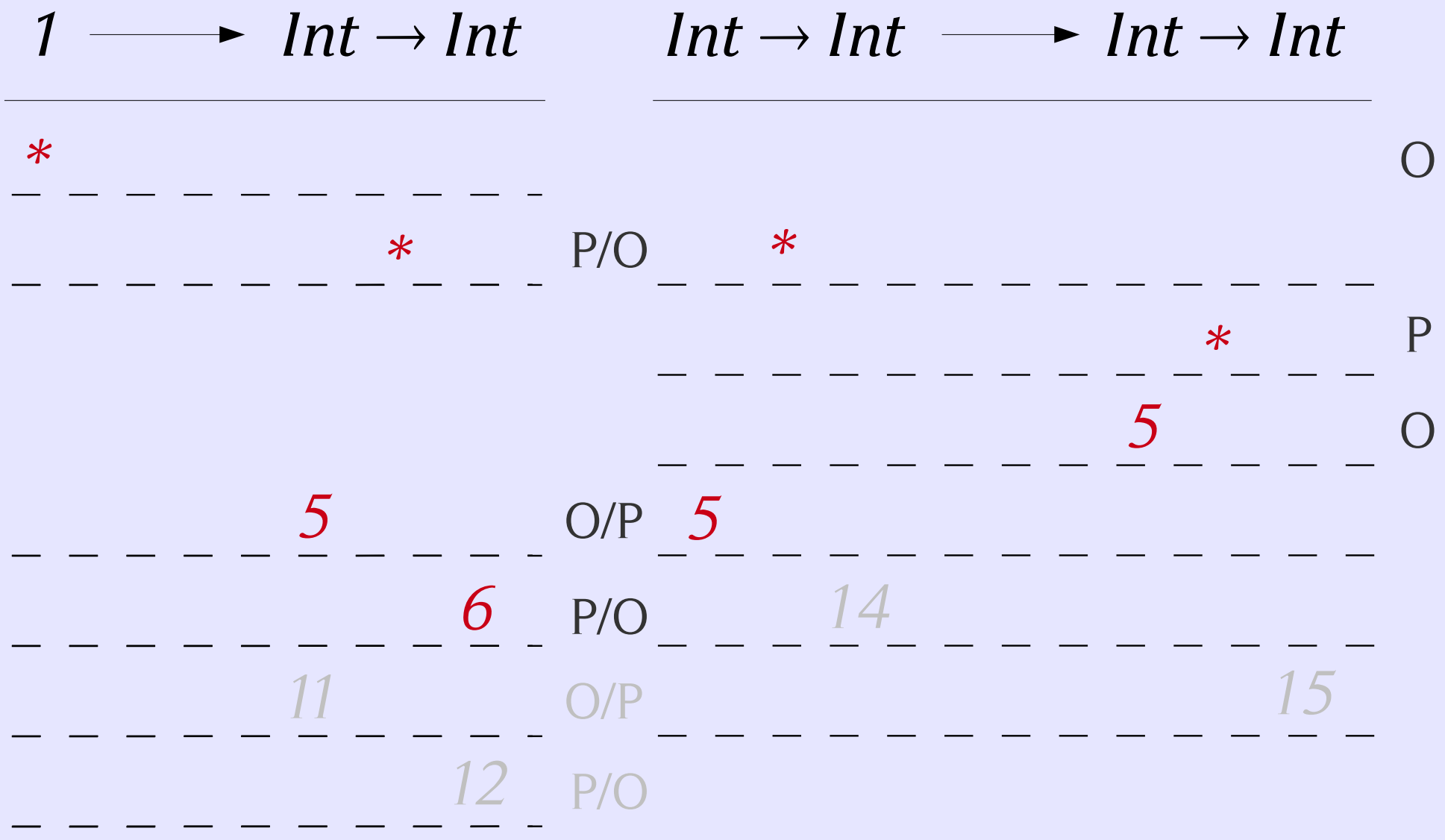
# Composition



⊢ λx.x+1 : int ⟶ int

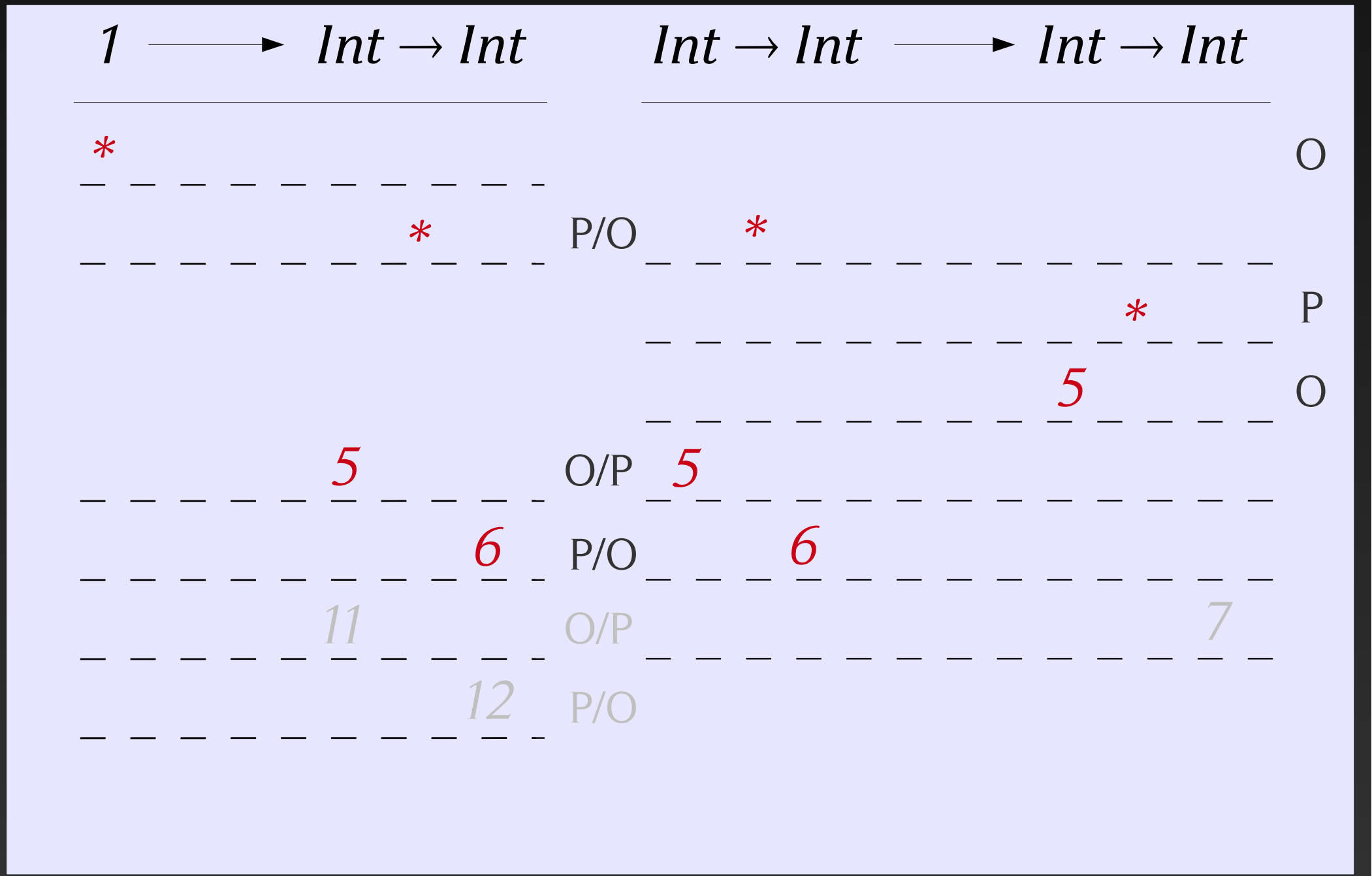
f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

# Composition



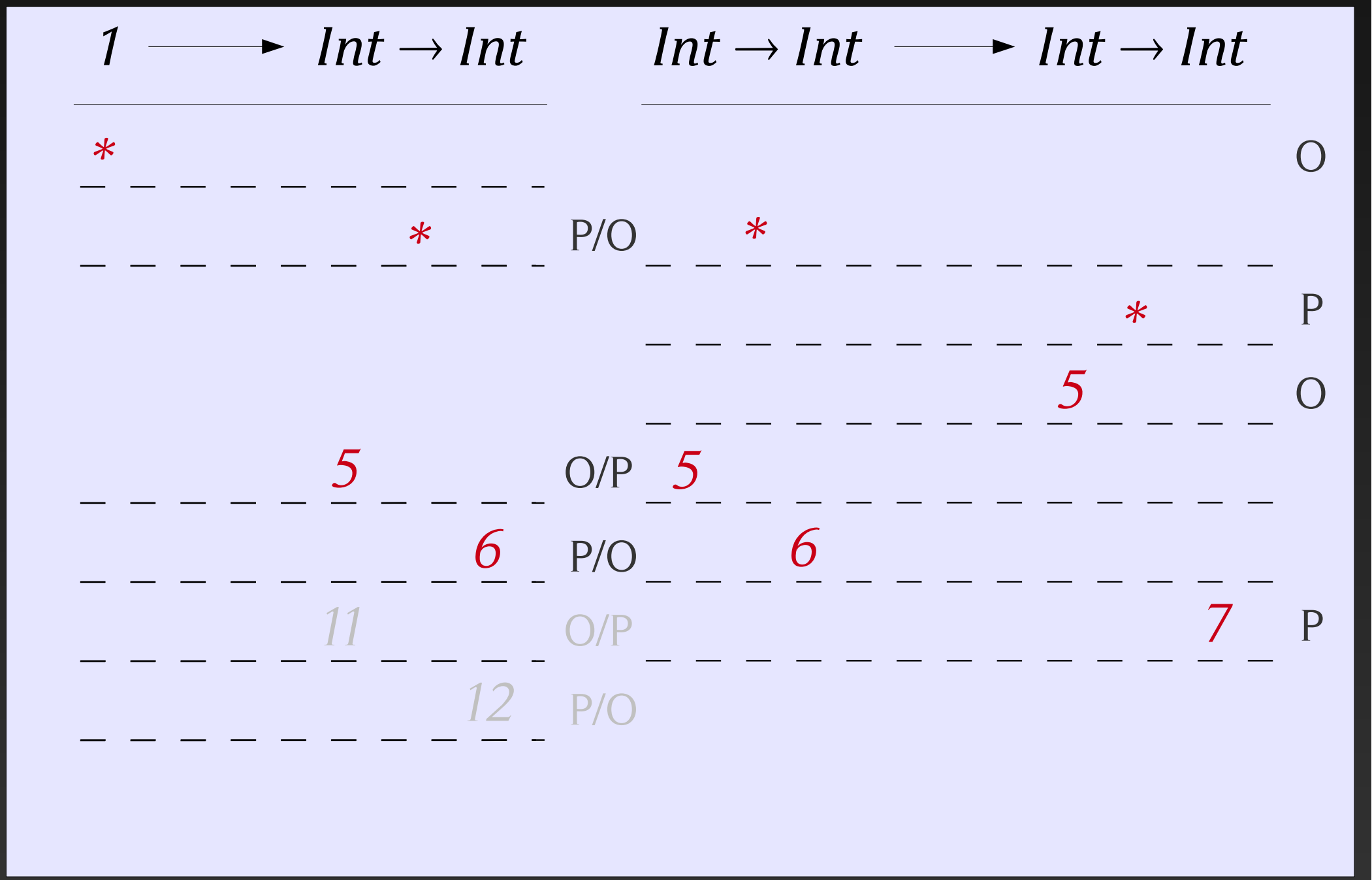
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

# Composition



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

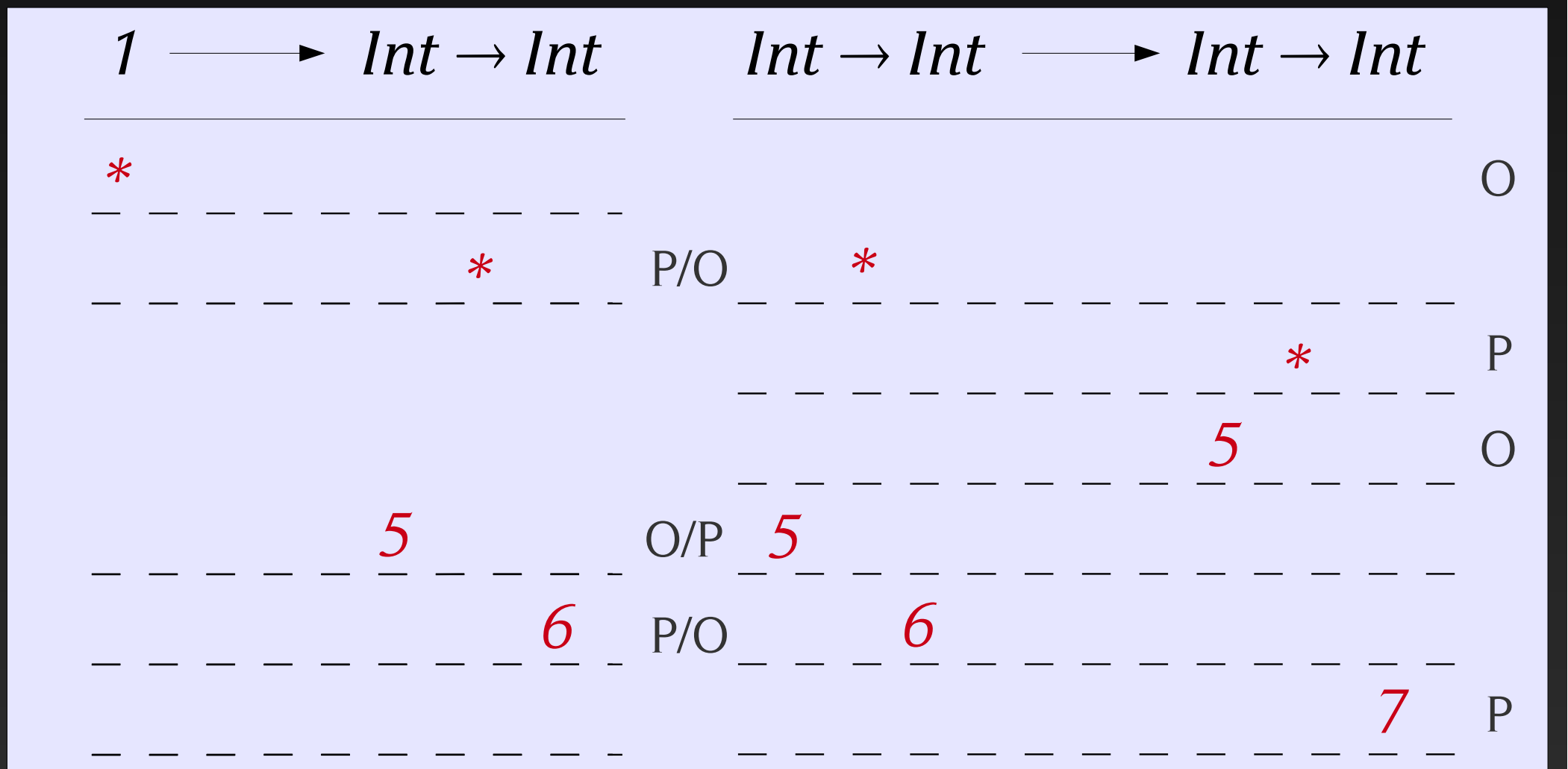
# Composition



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

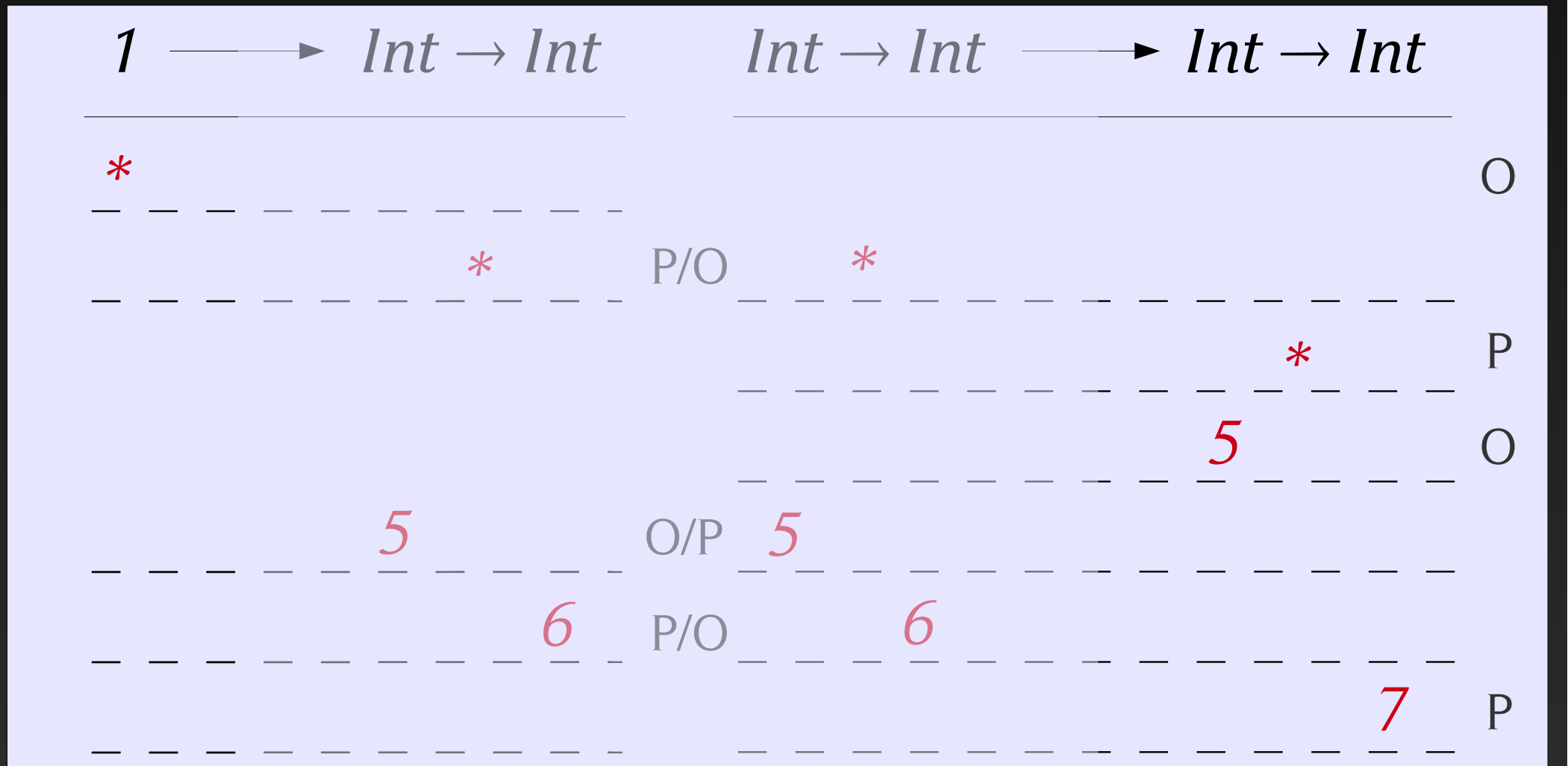
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

# Composition



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

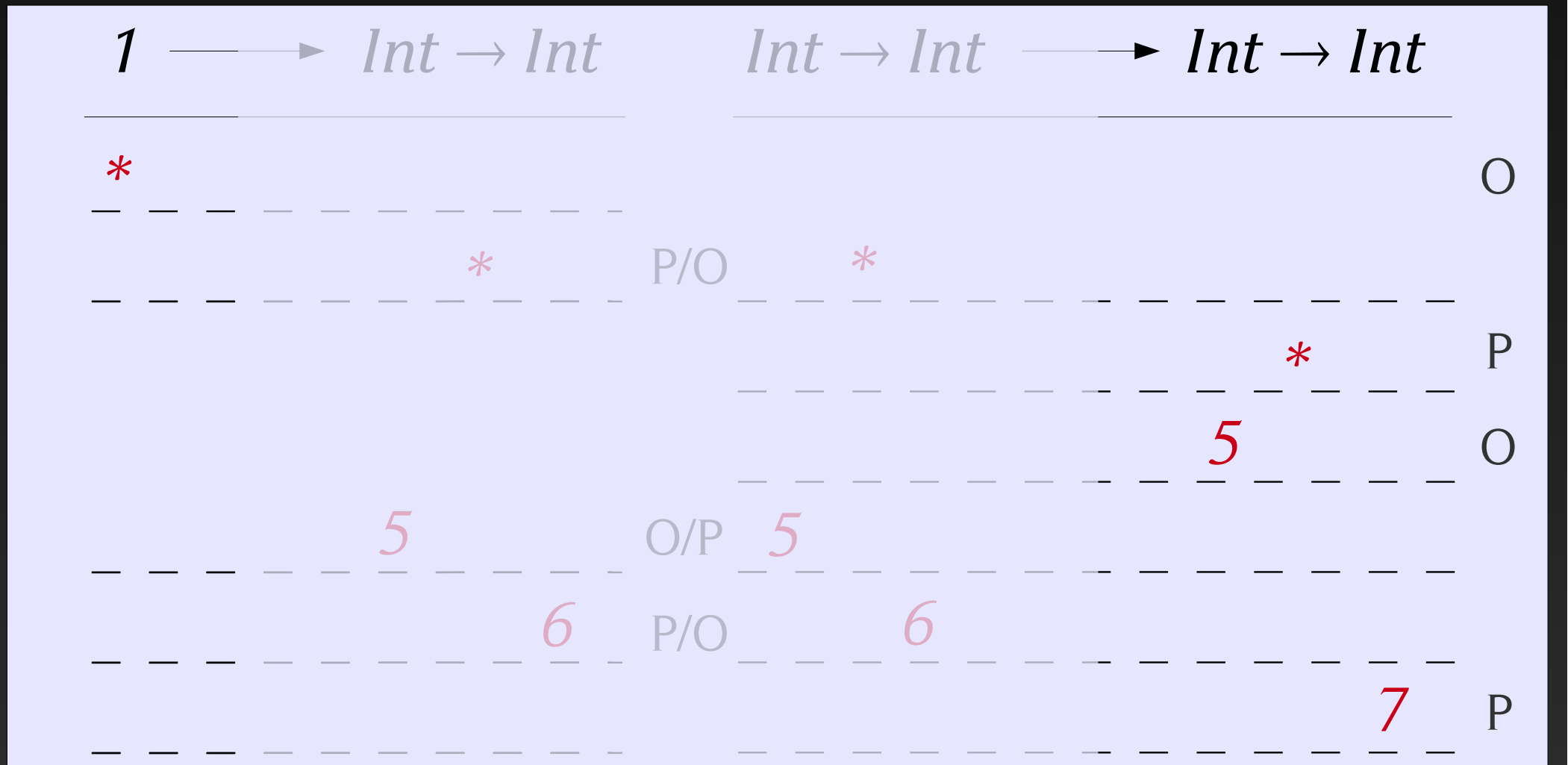
# Composition



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

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

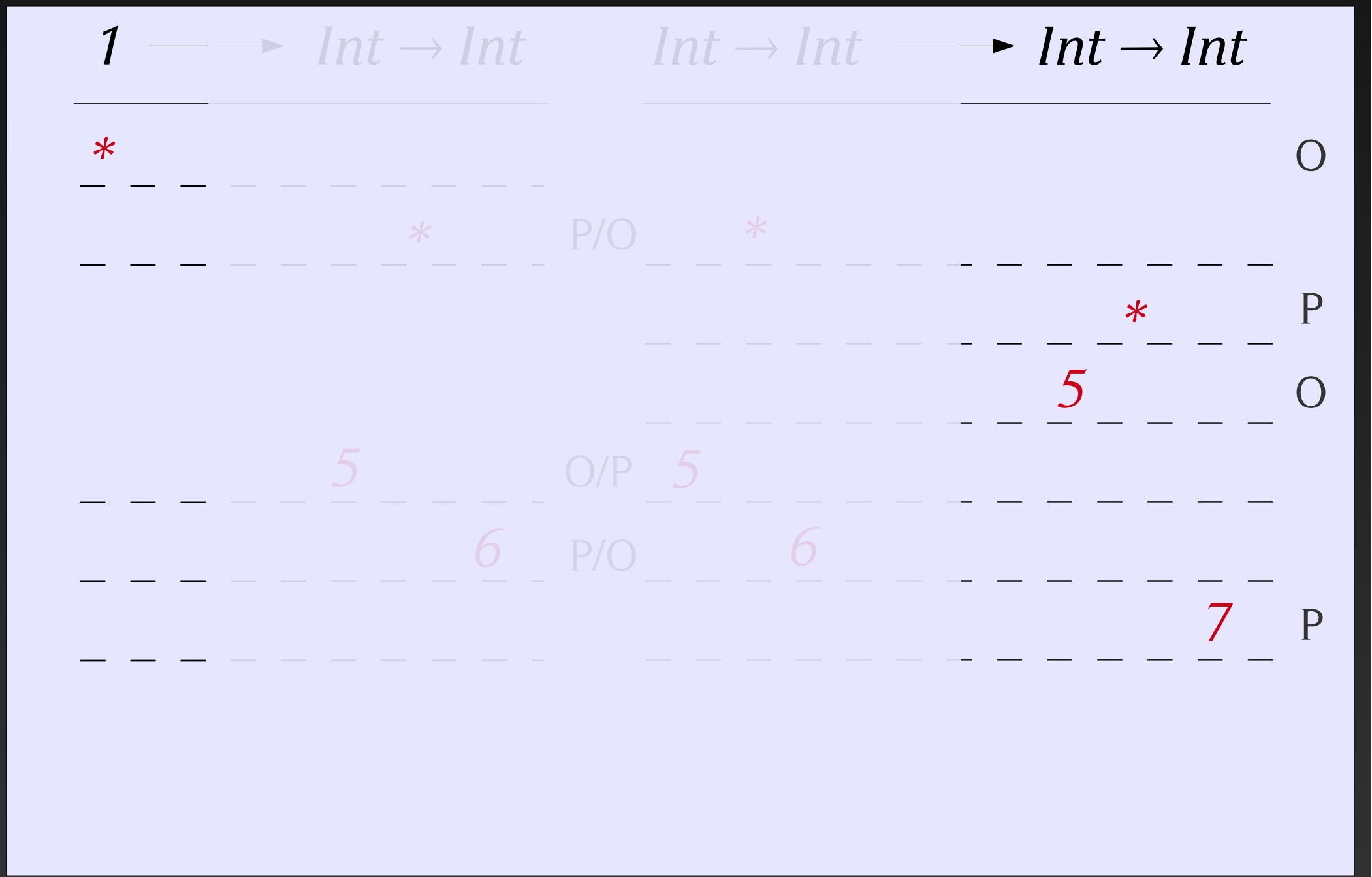
# Composition



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

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

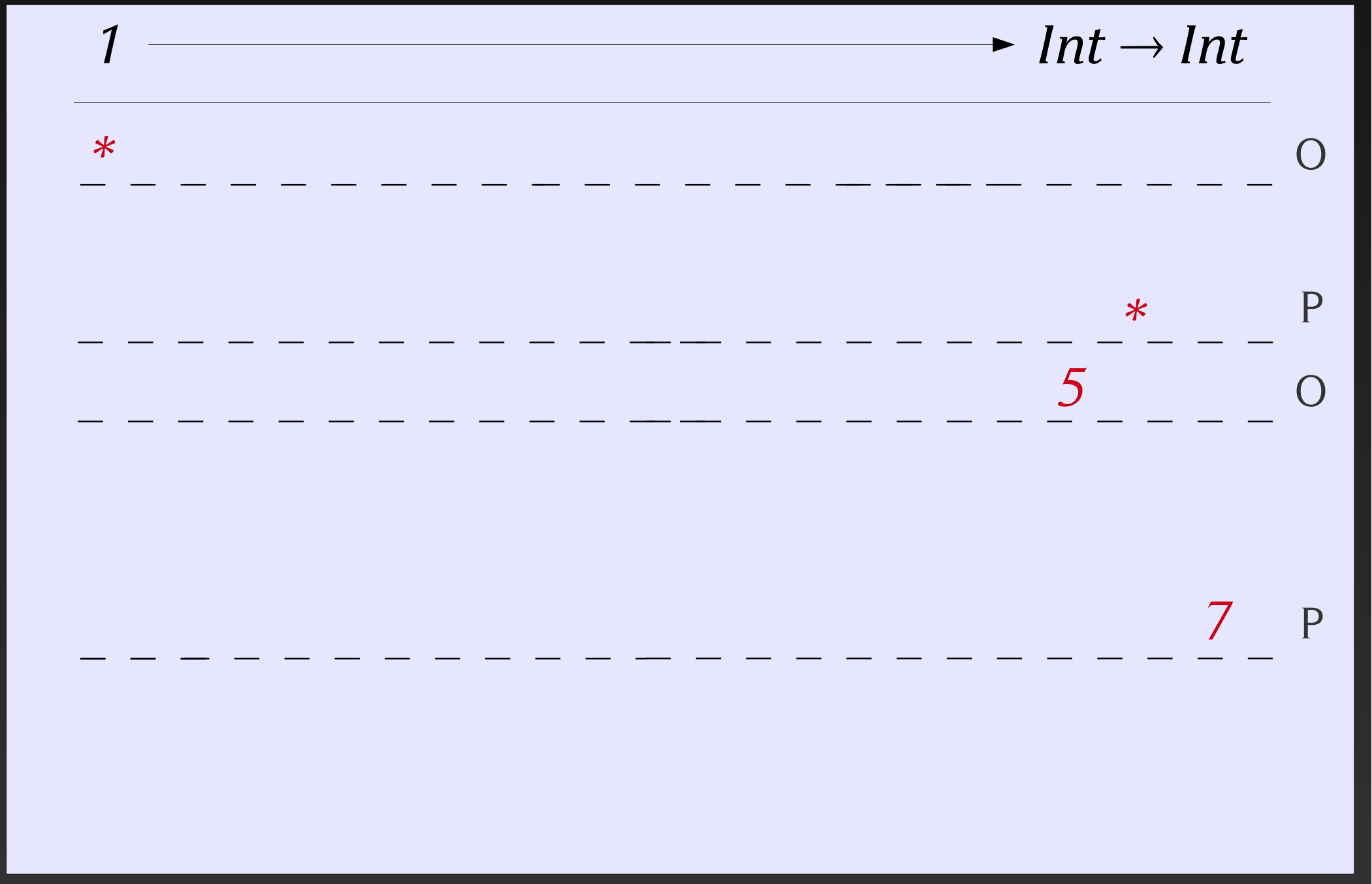
# Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

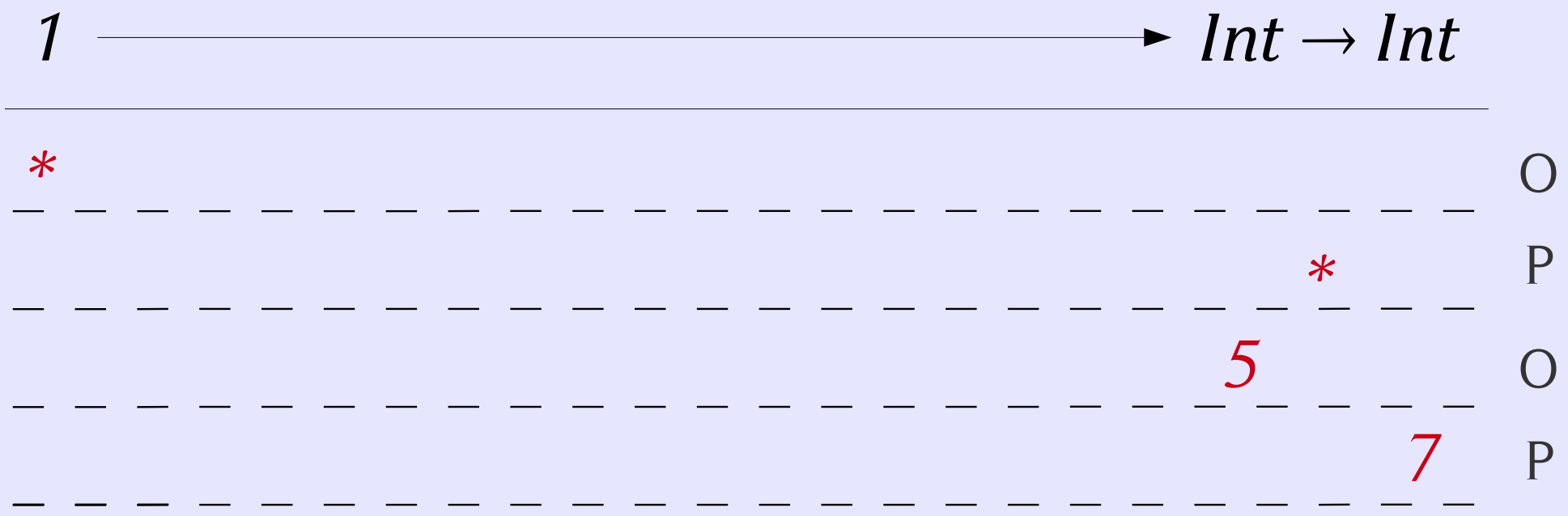
# Composition



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

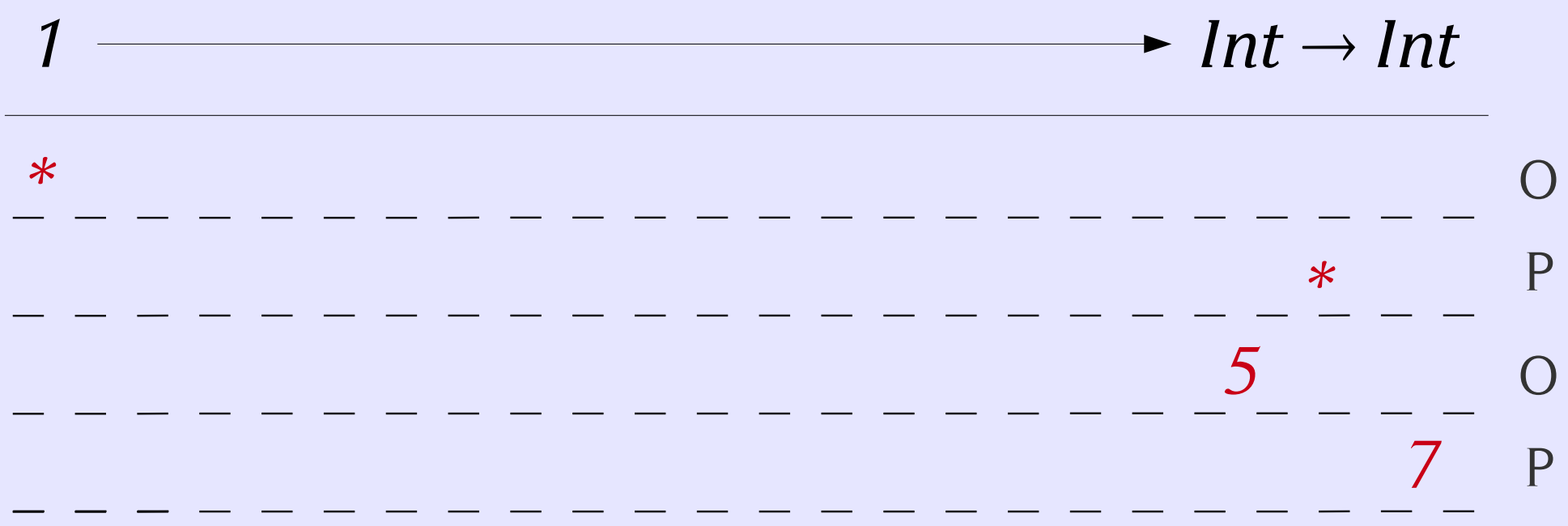
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

# Composition



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

# Composition

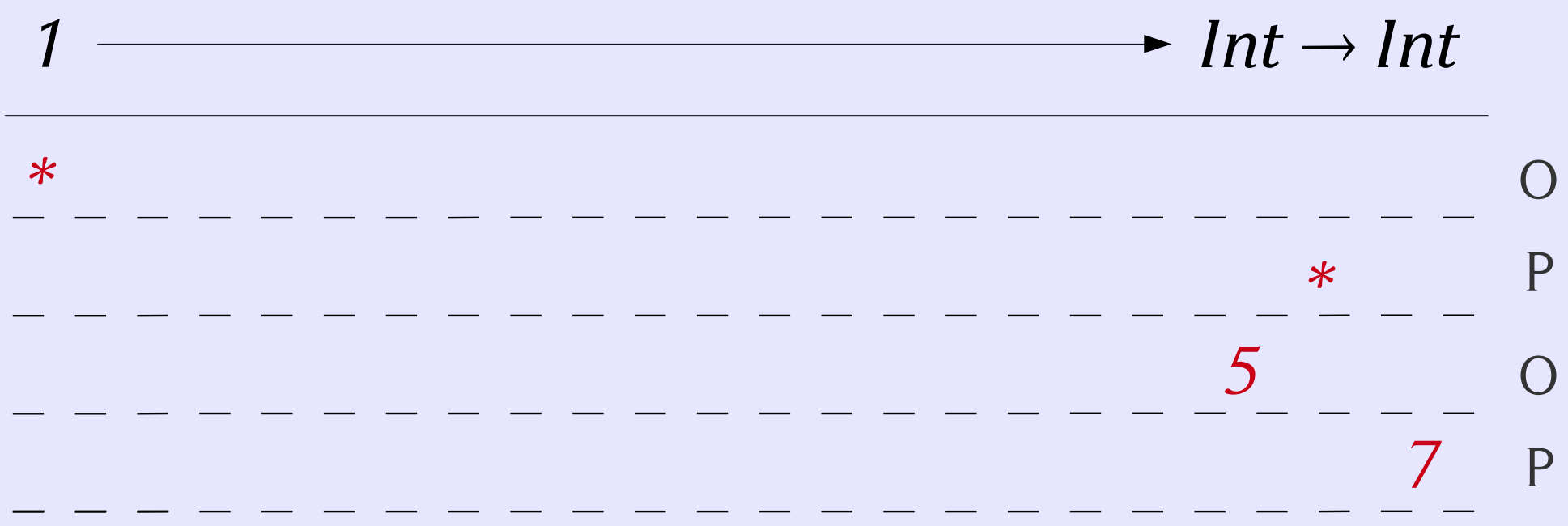



$\vdash \lambda x.x+1 : int \longrightarrow int$ ; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

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

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

# 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 ( ≠ Game Theory)
- Programs = *strategies* for Proponent
- Families (i.e. *categories*) of games

# From PCF to nominal games

*Full Abstraction for PCF* (early 90's)

- Two groups in the UK, one in Germany
- Roots in Mathematical Logic

# From PCF to nominal games

*Full Abstraction for PCF* (early 90's)

- Two groups in the UK, one in Germany
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

# From PCF to nominal games

*Full Abstraction for PCF* (early 90's)

- Two groups in the UK, one in Germany
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} realism?

# From PCF to nominal games

*Full Abstraction for PCF* (early 90's)

- Two groups in the UK, one in Germany
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} realism?

Nominal game semantics (2004-)

# Nominal sets

Think of a universe of *nominal sets:*

- there is a countably infinite set $\mathcal{N}$ of names
- all sets involve a finite amount of names

# Nominal sets

Think of a universe of *nominal sets:*

- there is a countably infinite set $\mathcal{N}$ of names
- all sets involve a finite amount of names

Formally, a nominal set $X$ consists of:

- a carrier set $X$
- an action $\bullet : \mathrm{PERM}(\mathcal{N}) \times X \longrightarrow X$
  - e.g. $(n_1\ n_2) \bullet n_1\, n_1\, n_2\, n_1 = n_2\, n_2\, n_1\, n_2$
- all elements of $X$ have finite support

# Nominal sets

Foundation for maths with names and name-binding

Think of a universe of *nominal sets:*

- there is a countably infinite set $\mathcal{N}$ of names
- all sets involve a finite amount of names

Formally, a nominal set $X$ consists of:

- a carrier set $X$
- an action $\bullet : \mathrm{PERM}(\mathcal{N}) \times X \longrightarrow X$
  - e.g. $(n_1\ n_2) \bullet n_1\, n_1\, n_2\, n_1 = n_2\, n_2\, n_1\, n_2$
- all elements of $X$ have finite support

# Nominal games

# Nominal games

$\lambda x.\mathtt{ref}(0) : \mathtt{com} \to \mathtt{intref}$

```
let f = [_] in { f() == f() }
```

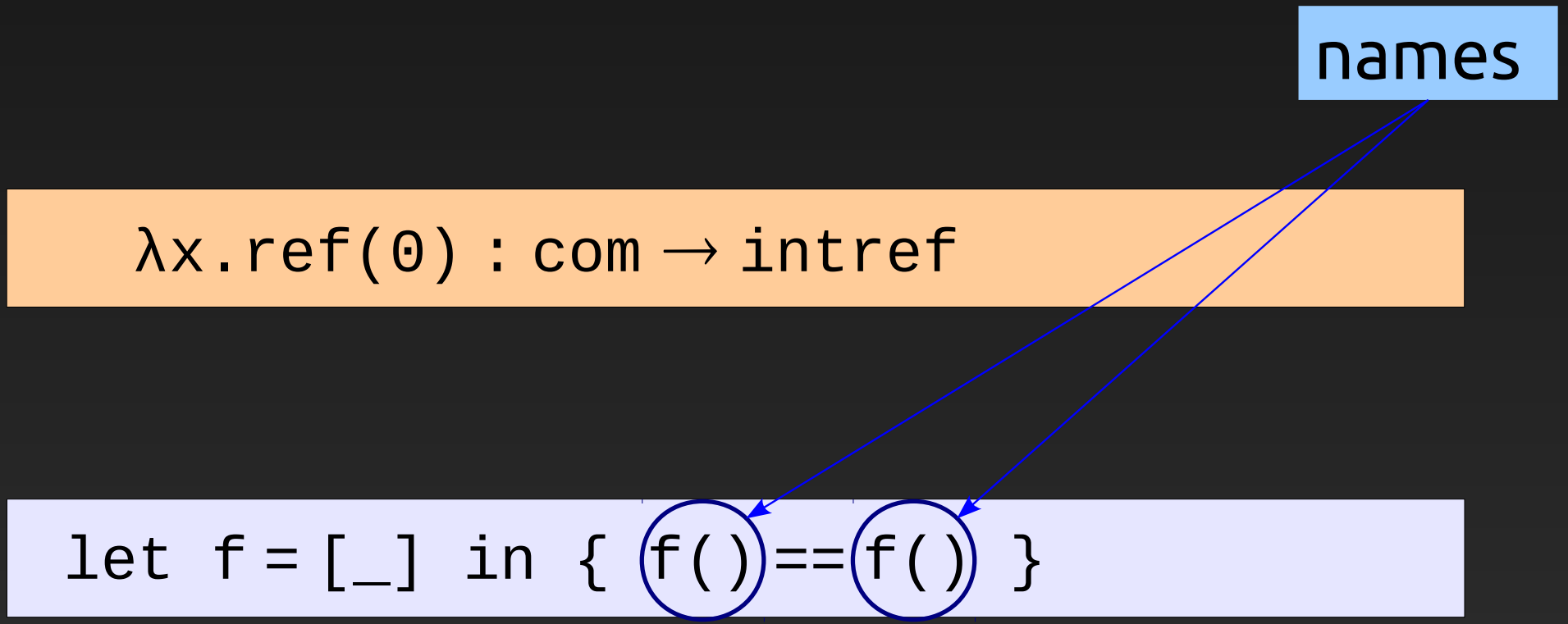
# Nominal games

$\lambda x.\mathtt{ref}(0) : \mathtt{com} \to \mathtt{intref}$

```
let f = [_] in { f() == f() }
```

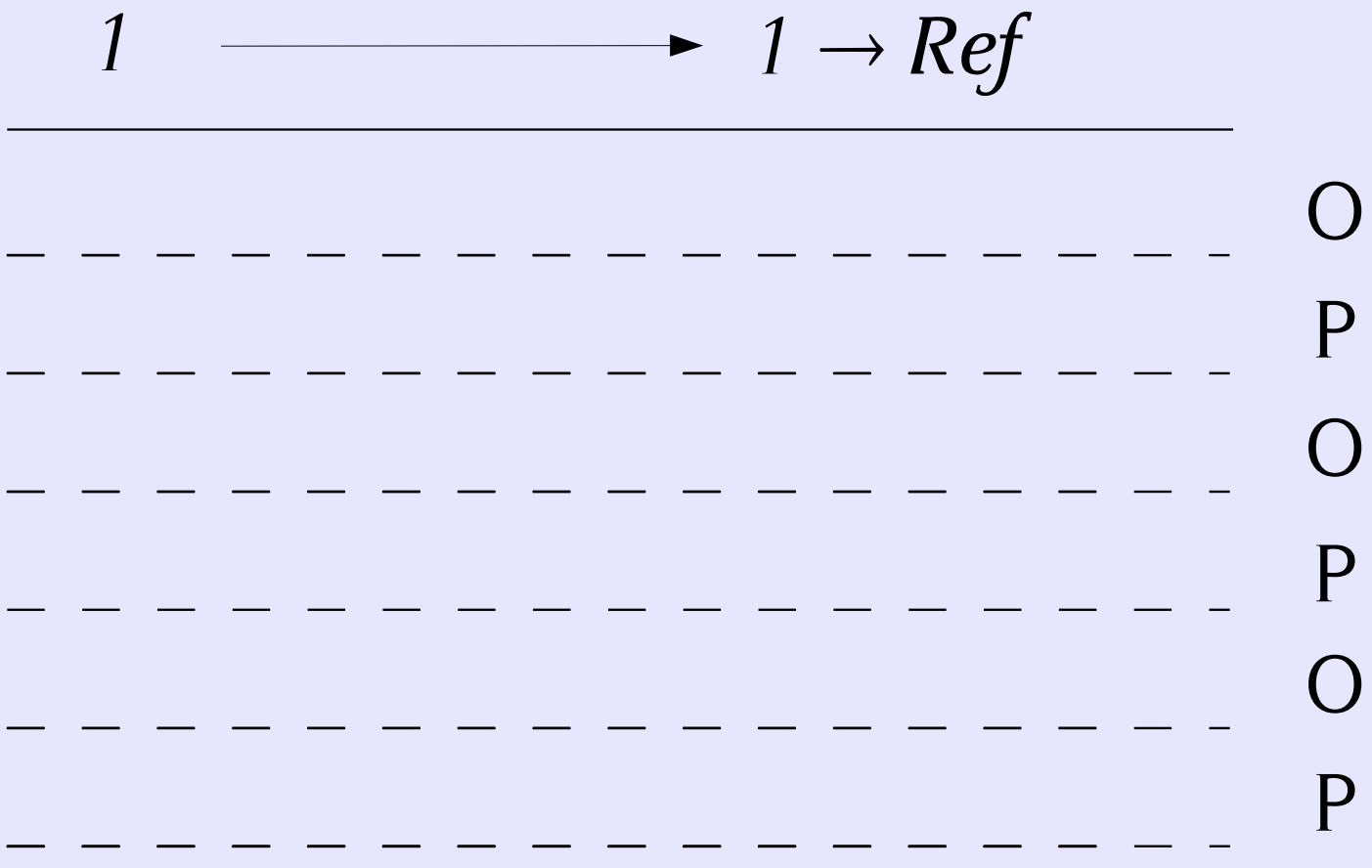
# Nominal games

## Games in nominal sets



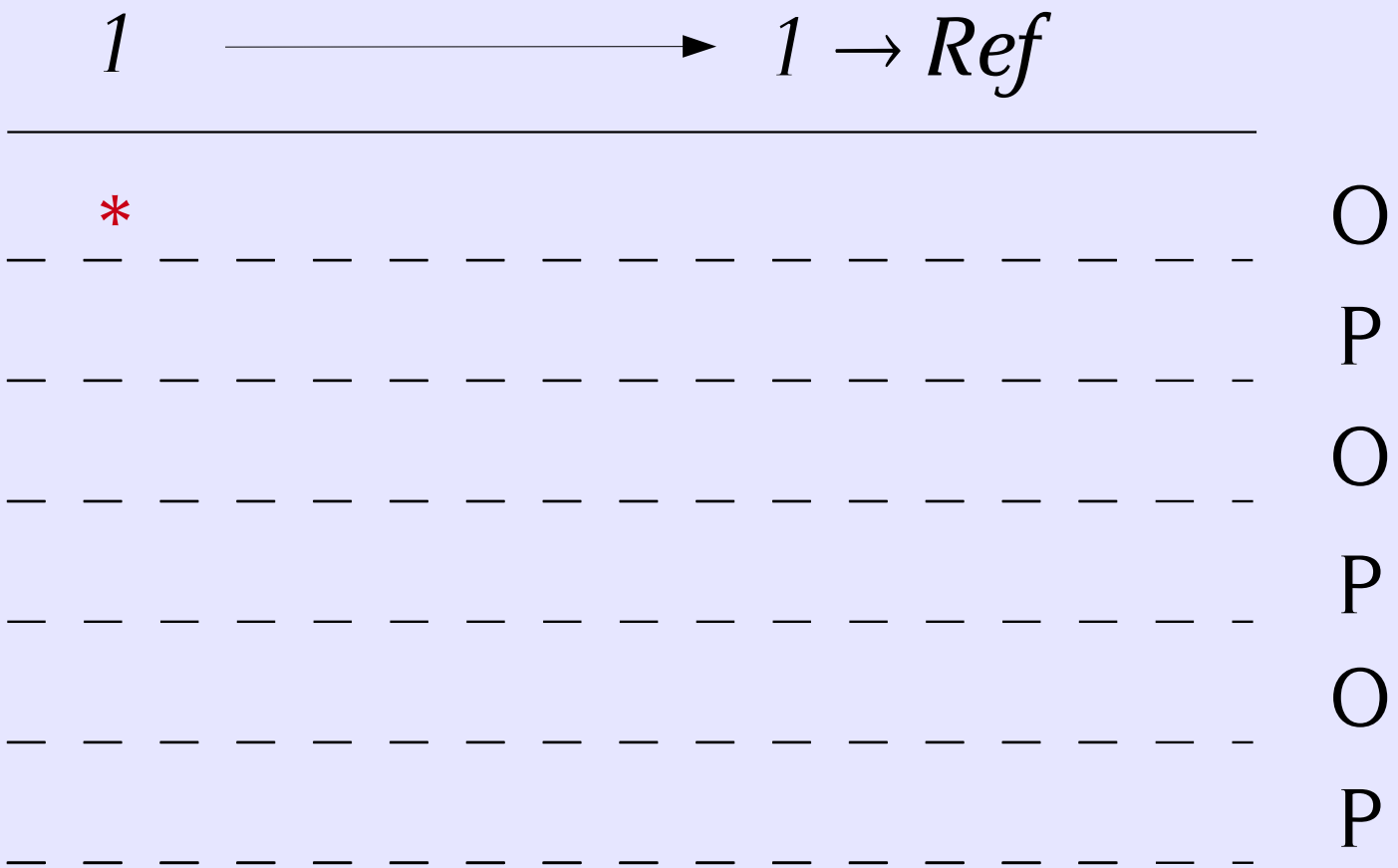
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



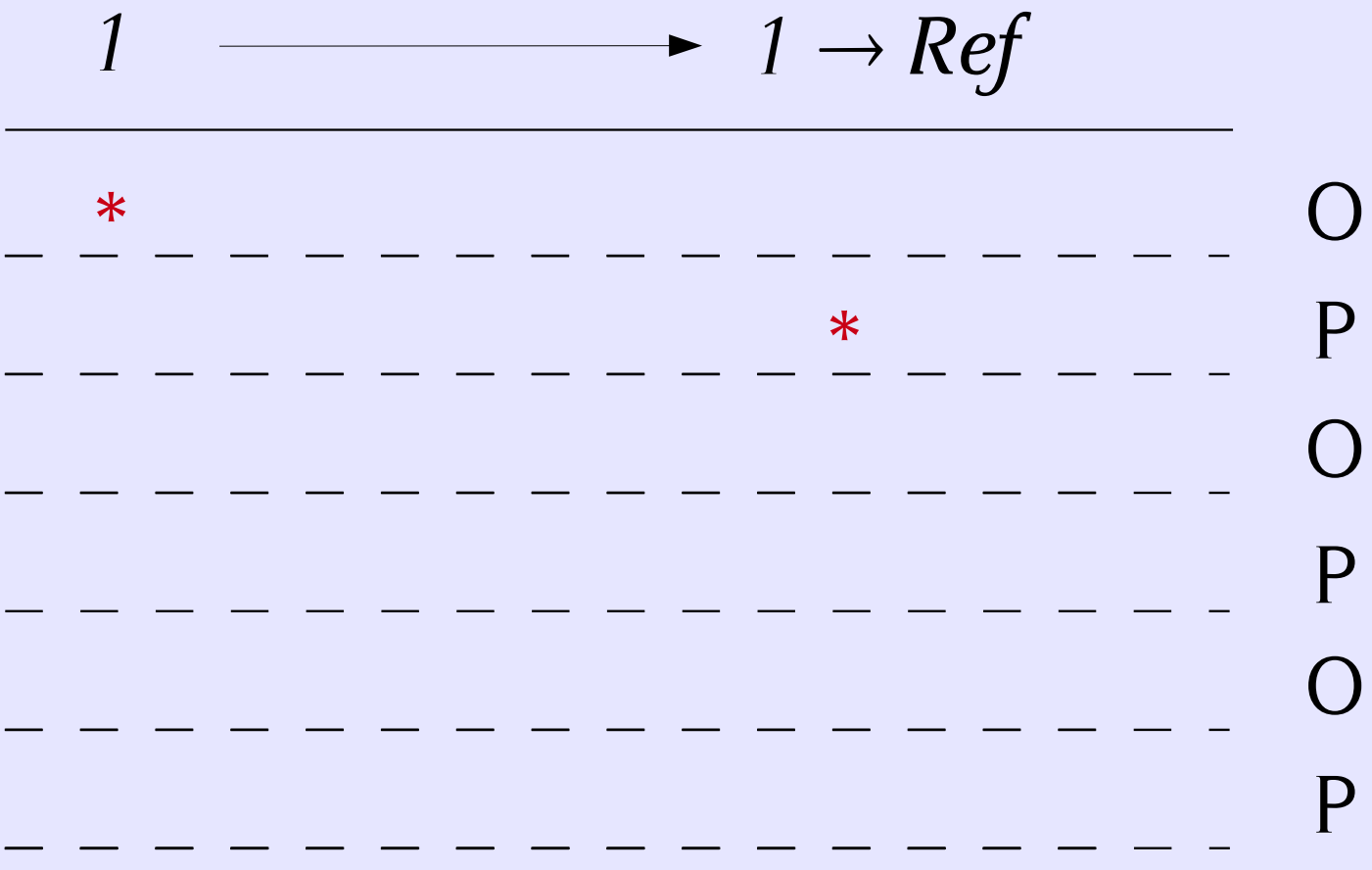
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



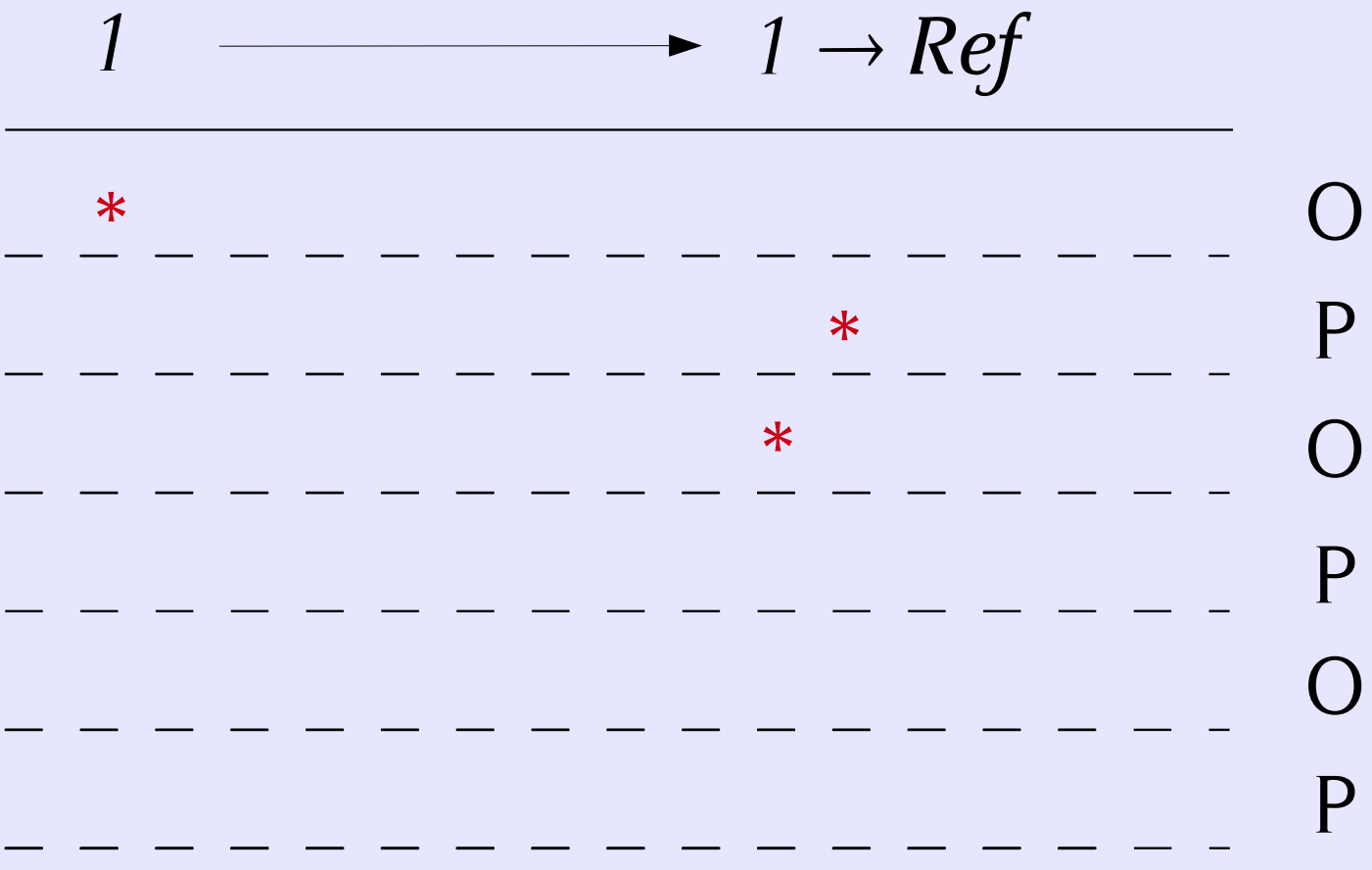
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



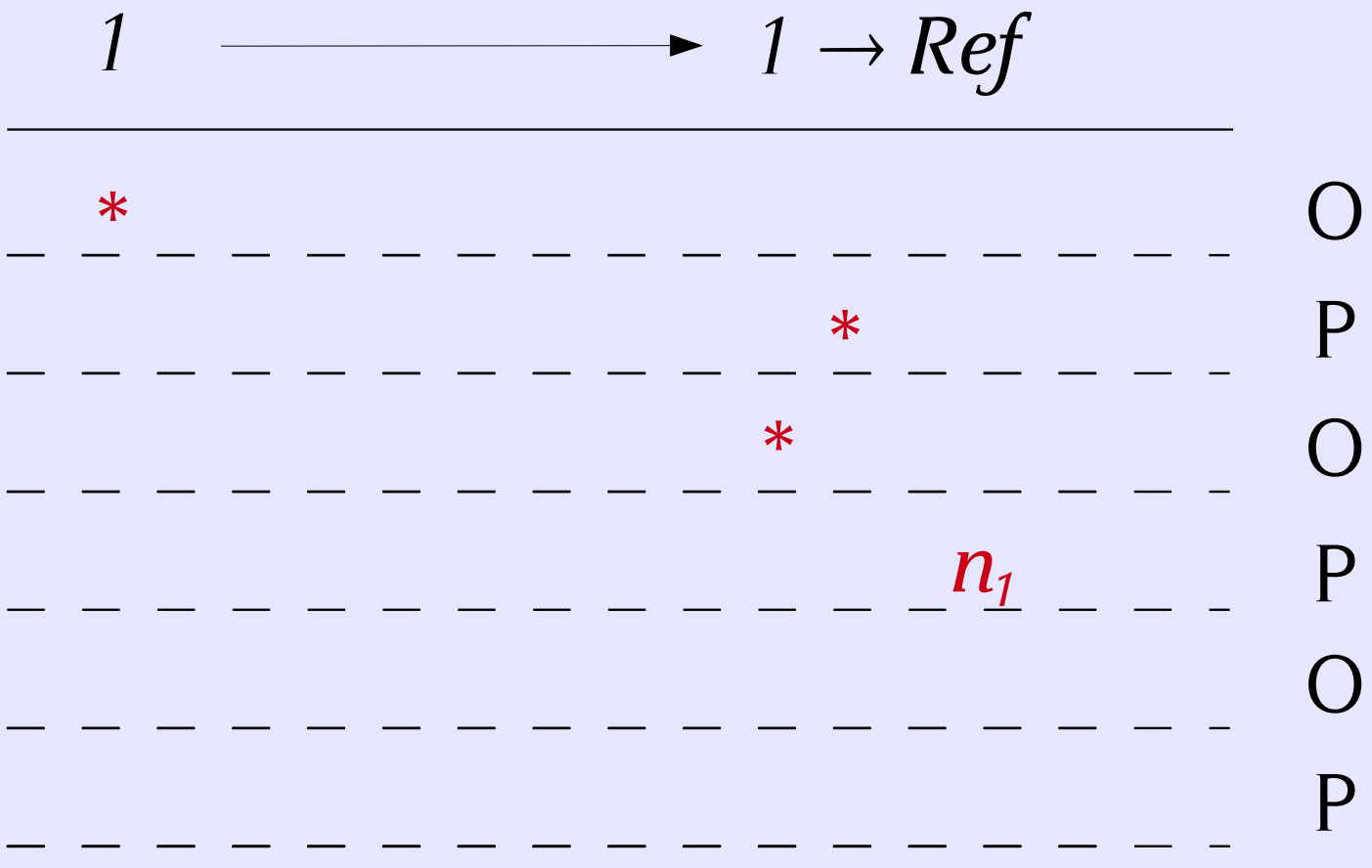
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



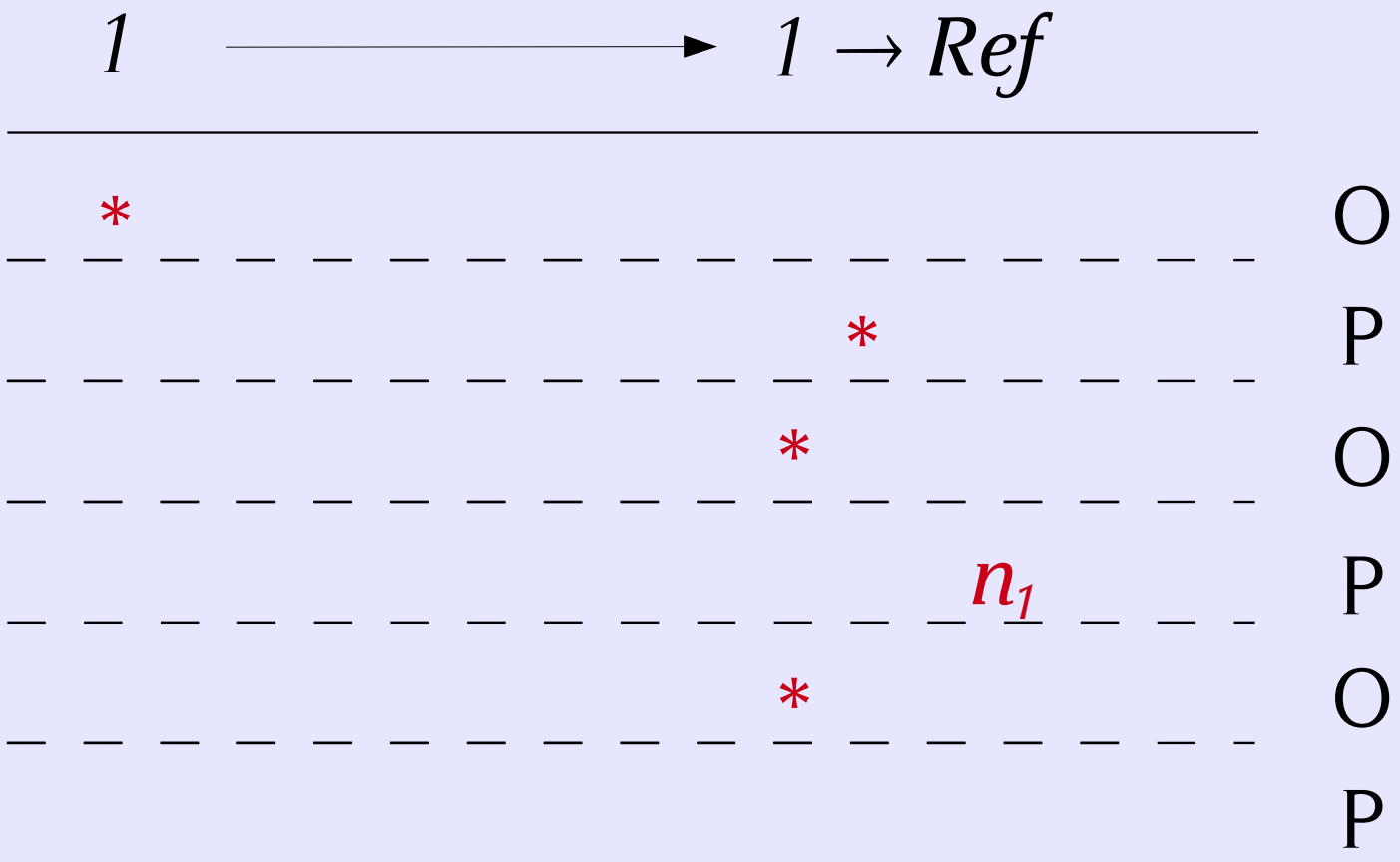
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



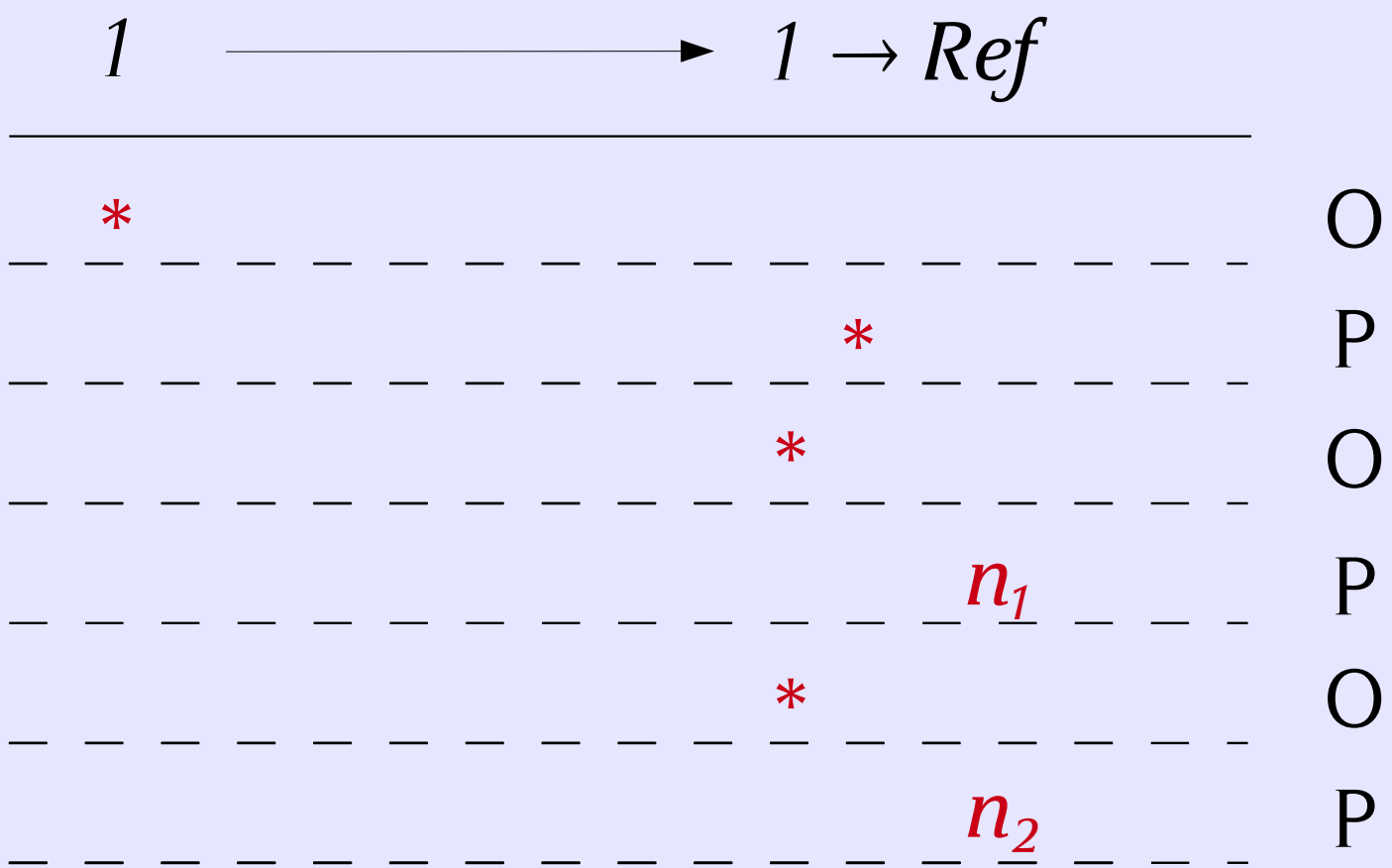
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



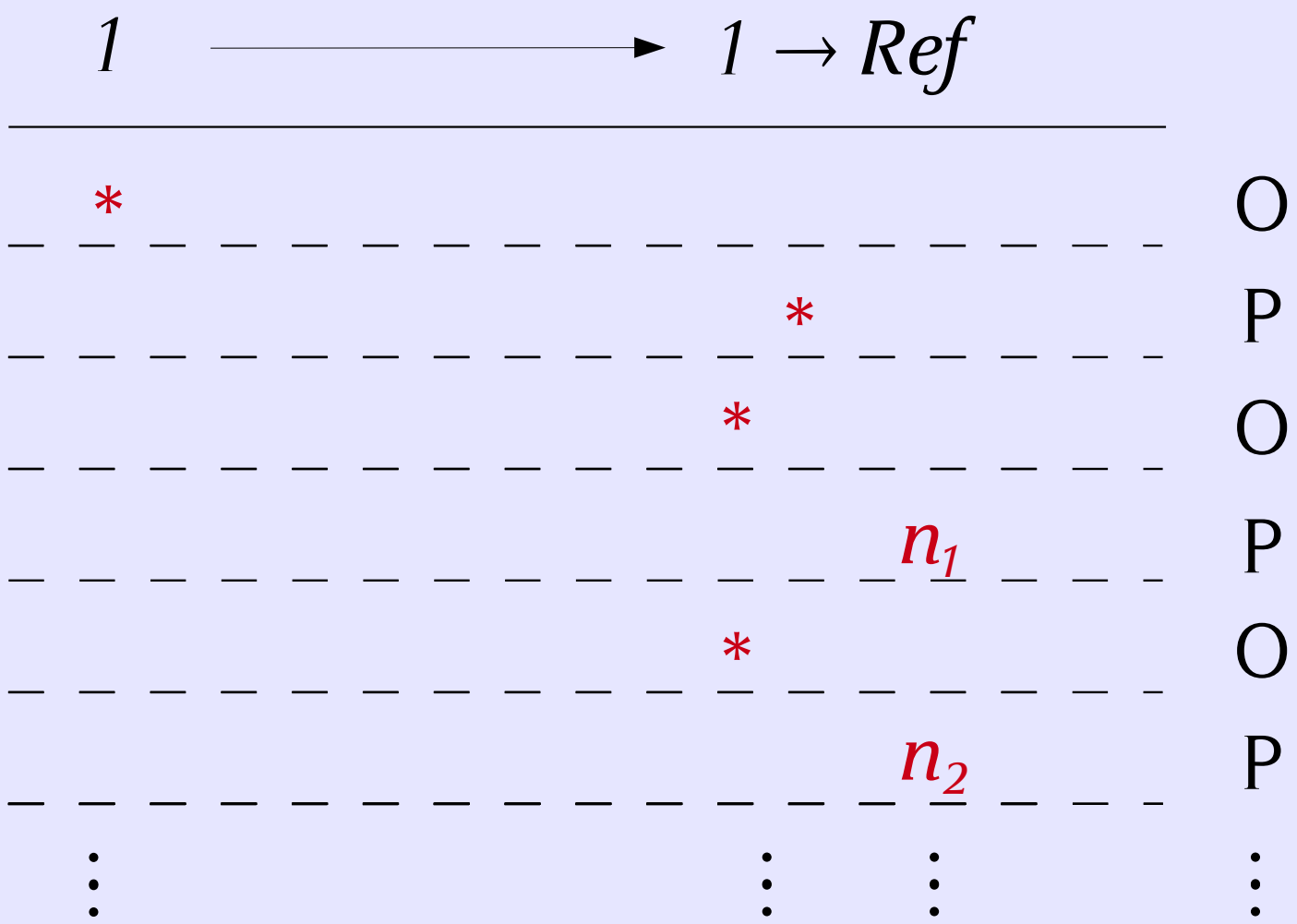
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



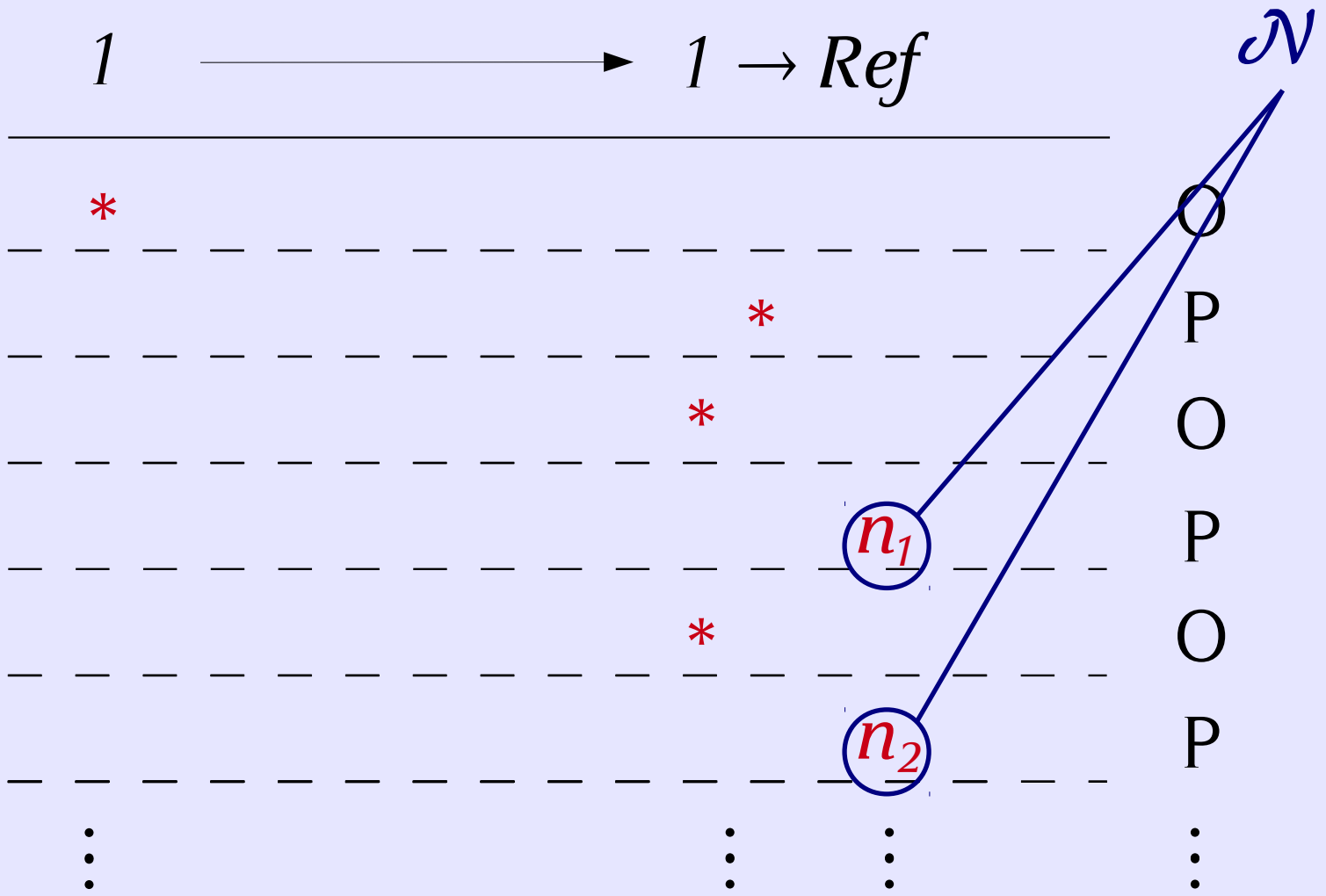
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



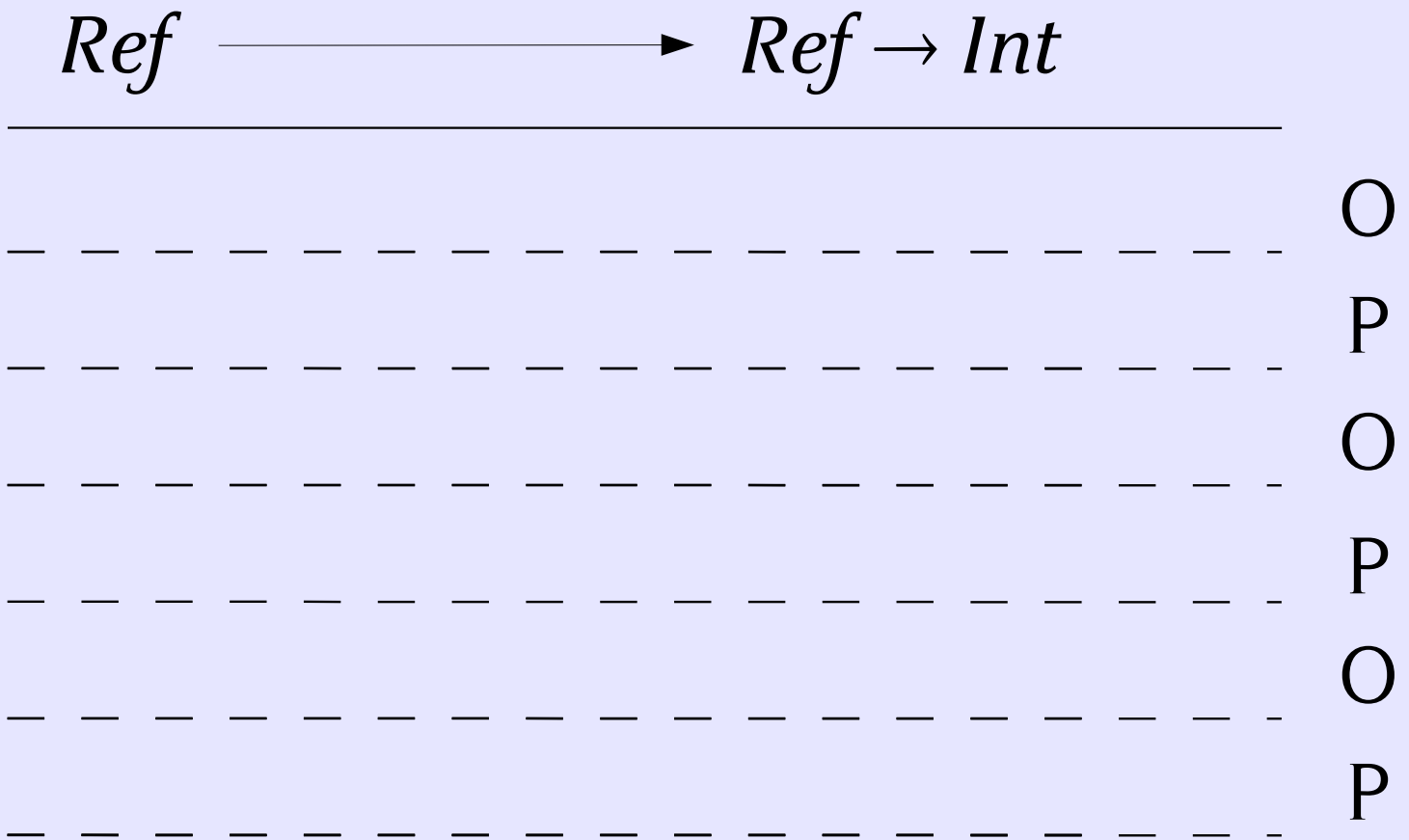
# Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



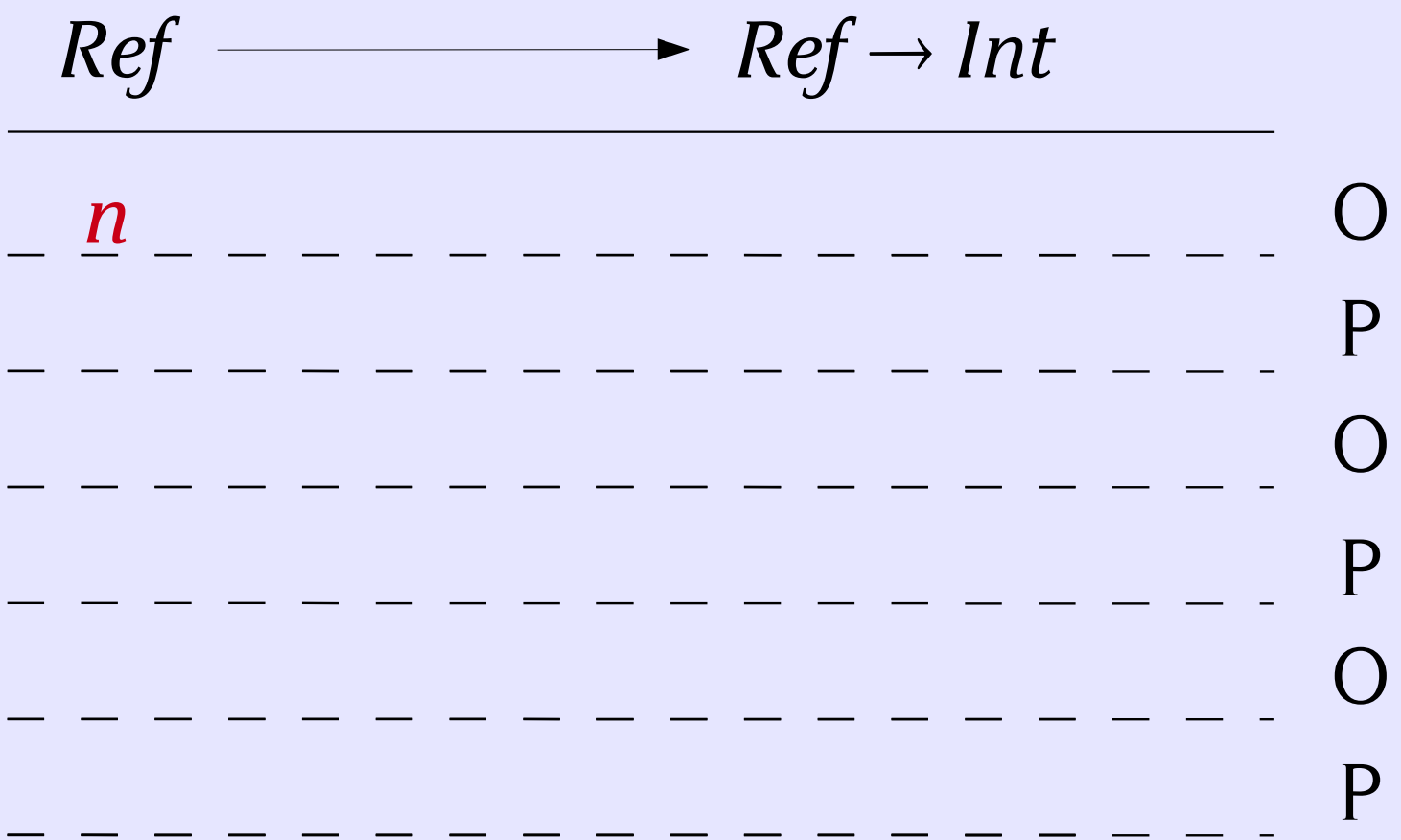
# Examples

$\texttt{x : ref} \vdash \texttt{λy.(x == y) : ref} \rightarrow \texttt{int}$



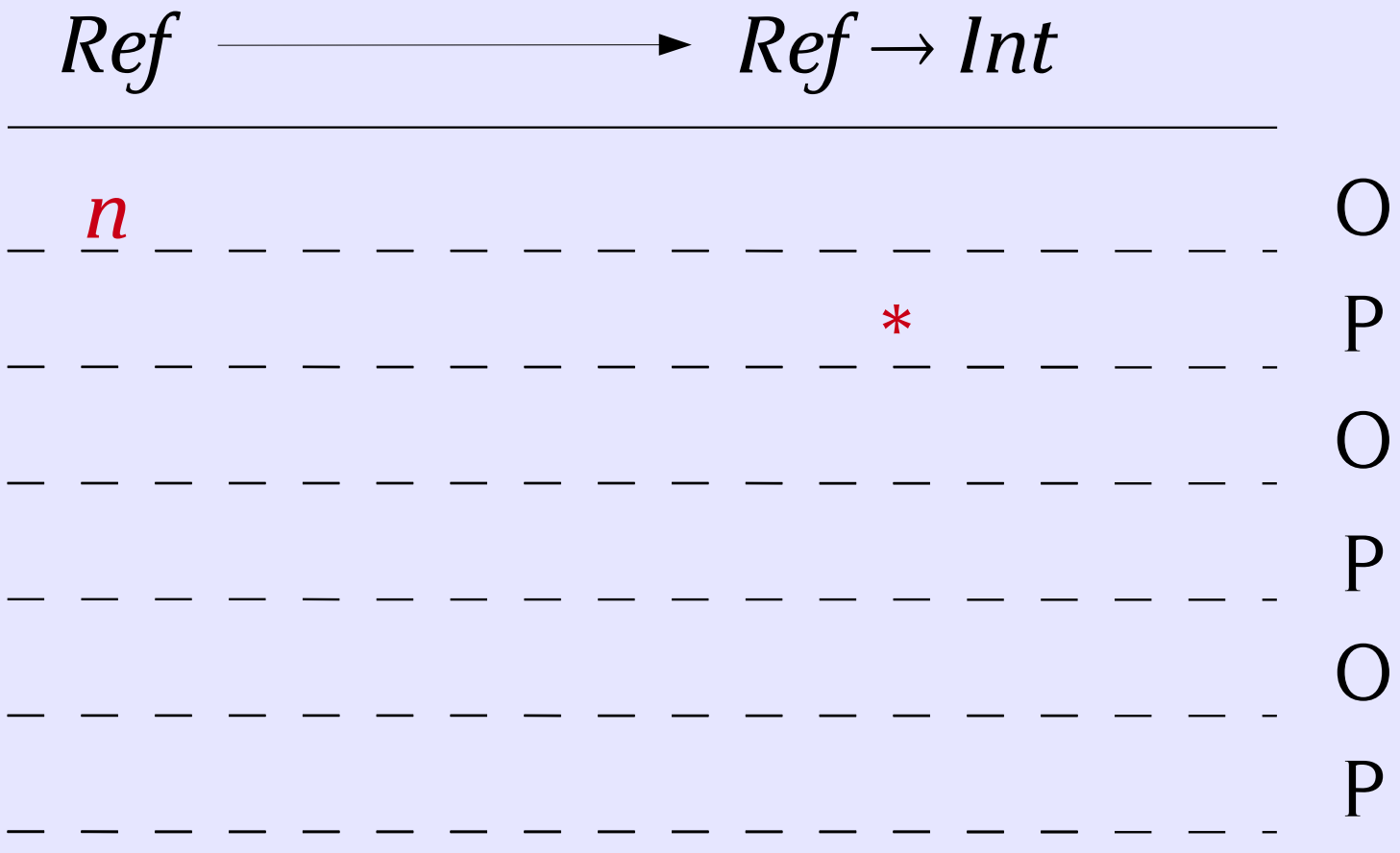
# Examples

$x : \texttt{ref} \vdash \lambda y.(x == y) : \texttt{ref} \to \texttt{int}$



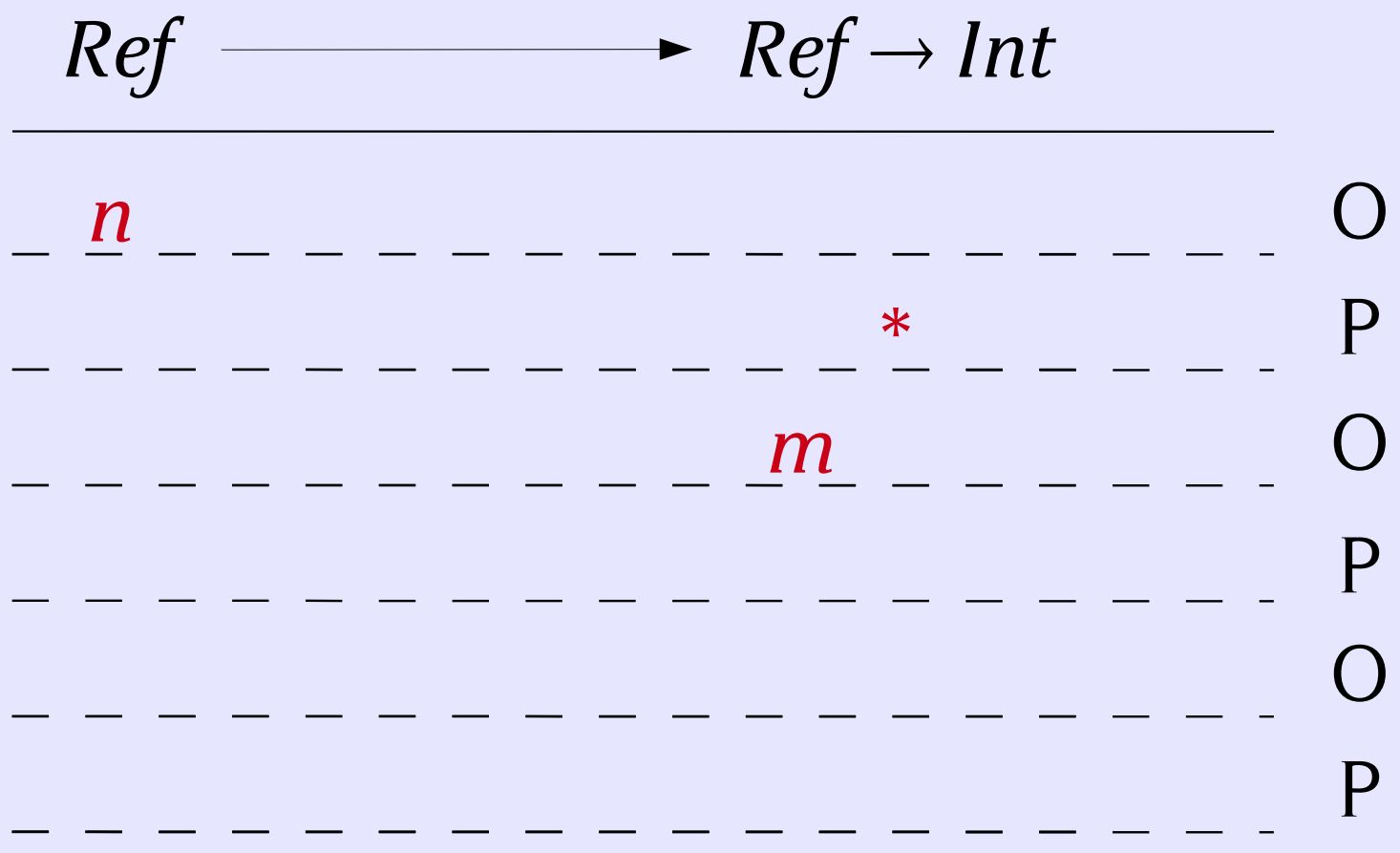
# Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$



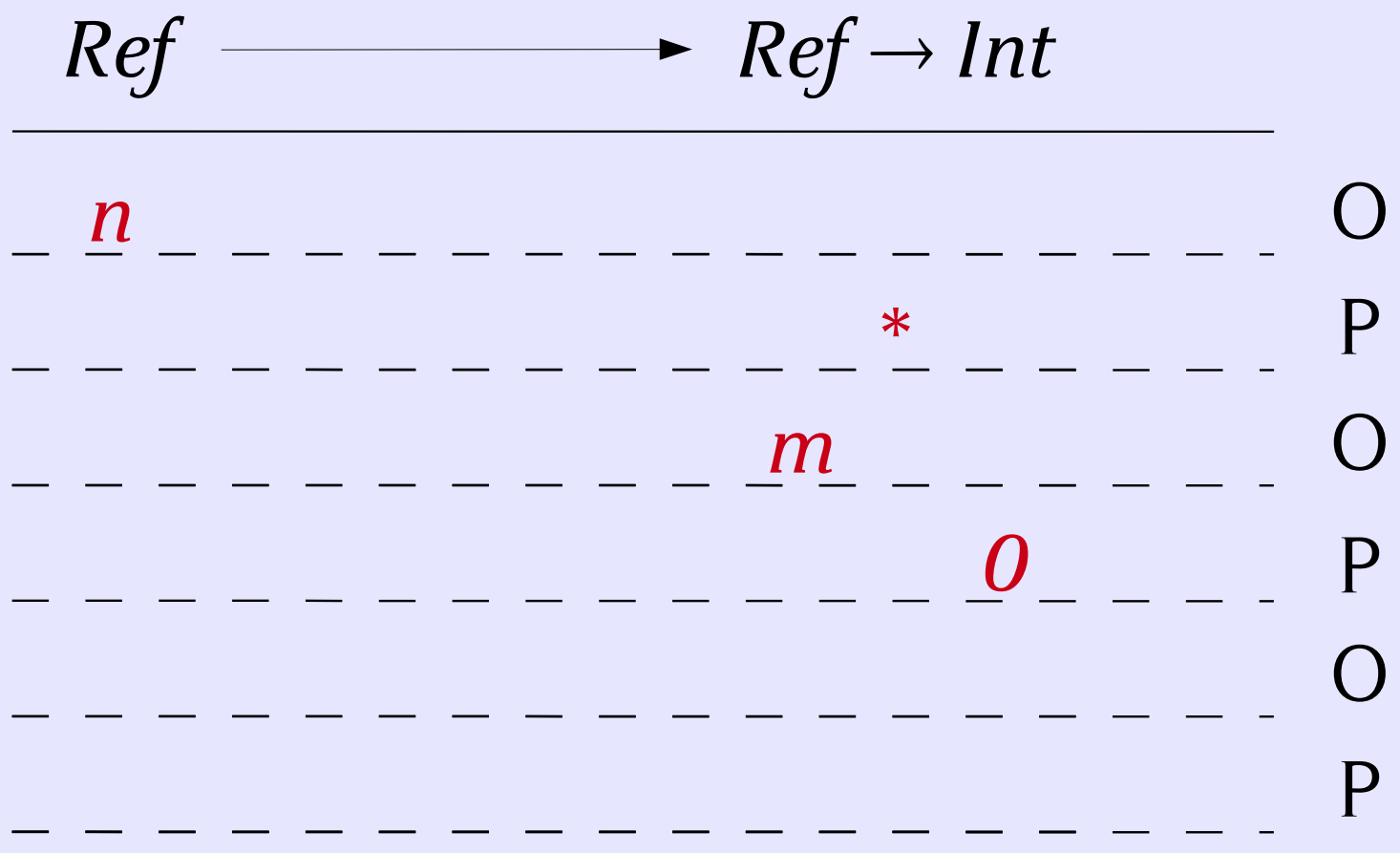
# Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$



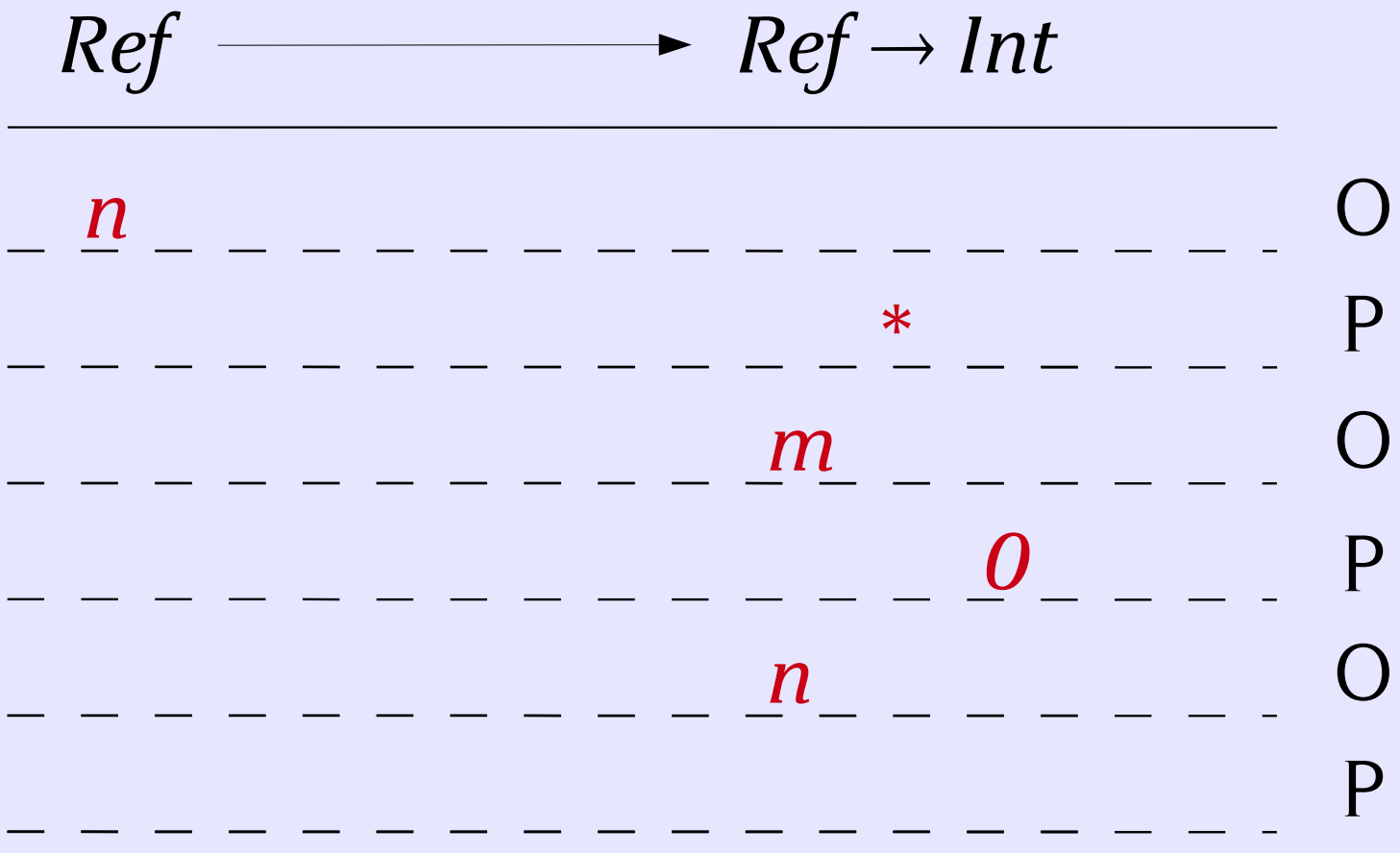
# Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$



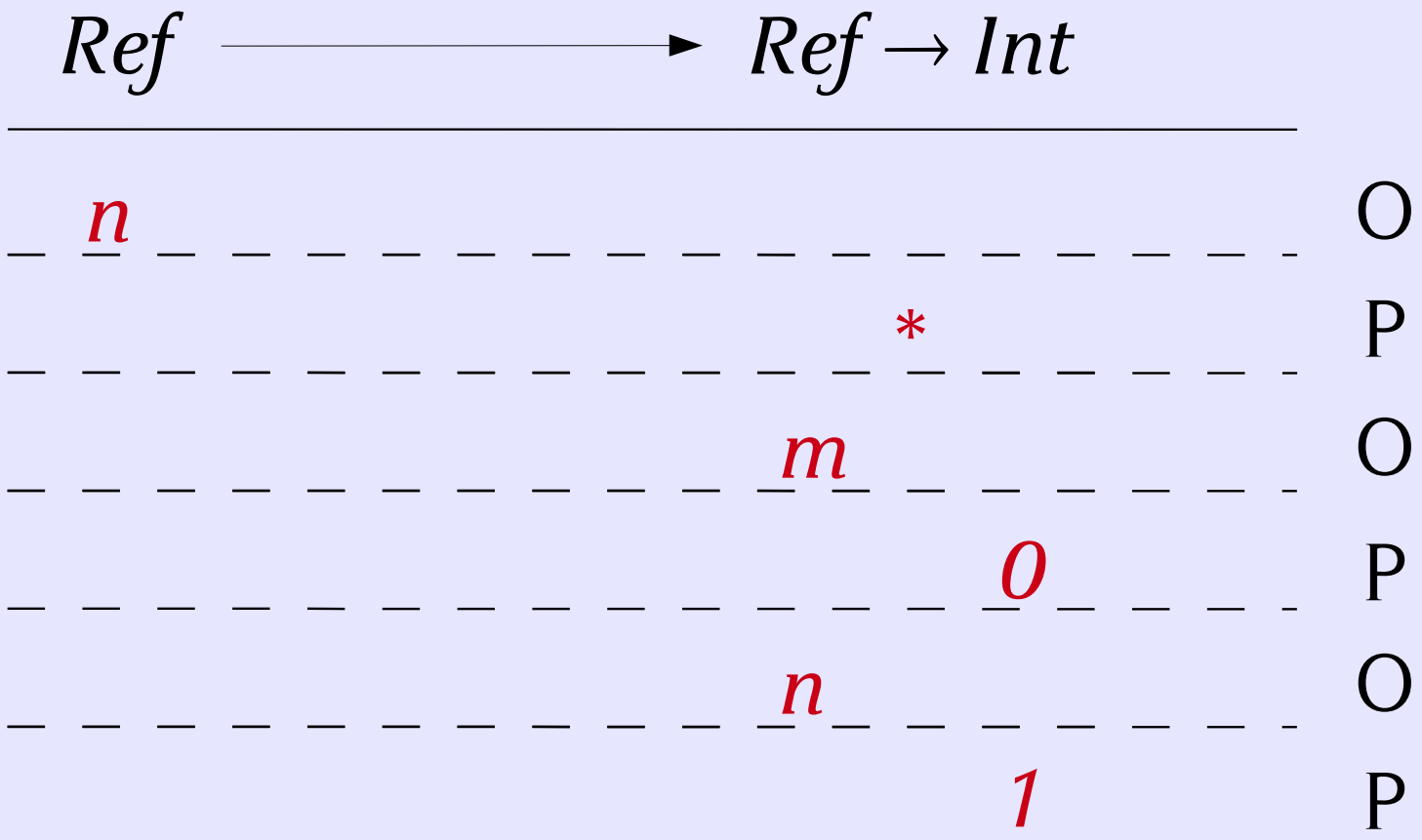
# Examples

$\texttt{x : ref} \vdash \lambda\texttt{y.(x == y) : ref} \rightarrow \texttt{int}$



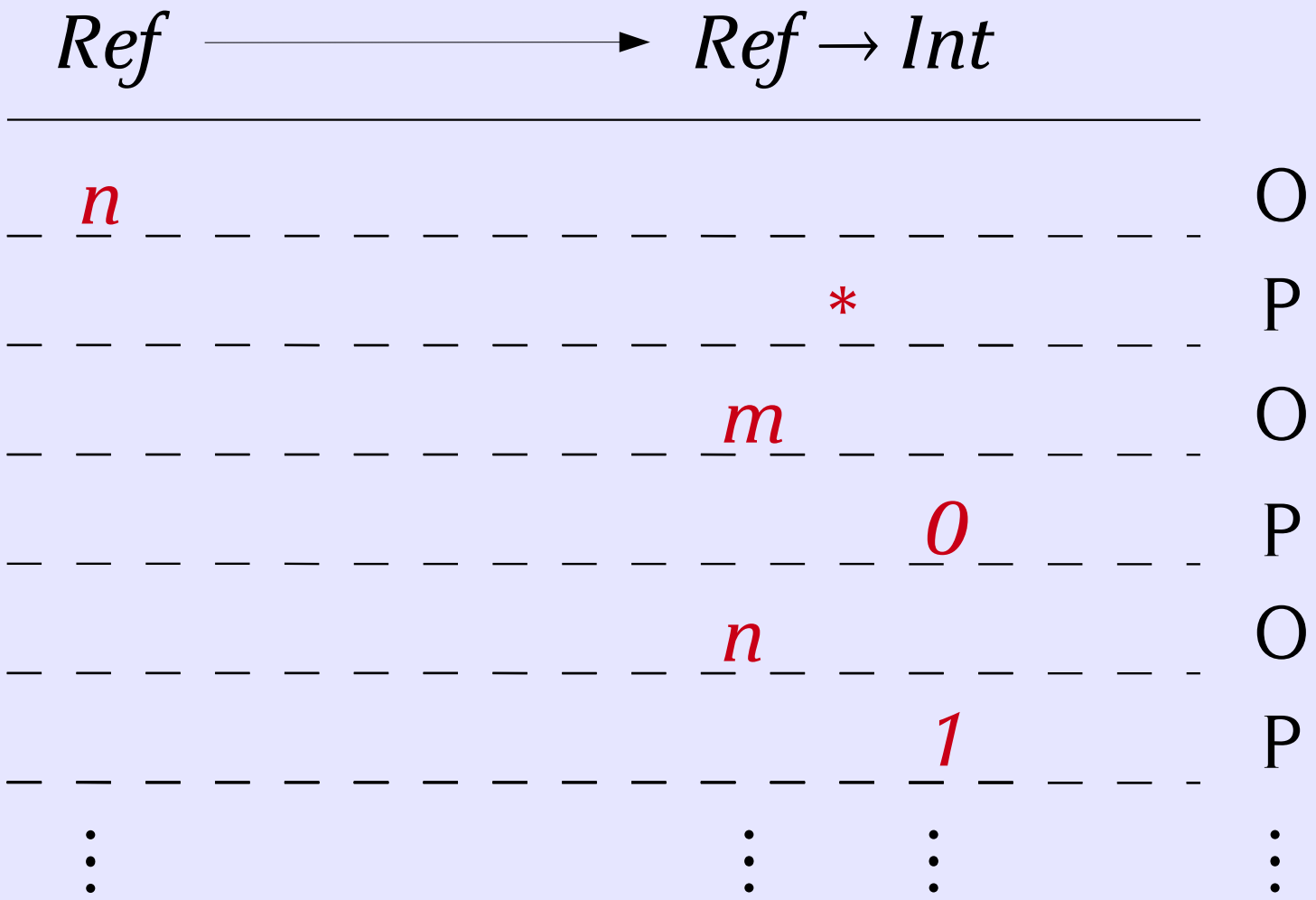
# Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$



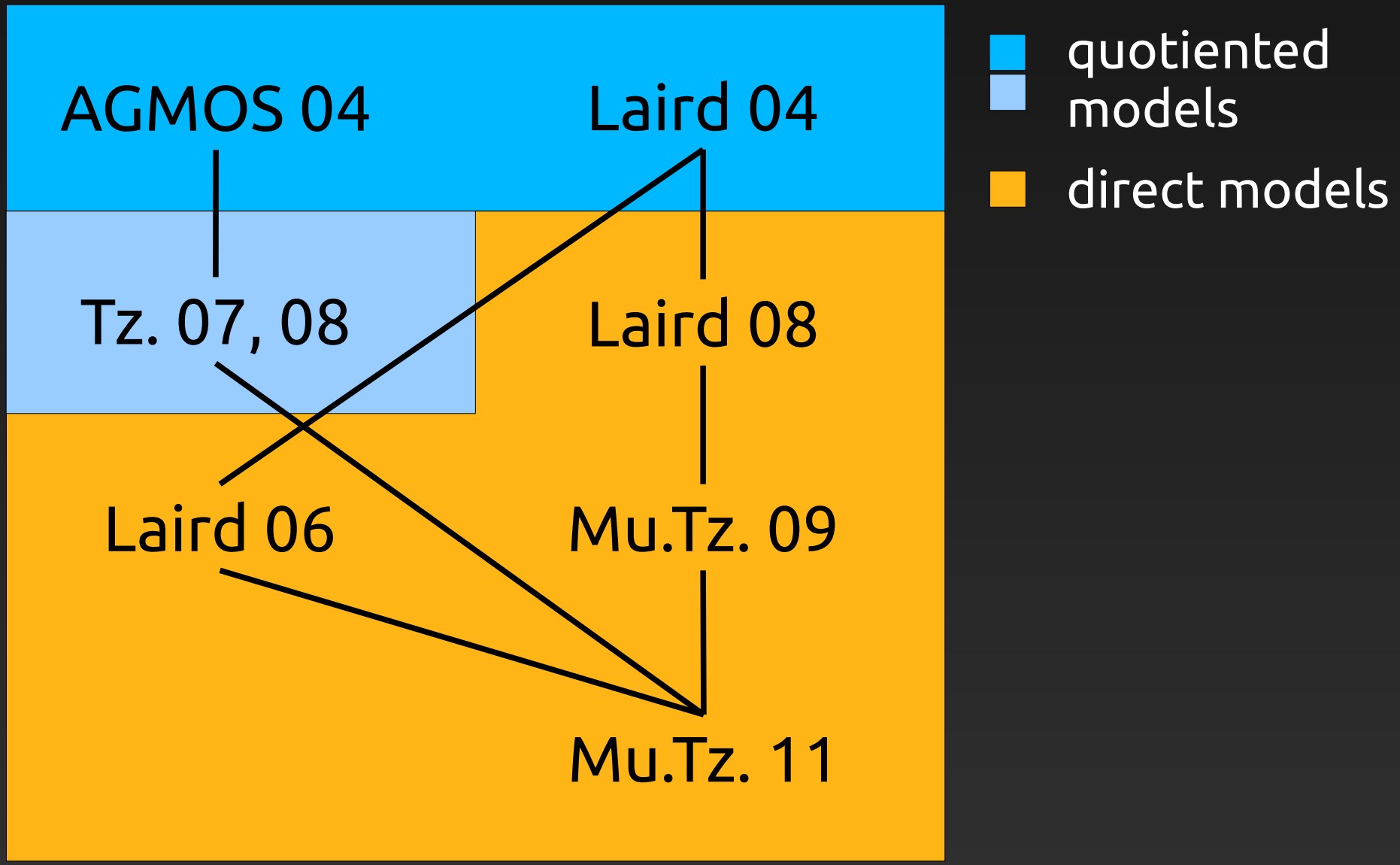
# Examples

$$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$$

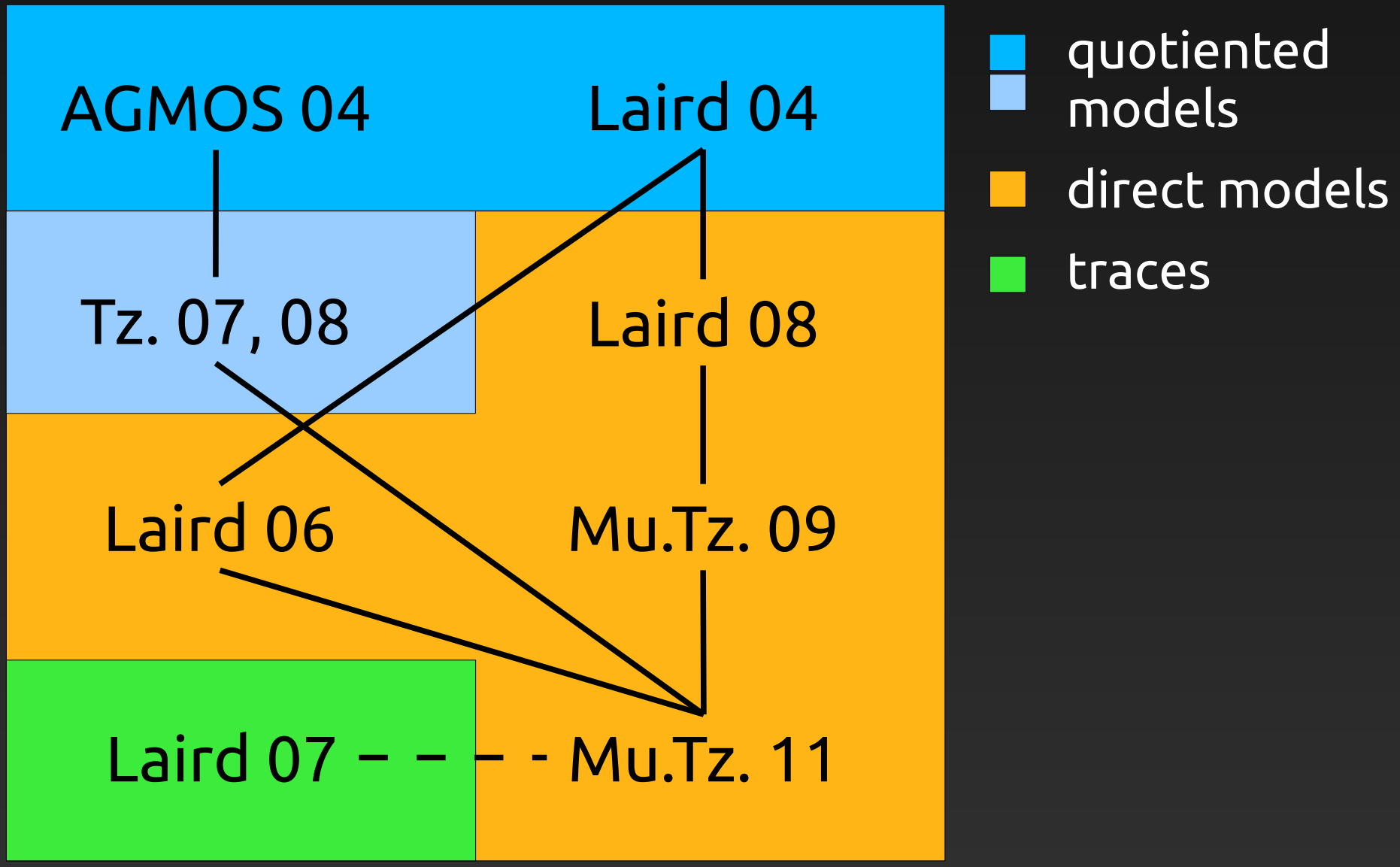


# Achievements

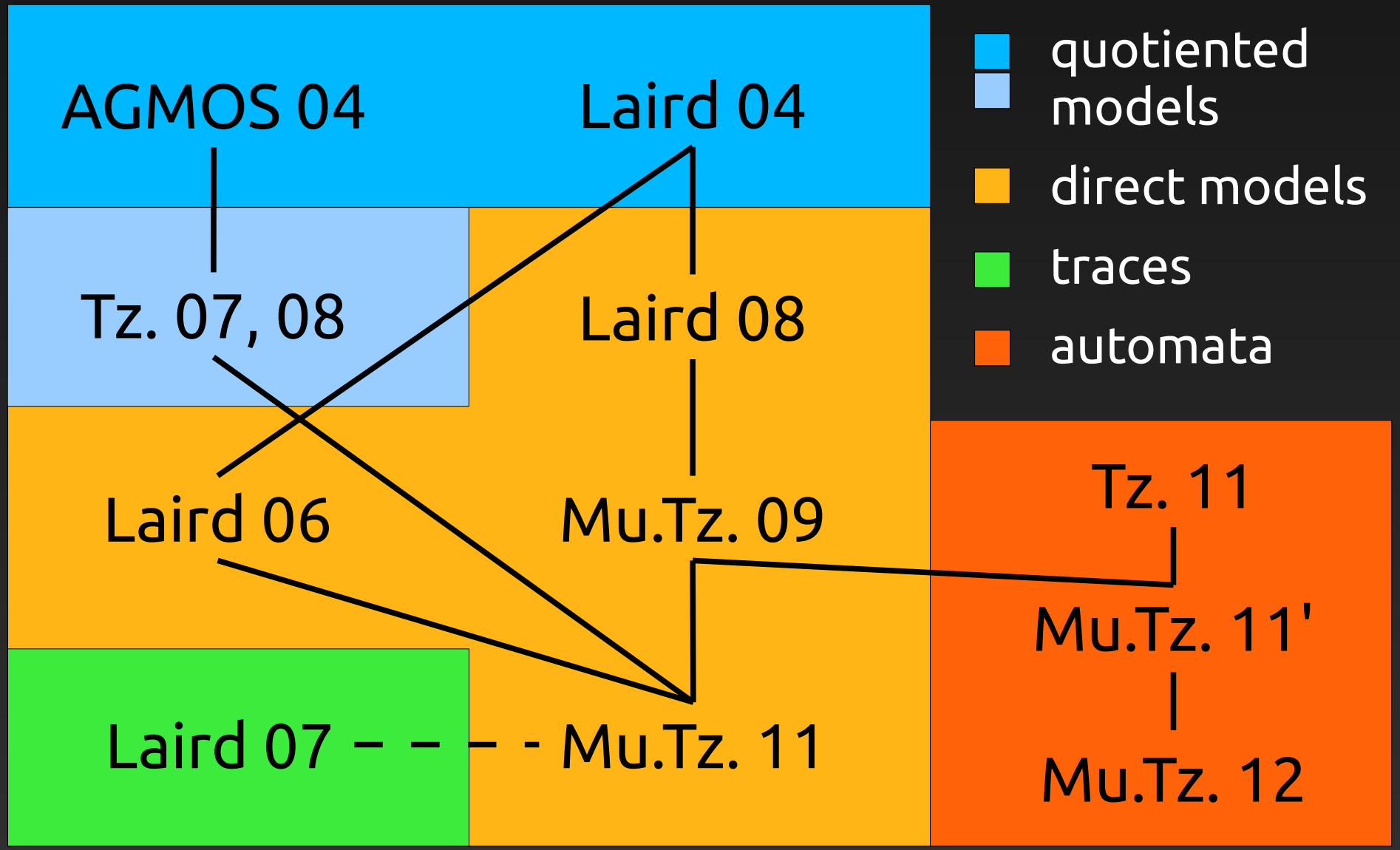
# Achievements



# Achievements



# Achievements



# Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: HO references, exceptions, ...

$$M \cong N \iff [\![M]\!] \cong [\![N]\!]$$

# Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: HO references, exceptions, ...

$$M \cong N \iff [\![M]\!] \cong [\![N]\!]$$

$$[\![A \to B]\!] = [\![A]\!] \Rightarrow (S \Rightarrow [\![B]\!] \otimes S)$$

$$S = \bigotimes_A (\mathbf{N}_A \Rightarrow [\![A]\!])$$

# Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: HO references, exceptions, ...

$$M \cong N \iff [\![M]\!] \cong [\![N]\!]$$

Characteristics:

- moves involving names
- moves-with-state (a set/list of names)
- “functional” behaviour + monads for effects

# Direct models

- Laird 06: higher-order channels
  Laird 08: pointers
- Mu.Tz. 09: integer references (Reduced ML)
  Mu.Tz. 11: HO references

$$M \cong N \iff \mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$$

# Direct models

- Laird 06: higher-order channels
  Laird 08: pointers
- Mu.Tz. 09: integer references (Reduced ML)
  Mu.Tz. 11: HO references

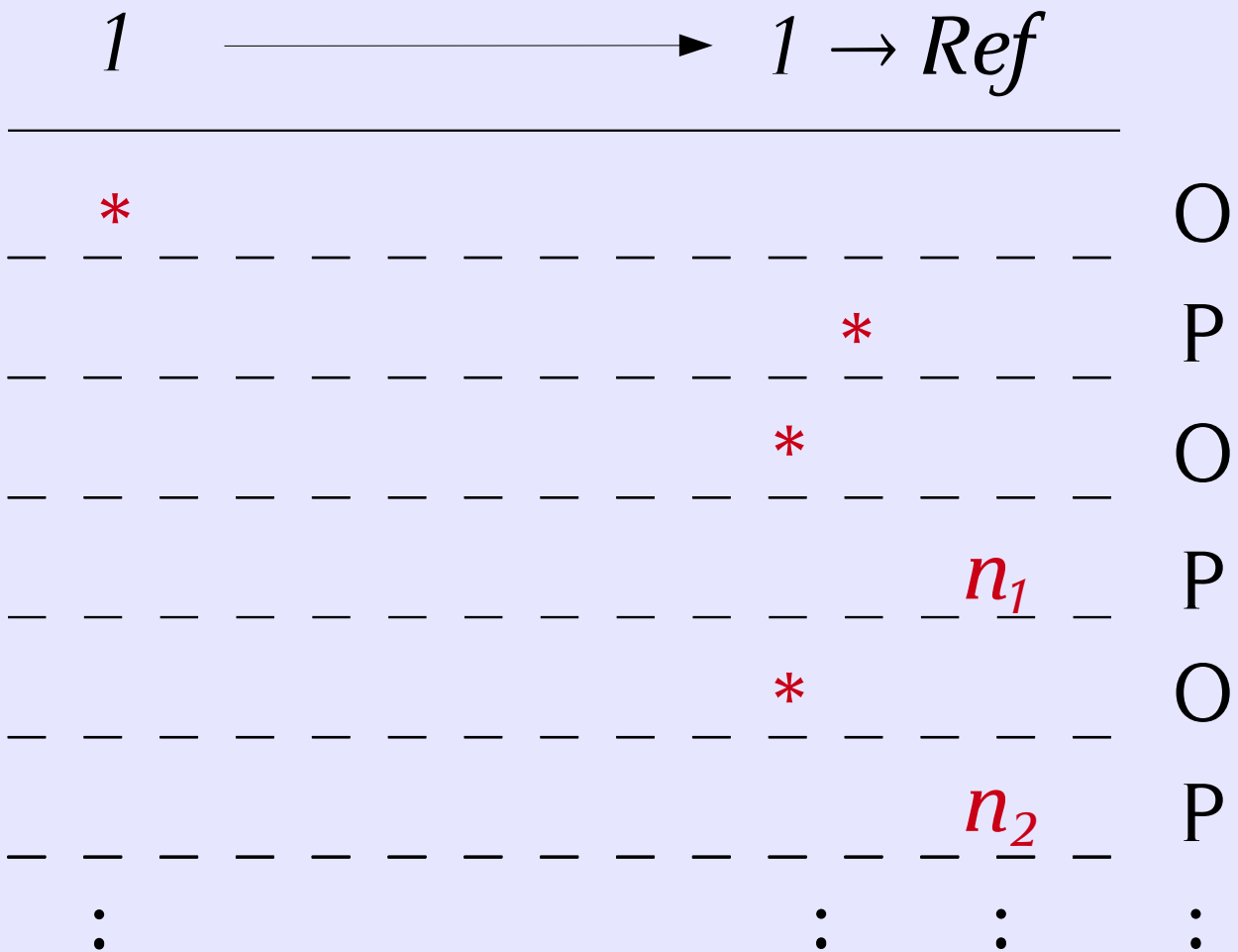
$$M \cong N \iff \mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$$

Characteristics:

- moves involving names/ moves-with-store
- name-availability conditions/ “direct” effects

# Algorithmics

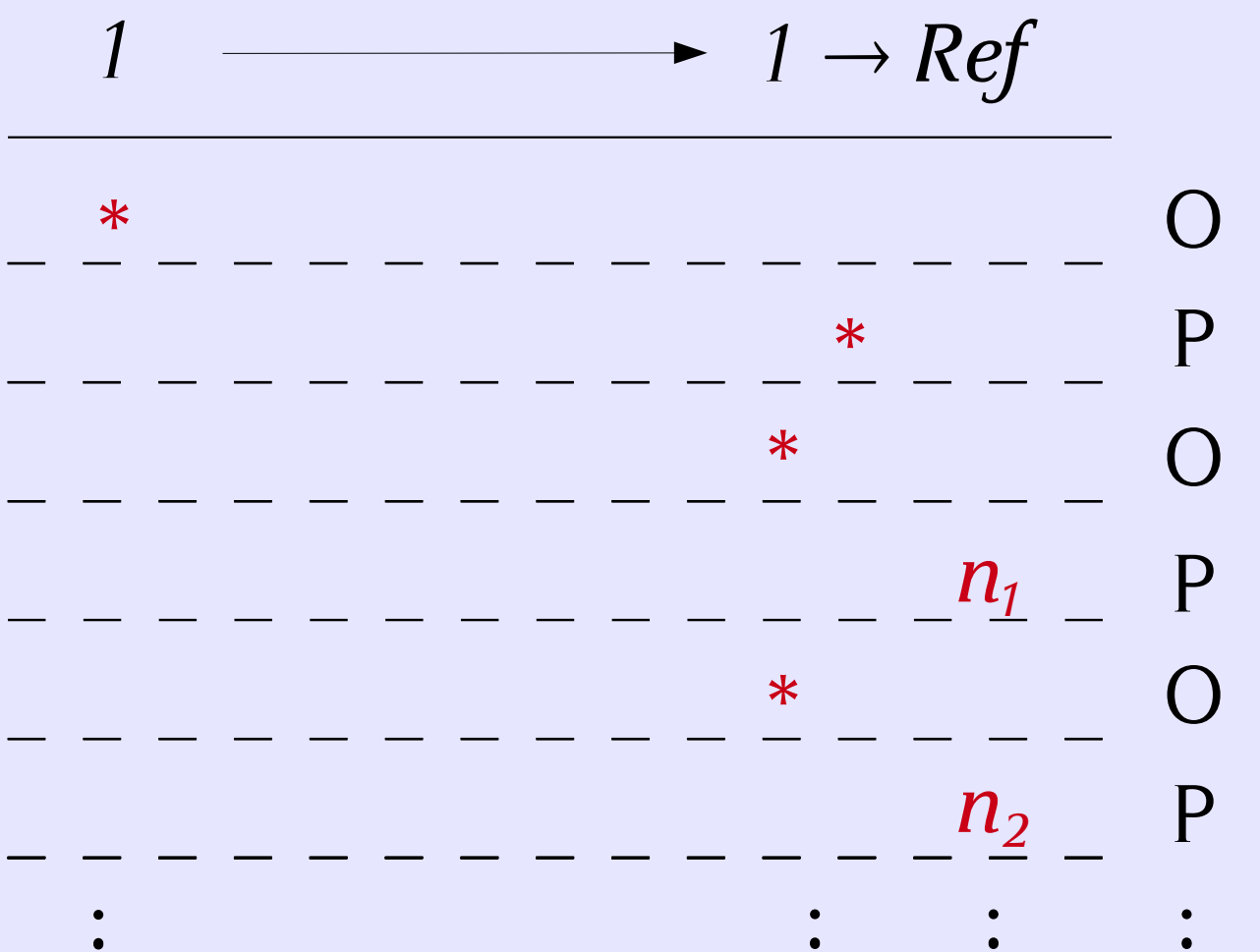
$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



# Algorithmics

$$M \cong N \iff \mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$$

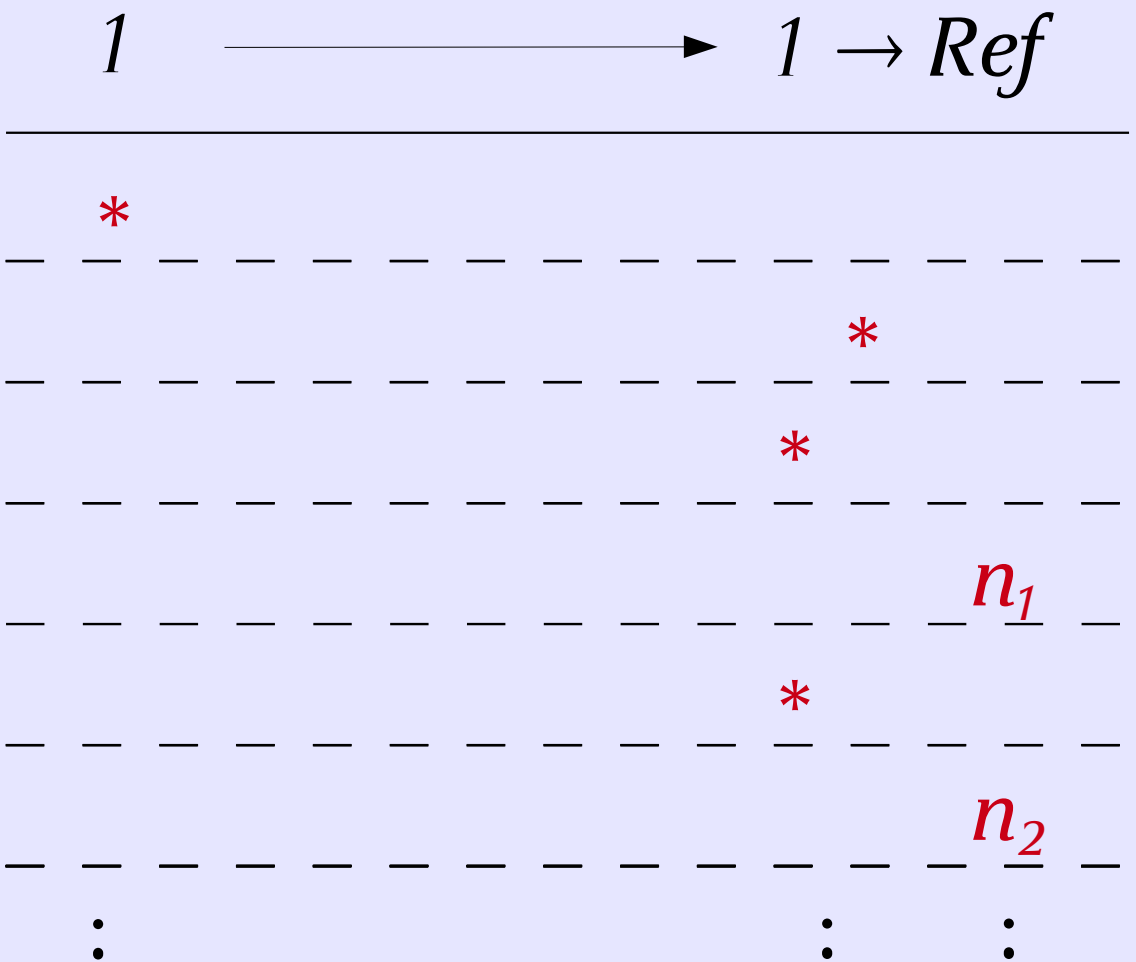
$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



# Algorithmics

$$M \cong N \iff \mathrm{comp}(\llbracket M \rrbracket) = \mathrm{comp}(\llbracket N \rrbracket)$$

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$




Programs

Game Semantics

Automata

# Fresh-register automata

$\texttt{λz.ref()} \longmapsto \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$

# Fresh-register automata

$\lambda z.\texttt{ref}() \longmapsto \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$

Automata with names

- Infinite alphabet $\mathcal{N}$
- Freshness recognition

# Fresh-register automata

$\lambda z.\texttt{ref}() \longmapsto \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$

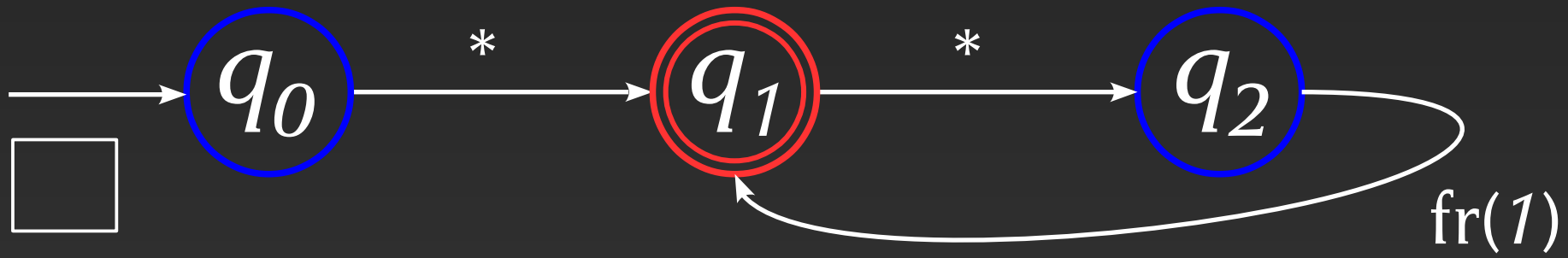
Automata with names

- Infinite alphabet $\mathcal{N}$
- Freshness recognition

Finite-state machines with registers

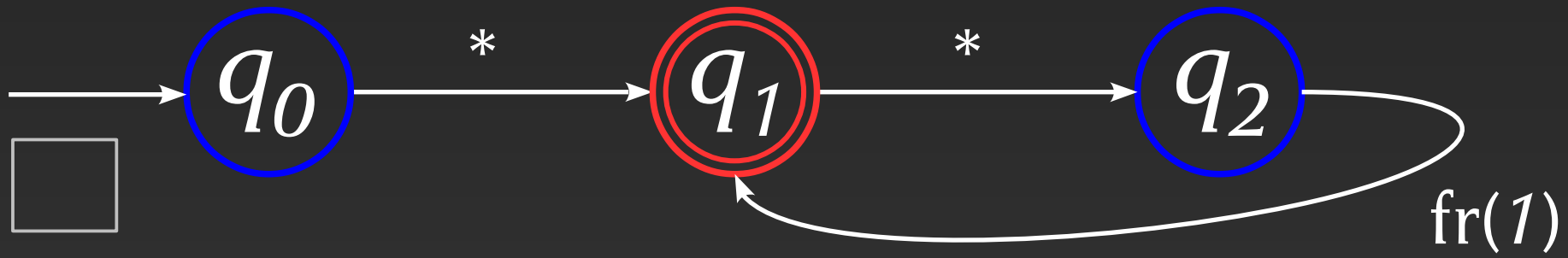
- *Kaminski & Francez (1994)*

# Fresh-register automata

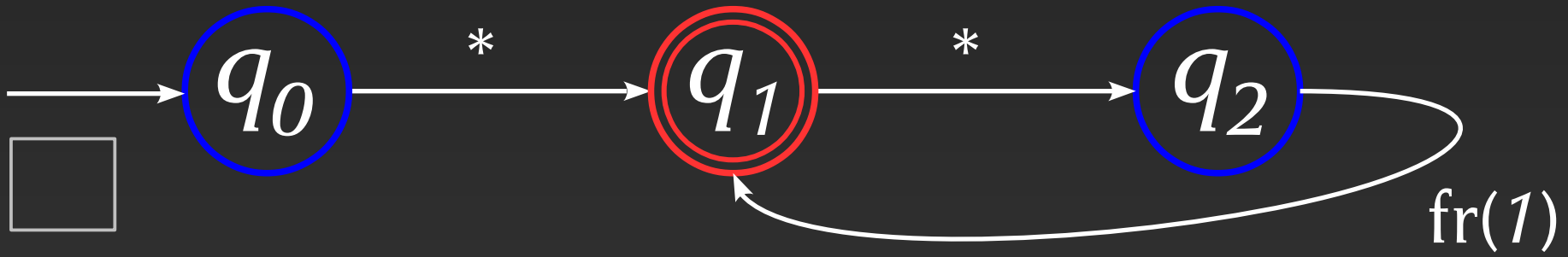


# Fresh-register automata

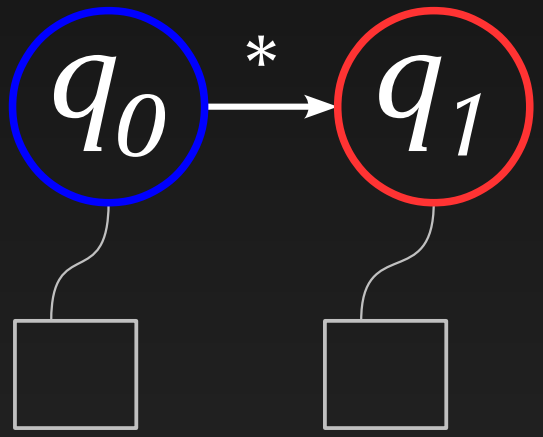
# Fresh-register automata

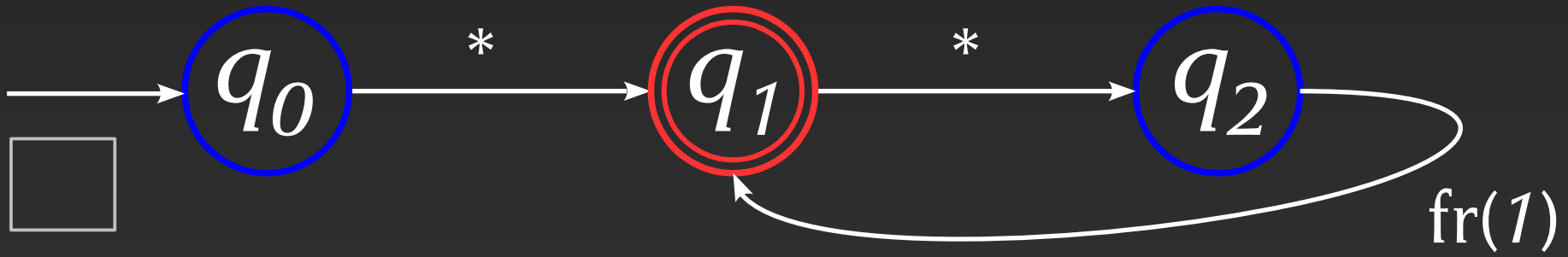


# Fresh-register automata

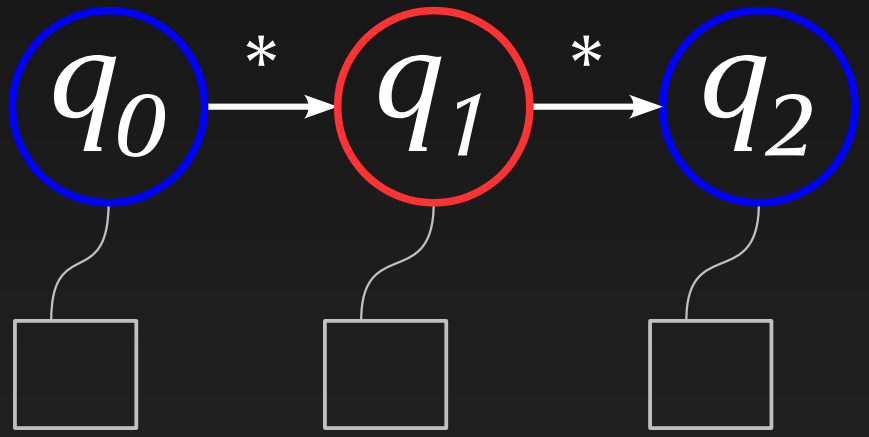



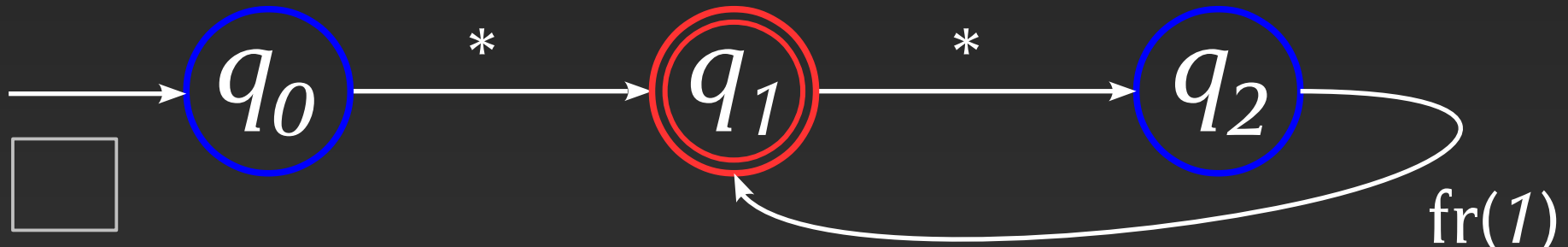
# Fresh-register automata



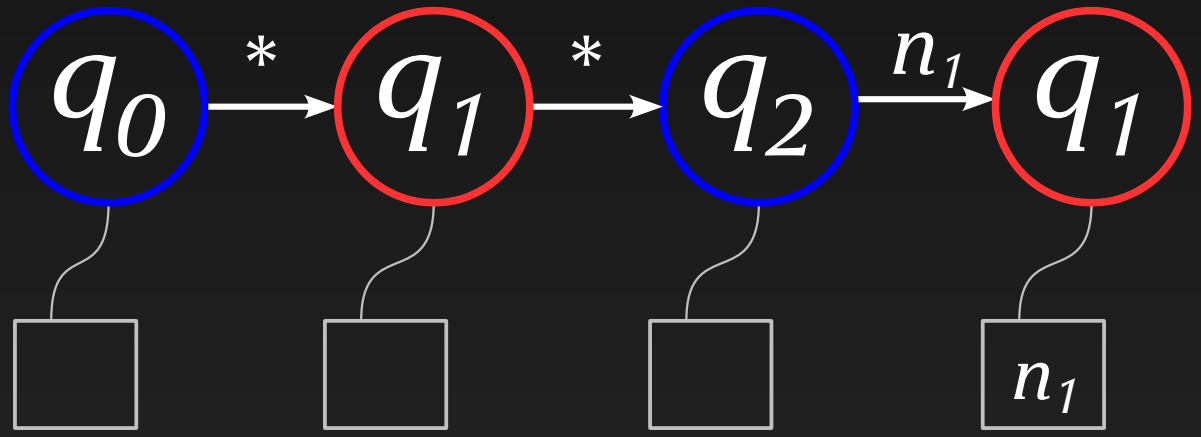



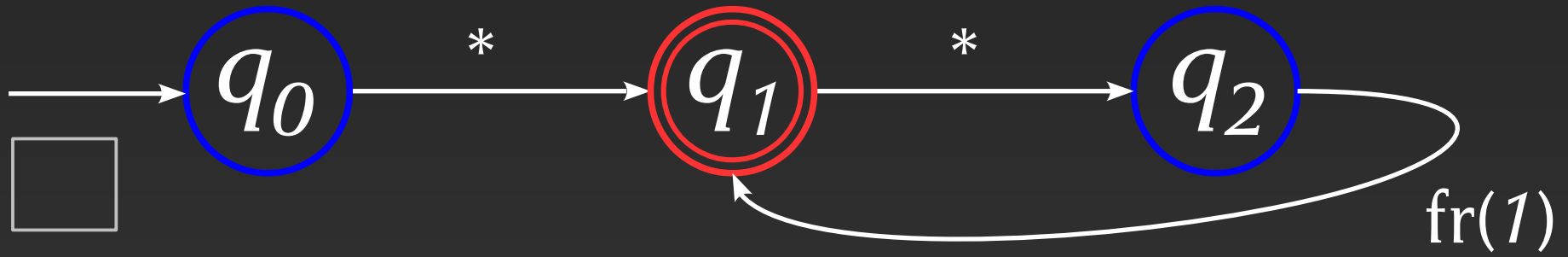
# Fresh-register automata



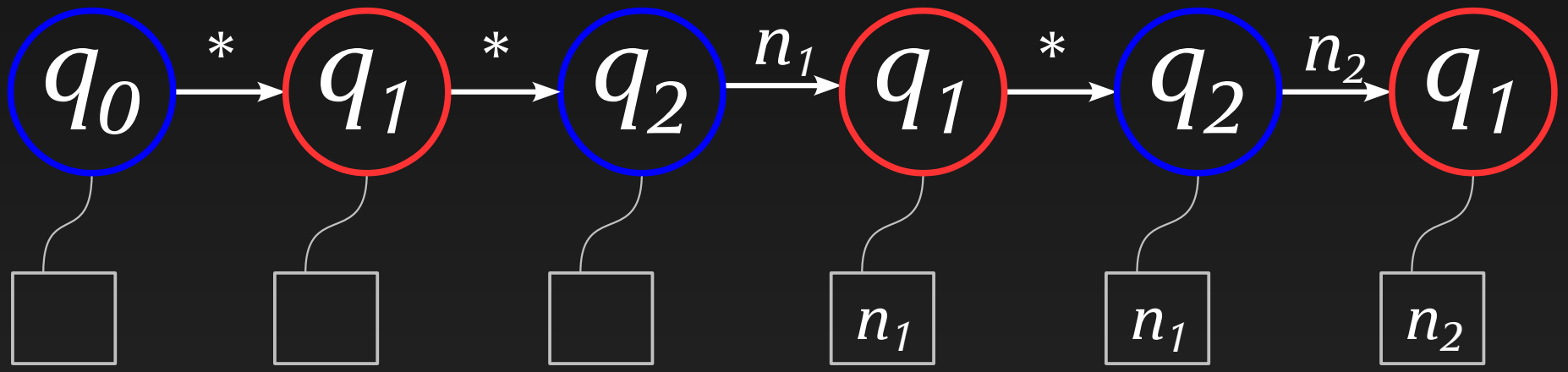



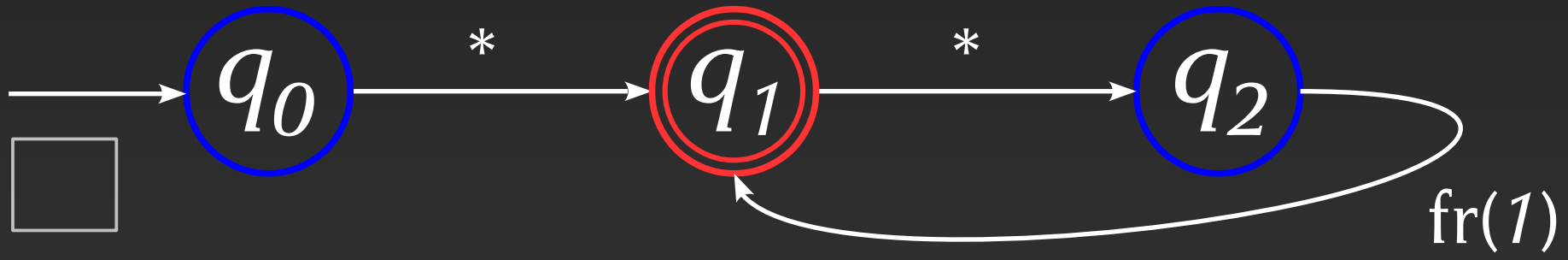
# Fresh-register automata



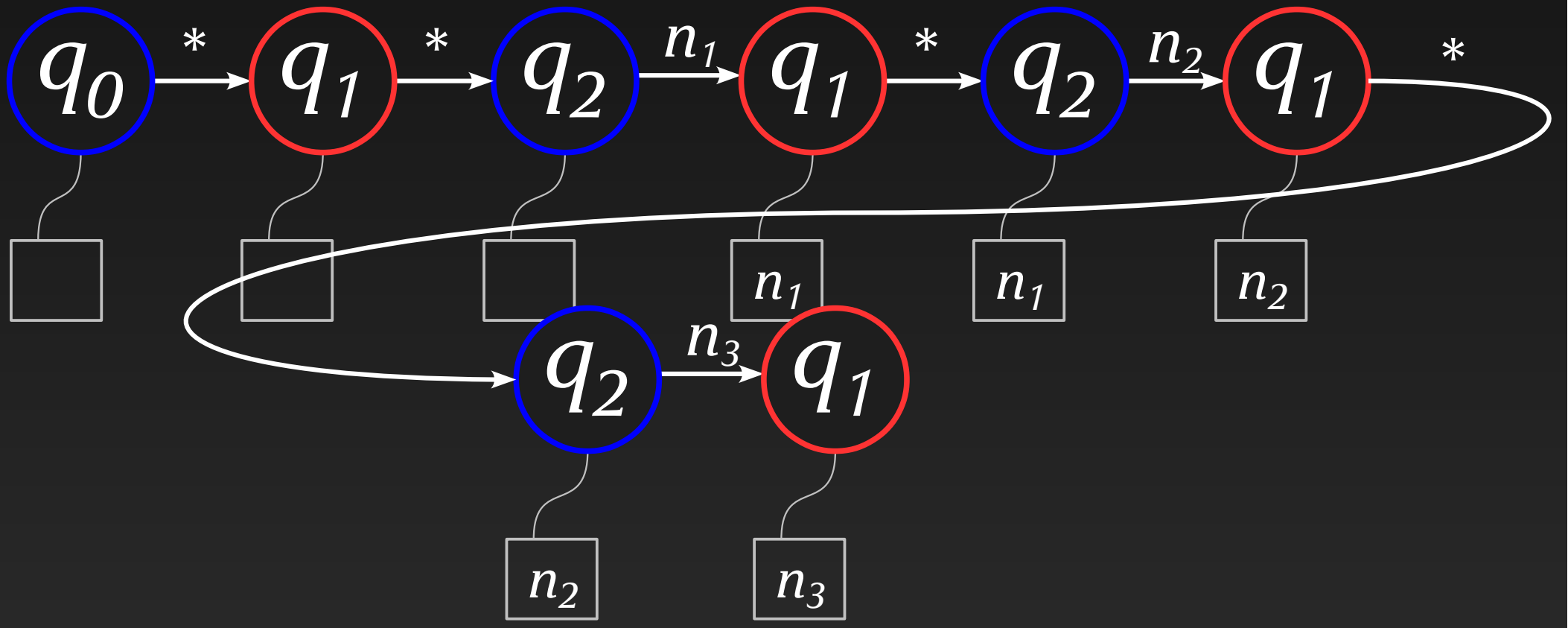



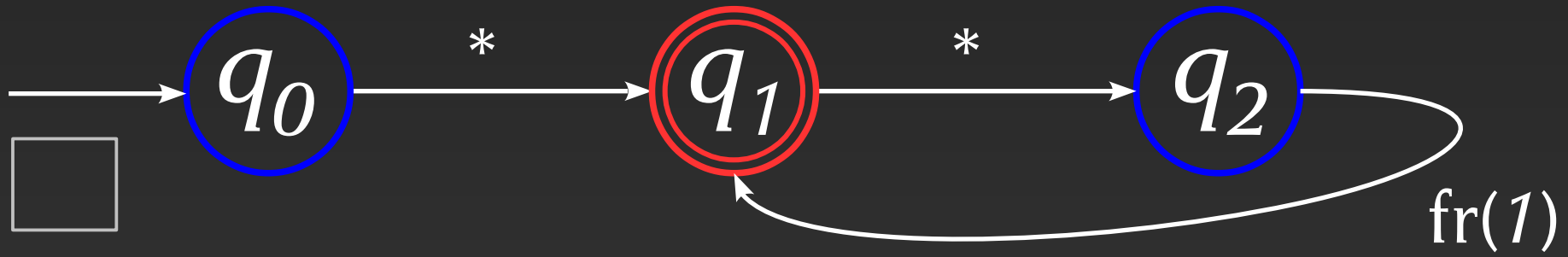
# Fresh-register automata



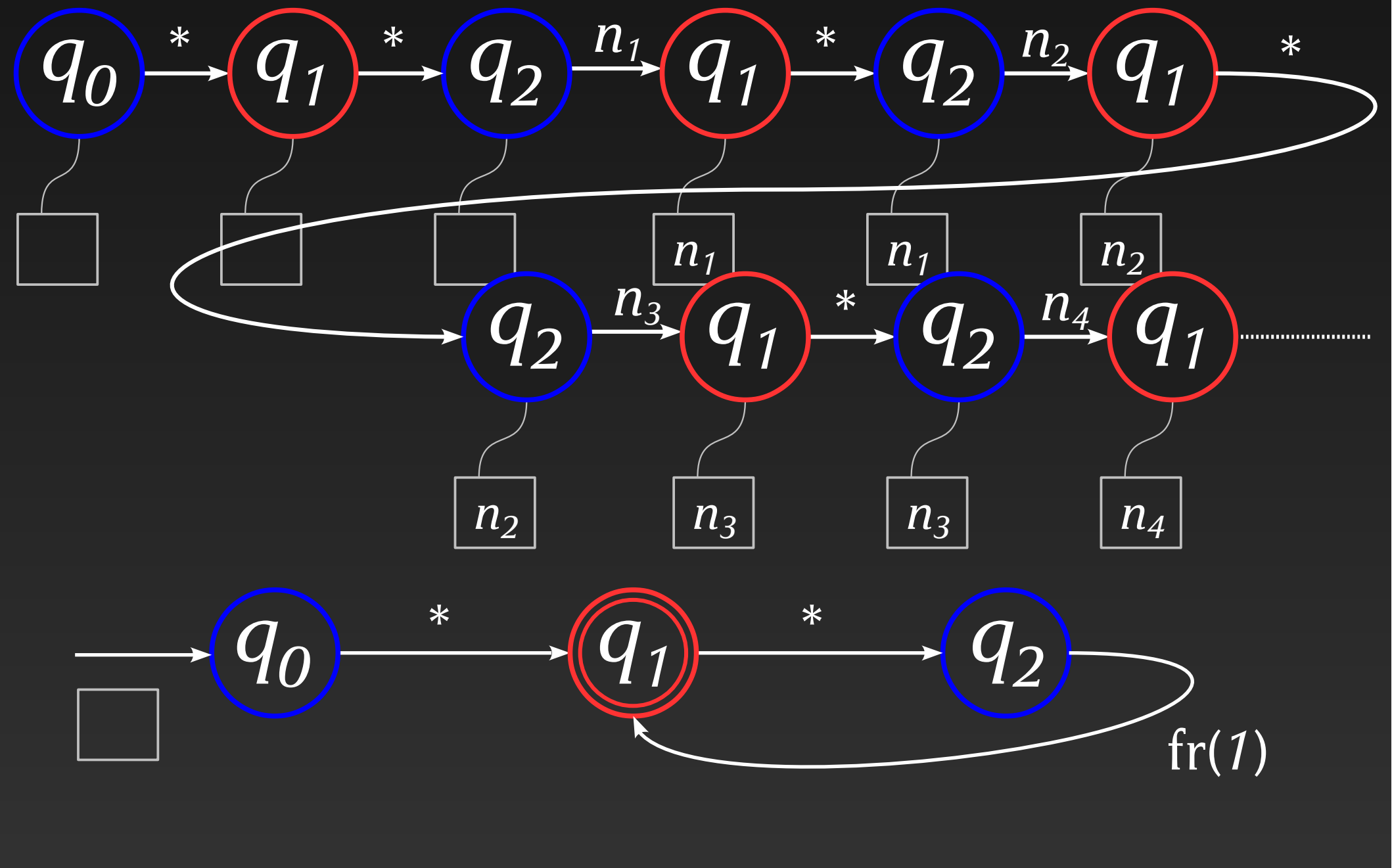



# Fresh-register automata

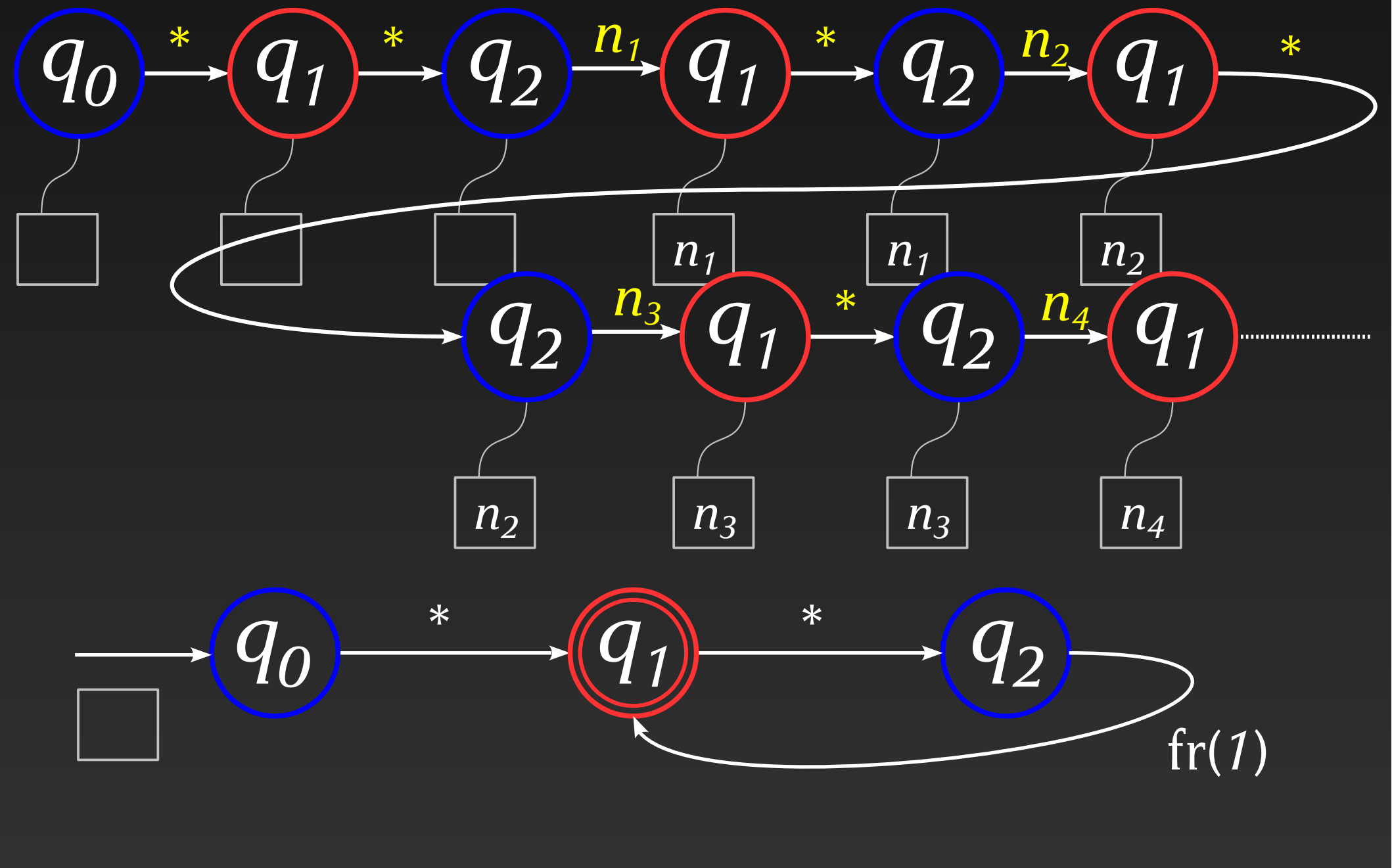


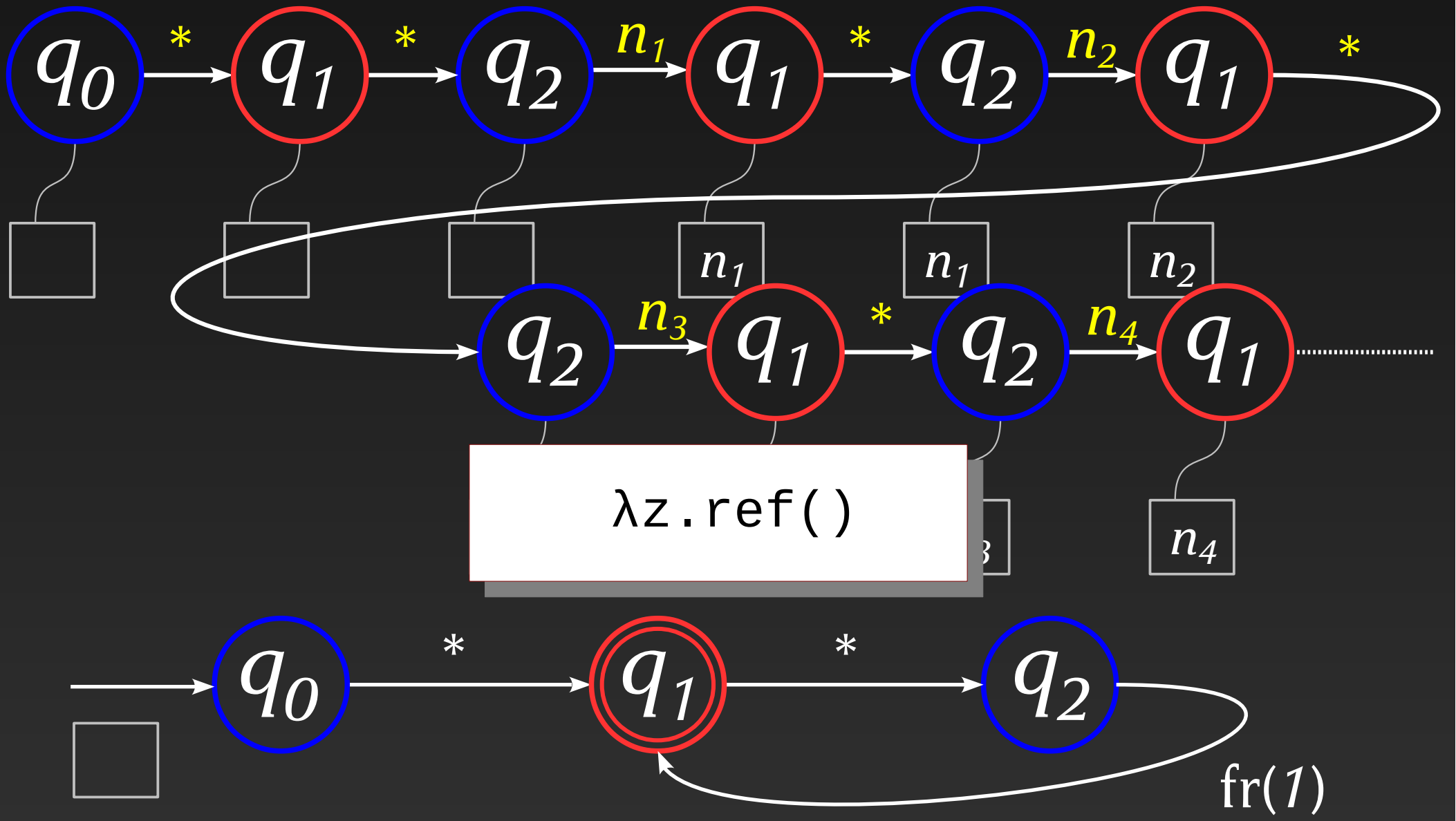
# Fresh-register automata



# Fresh-register automata



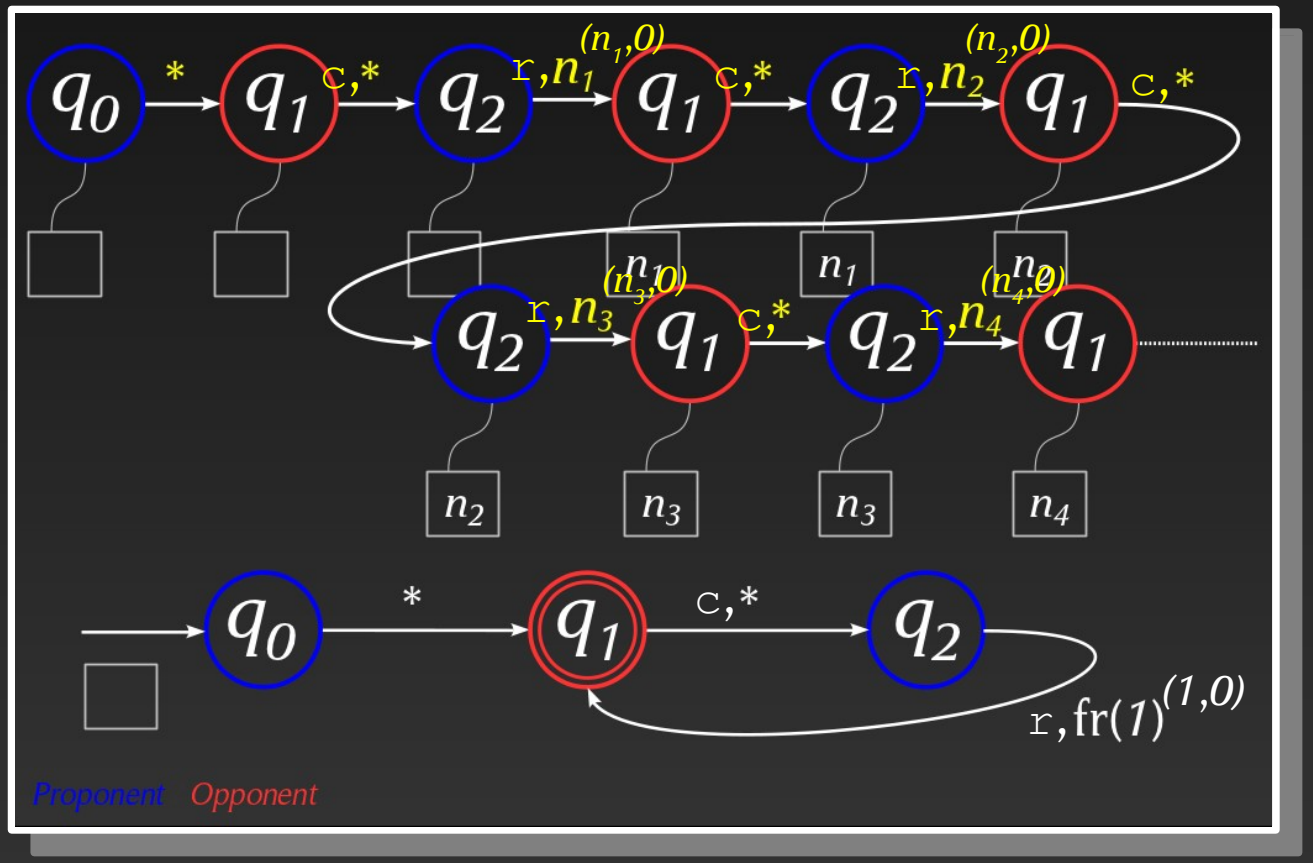
# Fresh-register automata




*Proponent* *Opponent*

# Automata

- Tz.11: Fresh-Register Automata
- Mu.Tz. 11': Algorithmic games for RedML*



# Automata

- Tz.11: Fresh-Register Automata
- Mu.Tz. 11': Algorithmic games for RedML*

$$M \cong N \iff \mathcal{A}_M \sim \mathcal{A}_N$$

# Automata

- Tz.11:  Fresh-Register Automata
- Mu.Tz. 11':  Algorithmic games for RedML*

$$M \cong N \iff \mathcal{A}_M \sim \mathcal{A}_N$$

- Mu.Tz. 12 : Algorithmic games for *full* intref's
  + Pushdown FRA's

# Automata

- Tz.11: Fresh-Register Automata
- Mu.Tz. 11': Algorithmic games for RedML*

$$M \cong N \iff \mathcal{A}_M \sim \mathcal{A}_N$$

- Mu.Tz. 12 : Algorithmic games for *full* intref's
  + Pushdown FRA's
- Mu.Tz. … : Algorithmic games for Java

# Games with names