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

```
λx. ref() : unit → unit ref
```

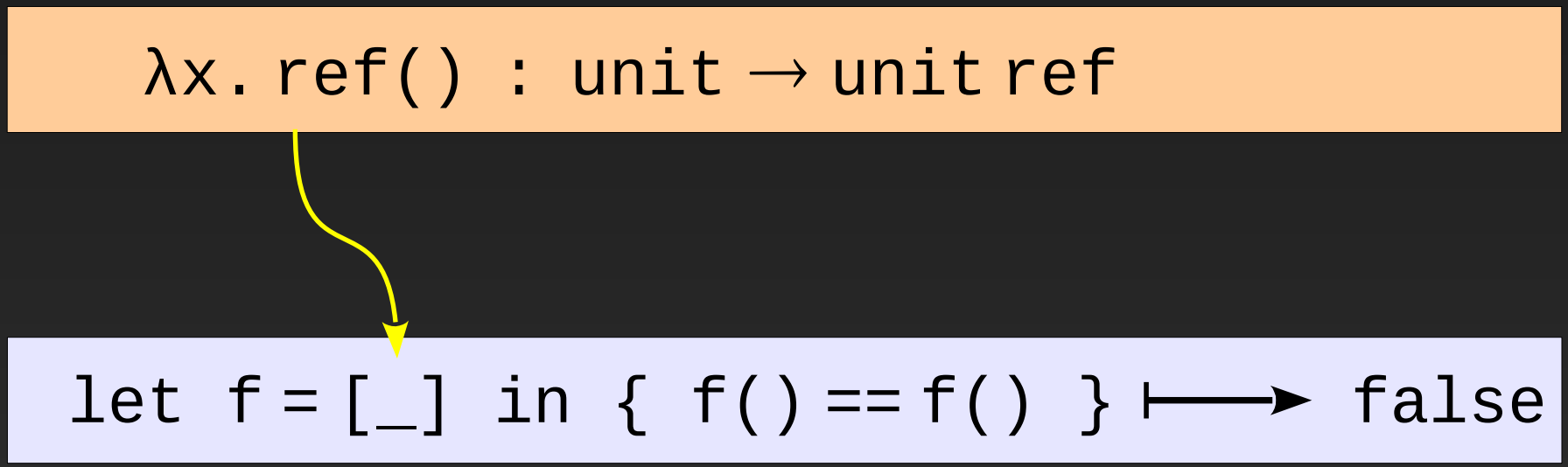
```
let f = [_] in { f() == f() }
```

Computation with names

```
λx. ref() : unit → unit ref
```

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

Computation with names



Example: Reduced ML

basic programming language with functions + integer references

Example: Reduced ML

basic programming language with functions + integer references

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

$$P \cong P'$$

same observable behaviour in every context

Example: Reduced ML

basic programming language with functions + integer references

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

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

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

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

Semantics

How to assign meaning to programs?

$$[\![\_]\!] : \mathrm{Syntax} \longrightarrow U$$

Semantics

How to assign meaning to programs?

$$[\,\_\,] : \text{Syntax} \longrightarrow U$$

such that:

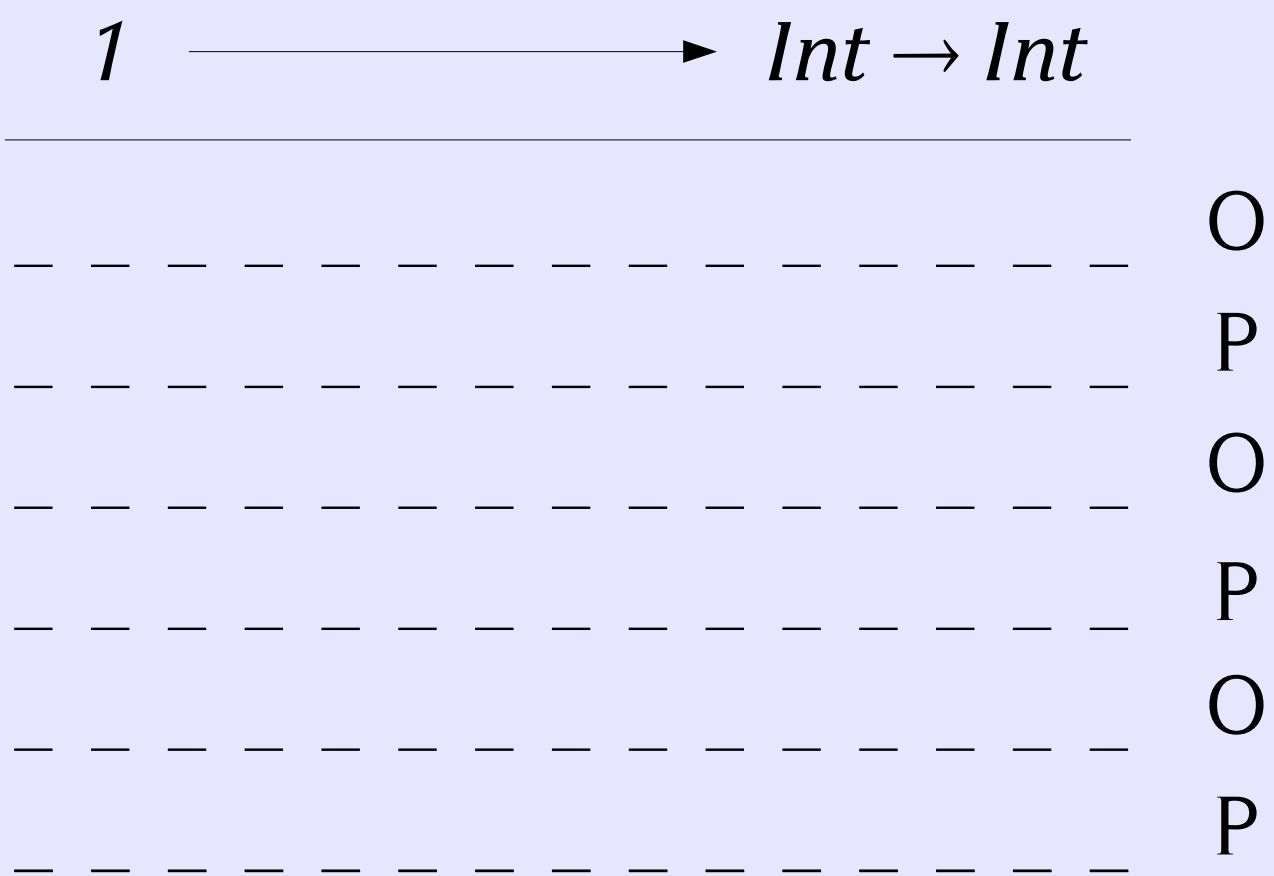
$$P \cong P' \Leftrightarrow [\,P\,] = [\,P'\,]$$

Game Semantics

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

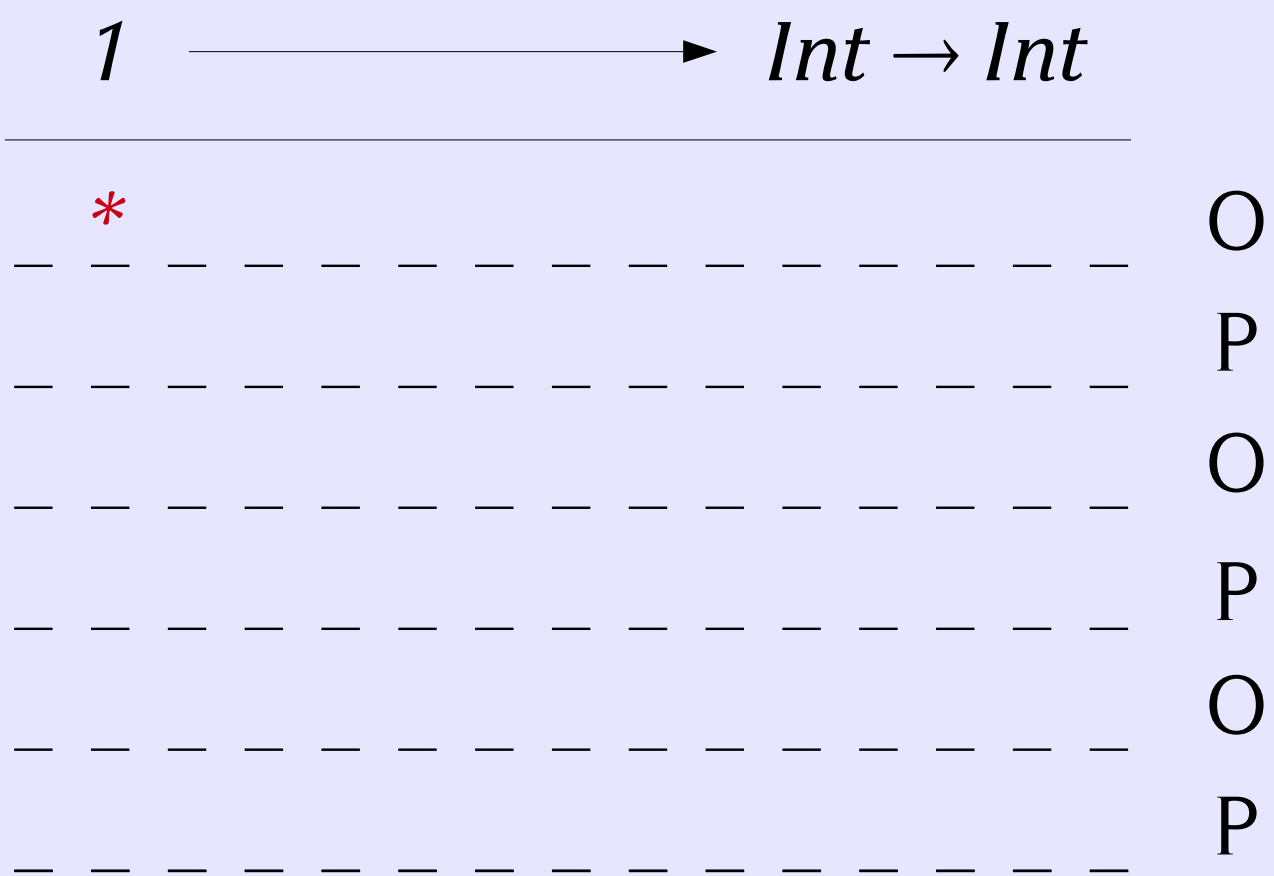
Example

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



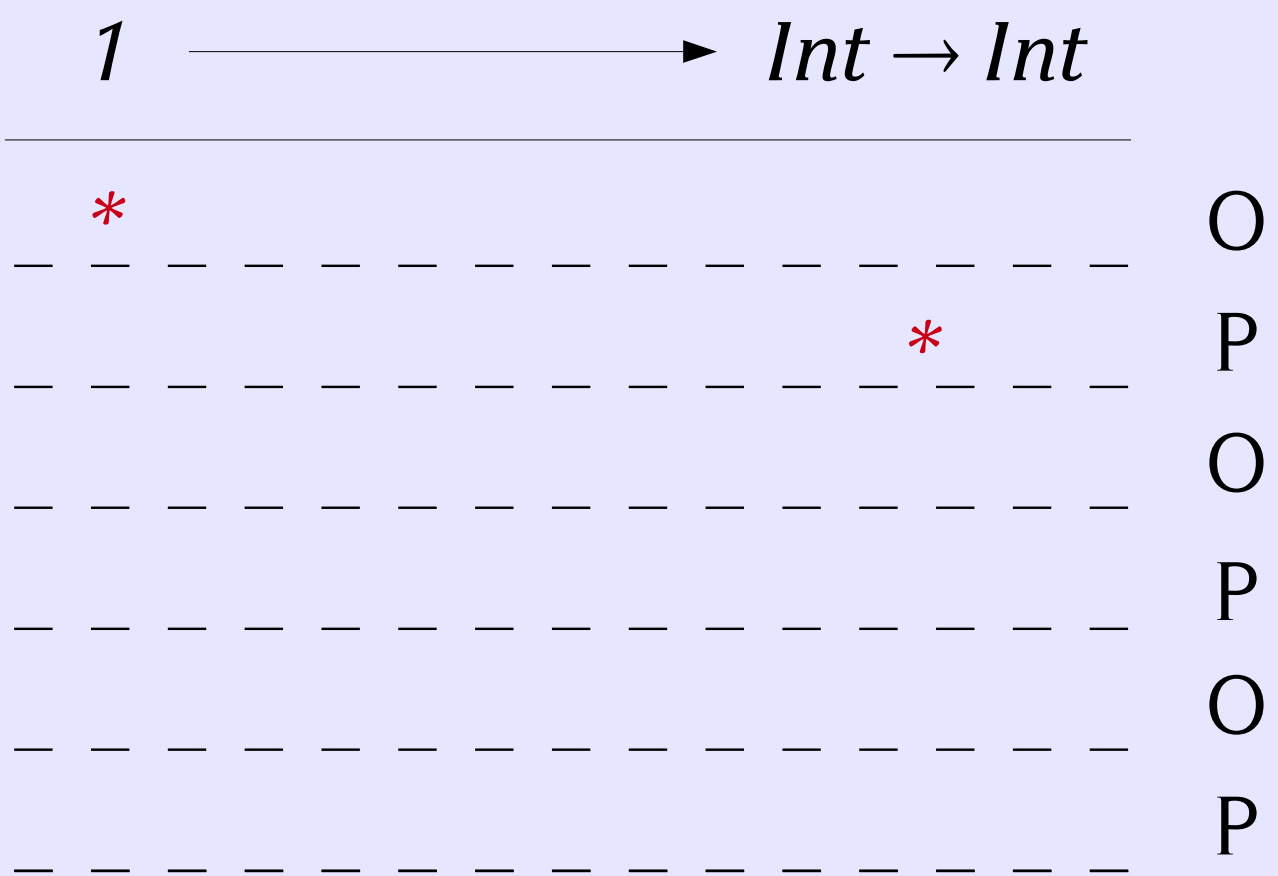
Example

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



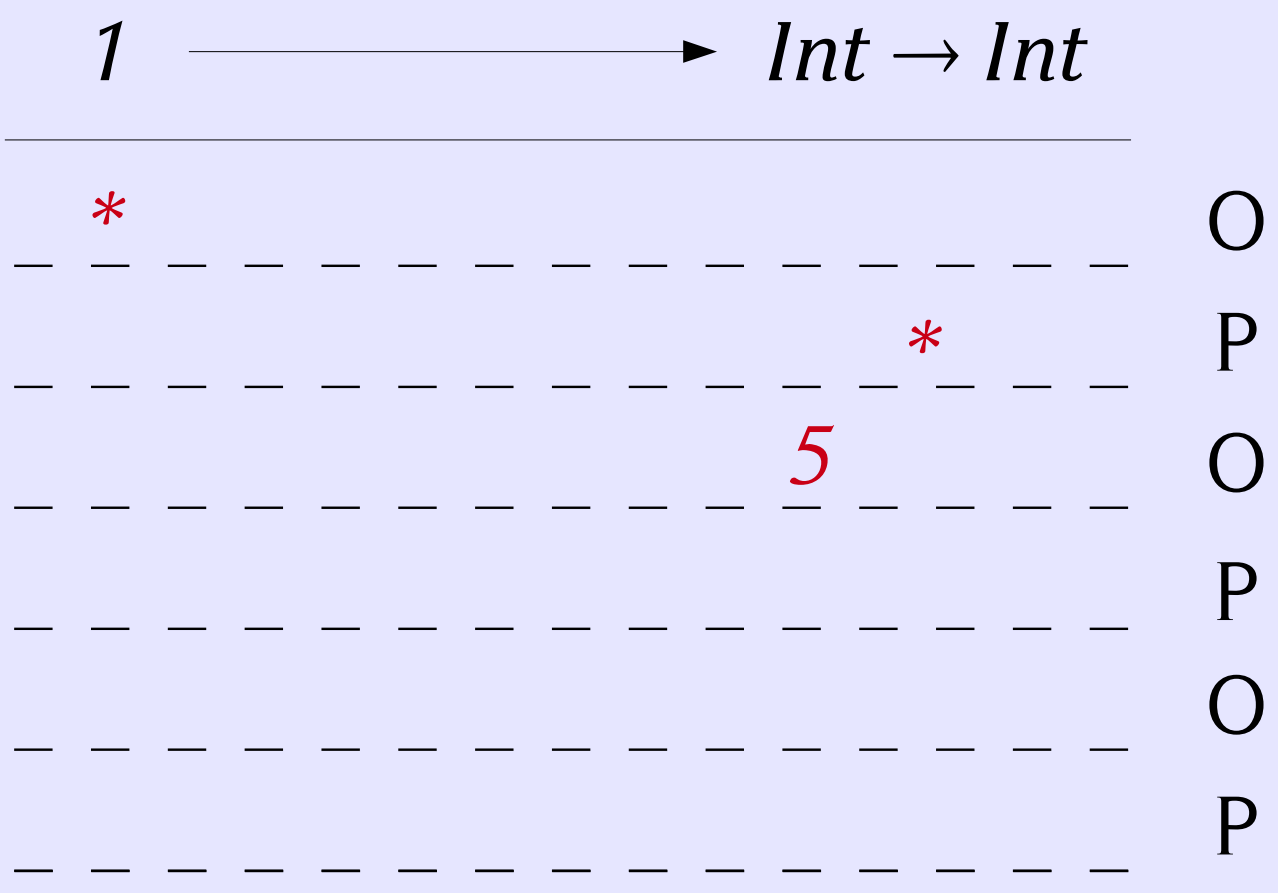
Example

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



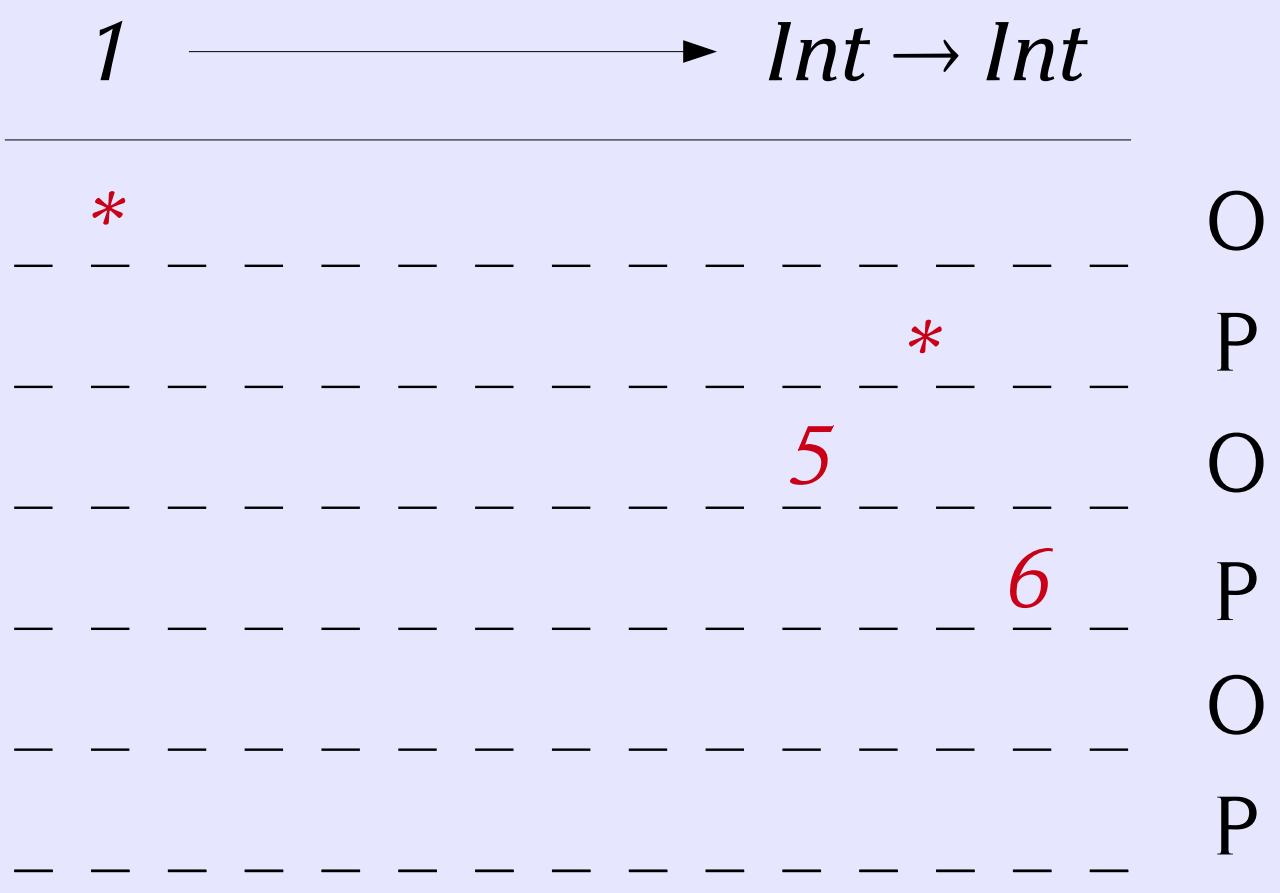
Example

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



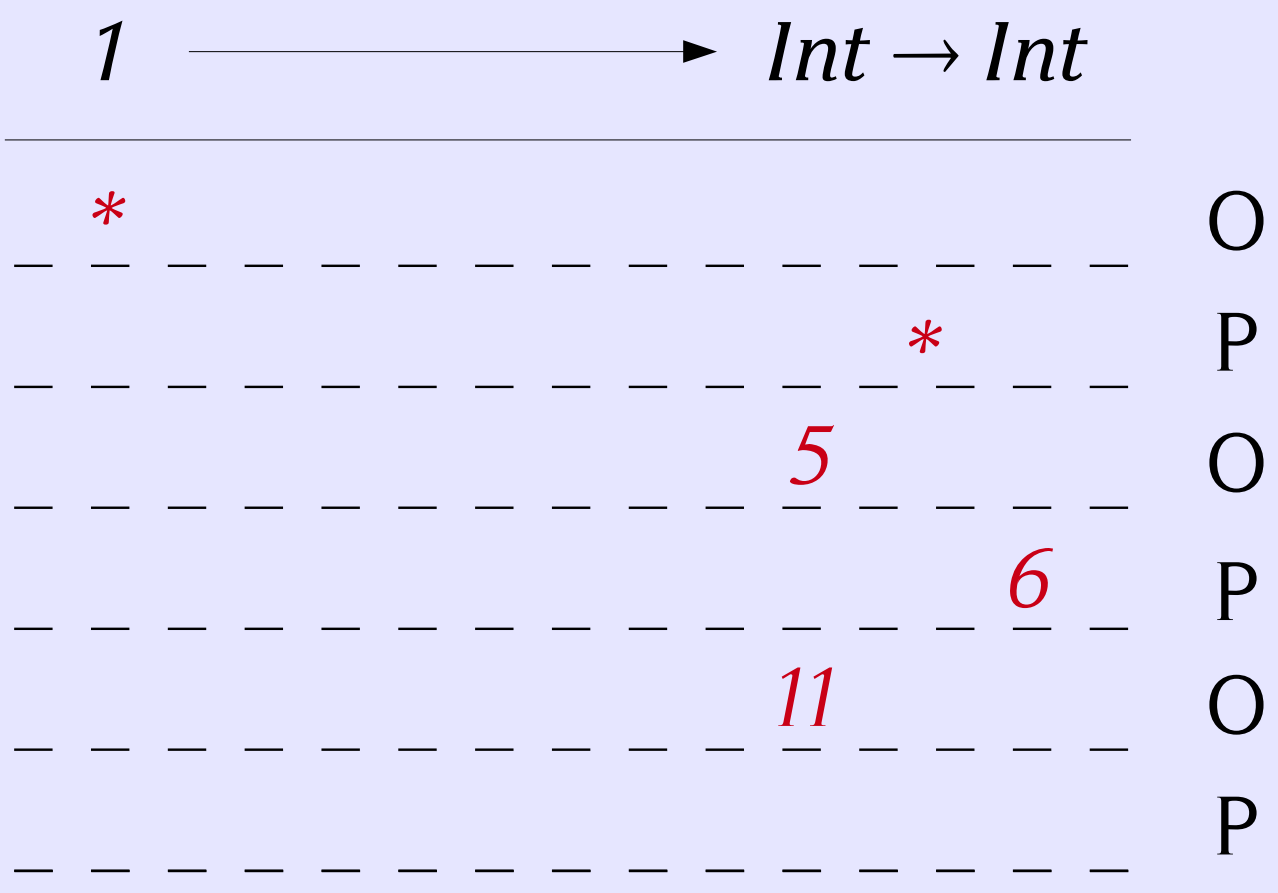
Example

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



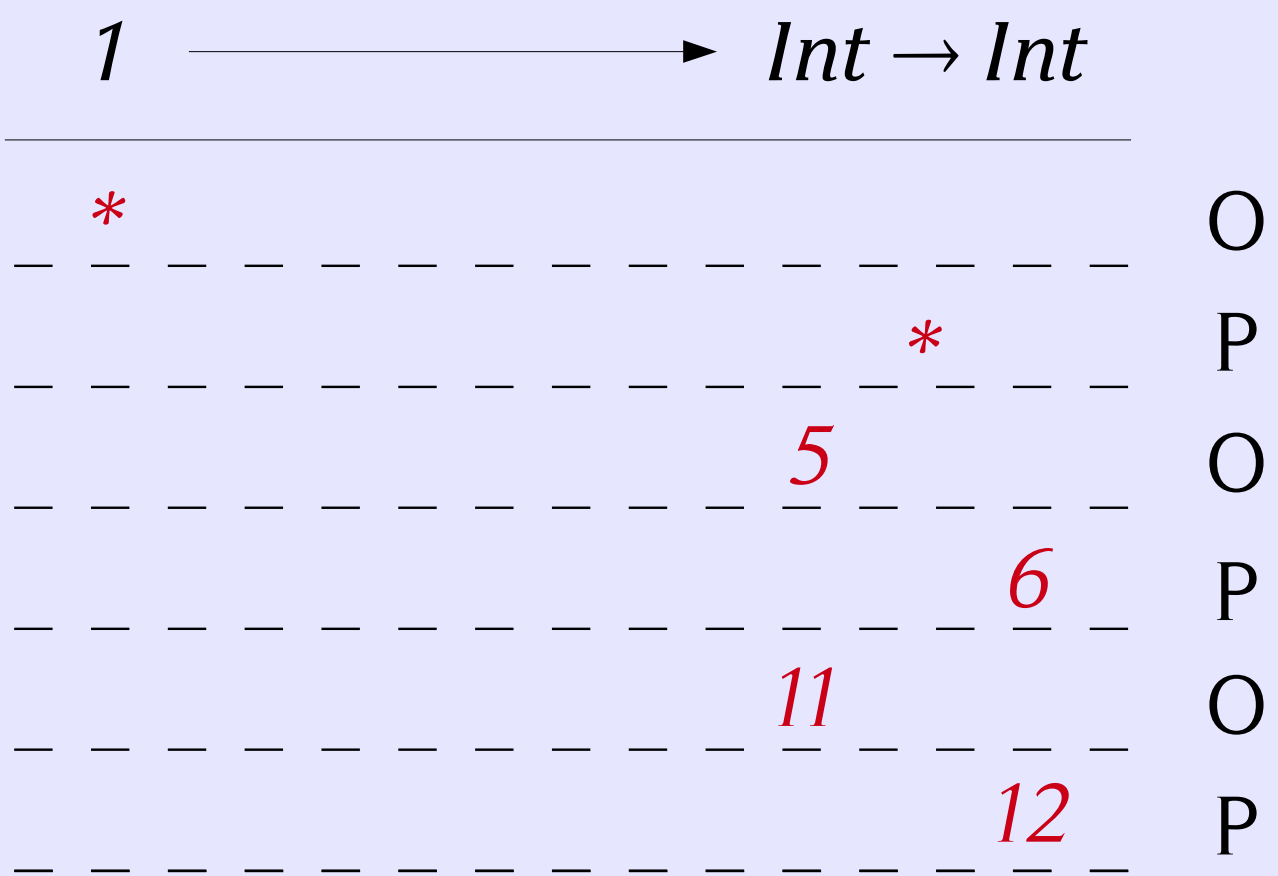
Example

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



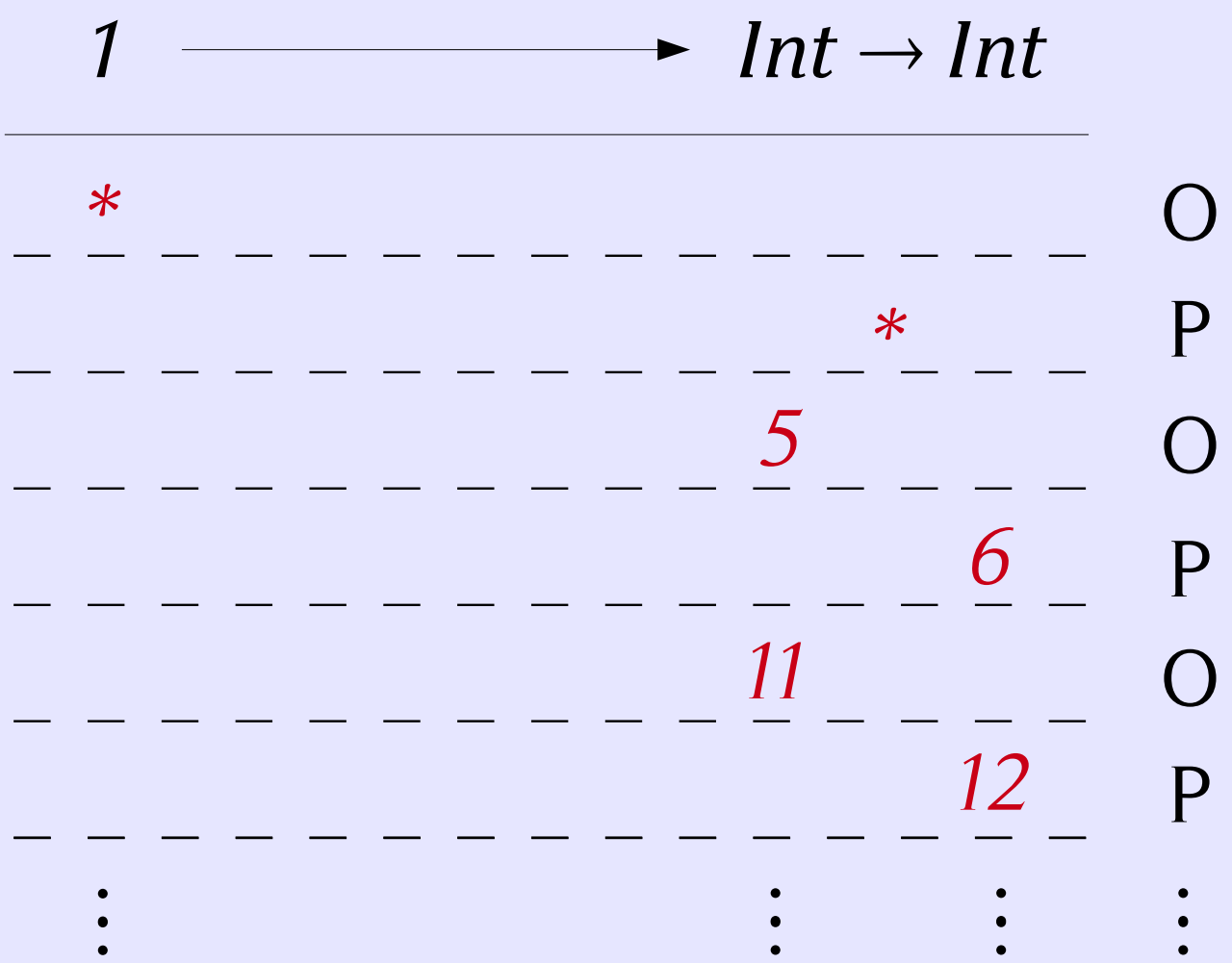
Example

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



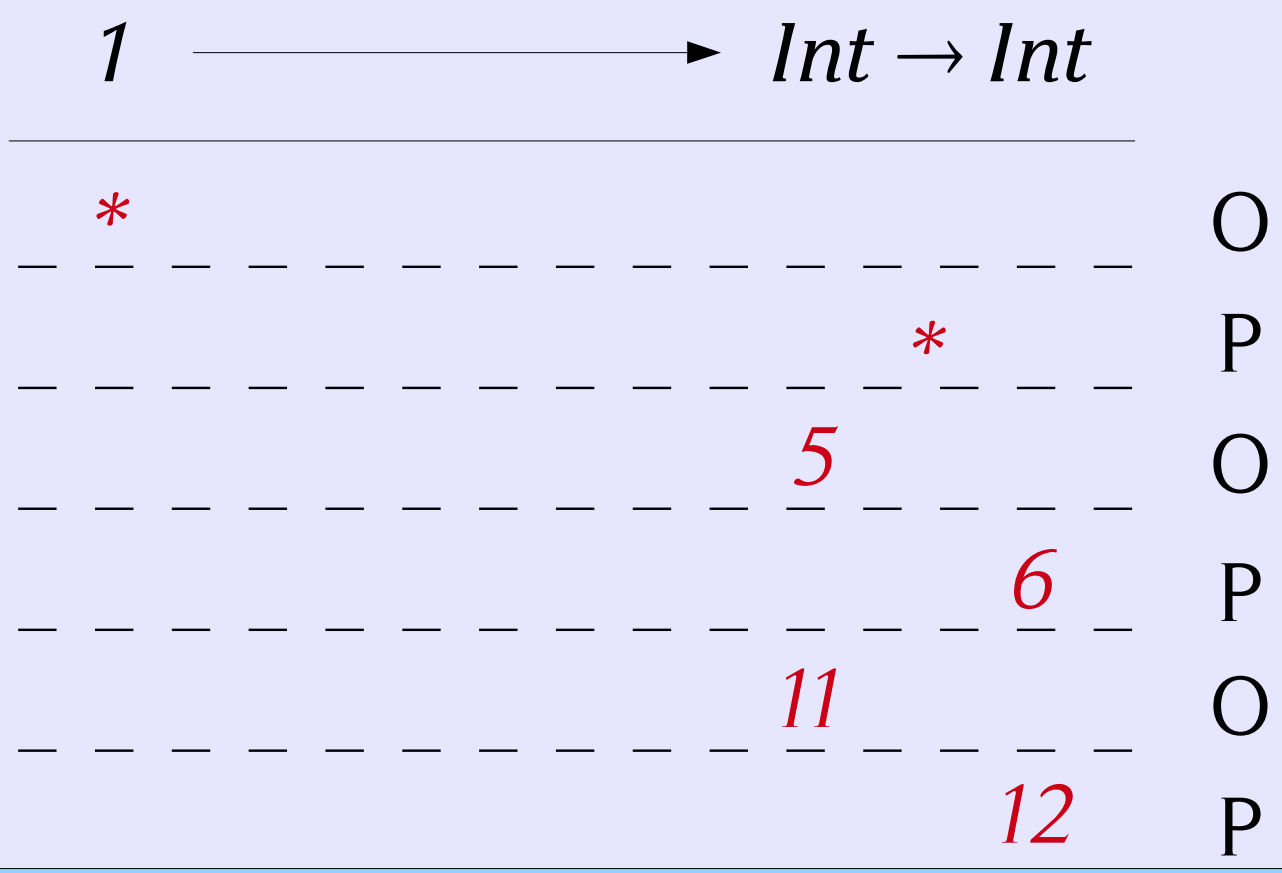
Example

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



Example

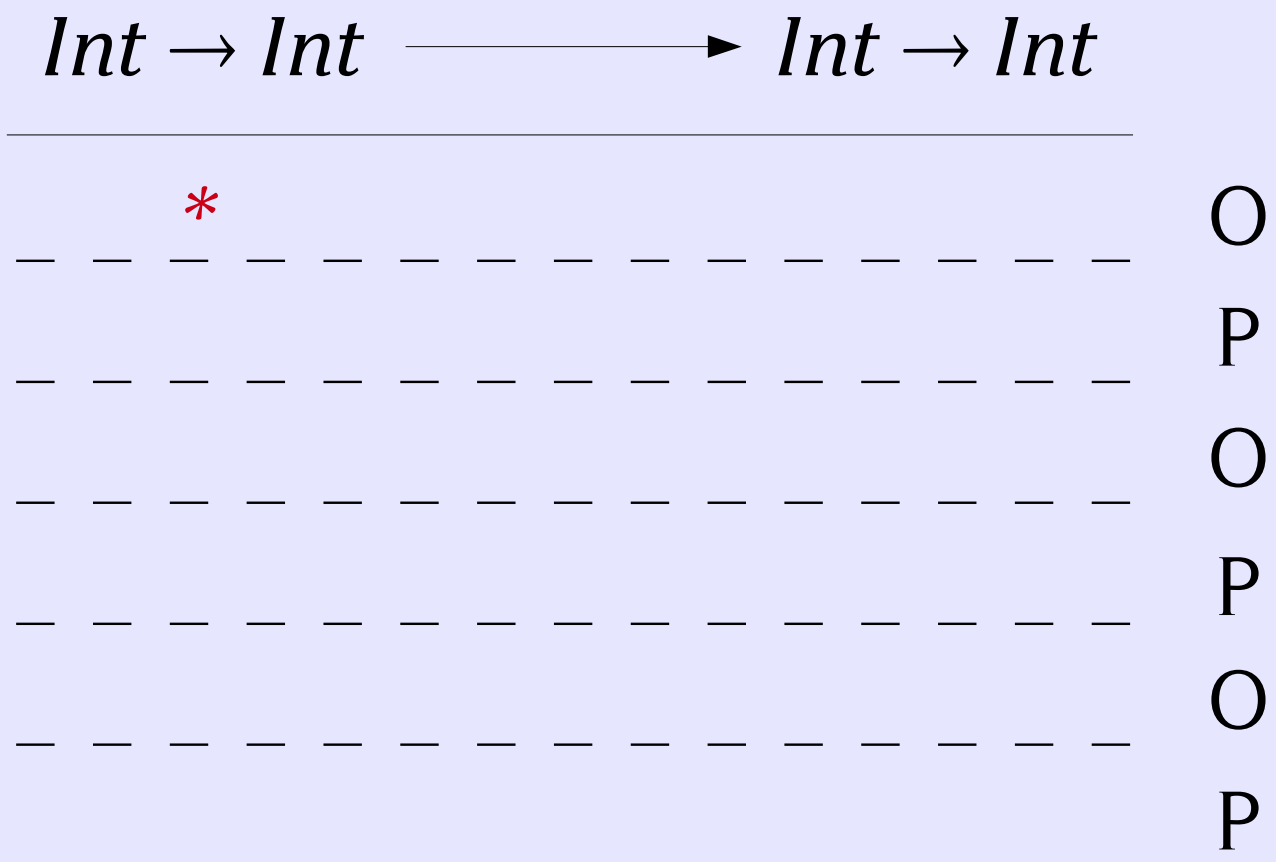
$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



$[\lambda x.x+1] = \{ * * 5\ 6\ 11\ 12 \ldots \}$

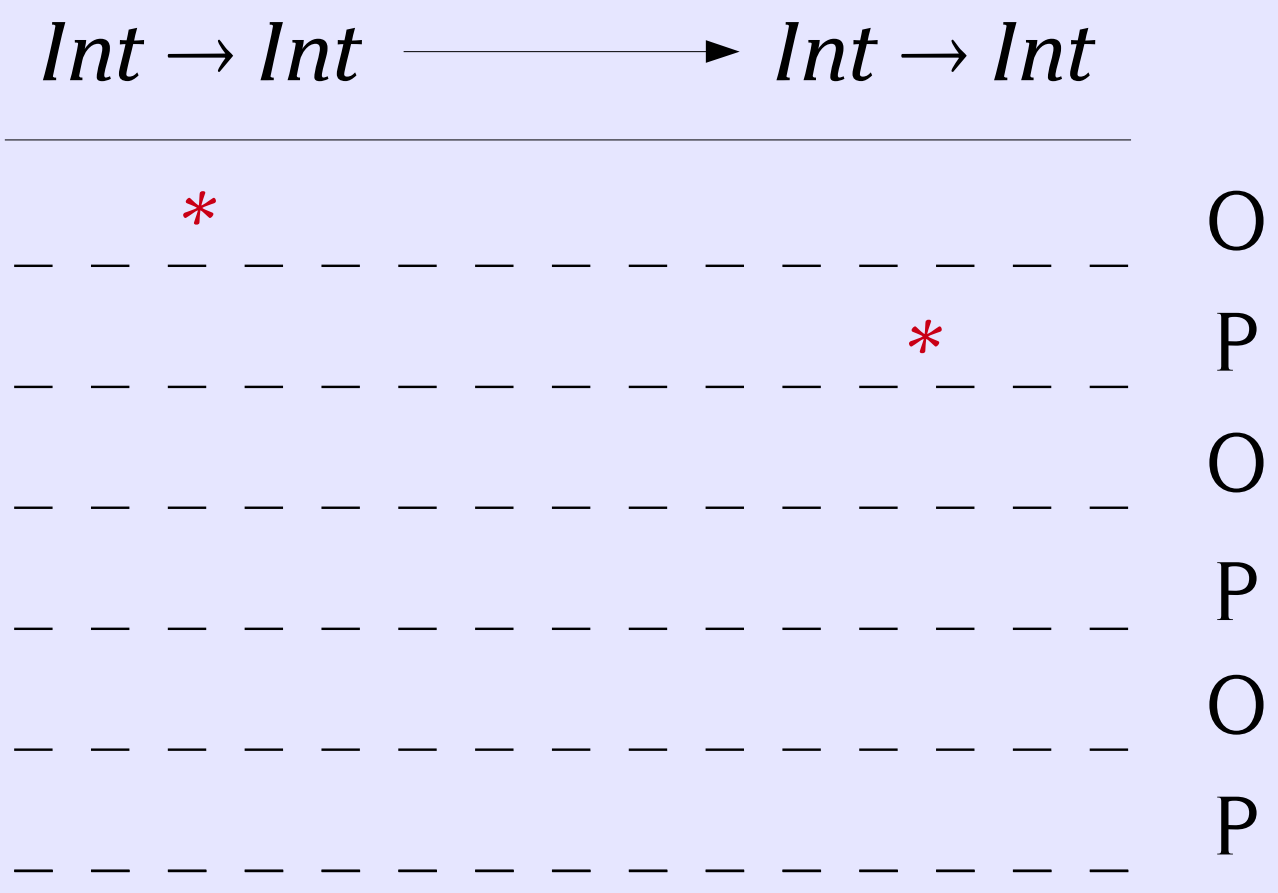
Example

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



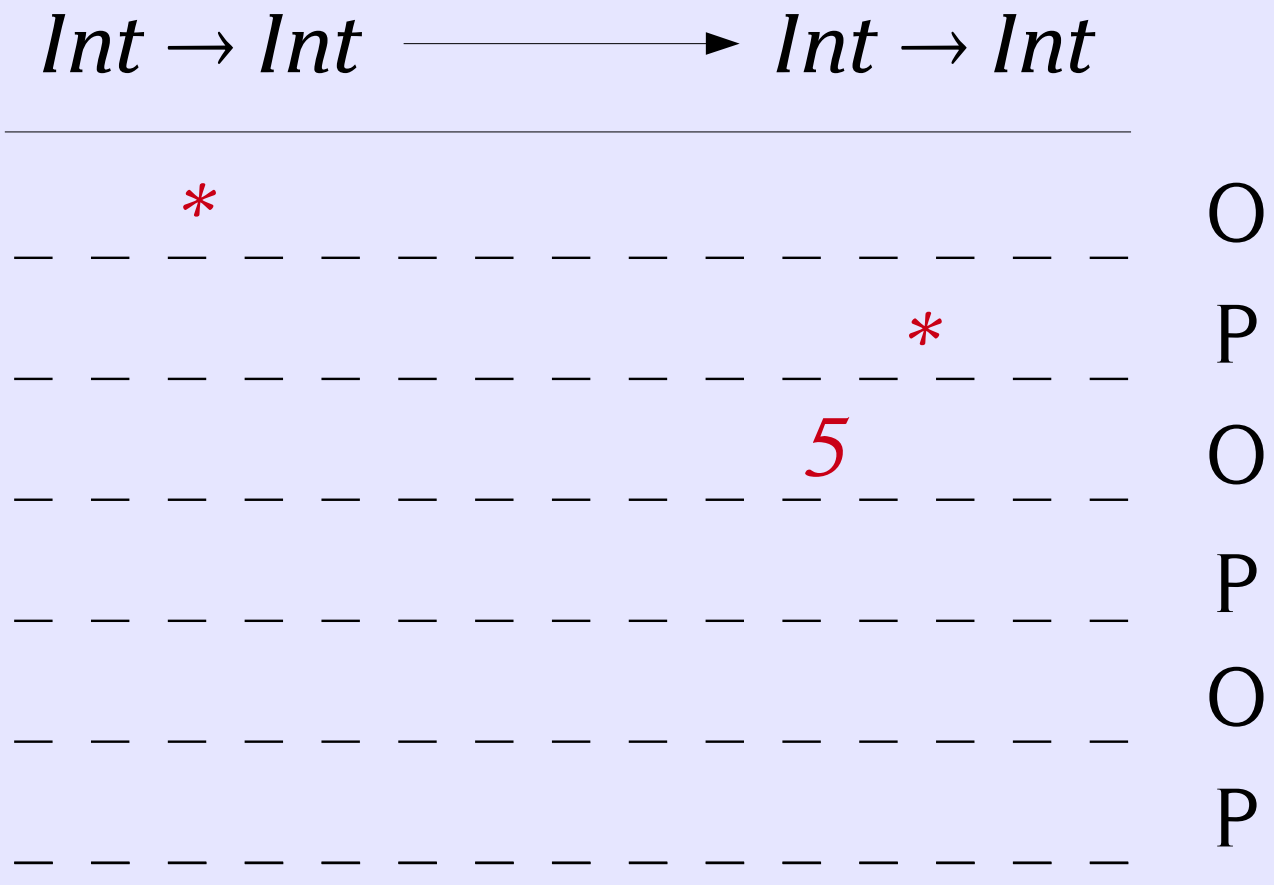
Example

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



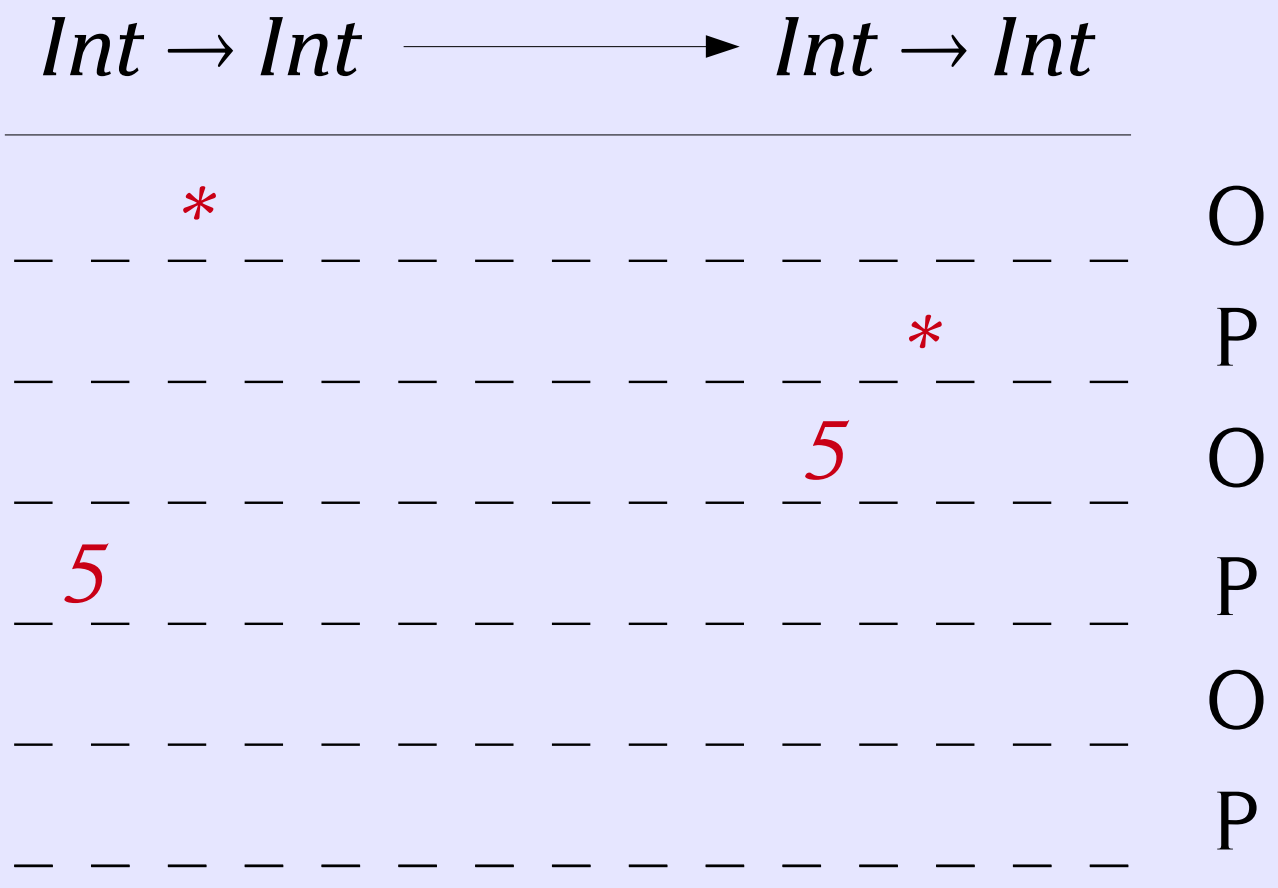
Example

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



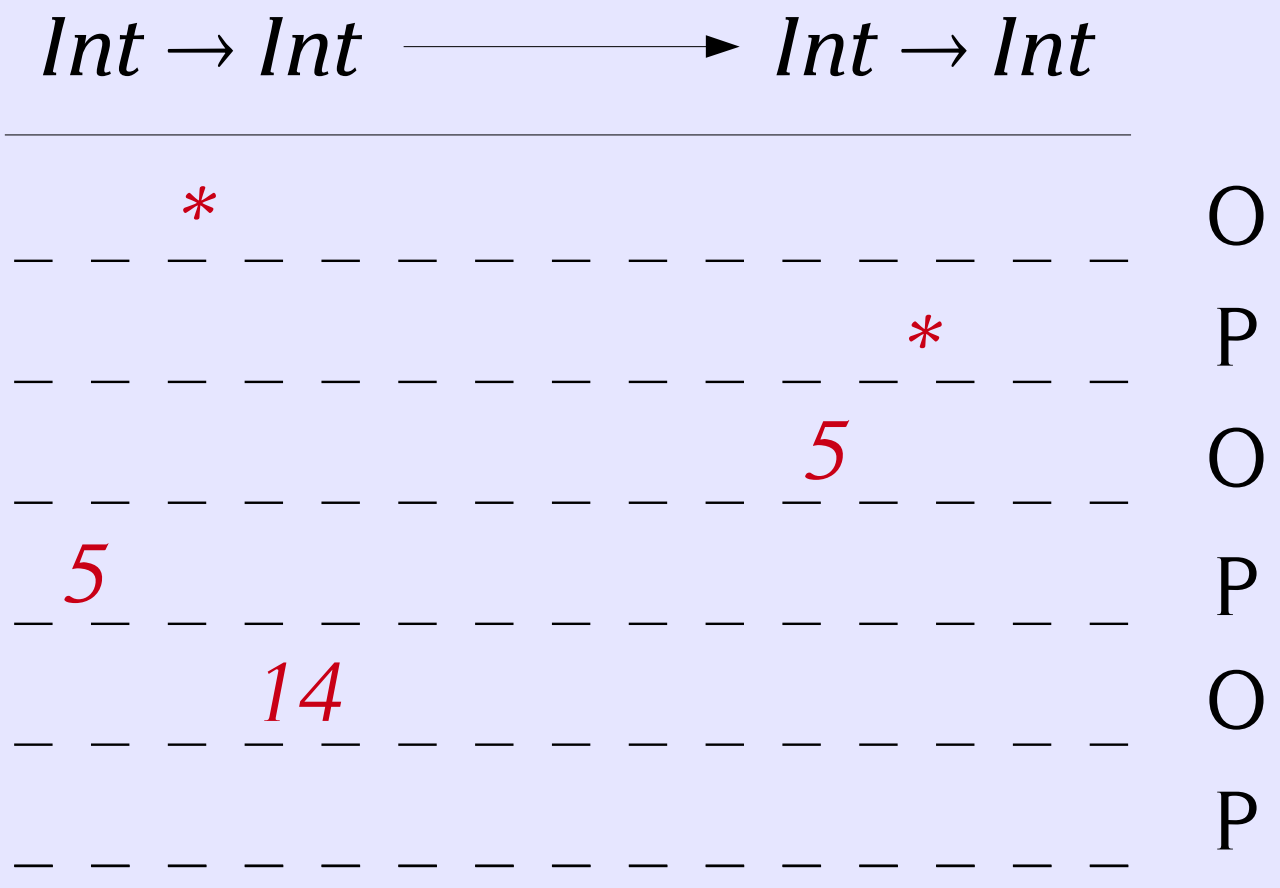
Example

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



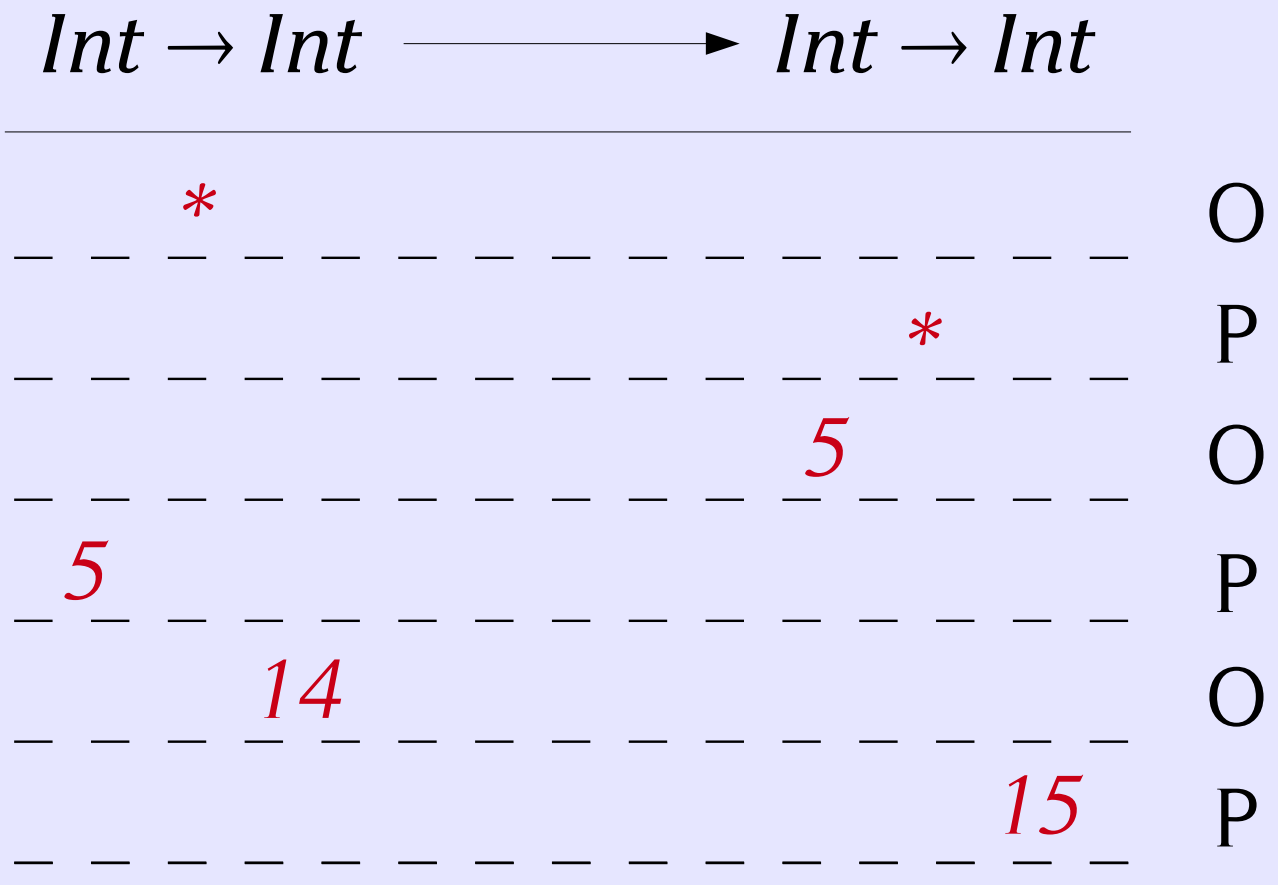
Example

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



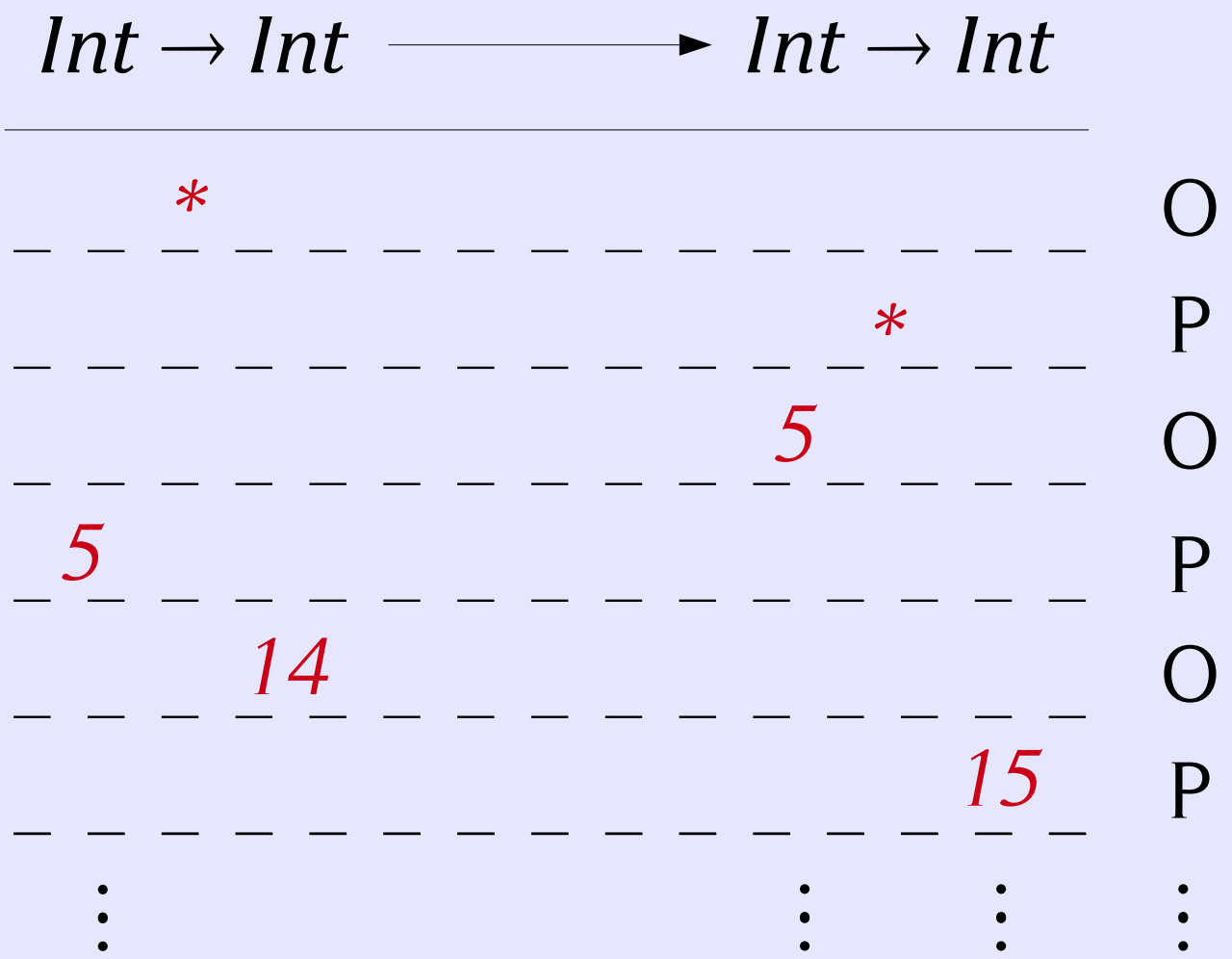
Example

`f : int→int ⊢ λx.f(x)+1 : int→int`



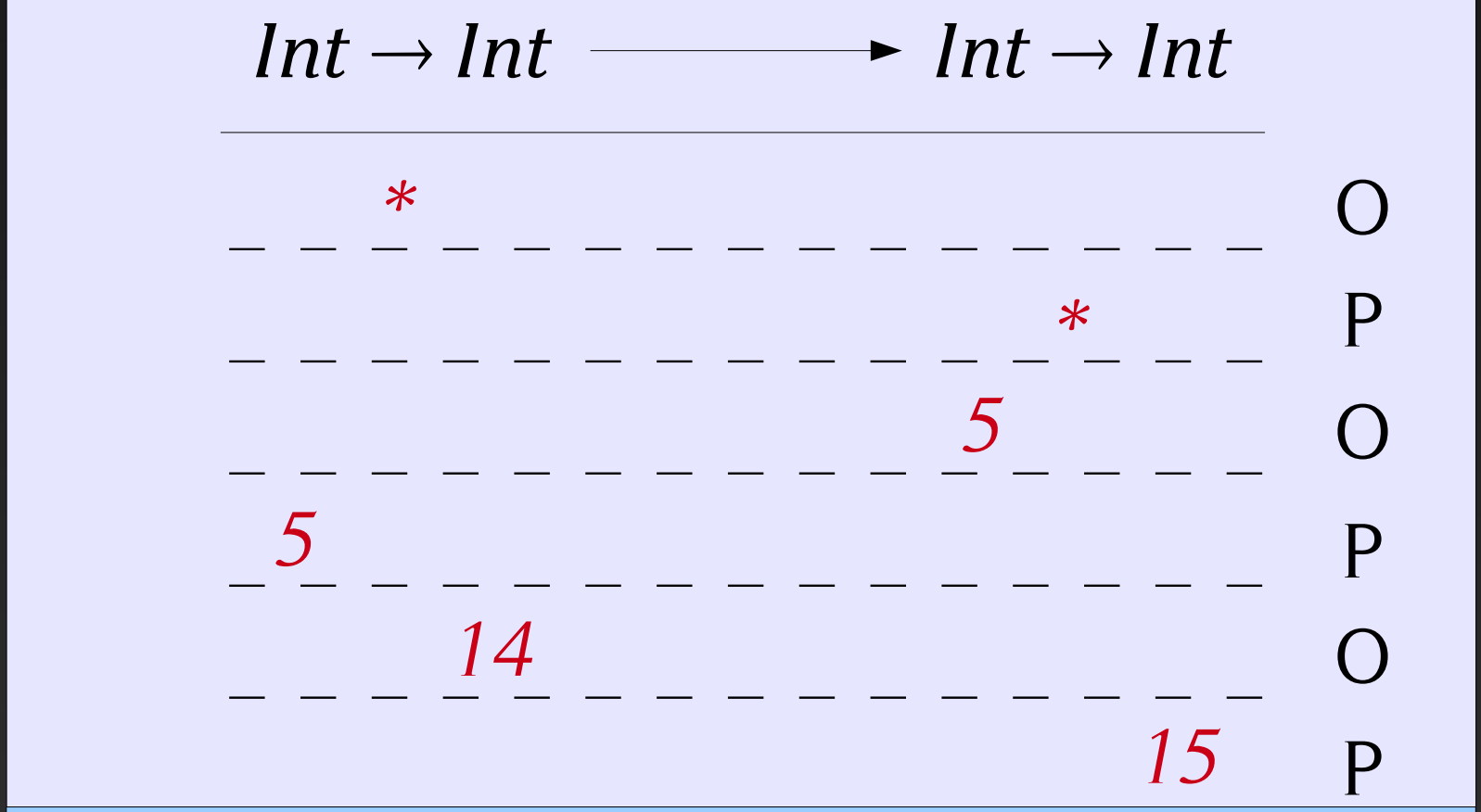
Example

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



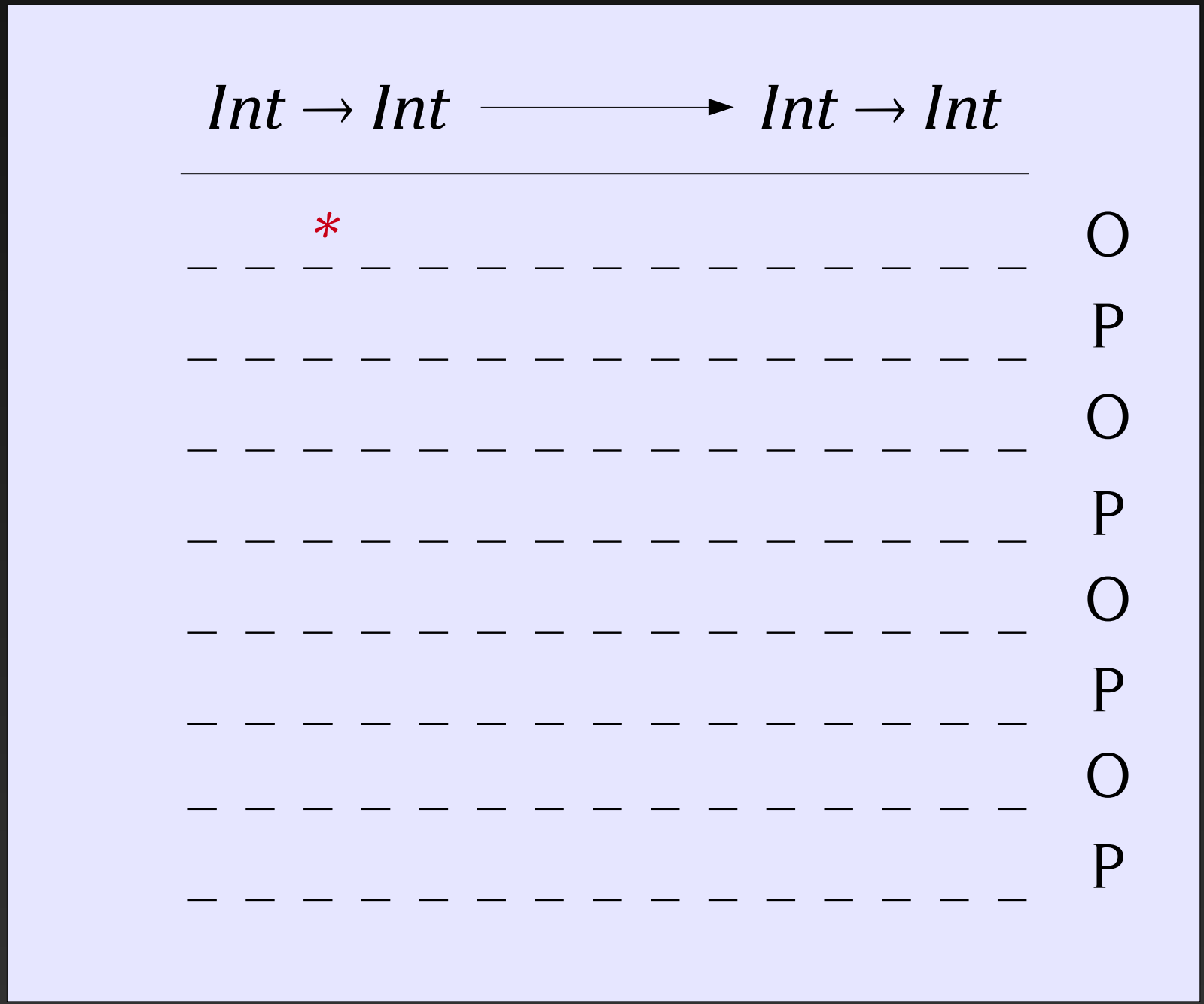
Example

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

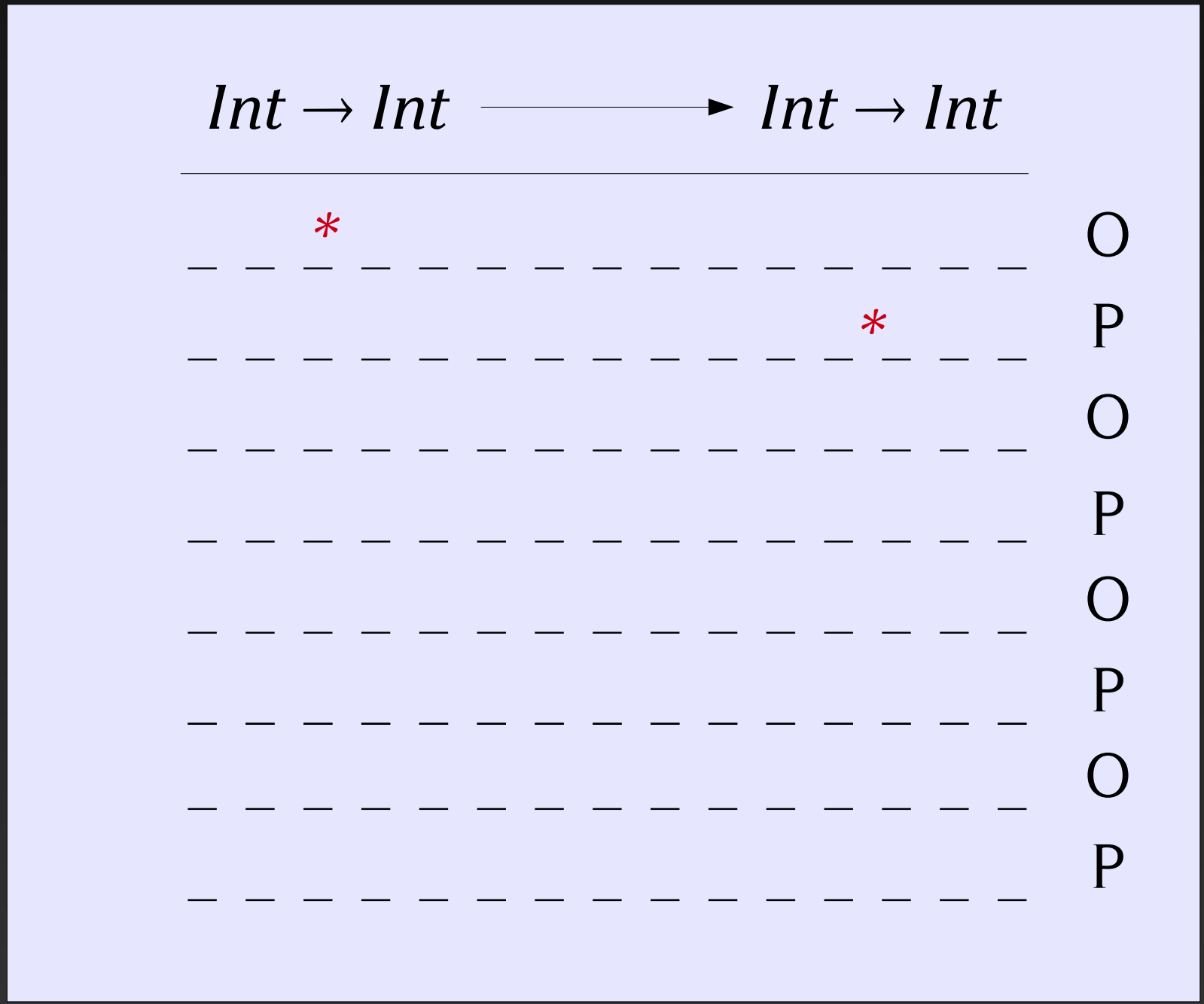


$[\lambda x.f(x)+1] = \{ * * 5\ 5\ 14\ 15 \ldots \}$

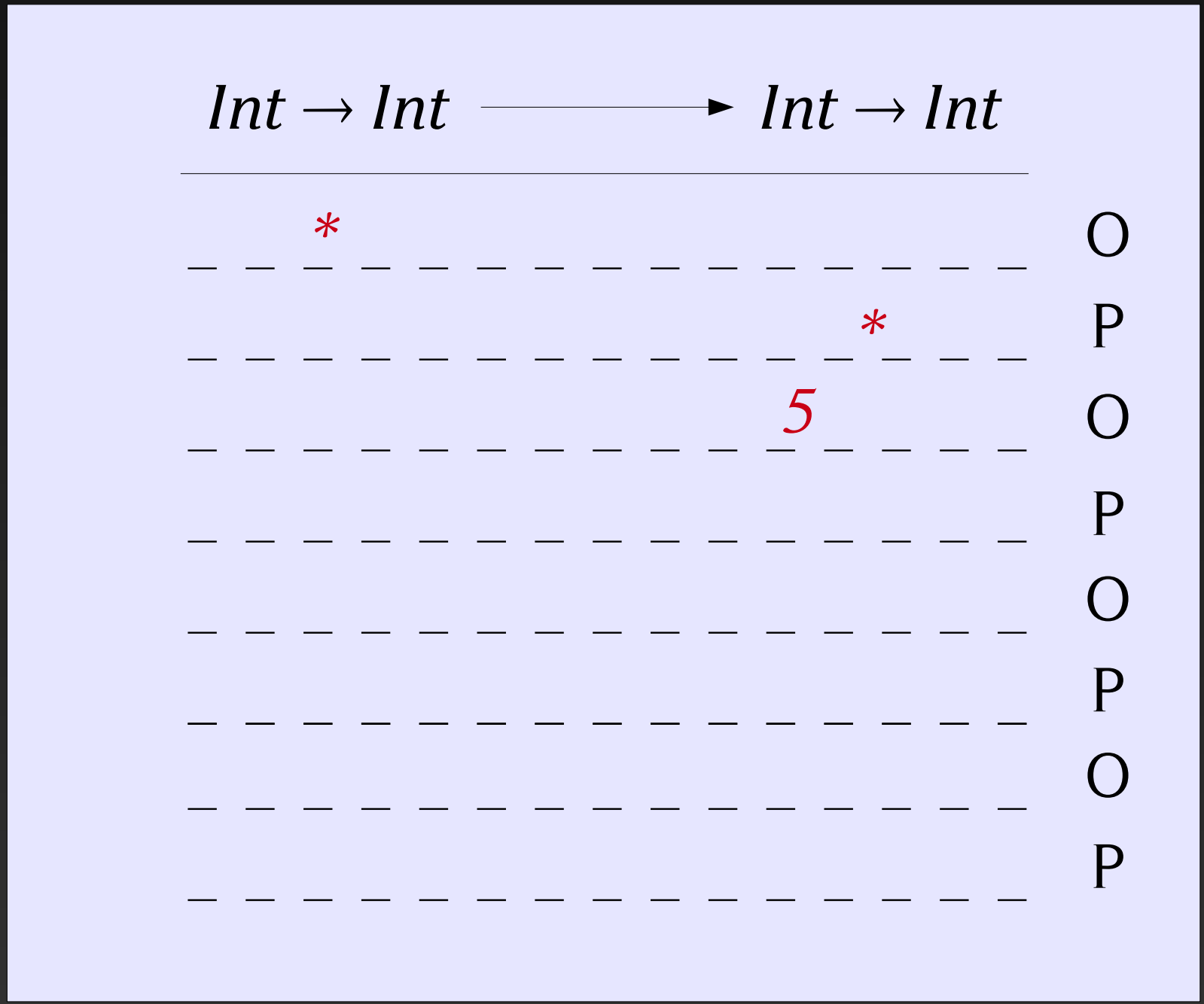
Example



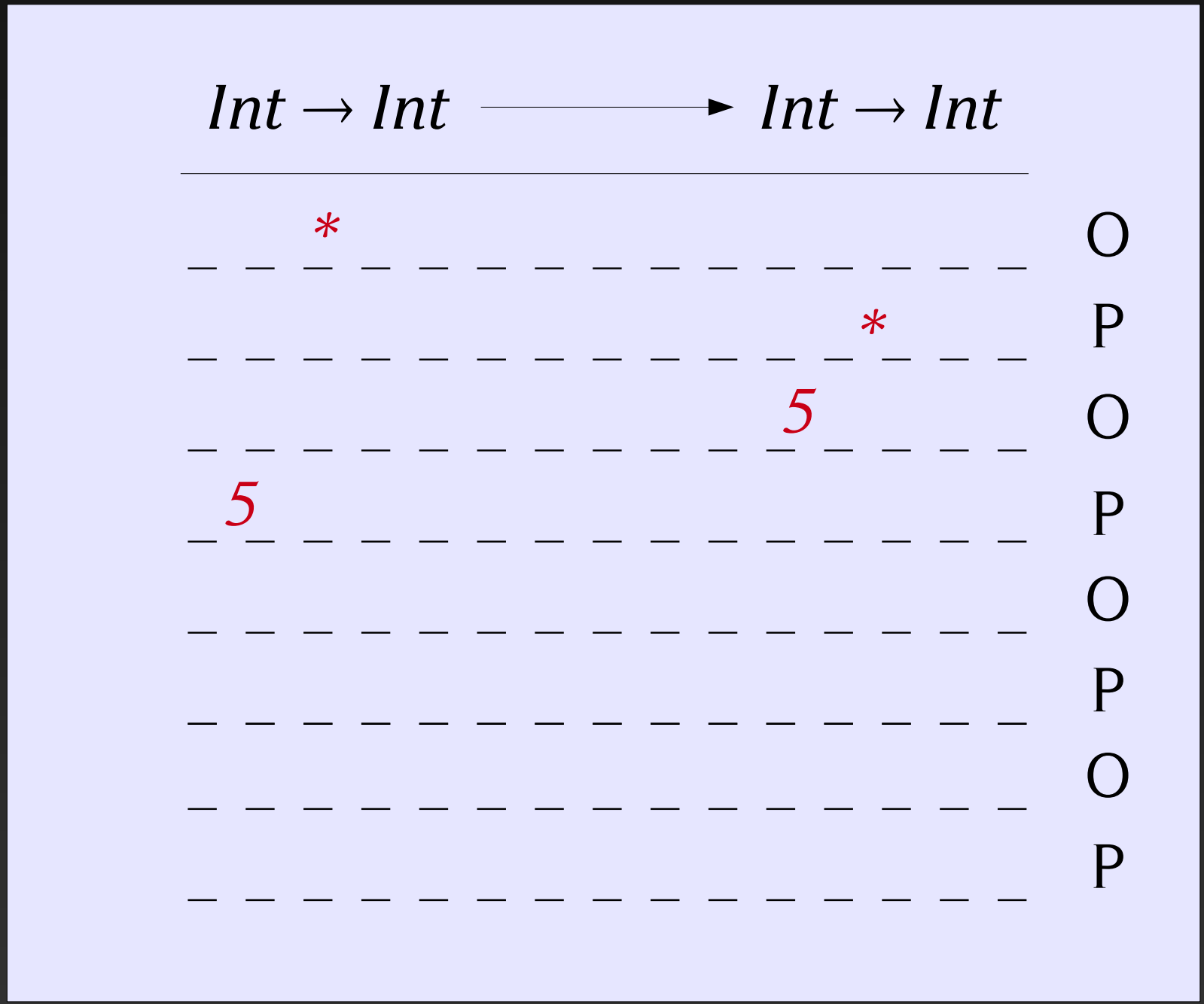
Example



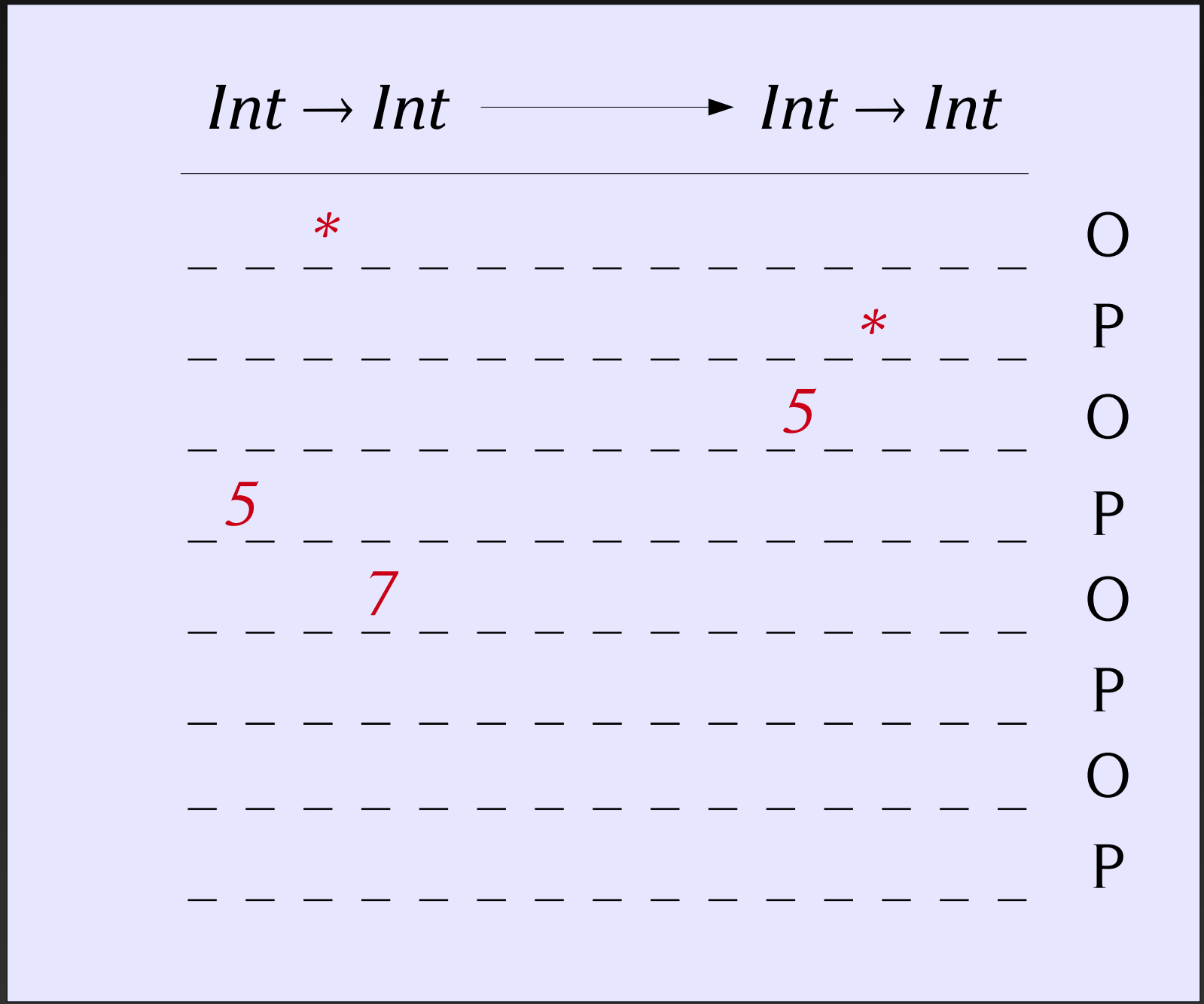
Example



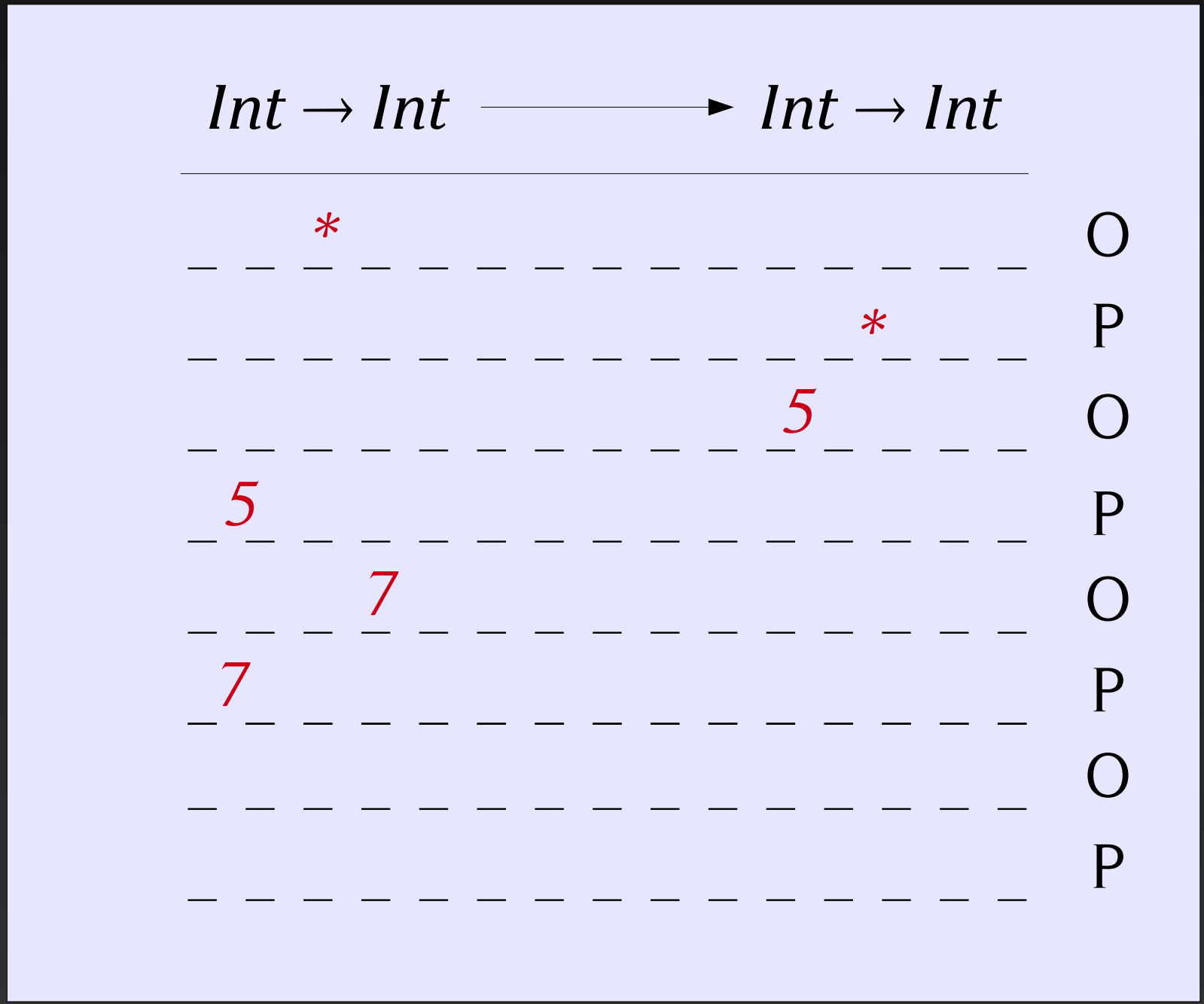
Example



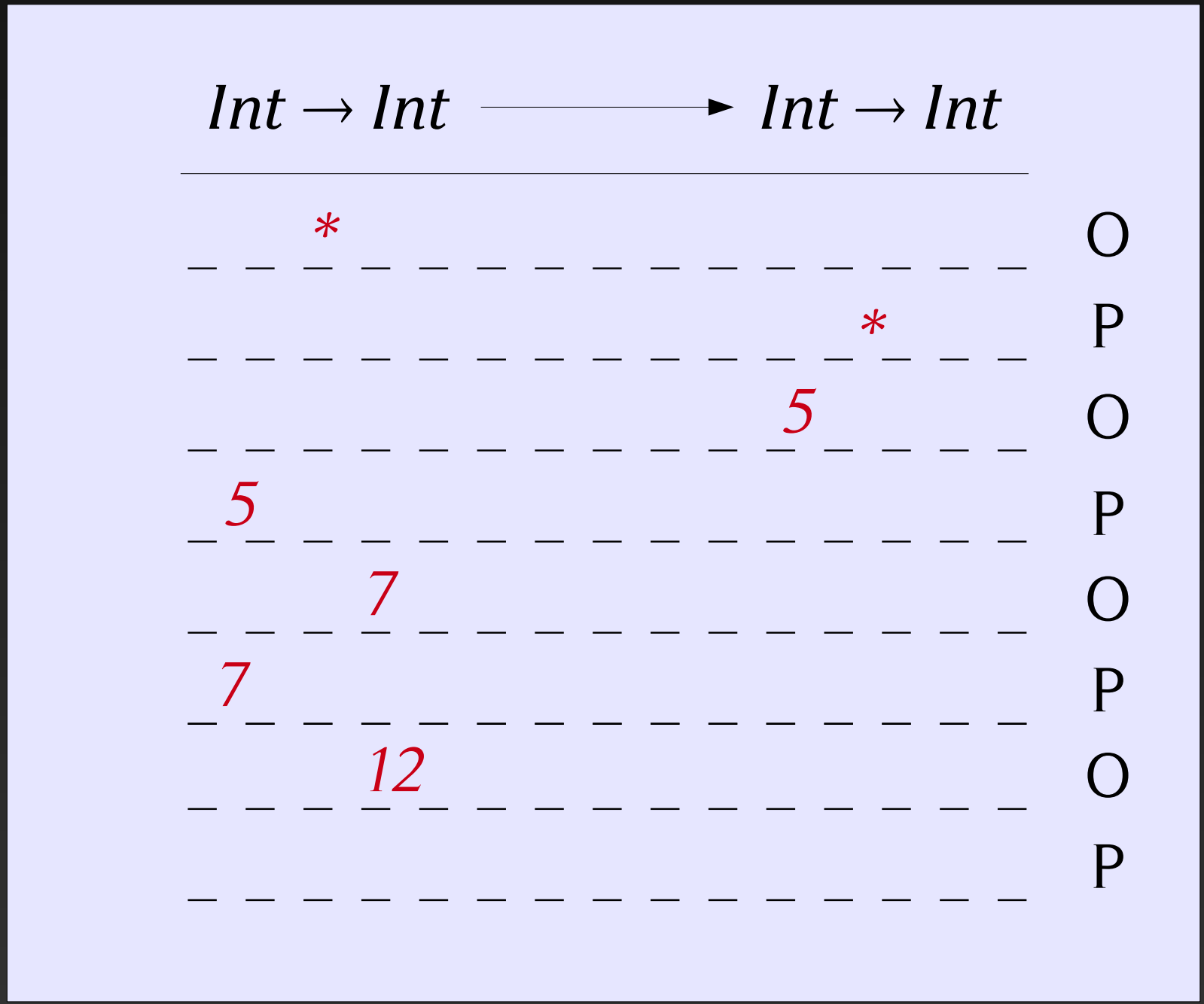
Example



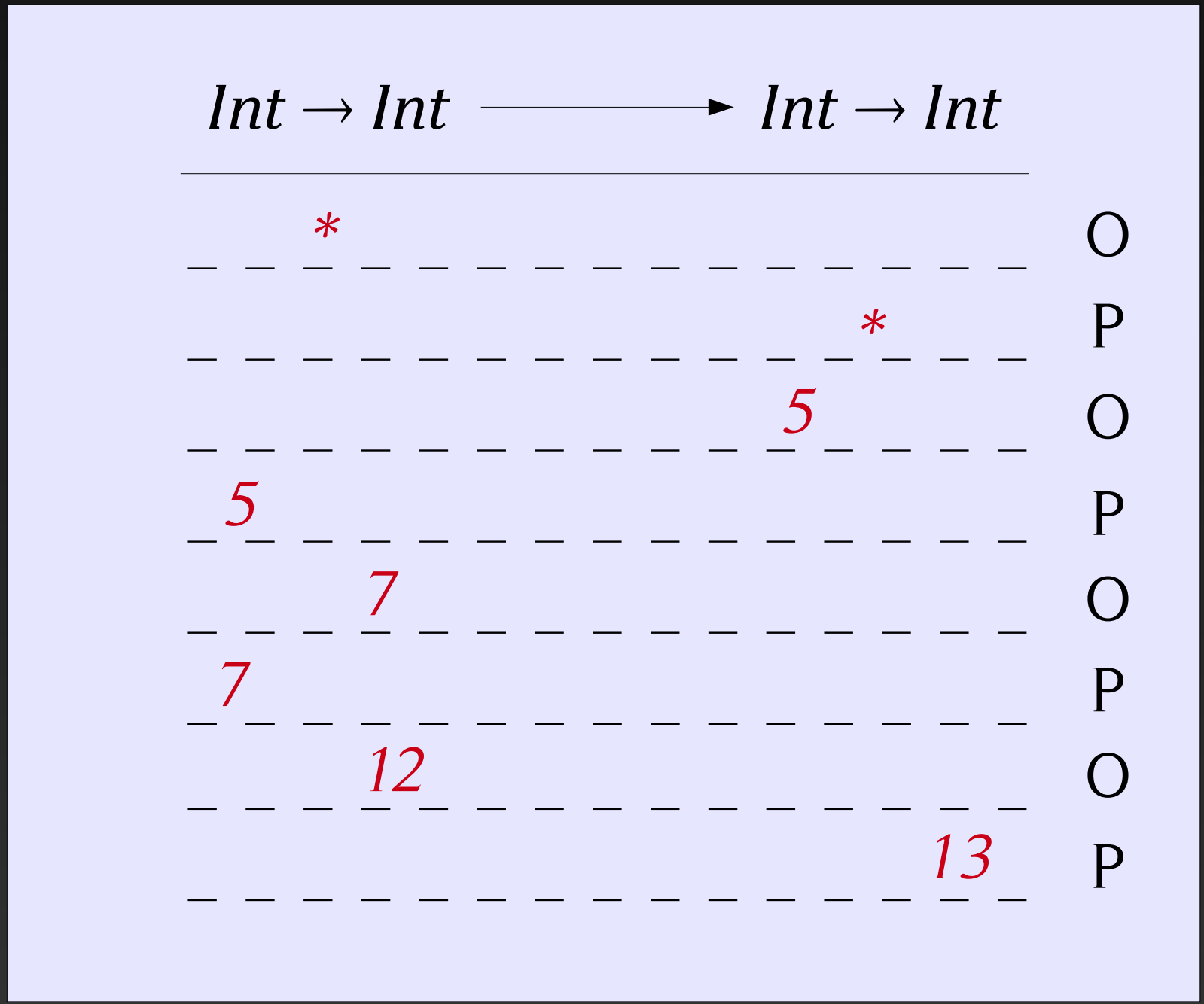
Example



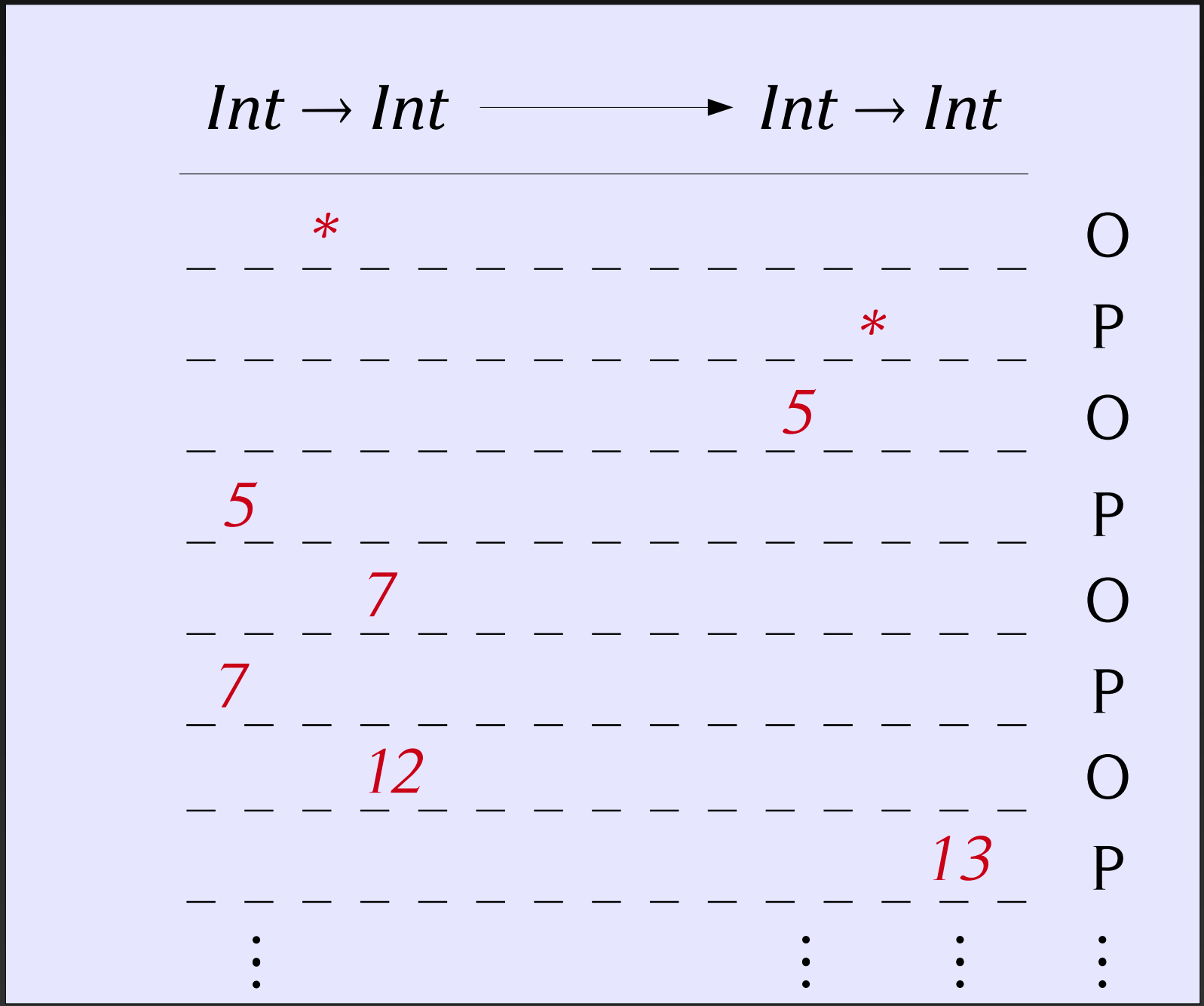
Example



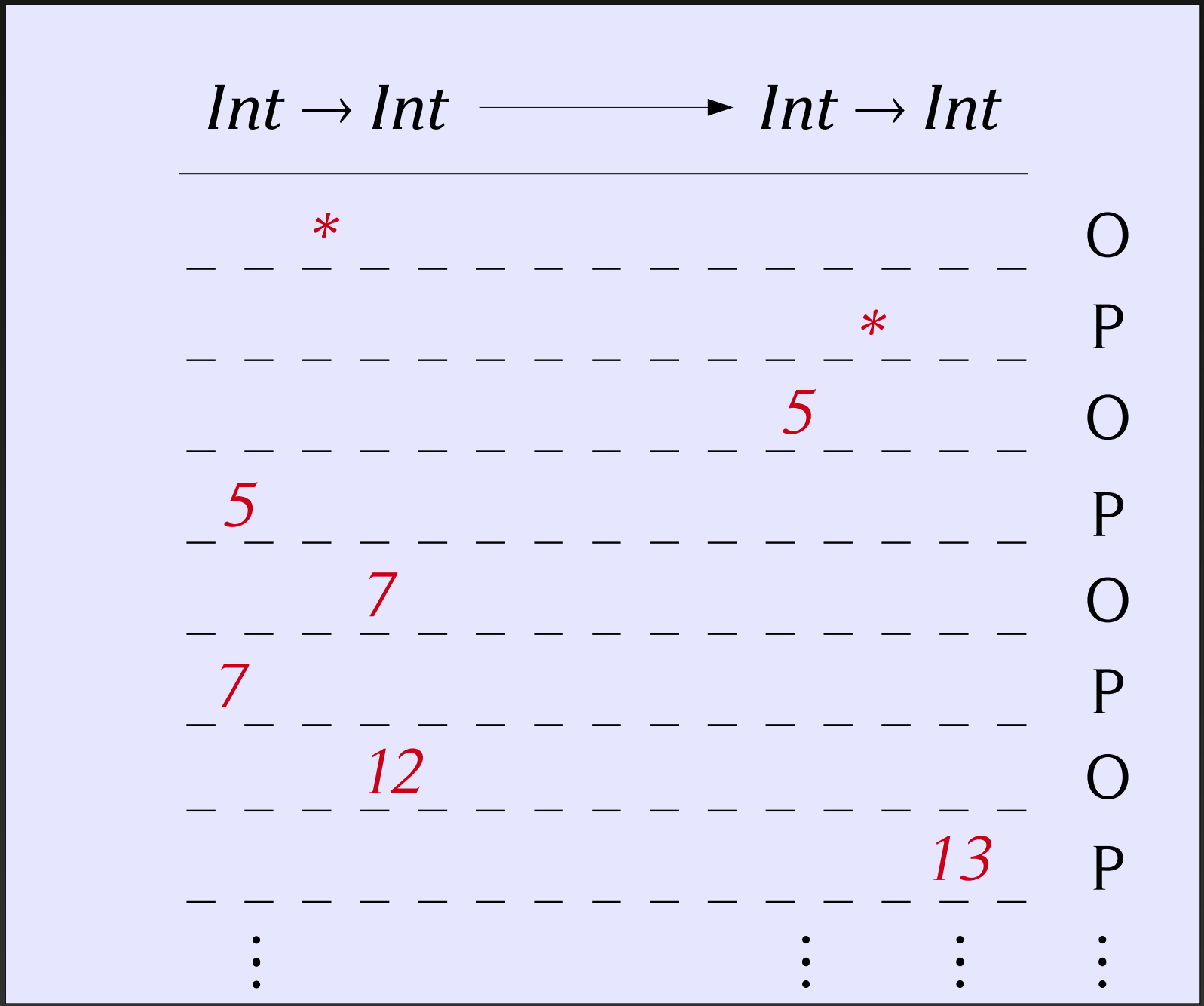
Example



Example



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



Game Semantics

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

Composition

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

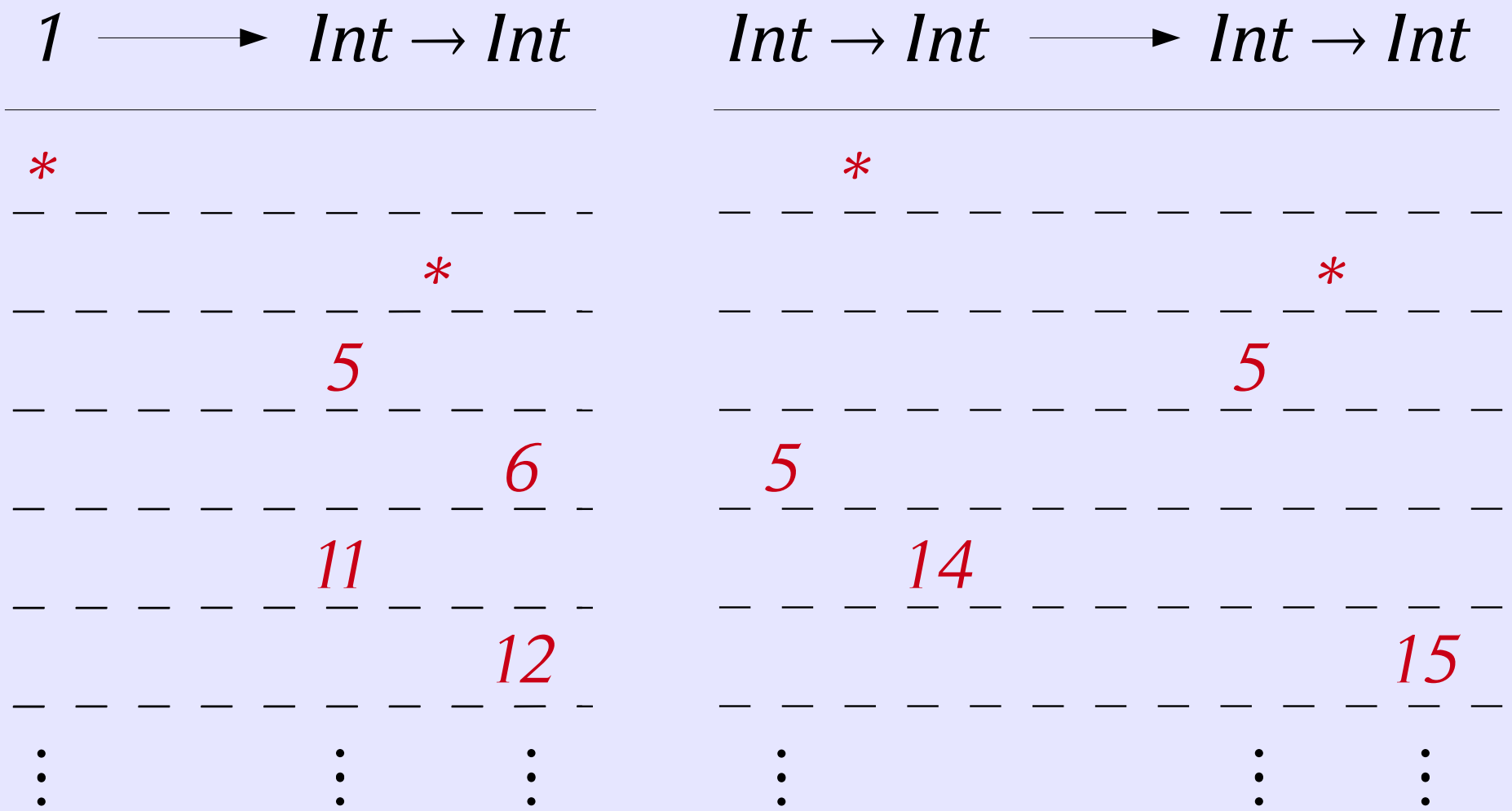
$f : \text{int} \to \text{int} \vdash \lambda x.f(x)+1 : \text{int} \to \text{int}$

Composition

`⊢ λx.x+1 : int → int`

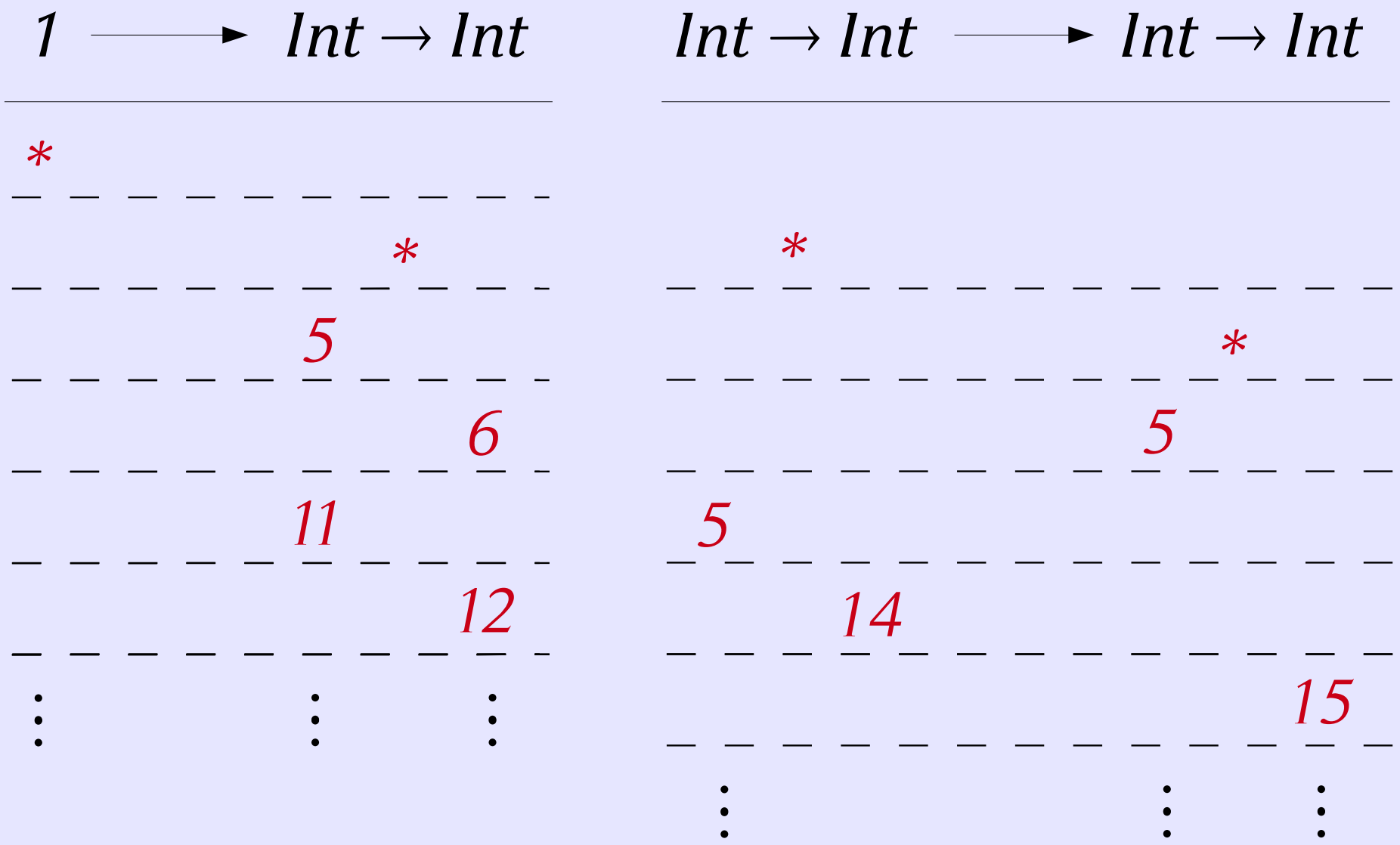
`f : int → int ⊢ λx.f(x)+1 : int → int`

Composition



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

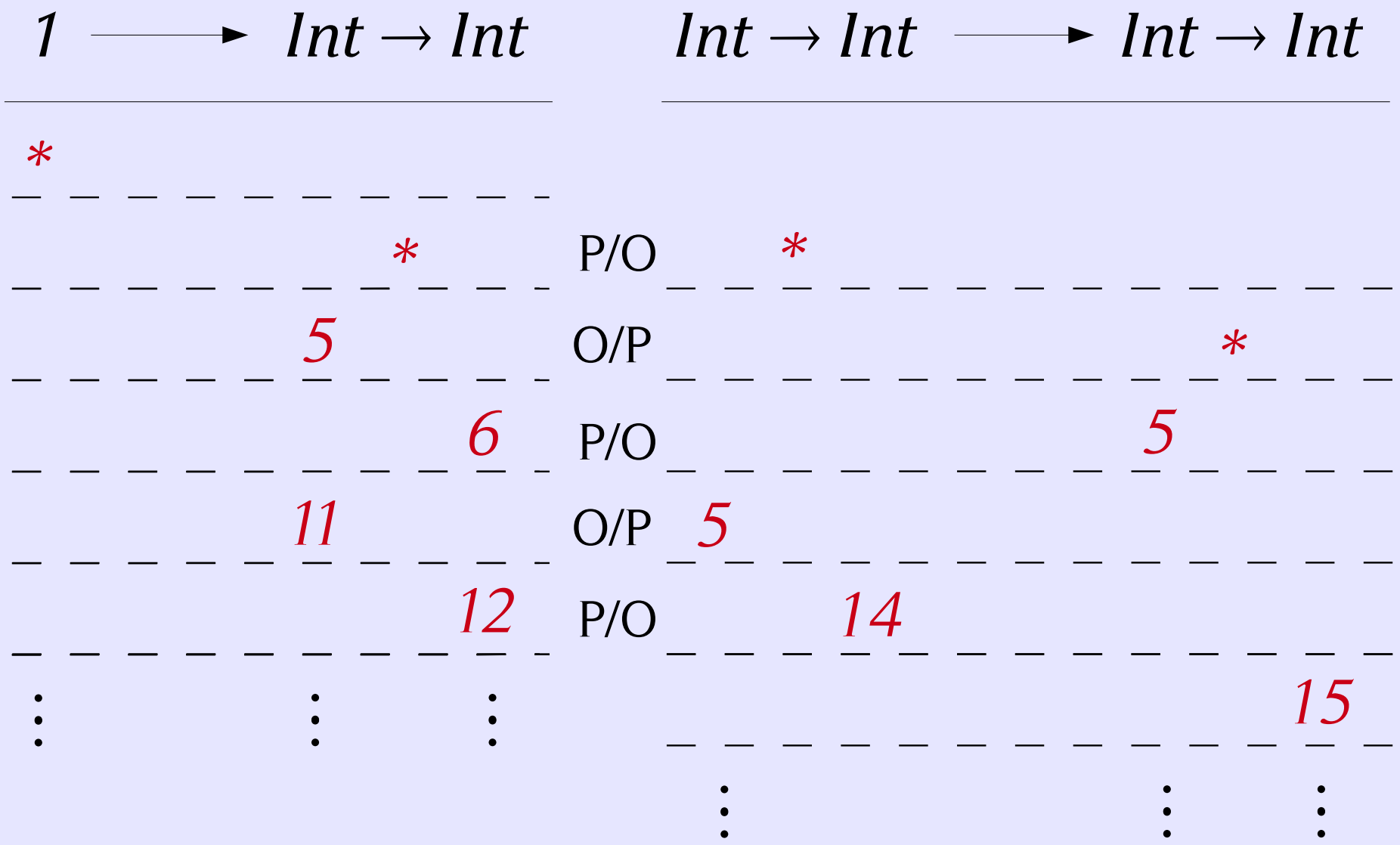
Composition



⊢ λx.x+1 : int → int

f : int → int ⊢ λx.f(x)+1 : int → int

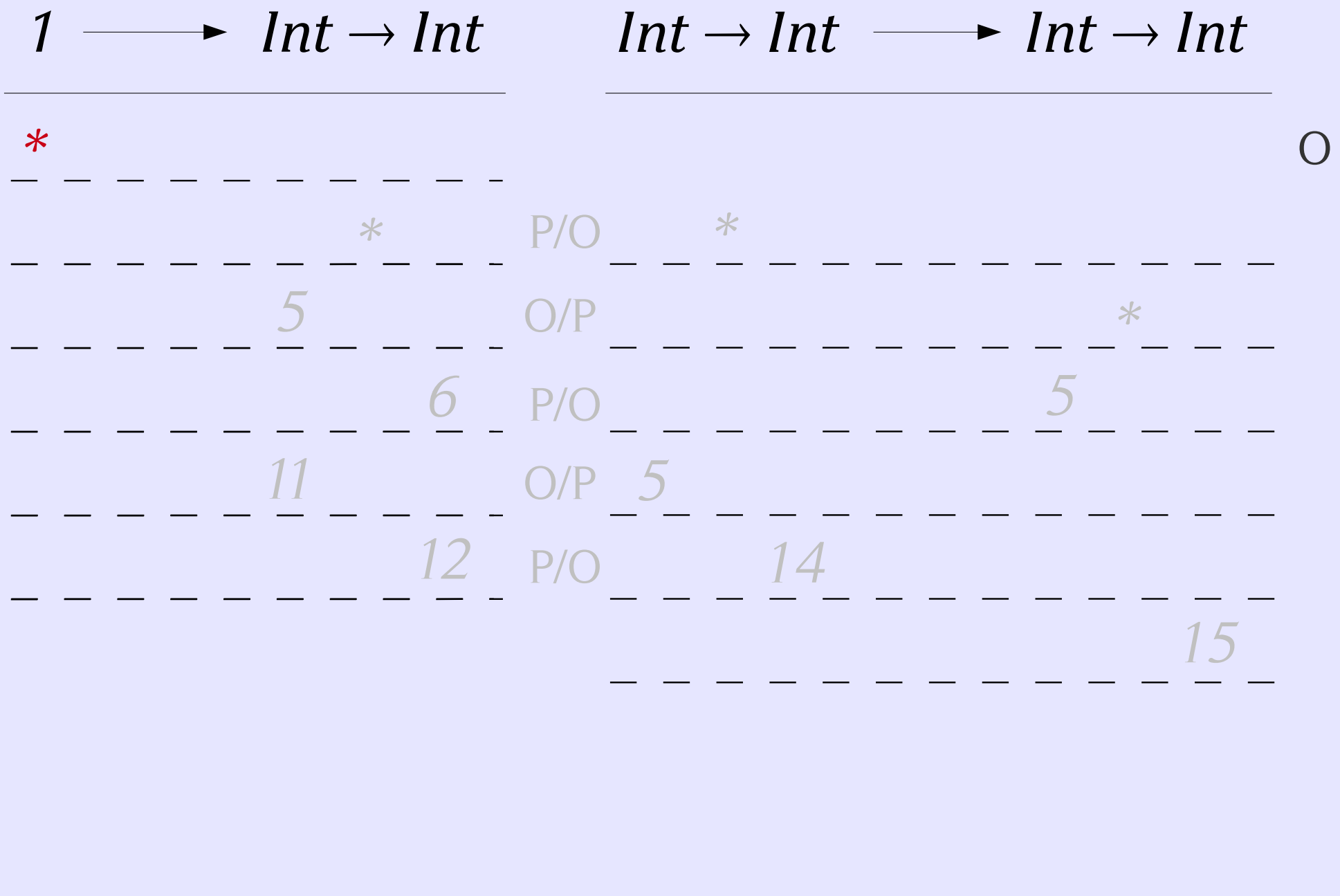
Composition



⊢ λx.x+1 : int → int

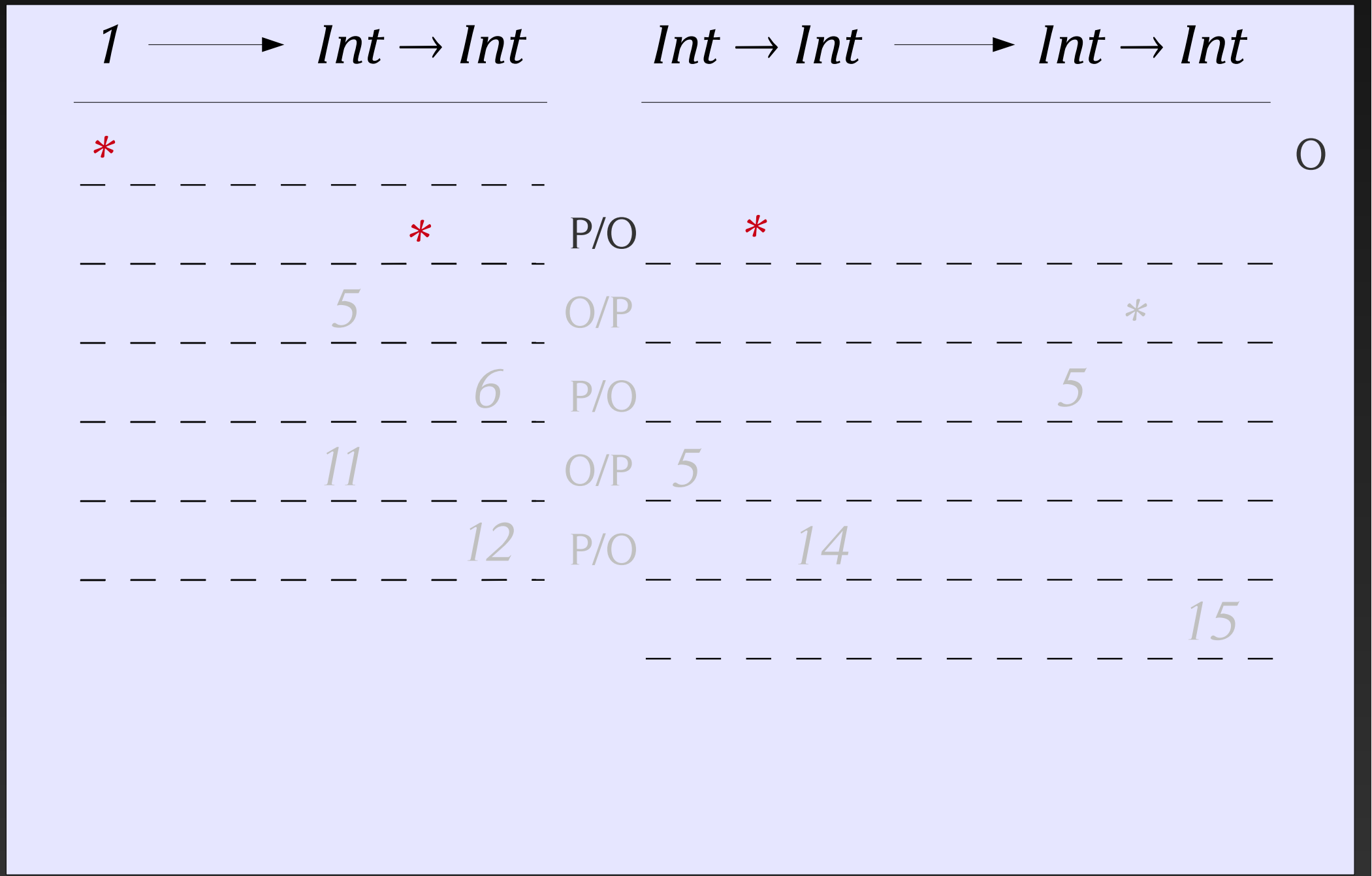
f : int → int ⊢ λx.f(x)+1 : int → int

Composition



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

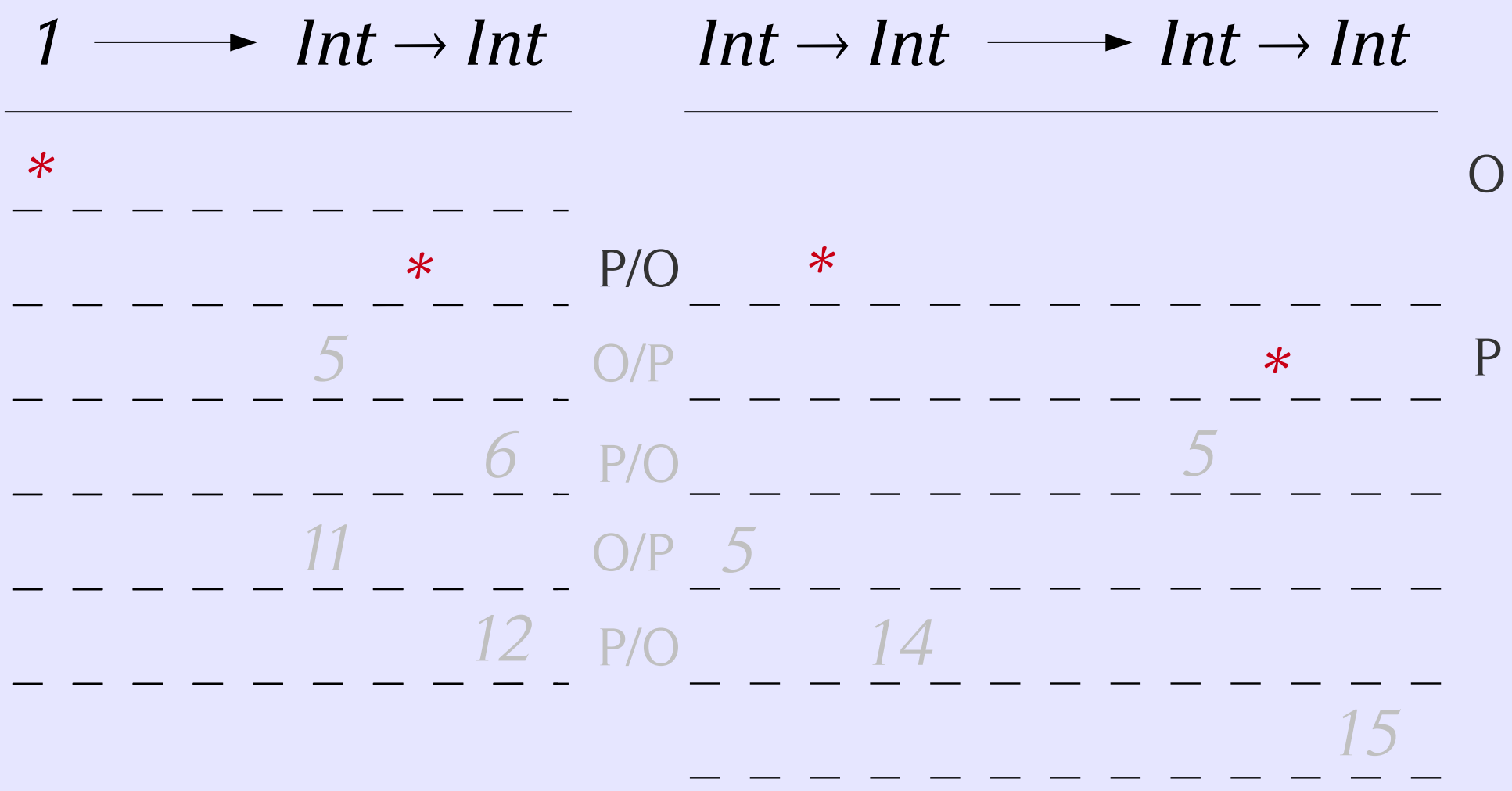
Composition



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

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

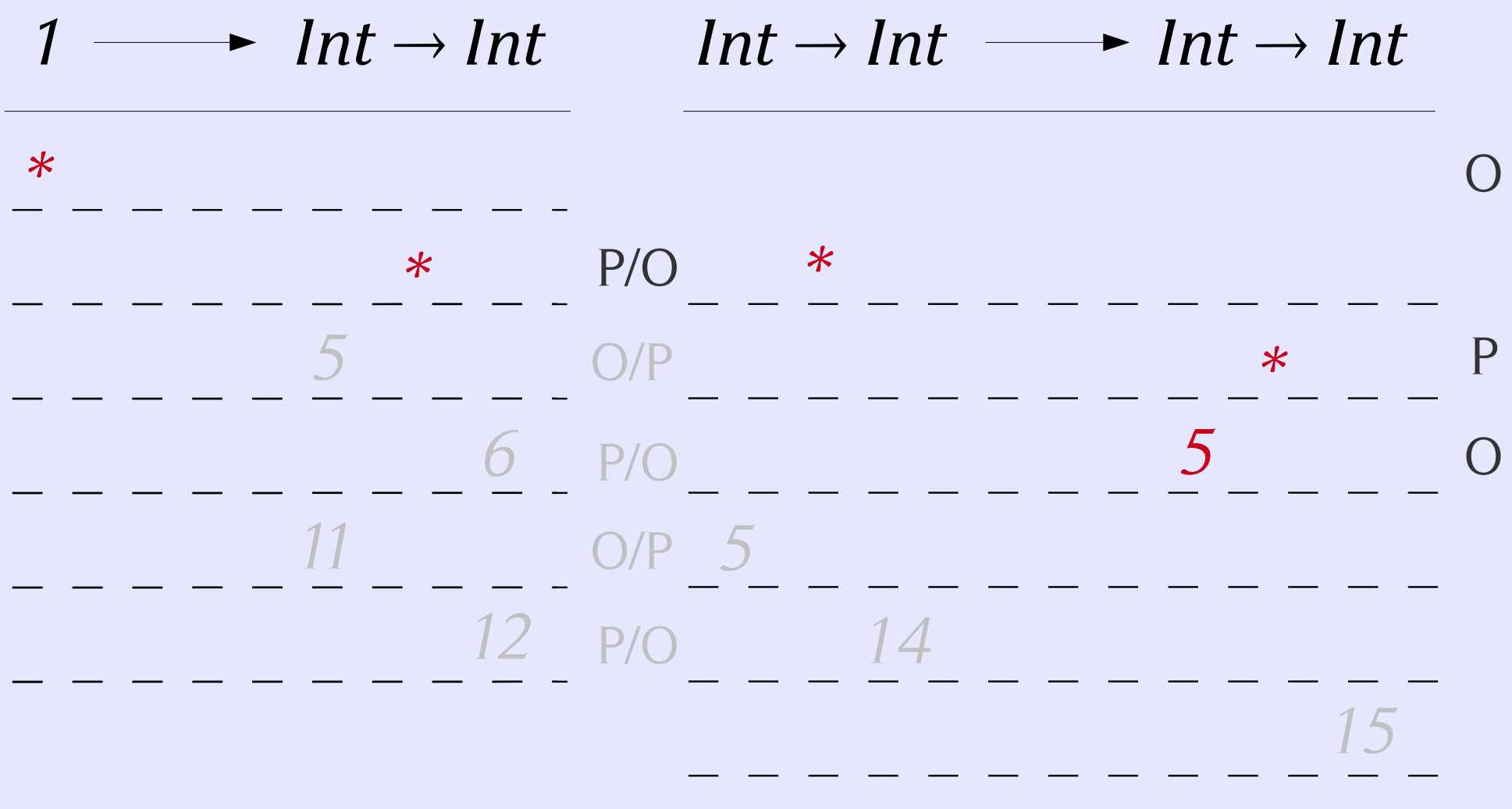
Composition



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

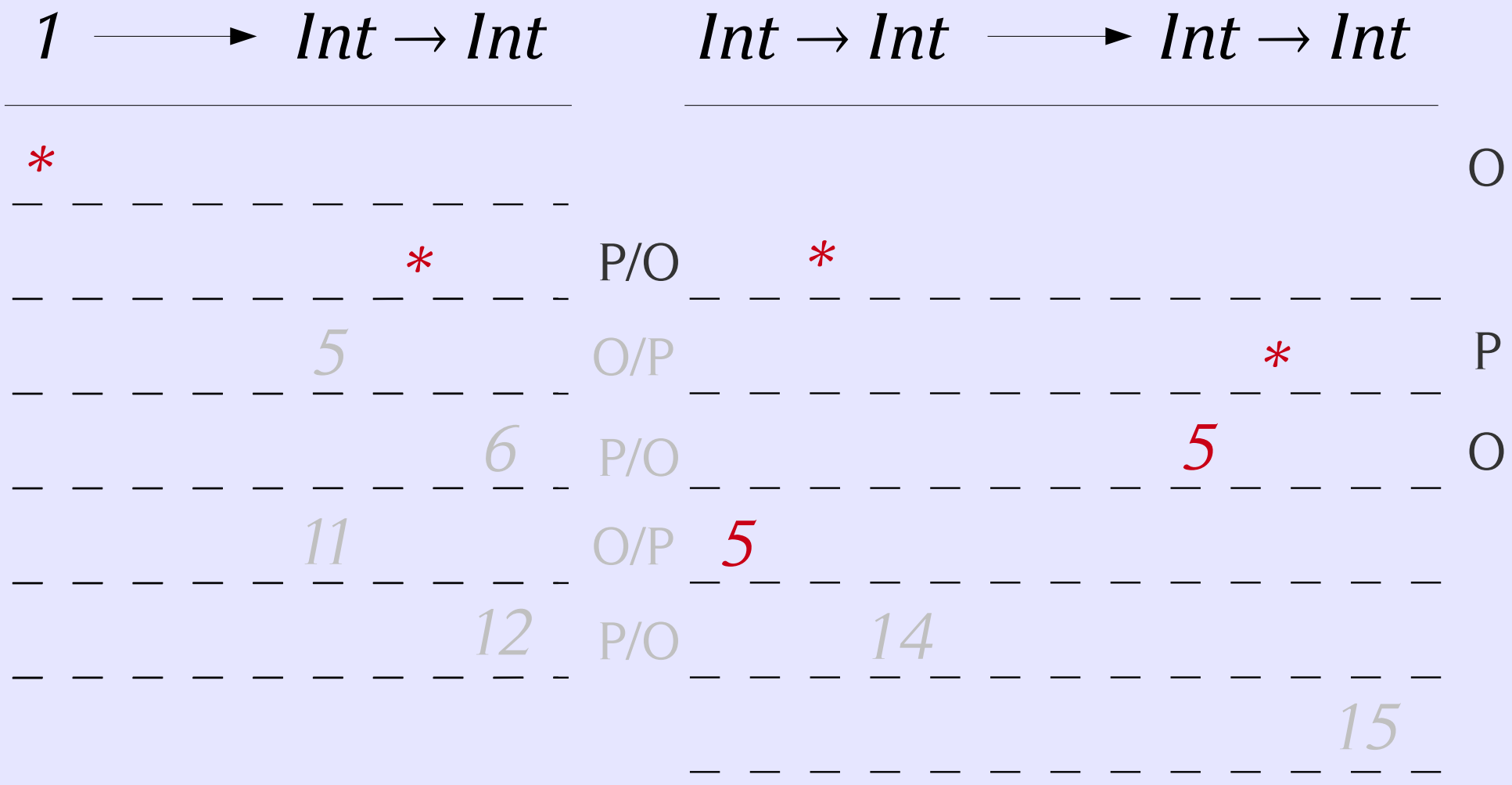
$f : \mathtt{int} \to \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \to \mathtt{int}$

Composition



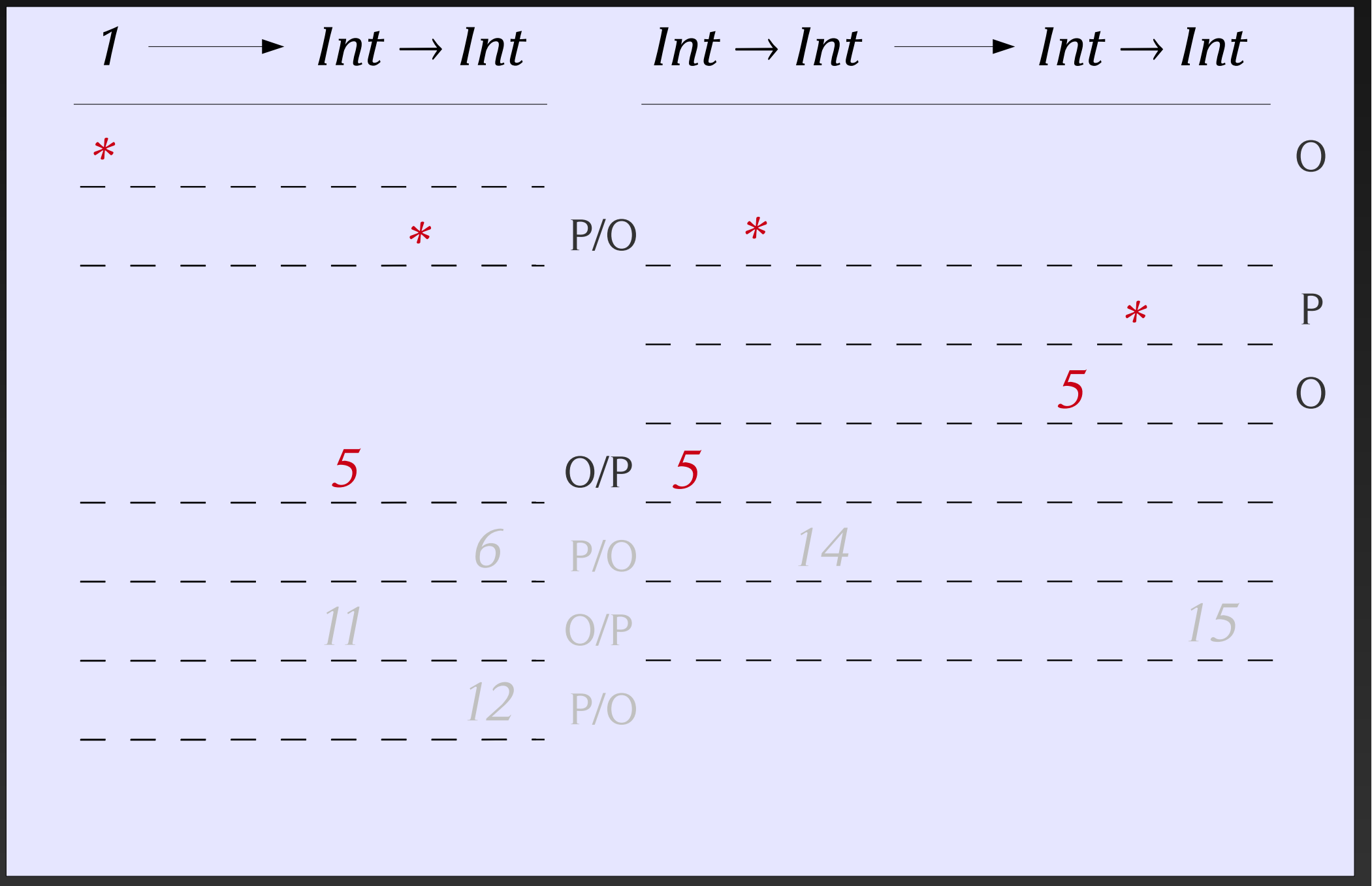
$\vdash \lambda x.x+1 : int \to int$ | $f : int \to int \vdash \lambda x.f(x)+1 : int \to int$

Composition



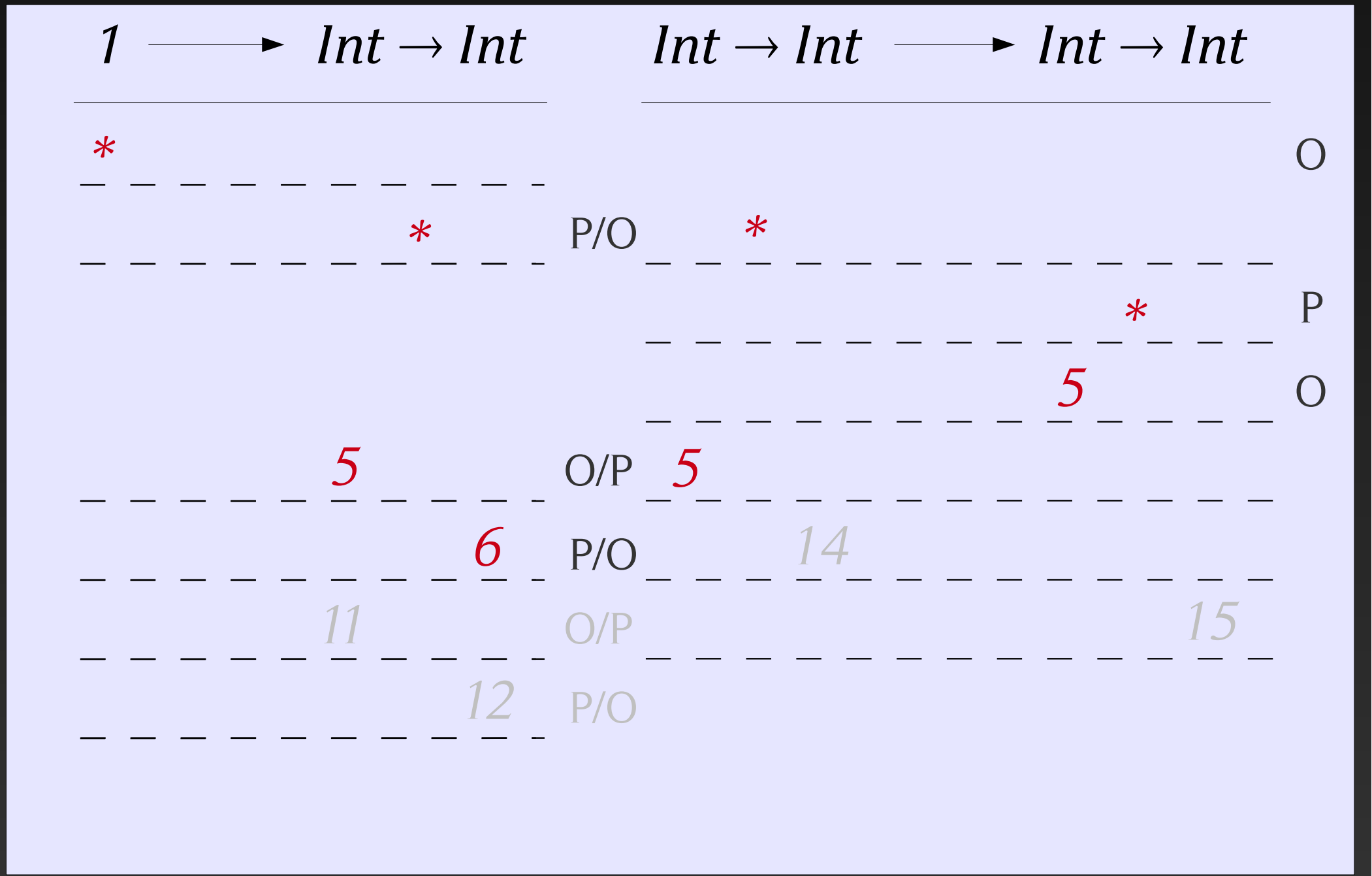
$\vdash \lambda x.x+1 : int \to int$ | $f : int \to int \vdash \lambda x.f(x)+1 : int \to int$

Composition



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

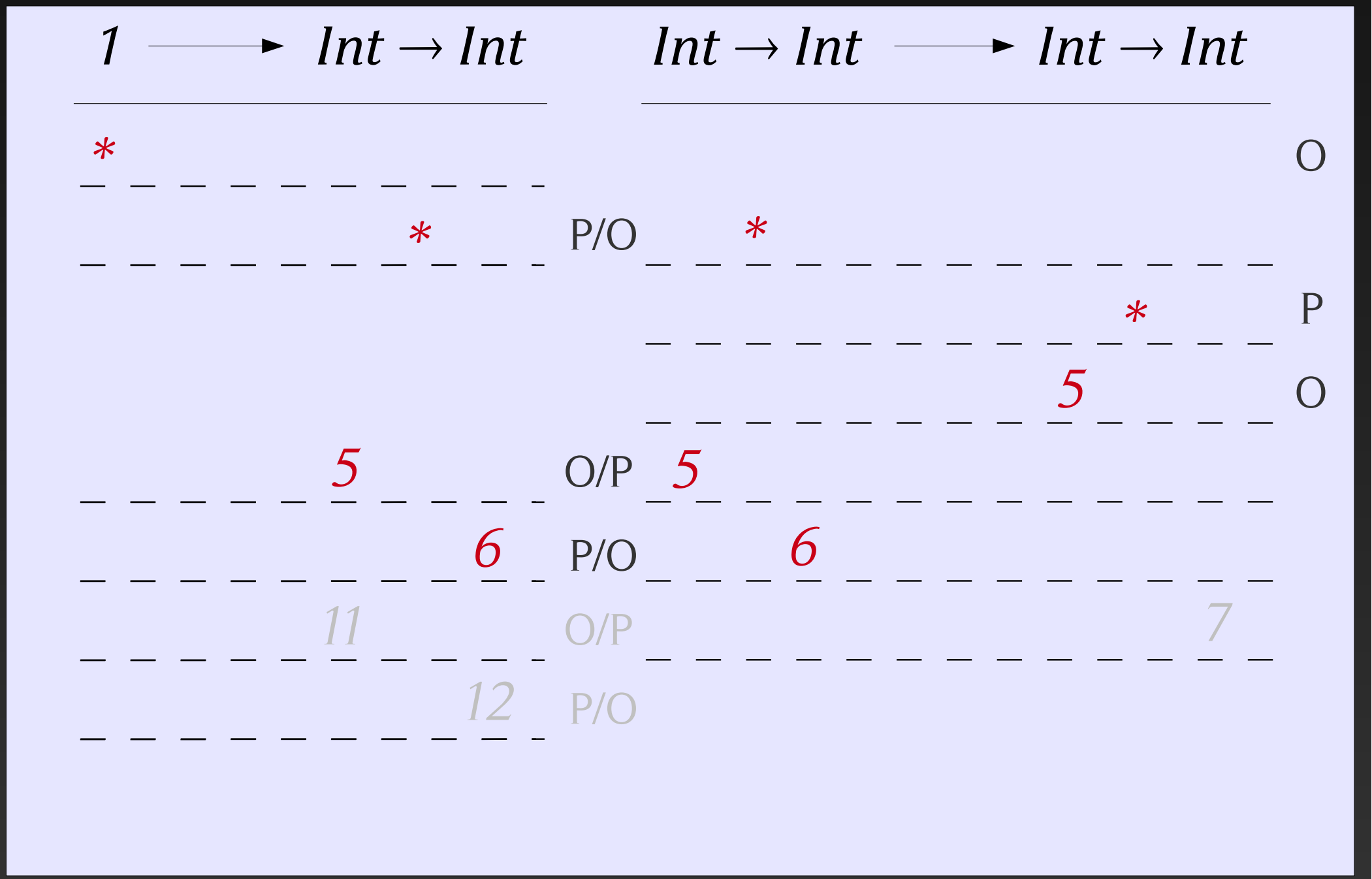
Composition



⊢ λx.x+1 : int → int

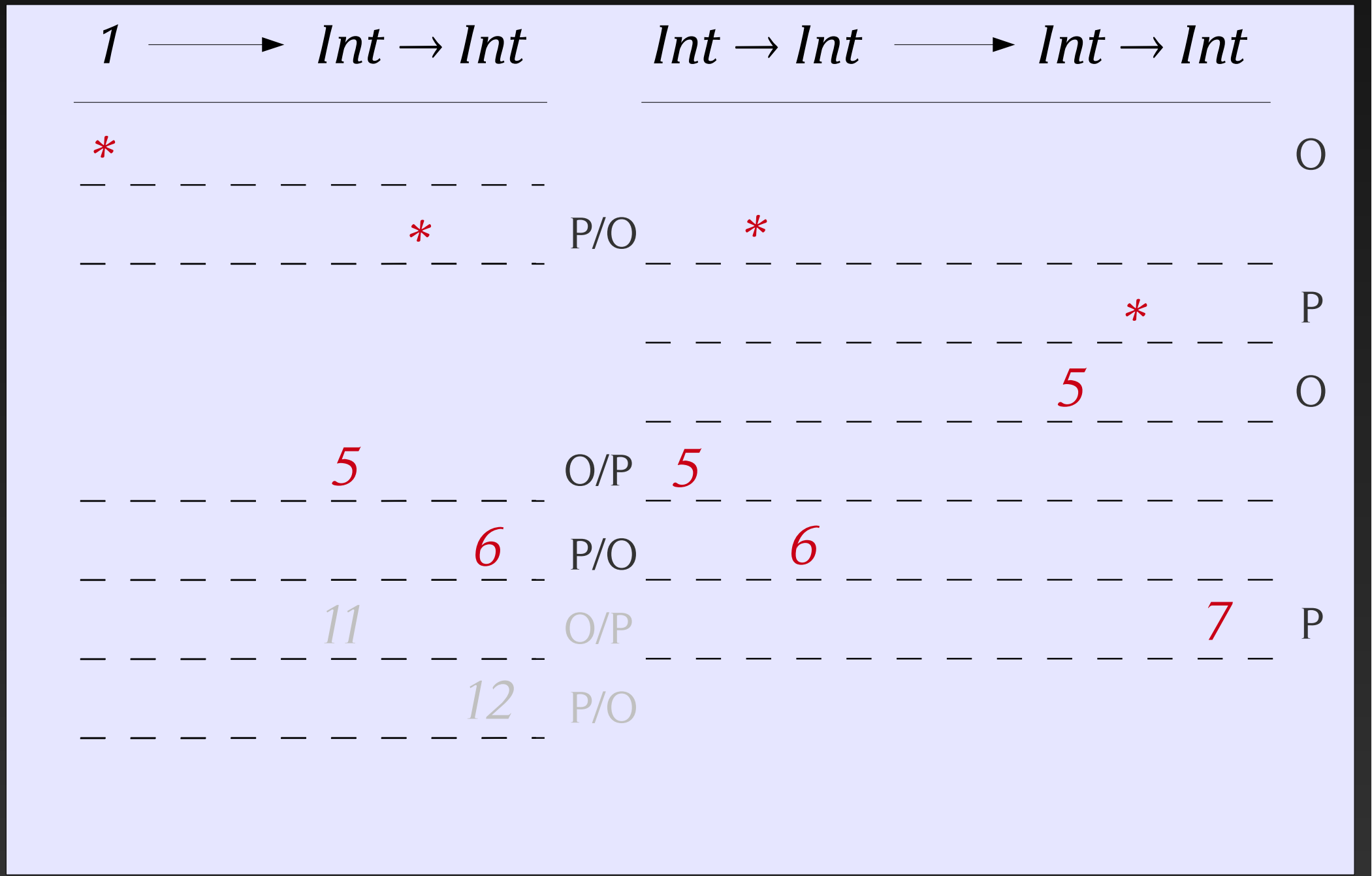
f : int → int ⊢ λx.f(x)+1 : int → int

Composition



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

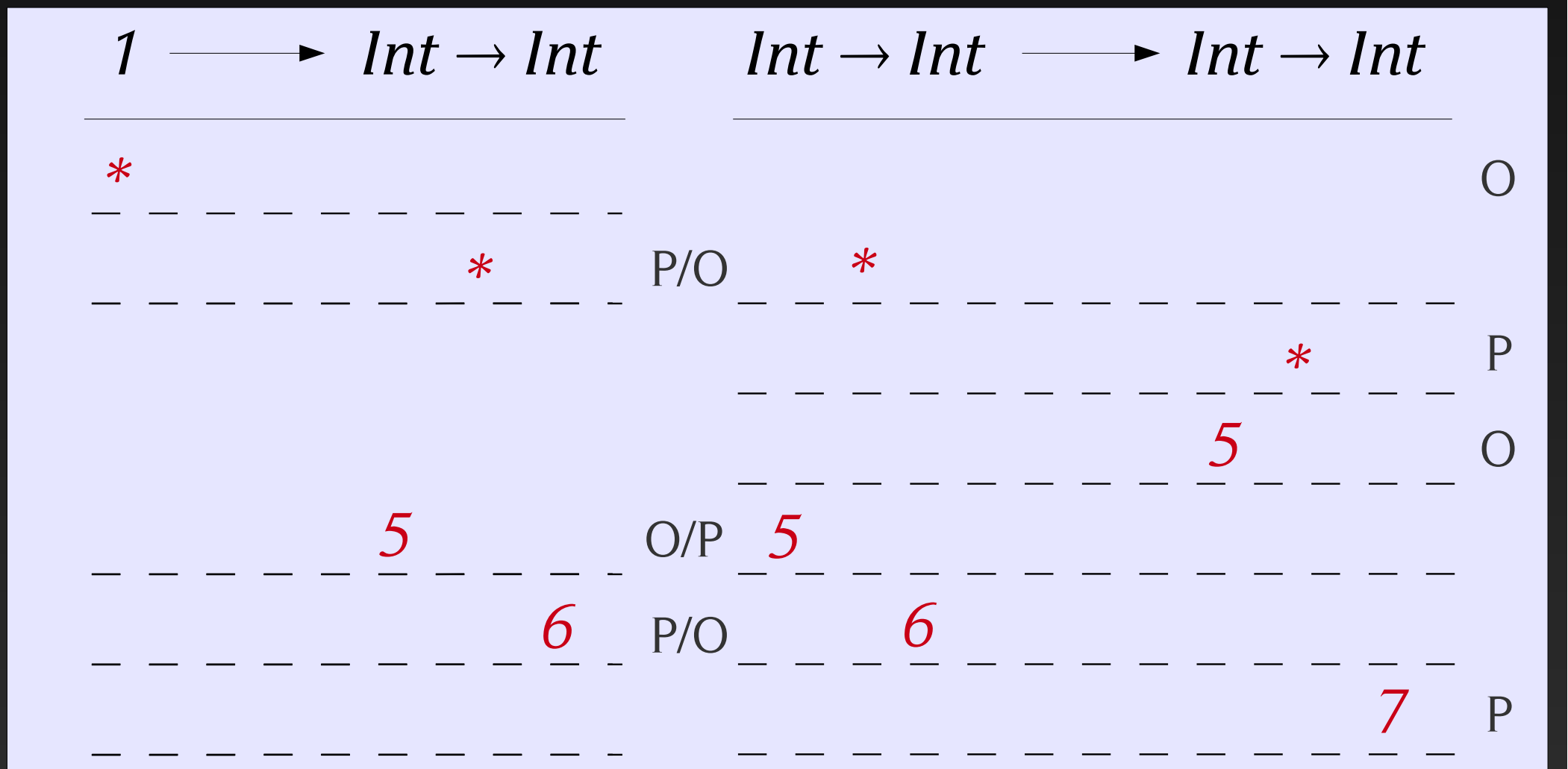
Composition



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

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

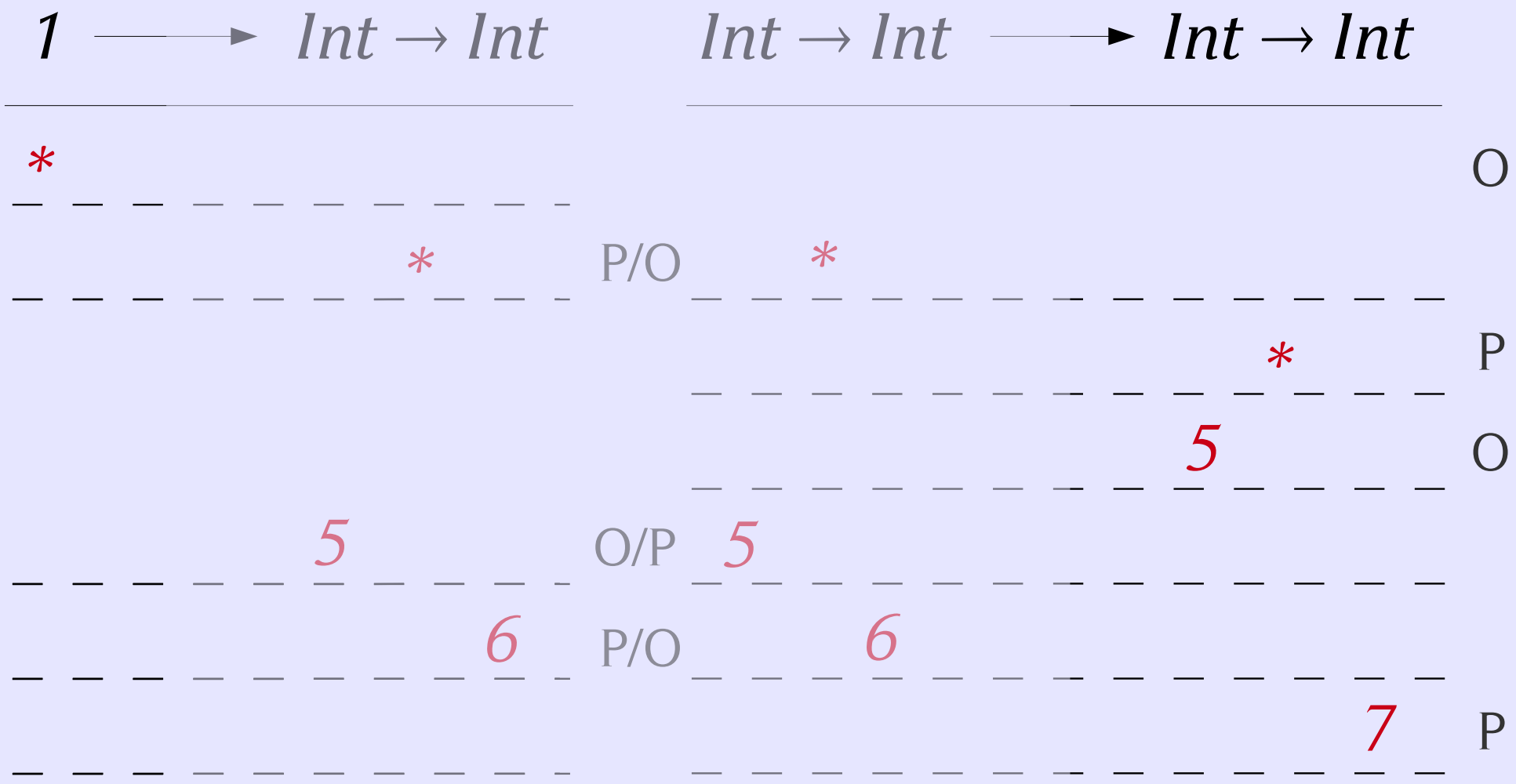
Composition



⊢ λx.x+1 : int → int

f : int → int ⊢ λx.f(x)+1 : int → int

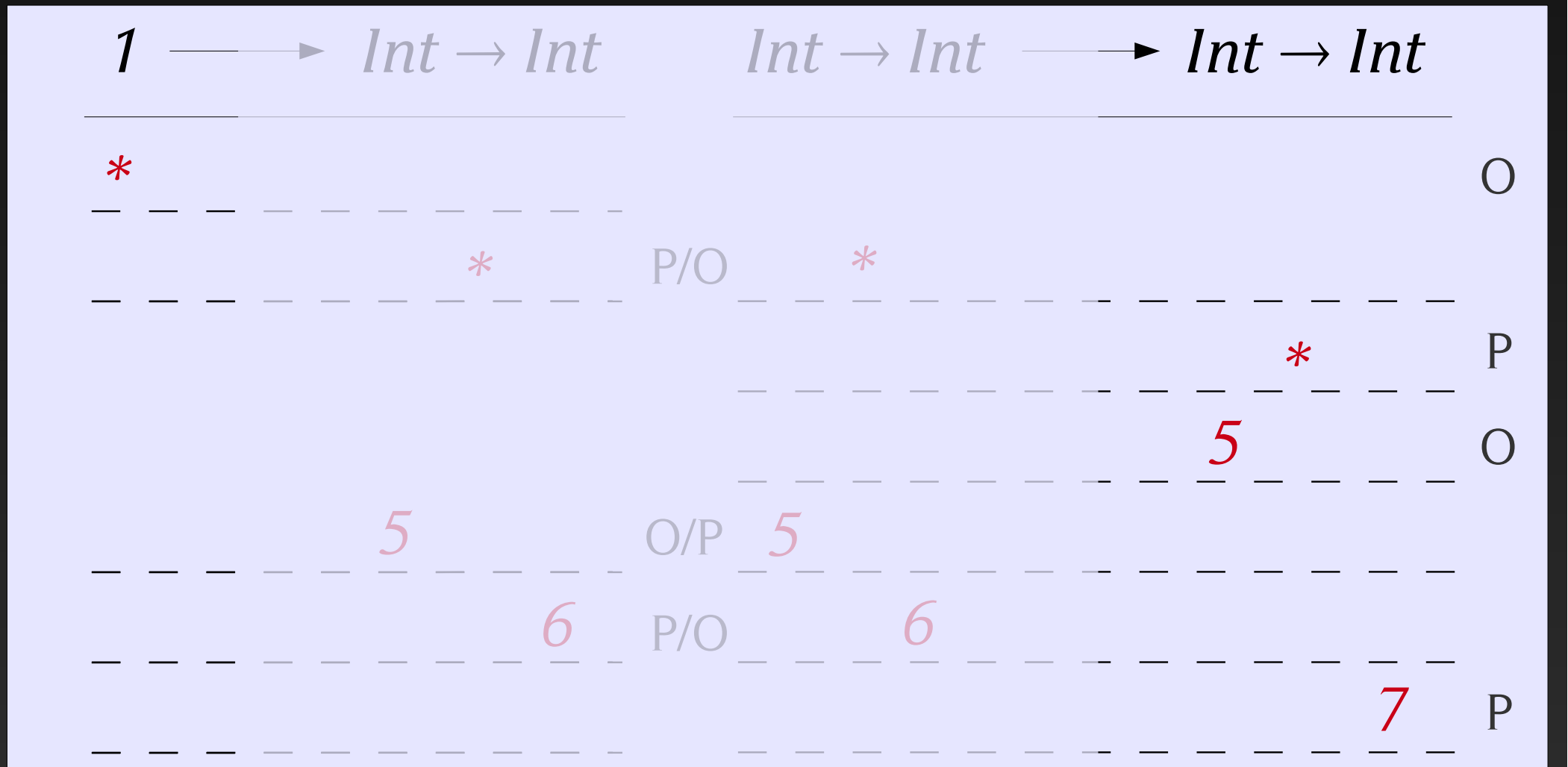
Composition



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

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

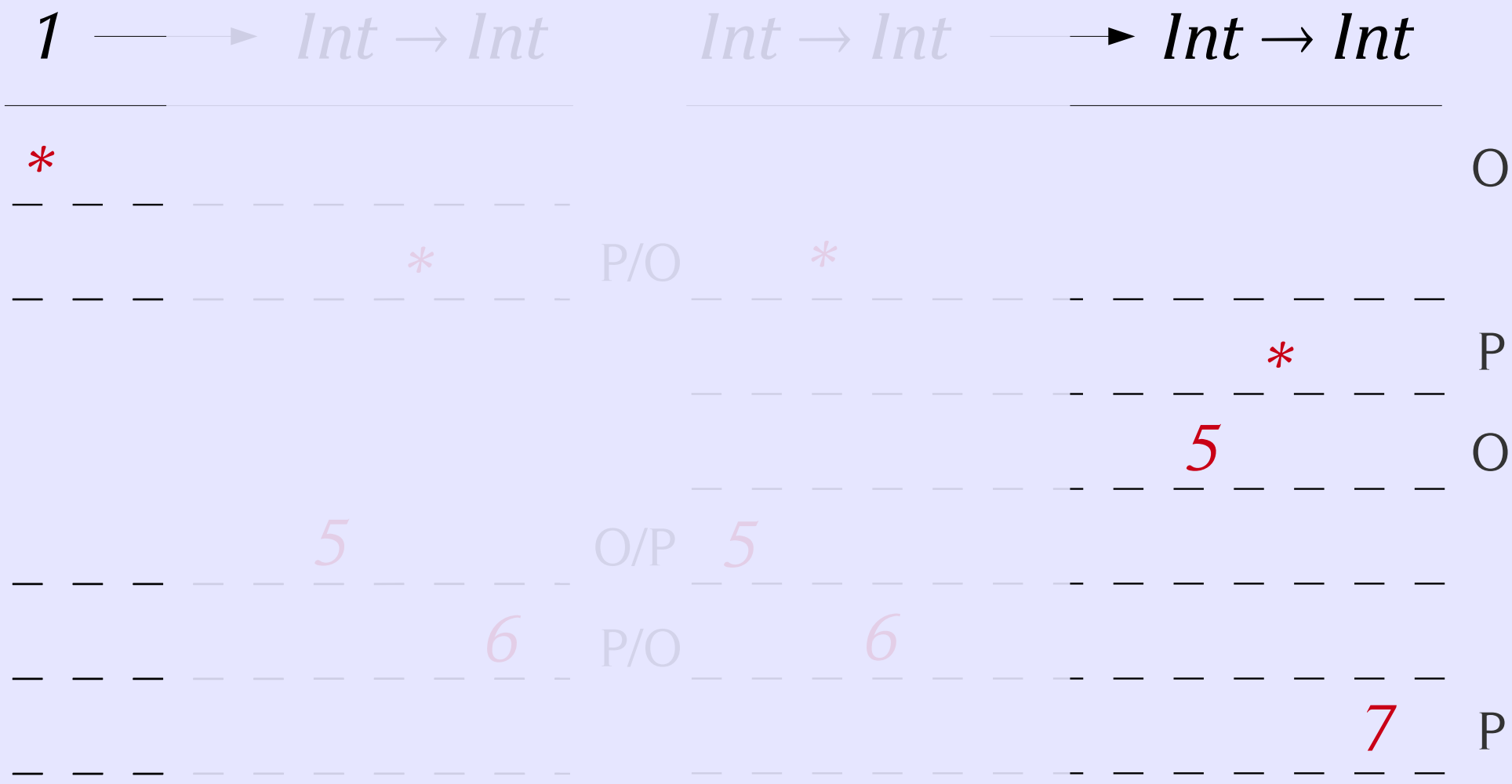
Composition



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

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

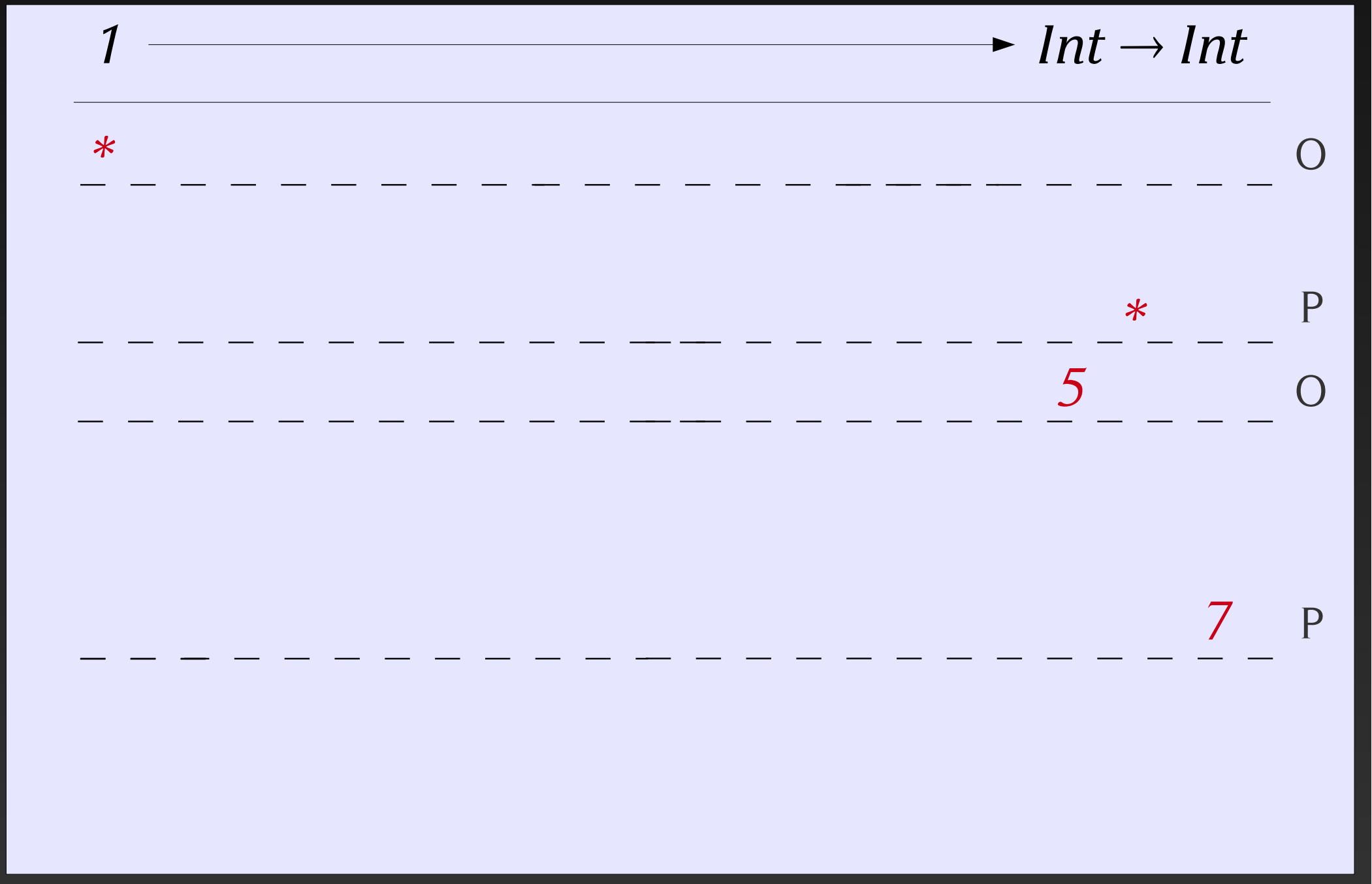
Composition



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

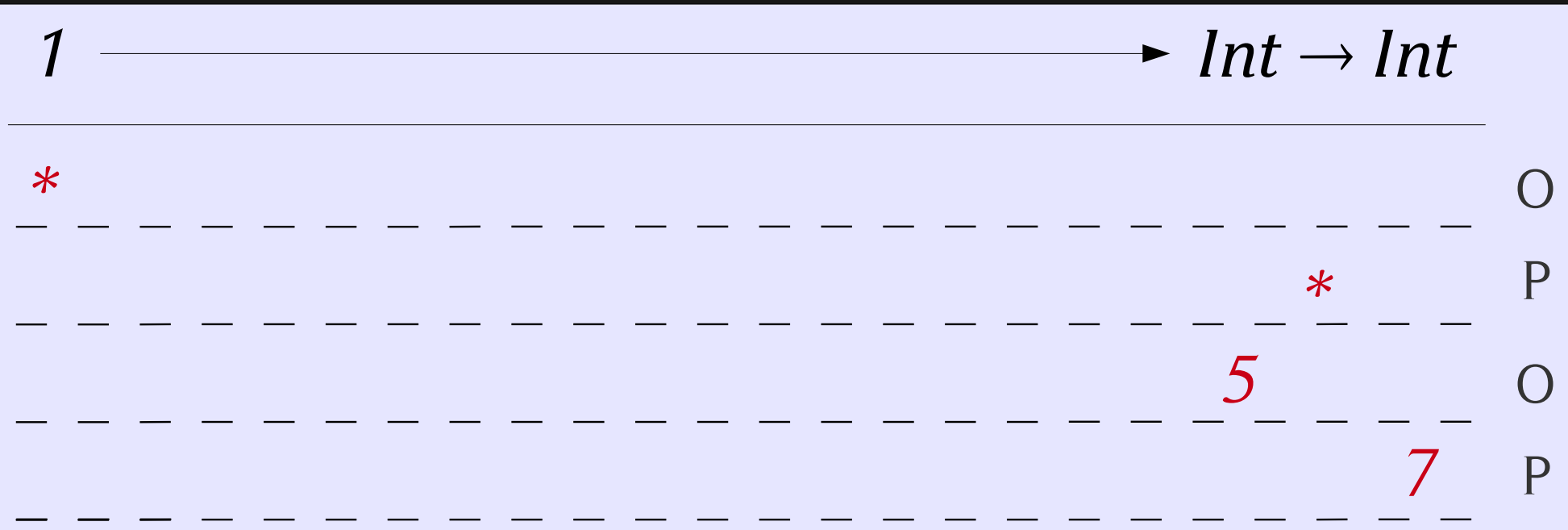
$f : \texttt{int} \to \texttt{int} \vdash \lambda x.f(x)+1 : \texttt{int} \to \texttt{int}$

Composition



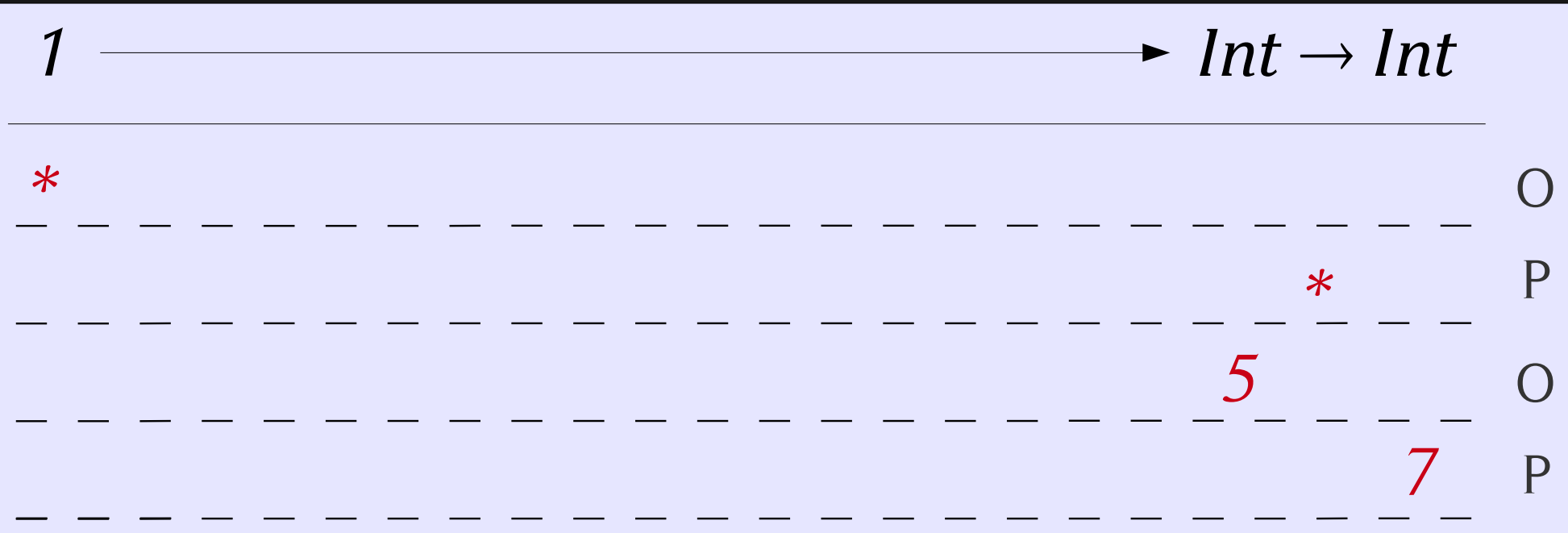
$\vdash \lambda x.x+1 : int \to int$ $\quad f : int \to int \vdash \lambda x.f(x)+1 : int \to int$

Composition



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

Composition



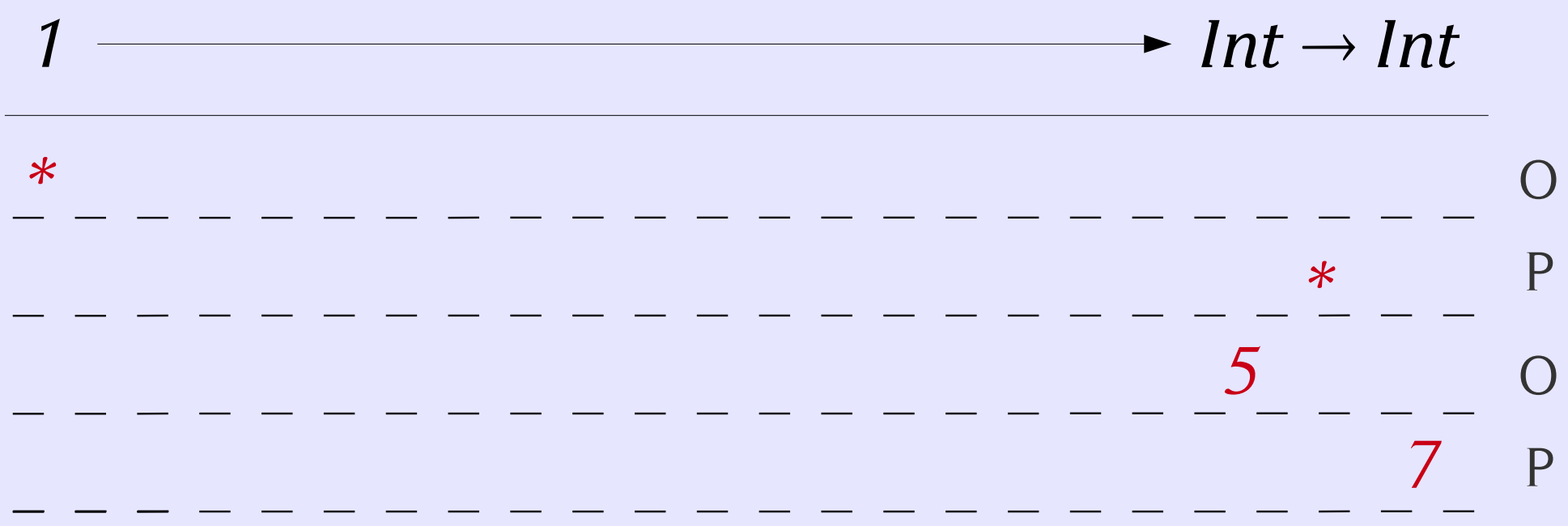
$\vdash \lambda x.x+1 : int \longrightarrow int$; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

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

⊢ λx.x+1 : int → int

f : int → int ⊢ λx.f(x)+1 : int → int

Composition



⊢ λx.x+1 : int → int ; f : int → int ⊢ λx.f(x)+1 : int → int

= ⊢ λx.x+2 : int → int

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

Game Semantics

- Computation is represented 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 games

$\lambda x.\, \texttt{ref}() : \texttt{unit} \to \texttt{unit ref}$

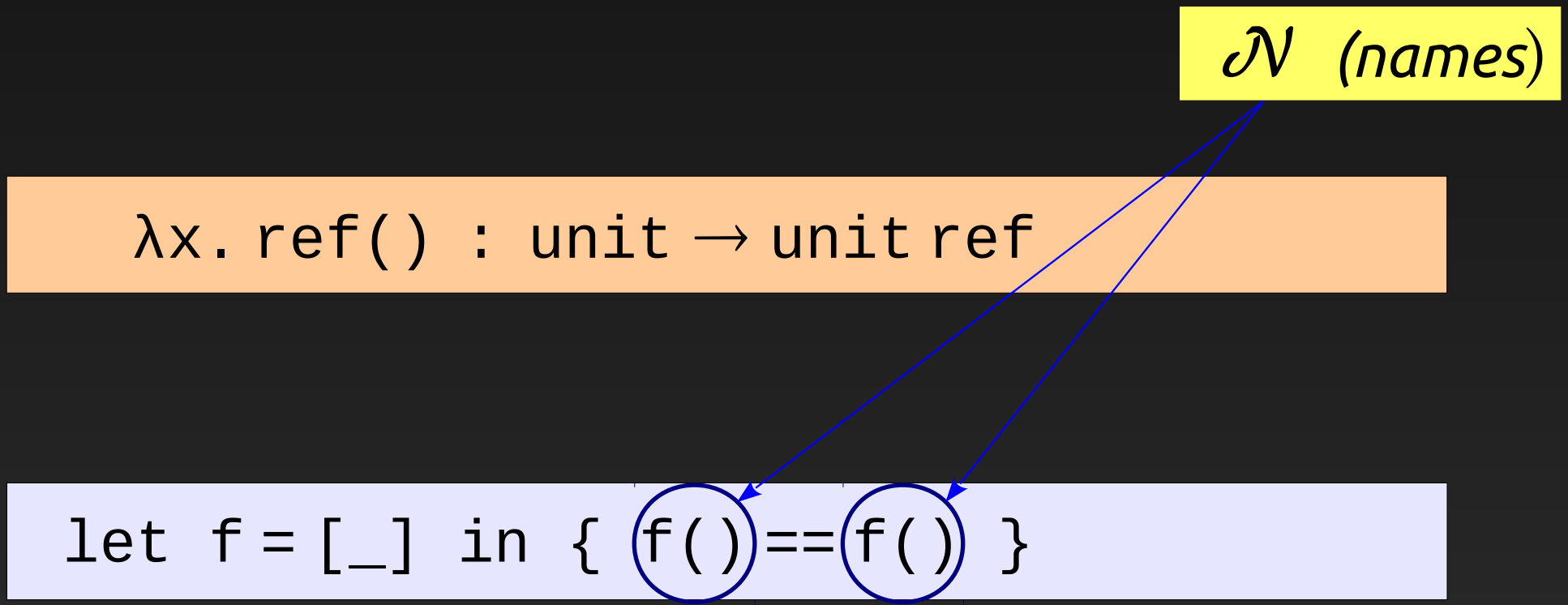
```
let f = [_] in { f() == f() }
```

Nominal games

```
λx. ref() : unit → unit ref
```

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

Nominal games



Nominal games

$\mathcal{N}$ *(names)*

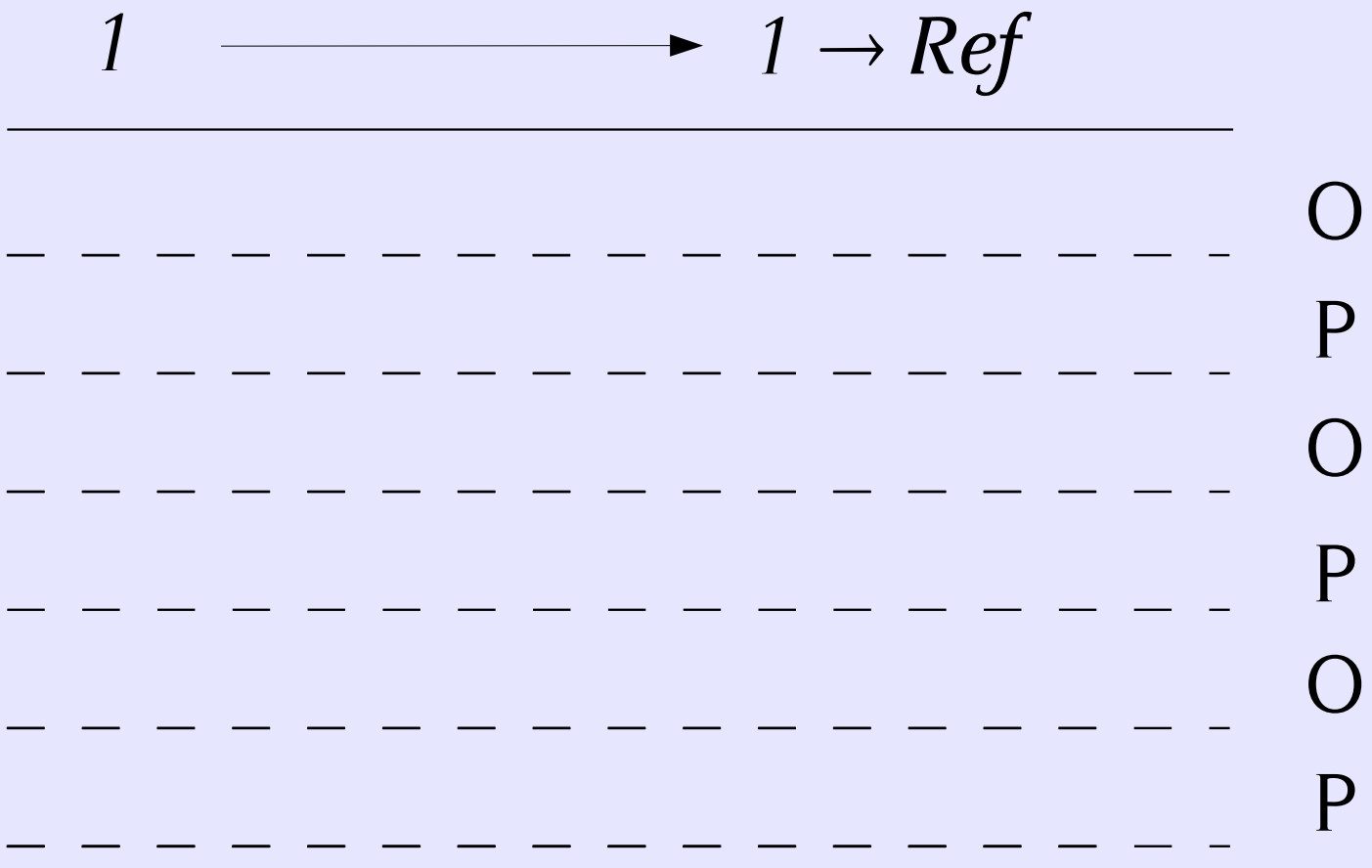
```
λx. ref() : unit → unit ref
```

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

Games in *Nominal Sets*

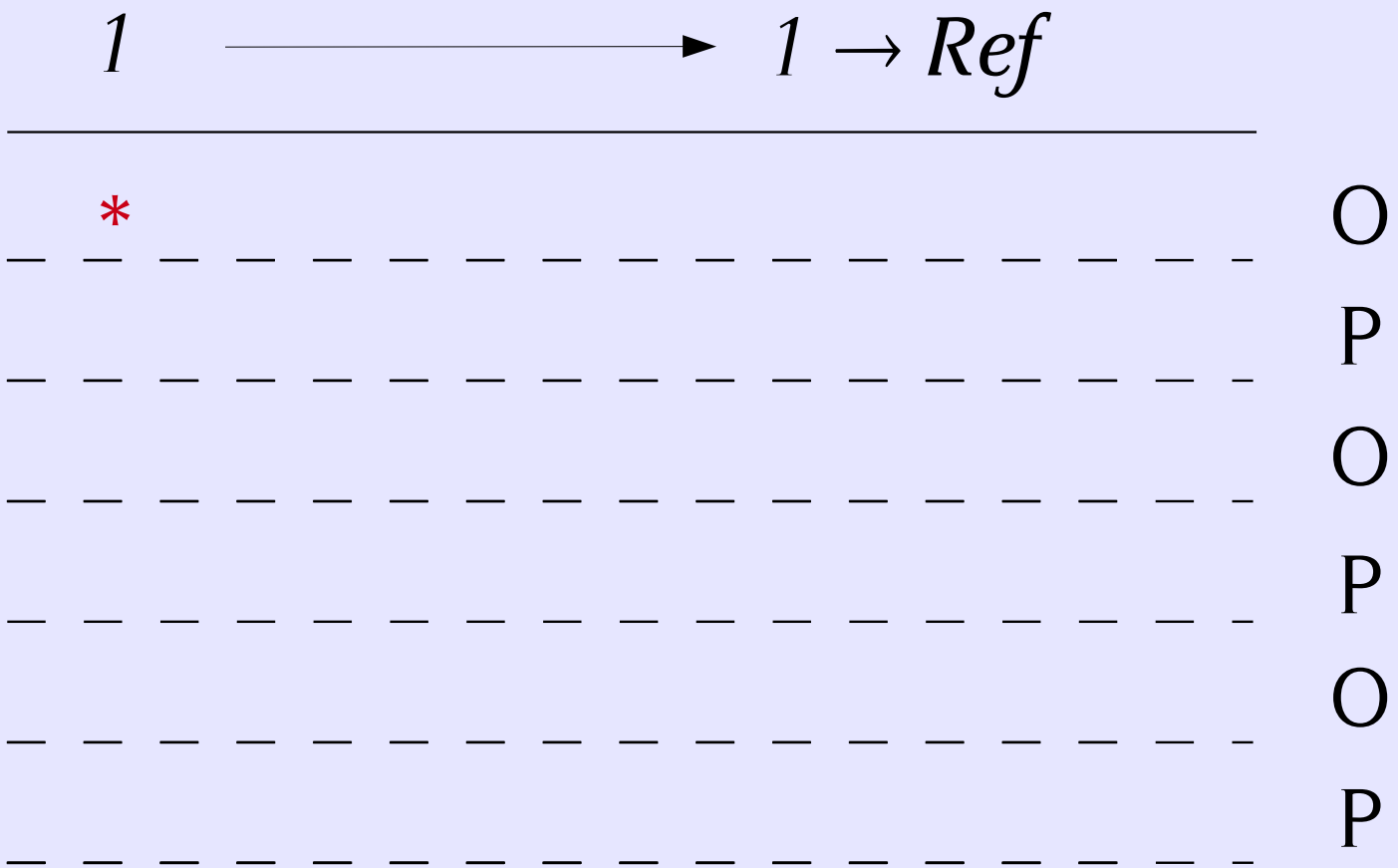
Examples

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



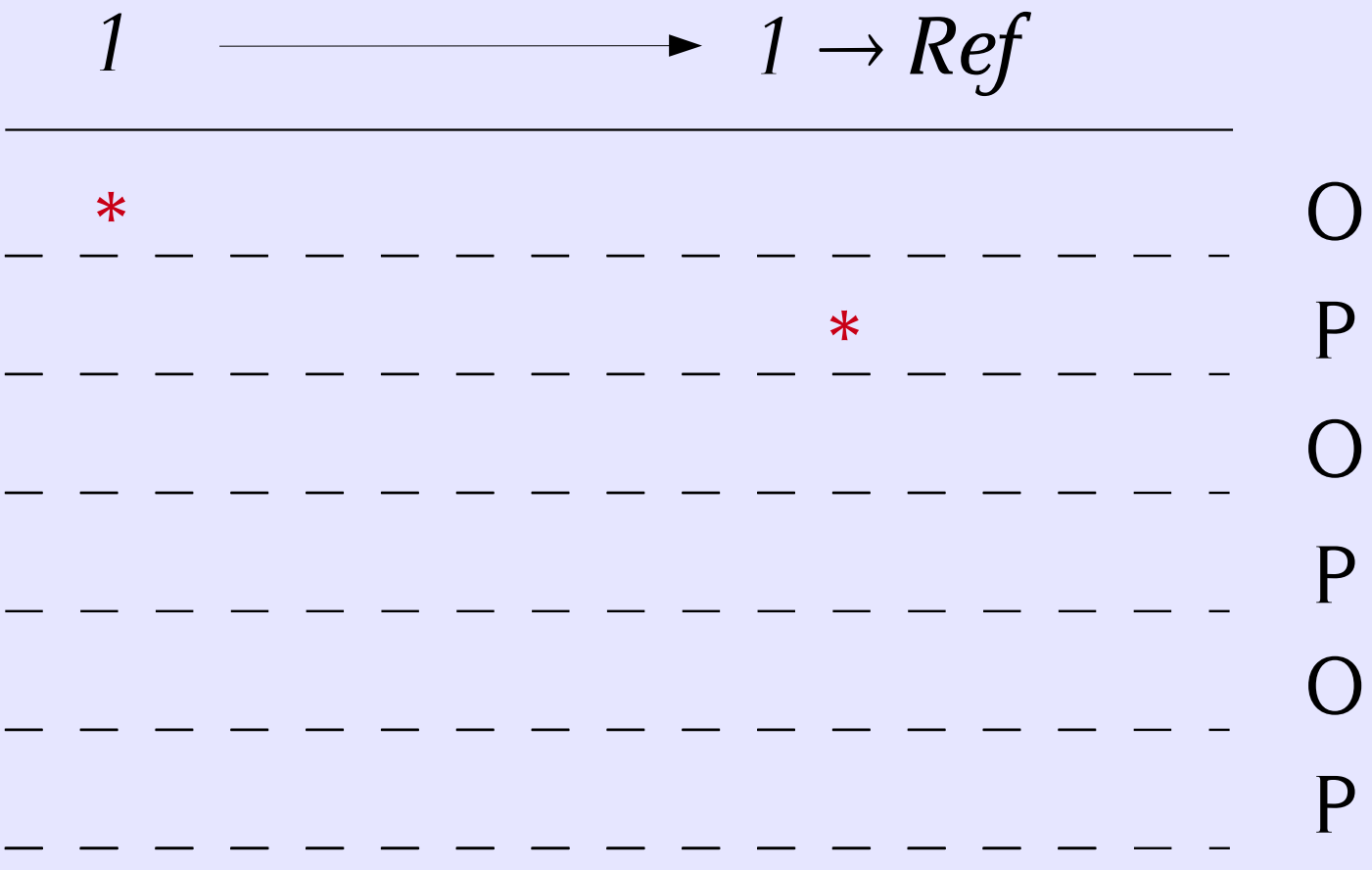
Examples

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



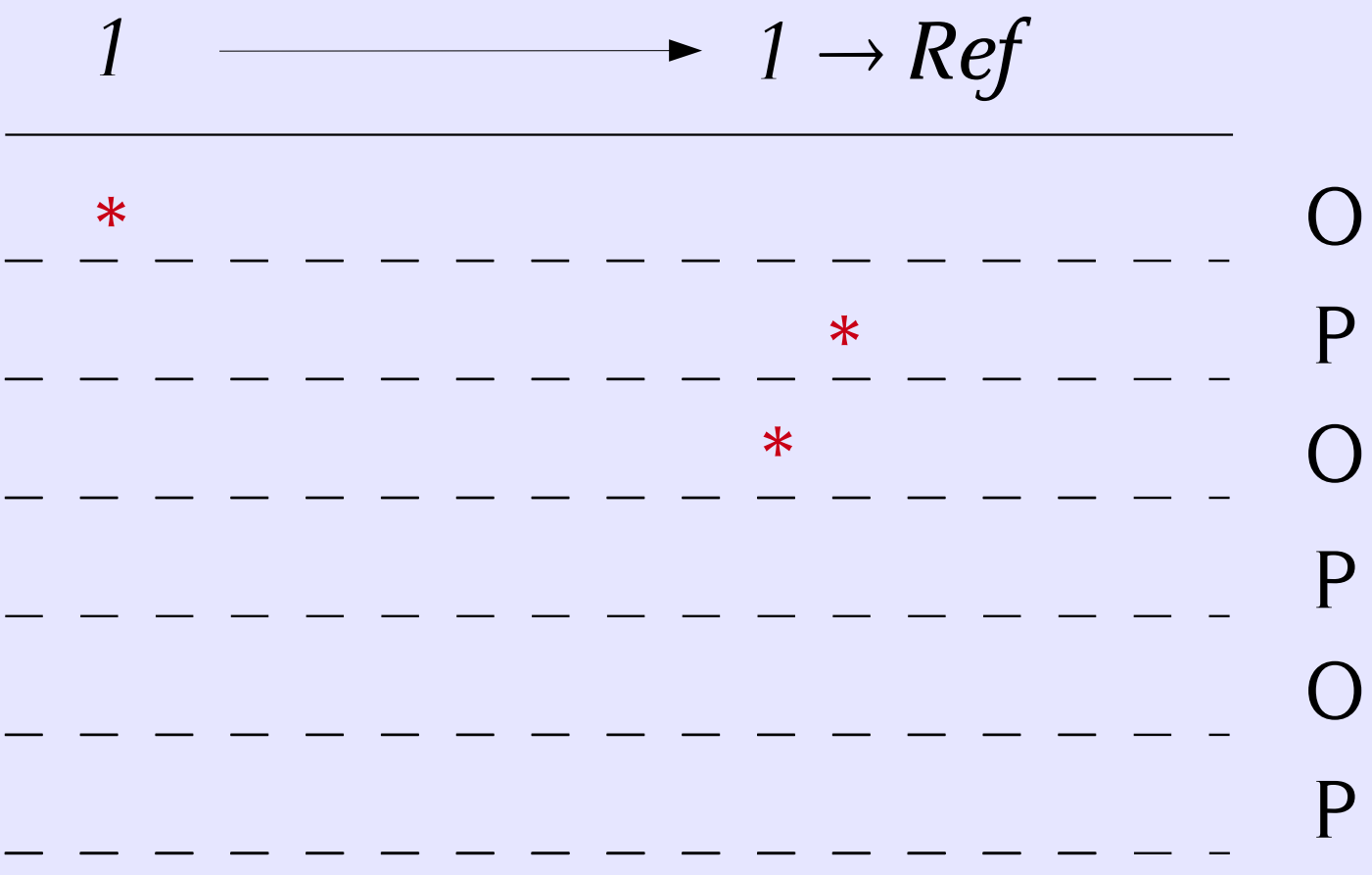
Examples

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



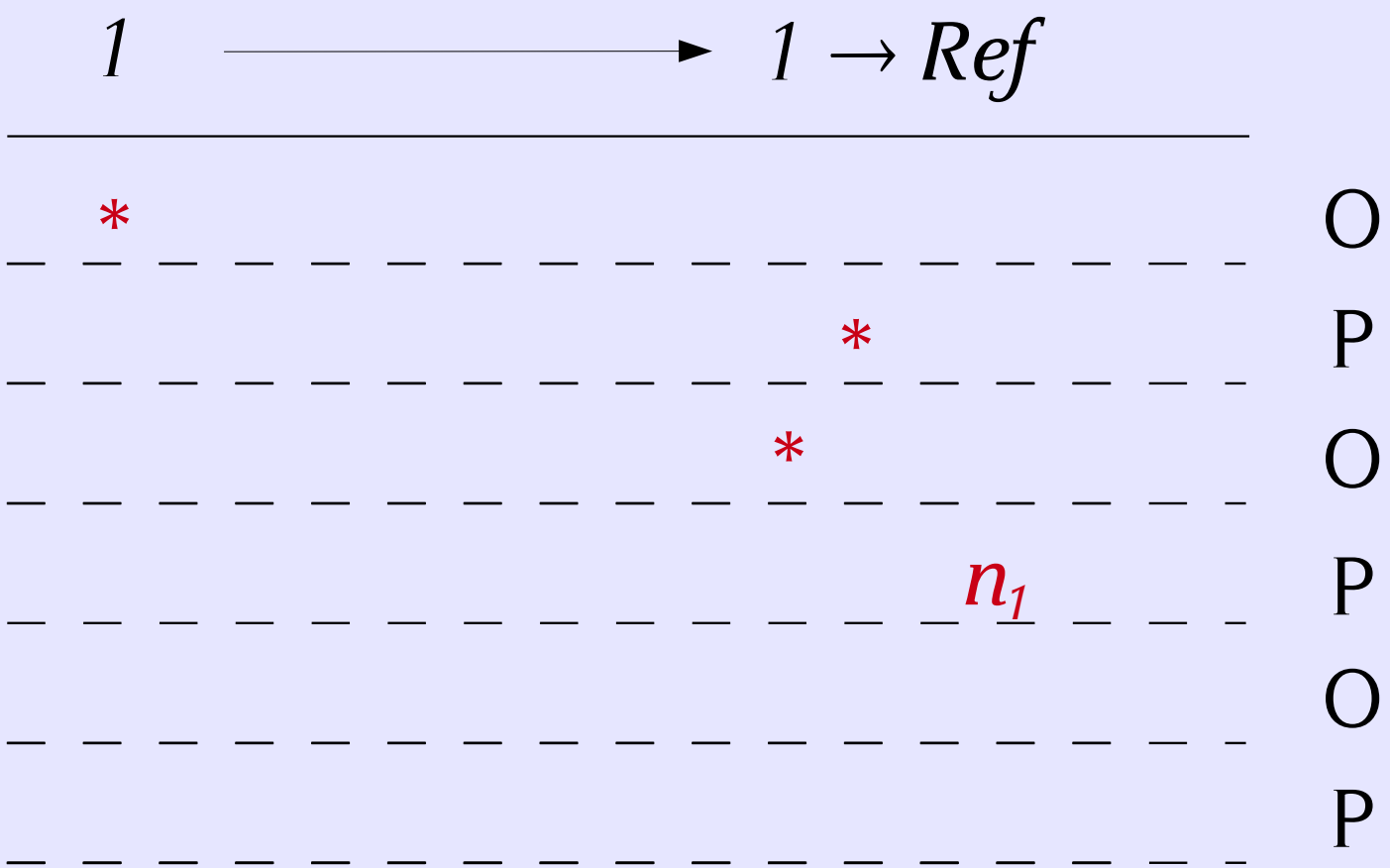
Examples

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



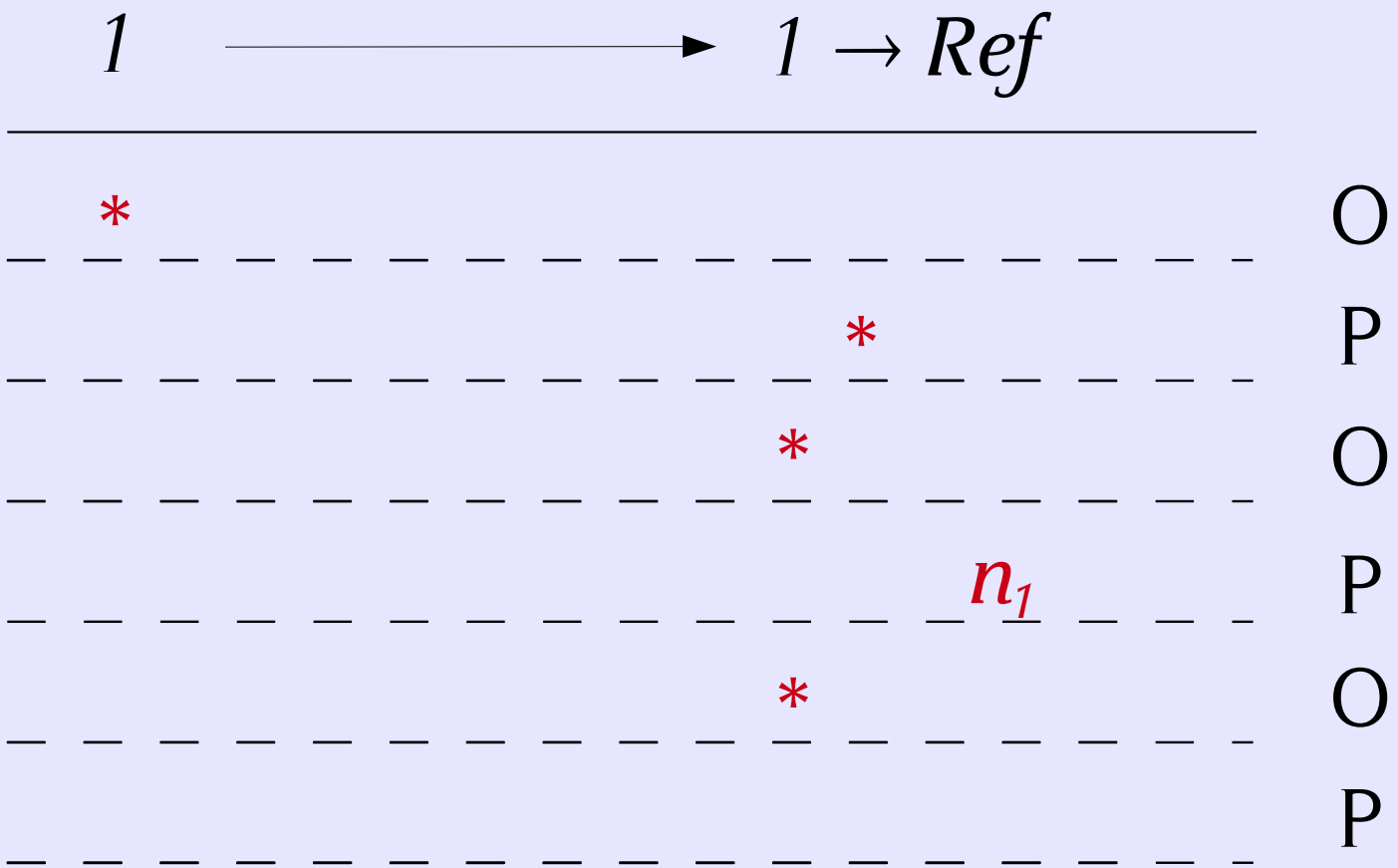
Examples

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



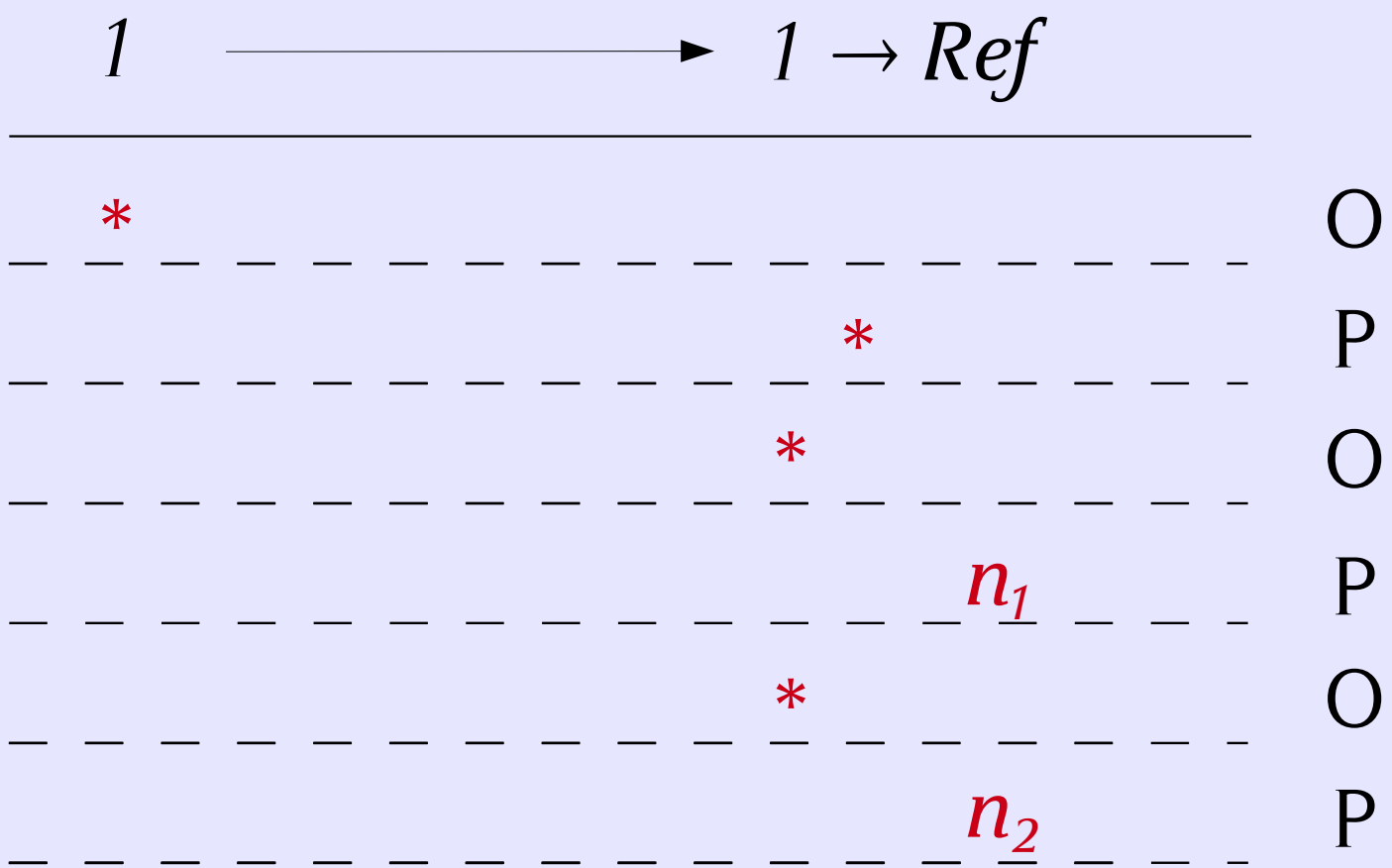
Examples

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



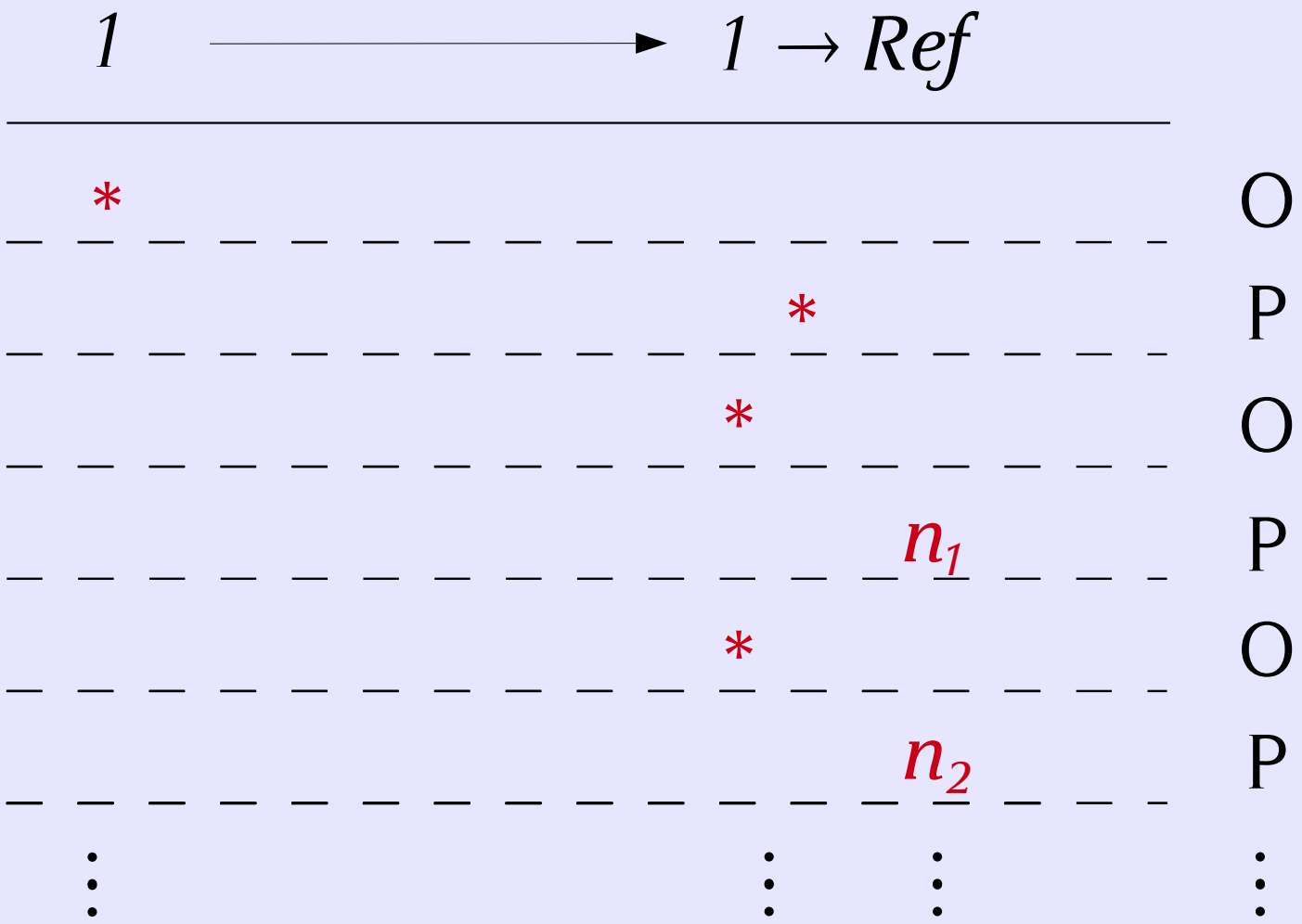
Examples

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



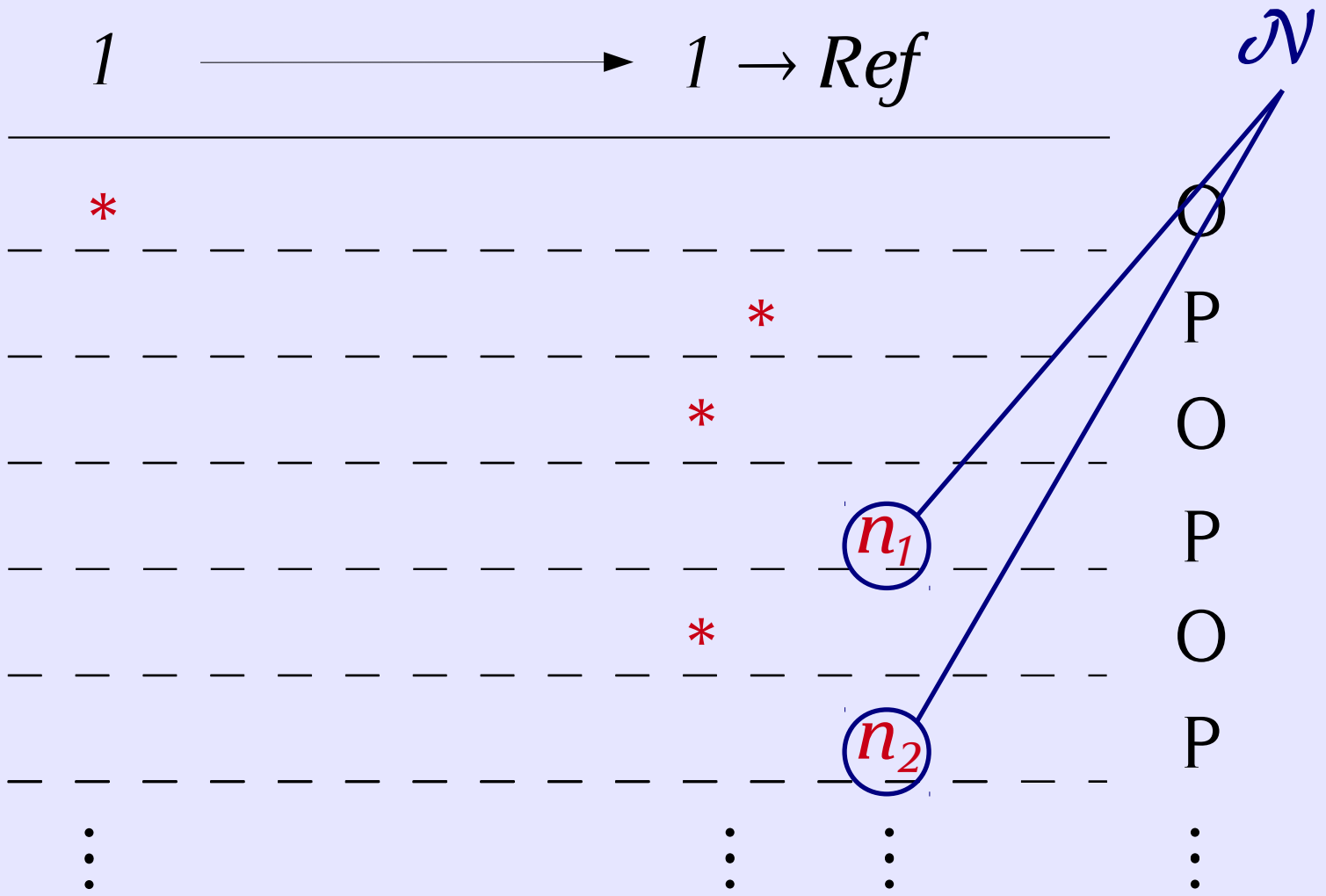
Examples

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



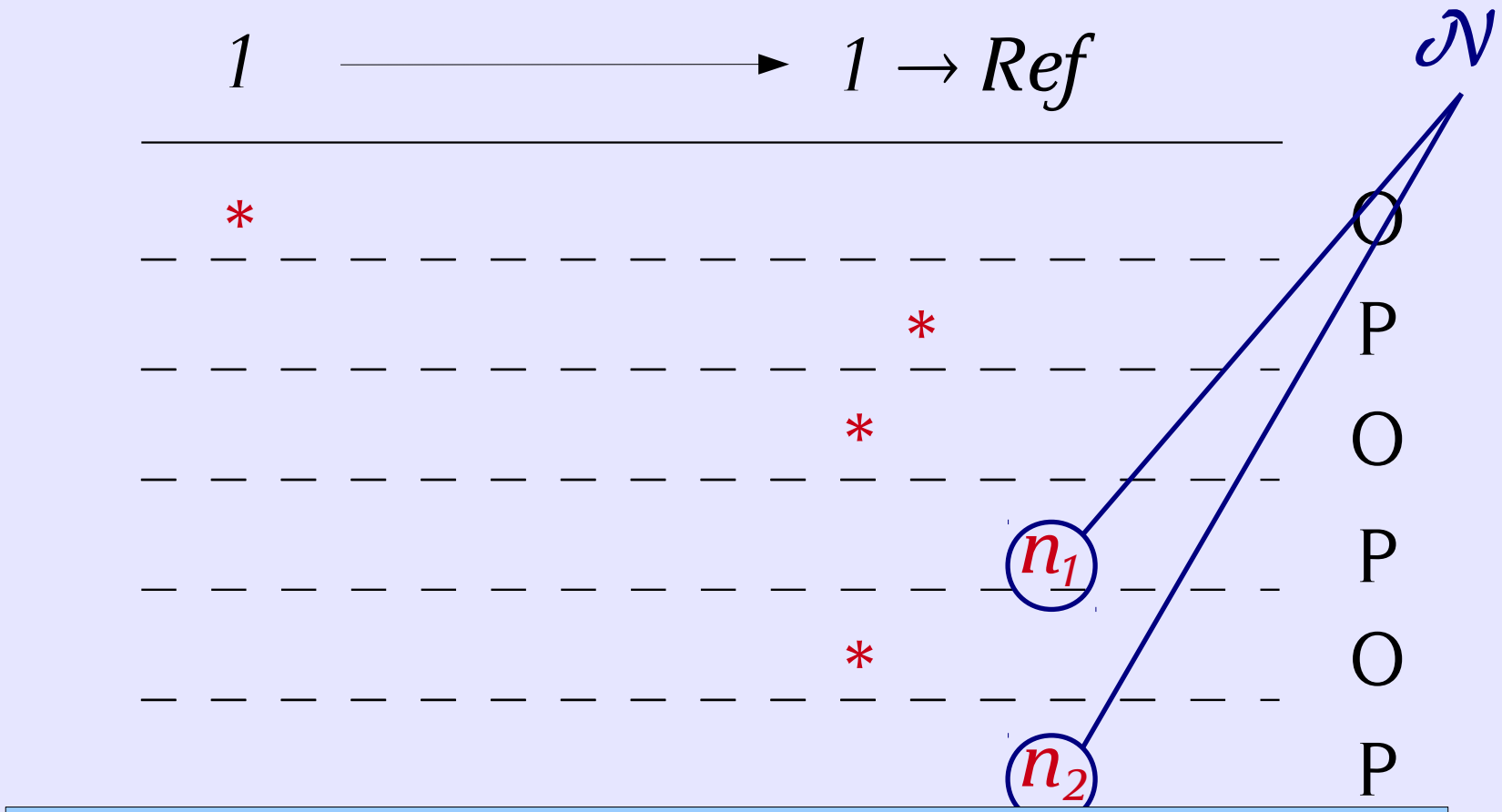
Examples

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



Examples

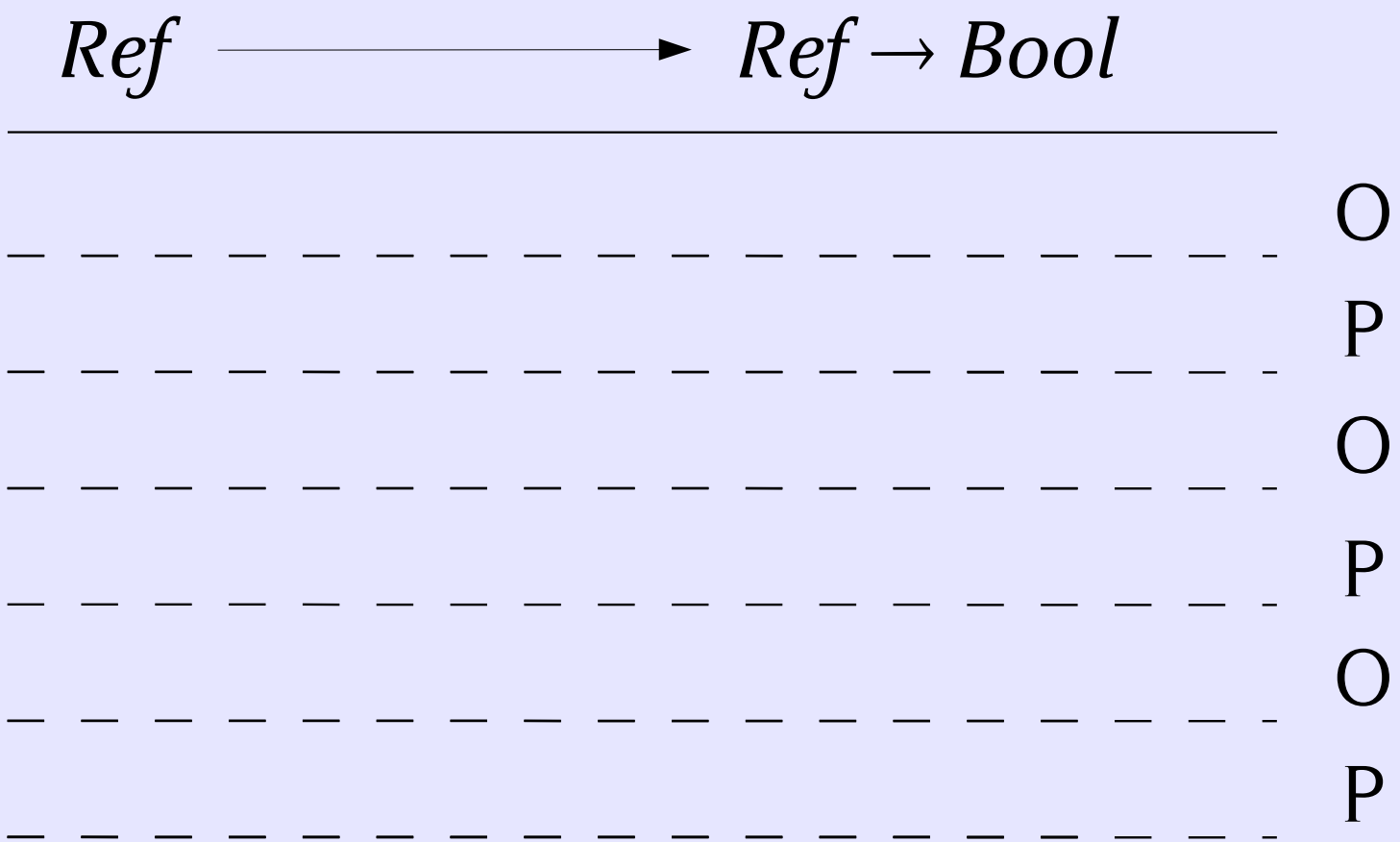
$\lambda x.\mathtt{ref}() : \mathtt{unit} \to \mathtt{ref}$



$[\![\lambda x.\mathtt{ref}()]\!] = \{ * * * n_1 * n_2 \ldots \}$

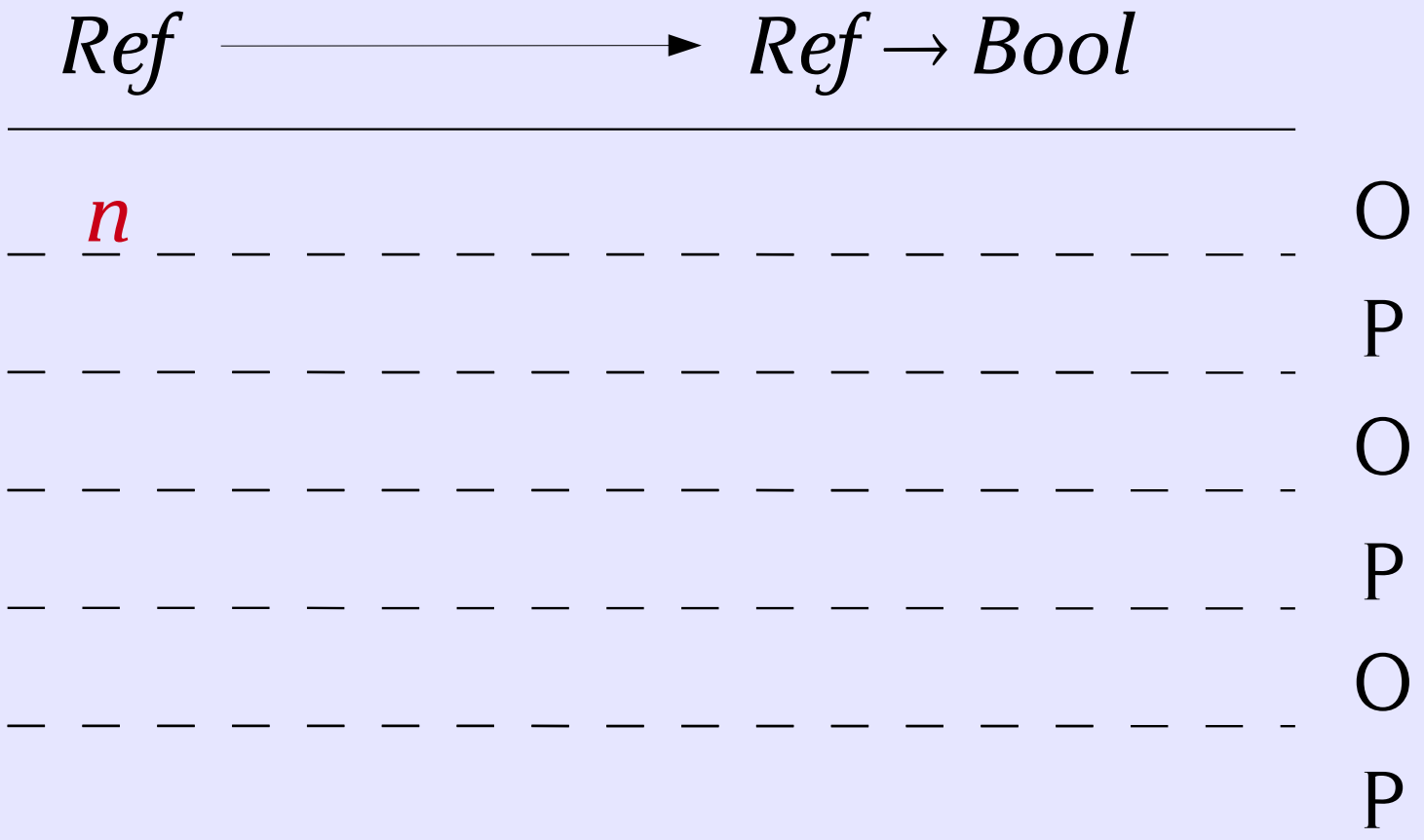
Examples

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



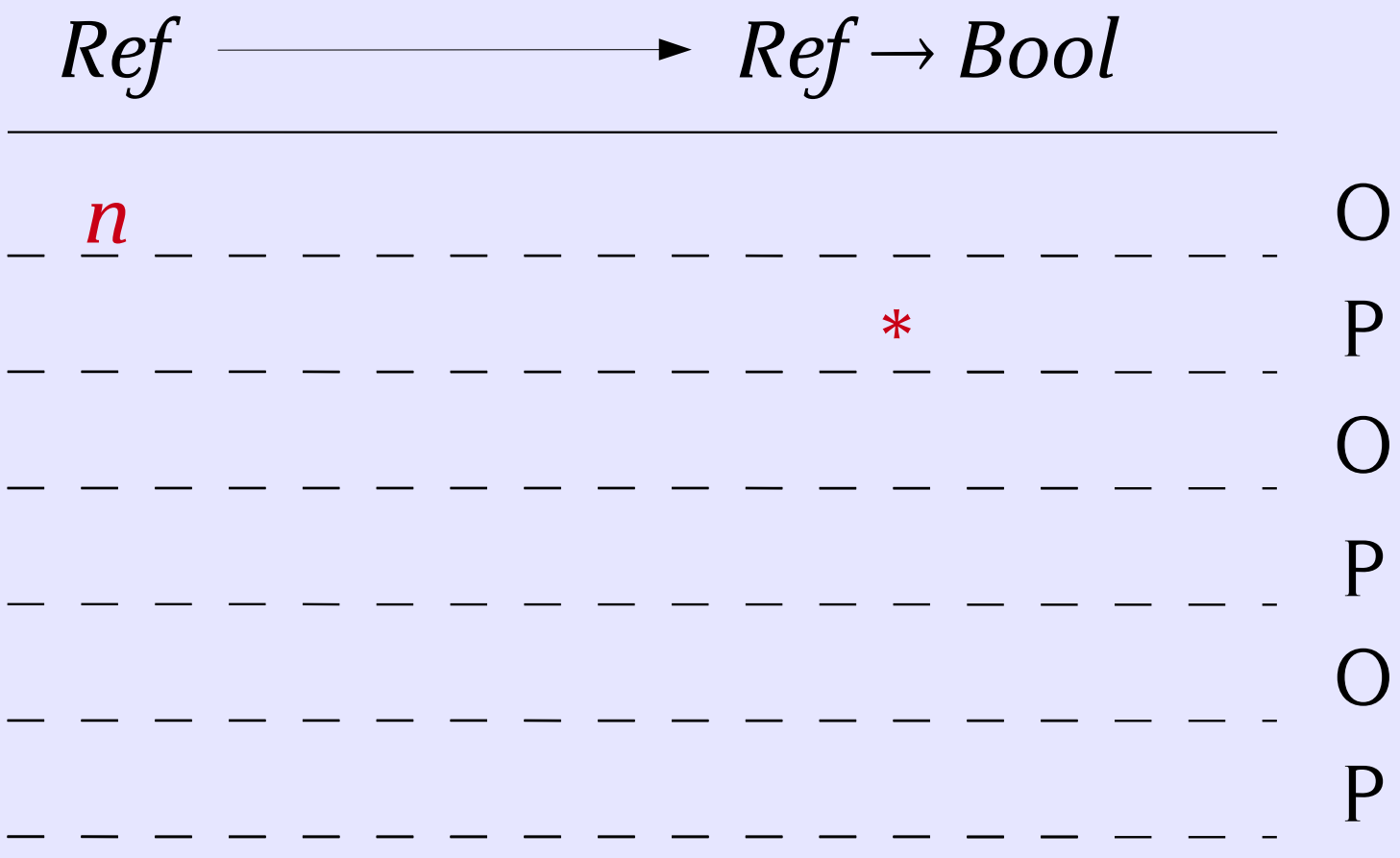
Examples

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



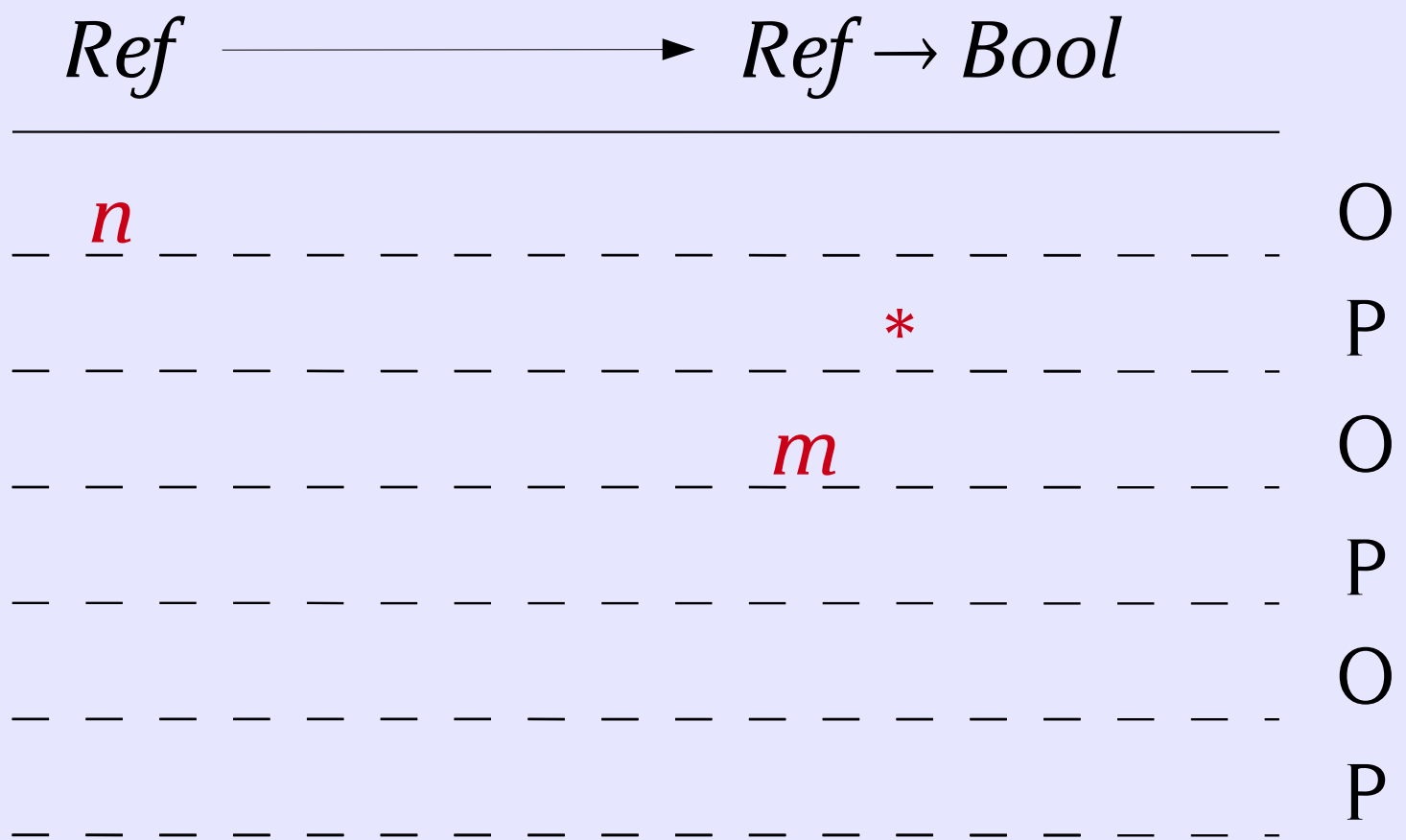
Examples

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



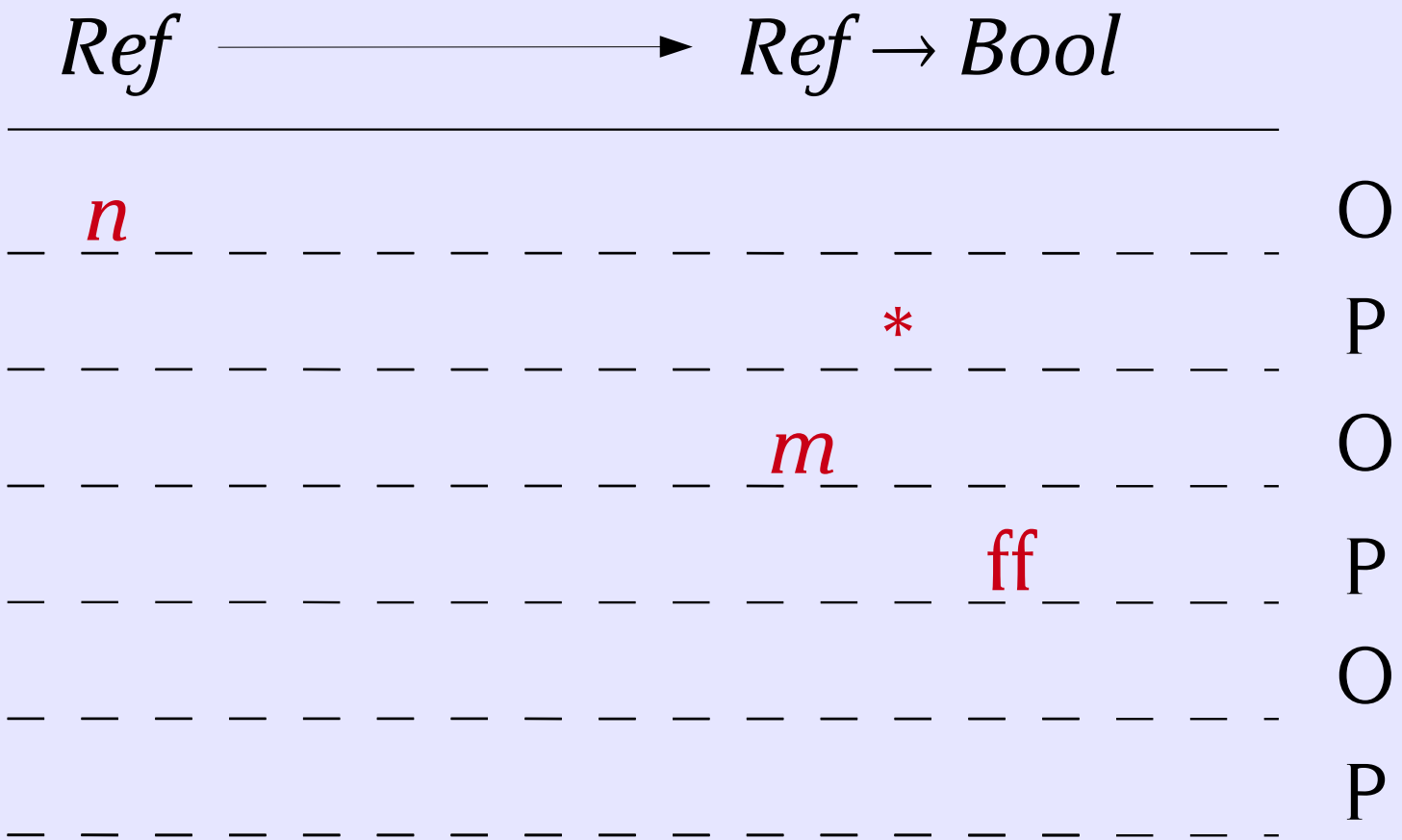
Examples

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



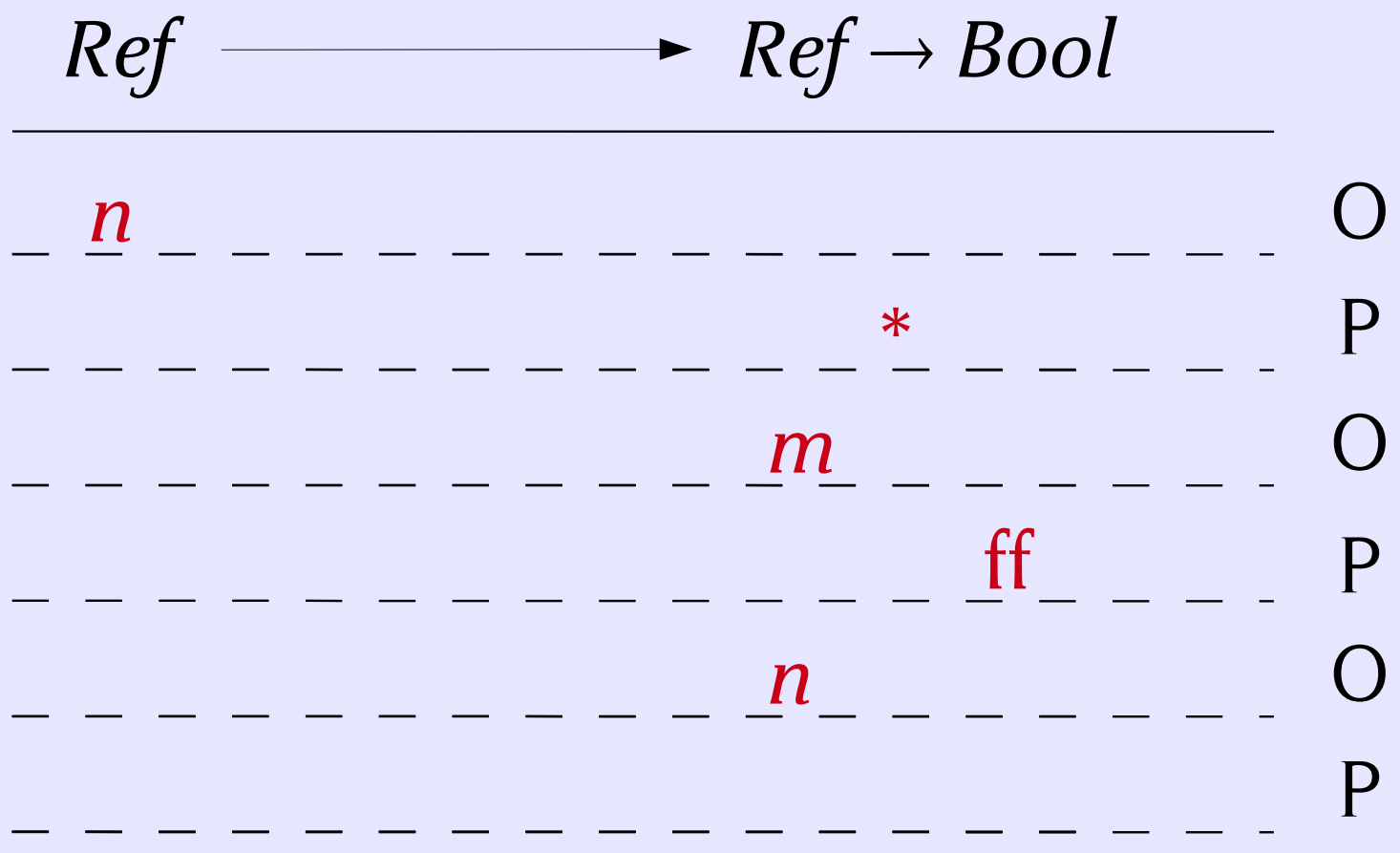
Examples

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



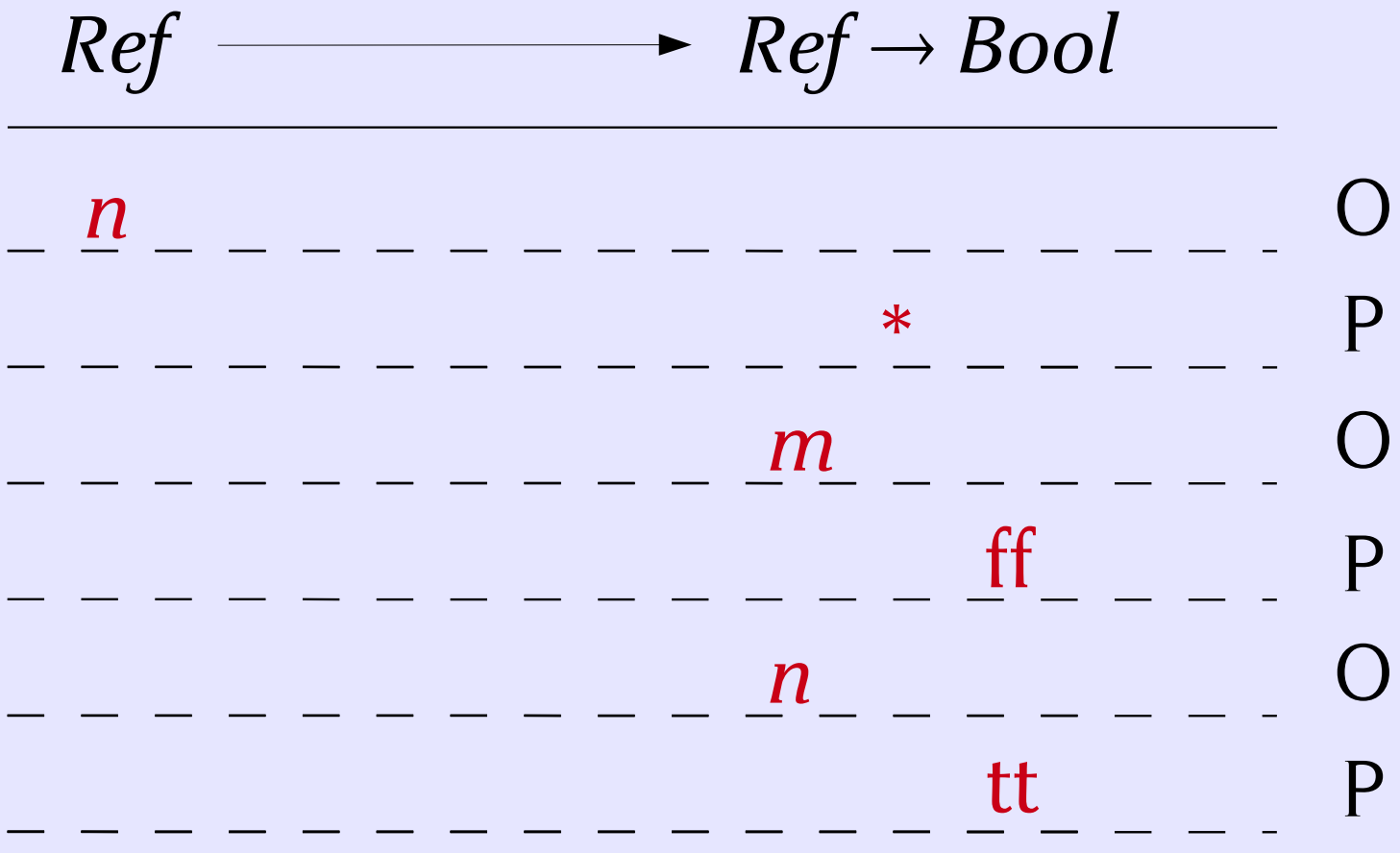
Examples

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



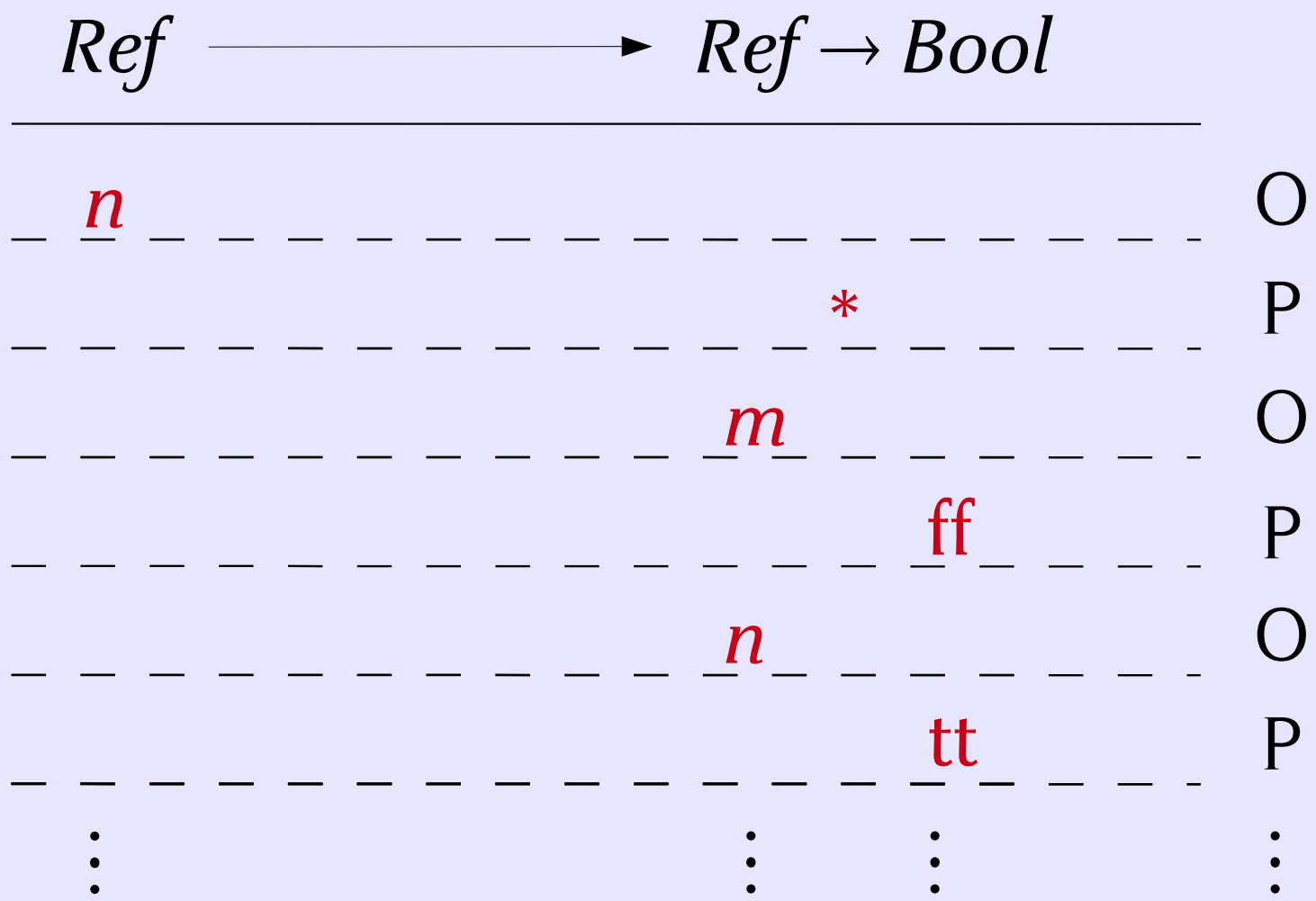
Examples

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



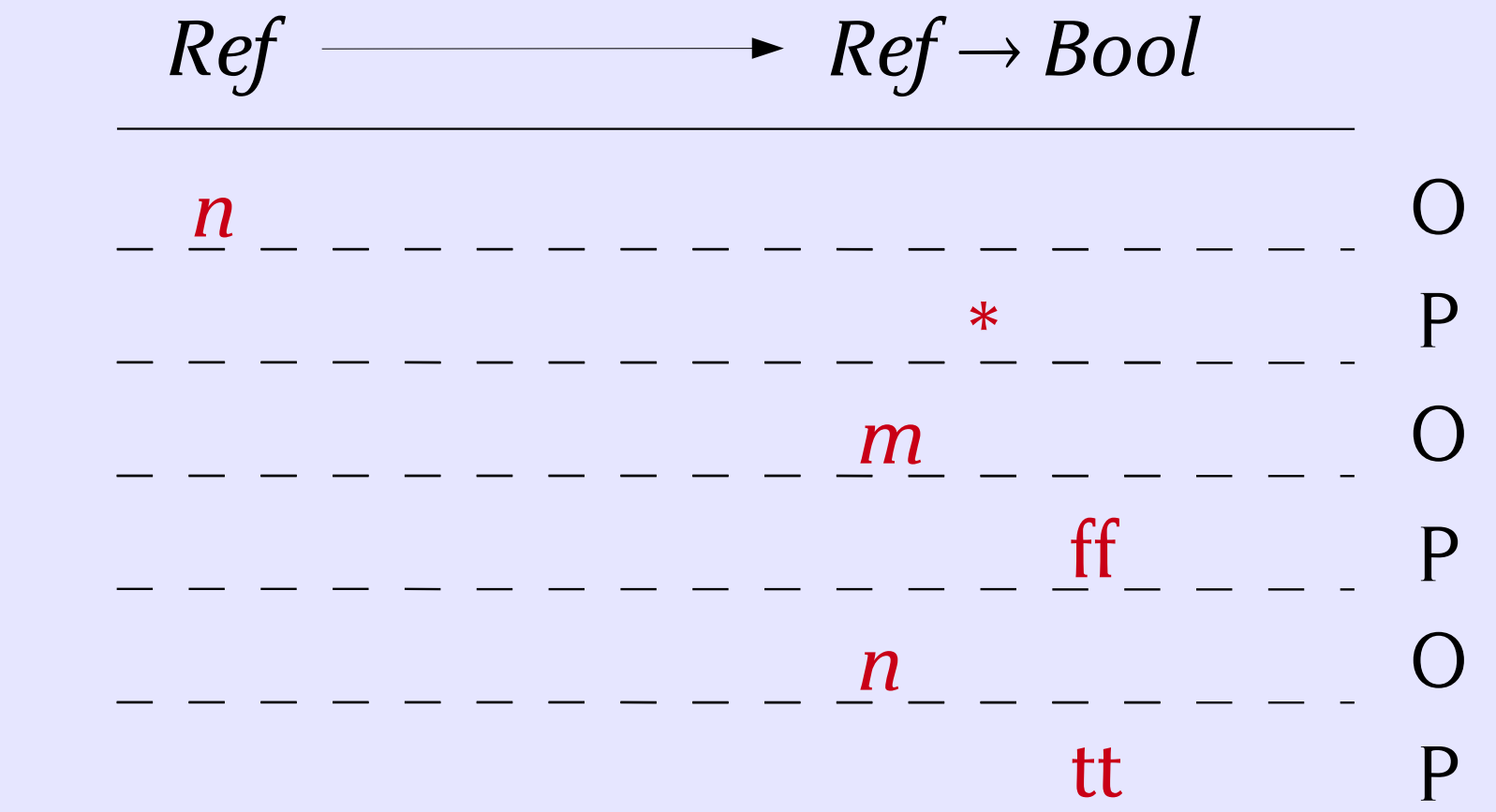
Examples

$\texttt{x : ref} \vdash \texttt{λy.(x == y) : ref} \to \texttt{bool}$



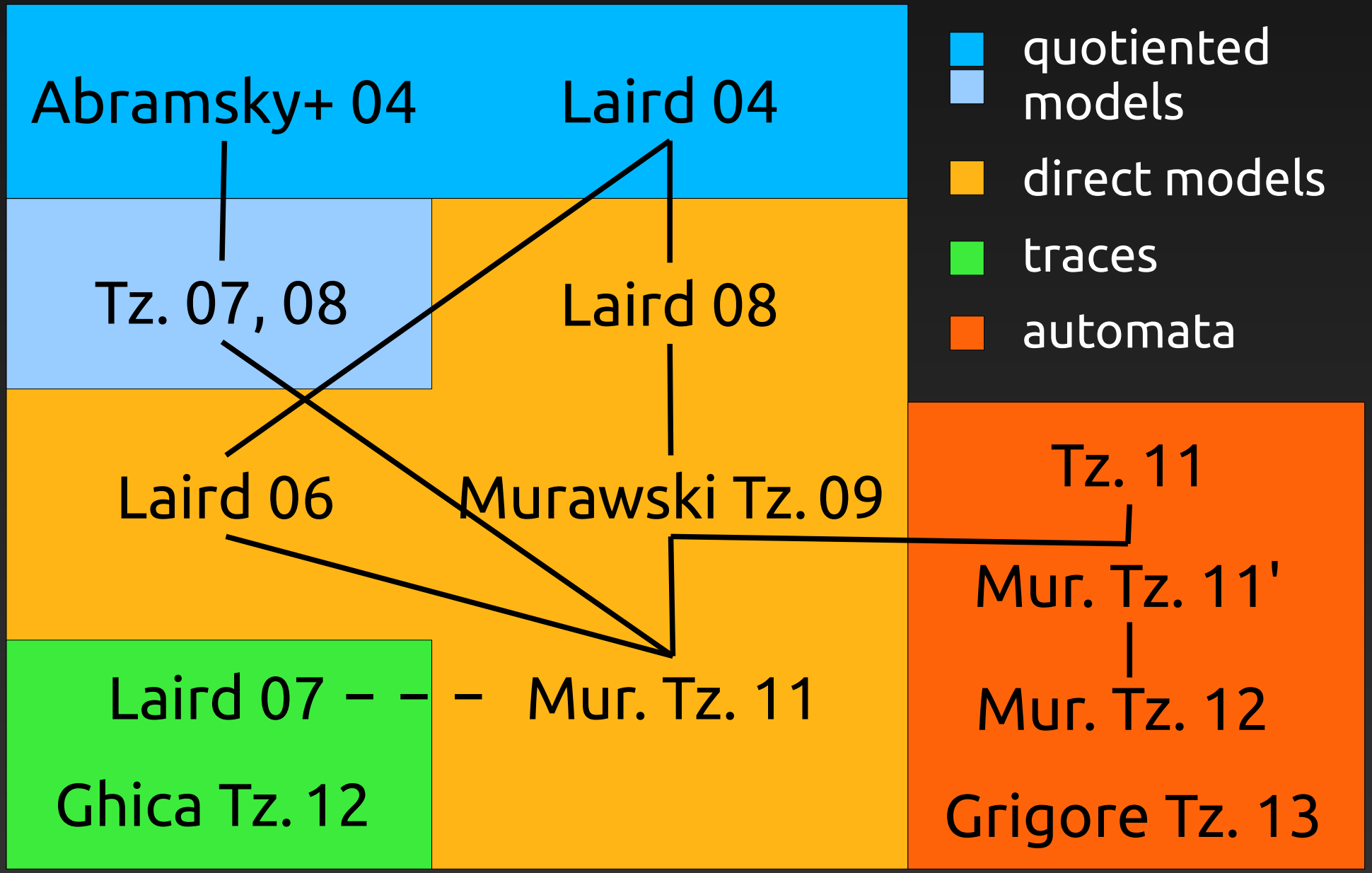
Examples

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



$[\lambda y.(x==y)] = \{\, n * m \;\; \mathrm{ff}\; n \;\; \mathrm{tt} \ldots \}$

Games with names



Quotiented models

- Abramsky+ 04, Laird 04: unit references
- Tz. 07, 08: higher-order references, exceptions

$$P \cong P' \;\Leftrightarrow\; [\,P\,] \cong [\,P'\,]$$

Quotiented models

- Abramsky+ 04, Laird 04: unit references
- Tz. 07, 08: higher-order references, exceptions

$$P \cong P' \;\Leftrightarrow\; [\,P\,] \cong [\,P'\,]$$

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

Direct models

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

$$P \cong P' \Leftrightarrow [\![P]\!] = [\![P']\!]$$

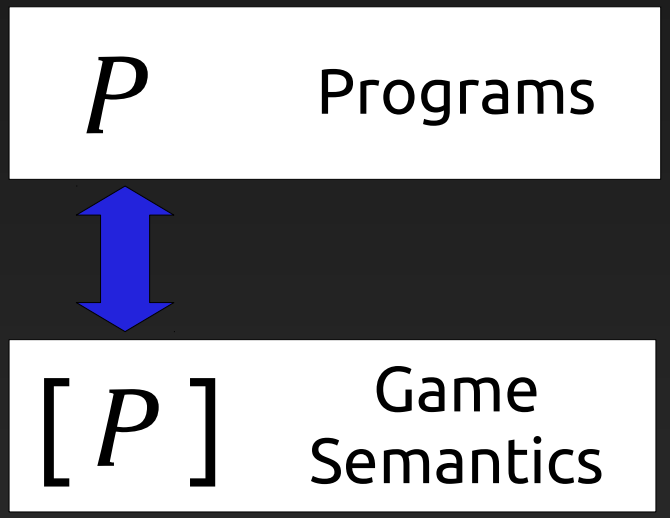
Direct models

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

$$P \cong P' \Leftrightarrow [\,P\,] = [\,P'\,]$$

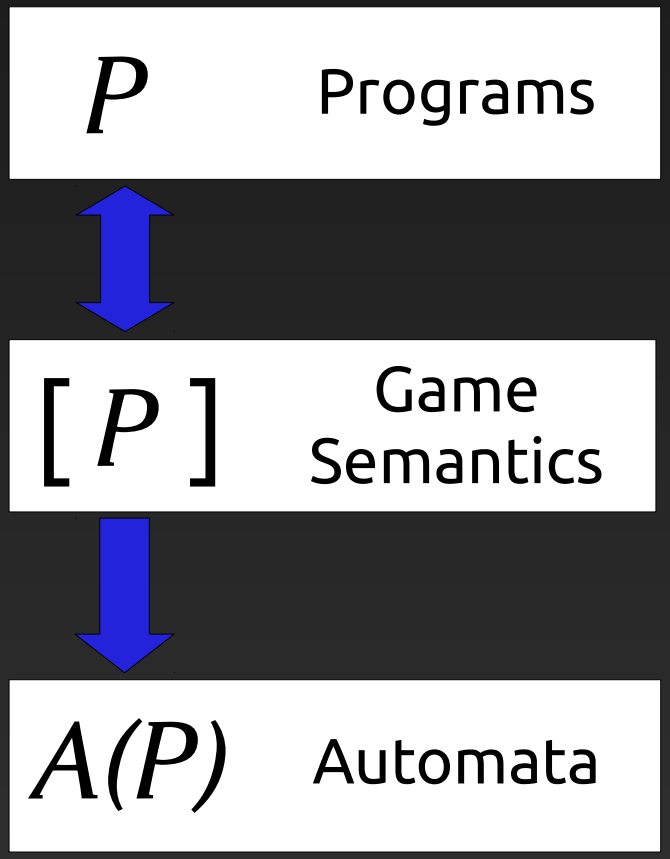
  - moves contain names
  - moves-with-store
  - name-availability conditions/ “direct” effects

Algorithmics



$$P \cong P' \Leftrightarrow [\,P\,] = [\,P'\,]$$

Algorithmics



$$P \cong P' \Leftrightarrow [\,P\,] = [\,P'\,]$$

$$[\lambda x.\mathtt{ref}()] = \{ * * * n_1 * n_2 \ldots \}$$

$$[\,P\,] = [\,P'\,] \Leftrightarrow A(P) \sim A(P')$$

Fresh-register automata

$$[\lambda x.\texttt{ref}()] = \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$$

Automata with names

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

Fresh-register automata

$$[\lambda x.\texttt{ref}()] = \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$$

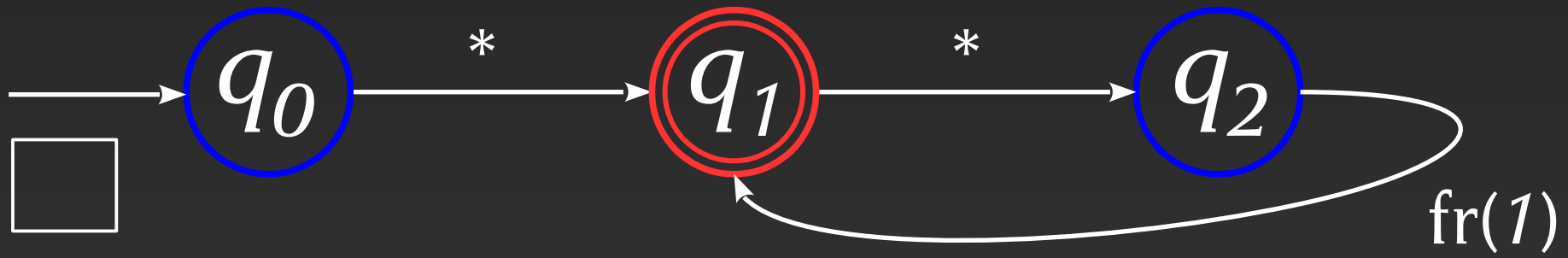
Automata with names

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

Finite-state machines with registers

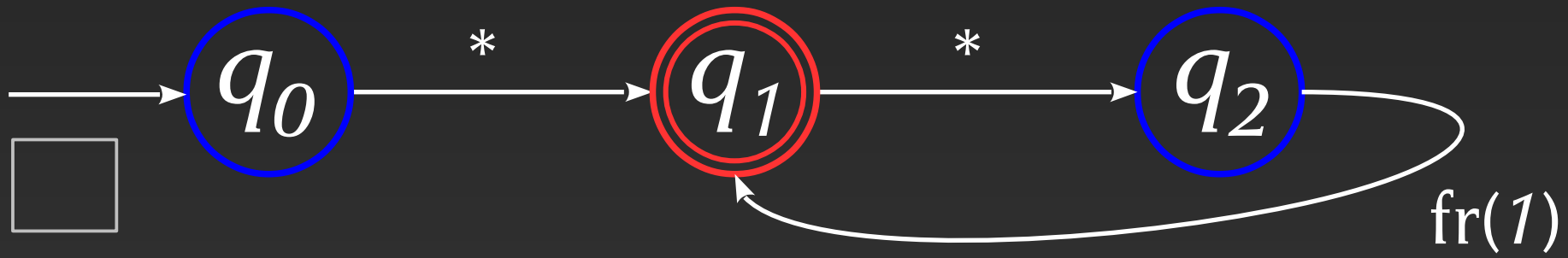
- *Kaminski & Francez (1994)*

Fresh-register automata

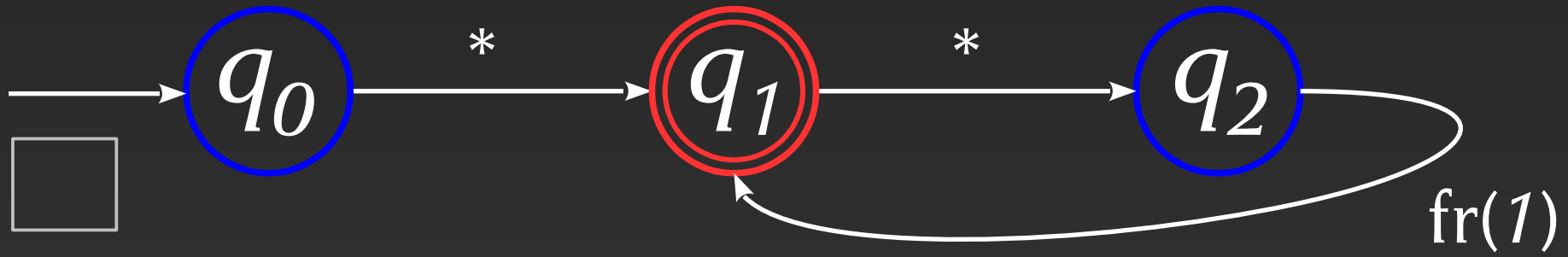


Fresh-register automata

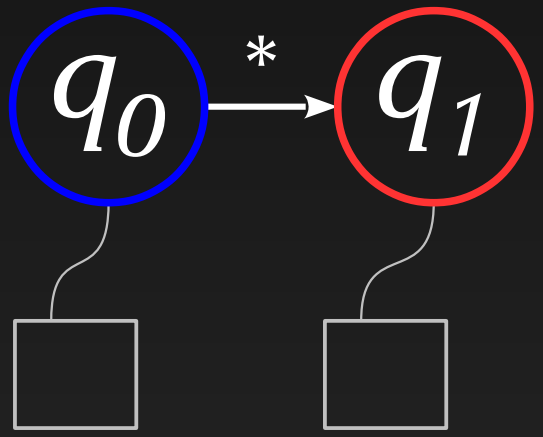
Fresh-register automata

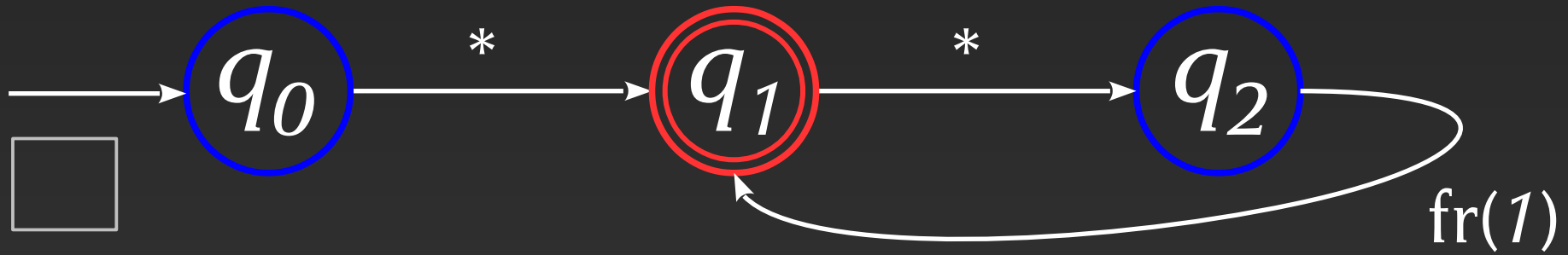


Fresh-register automata

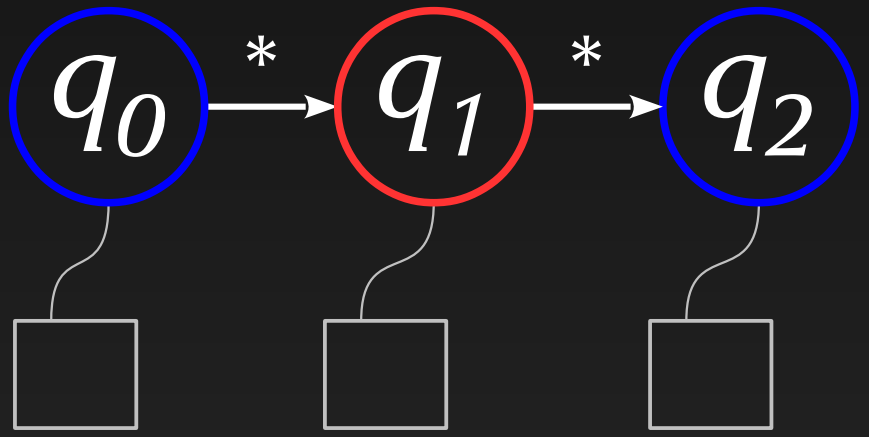


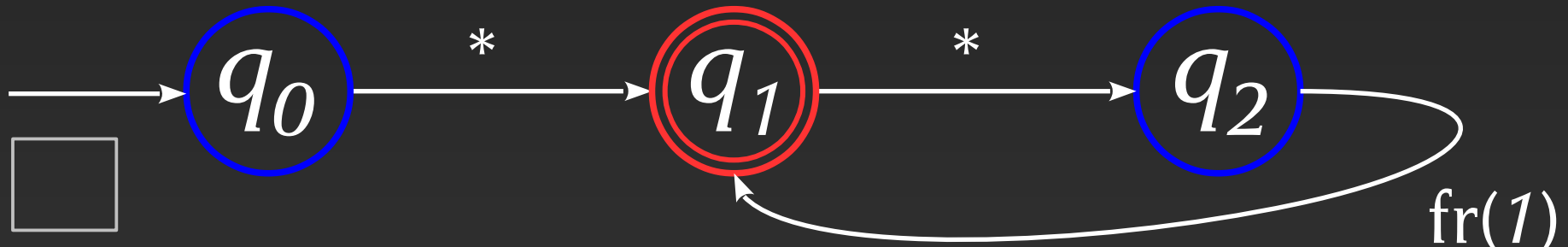
Fresh-register automata



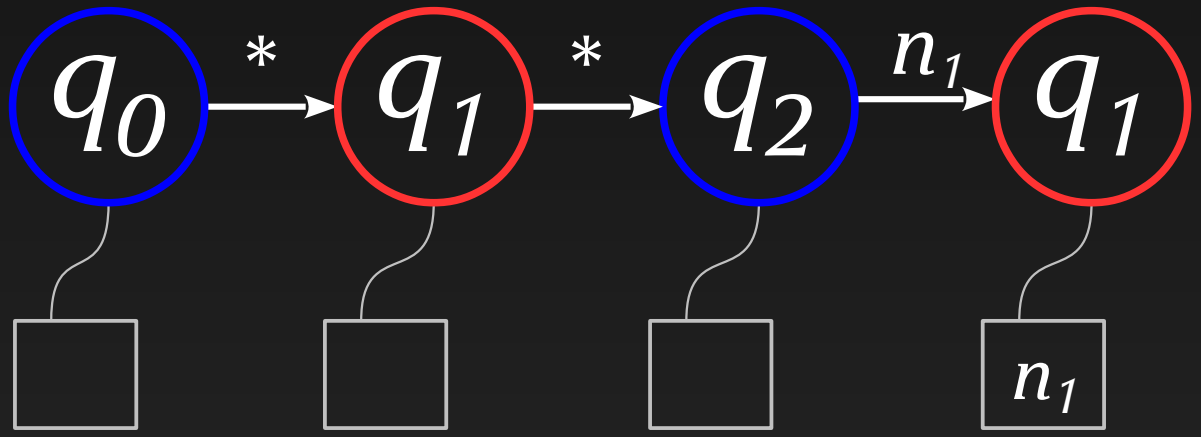


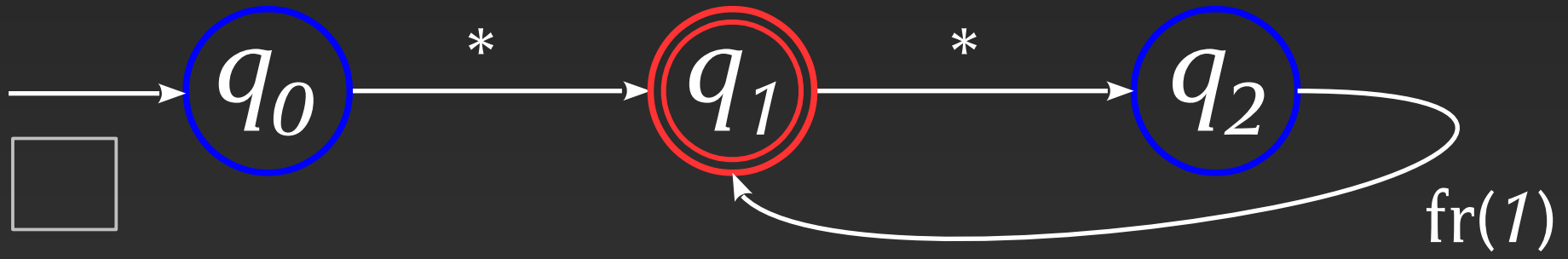
Fresh-register automata



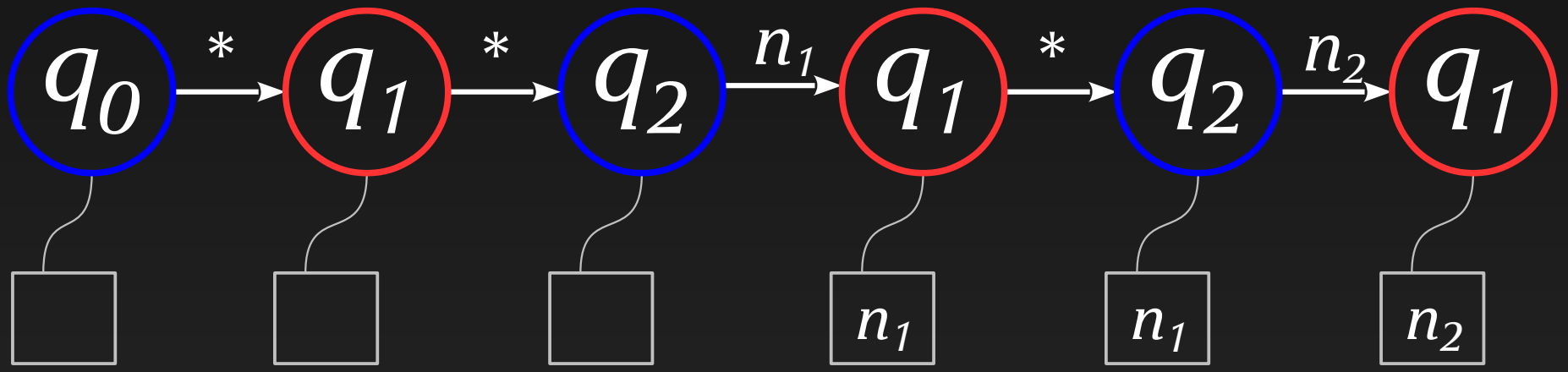


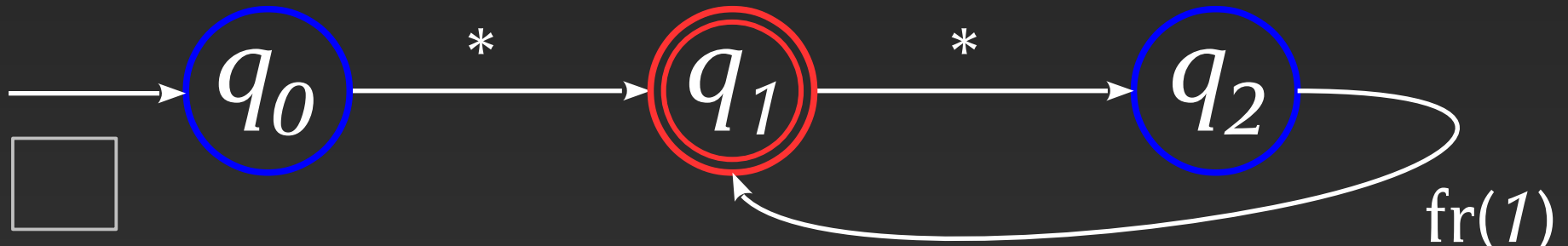
Fresh-register automata



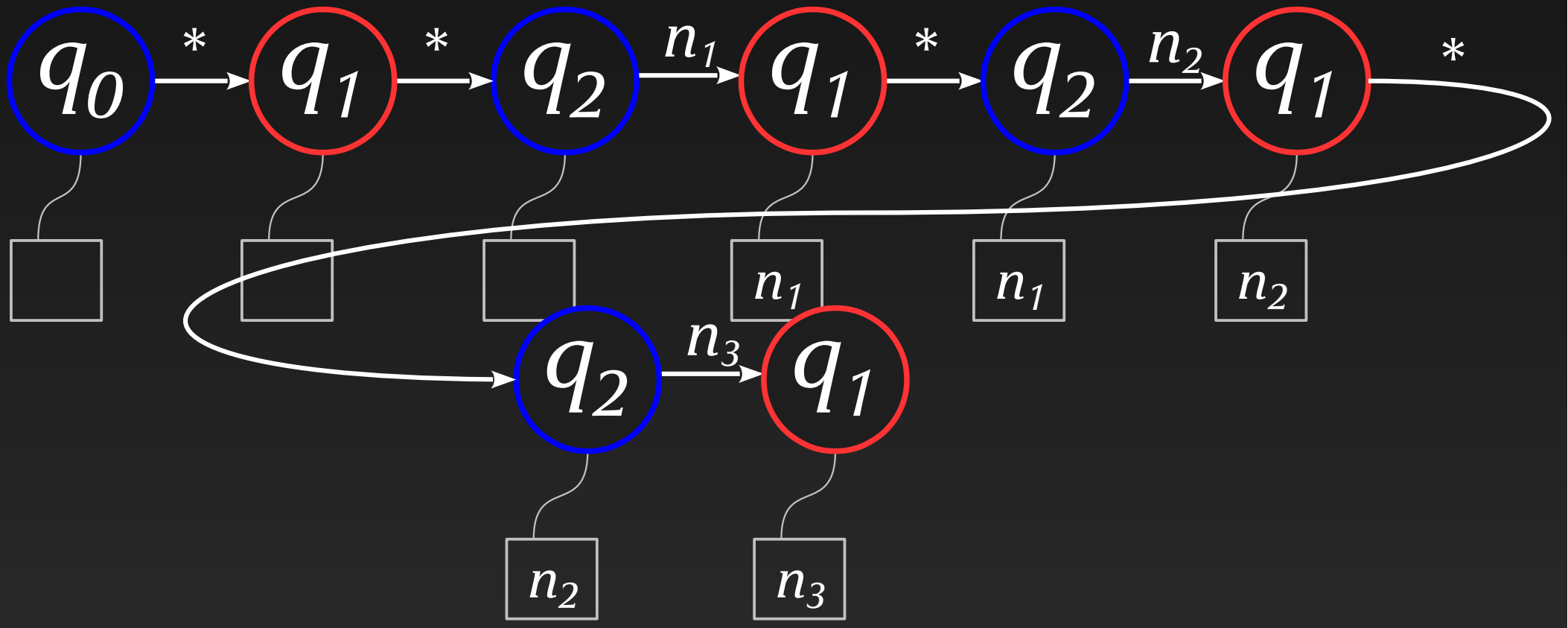


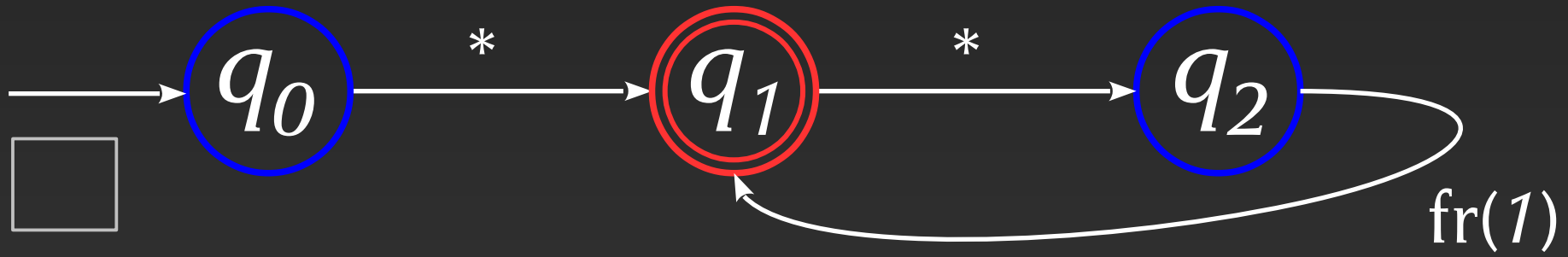
Fresh-register automata



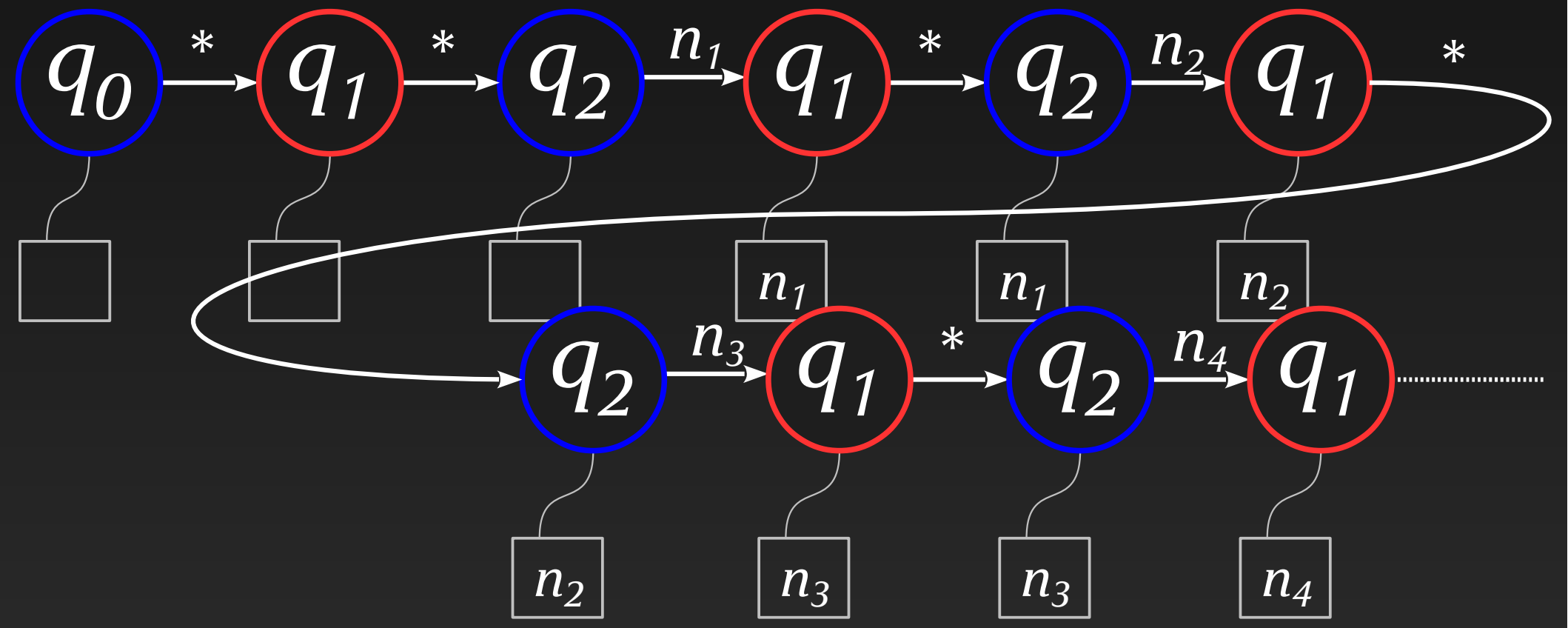


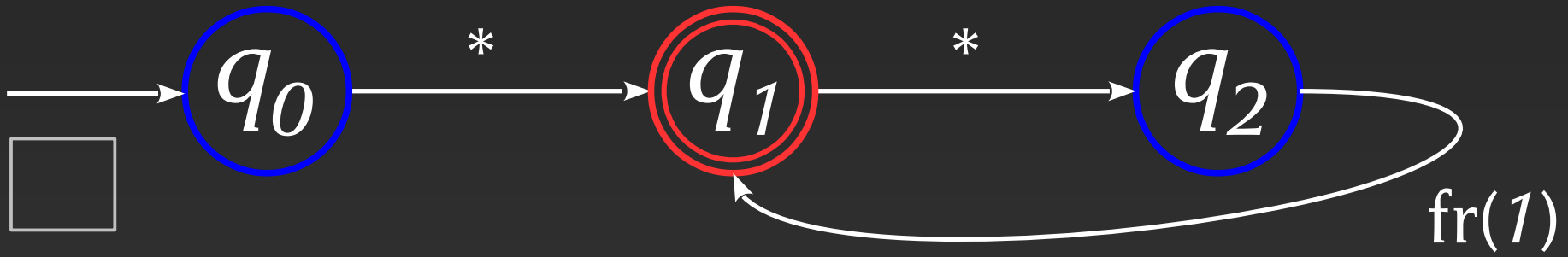
Fresh-register automata



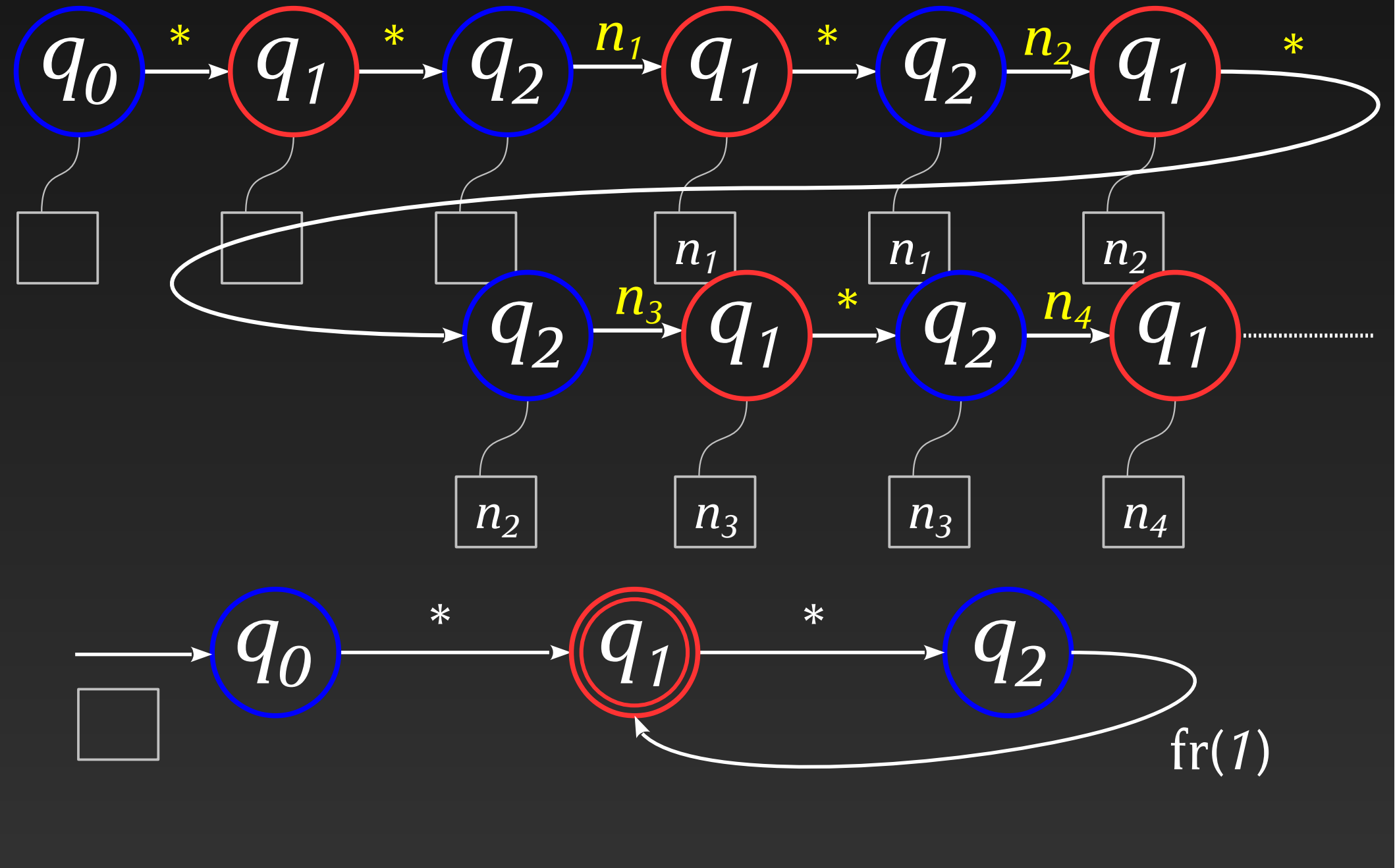


Fresh-register automata

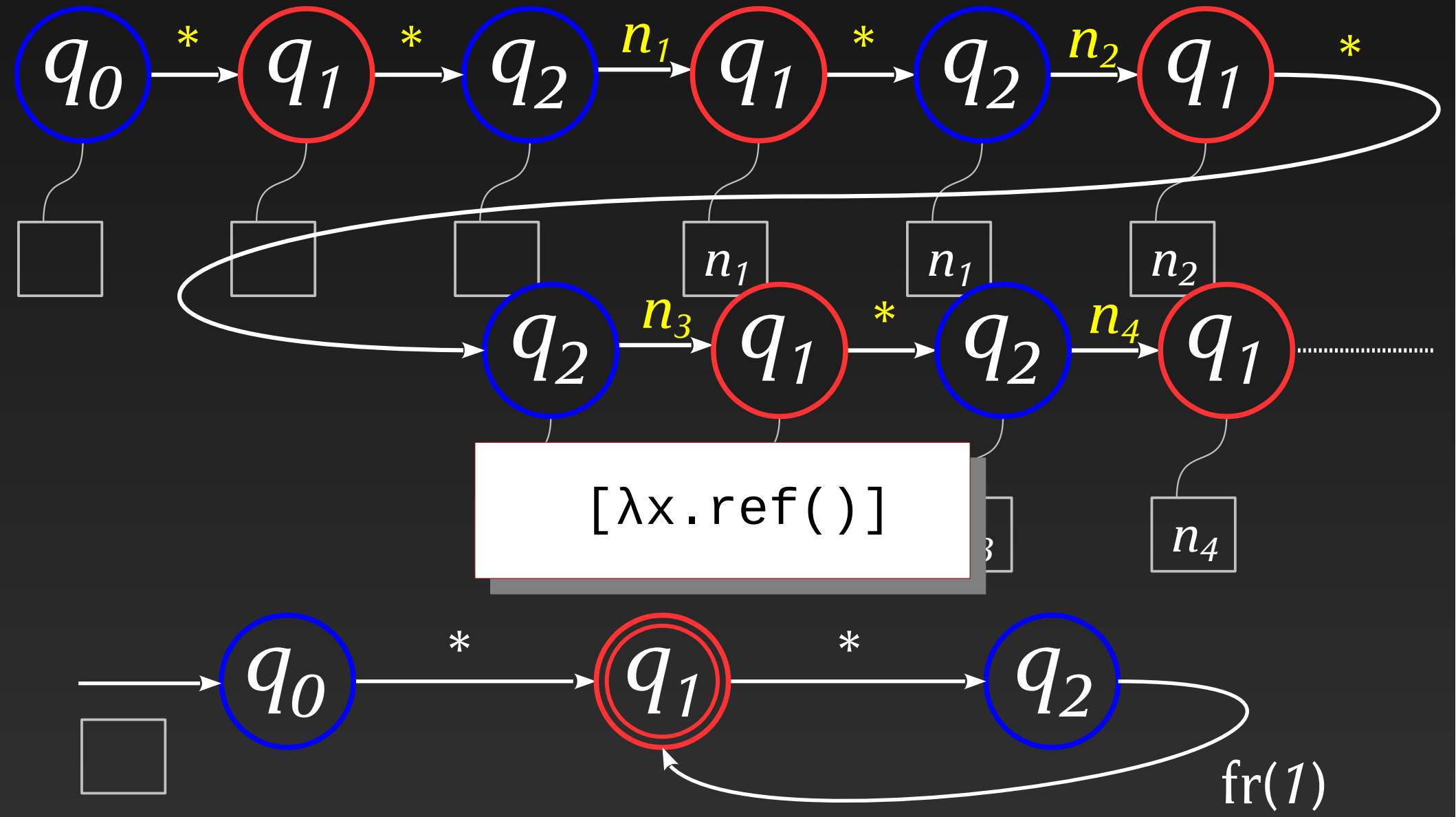




Fresh-register automata



Fresh-register automata



Proponent *Opponent*

Automata

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

Automata

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

$$P \cong P' \iff A(P) \sim A(P')$$

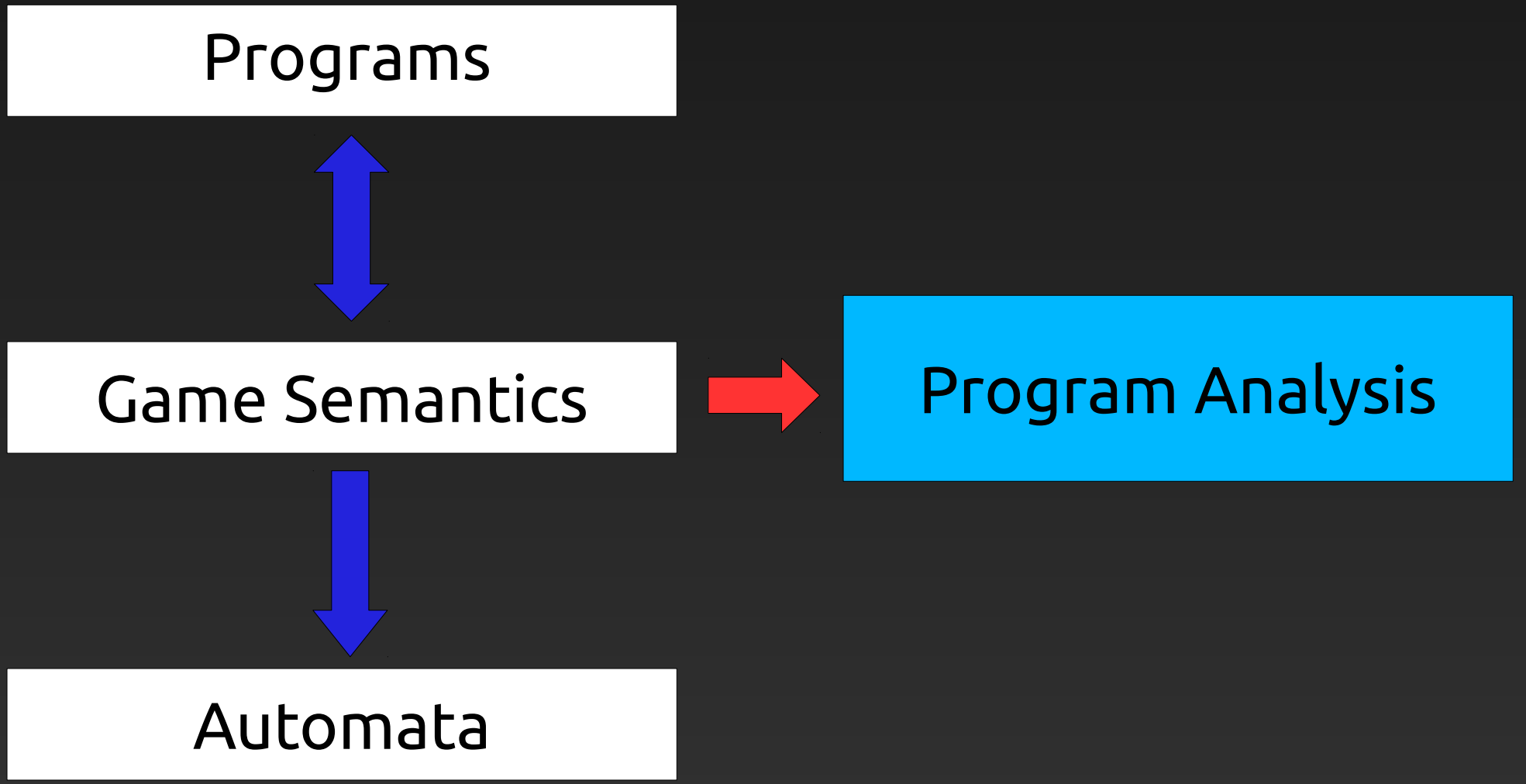
Automata

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

$$P \cong P' \iff A(P) \sim A(P')$$

- Mur. Tz. 12 : Algorithmic games for *full* intref's
- Mur. Tz. … : Algorithmic games for Java
- Grigore Tz. 13: History-Register Automata

Games with names



Games with names

thank you!