Games with names

Nikos Tzevelekos

University of Oxford

What this talk is about

Generation of new resources is a pervasive feature in computation

We present developments in game semantics for new resources

Computation with new resources

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

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

Computation with new resources

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

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

Computation with new resources

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

$\mathtt{let}\ f = [\_]\ \mathtt{in}\ \{\ f() == f()\ \} \longmapsto \mathtt{false}$

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references

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

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references

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

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

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

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references

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

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

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

$$f : \mathsf{intref} \to \mathsf{com} \ \vdash \ \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \mathsf{let}\, y = \mathsf{ref}\, (0)\, \mathsf{in}\, f(x); (y := !x); y$$
$$\cong \quad \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, f(x); x \ : \mathsf{intref}$$

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.())`
- In Idealized Algol:
  - no notion of *reference equality test*
  - spurious non-equivalences:

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

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

Bad variables

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

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

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

In call-by-name IA:

- Equivalence *not* affected by `mkvar`
- Approximation *is* affected

Both affected in:

- CBN IA + control
- CBN IA + non-det.
- call-by-value IA

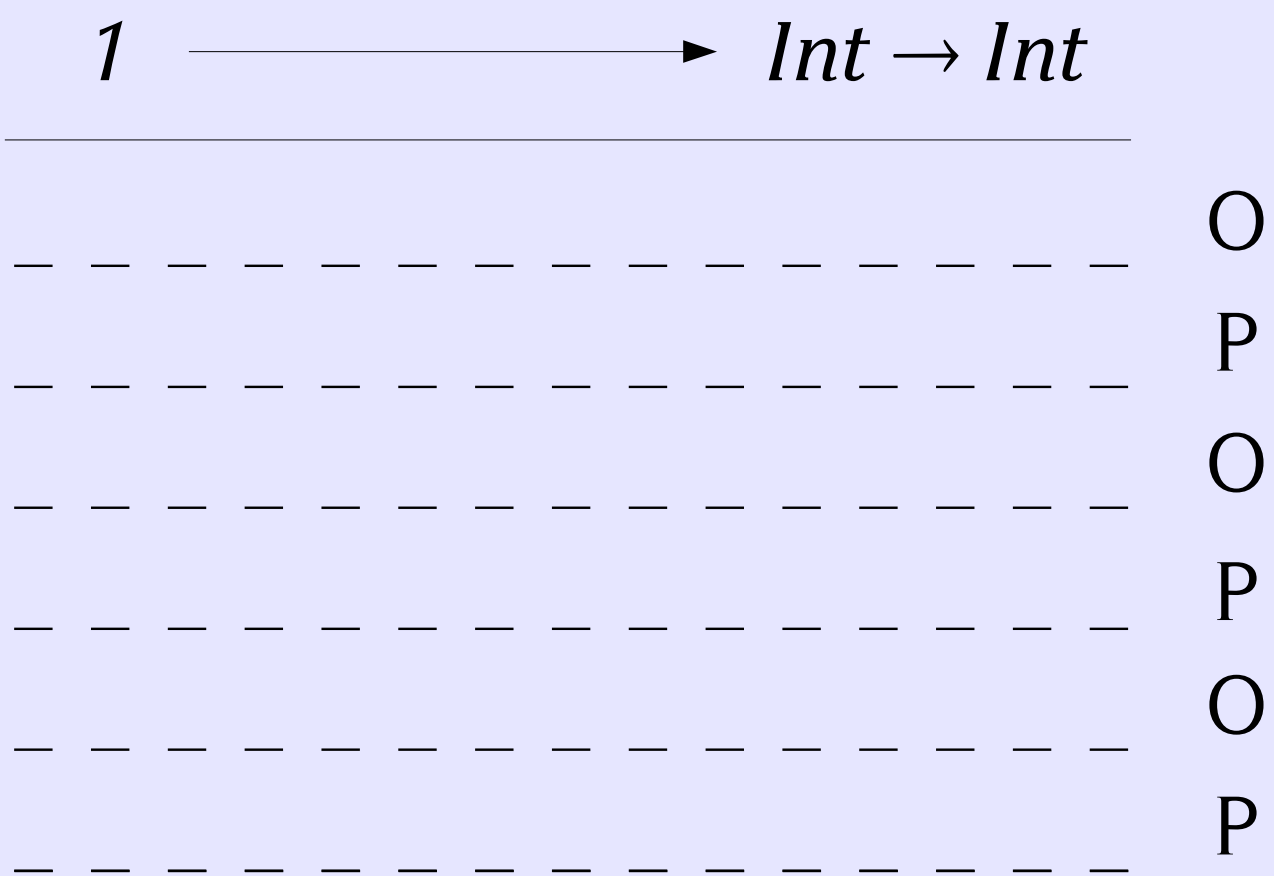
Game Semantics

Game Semantics

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

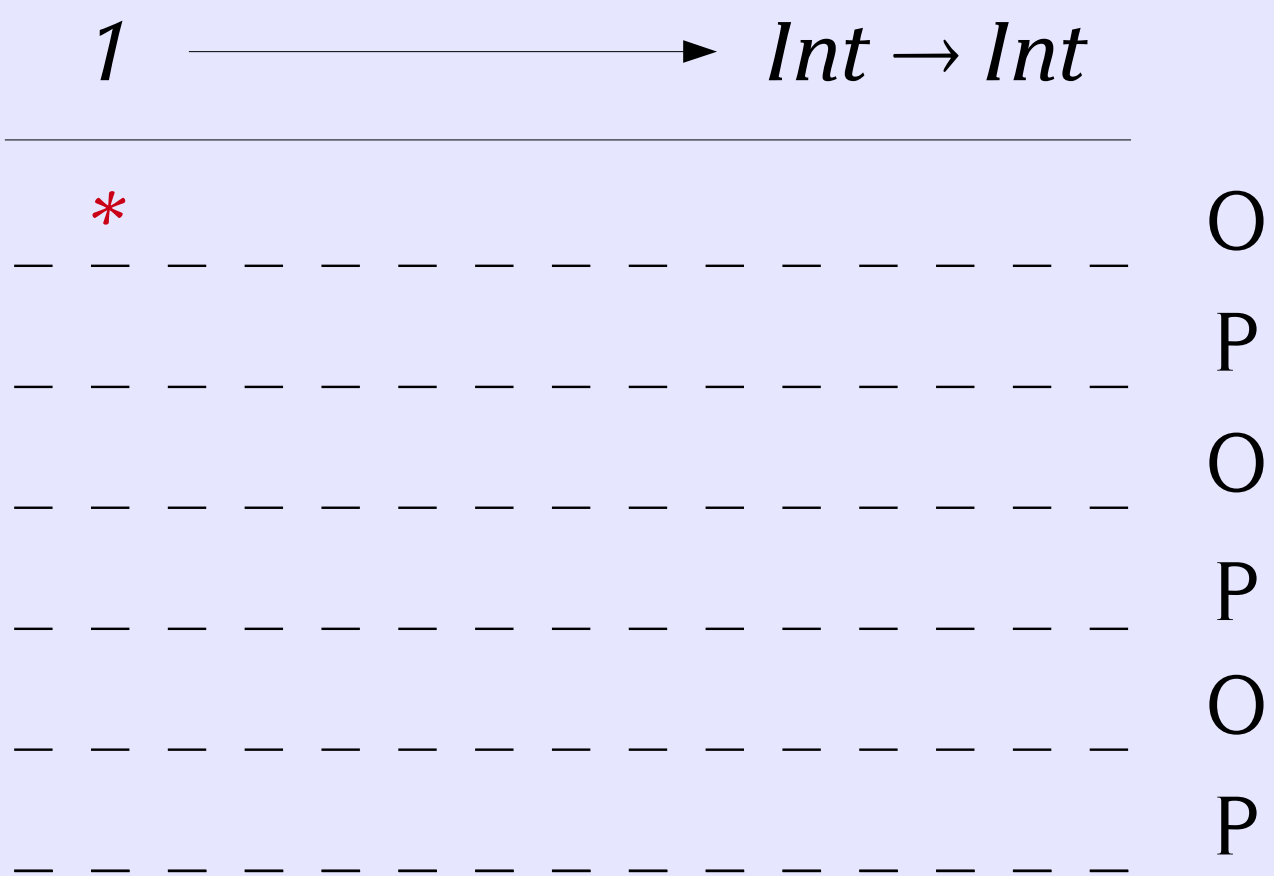
Example

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



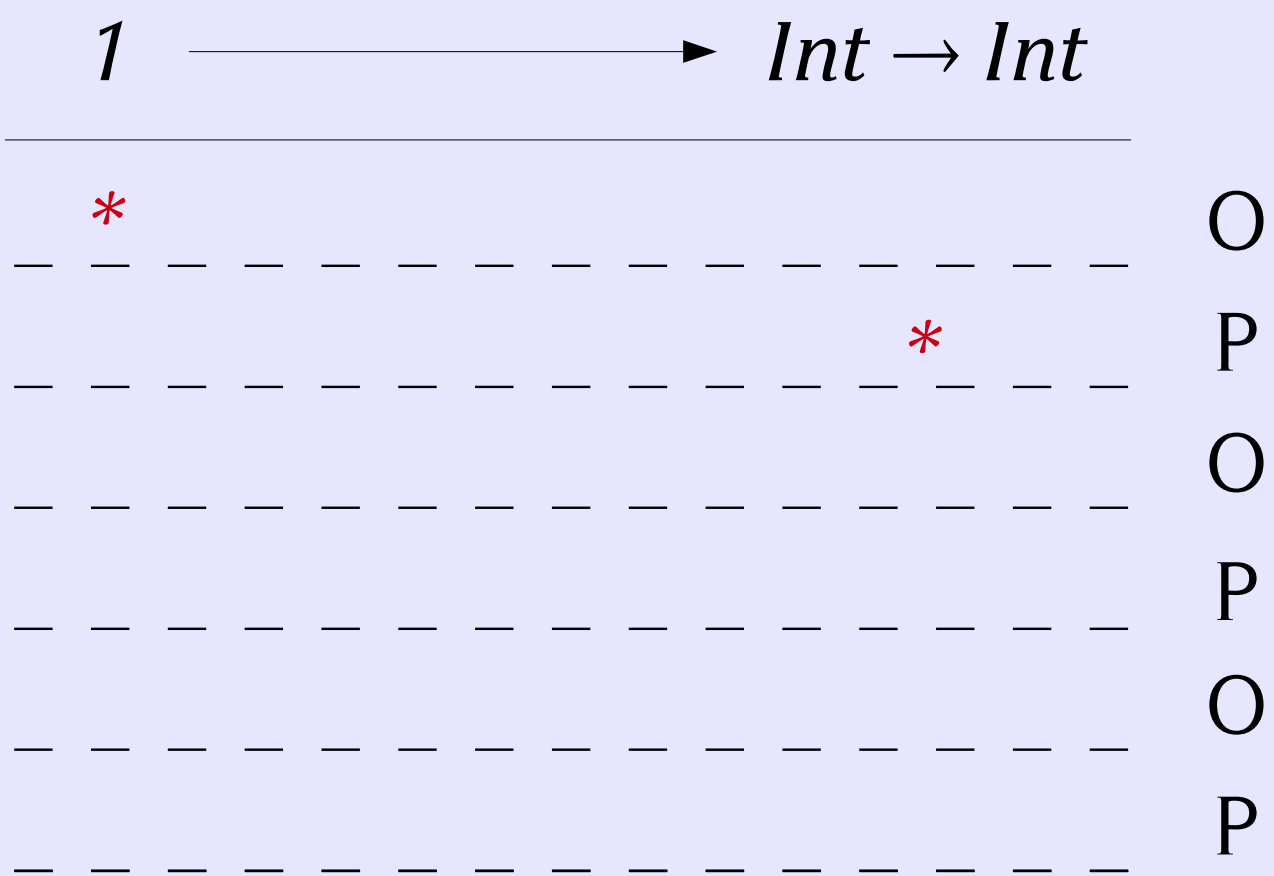
Example

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



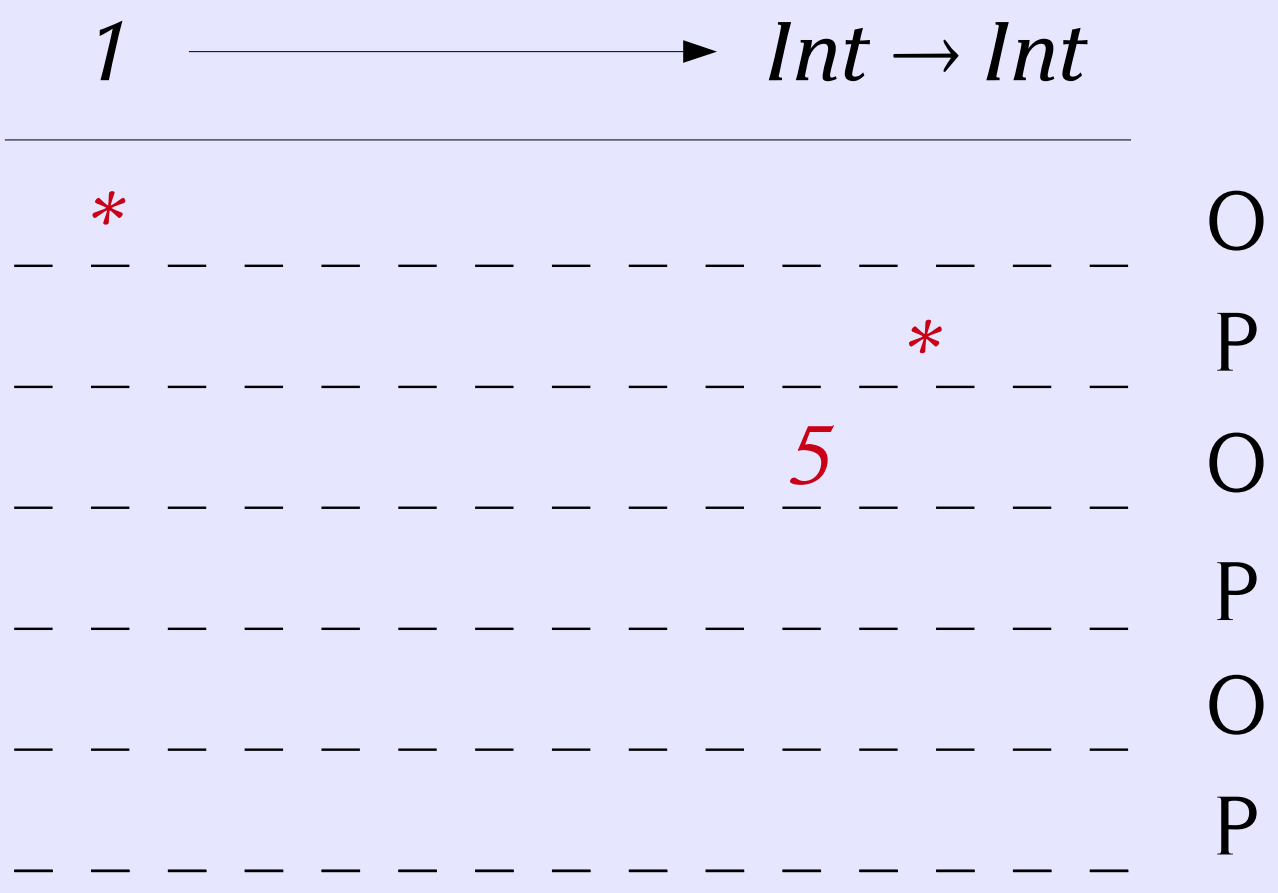
Example

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



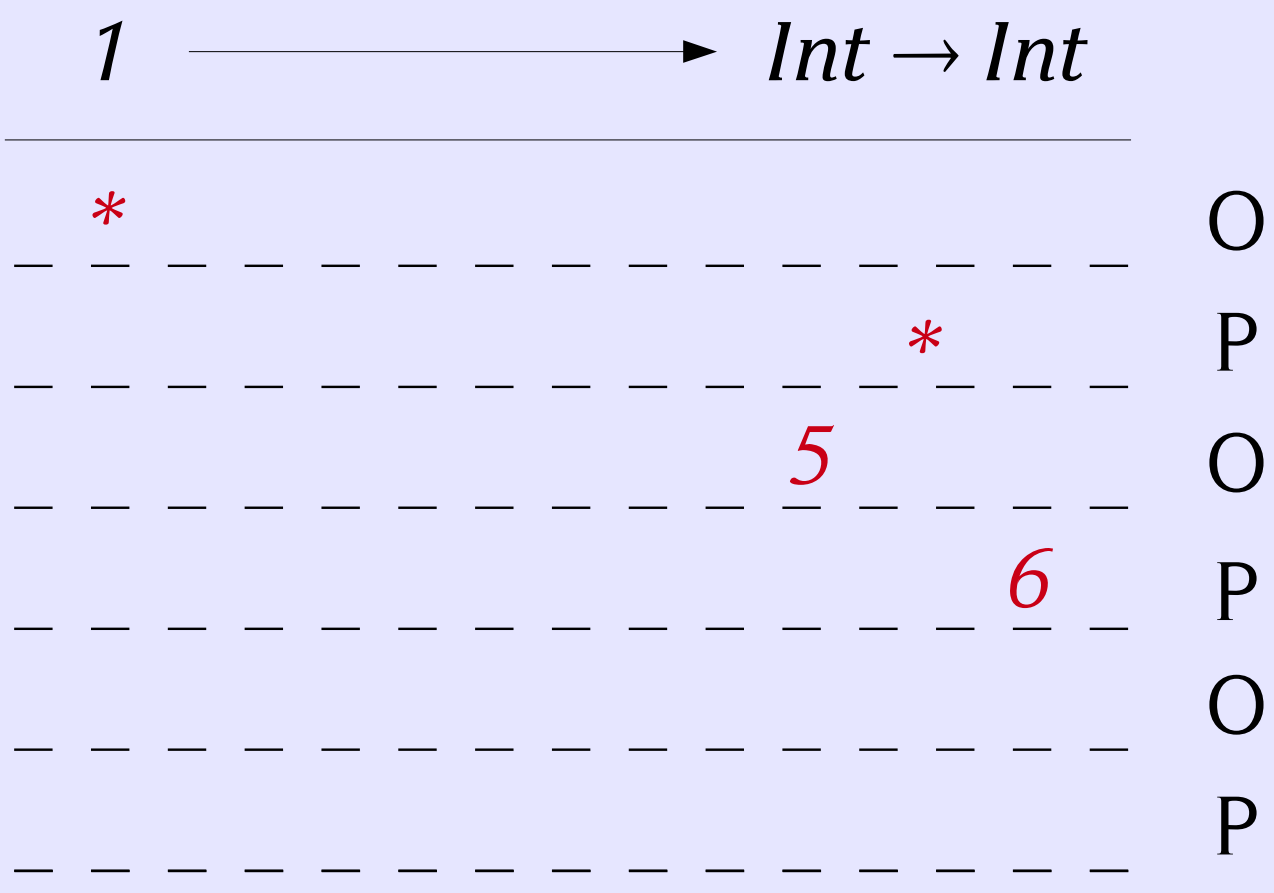
Example

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



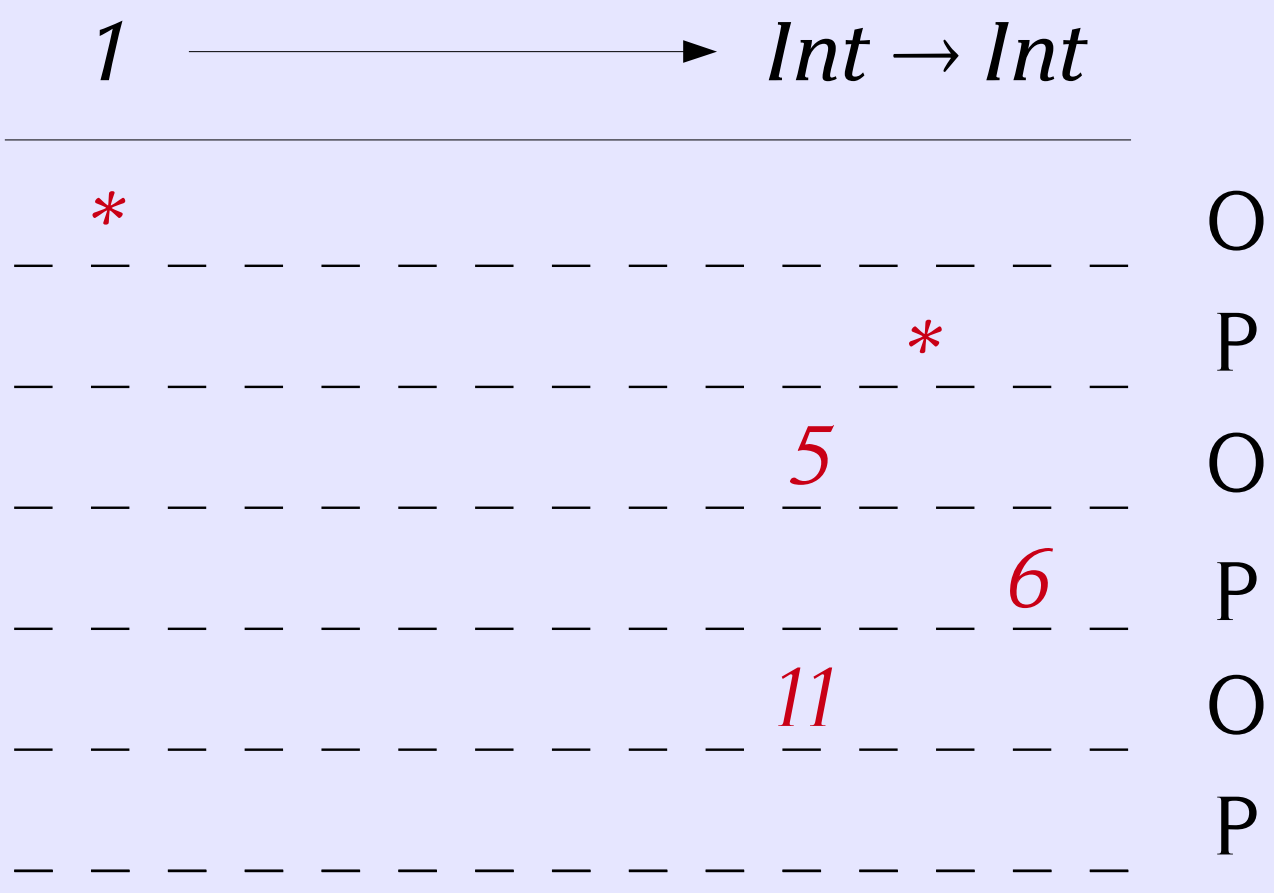
Example

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



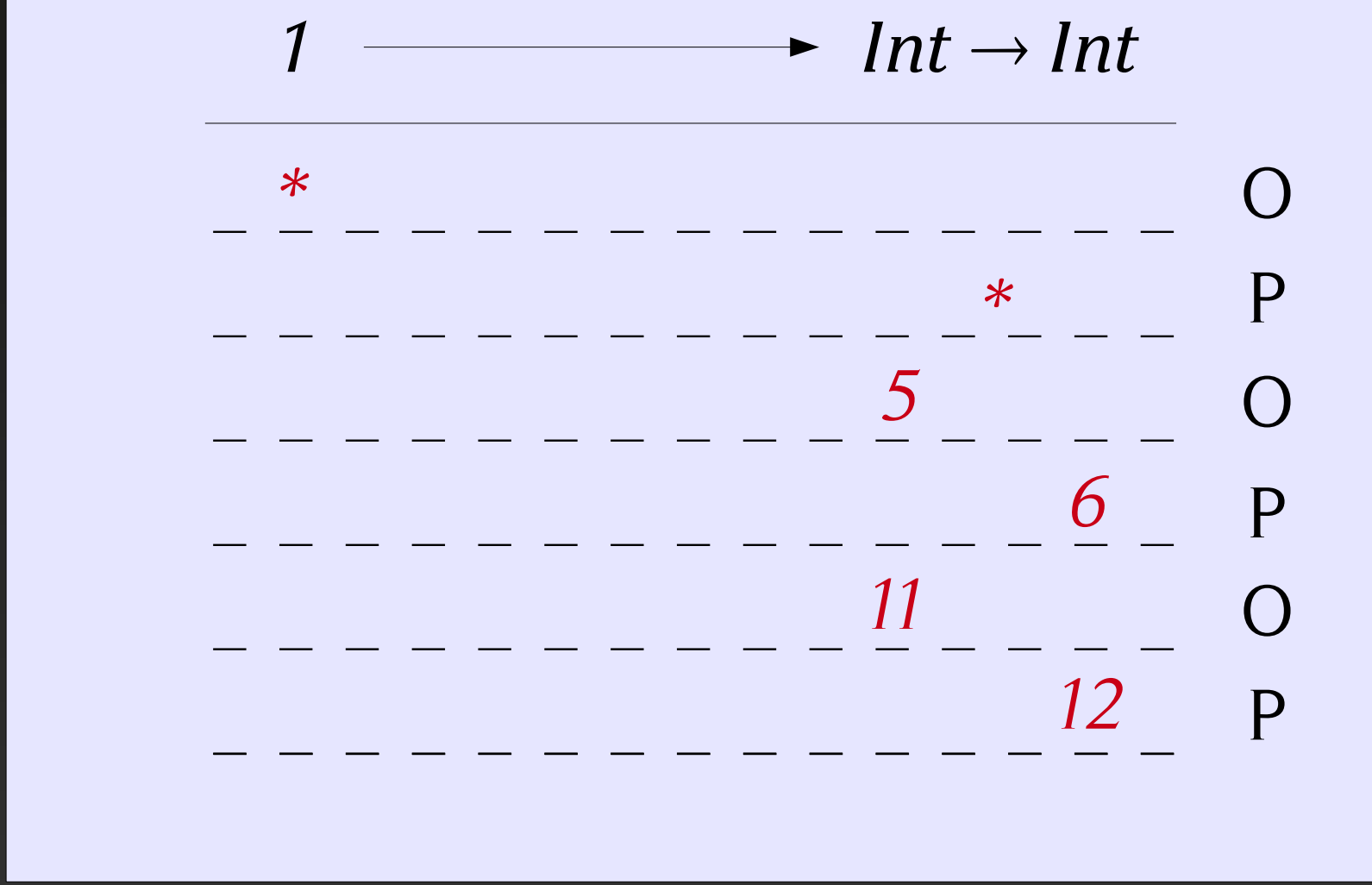
Example

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



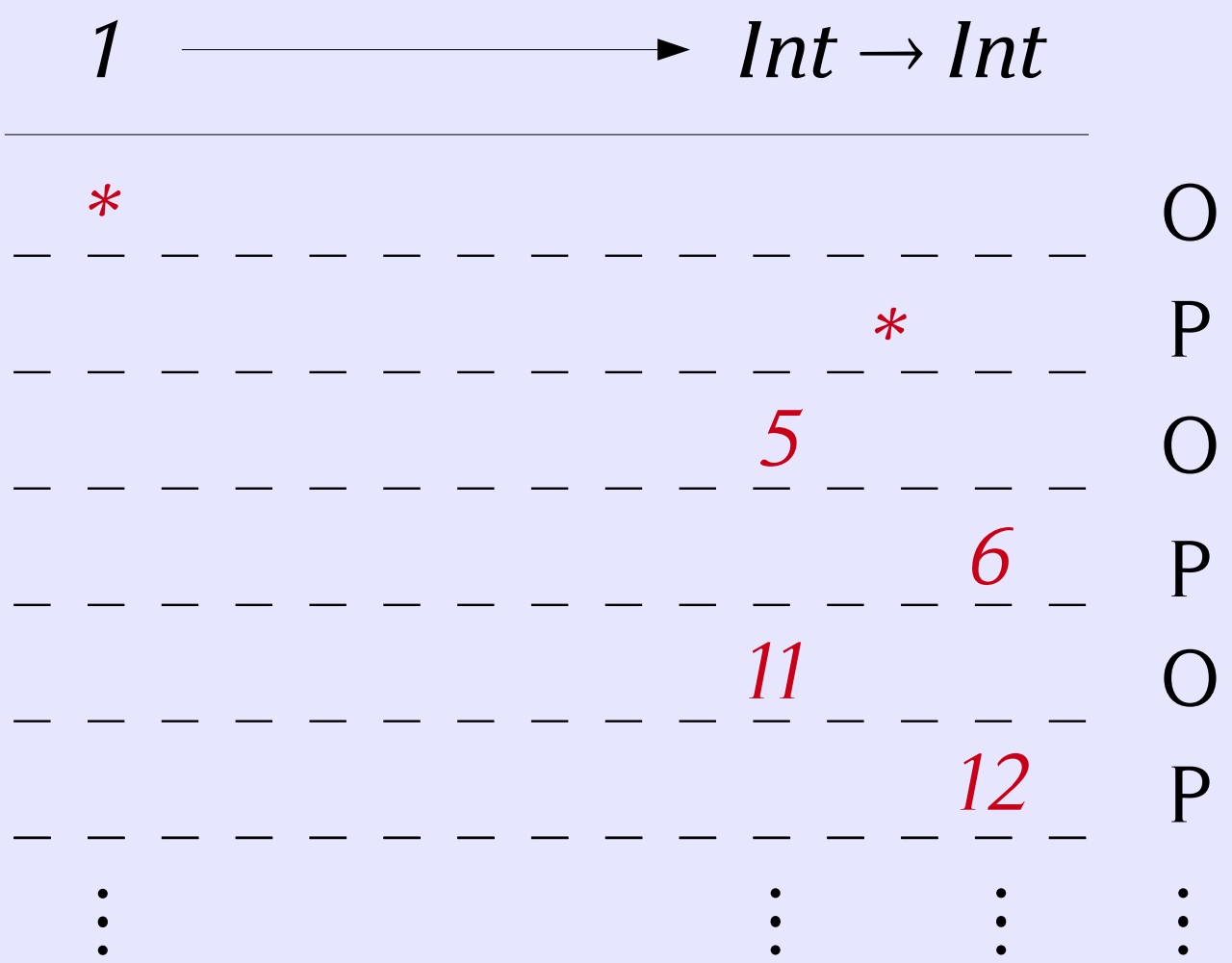
Example

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



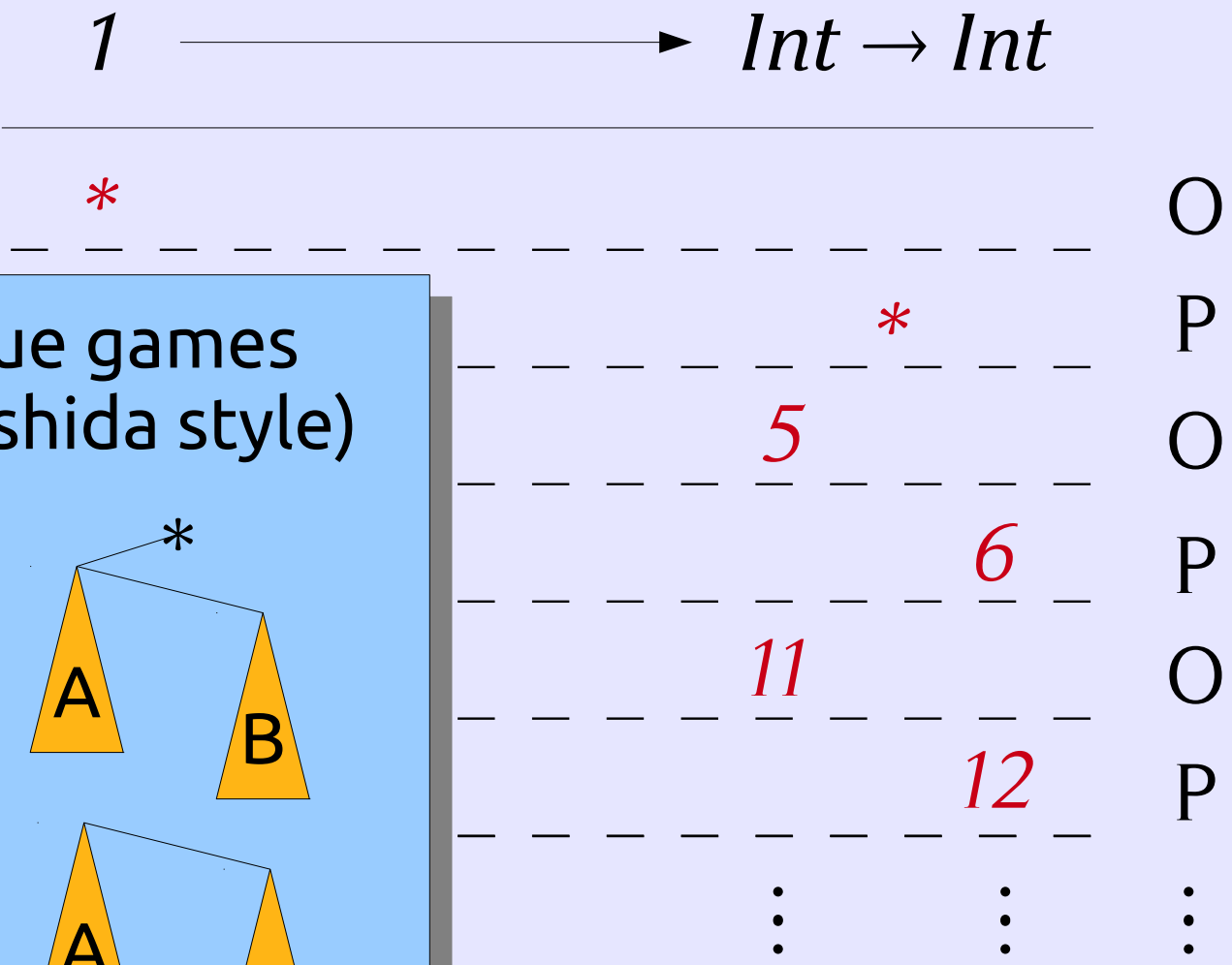
Example

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



Example

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



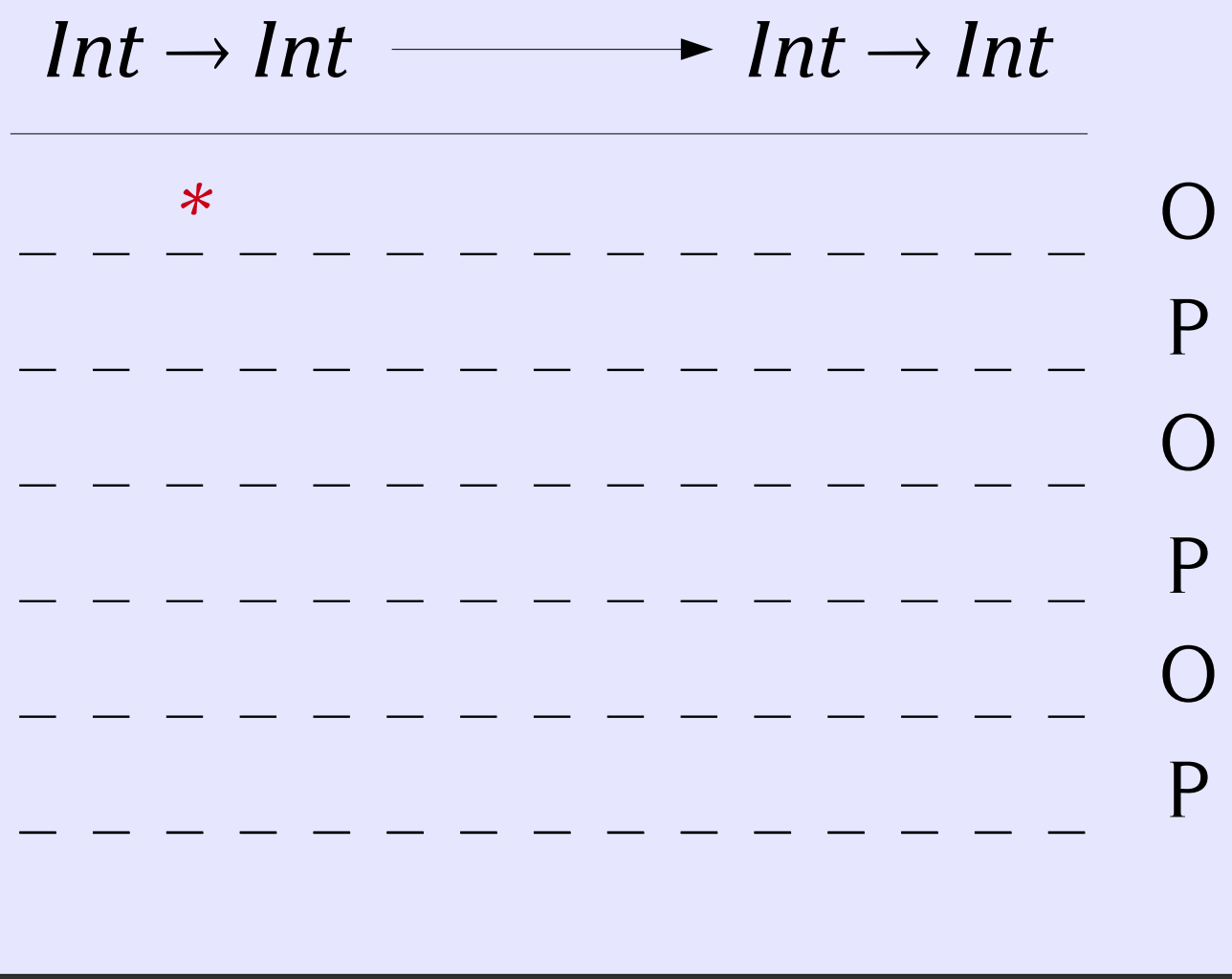
Call-by-value games (Honda-Yoshida style)

$A \to B$:

$A \to B$:

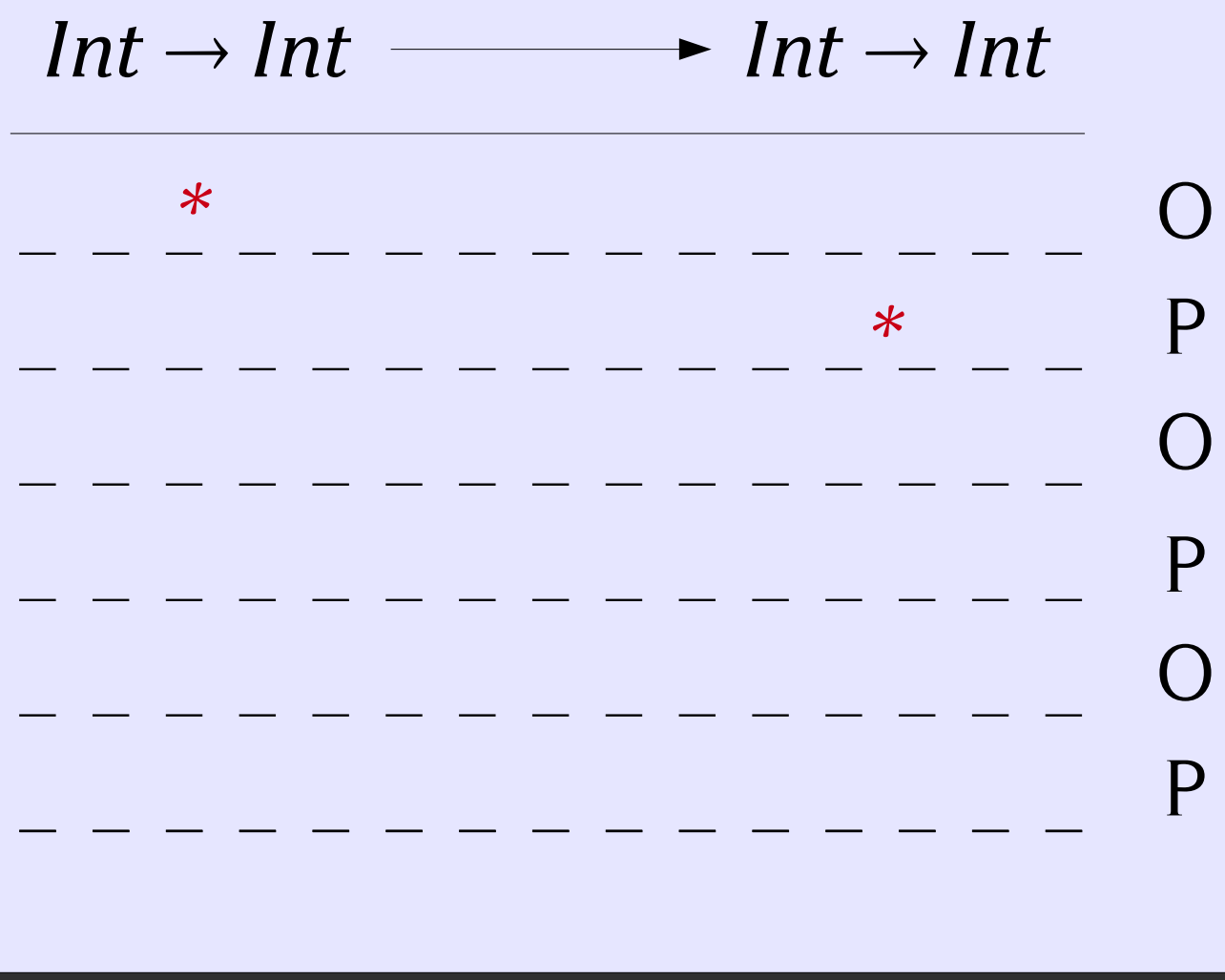
Example

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



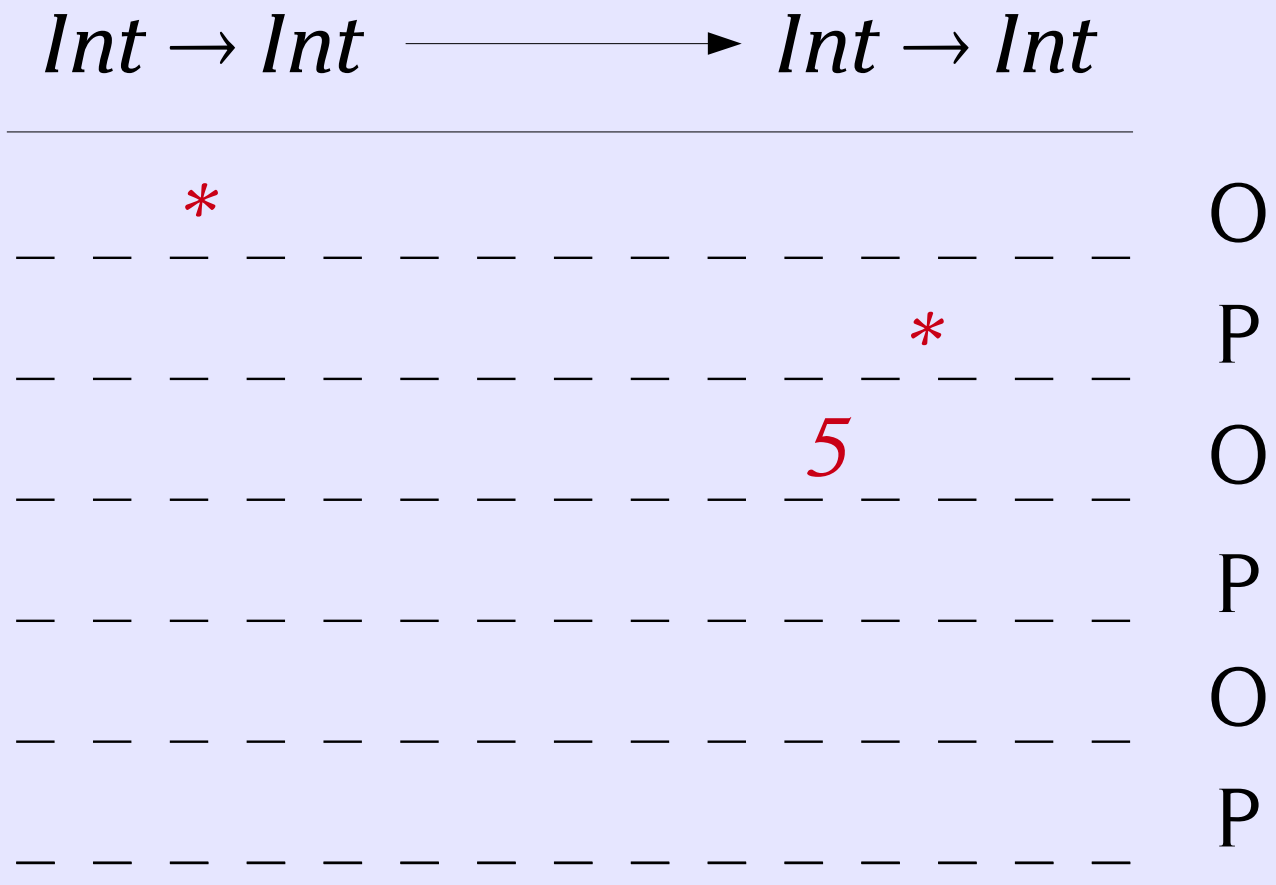
Example

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



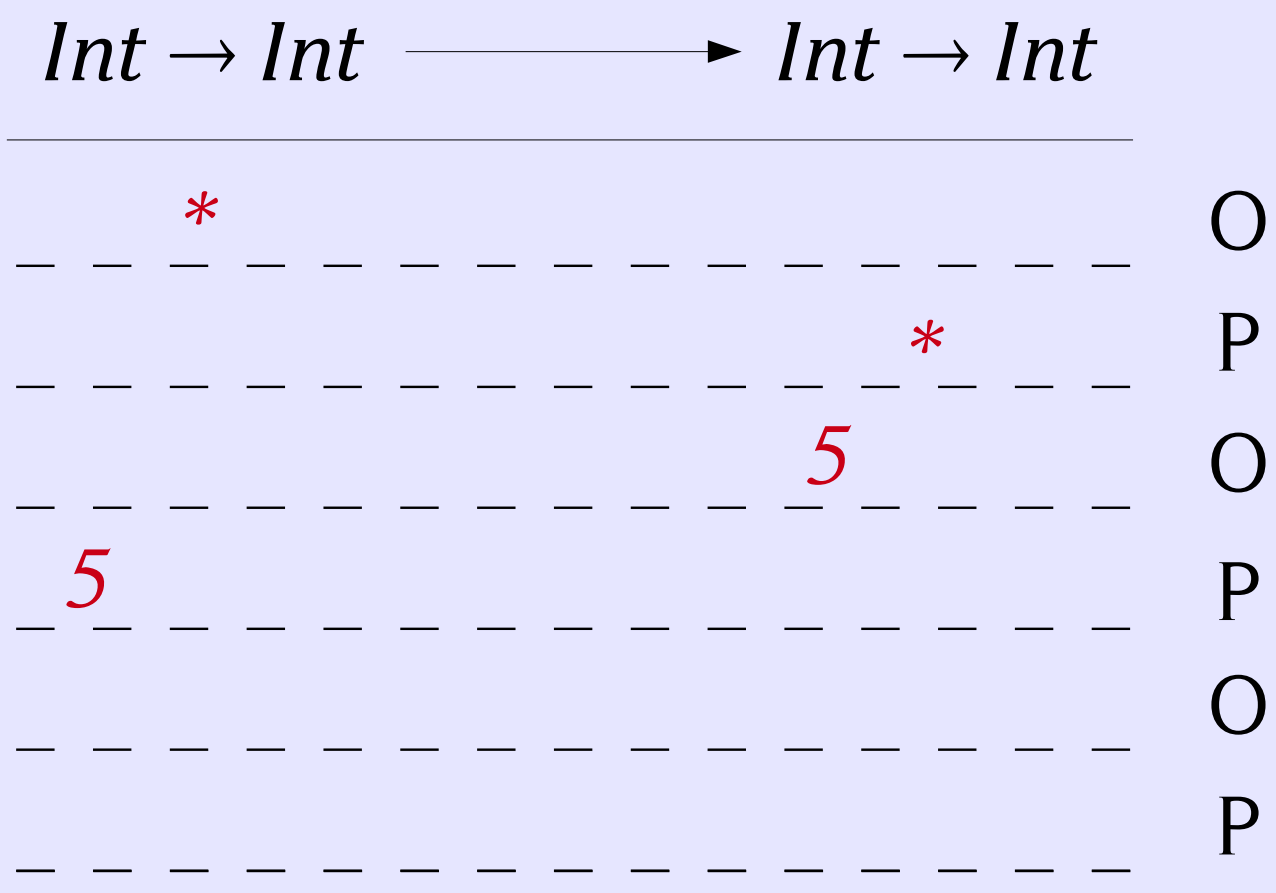
Example

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



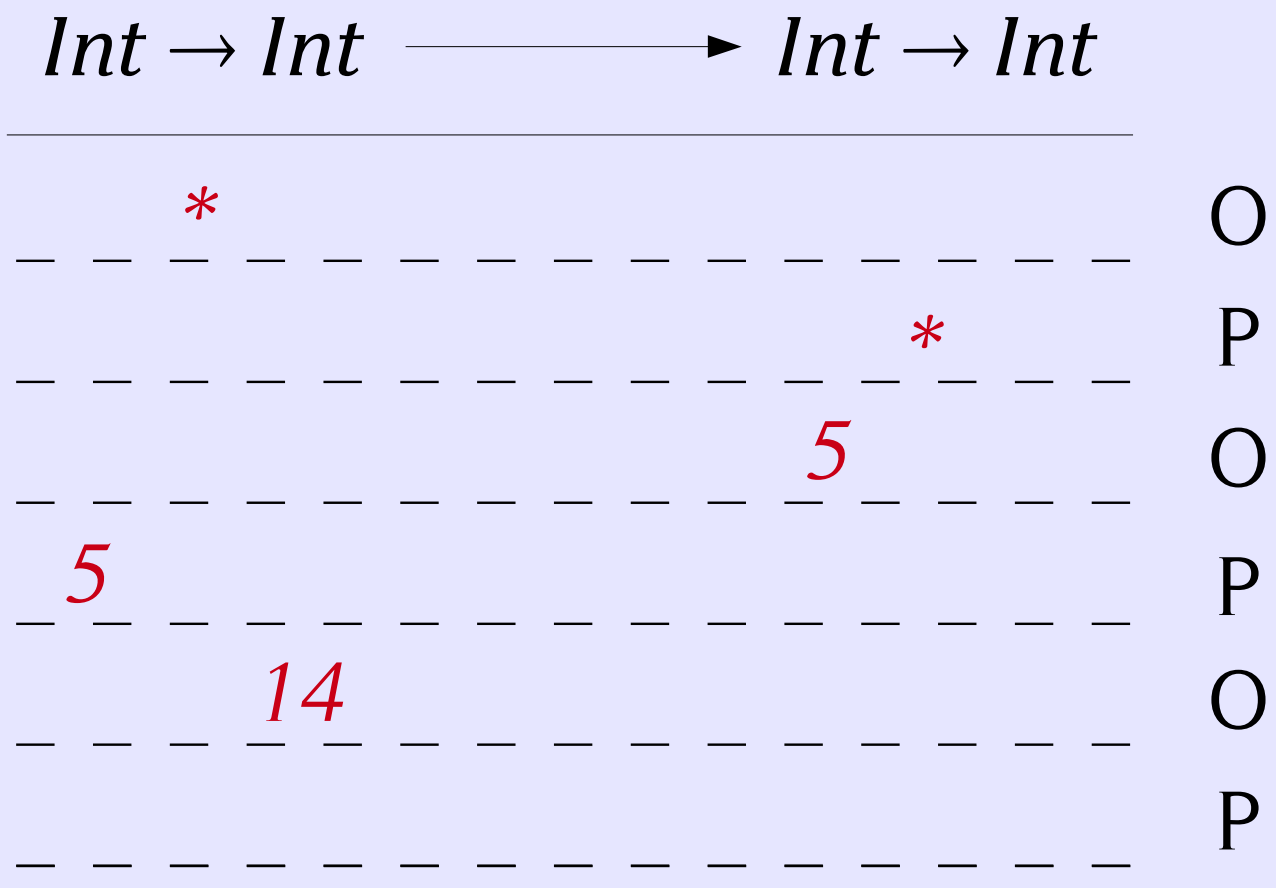
Example

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



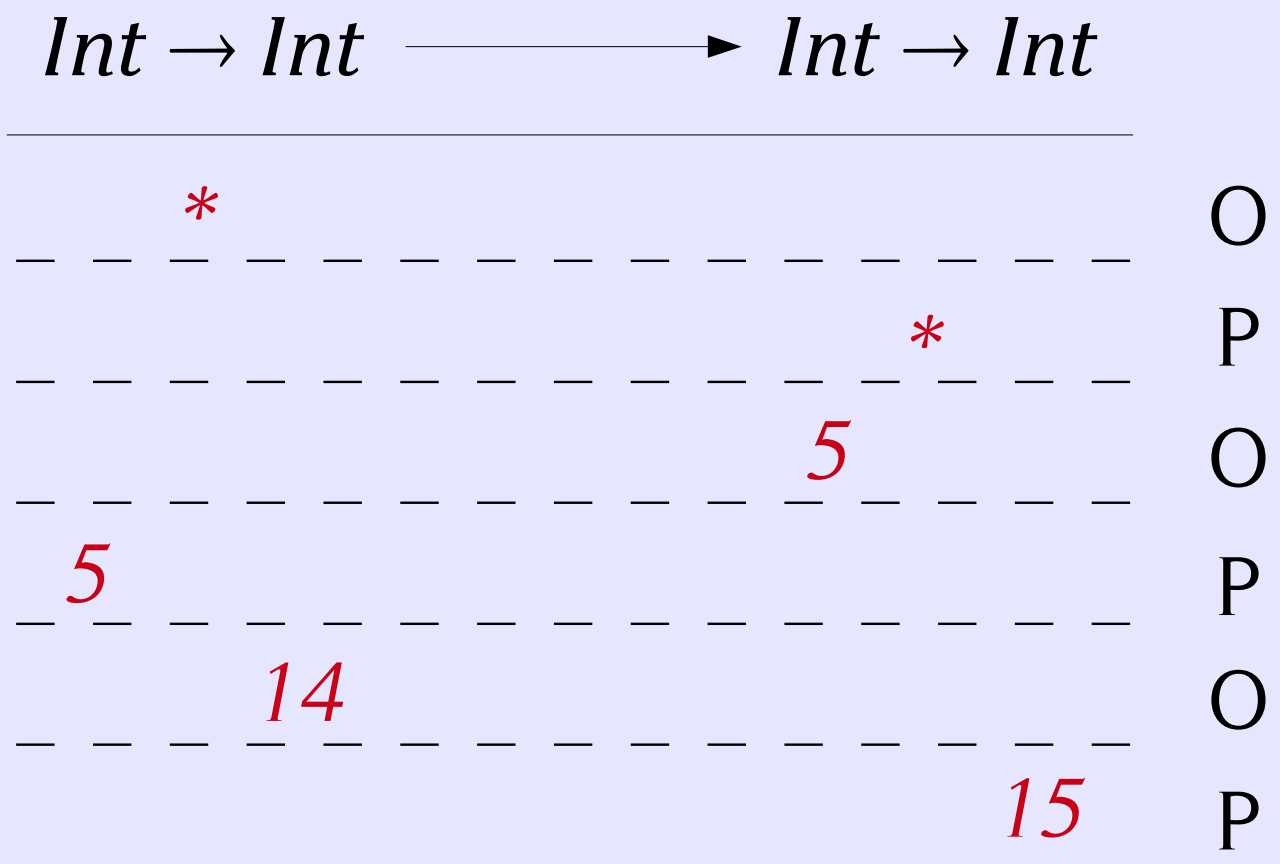
Example

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



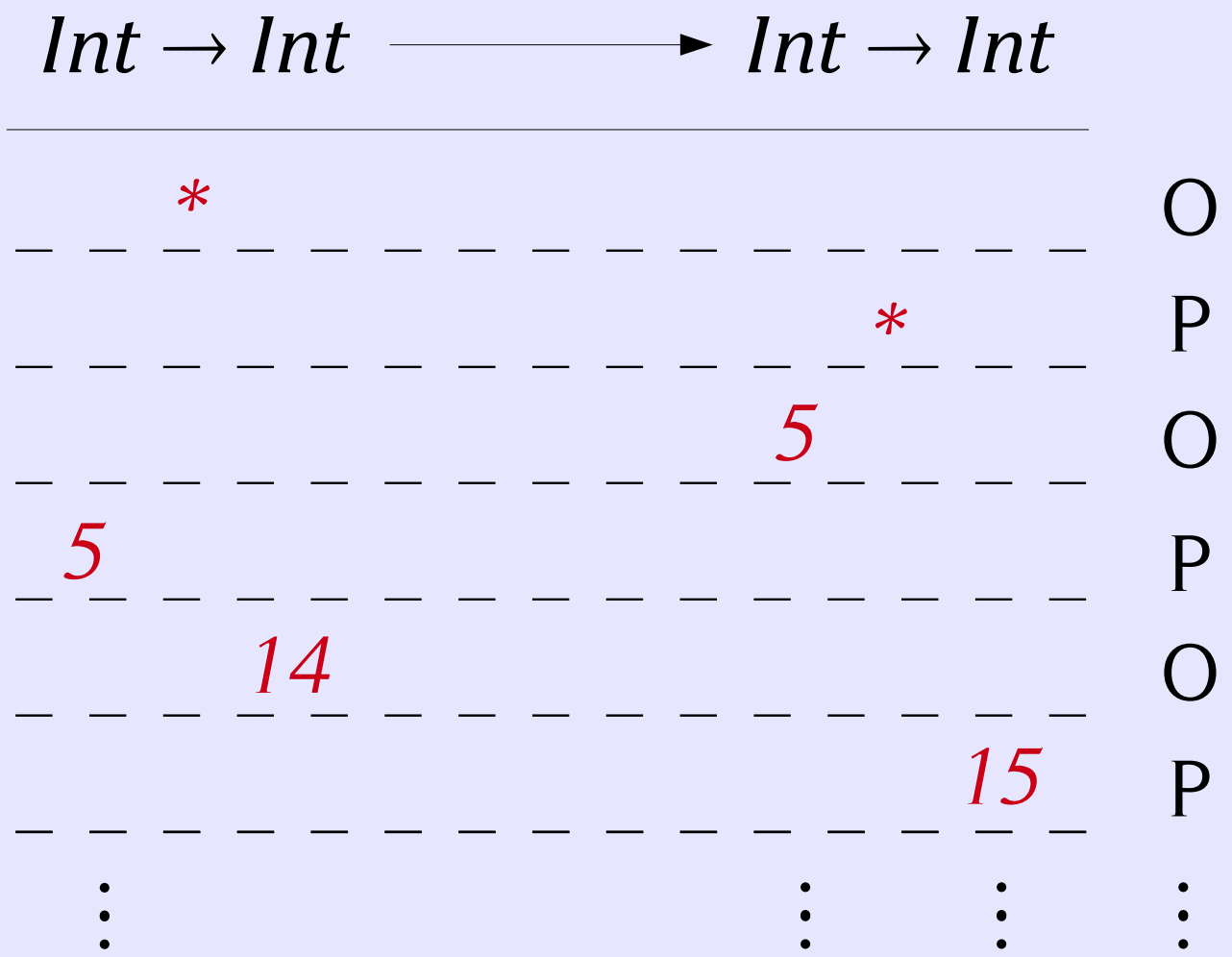
Example

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

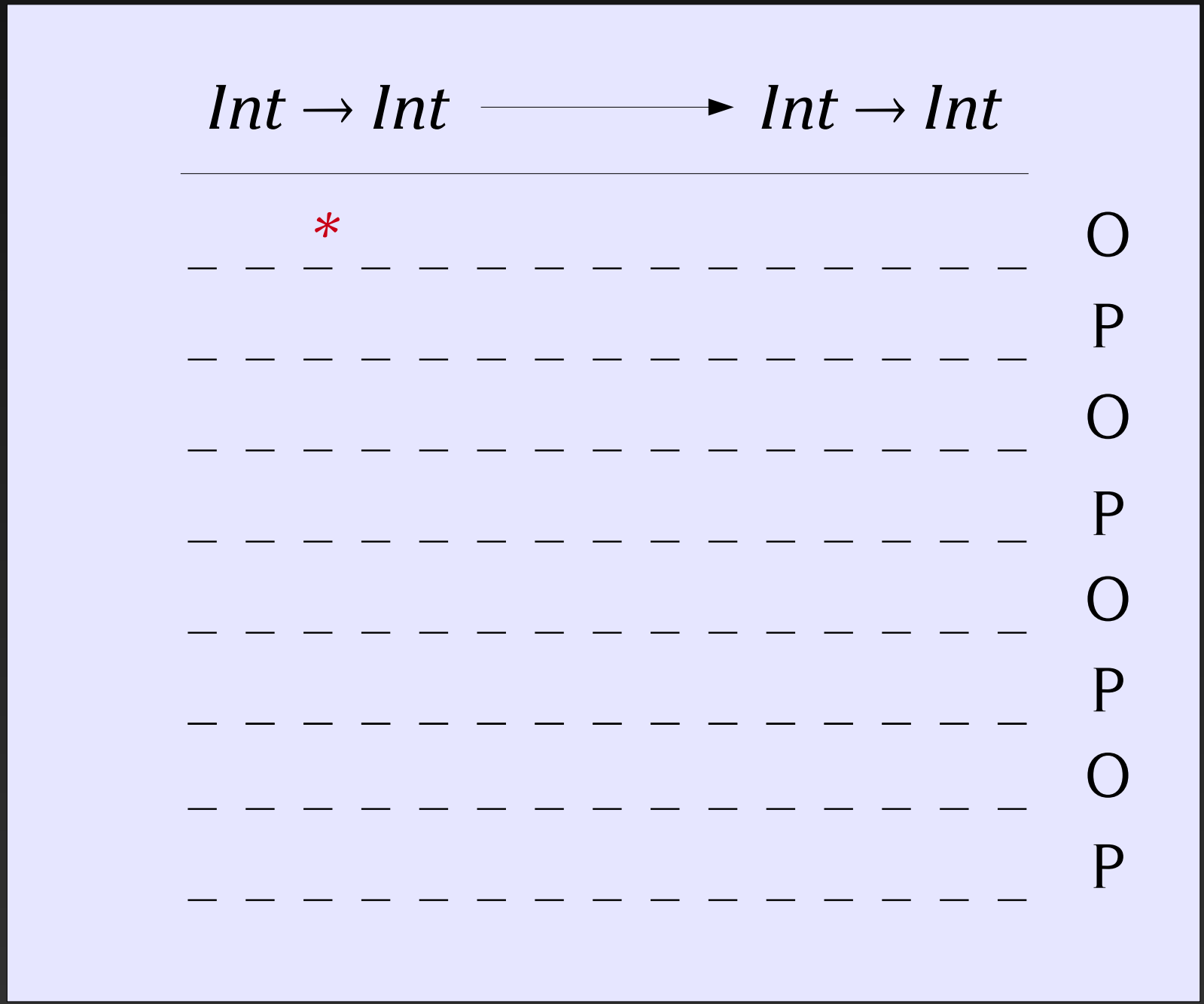


Example

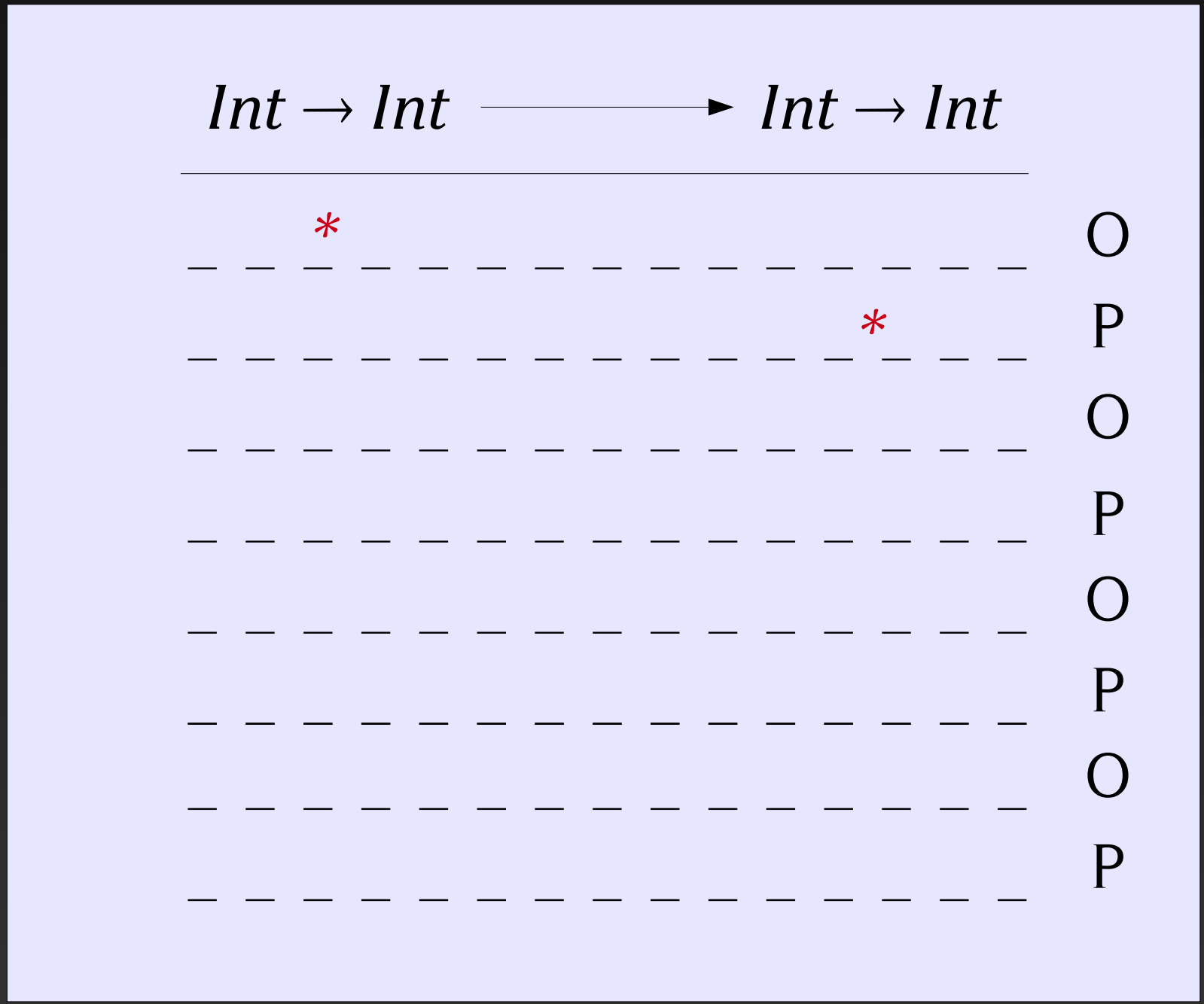
$\texttt{f : int} \to \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \to \texttt{int}$



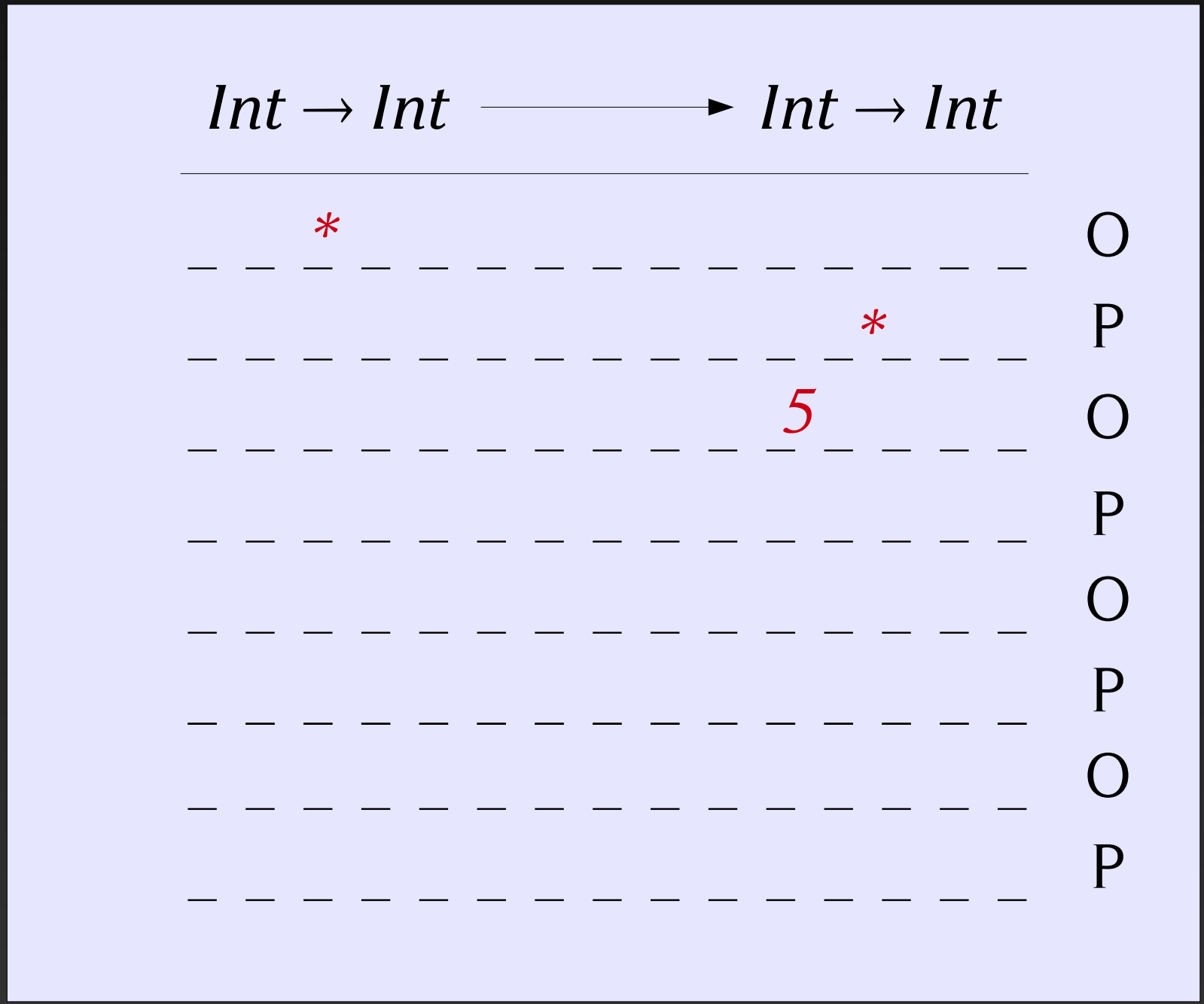
Example



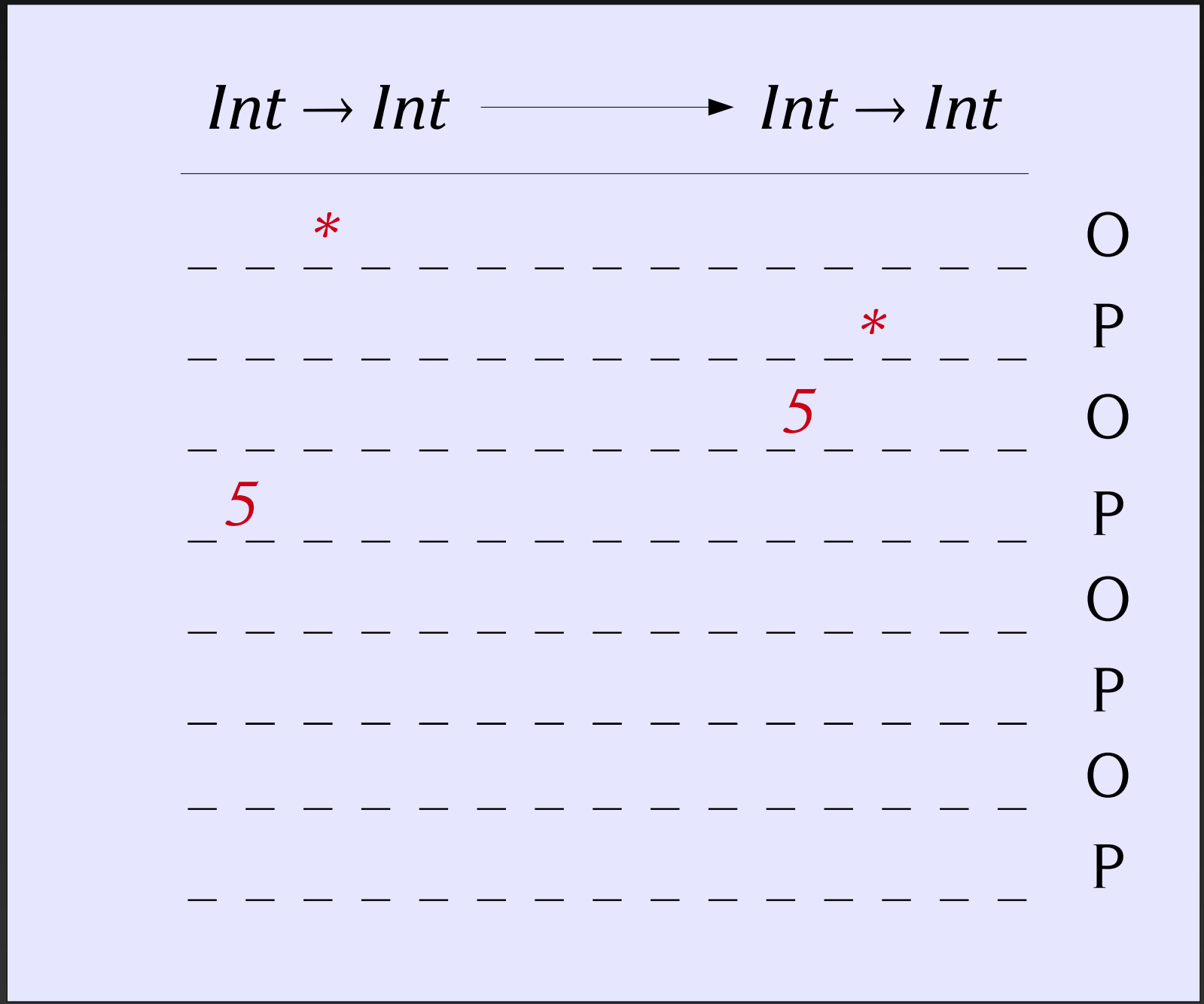
Example



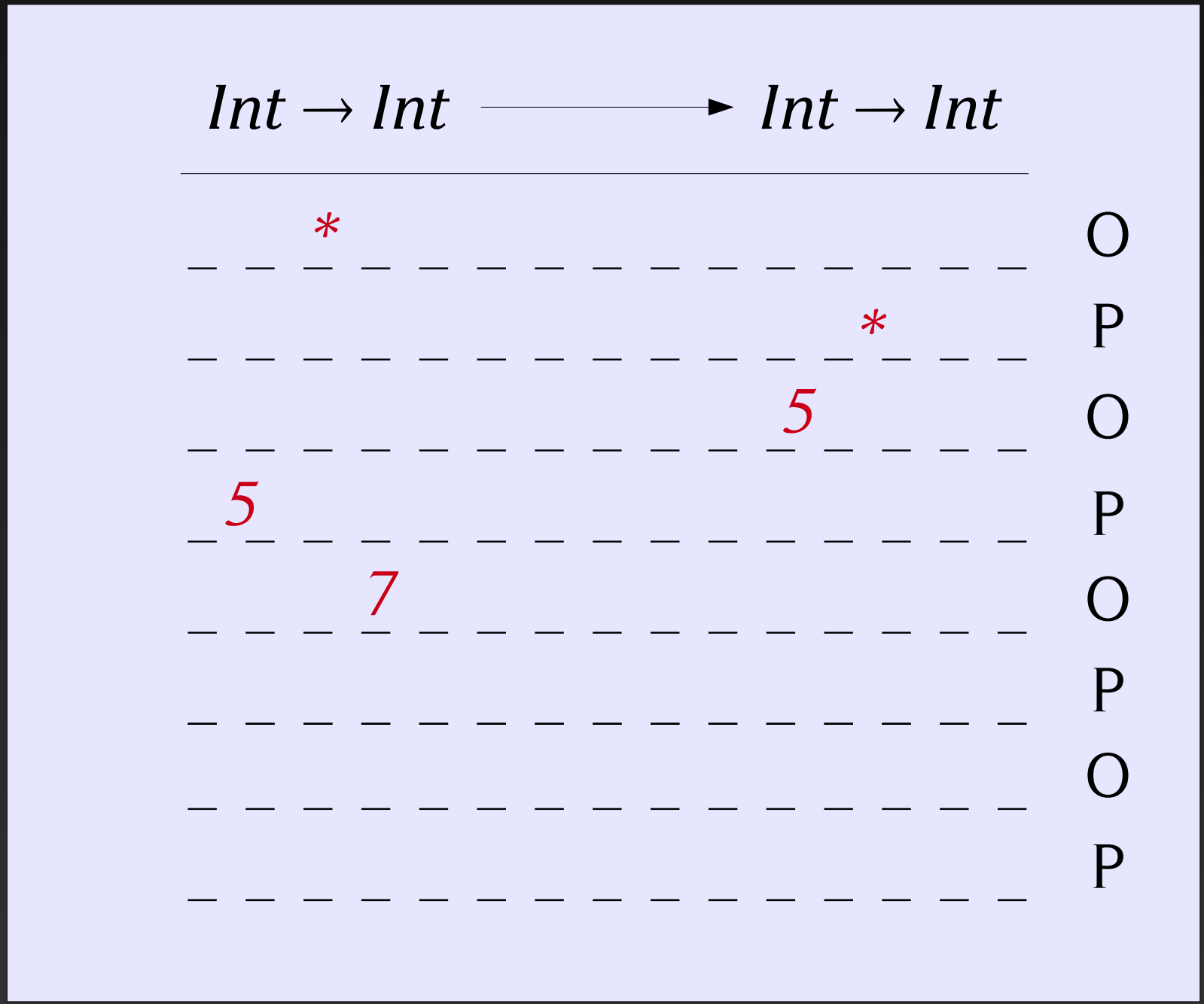
Example



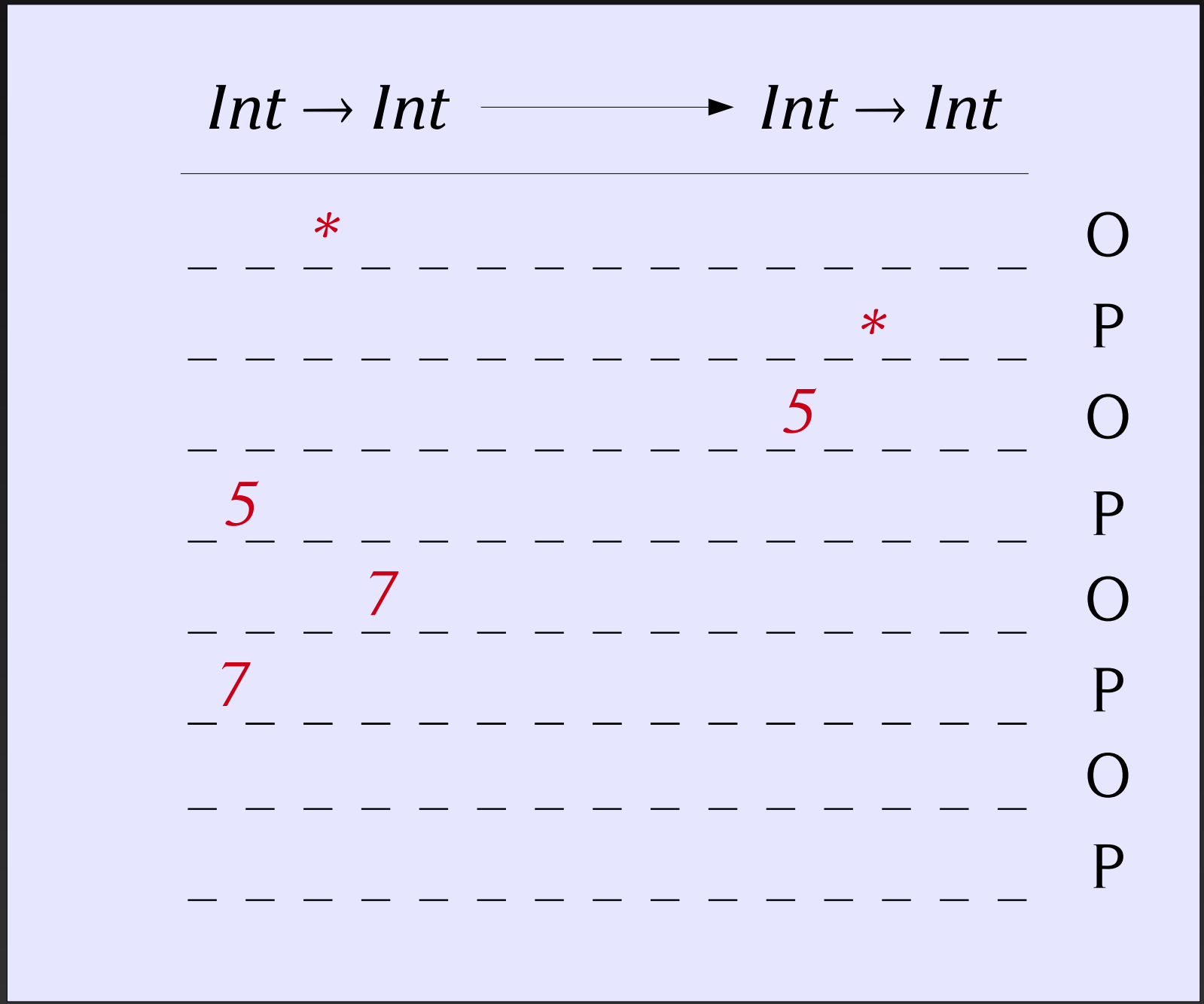
Example



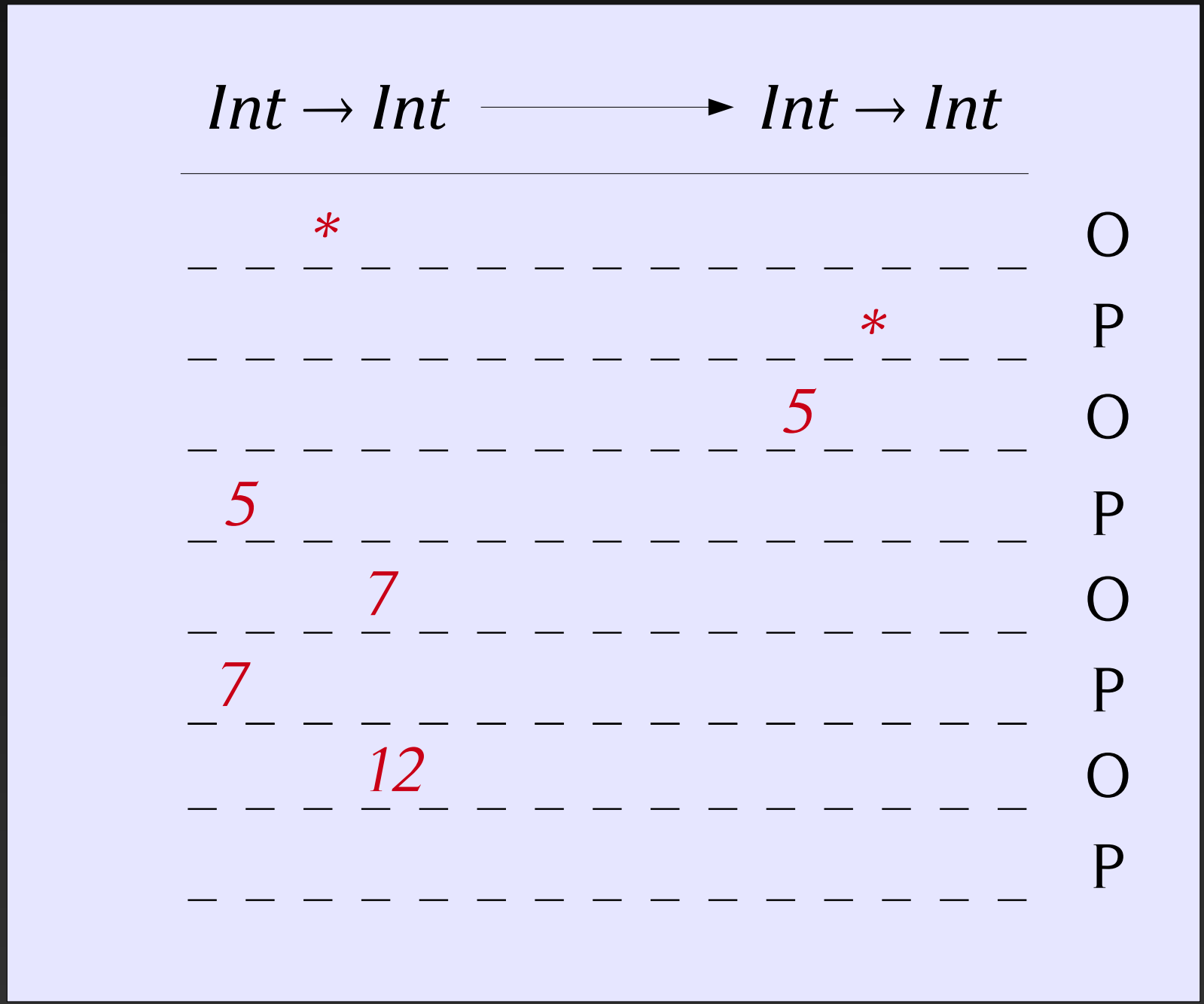
Example



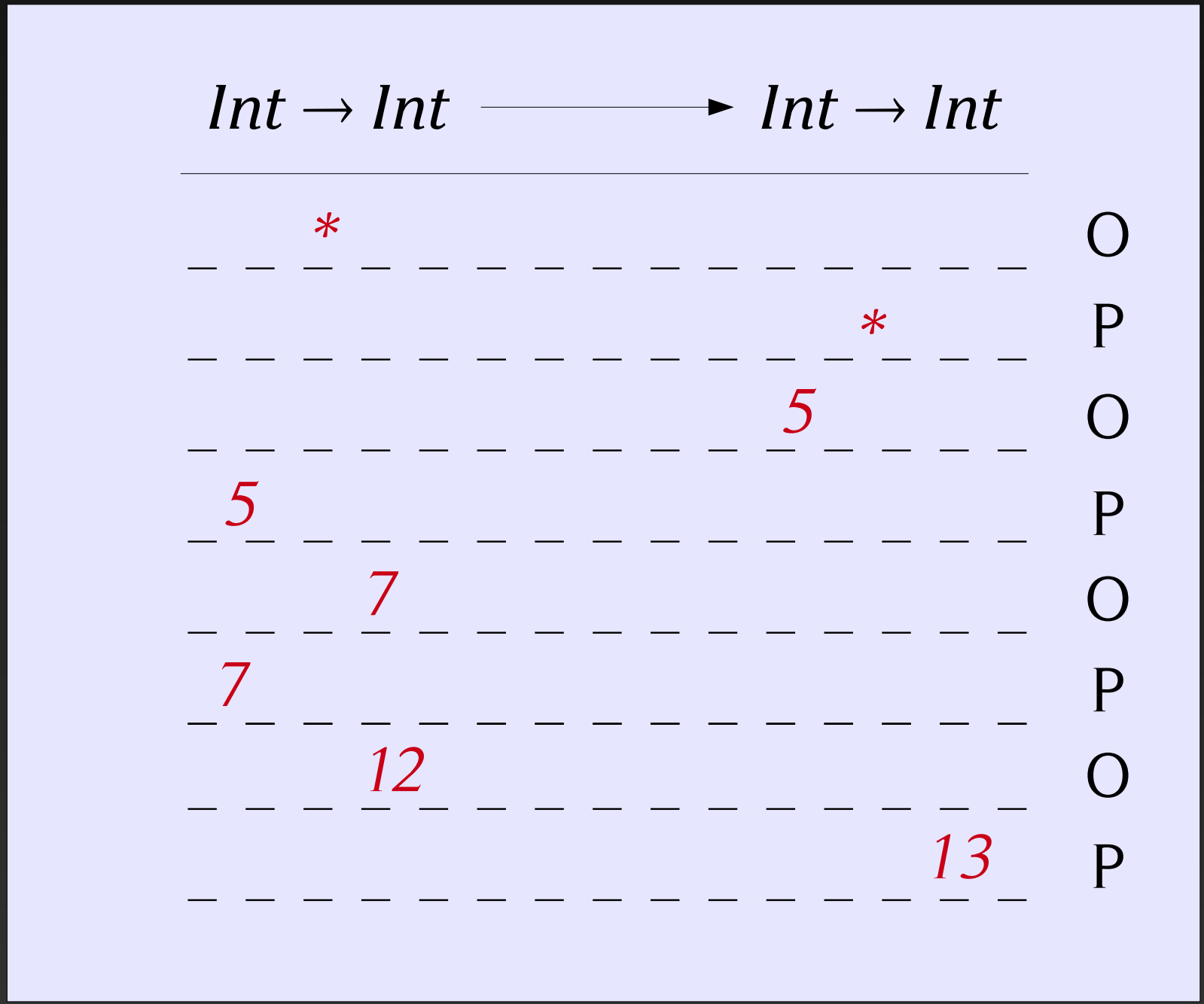
Example



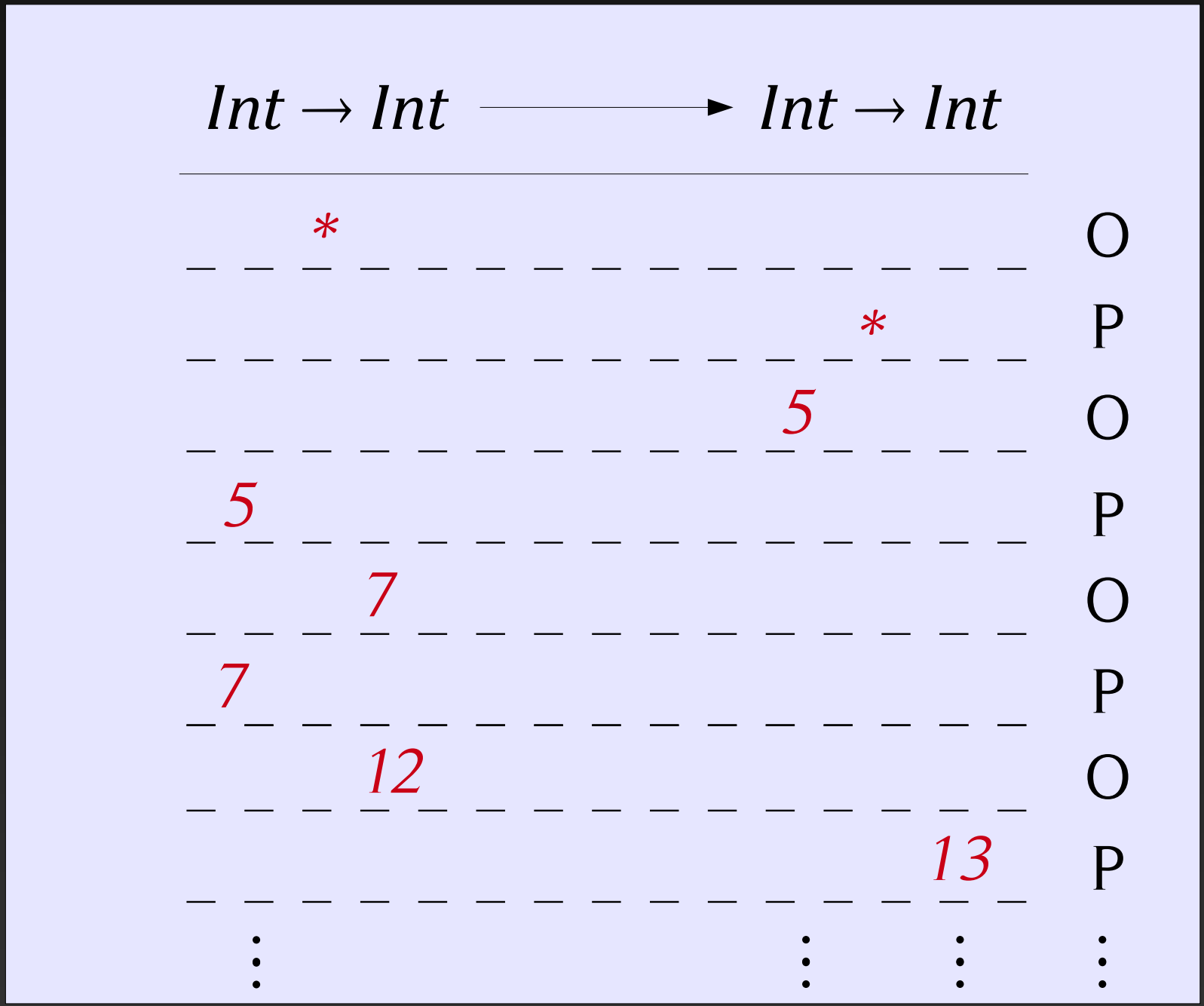
Example



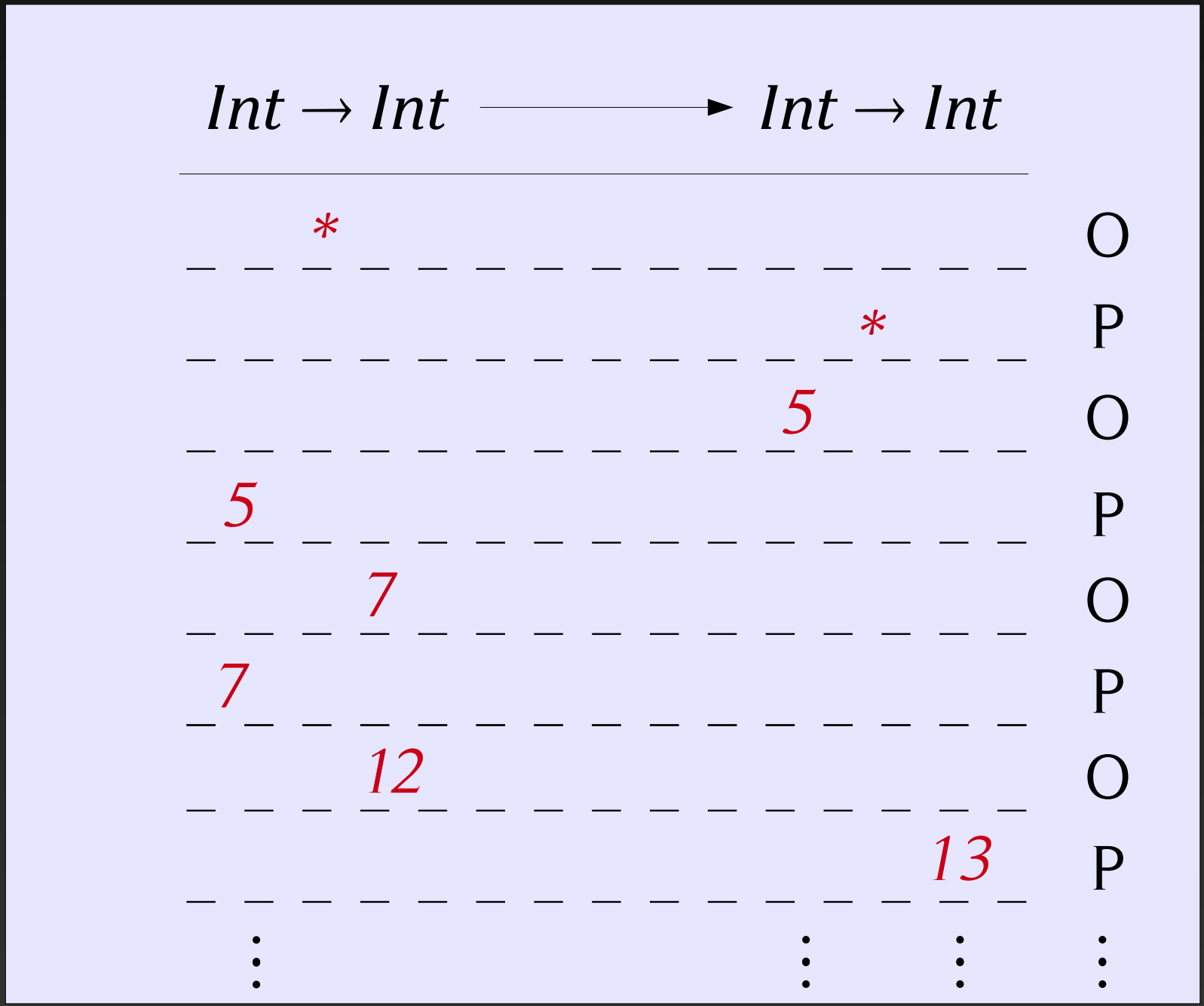
Example



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

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

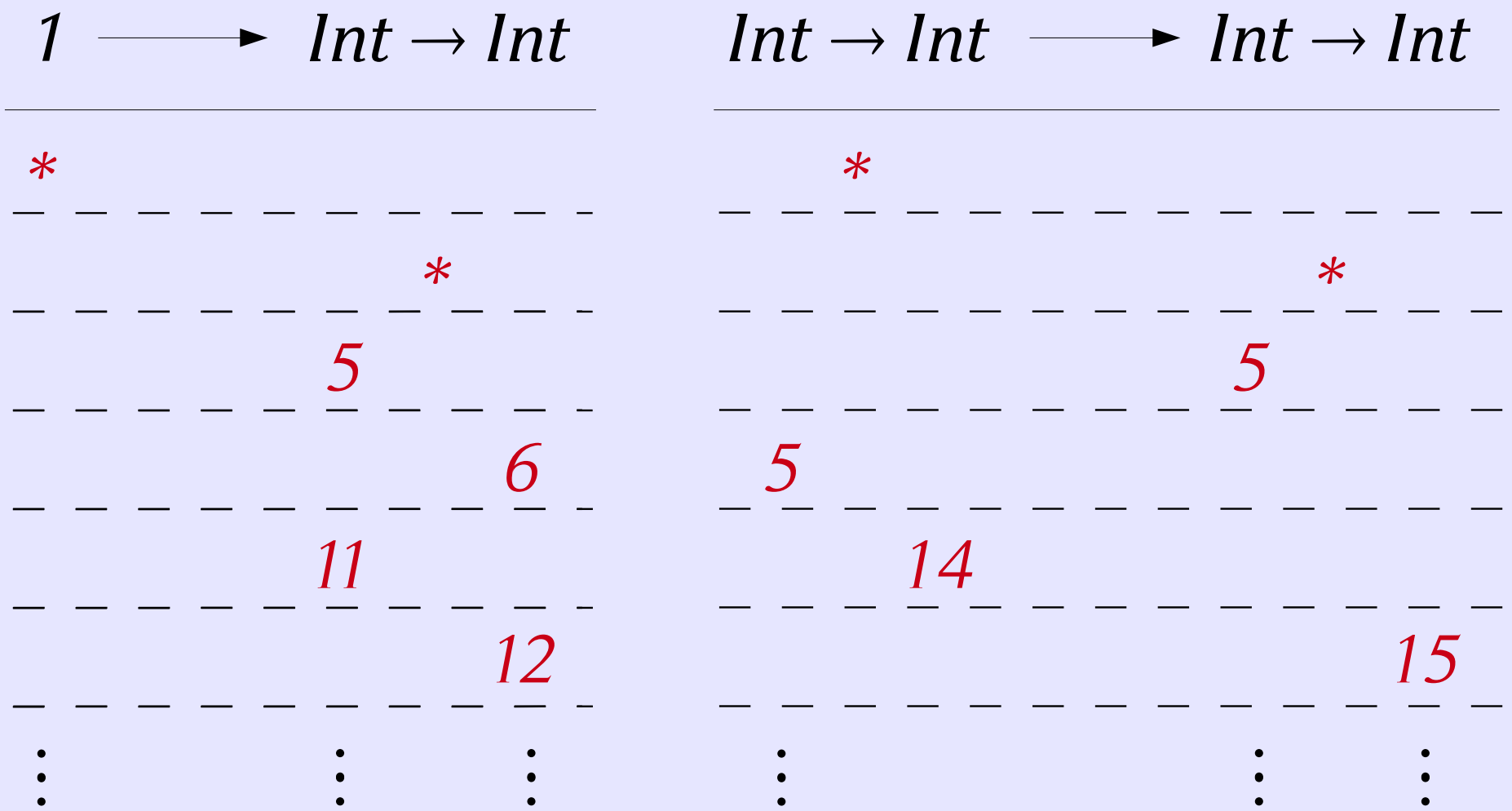
$f : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

Composition

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

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

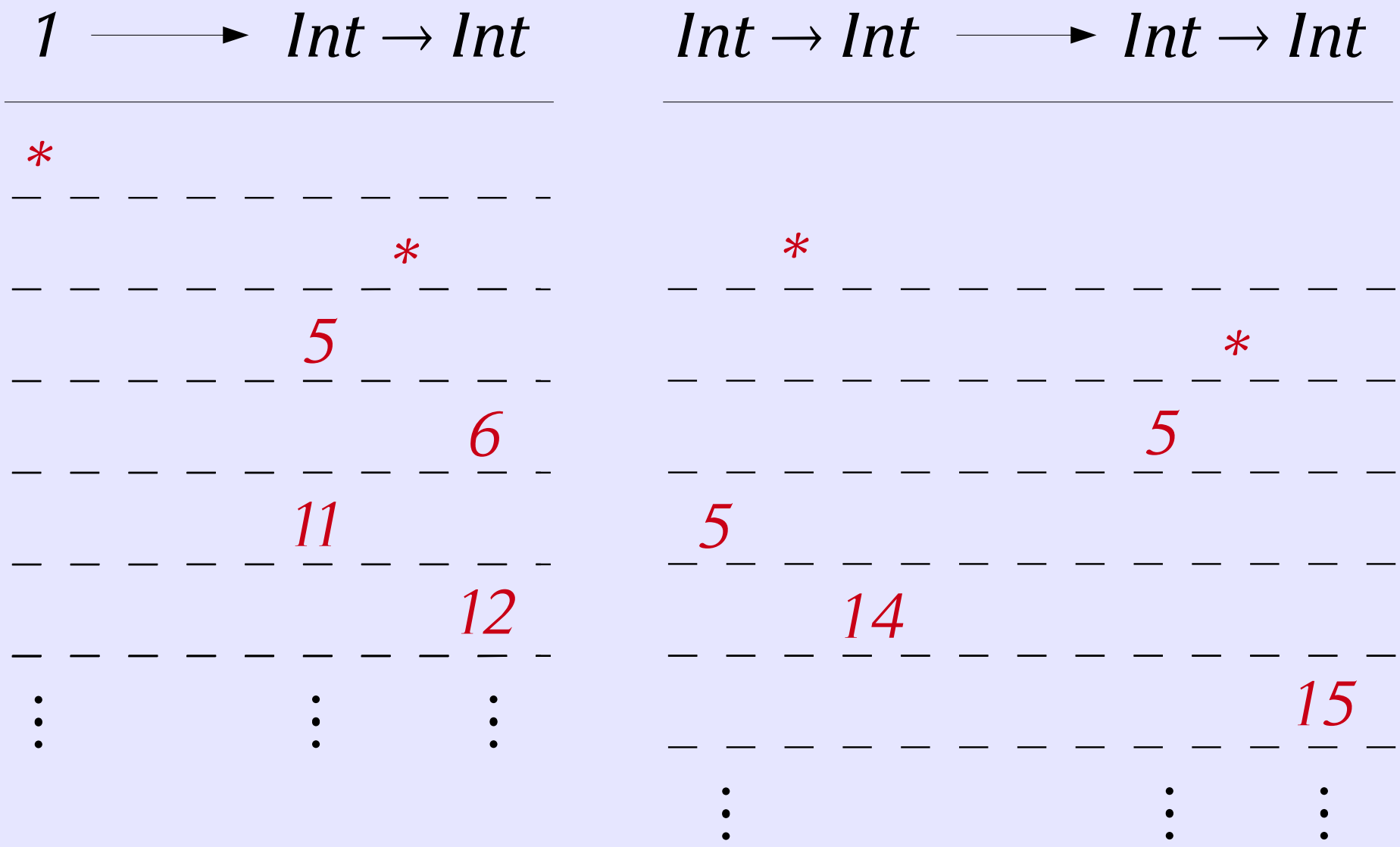
Composition



`⊢ λx.x+1 : int ⟶ int`

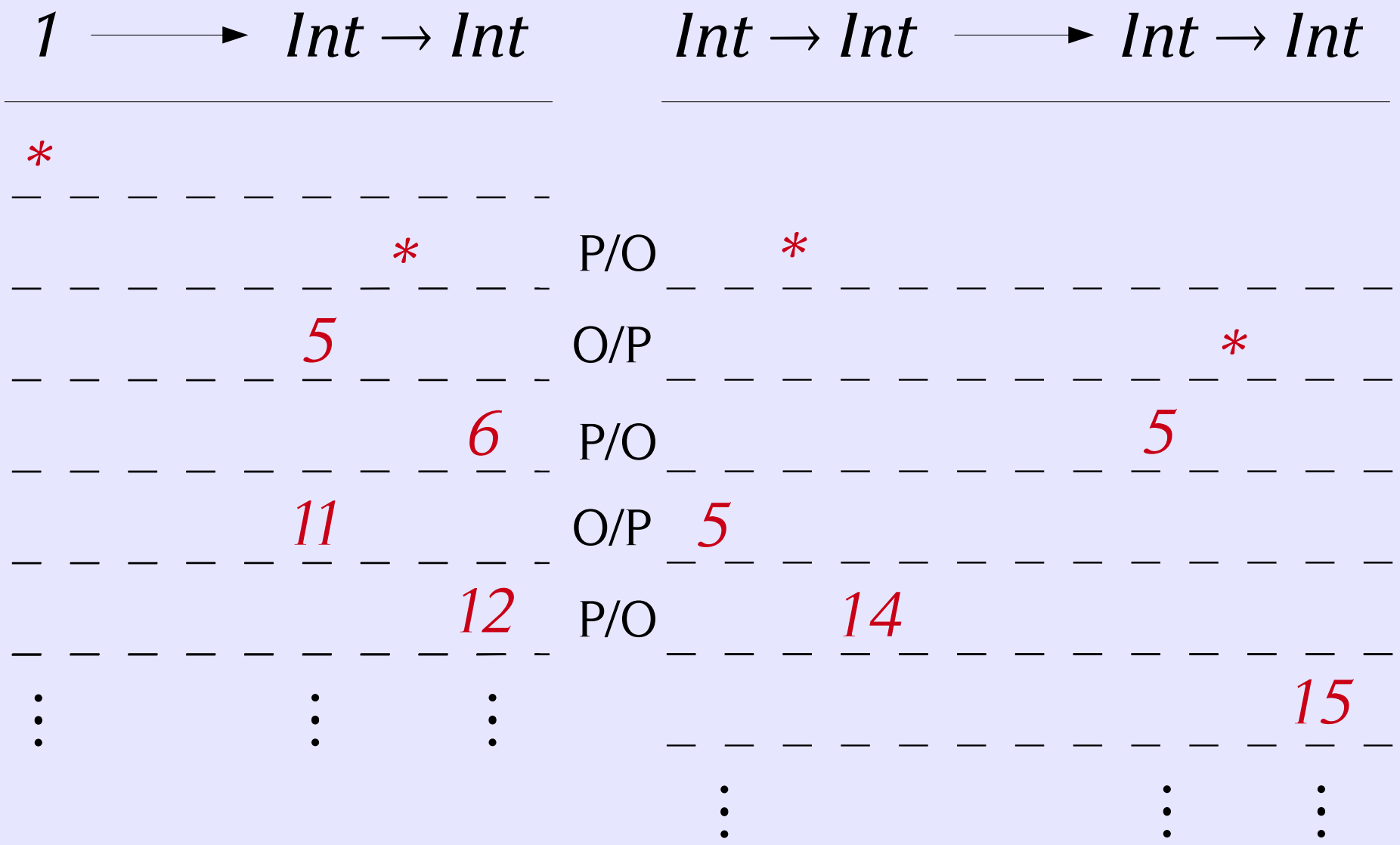
`f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int`

Composition



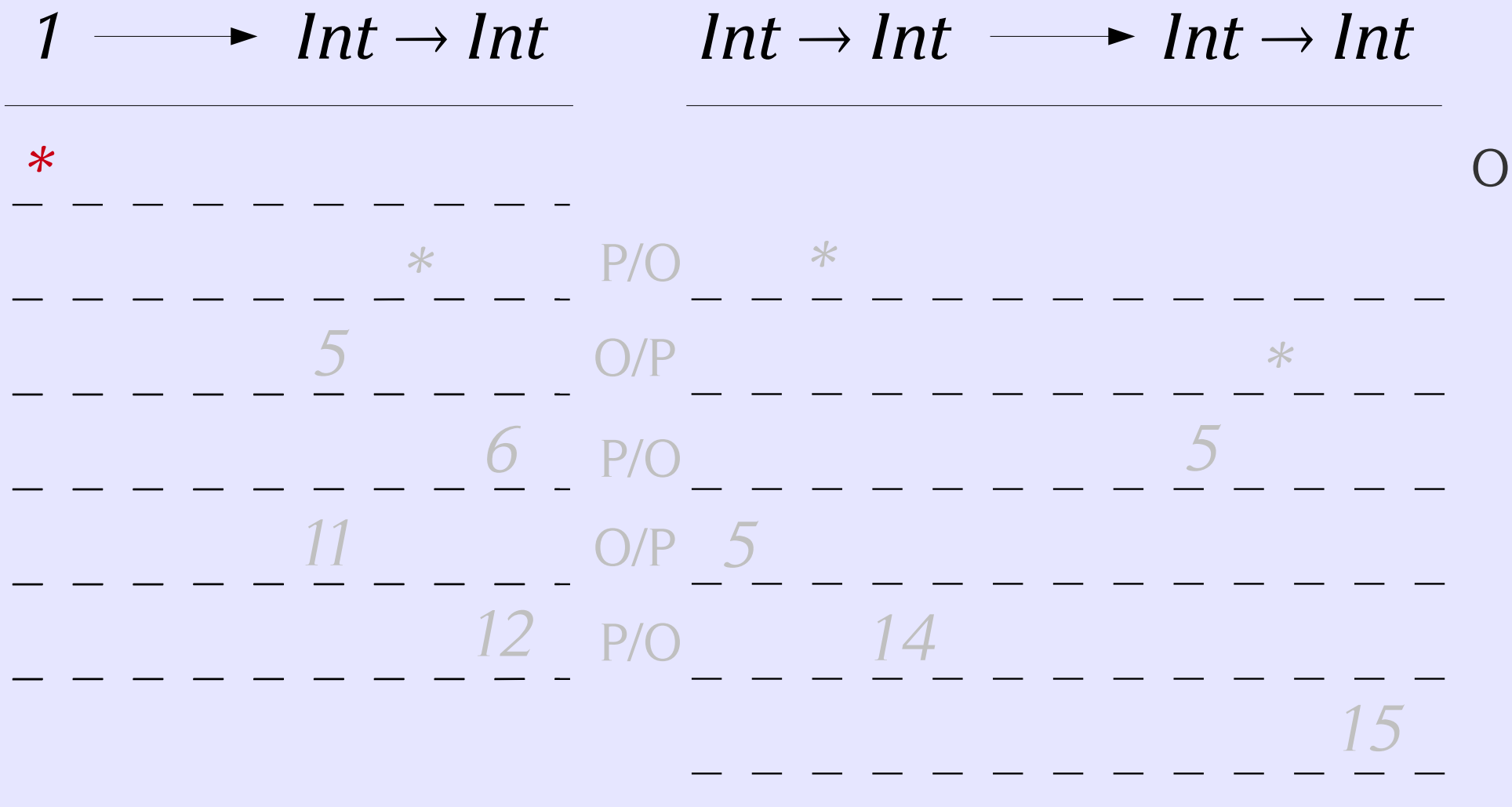
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

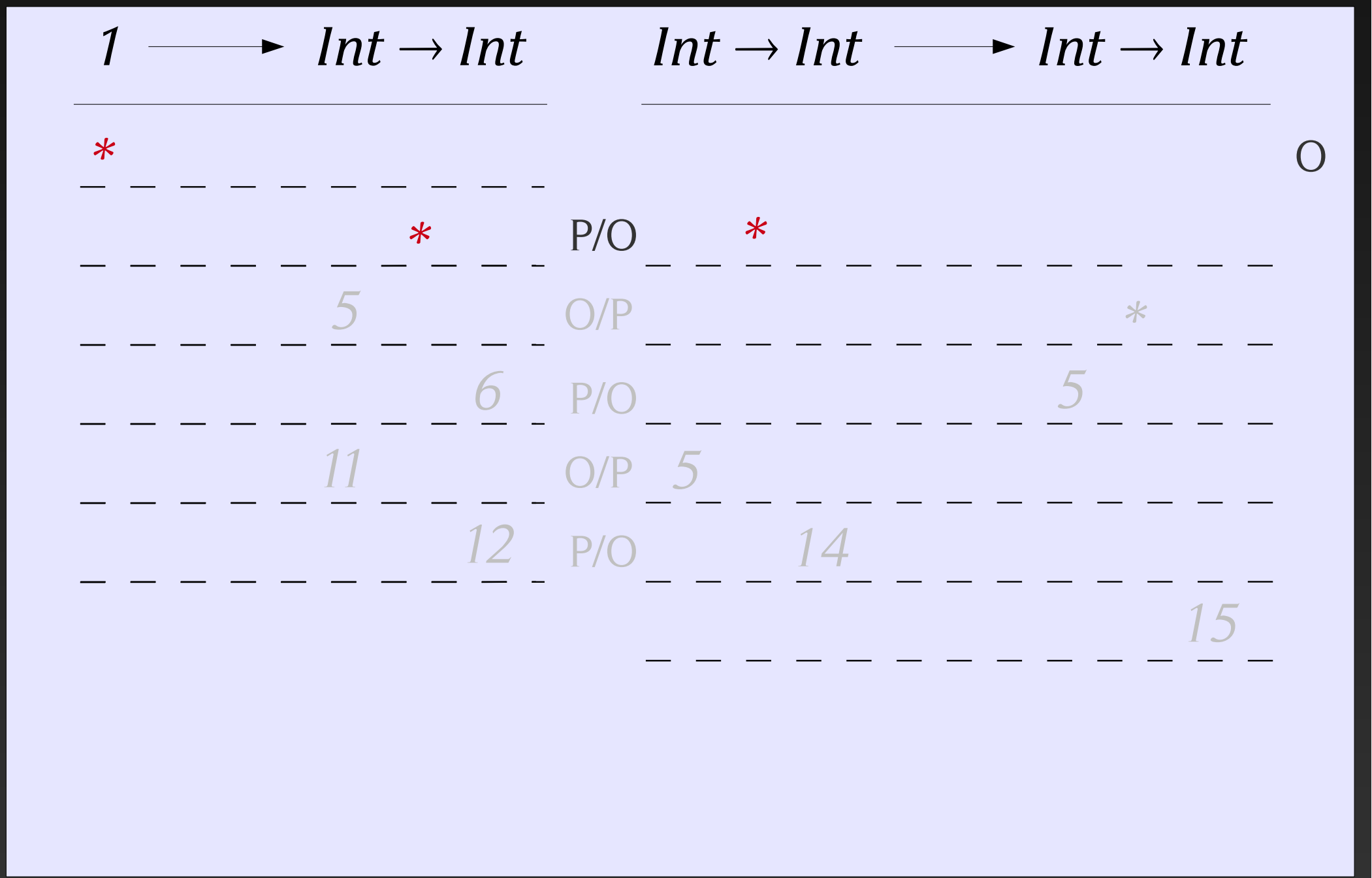
Composition



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

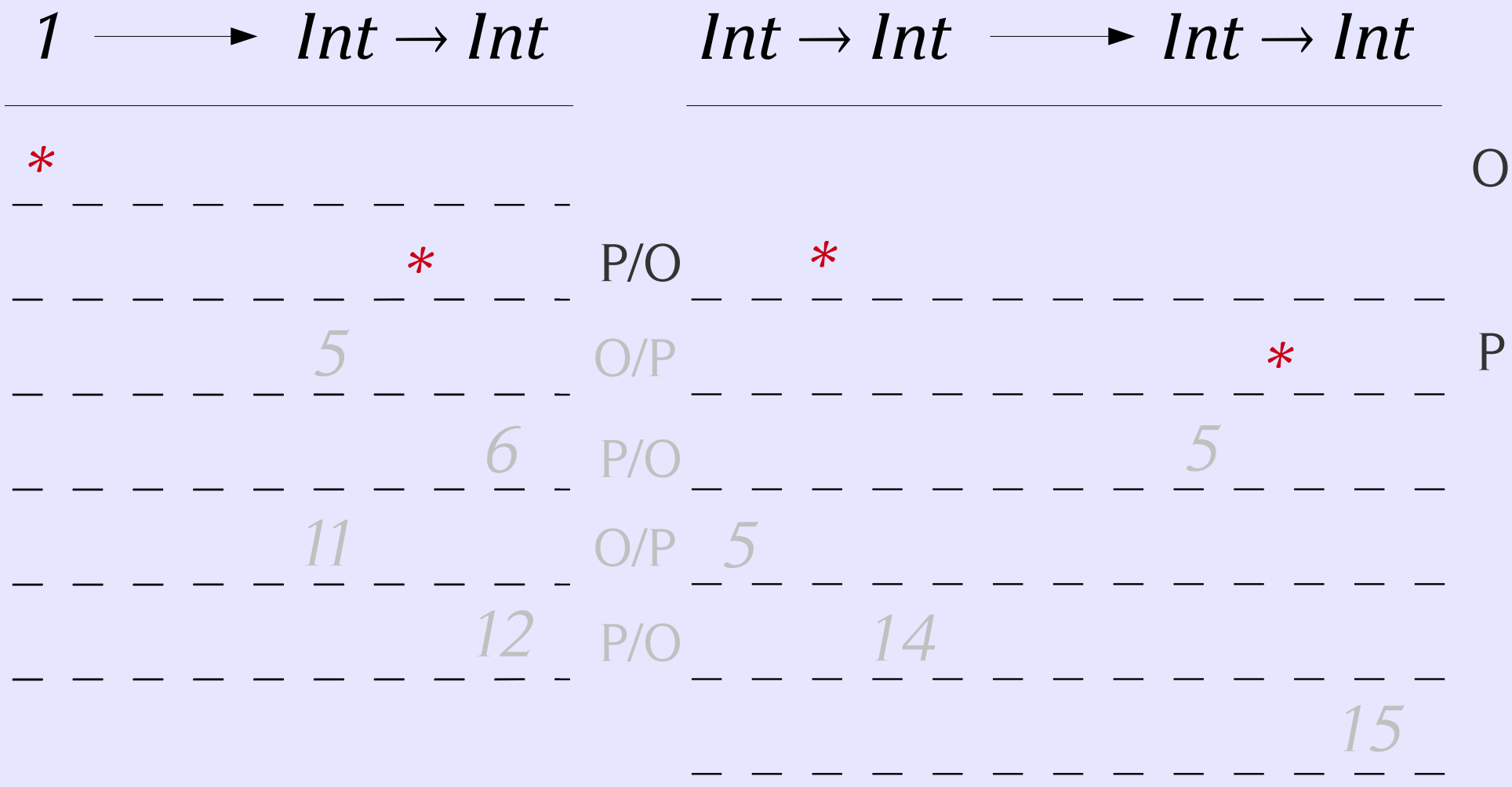
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

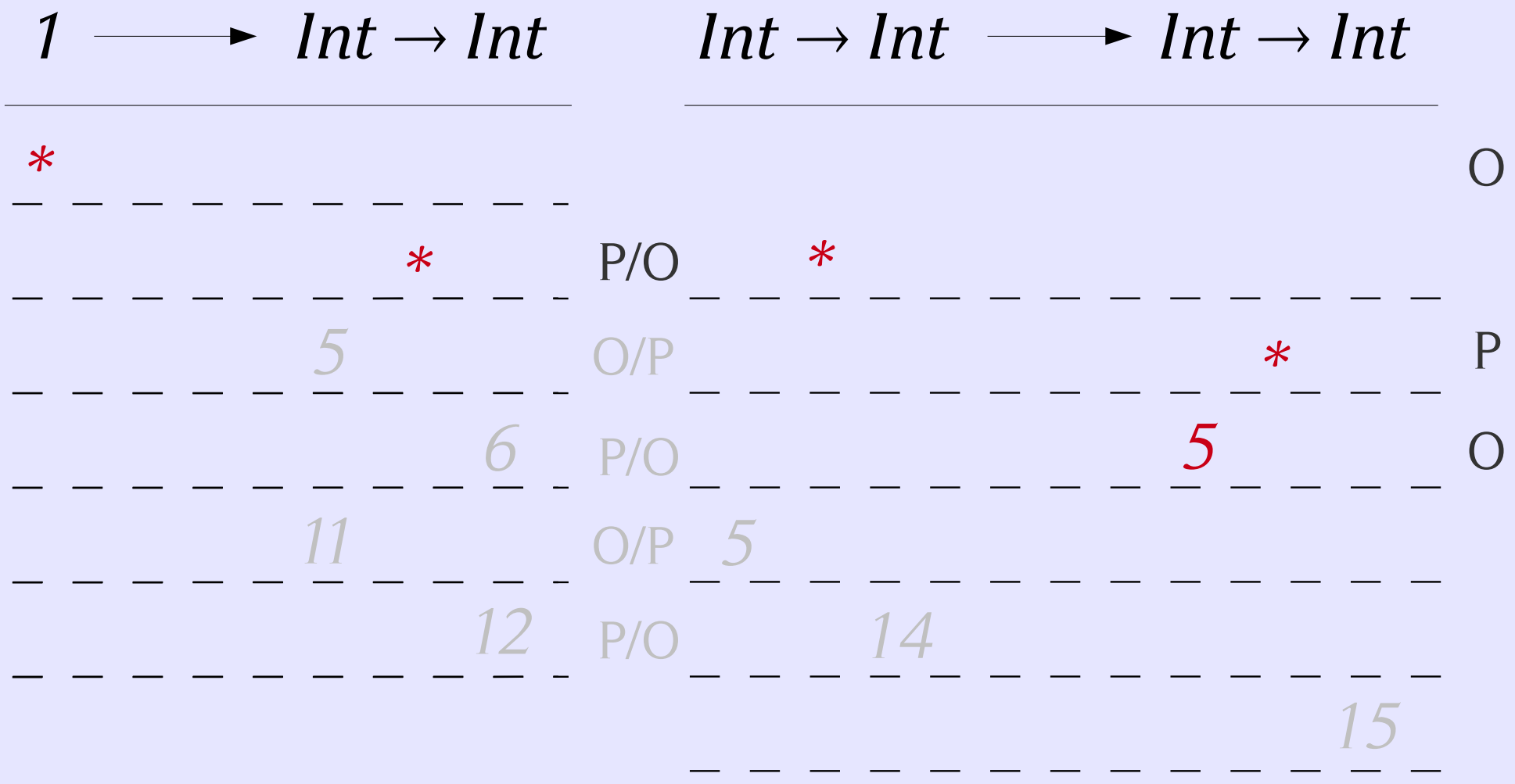
Composition



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

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

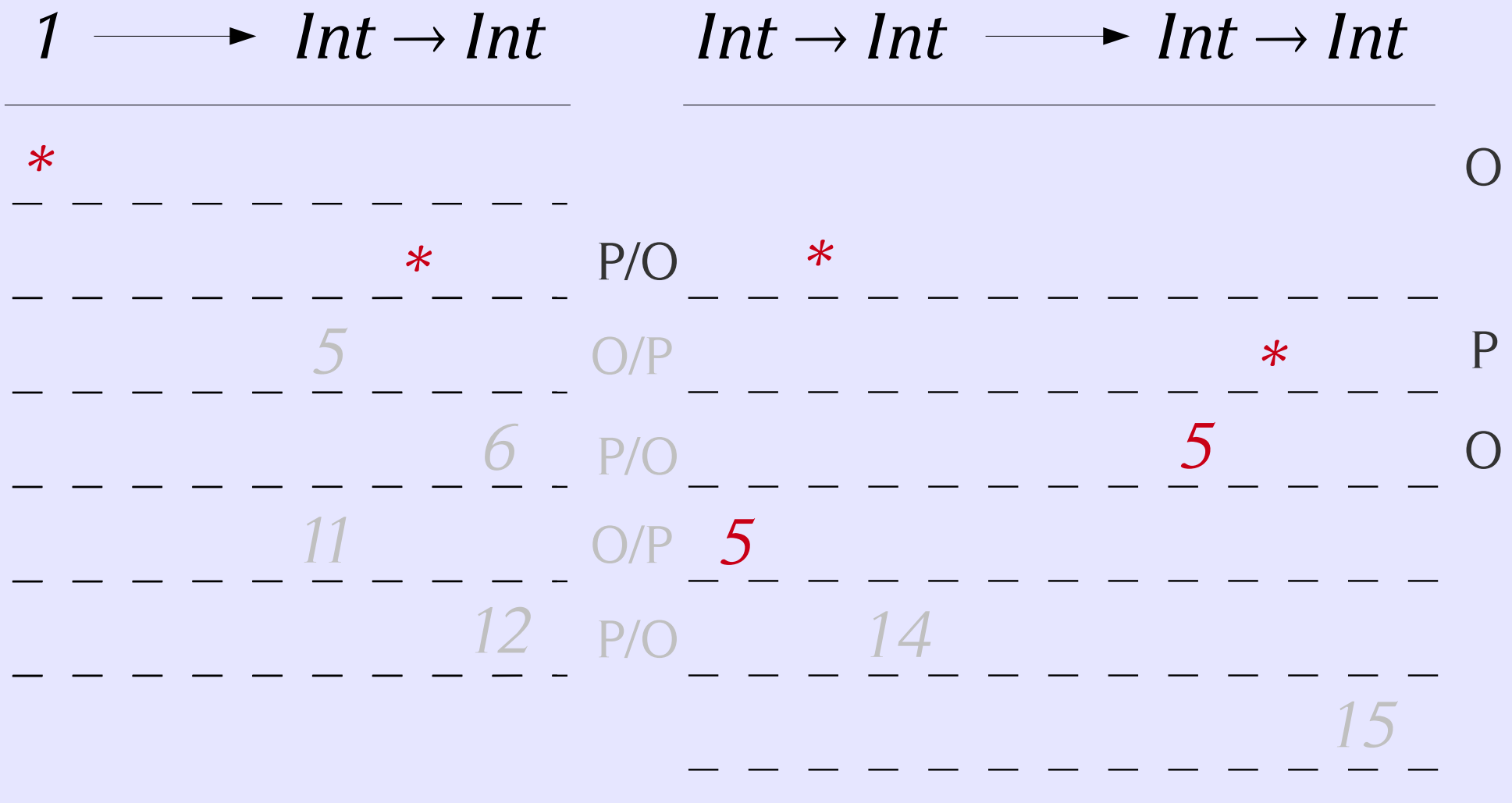
Composition



`⊢ λx.x+1 : int ⟶ int`

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

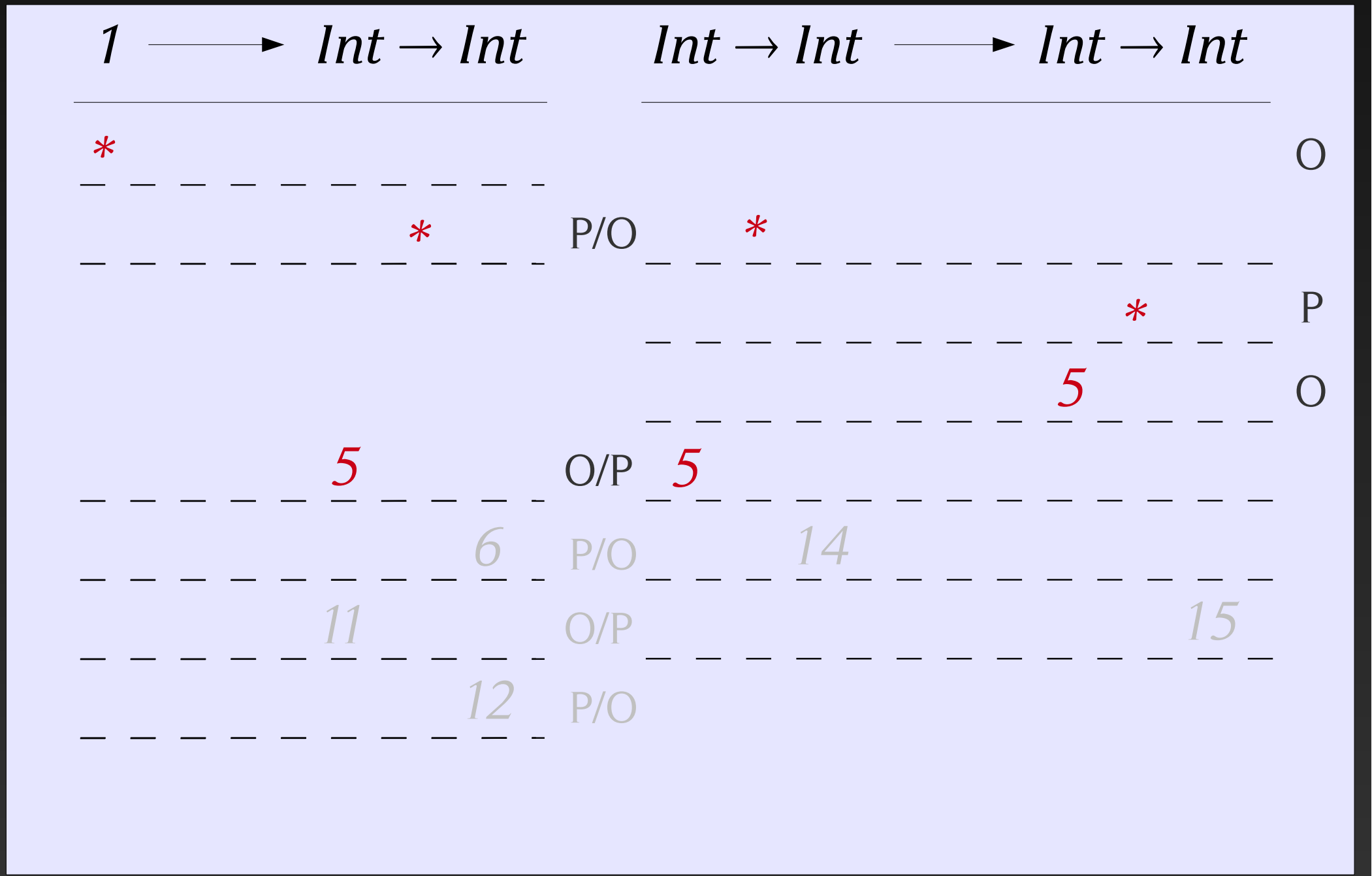
Composition



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

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

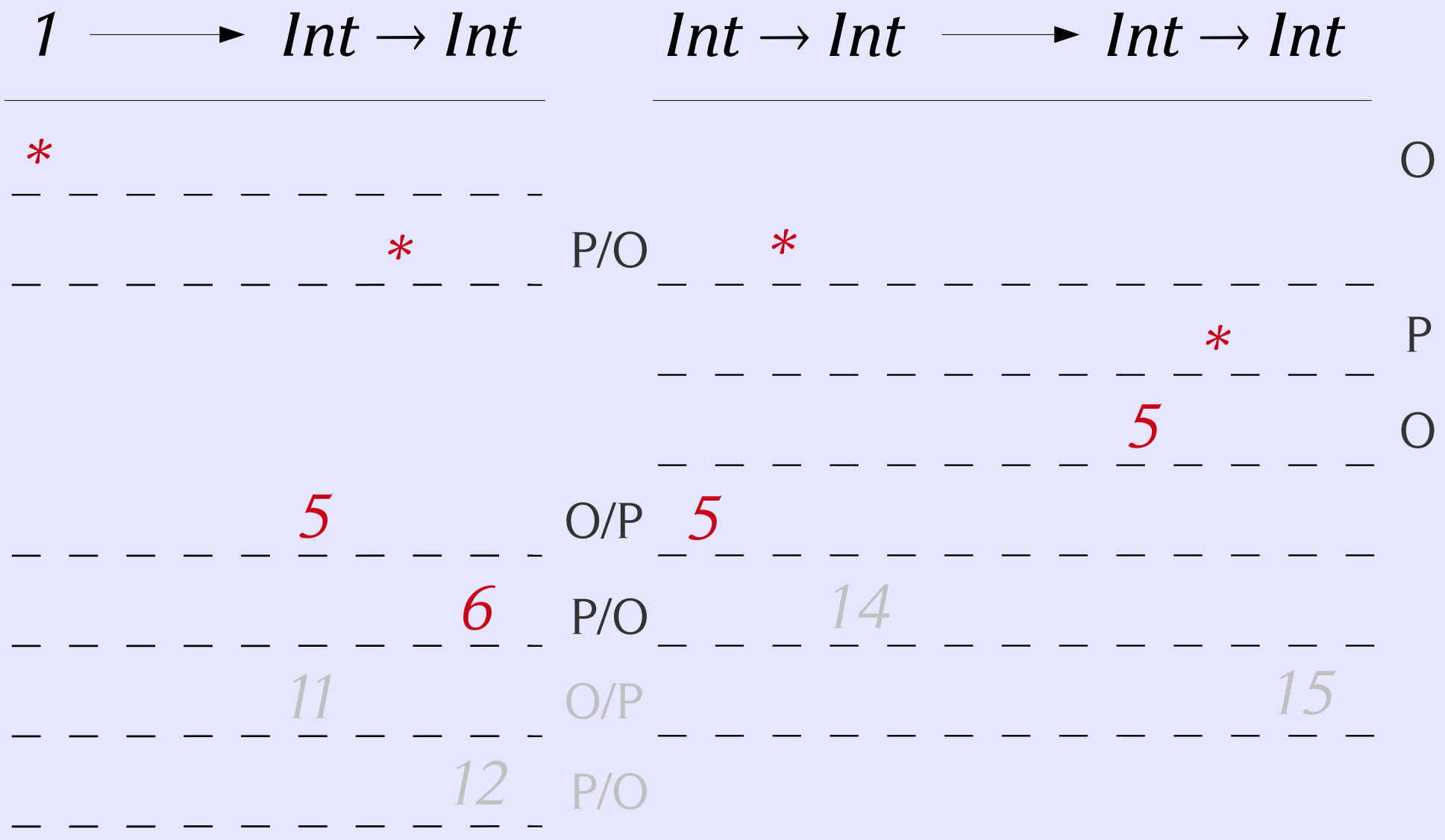
Composition



⊢ λx.x+1 : int ⟶ int

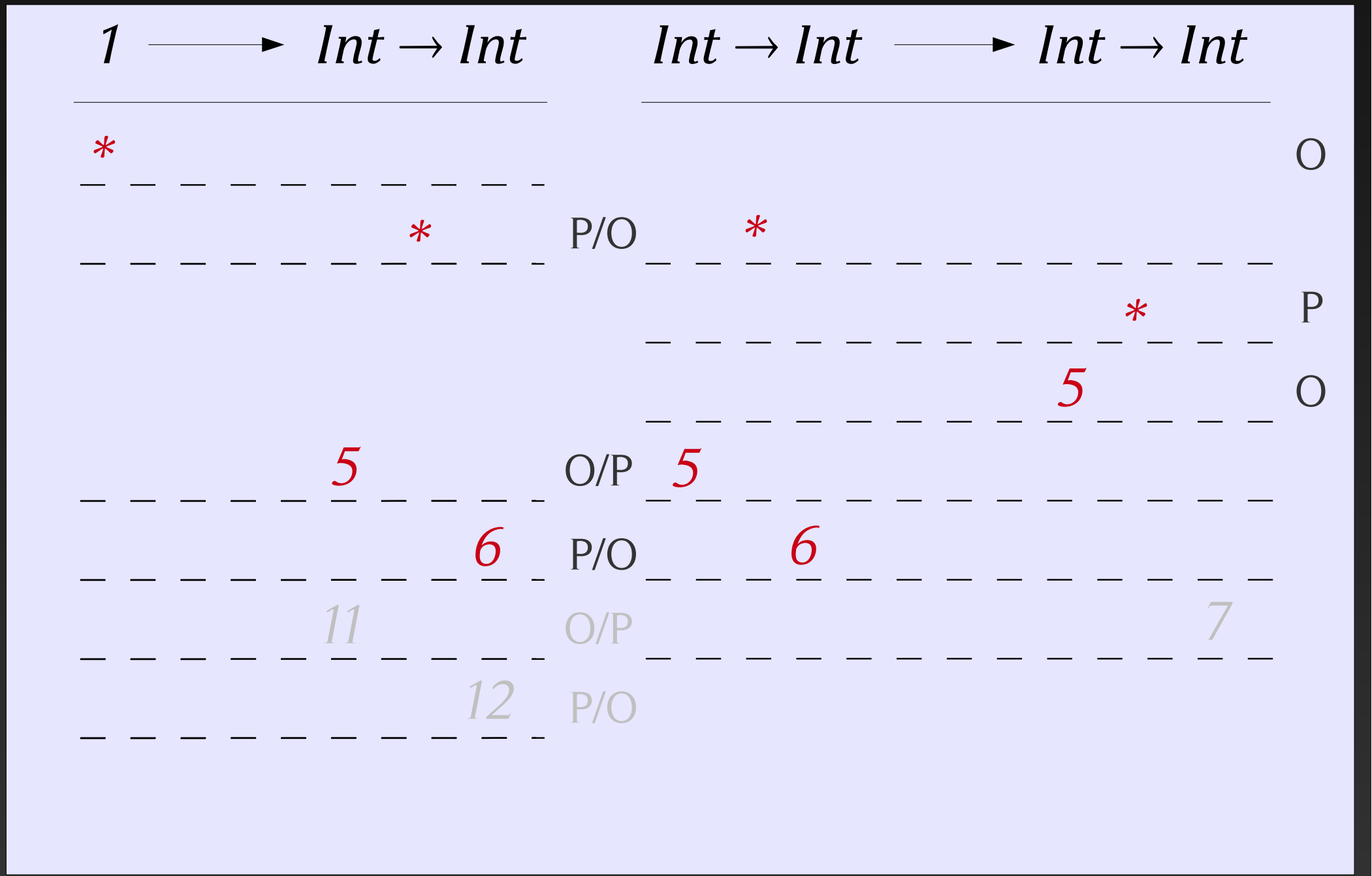
f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

Composition



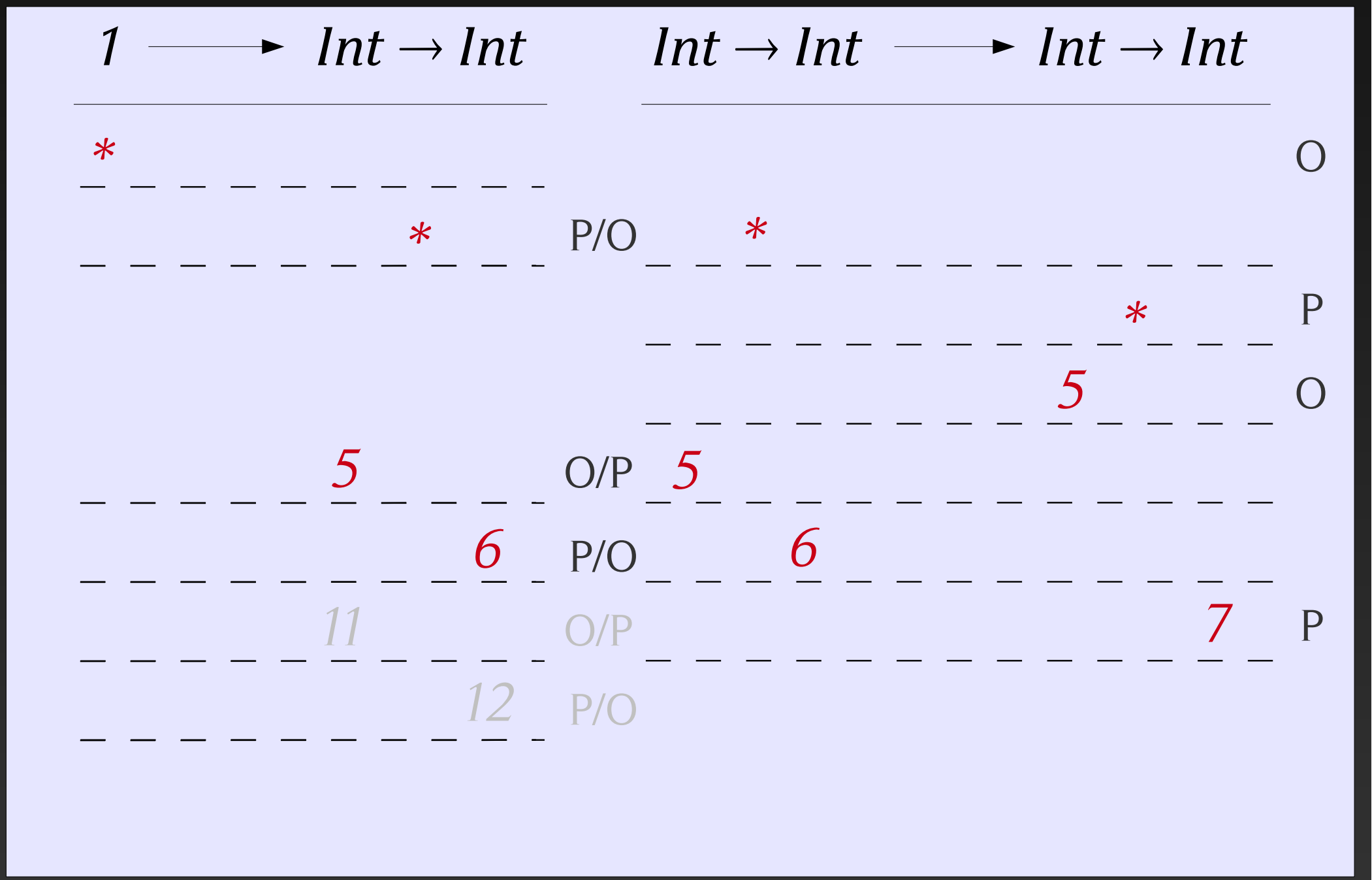
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

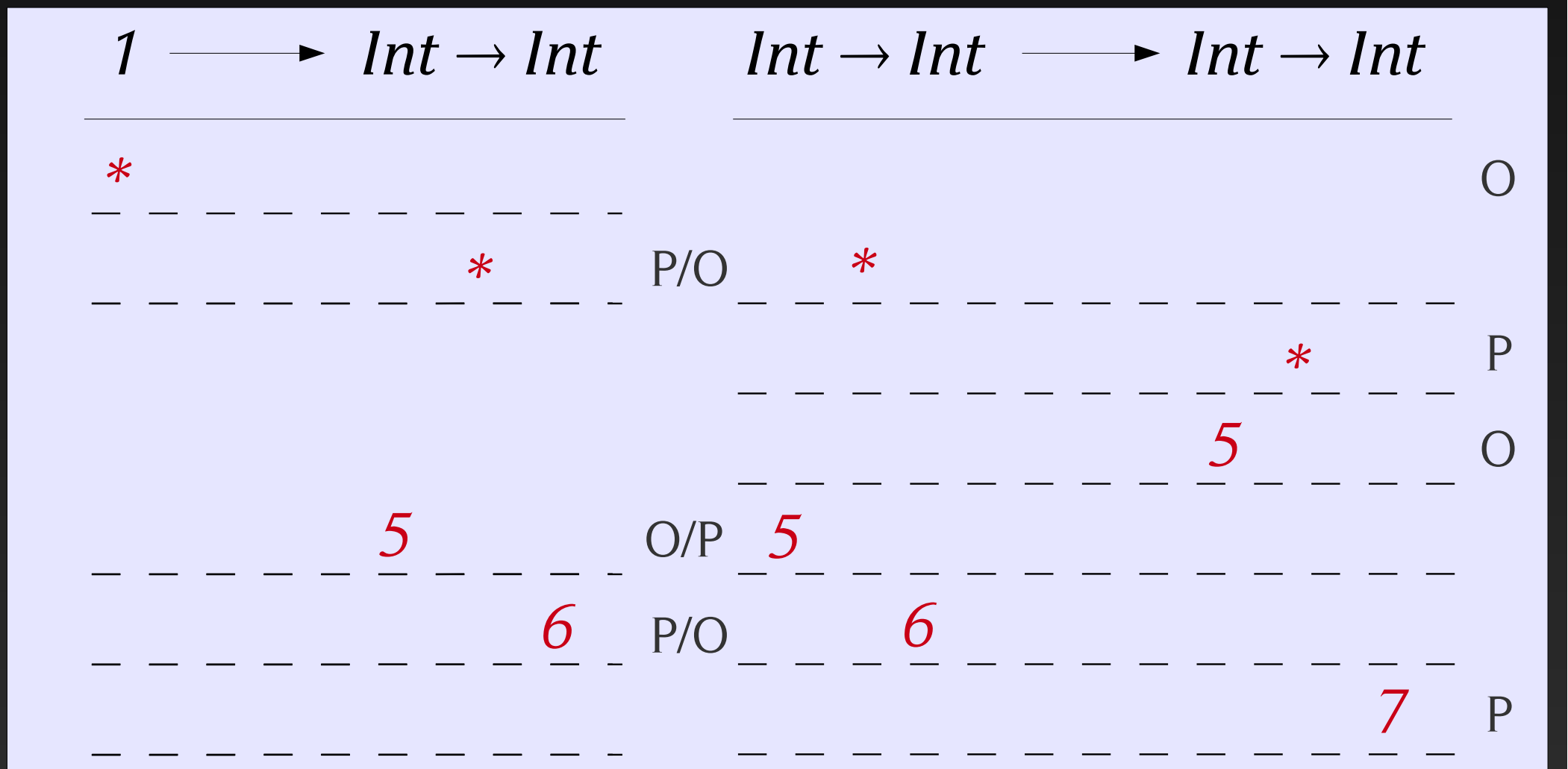
Composition



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

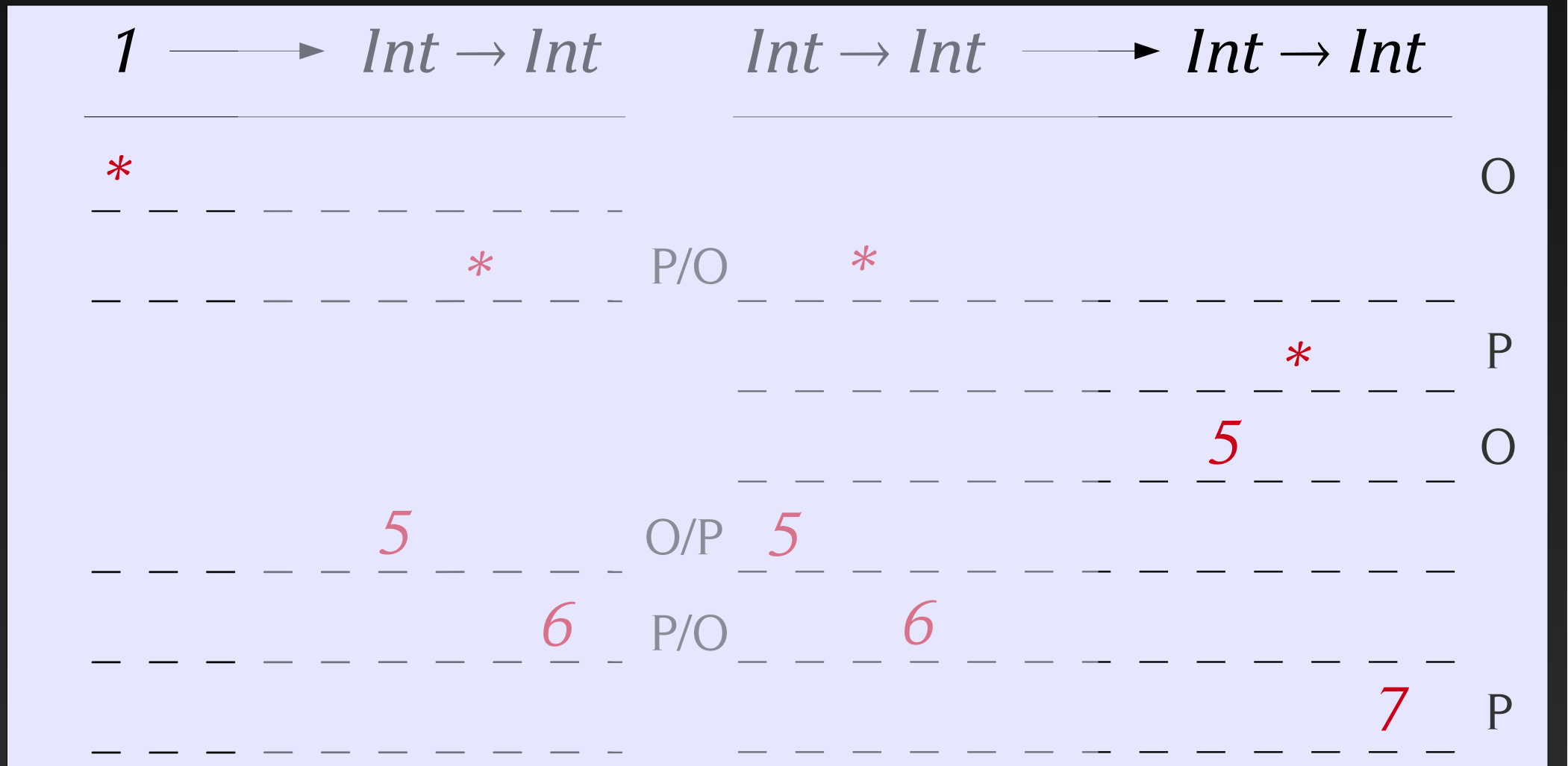
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

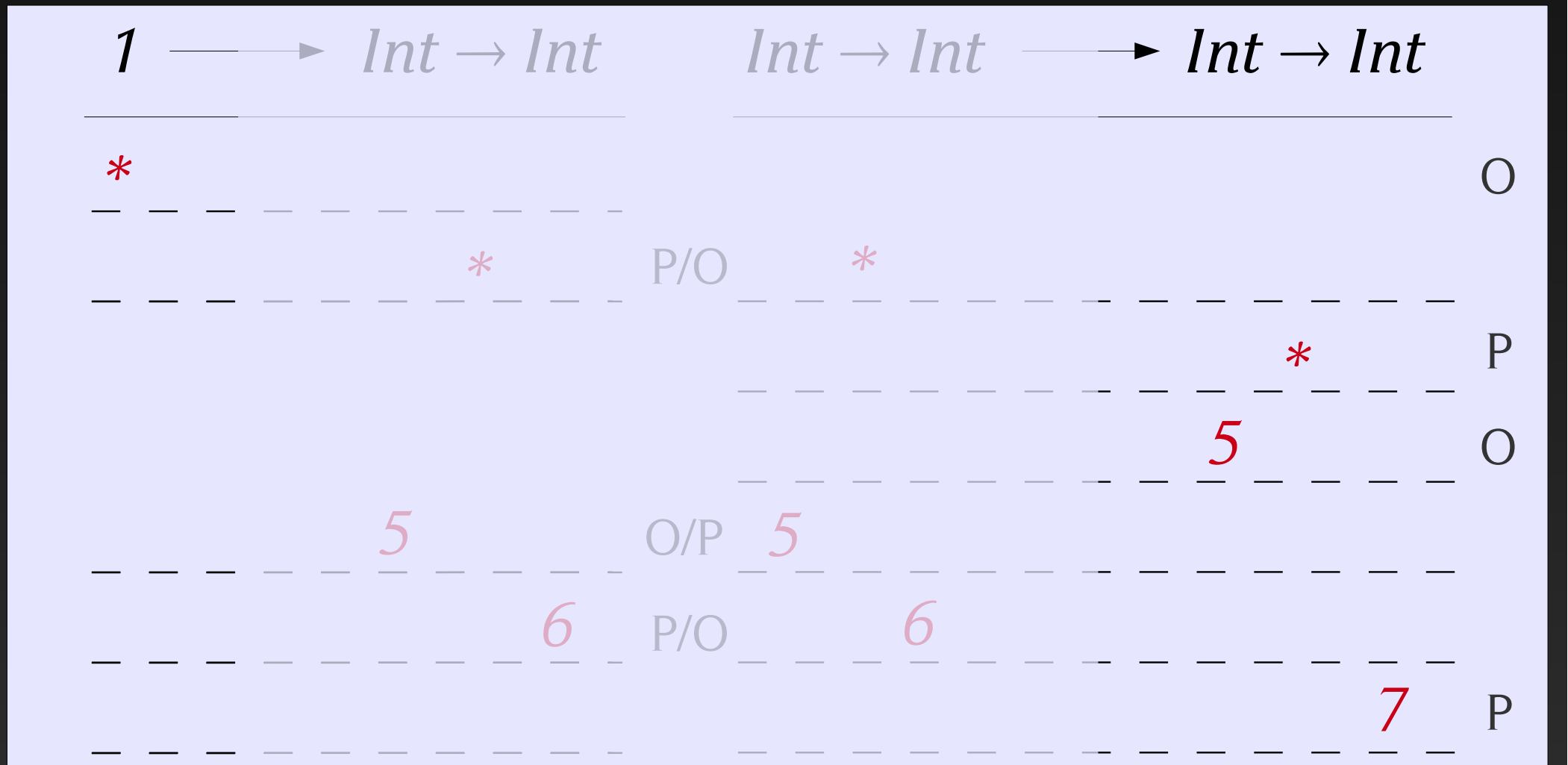
Composition



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

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

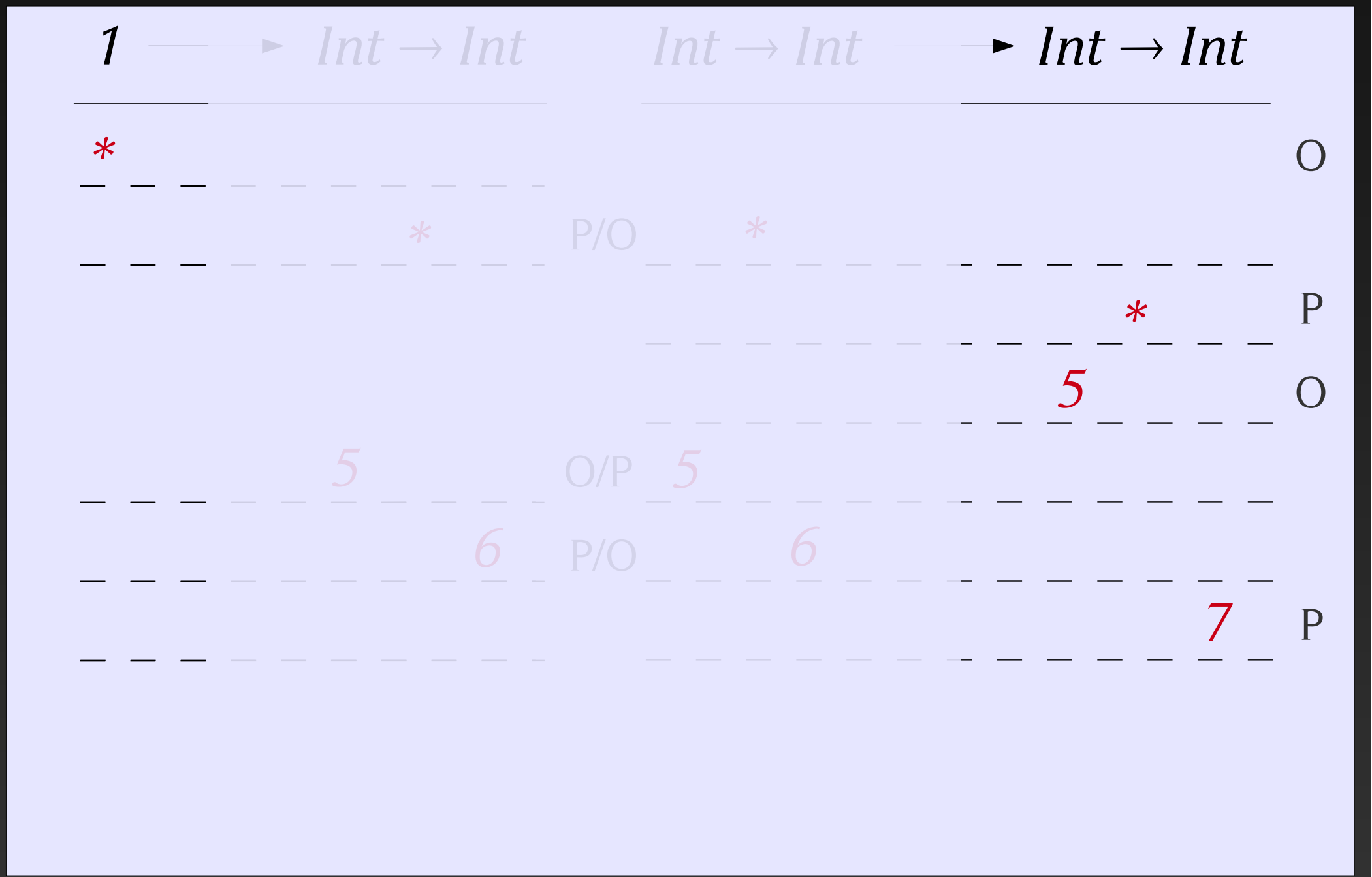
Composition



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

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

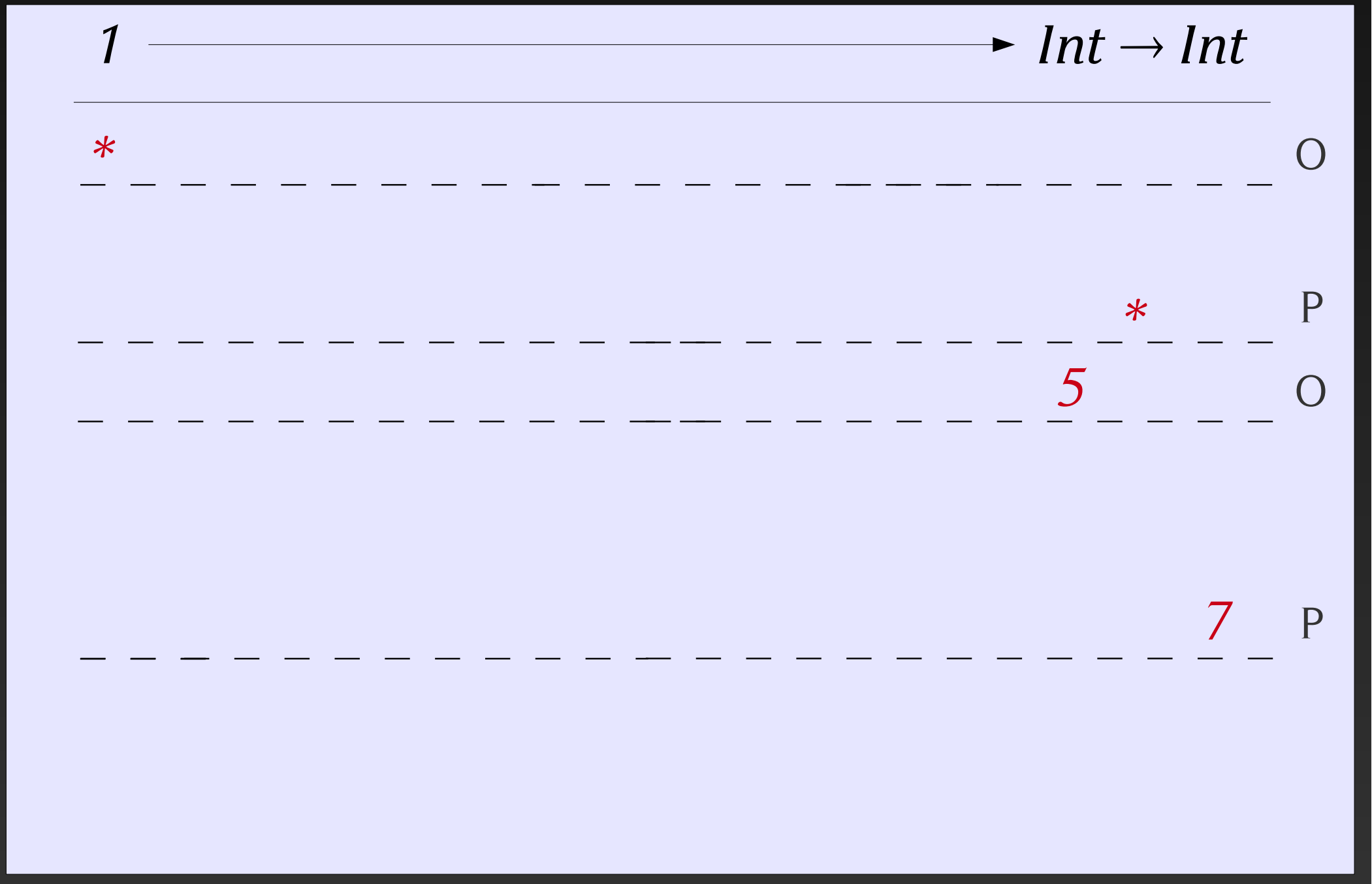
Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

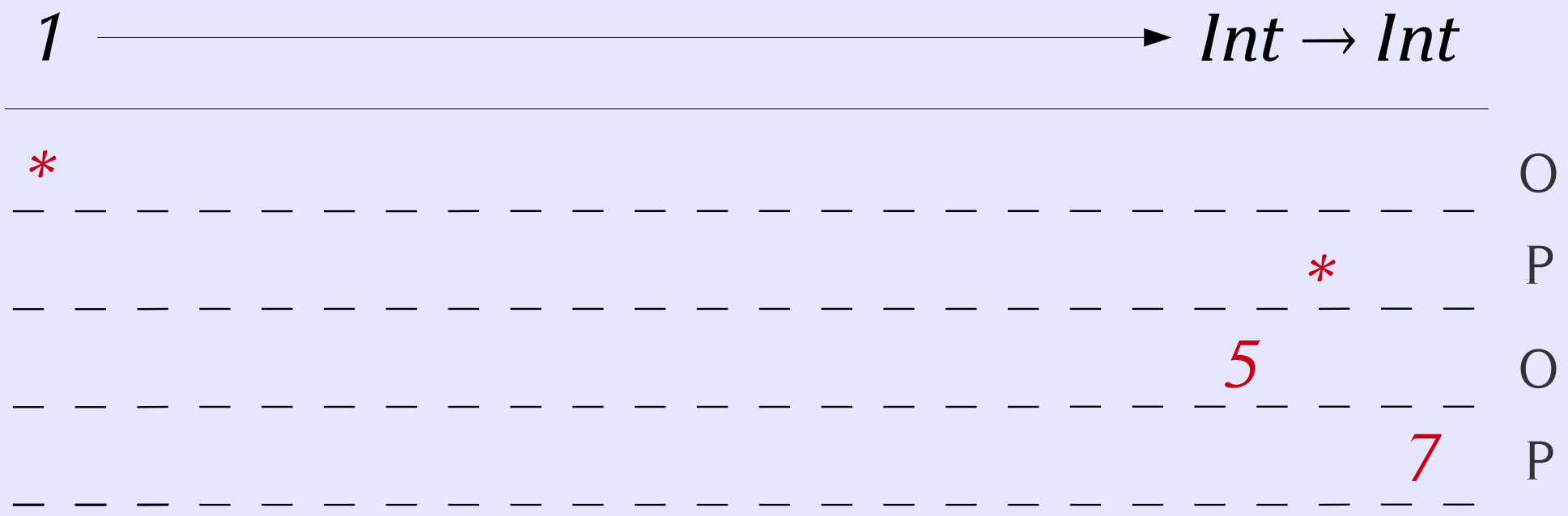
Composition



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

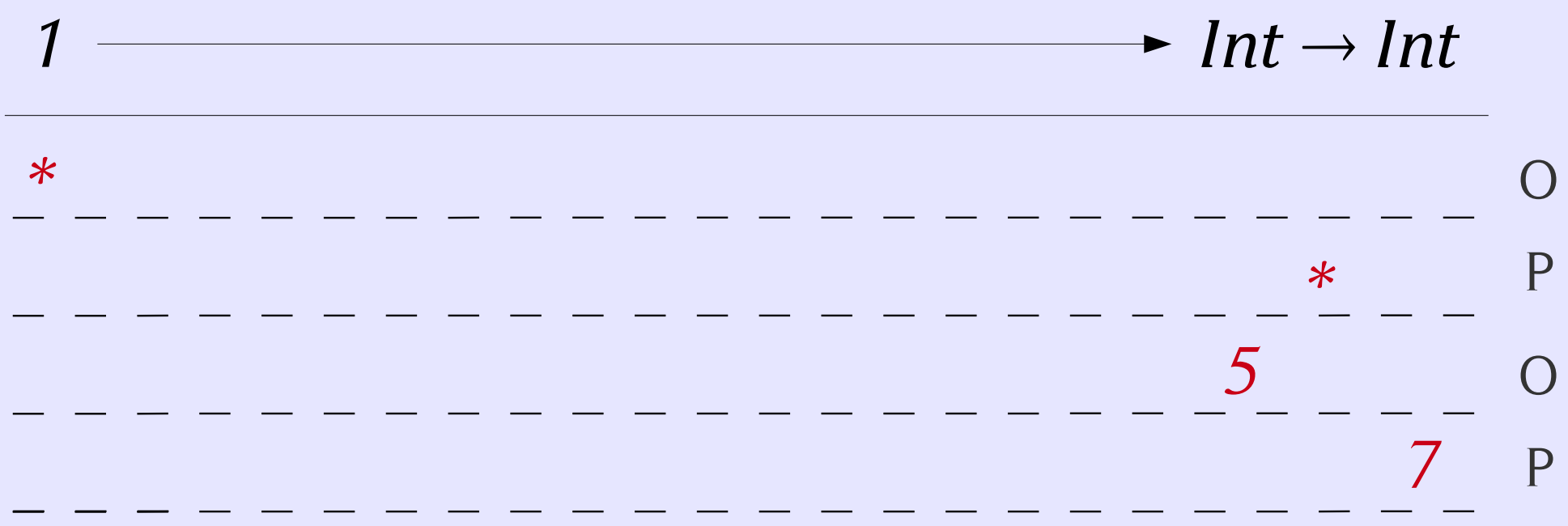
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

Composition

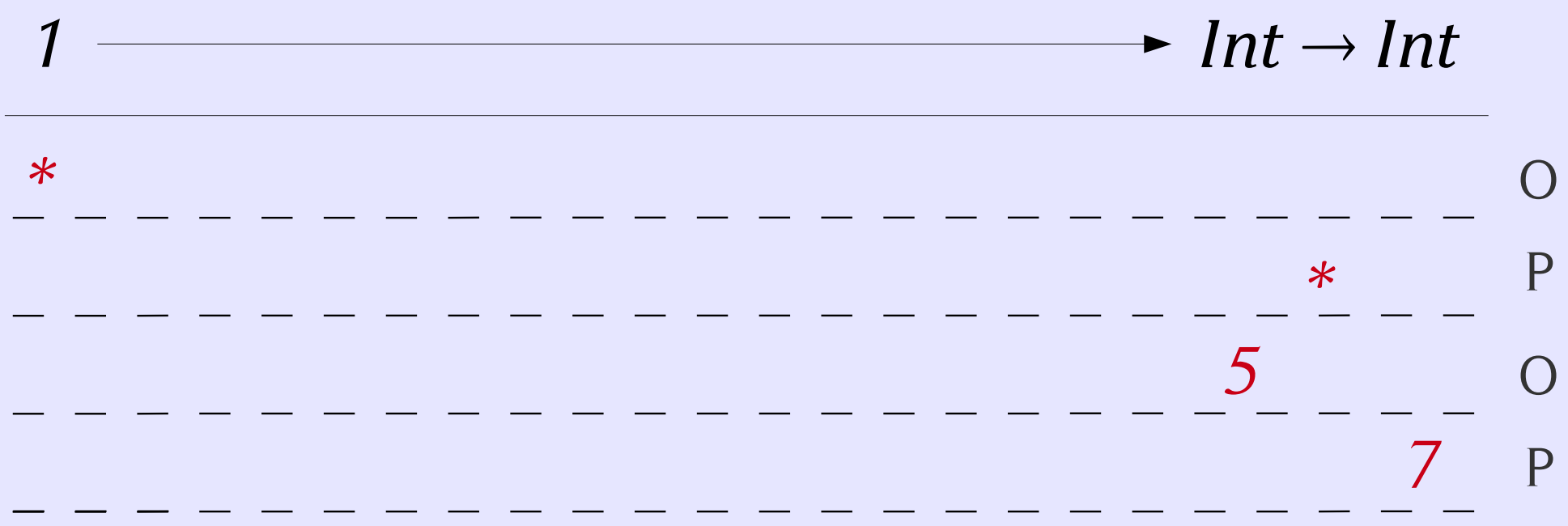


$\vdash \lambda x.x+1 : int \longrightarrow int$; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

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

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

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

Road to nominal games

Full Abstraction for PCF (early 90's)

- Three groups (AJM, HO, N)
- Roots in Mathematical Logic

Road to nominal games

Full Abstraction for PCF (early 90's)

- Three groups (AJM, HO, N)
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

Road to nominal games

Full Abstraction for PCF (early 90's)

- Three groups (AJM, HO, N)
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} new resources?

Road to nominal games

Full Abstraction for PCF (early 90's)

- Three groups (AJM, HO, N)
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} new resources?

Nominal game semantics (2004-)

Nominal techniques

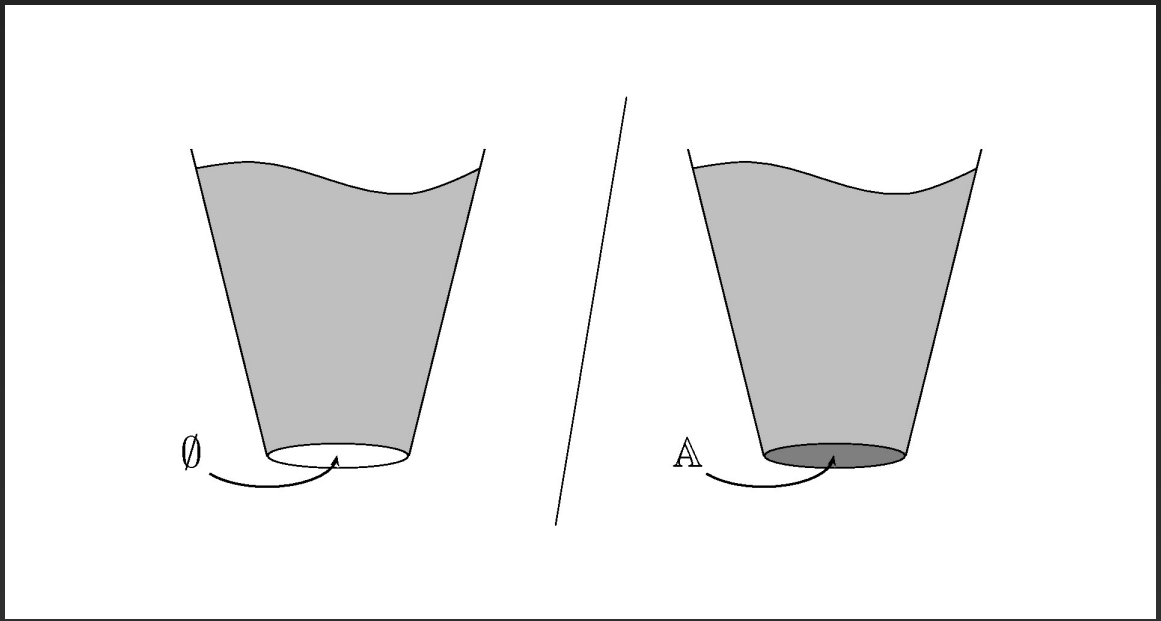
- A foundational theory for doing maths with names and name-binding

Nominal techniques

- A foundational theory for doing maths with names and name-binding
- Two presentations:
  - FM-set theory [Gabbay&Pitts 99, Gabbay 01]
  - Nominal sets [Pitts 01]

FM-sets

- Fraenkel-Mostowski model of ZFA:
  - Basis: empty set + inf. set $\mathcal{A}$ of atoms
  - Constructions have finite support



Nominal sets

- Countably infinite set $\mathcal{N}$ of names

Nominal sets

- Countably infinite set $\mathcal{N}$ of names
- 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

- NOM: nominal sets and equivariant functions

Nominal games

Nominal games

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

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

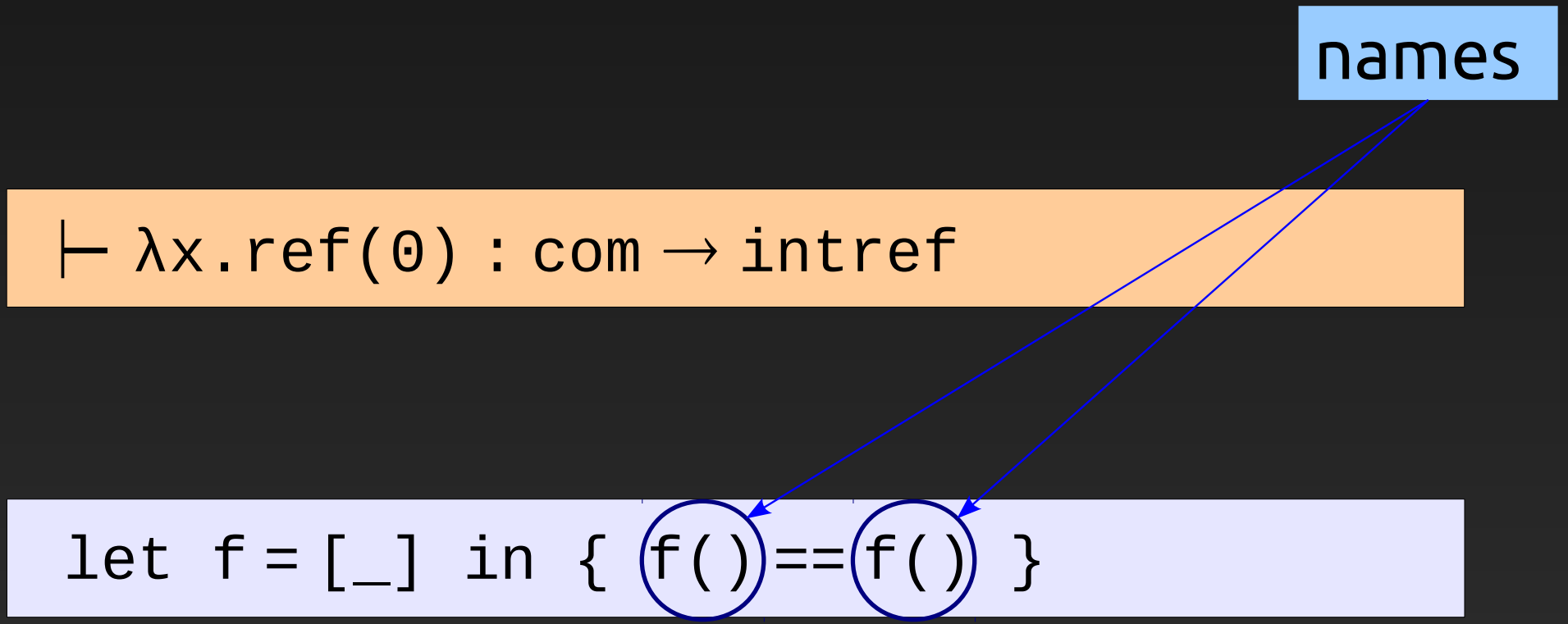
Nominal games

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

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

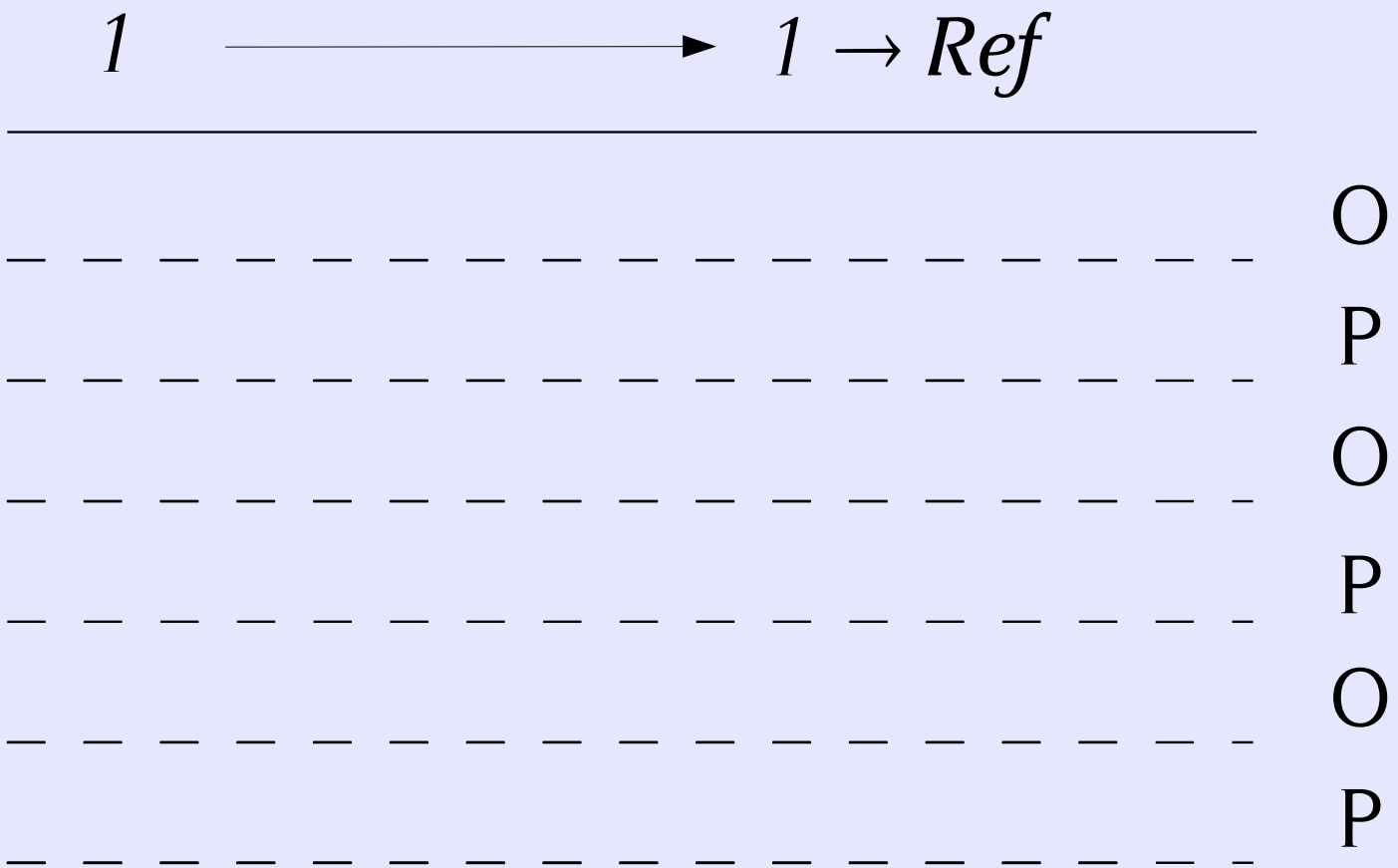
Nominal games

Games in nominal sets



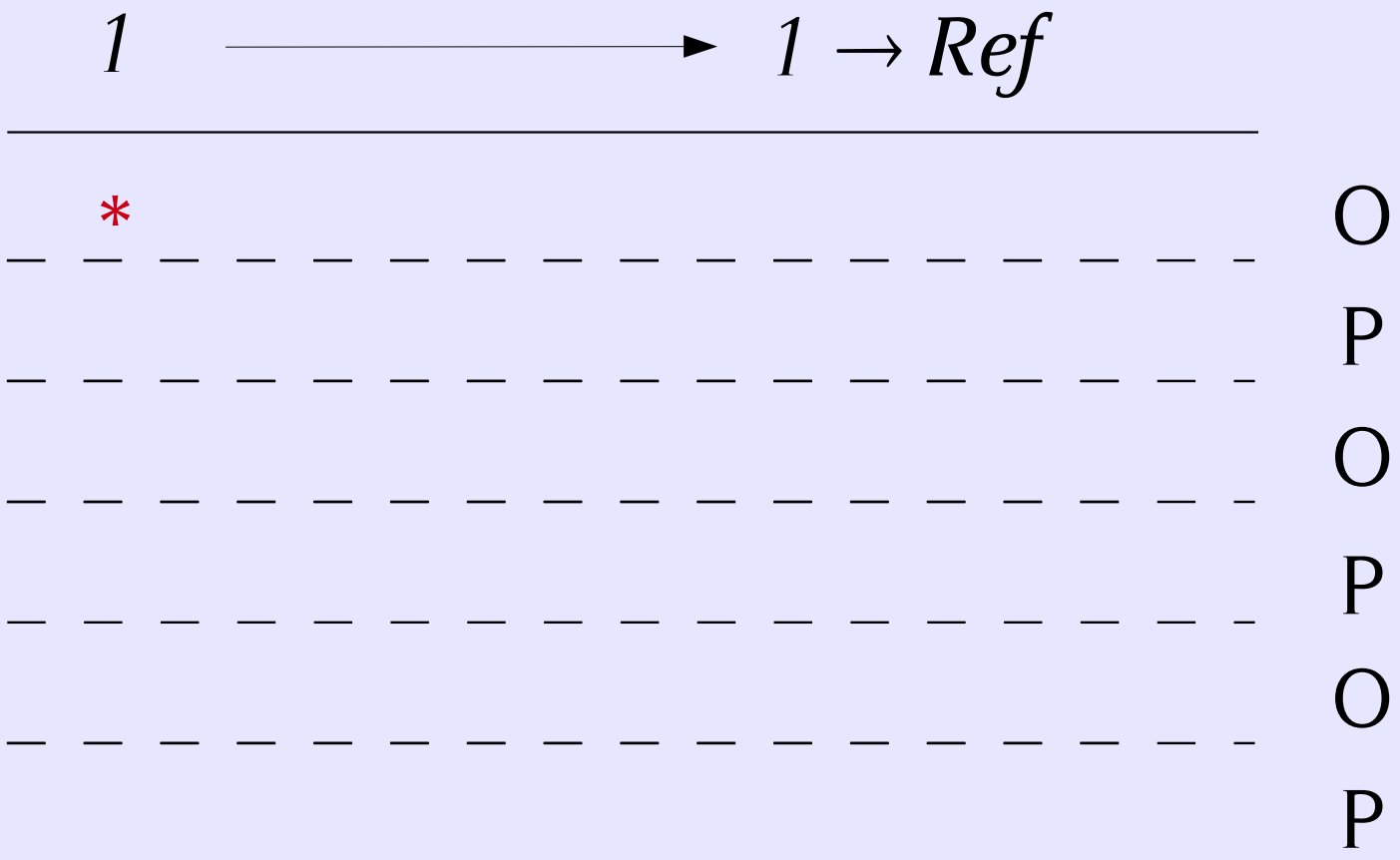
Examples

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



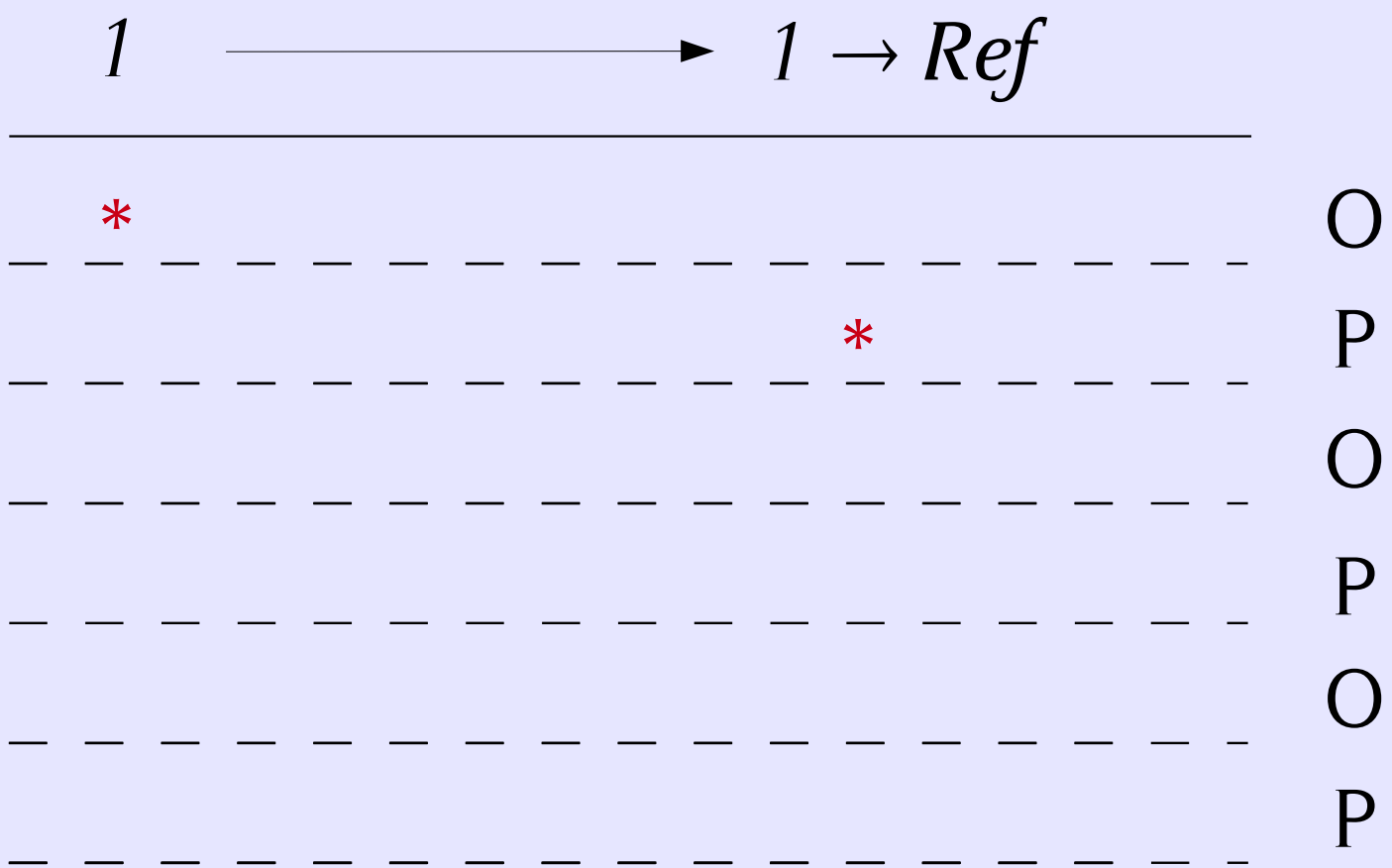
Examples

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



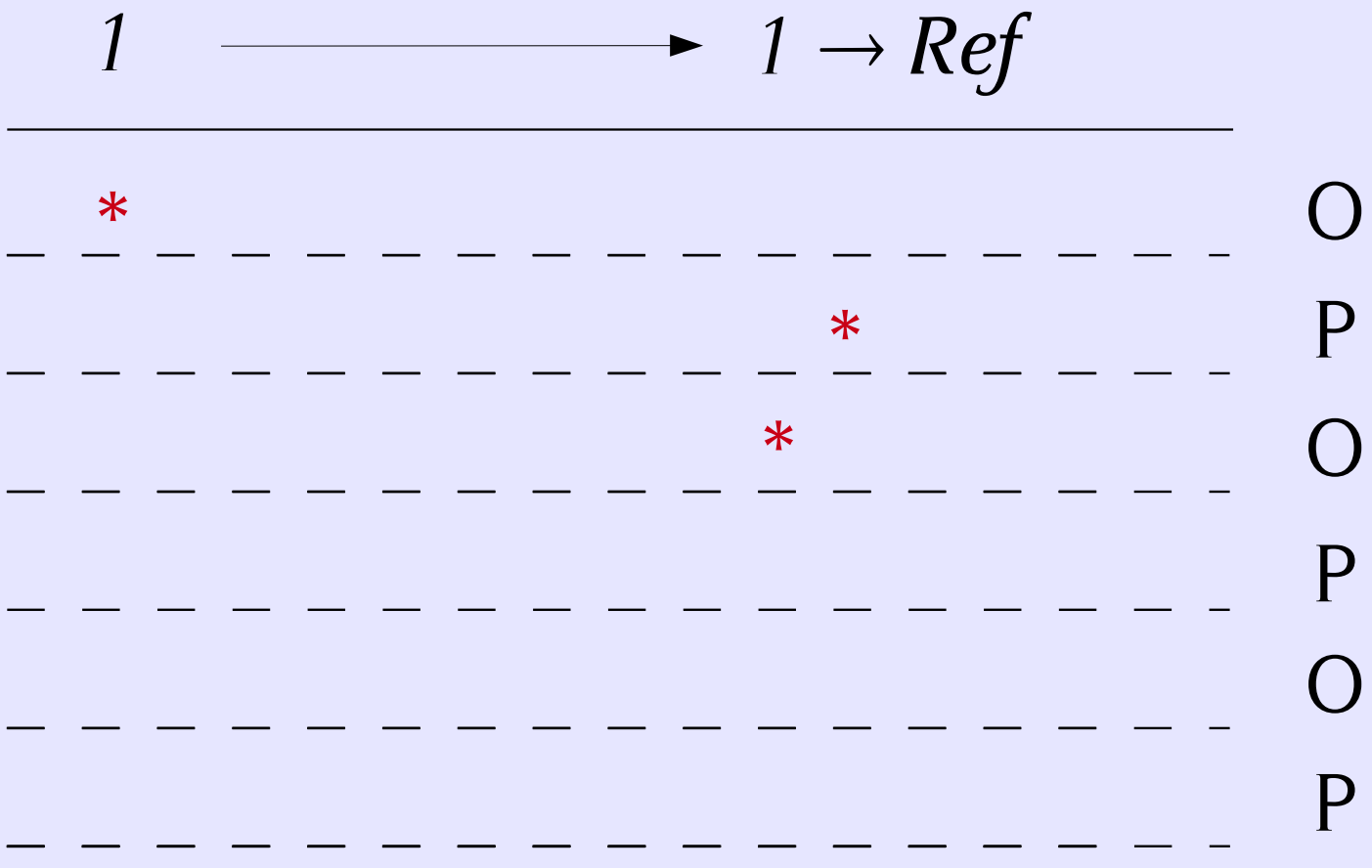
Examples

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



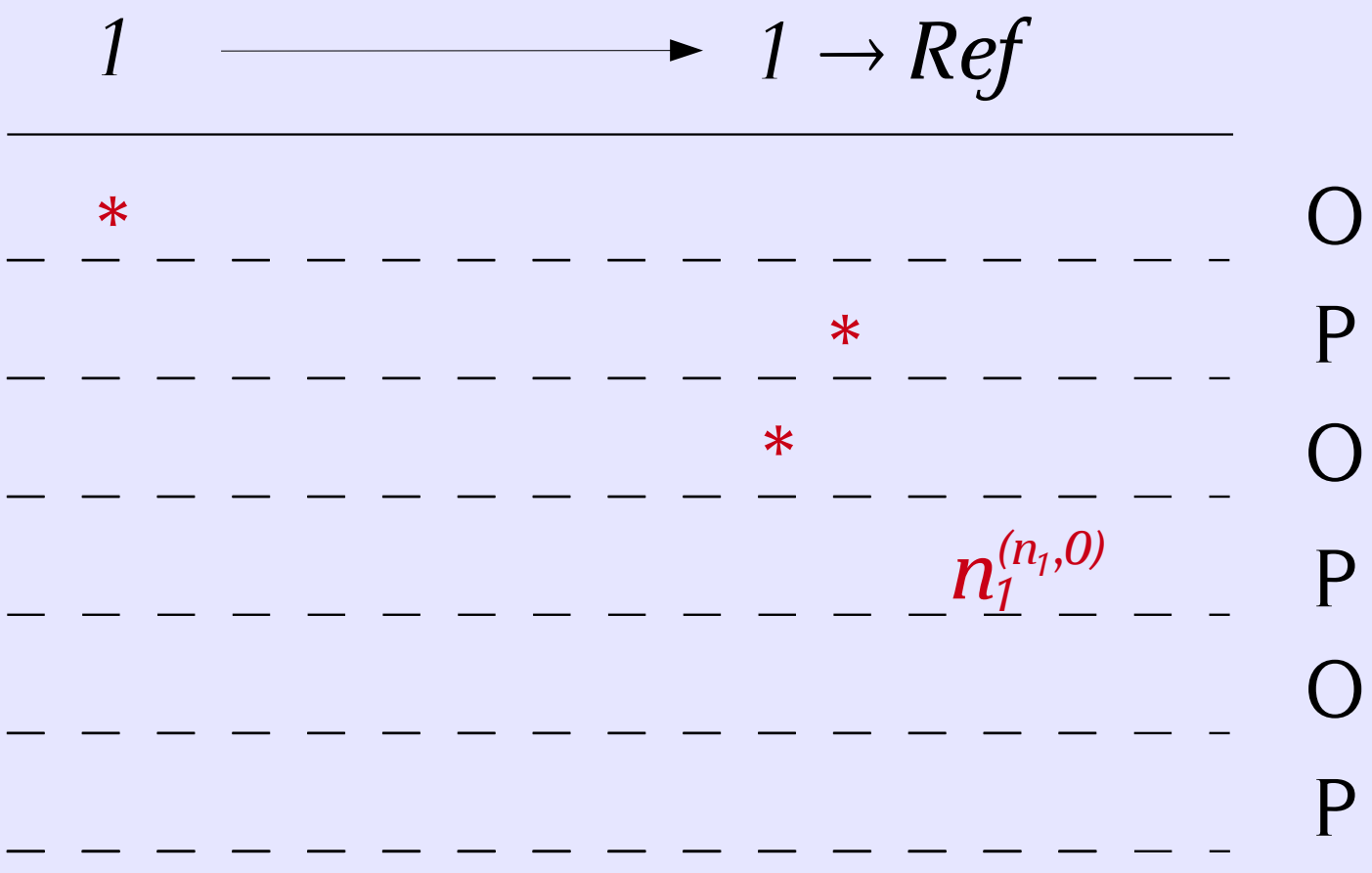
Examples

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



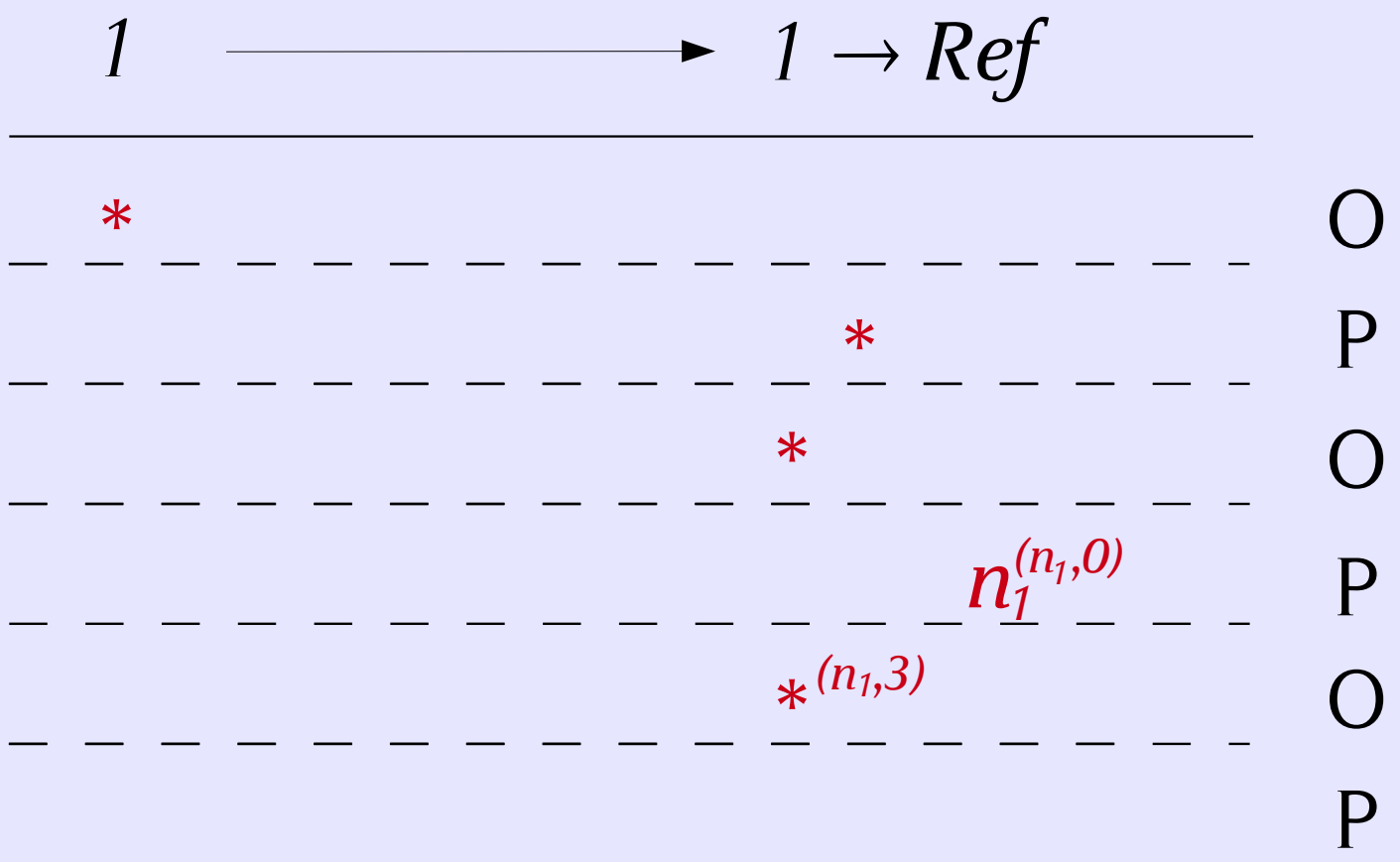
Examples

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



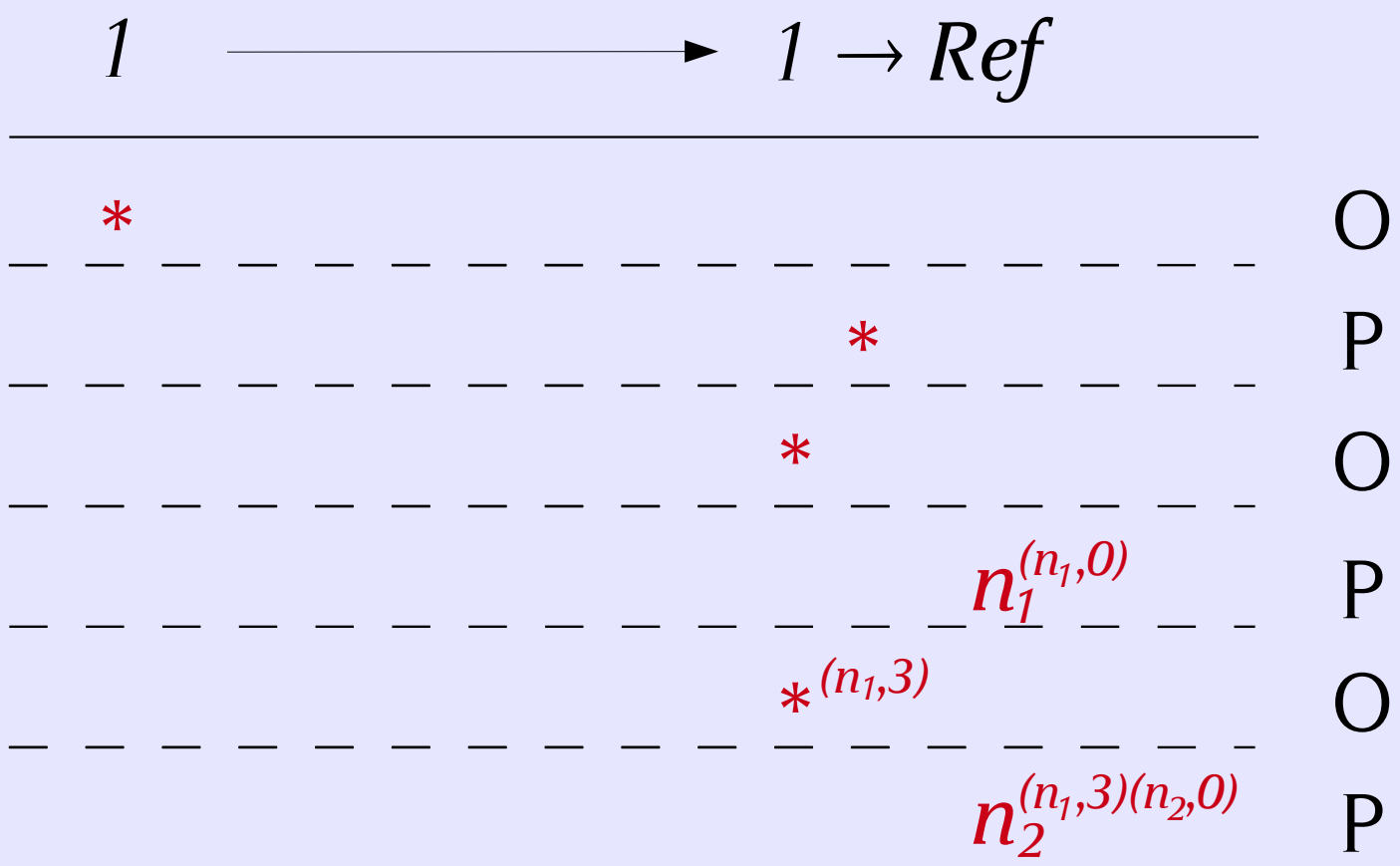
Examples

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



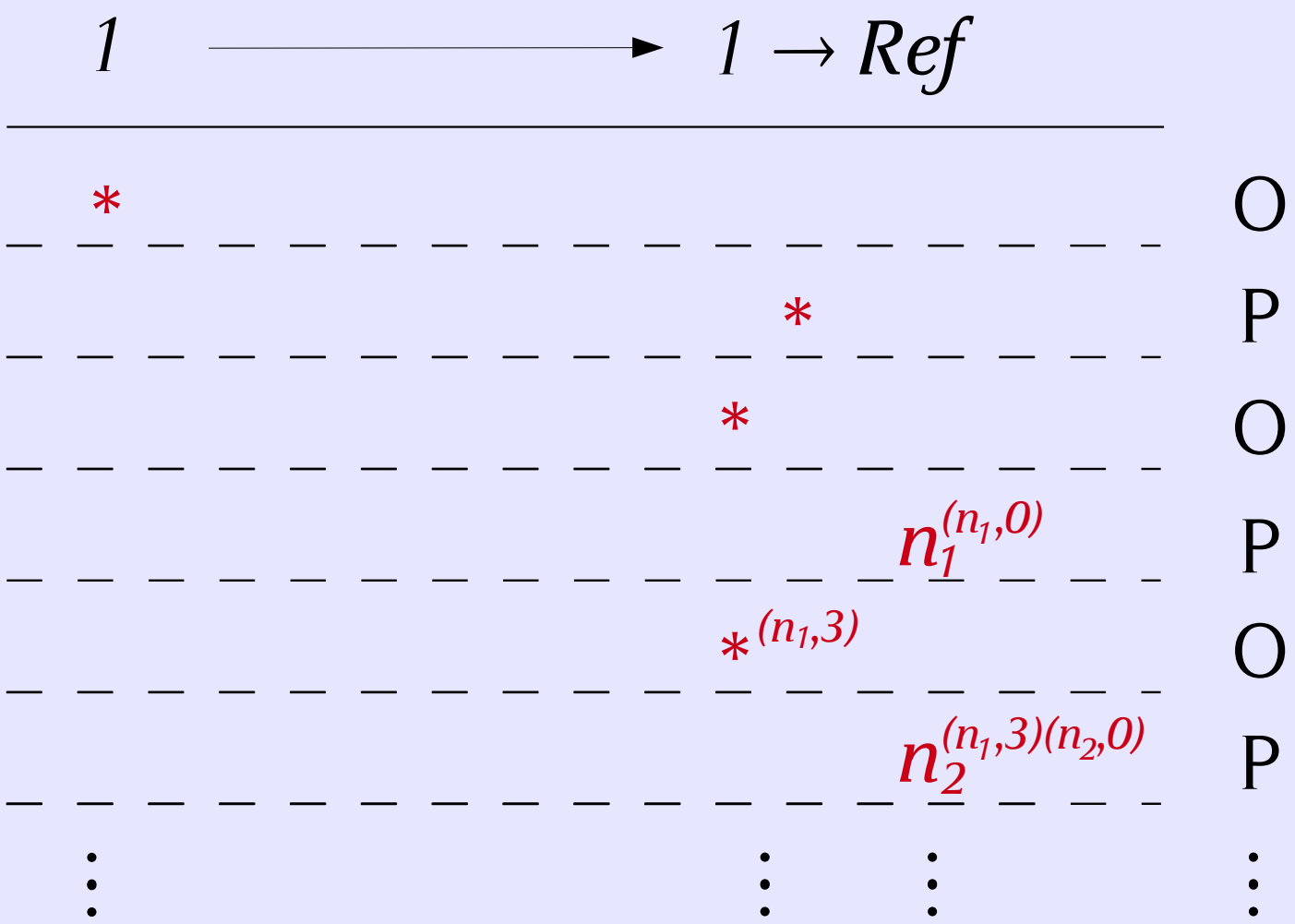
Examples

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



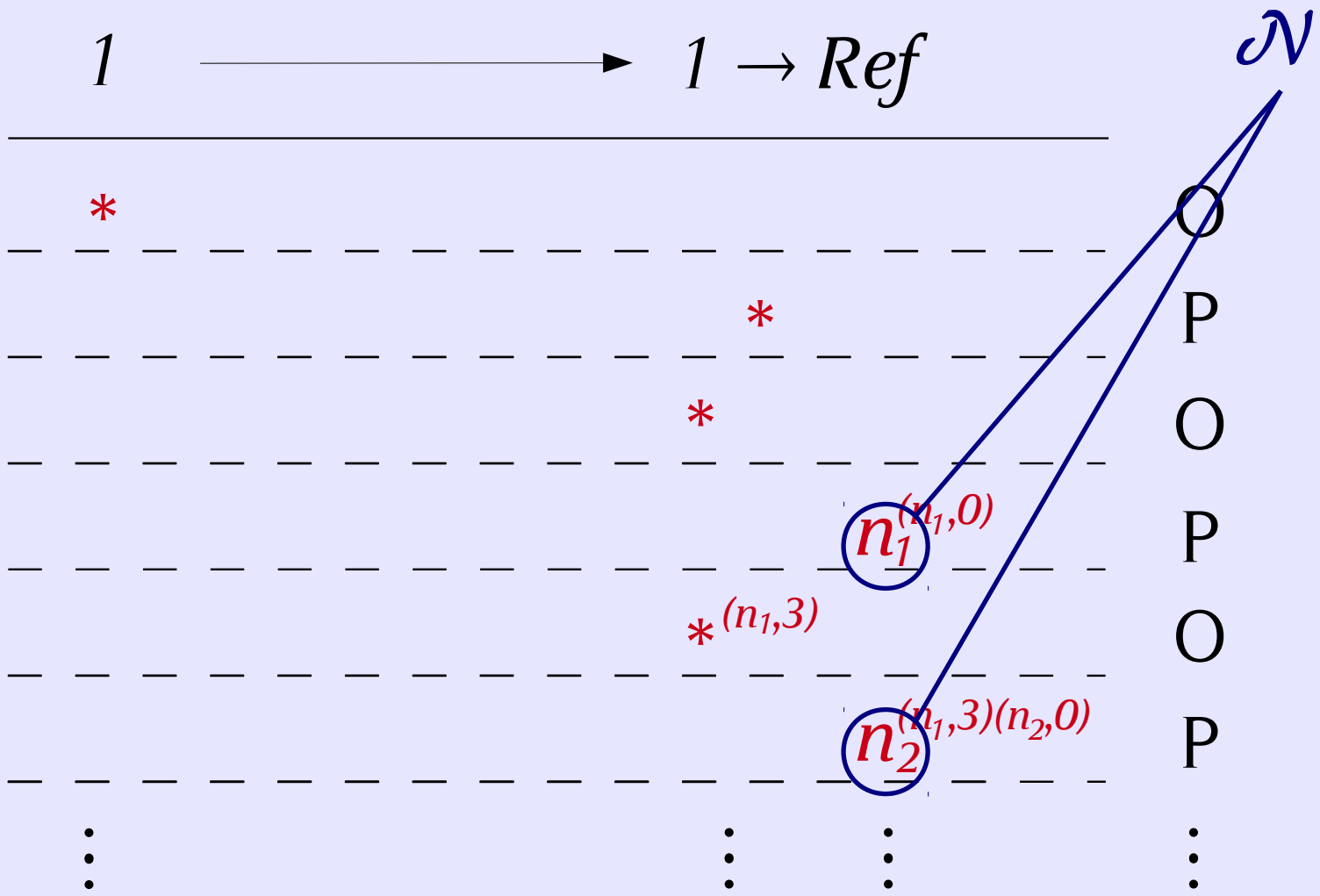
Examples

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



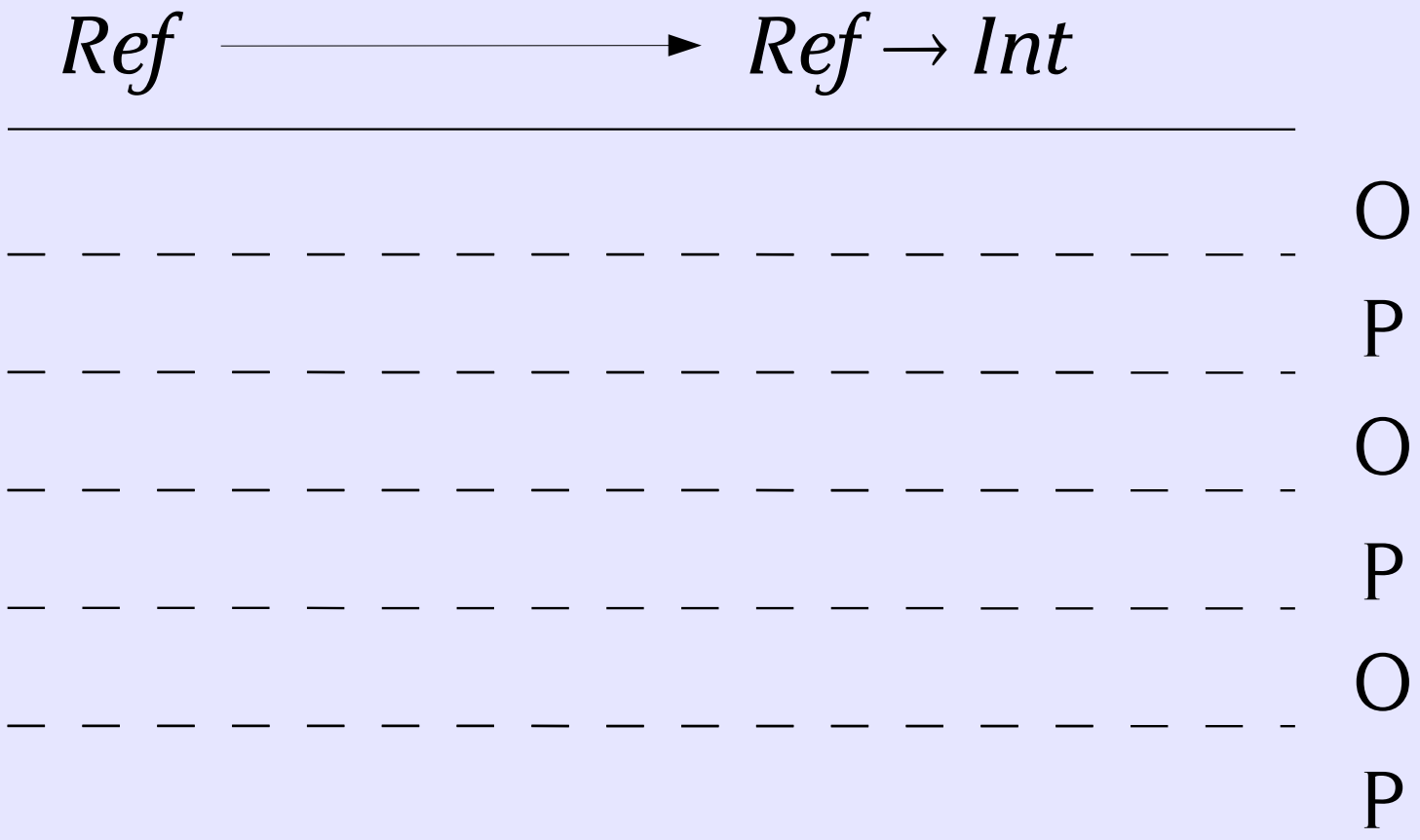
Examples

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



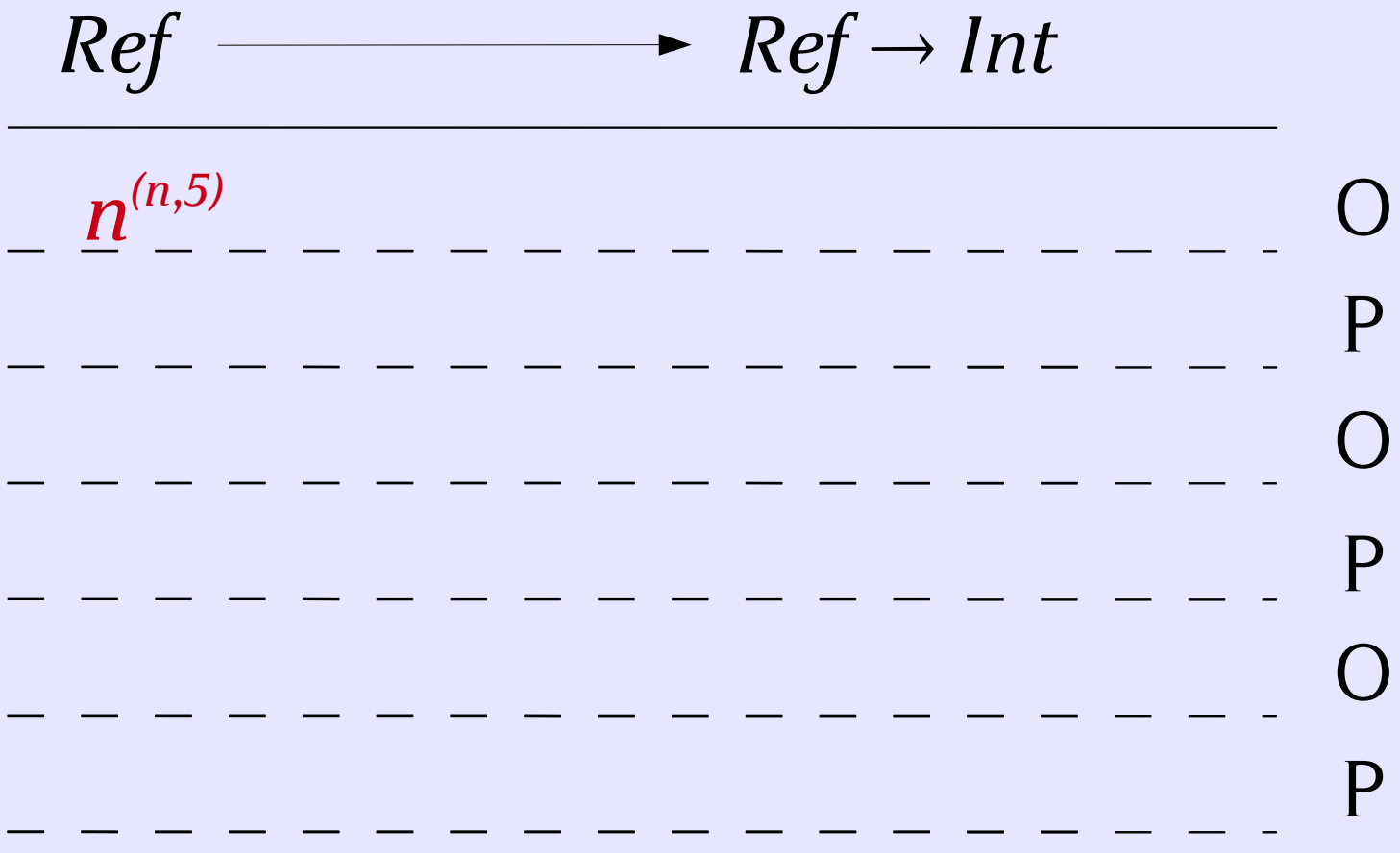
Examples

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



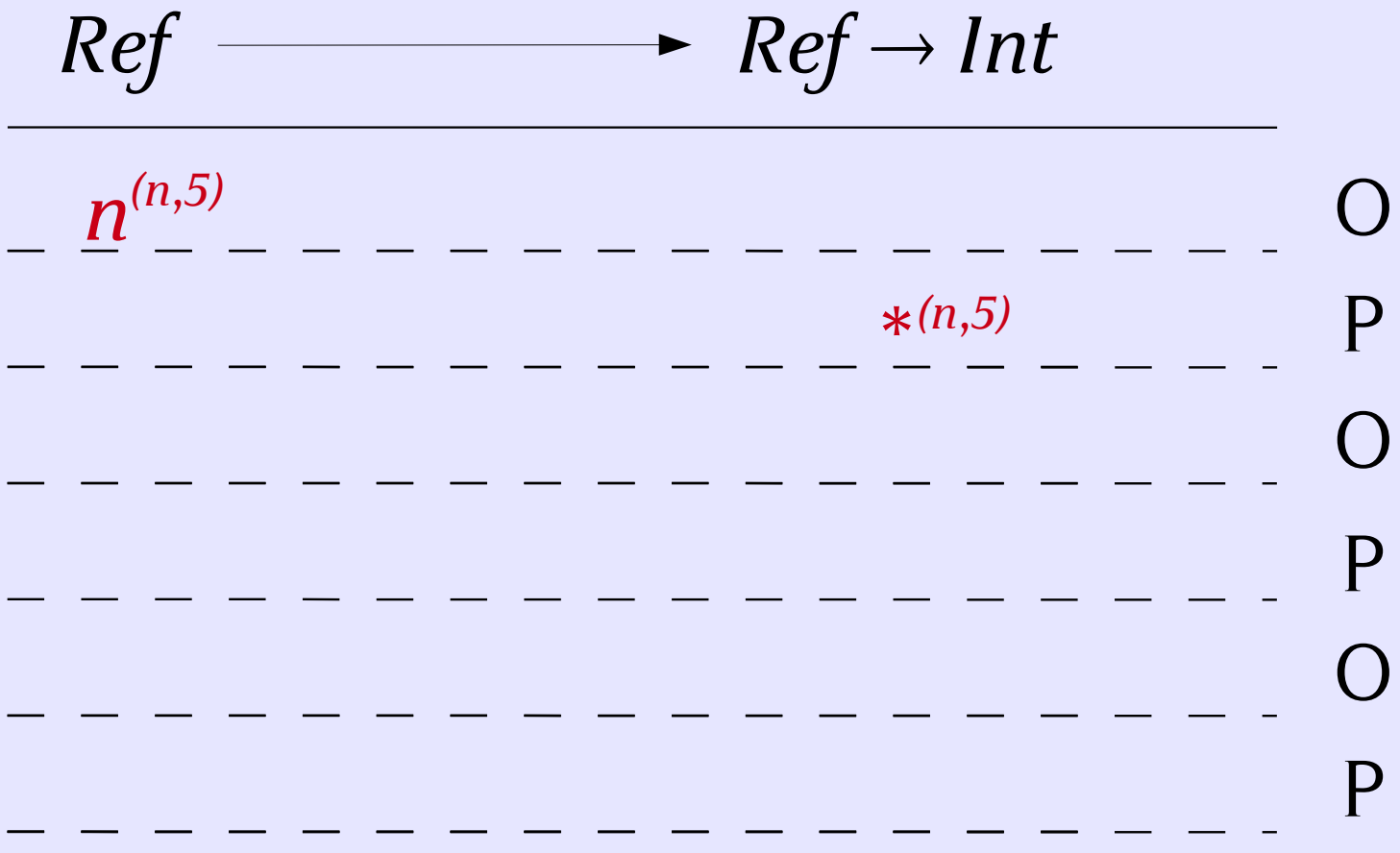
Examples

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



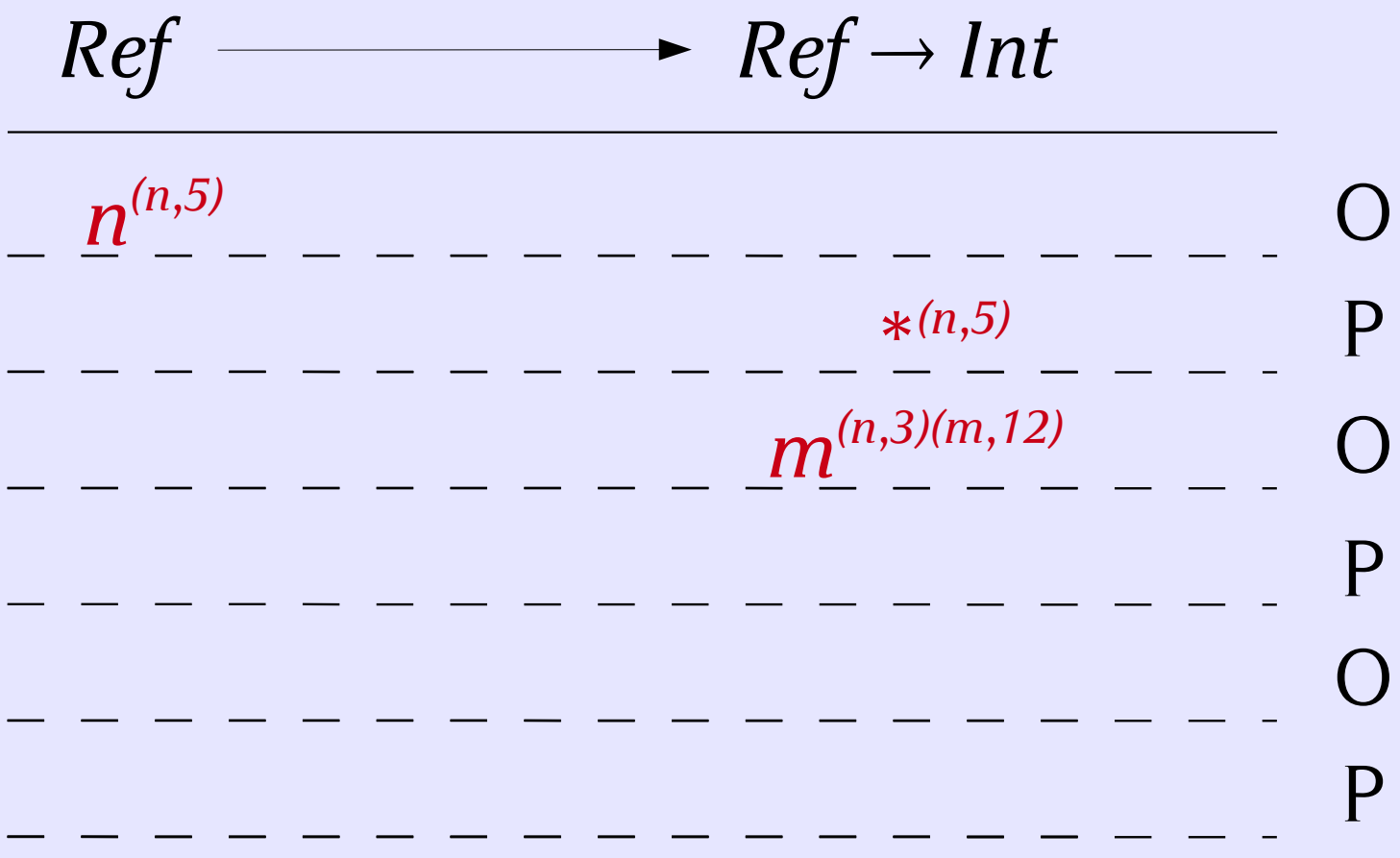
Examples

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



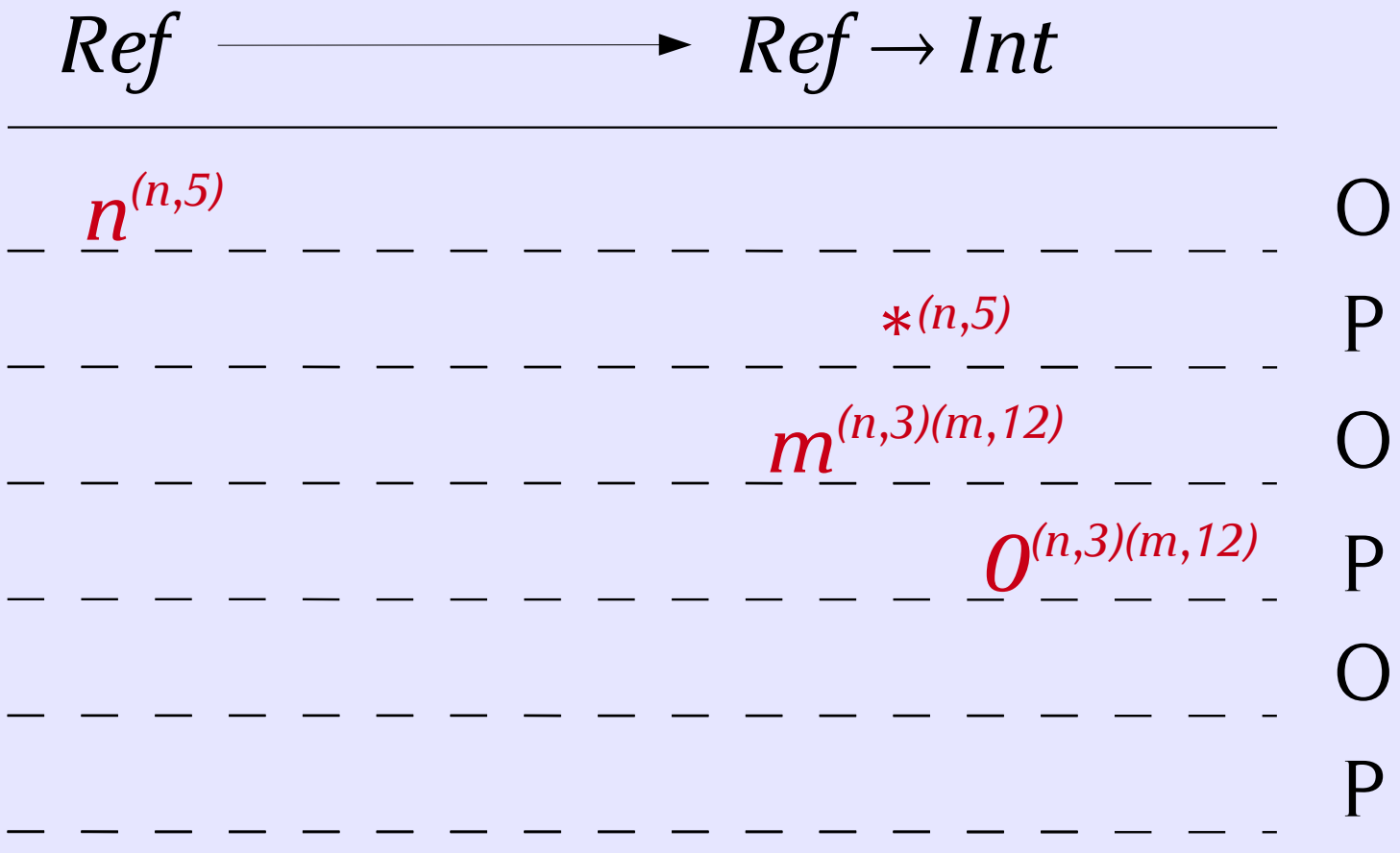
Examples

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



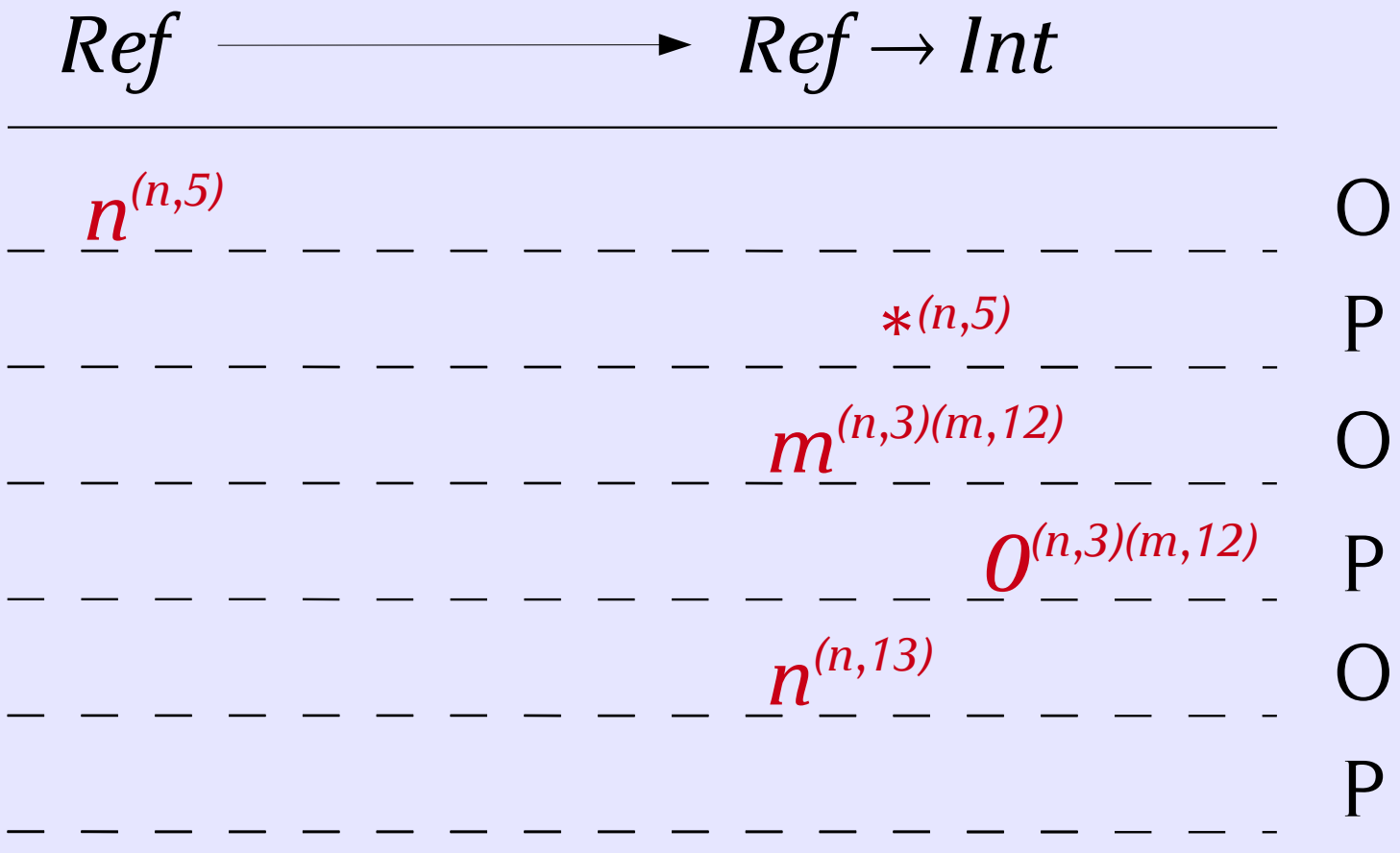
Examples

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



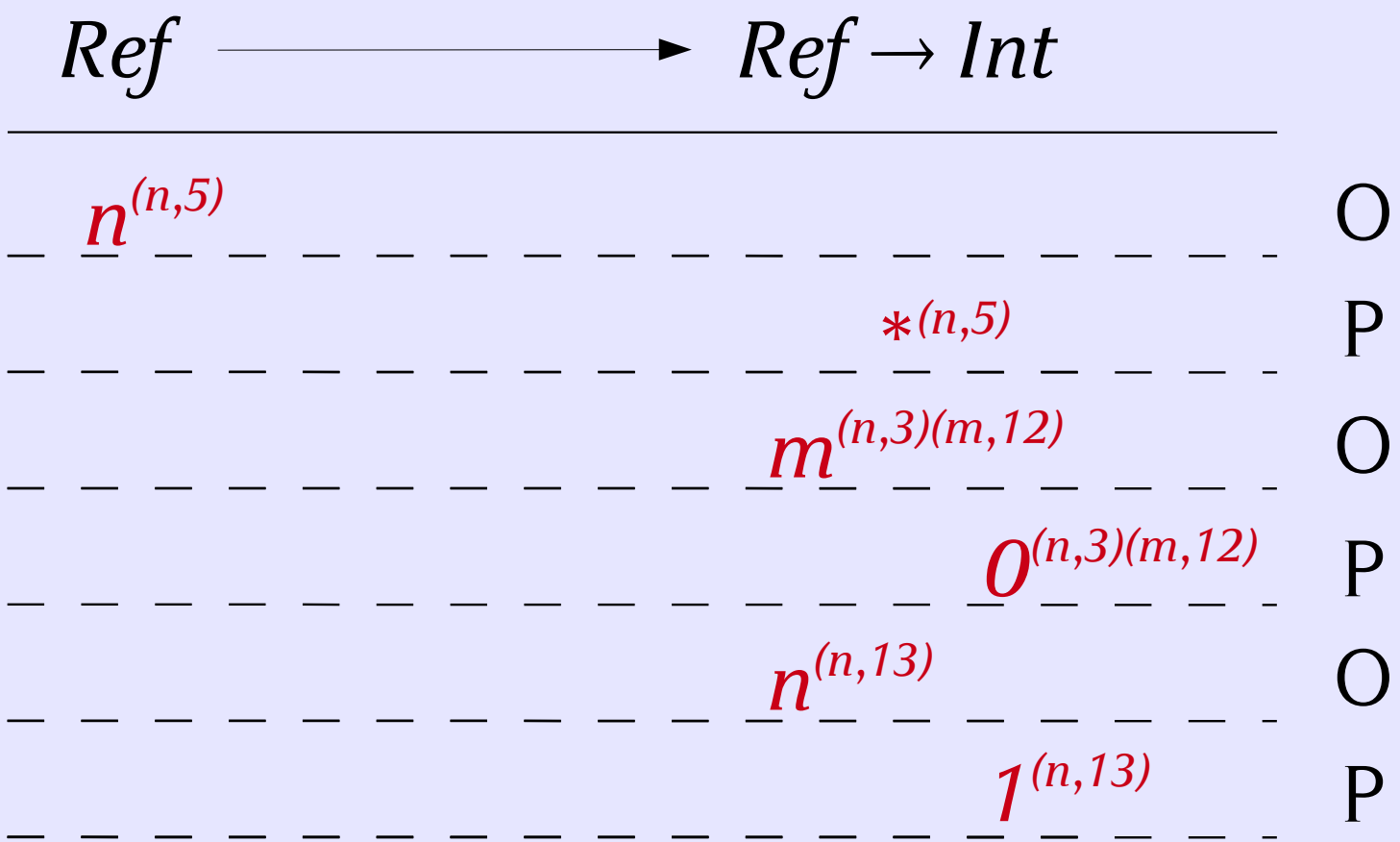
Examples

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



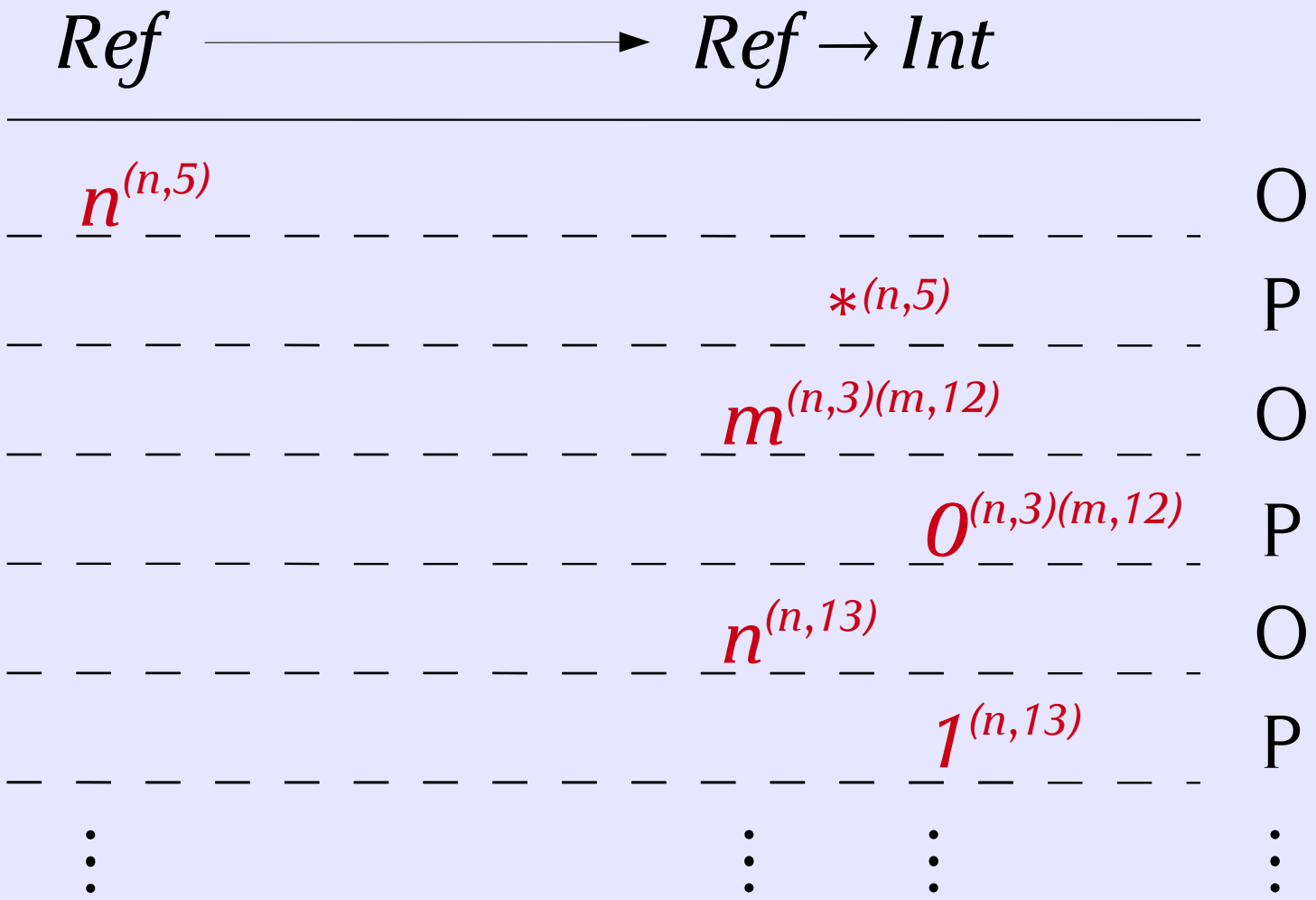
Examples

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



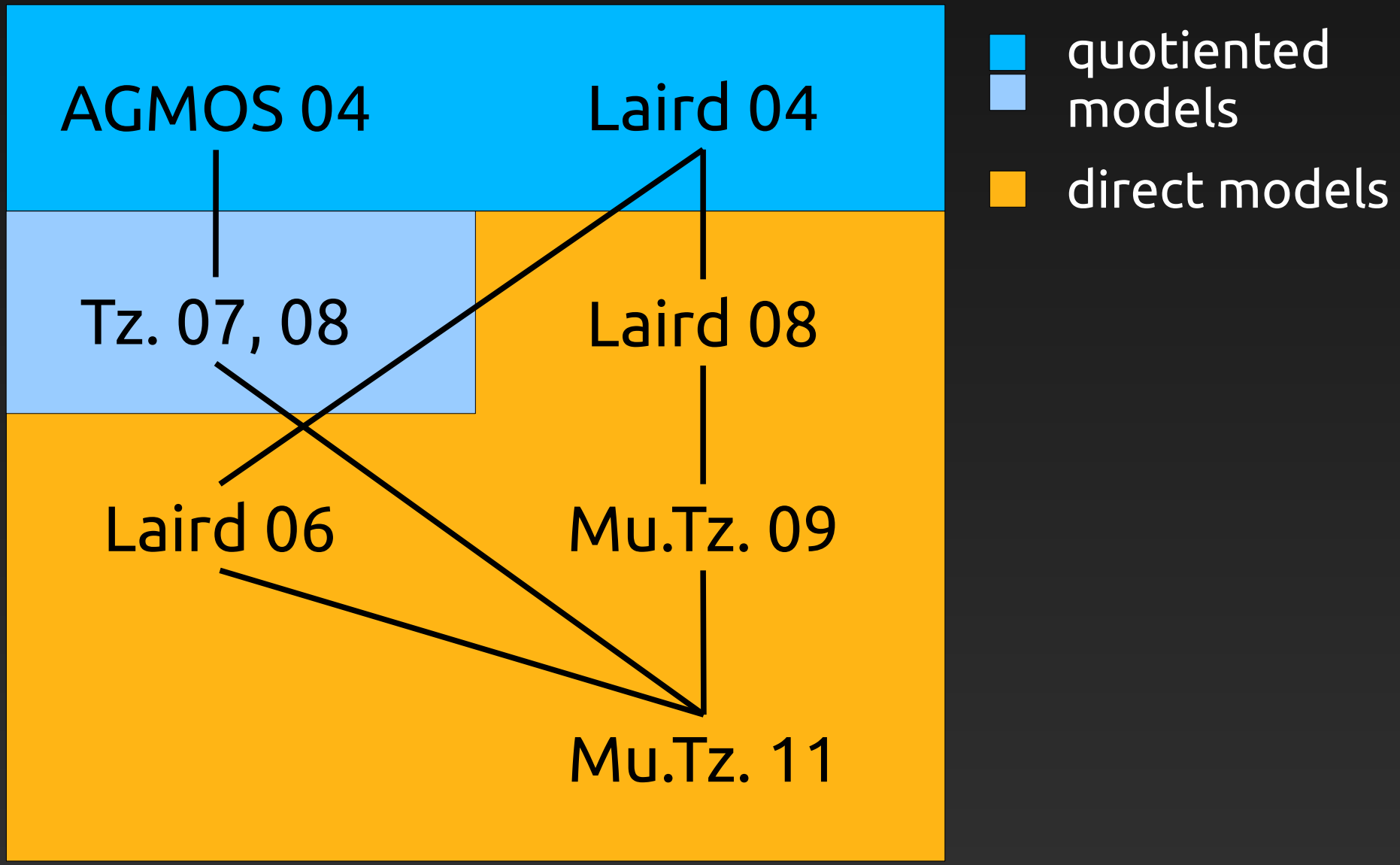
Examples

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

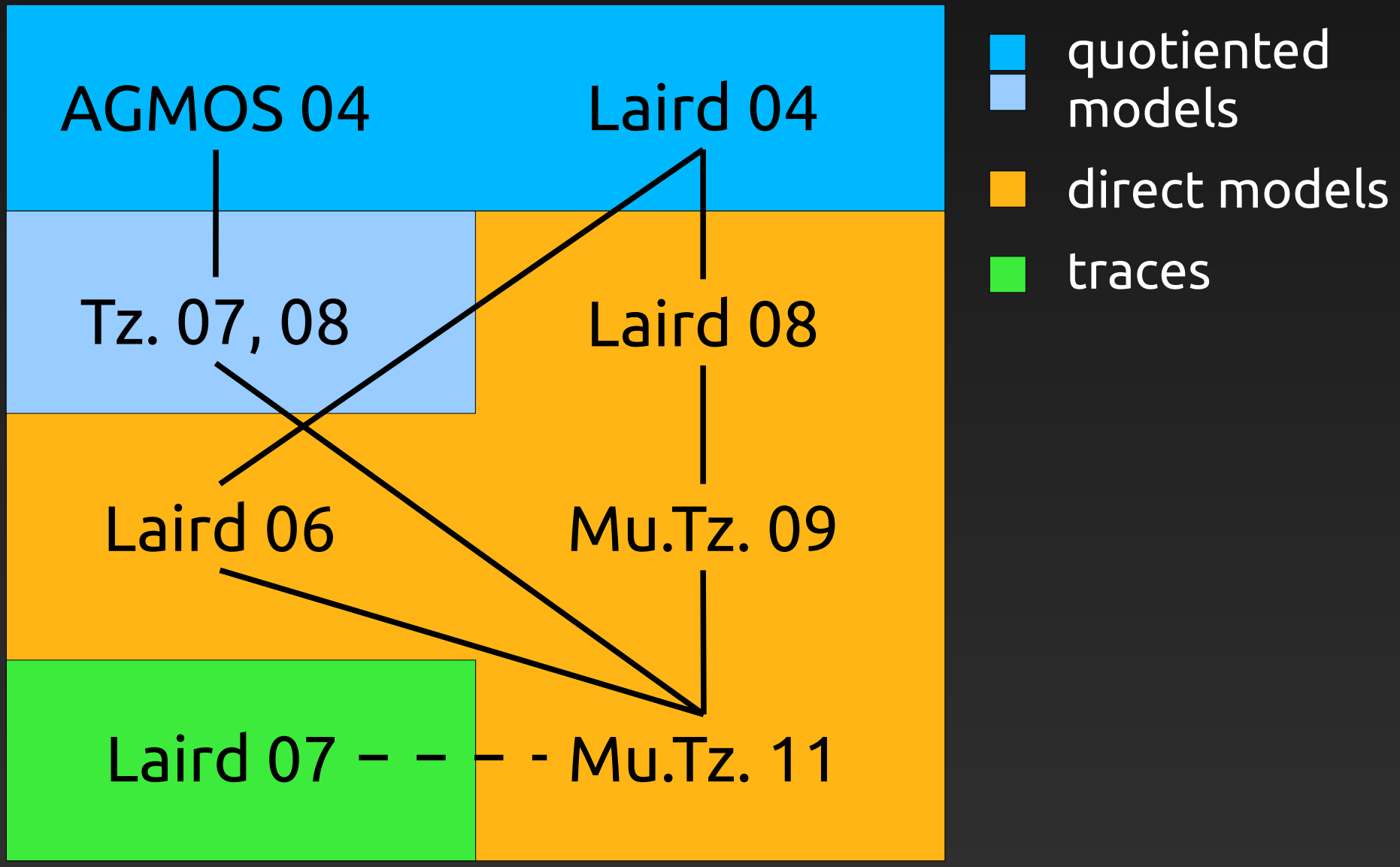


Achievements

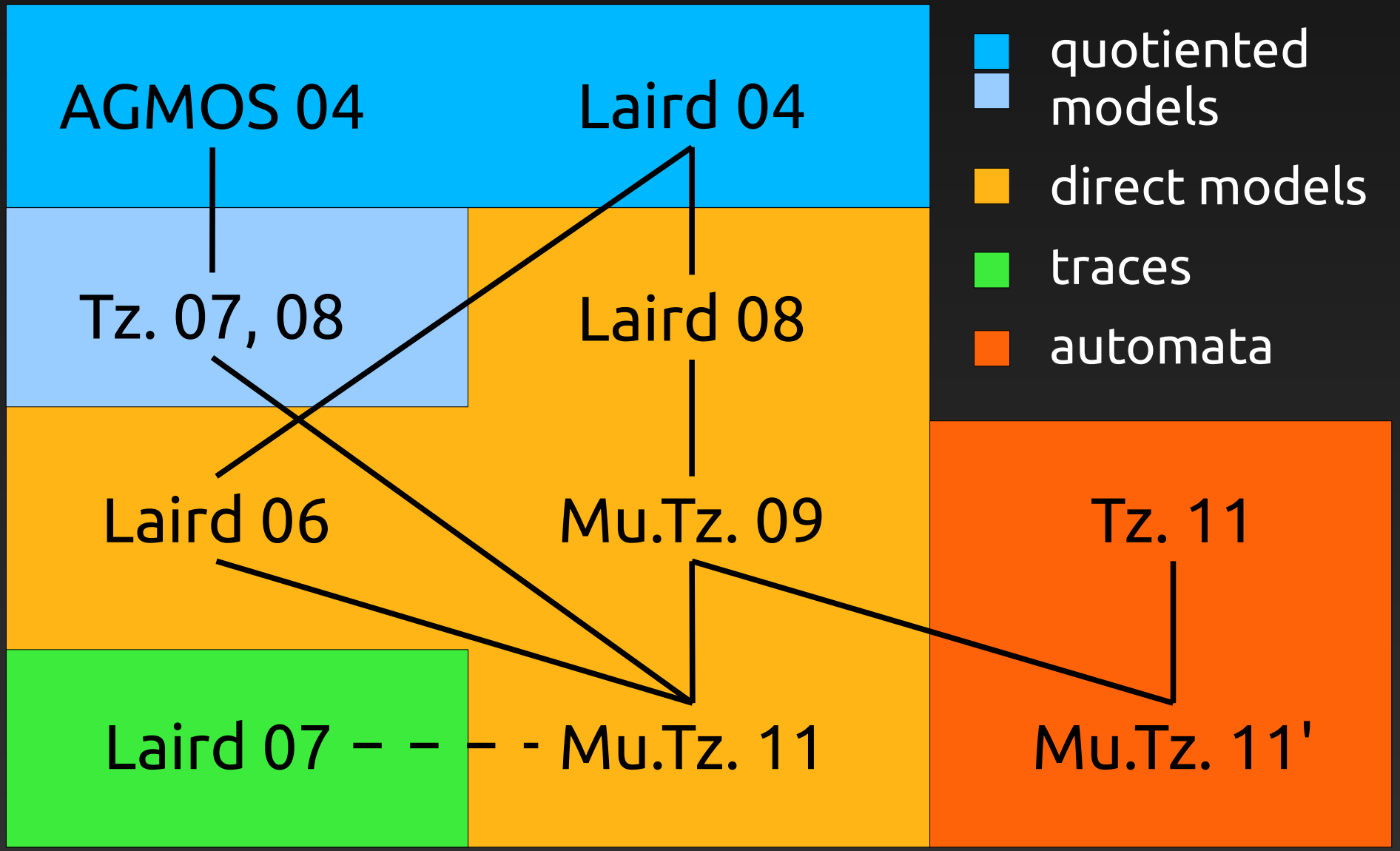
Achievements



Achievements



Achievements



Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: general name-features

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

Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: general name-features

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

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

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

Quotiented models

- AGMOS 04, Laird 04: nu-calculus
- Tz. 07, 08: general name-features

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

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: general 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: general references

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

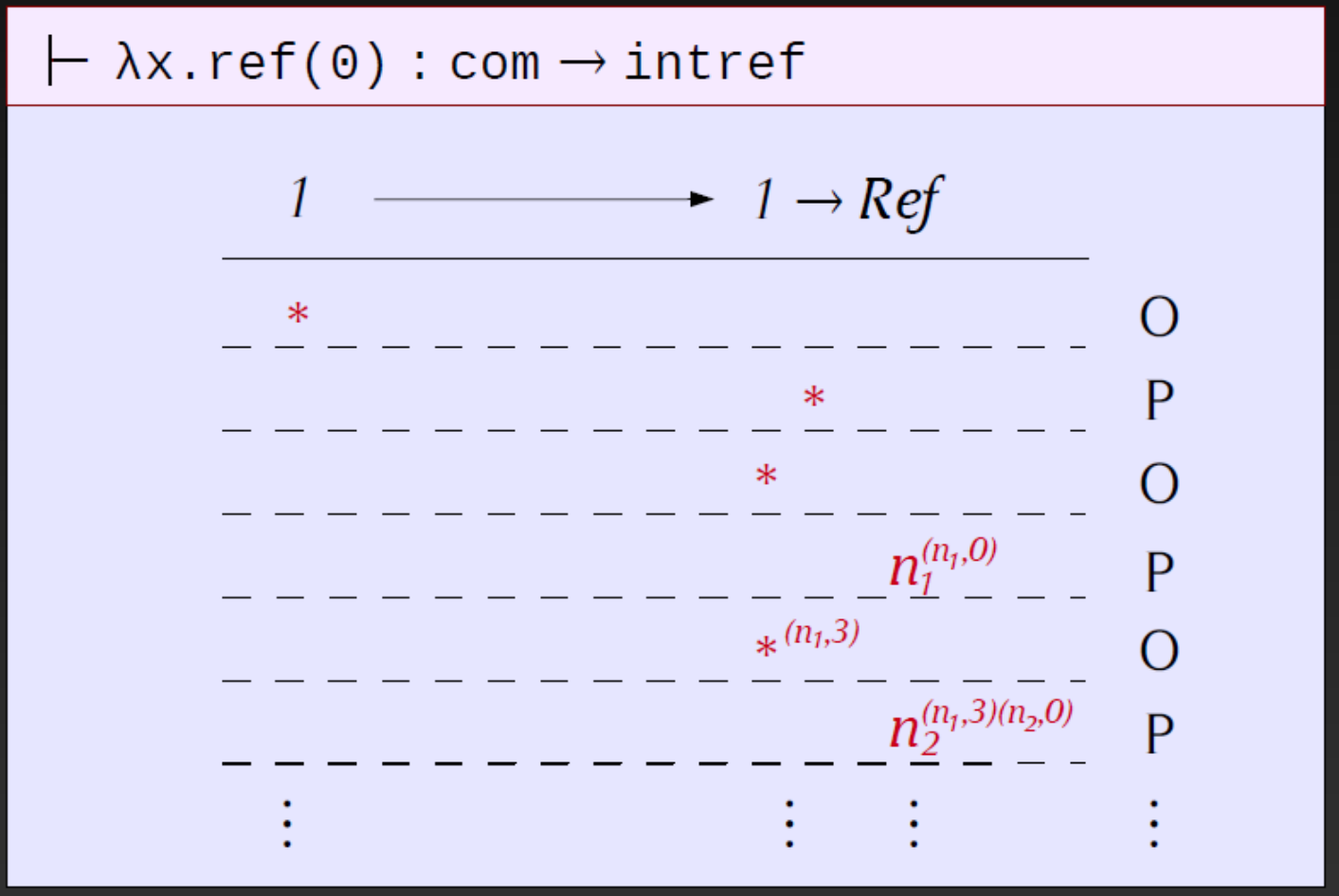
Characteristics:

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

Direct models

- Laird 06: hig
  Laird 08: po
- Mu.Tz. 09: ir
  Mu.Tz. 11: g

$$M \cong N$$



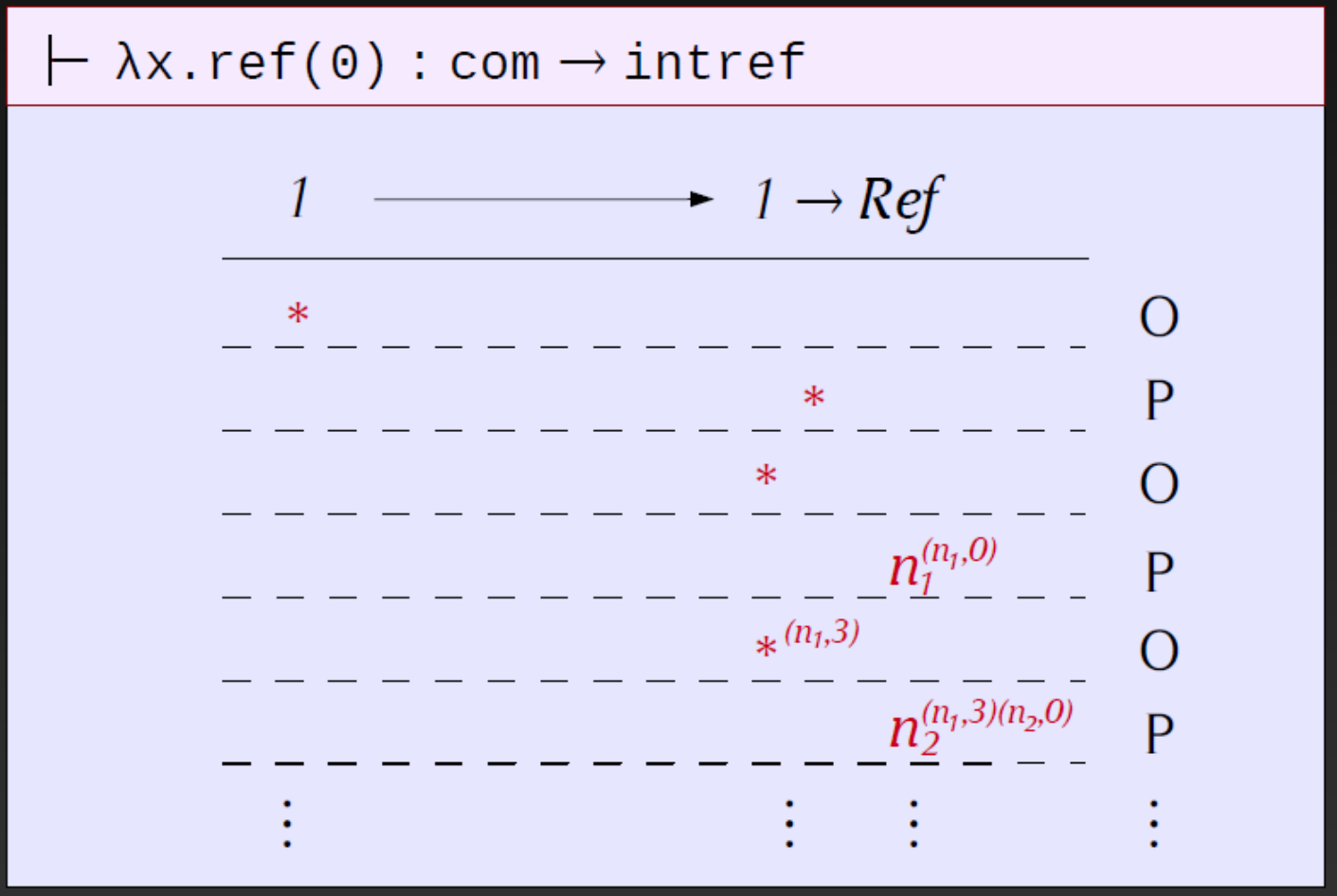
Characteristics

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

Direct models

- Laird 06: hig
  Laird 08: po
- Mu.Tz. 09: ir
  Mu.Tz. 11: g

$$M \cong N$$



Characteristics

- moves involving names/ moves-with-store
- name-availability conditions/ "d

Algorithmics

Fresh-register automata

$$\texttt{λz.ref(0)} \longmapsto \{\ * * * n_1 * n_2 * n_3 \ldots \mid\ n_i\text{'s distinct}\ \}$$

Fresh-register automata

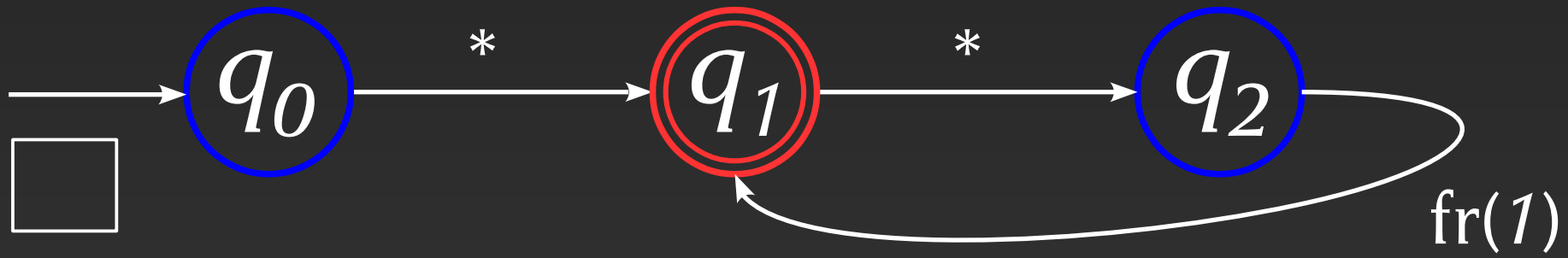
$\lambda z.\texttt{ref}(0) \longmapsto \{ * * * n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$

Automata with names

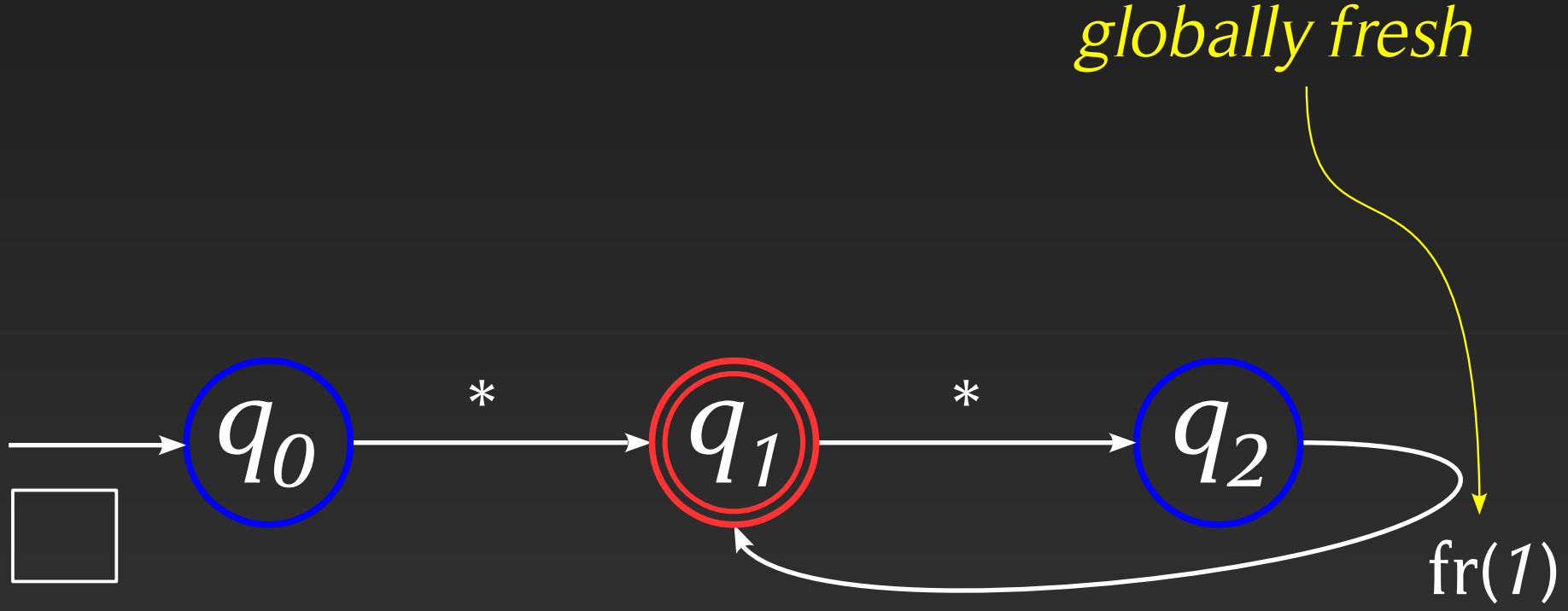
- Infinite alphabet $\mathcal{N}$
- Freshness recognition

Finite-state machines with registers

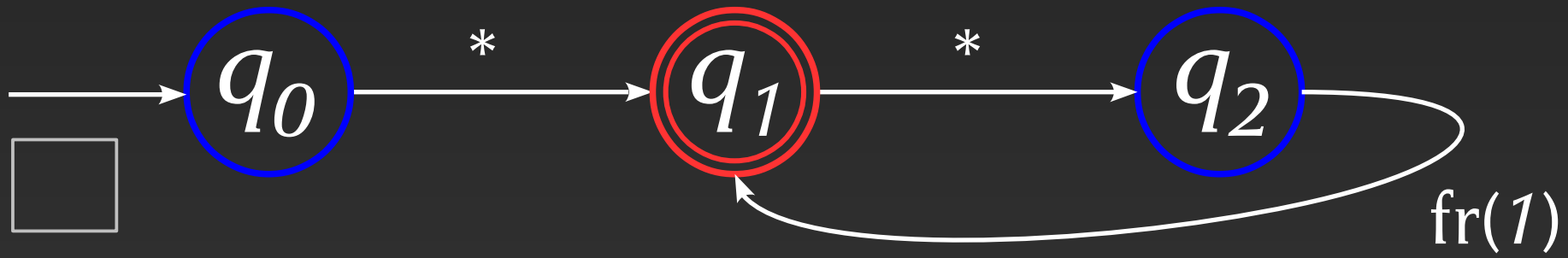
Fresh-register automata



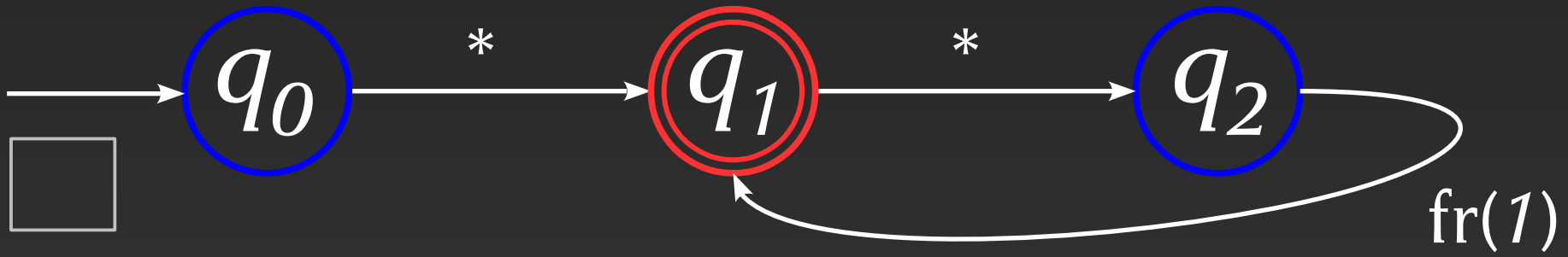
Fresh-register automata



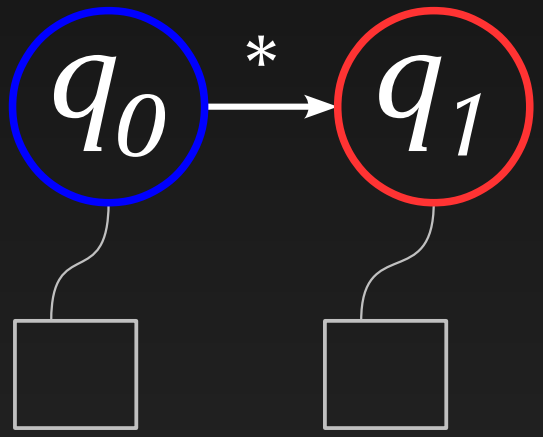
Fresh-register automata

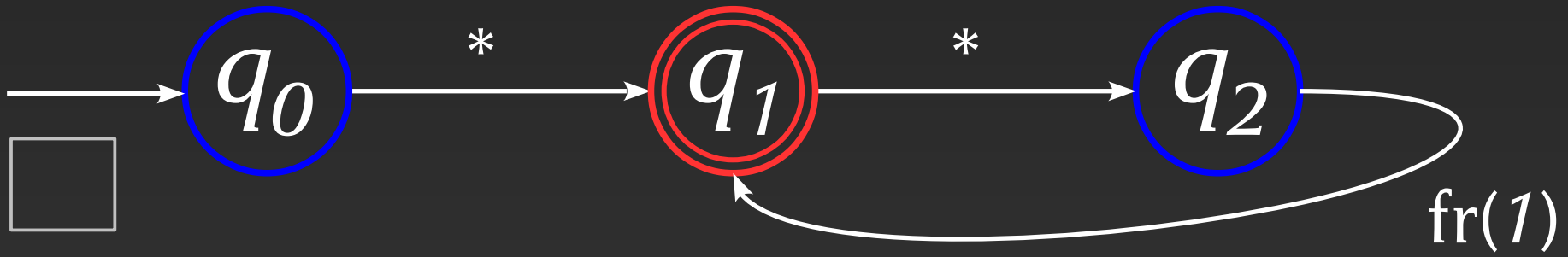


Fresh-register automata

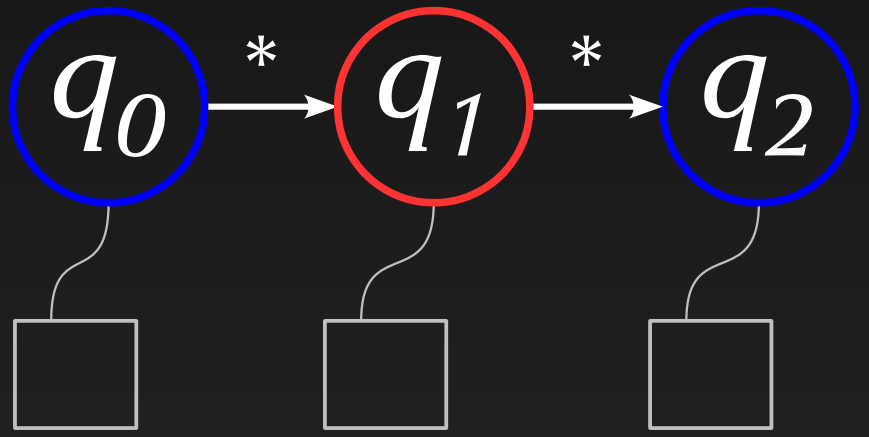


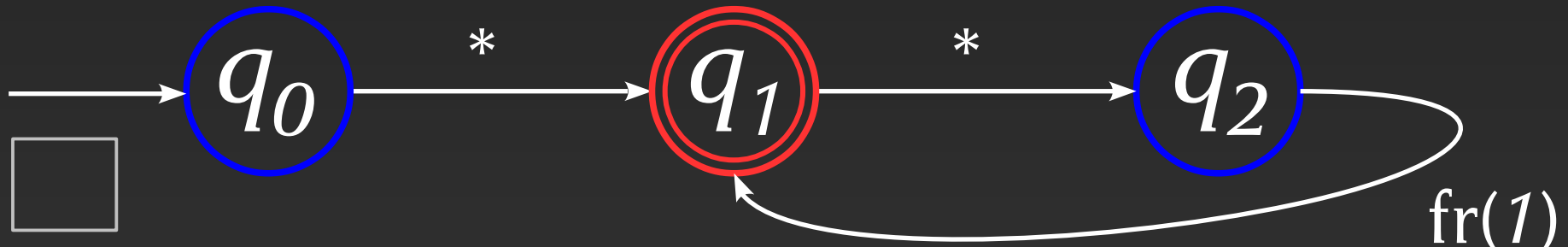
Fresh-register automata



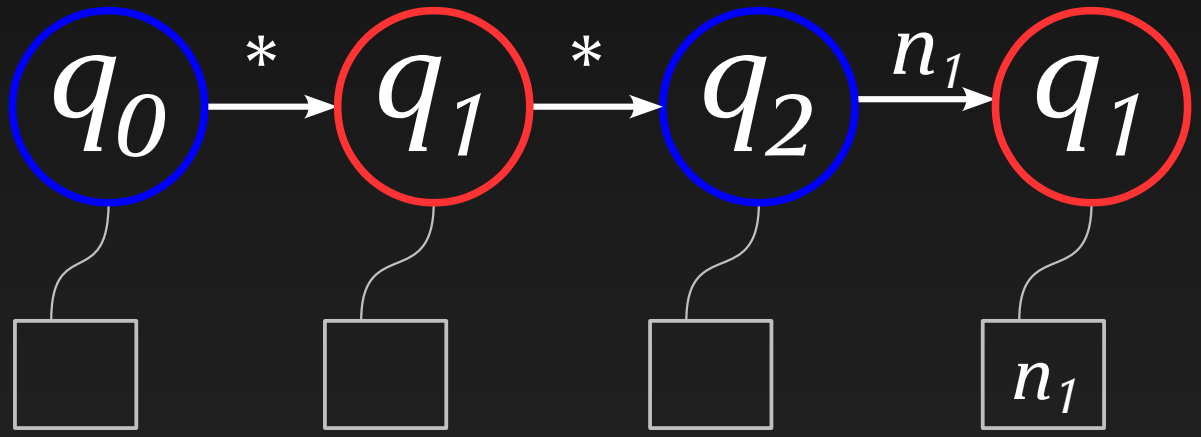


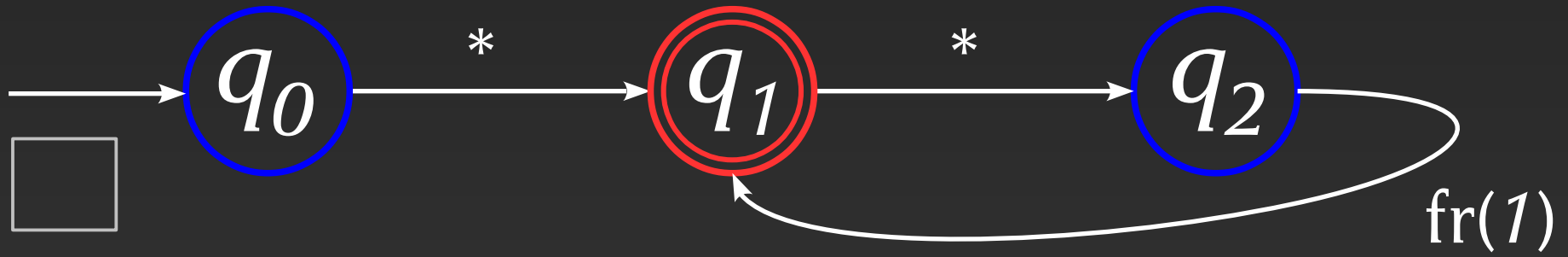
Fresh-register automata



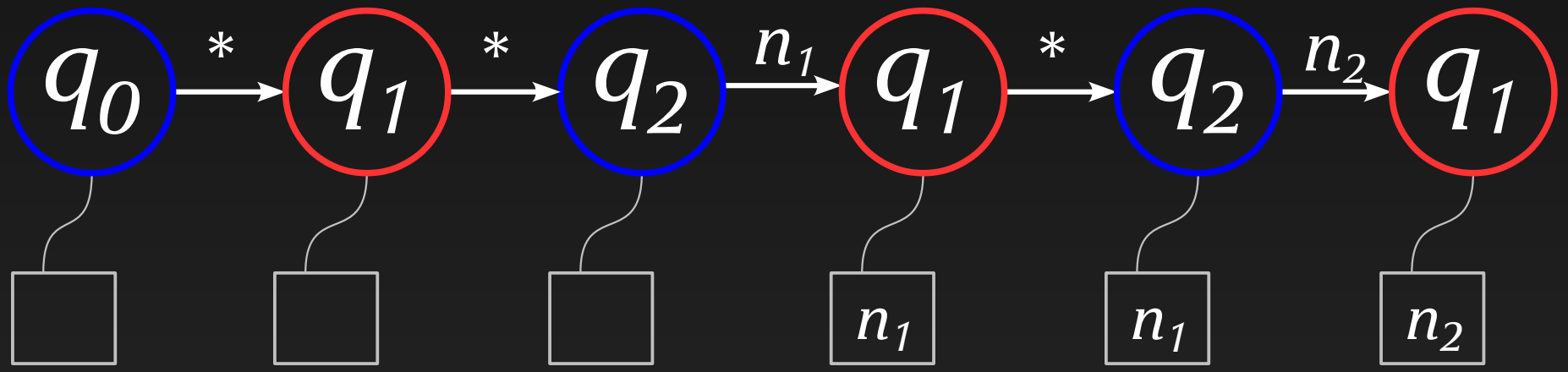


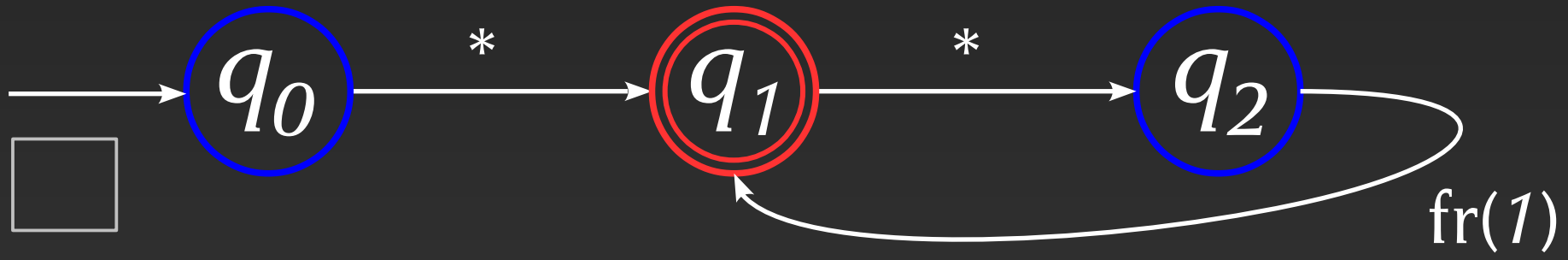
Fresh-register automata



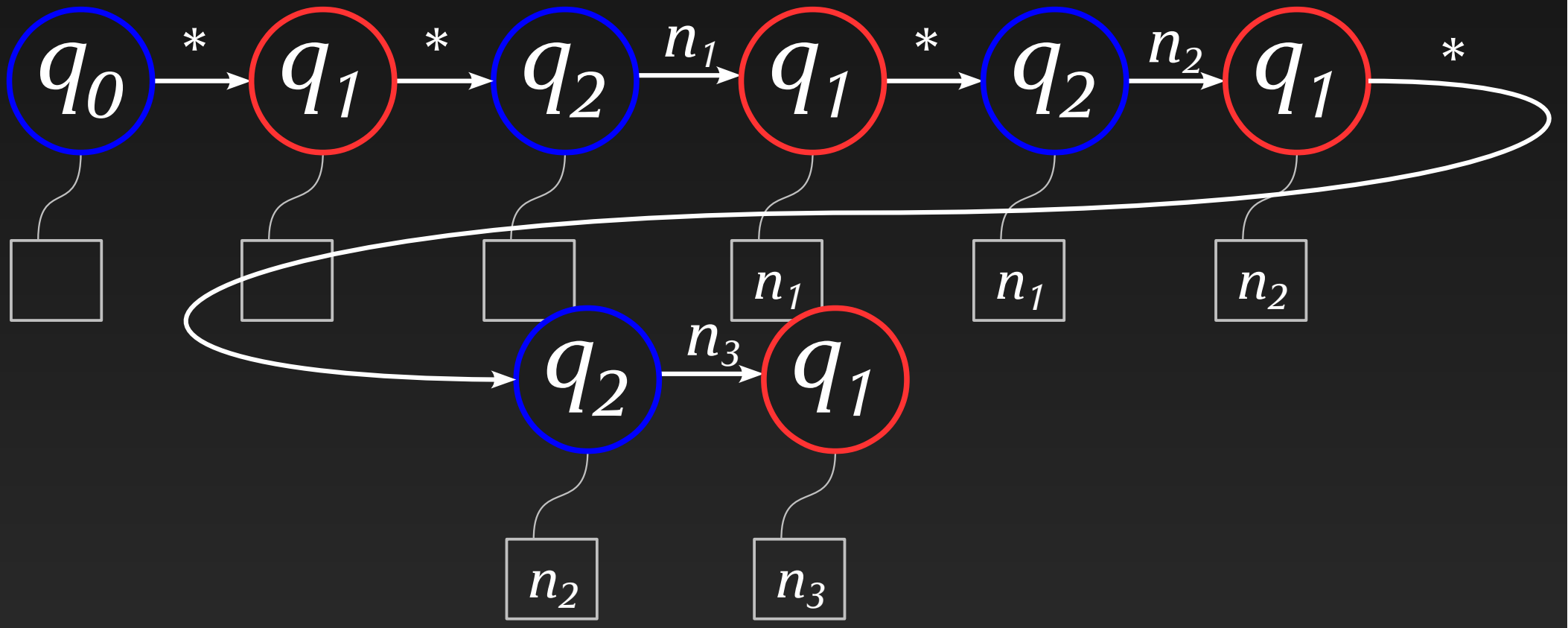


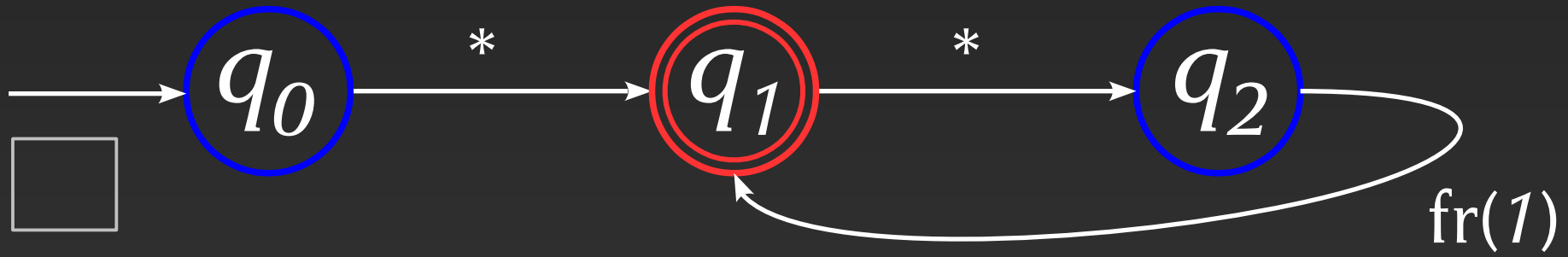
Fresh-register automata



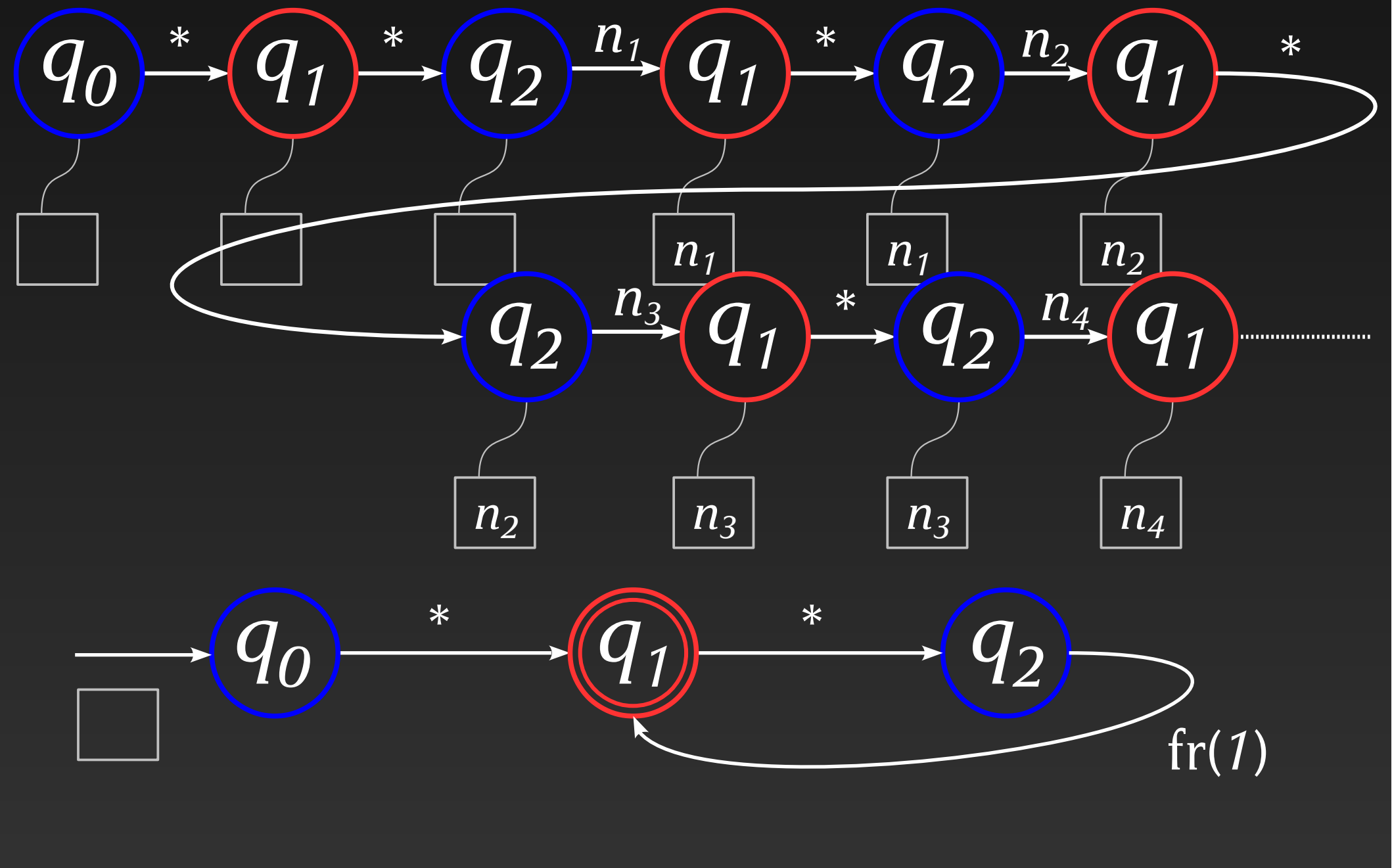


Fresh-register automata

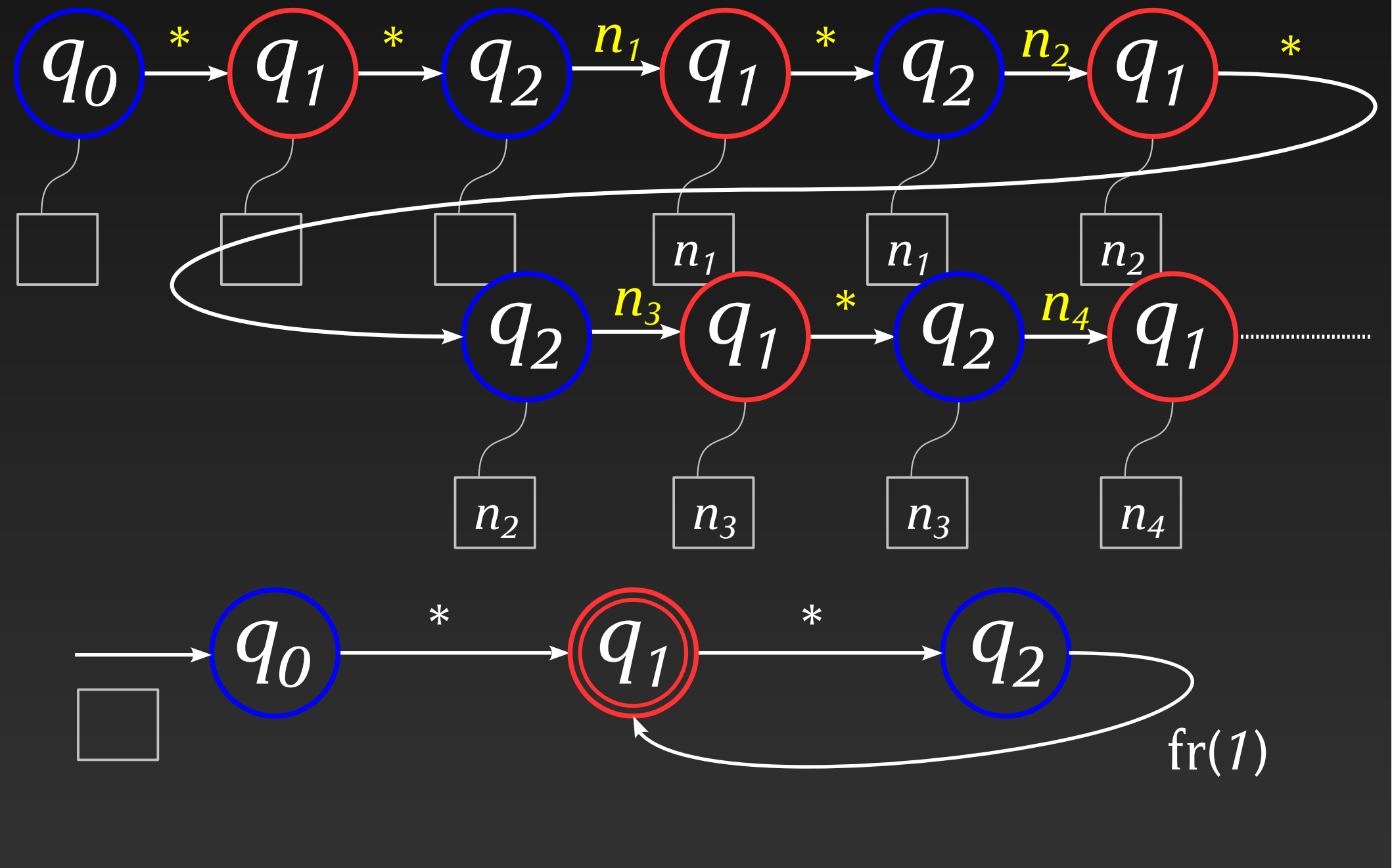




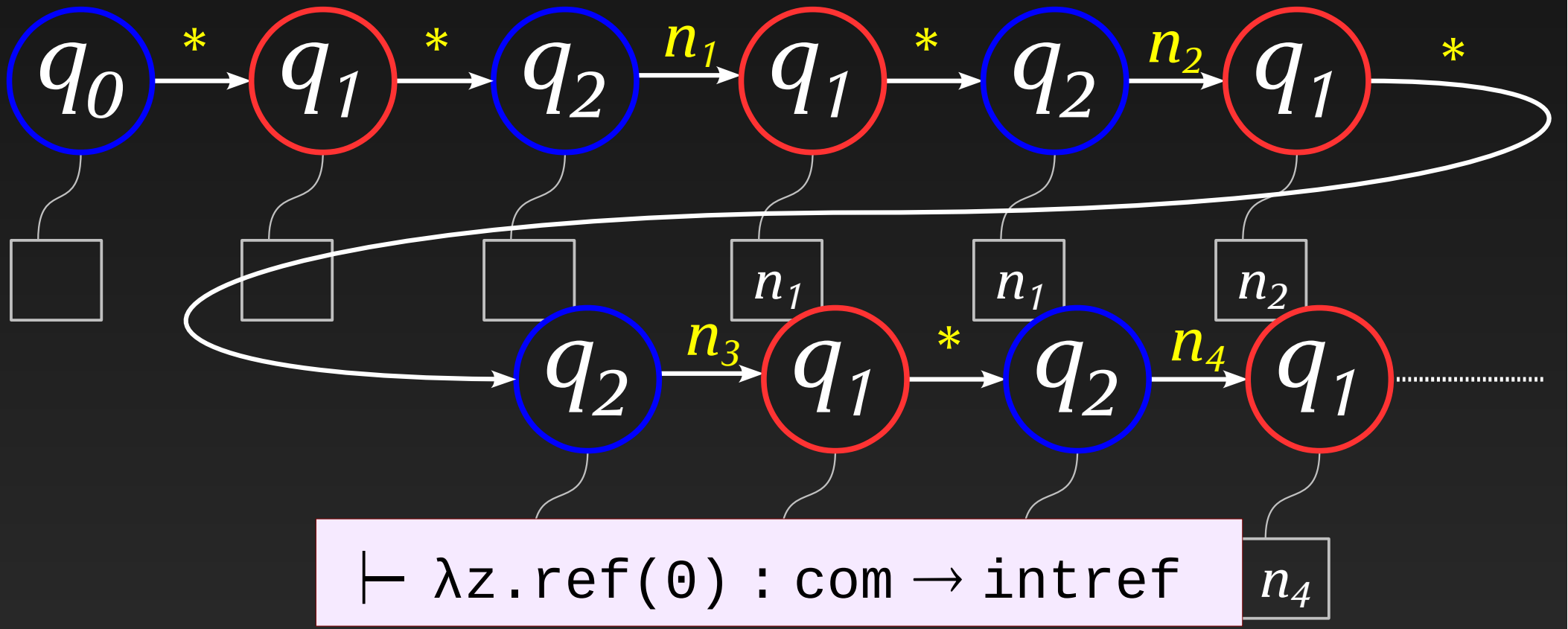
Fresh-register automata



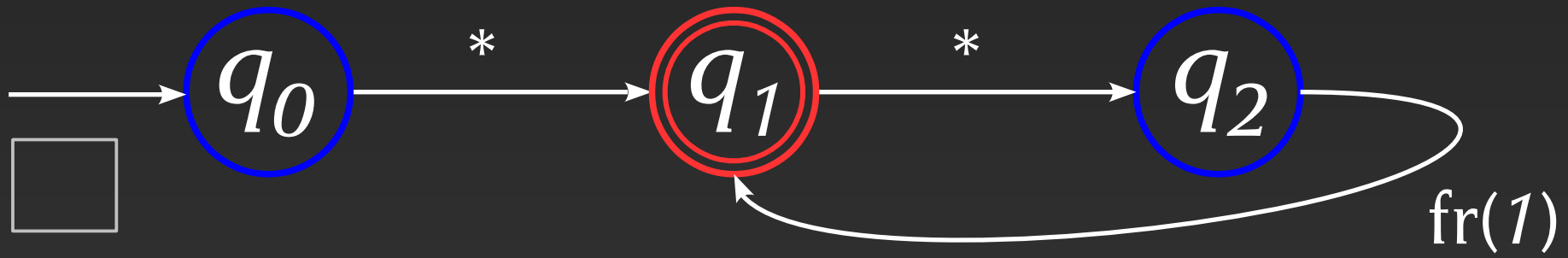
Fresh-register automata



Fresh-register automata

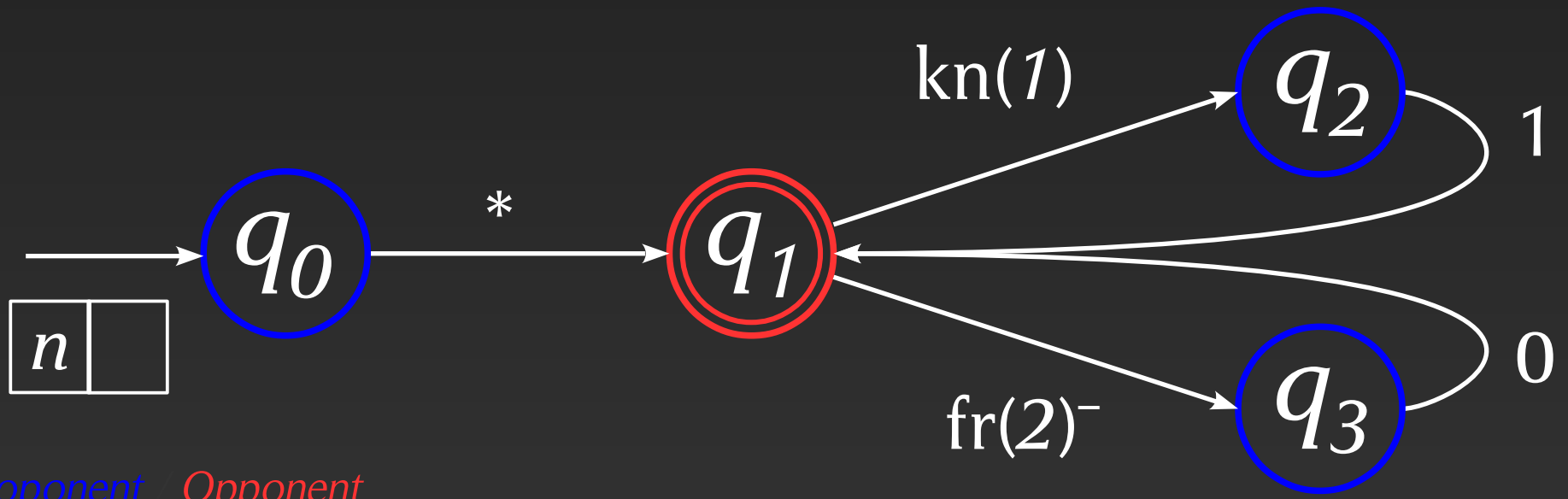


$\vdash \lambda z.\mathtt{ref}(0) : \mathtt{com} \to \mathtt{intref}$

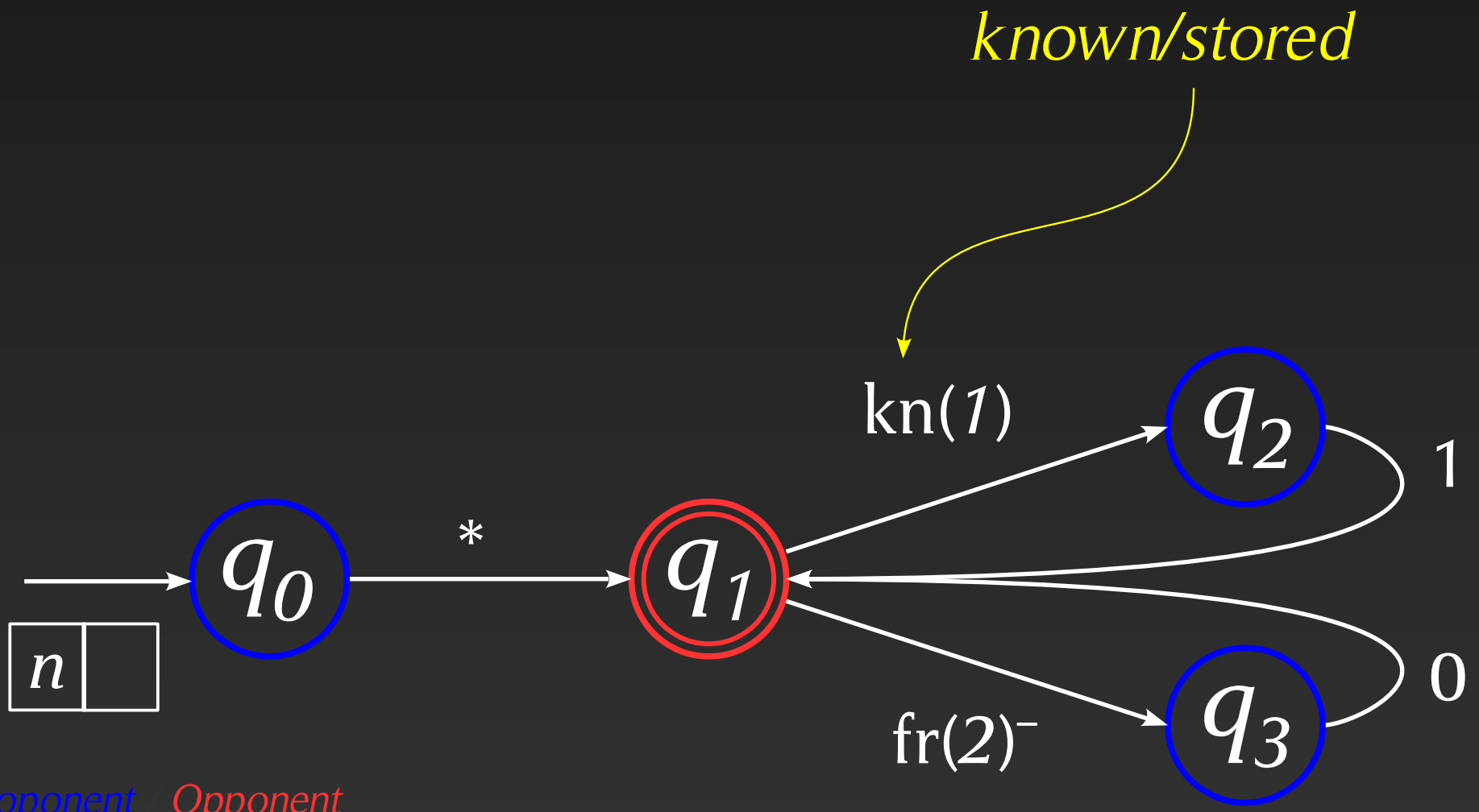


Proponent *Opponent*

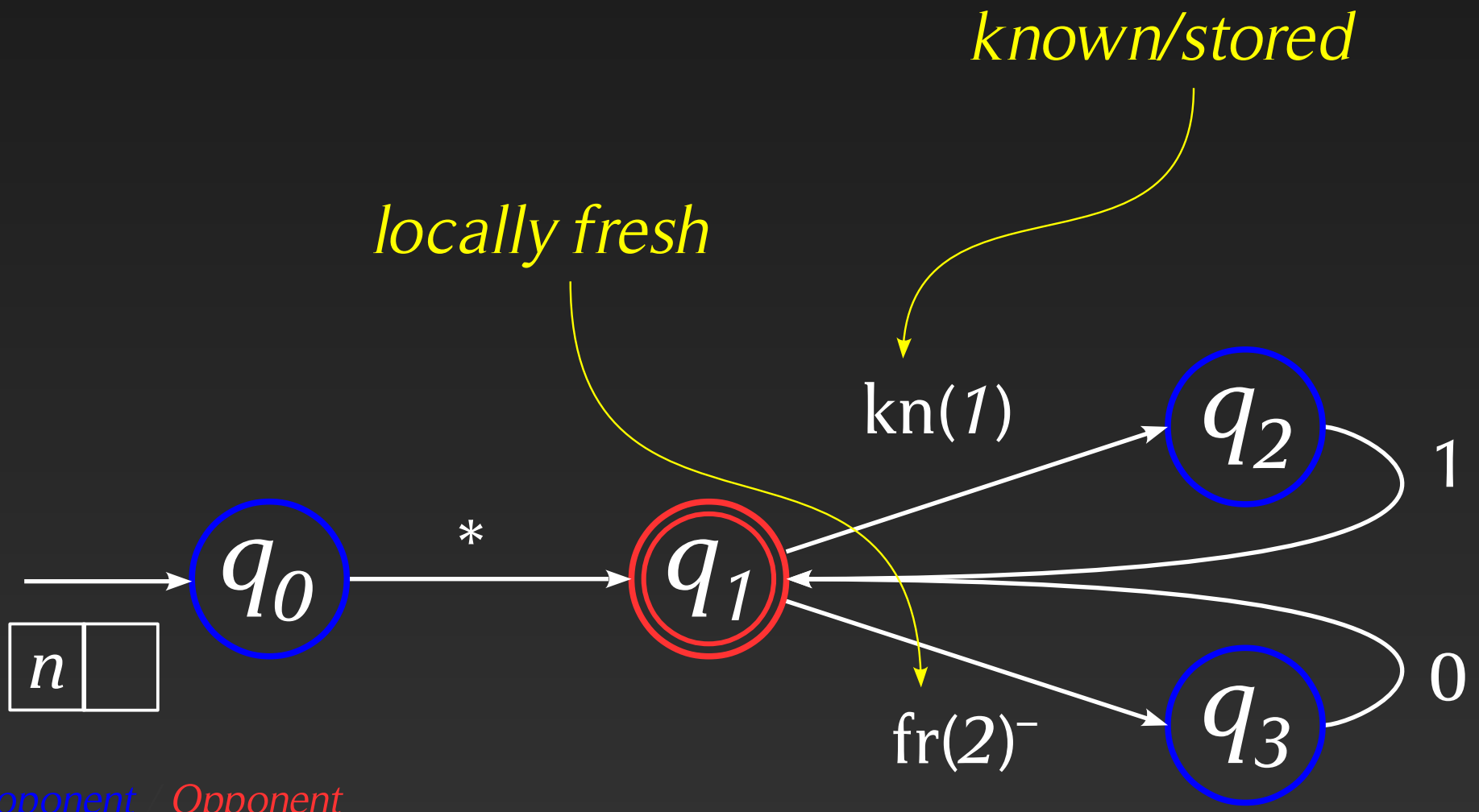
Fresh-register automata



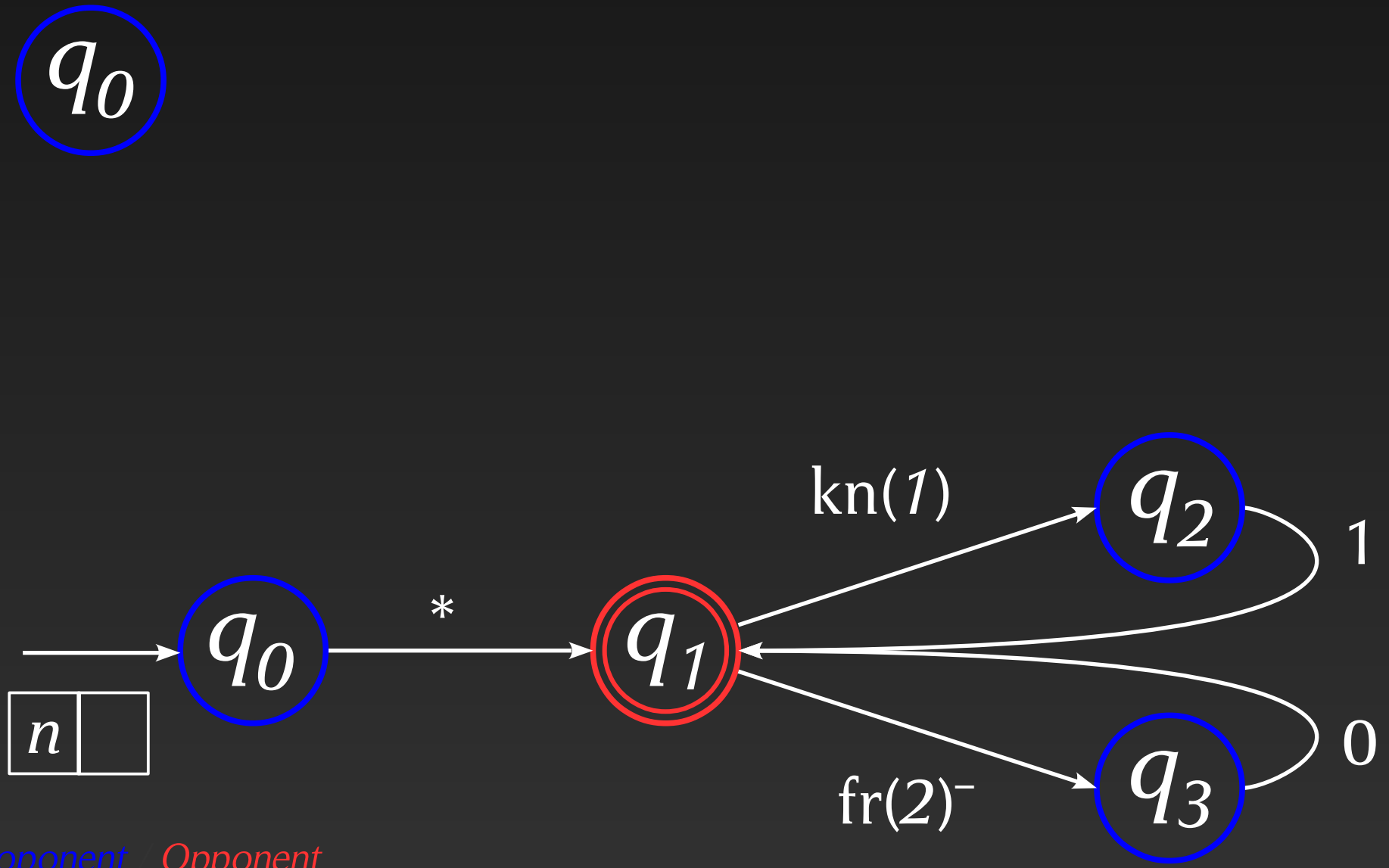
Fresh-register automata



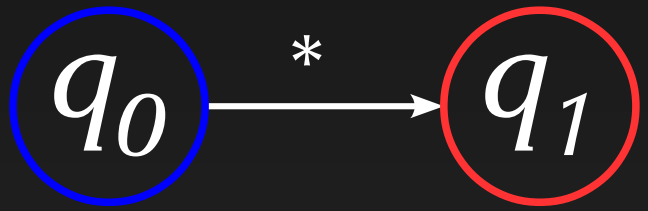
Fresh-register automata

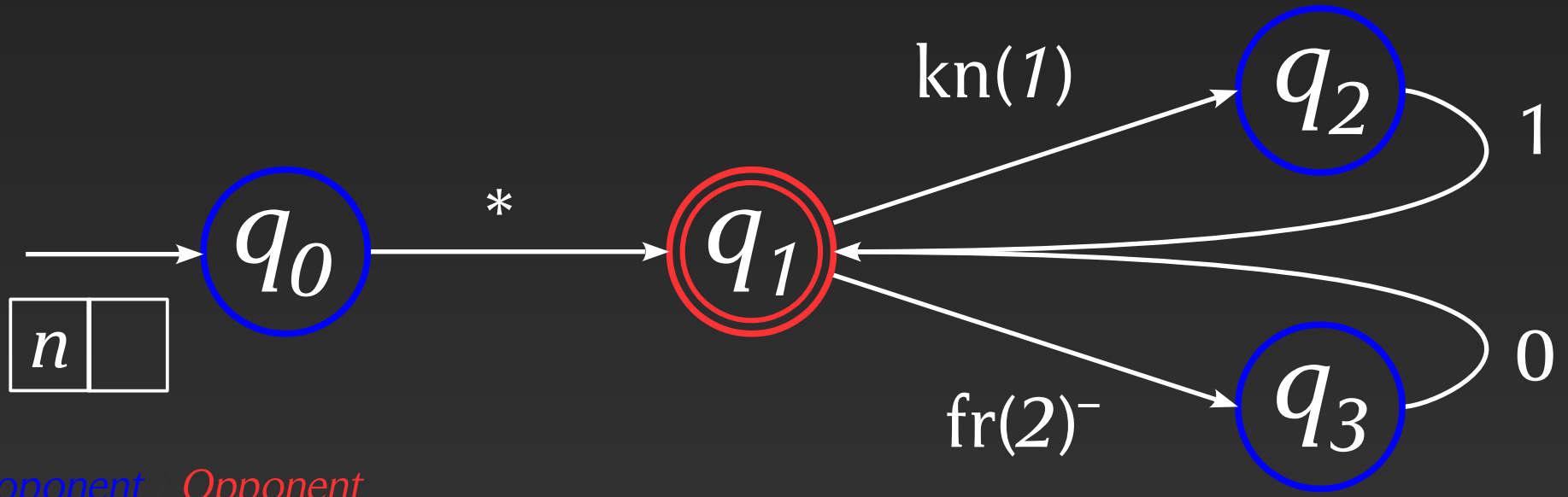


Fresh-register automata

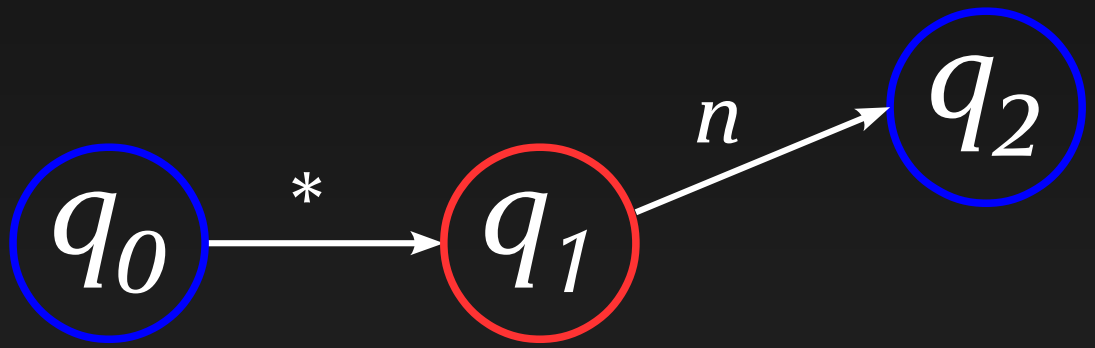


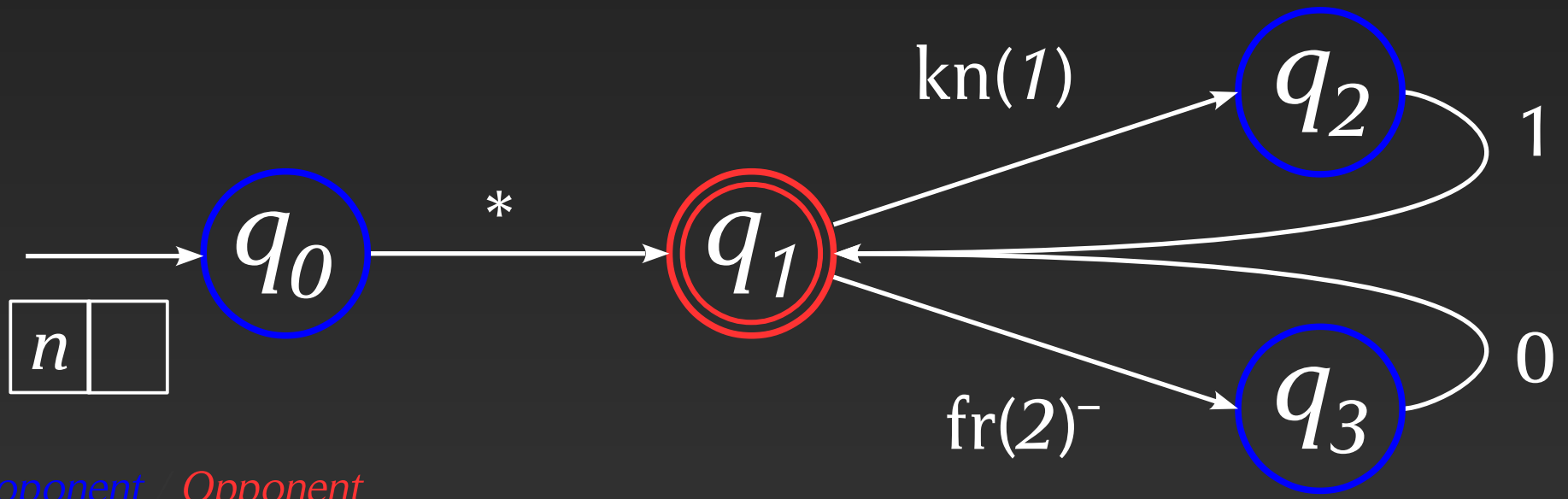
Fresh-register automata



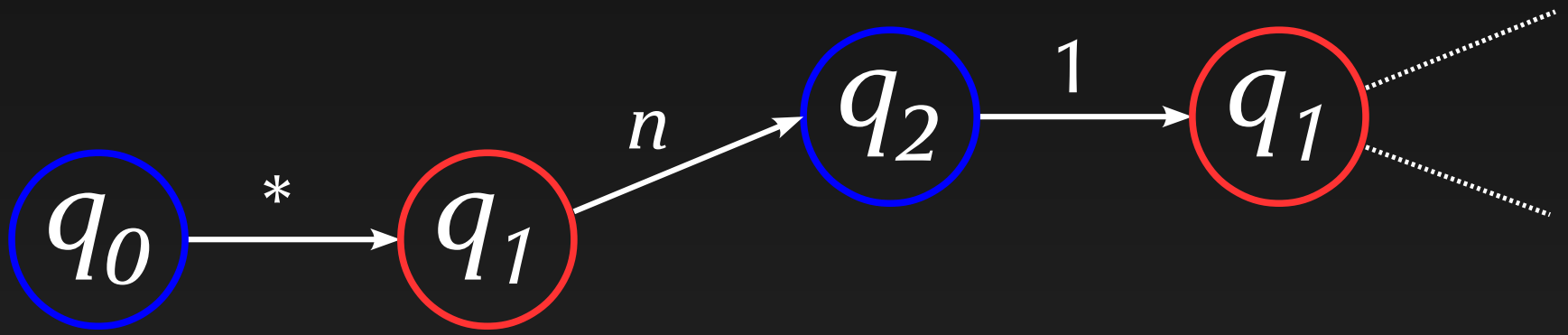


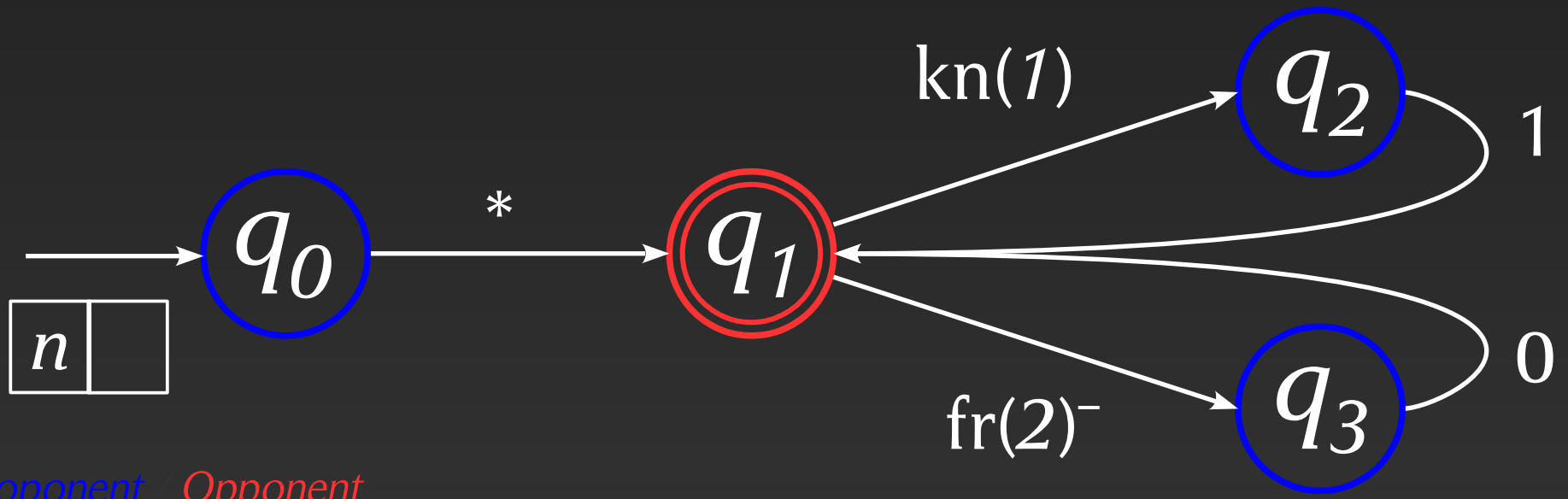
Fresh-register automata





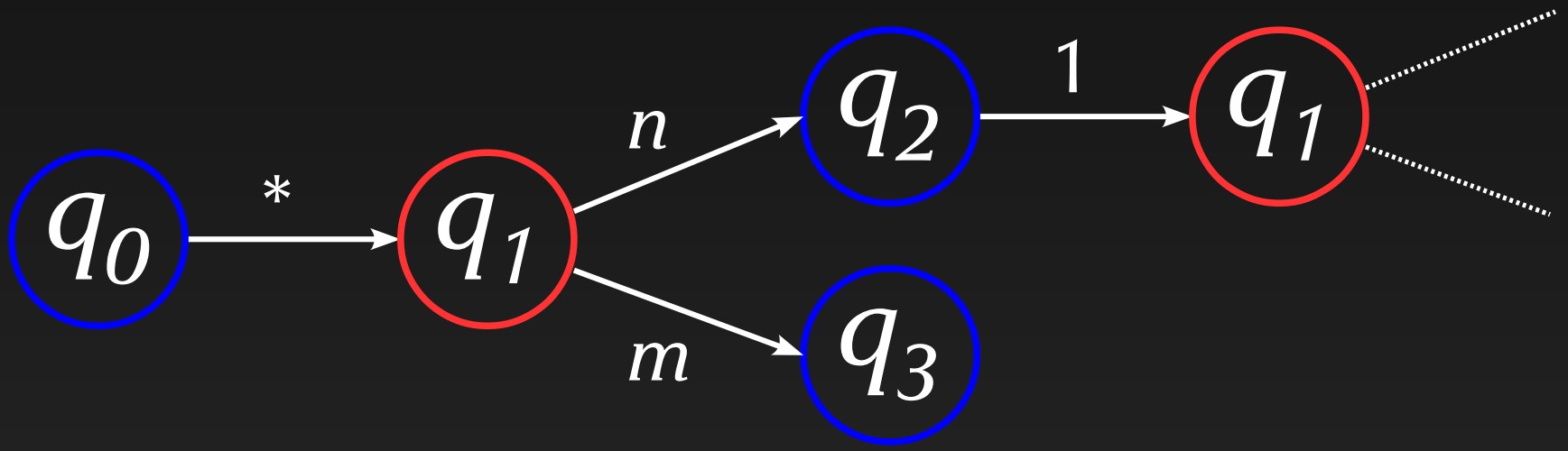
Fresh-register automata

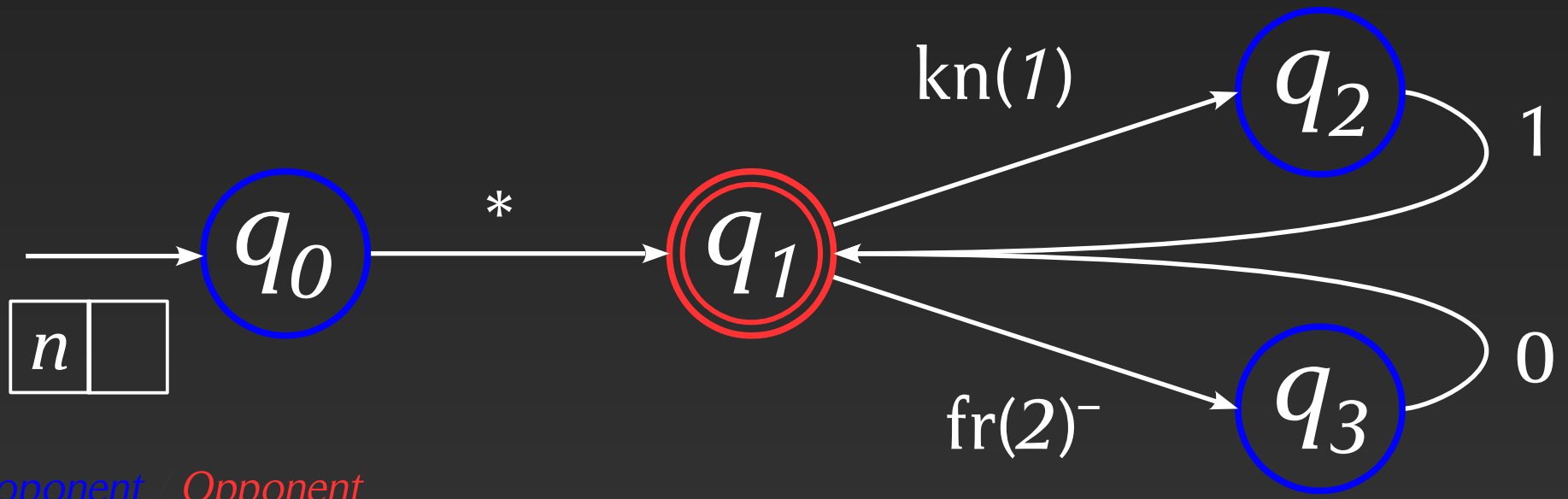




Proponent *Opponent*

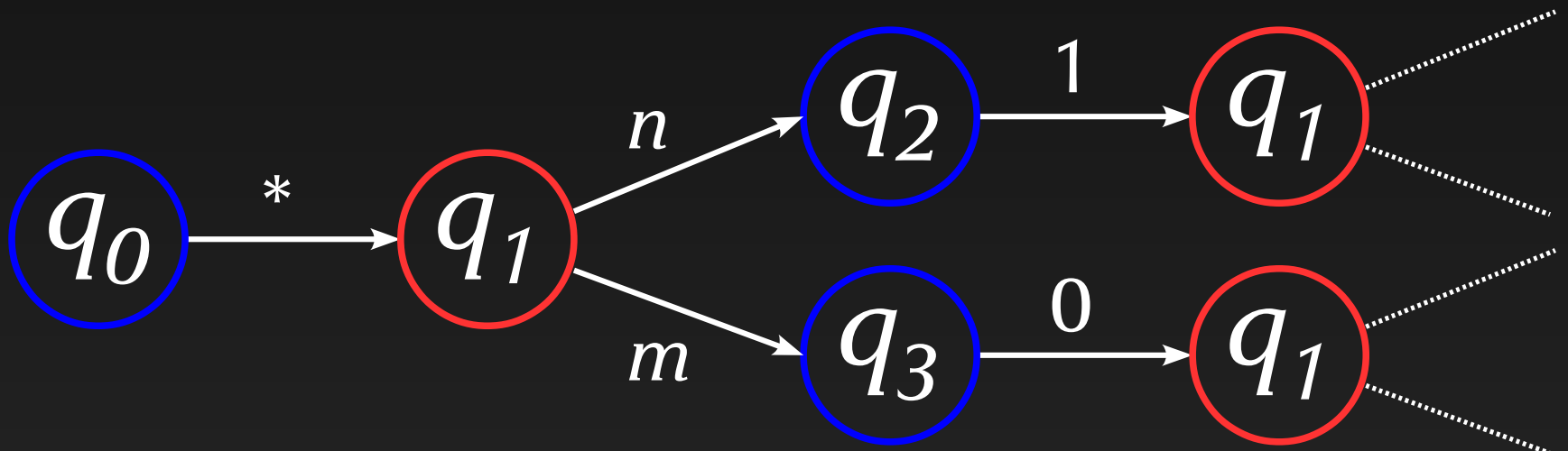
Fresh-register automata

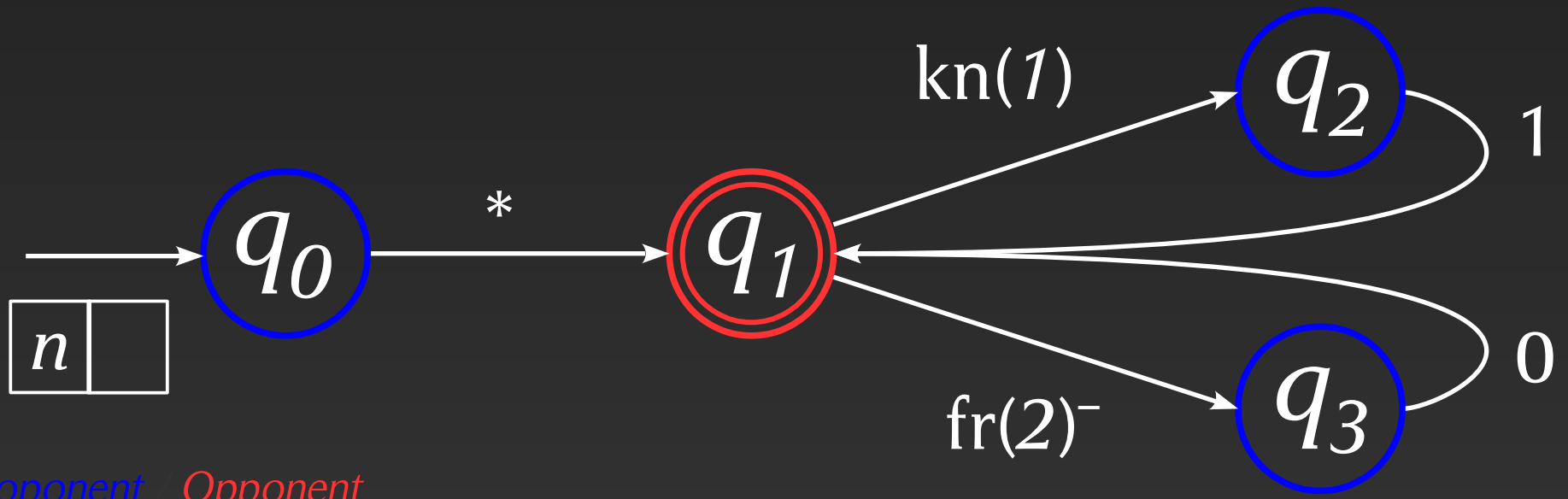




Proponent *Opponent*

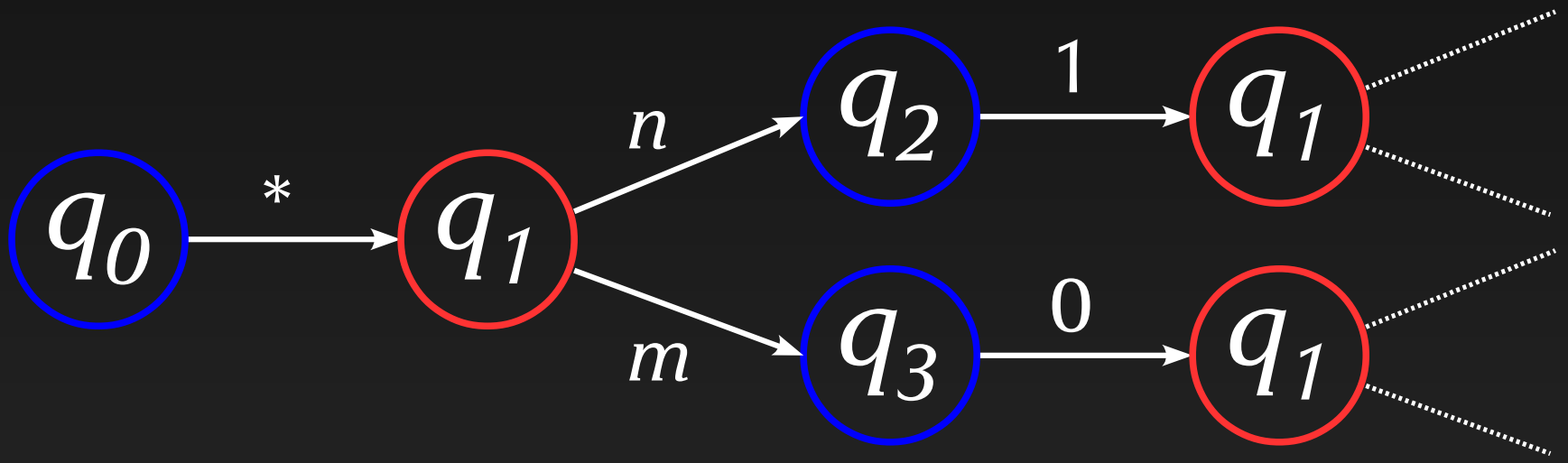
Fresh-register automata



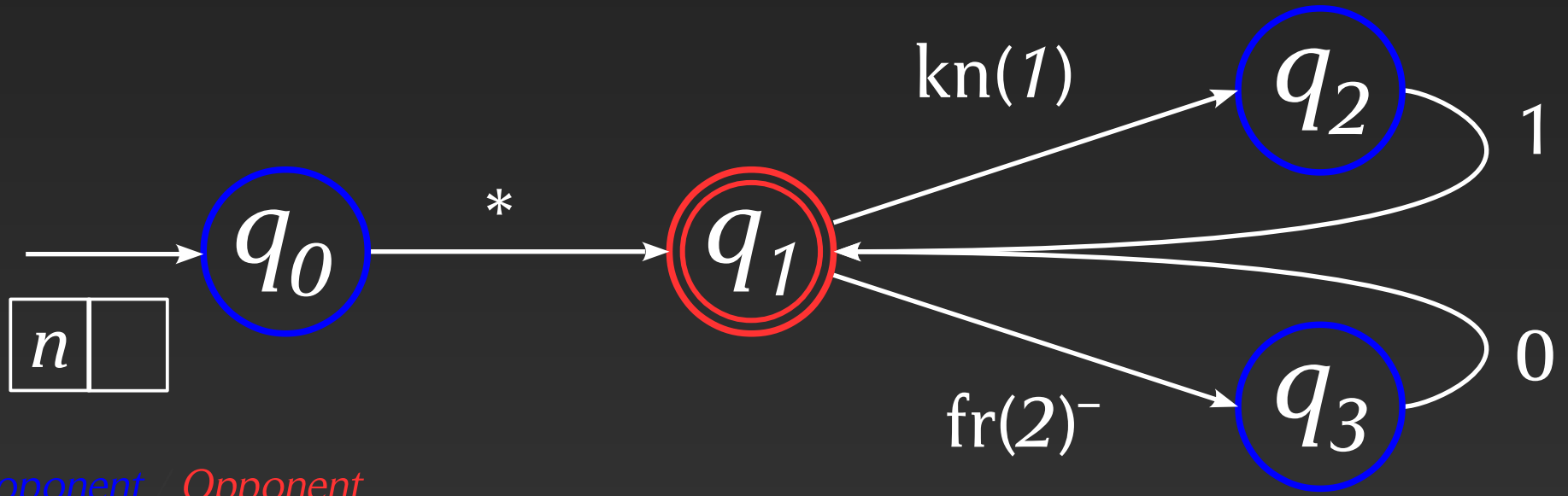


Proponent *Opponent*

Fresh-register automata



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



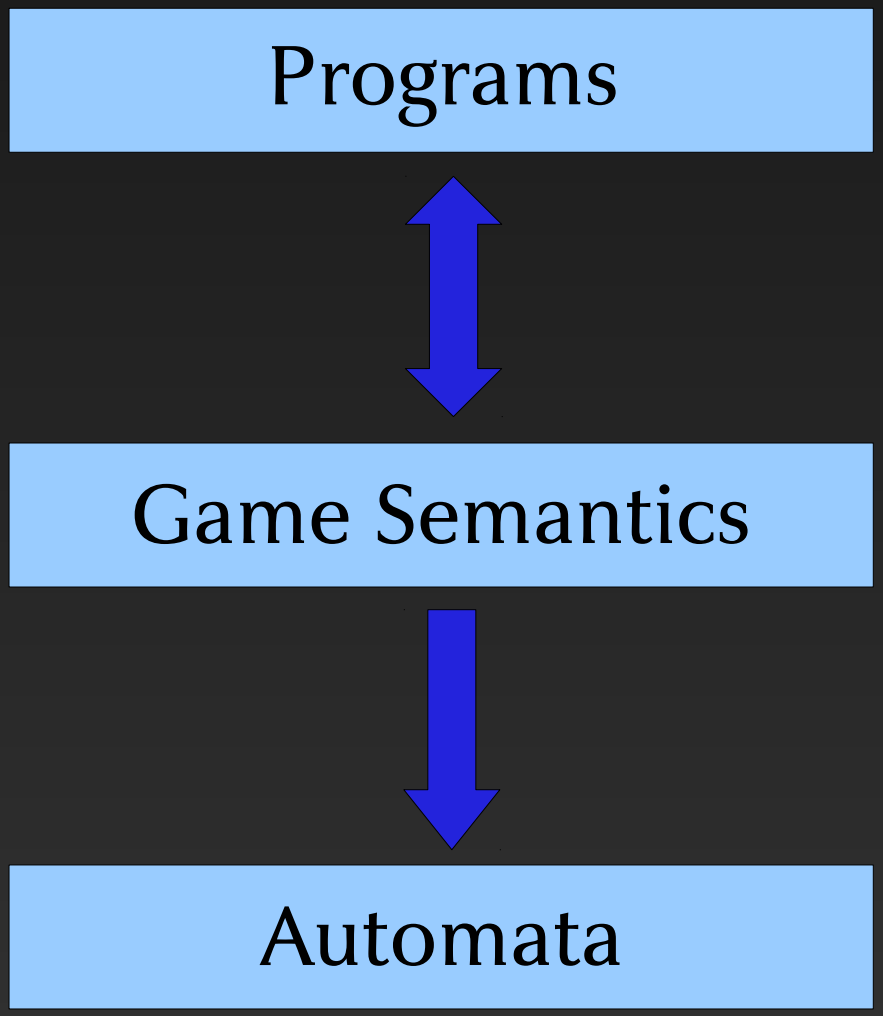
Automata

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

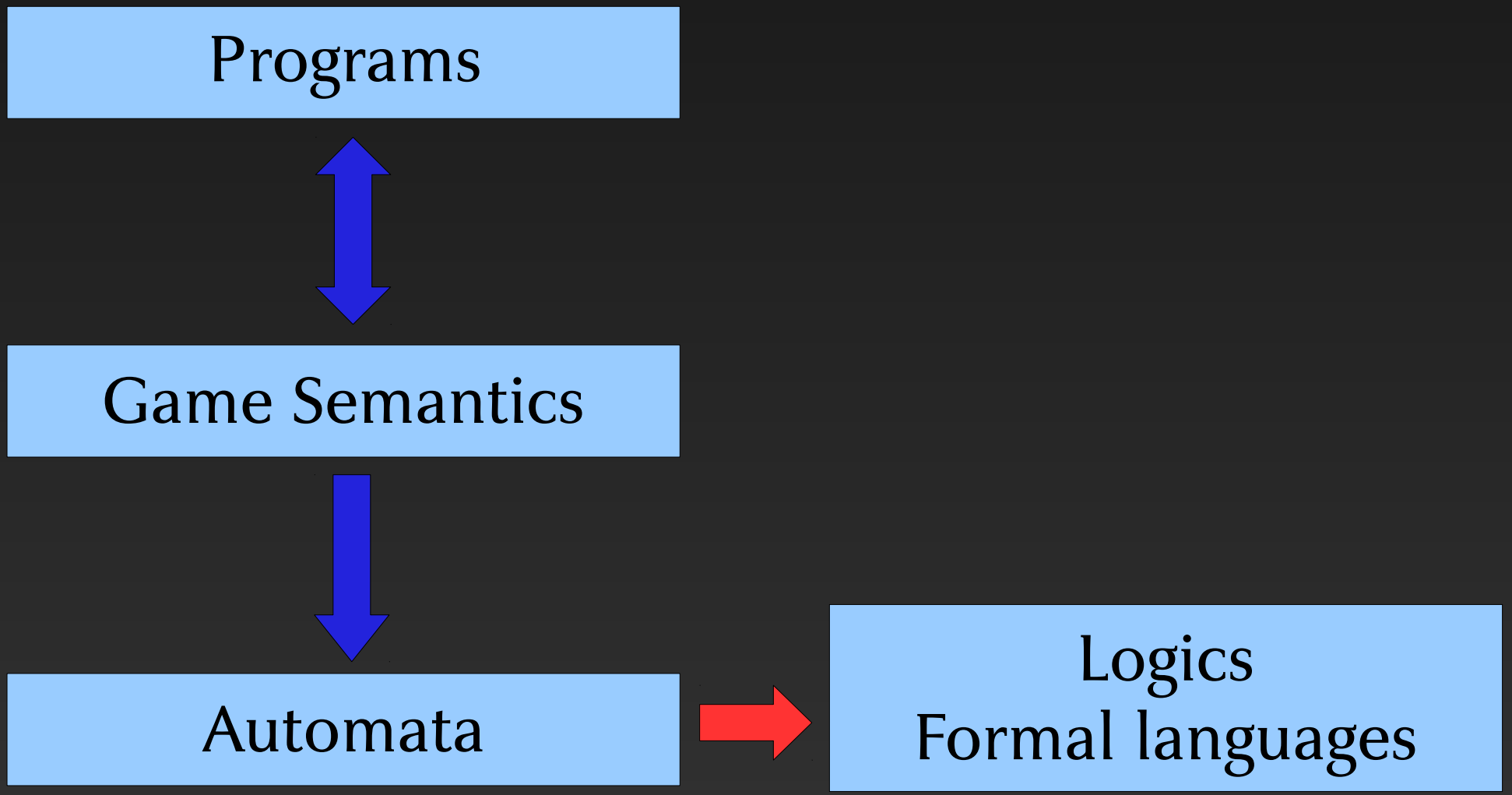
$$M \cong N \iff \mathcal{A}_M \sim \mathcal{A}_N$$

- Ghi.Tz. 11: System-level games

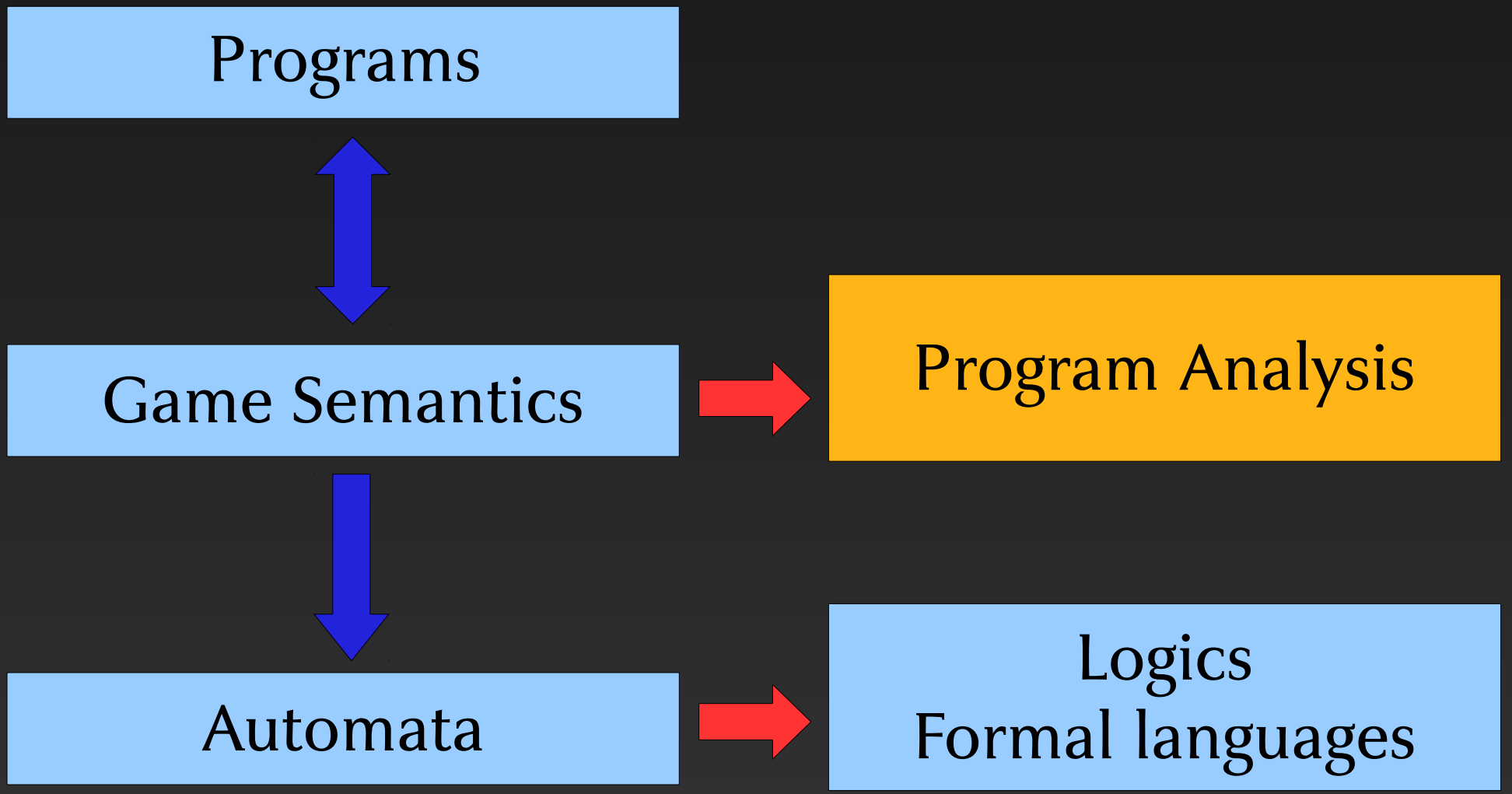
Reasoning about new resources



Reasoning about new resources



Reasoning about new resources



Reasoning about new resources

thank you!