Games with names

Nikos Tzevelekos

Oxford University Computing Laboratory

What this talk is about

Generation of new resources (*names*)
is a pervasive feature in computation

We present games and related formalisms
for the semantics of new resources

Ingredients

Game Semantics

Nominal Sets

Fresh-Register Automata

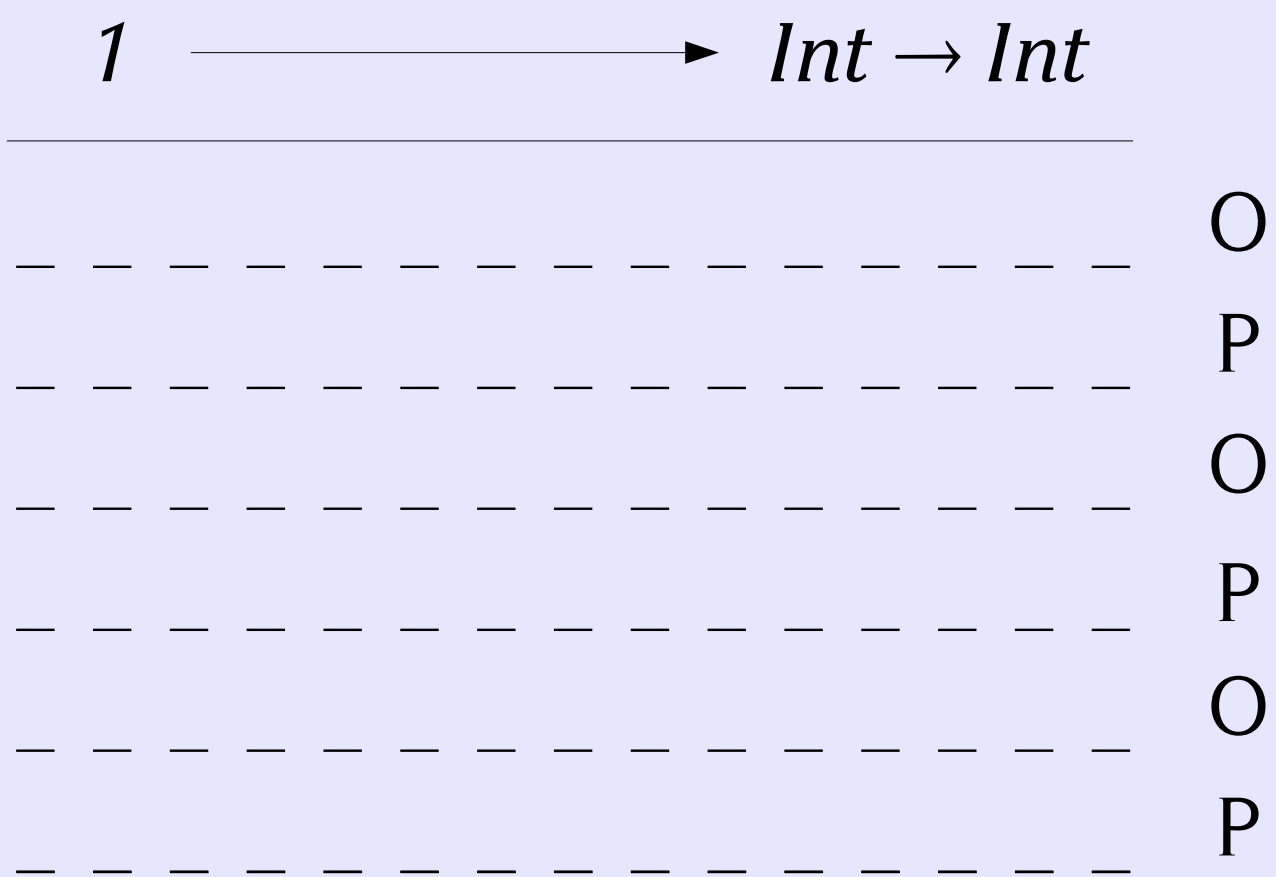
Game Semantics

Game Semantics

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

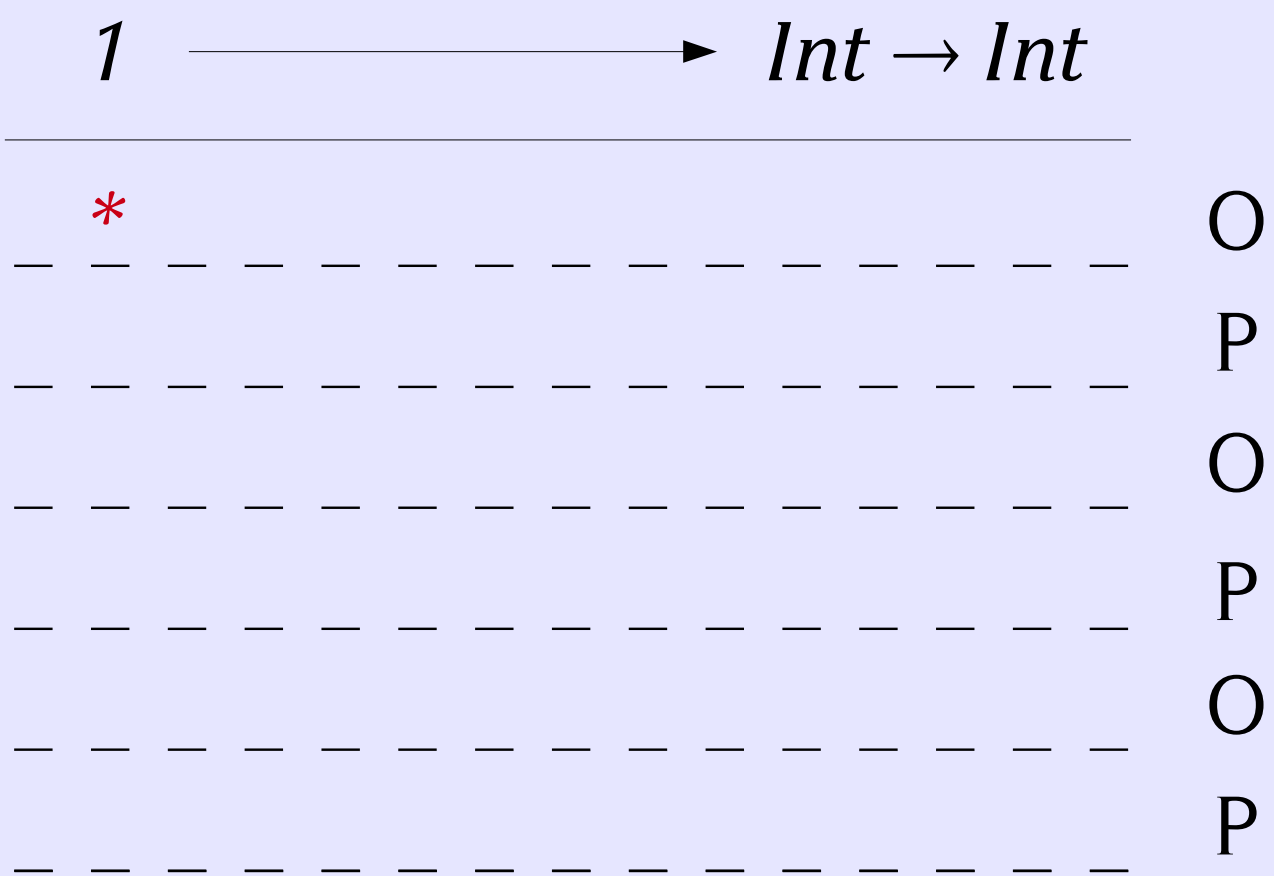
Example

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



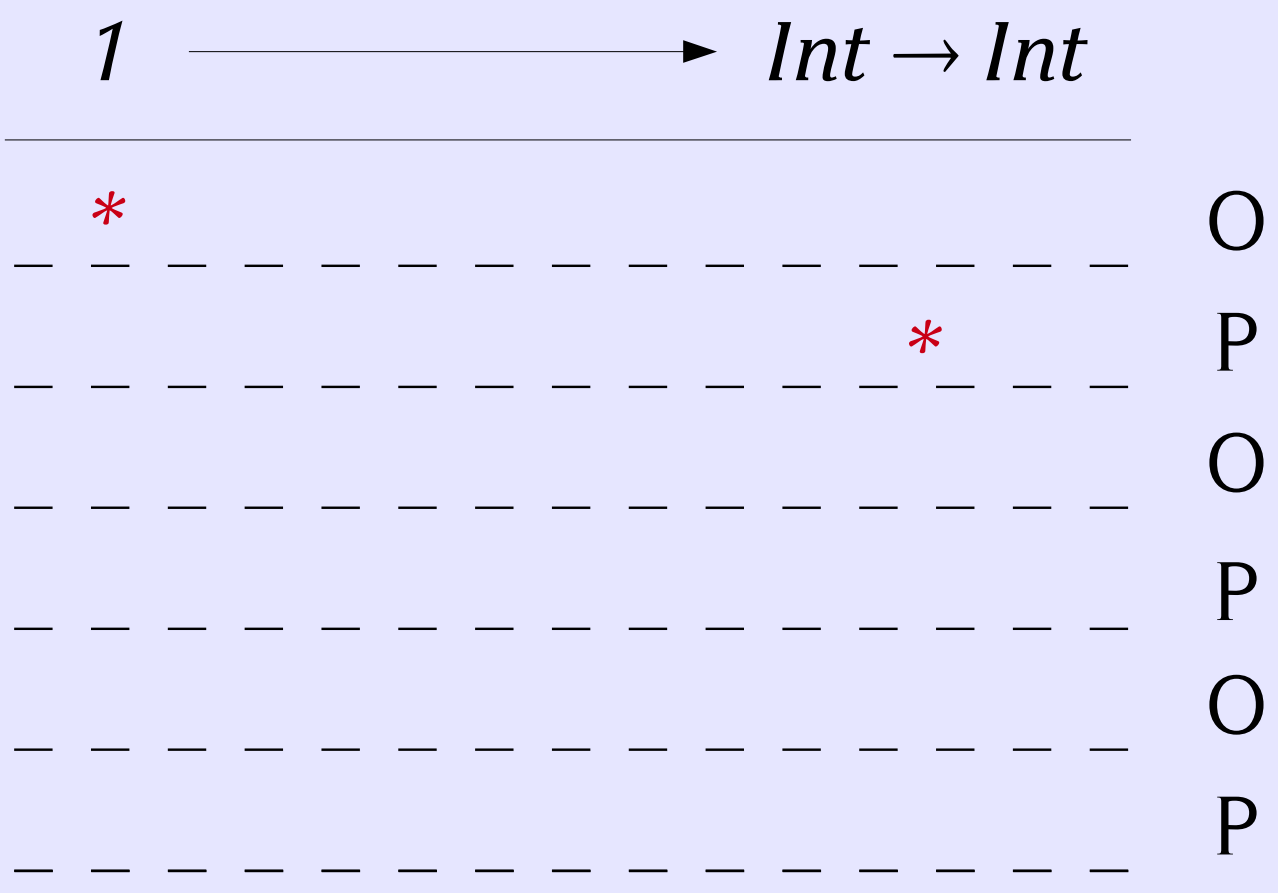
Example

$\vdash$ `λx.x+1 : int → int`



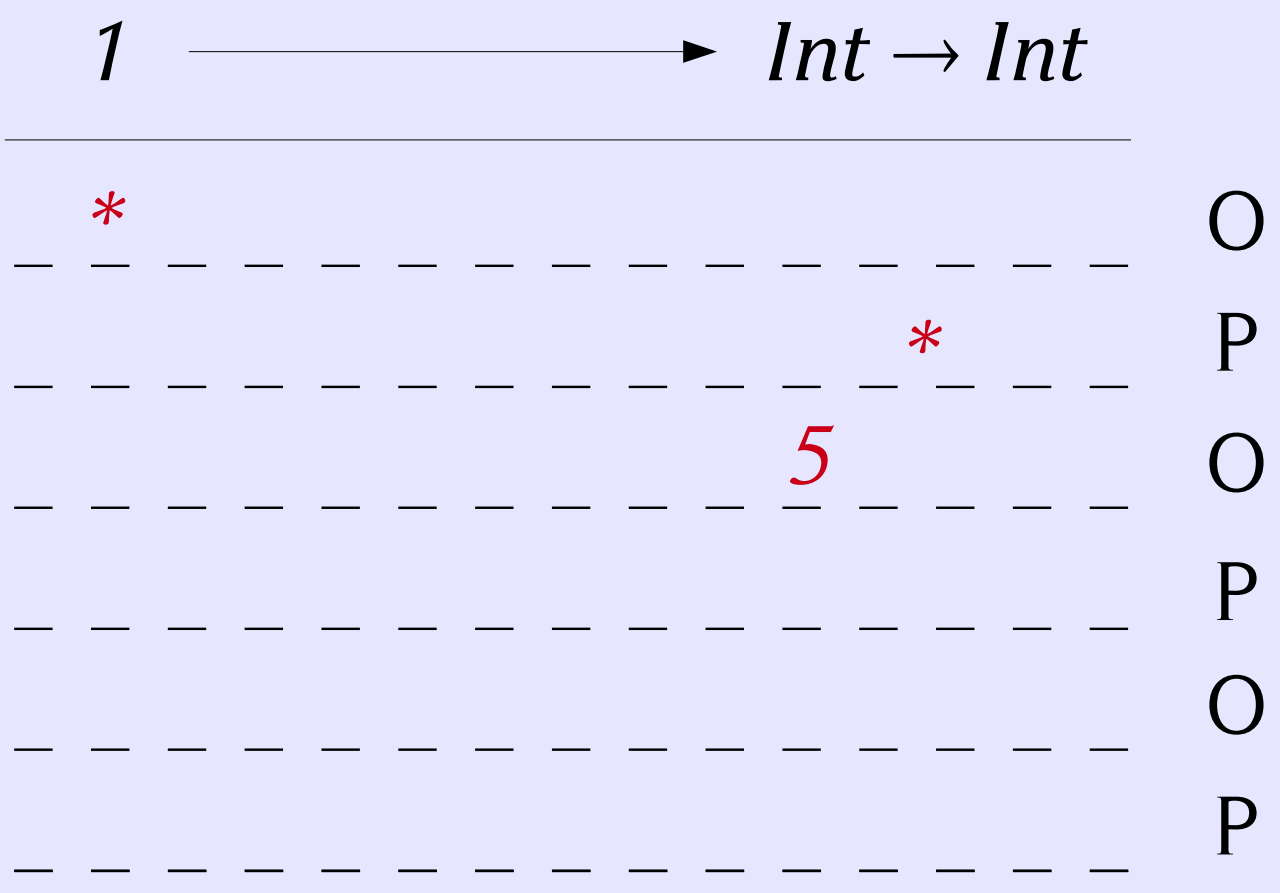
Example

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



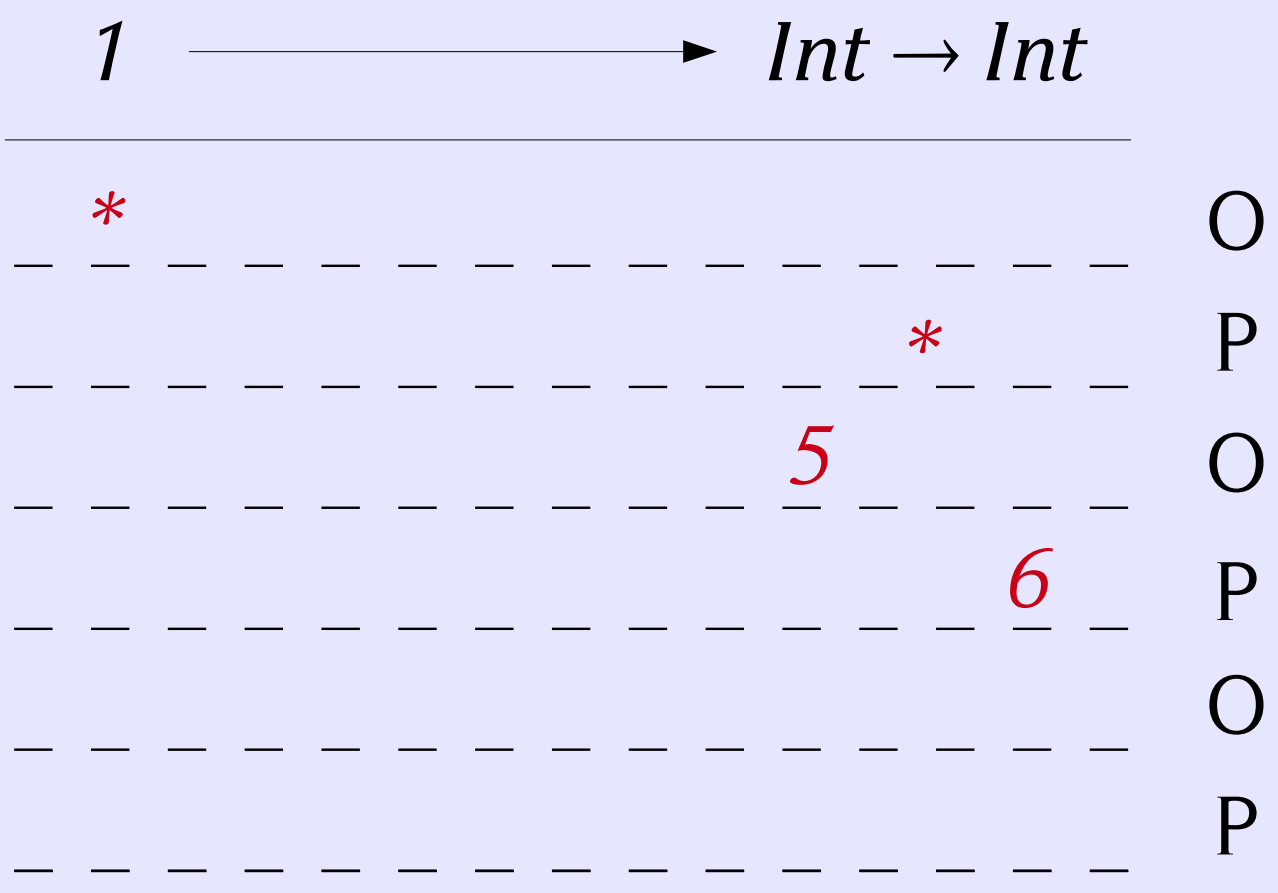
Example

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



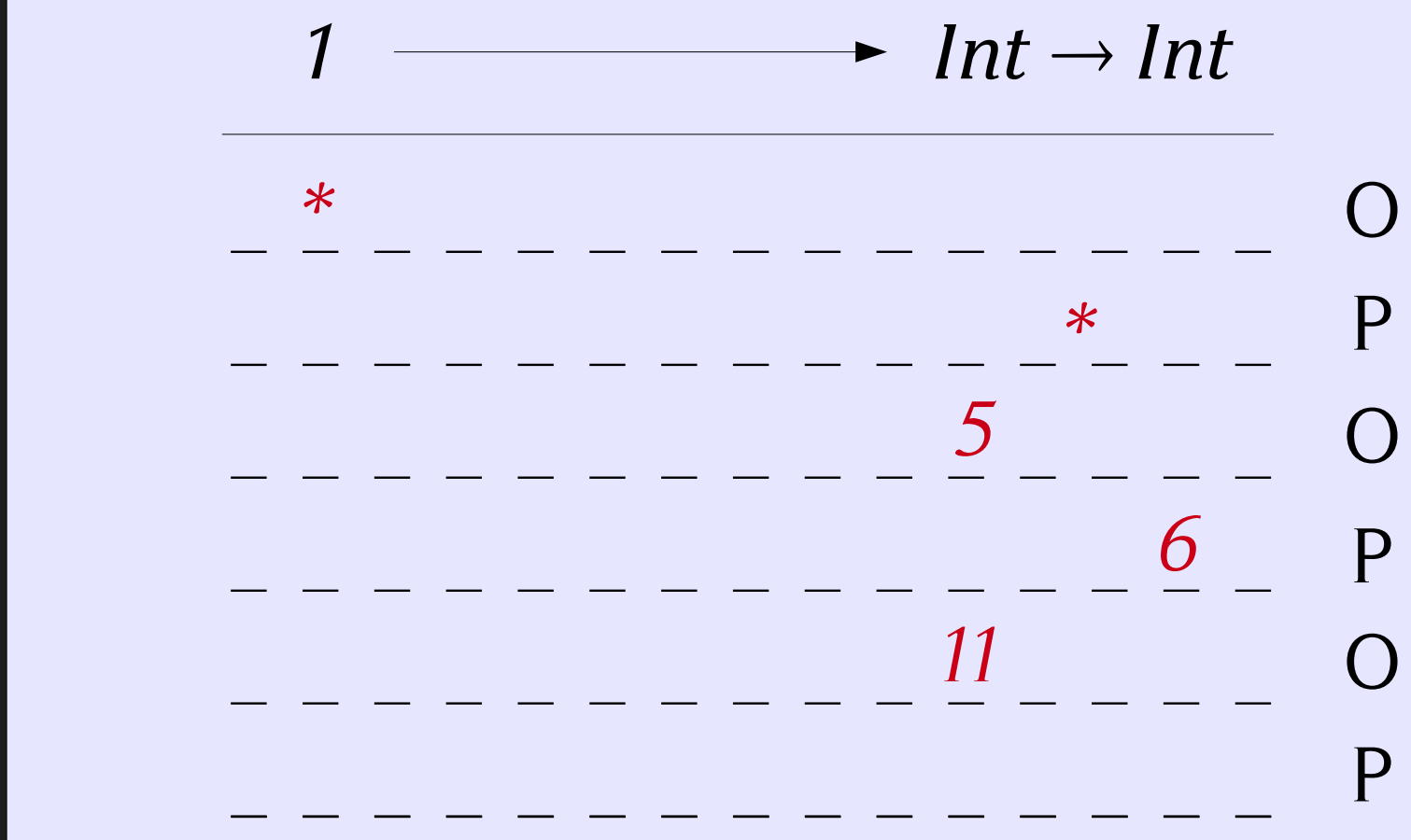
Example

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



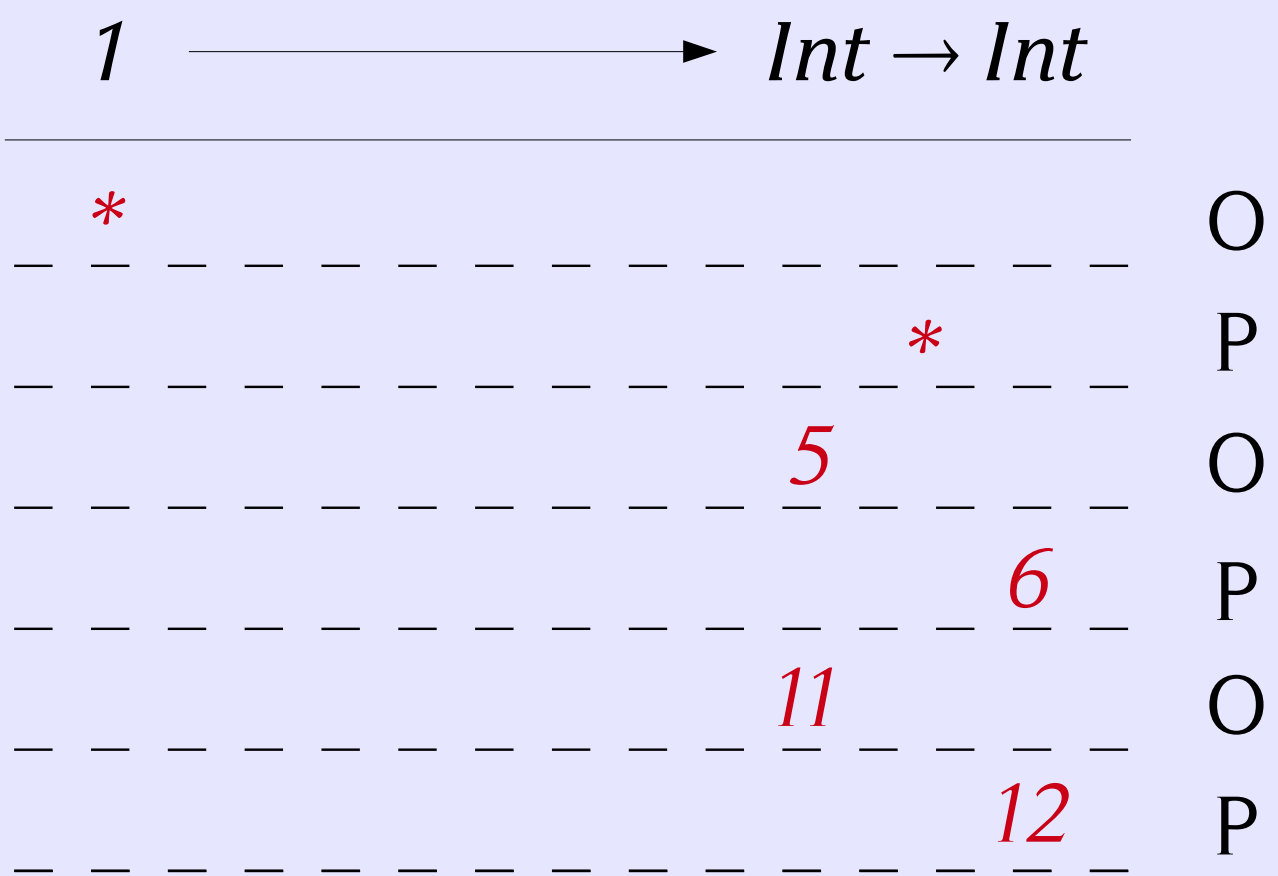
Example

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



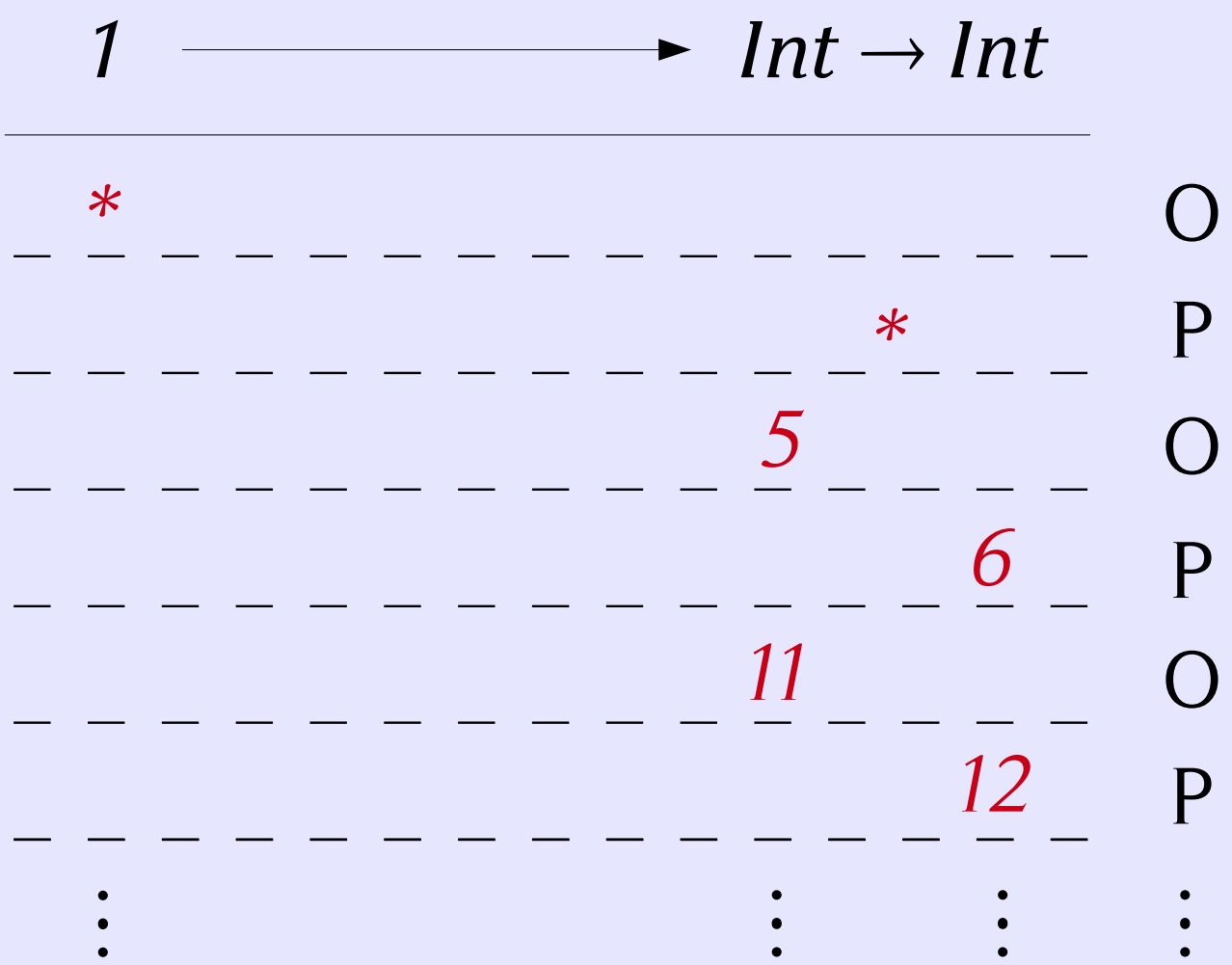
Example

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



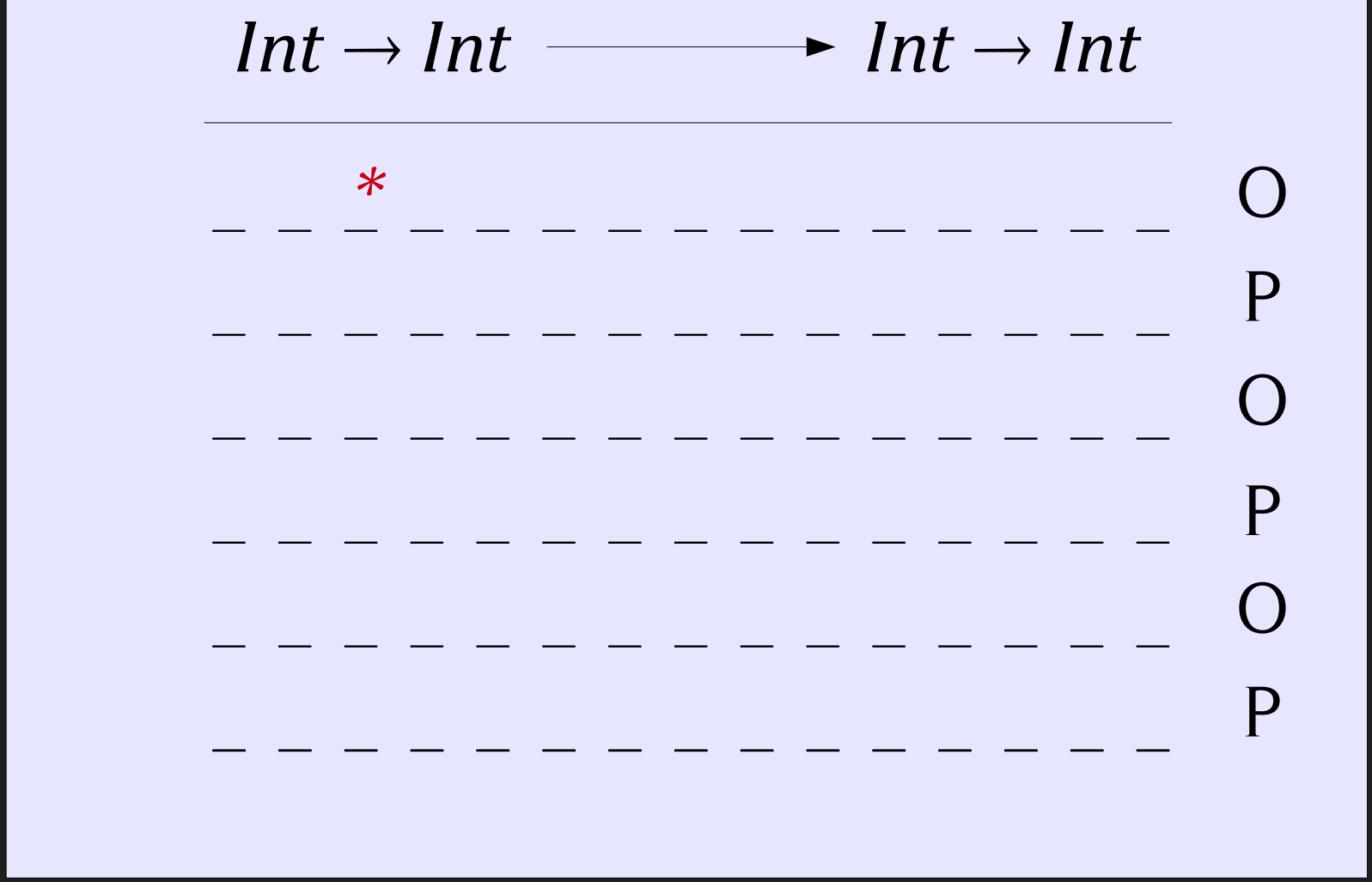
Example

$\vdash$ `λx.x+1 : int → int`



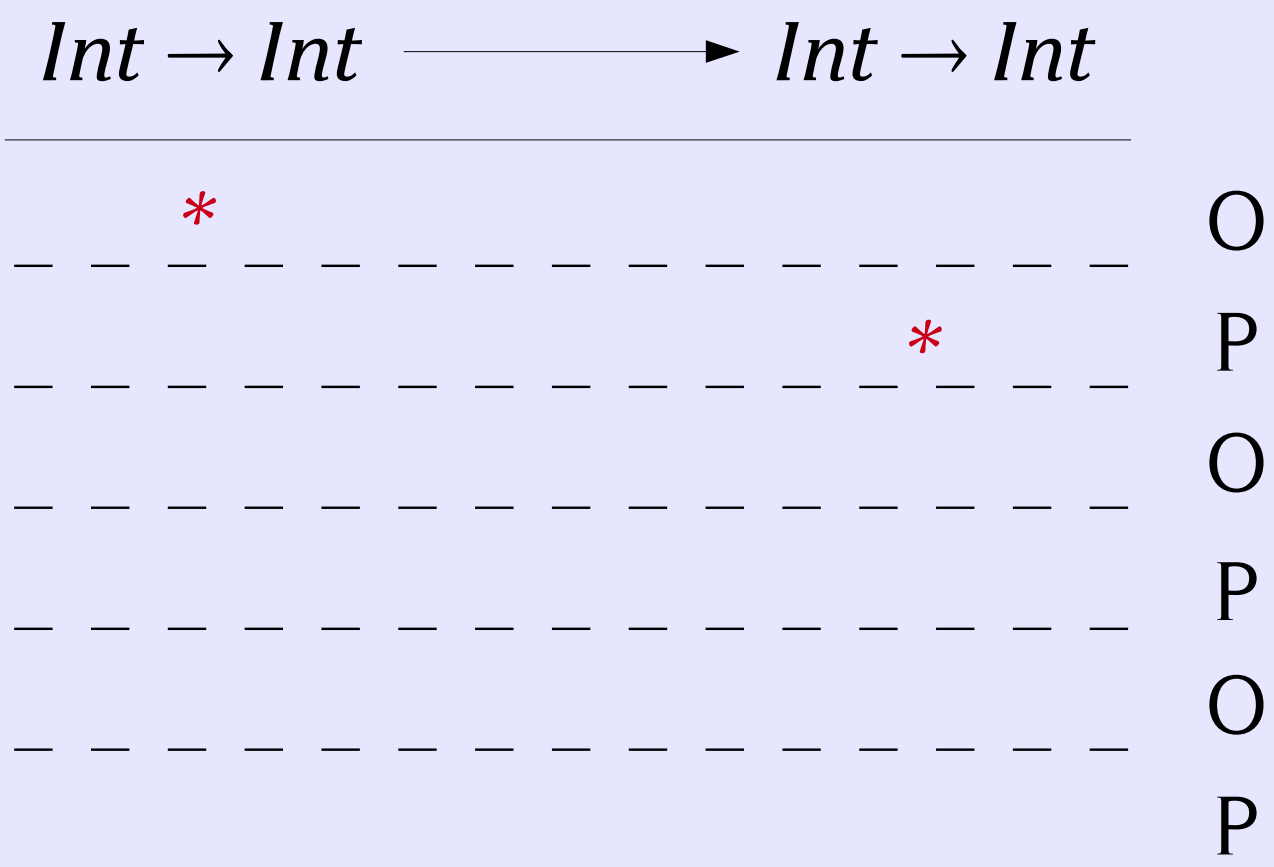
Example

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



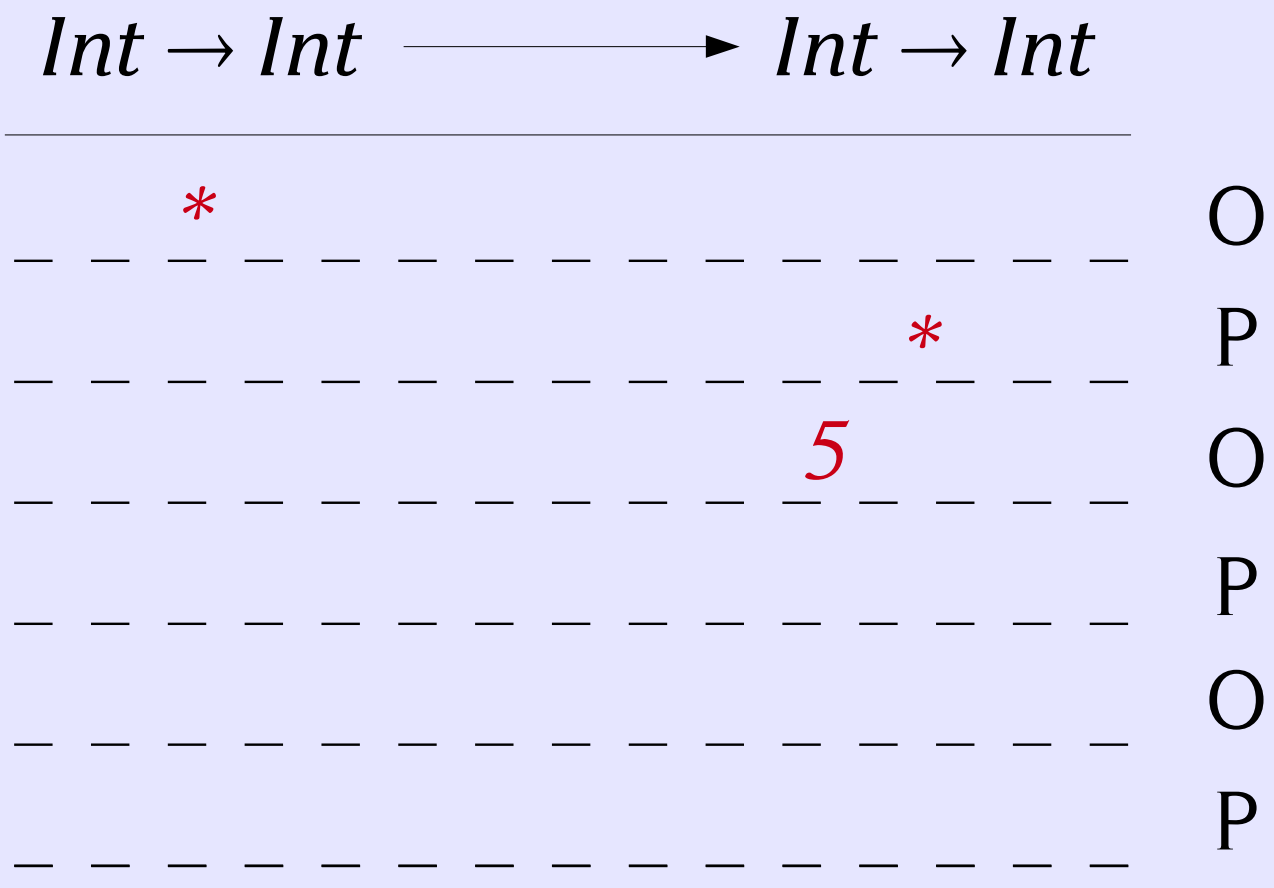
Example

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



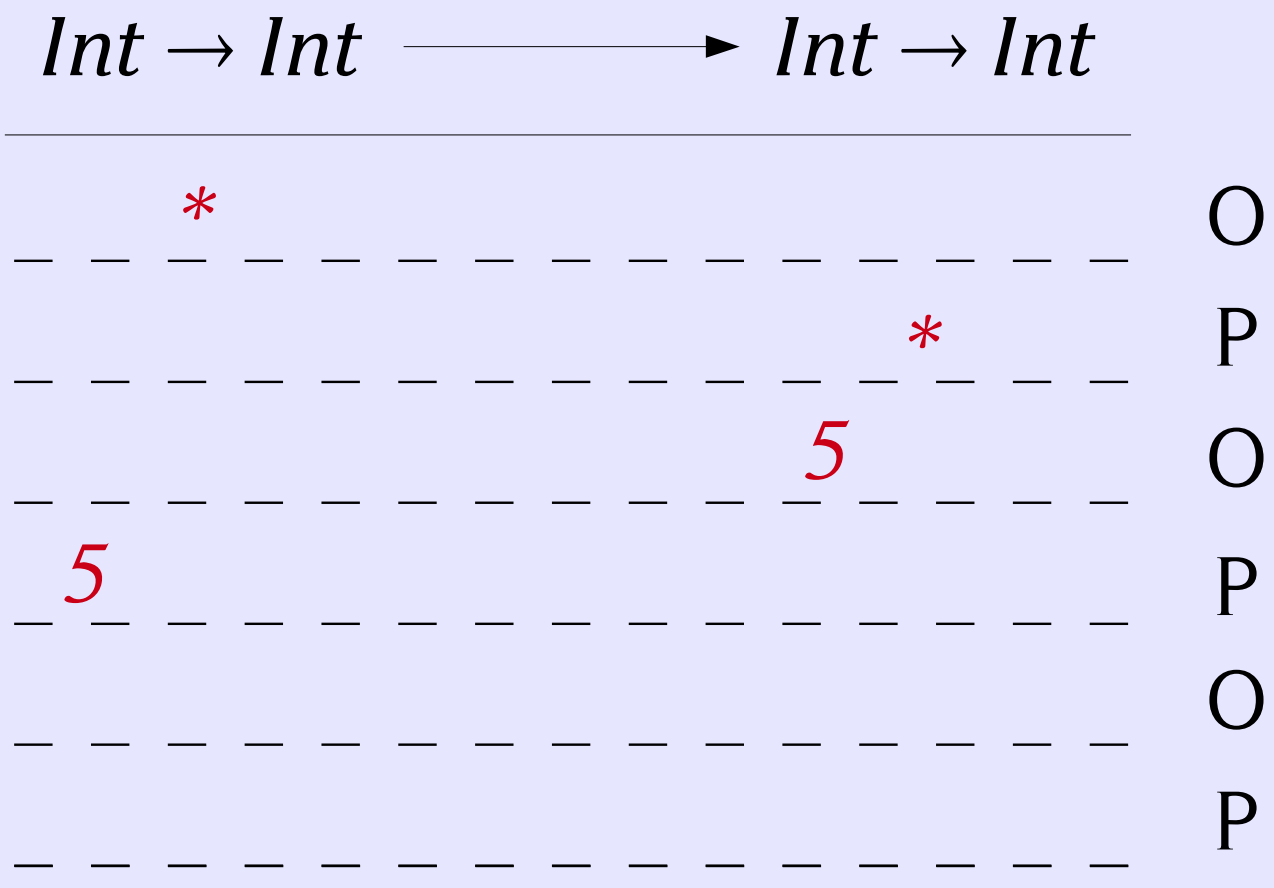
Example

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



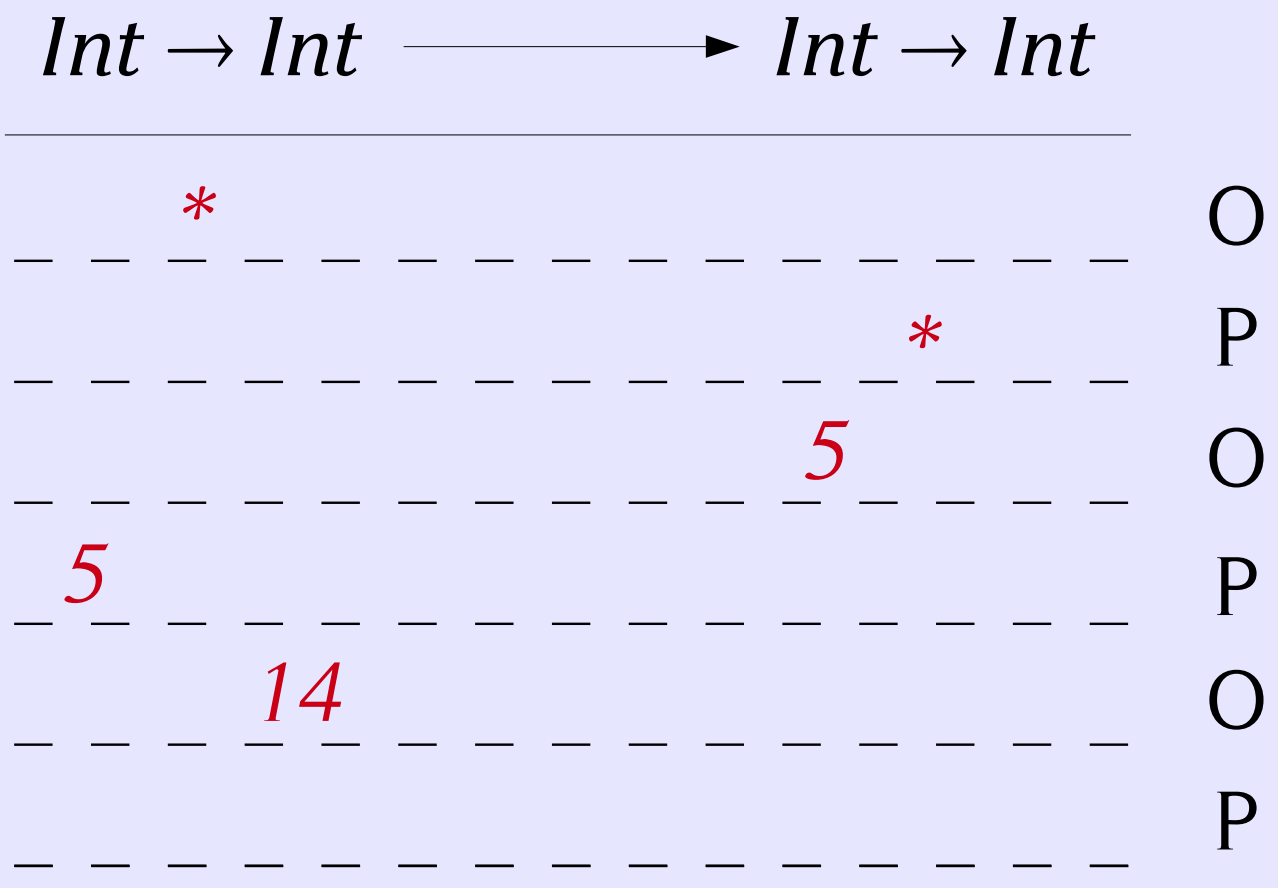
Example

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



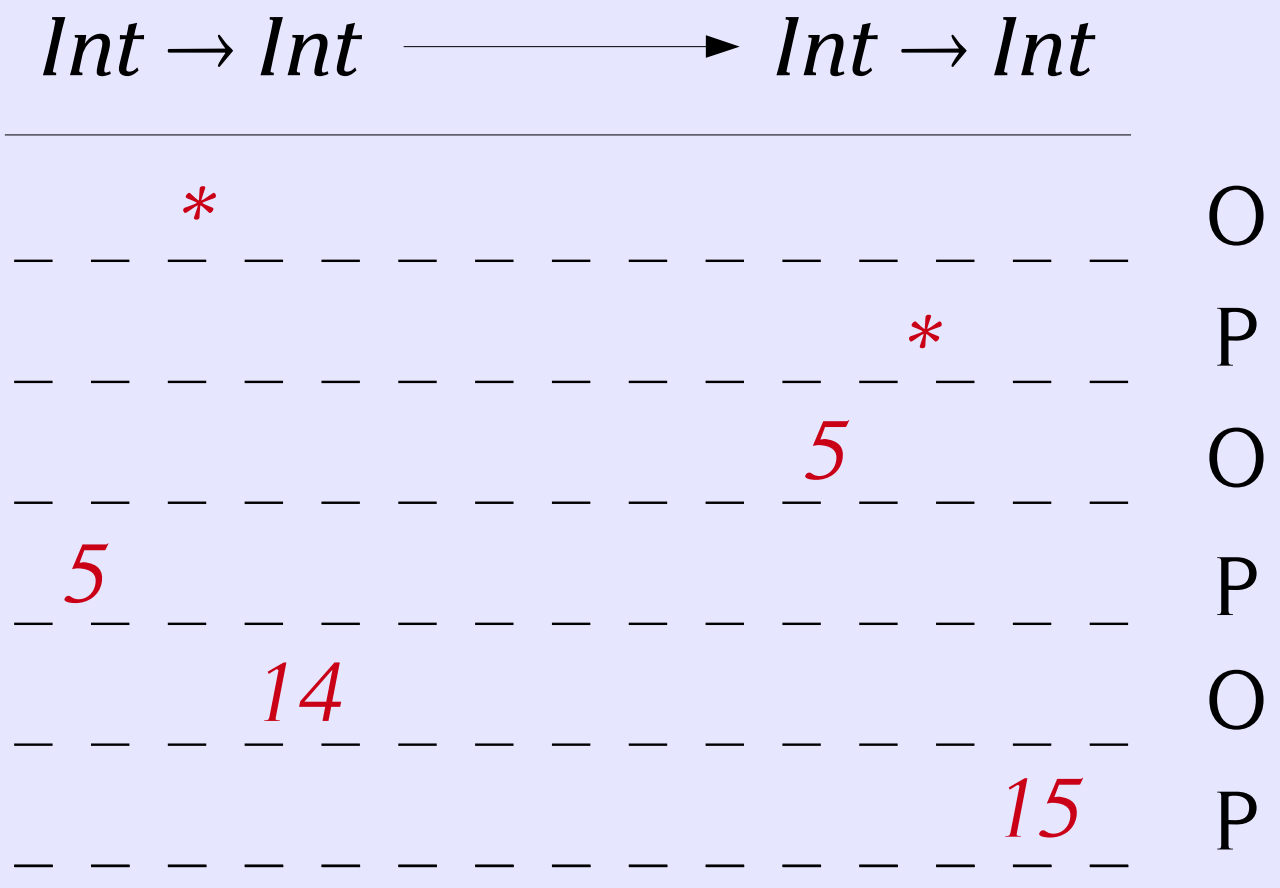
Example

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



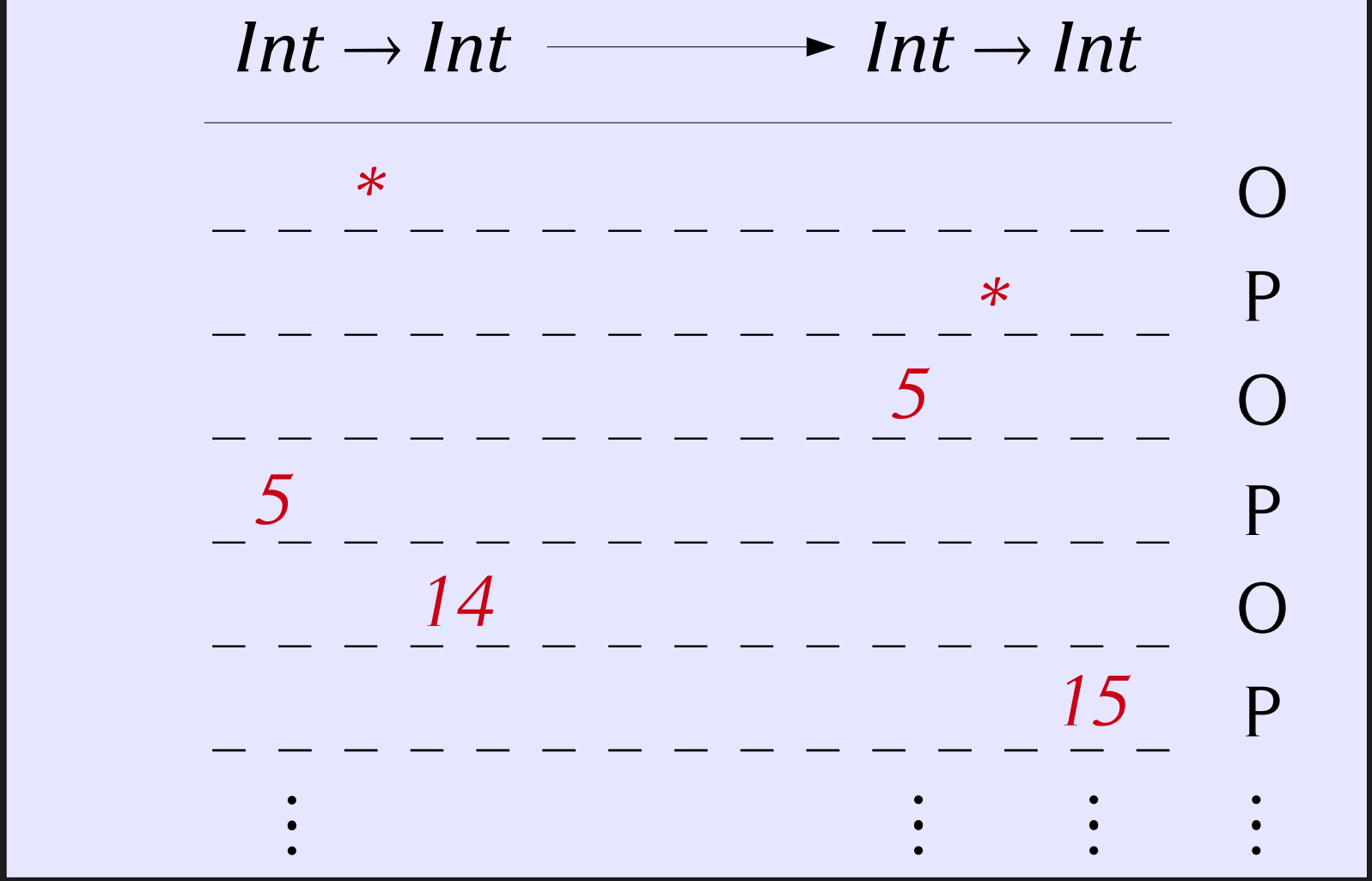
Example

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

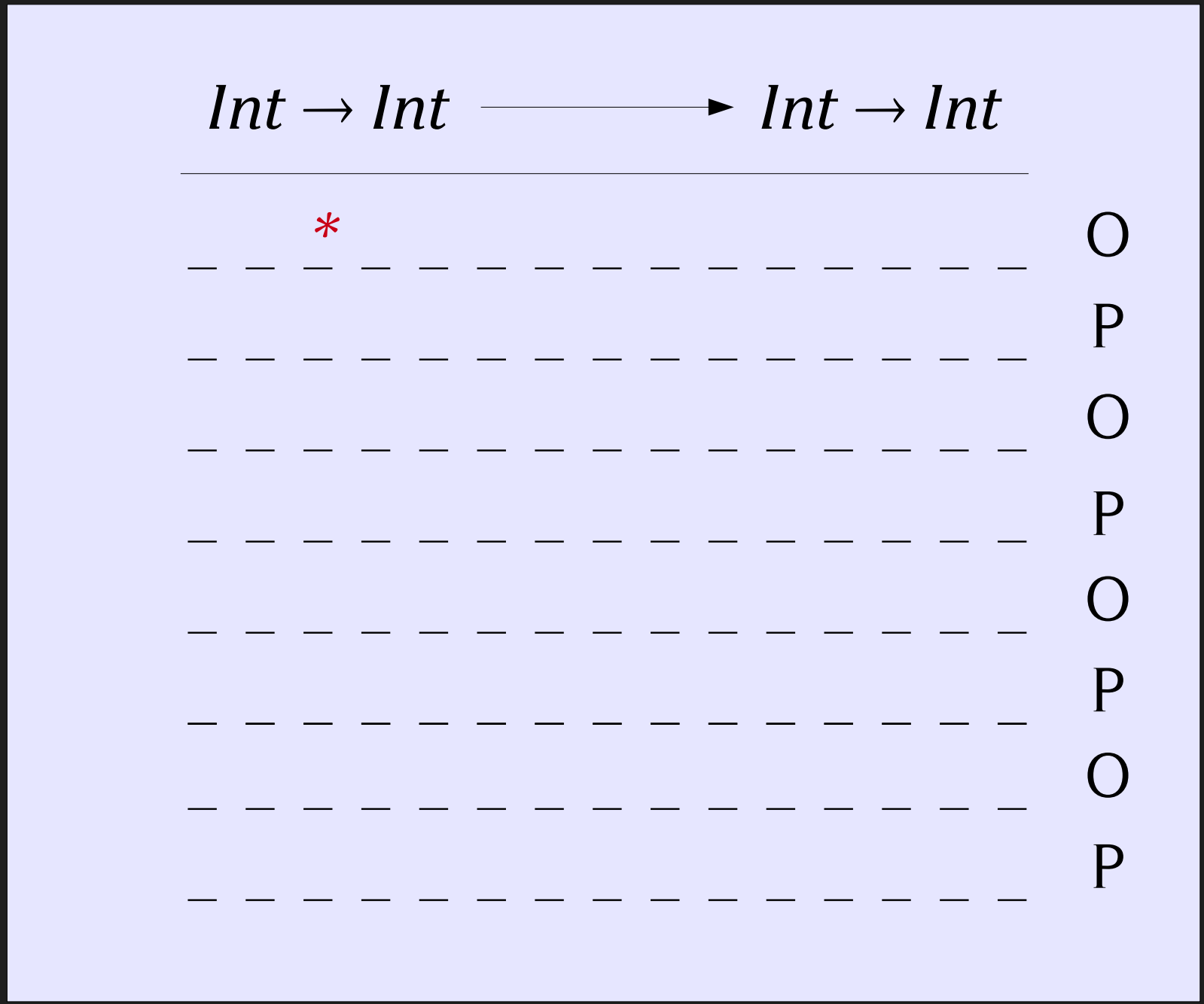


Example

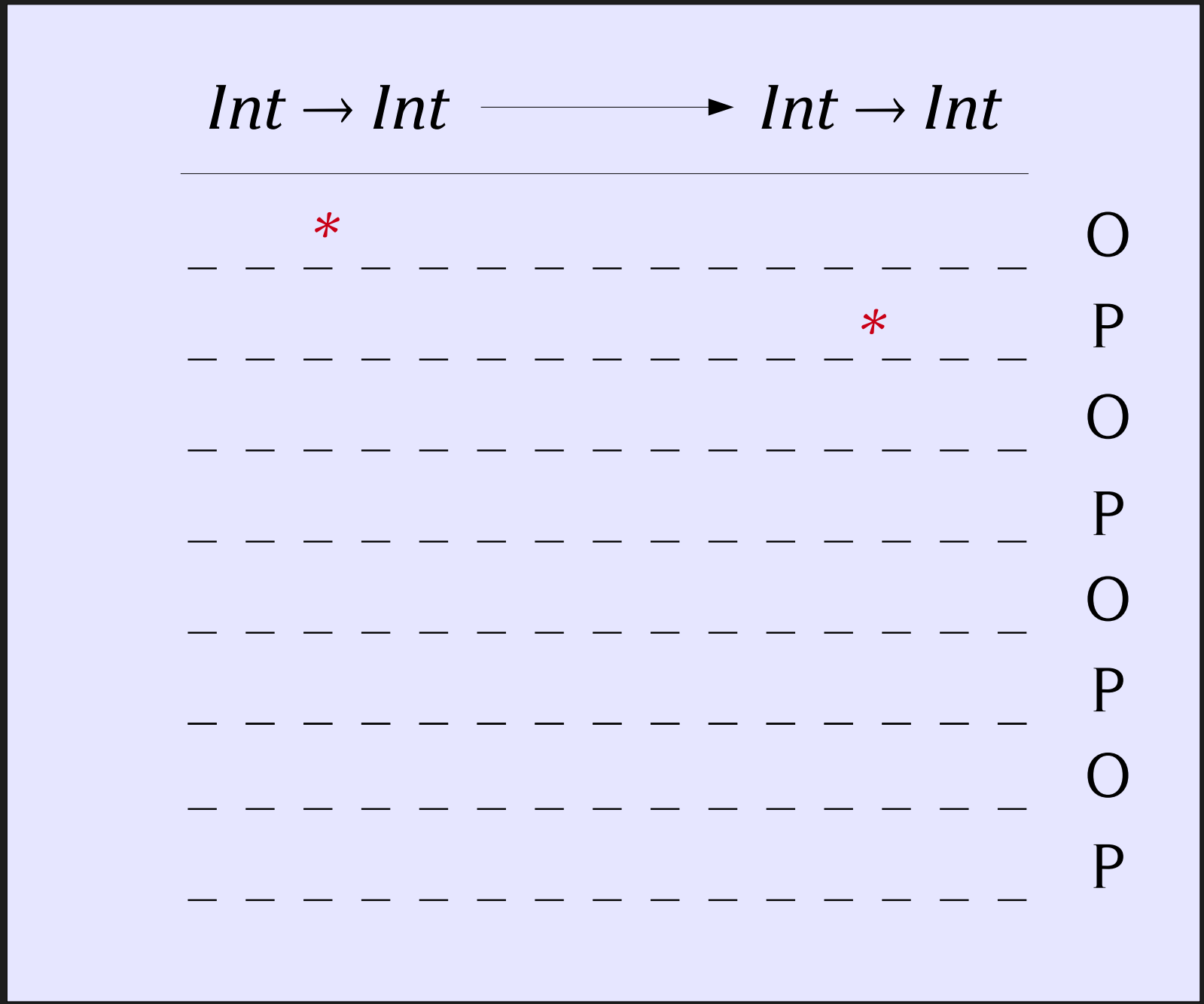
$\texttt{f : int} \to \texttt{int} \vdash \texttt{λx.f(x)+1 : int} \to \texttt{int}$



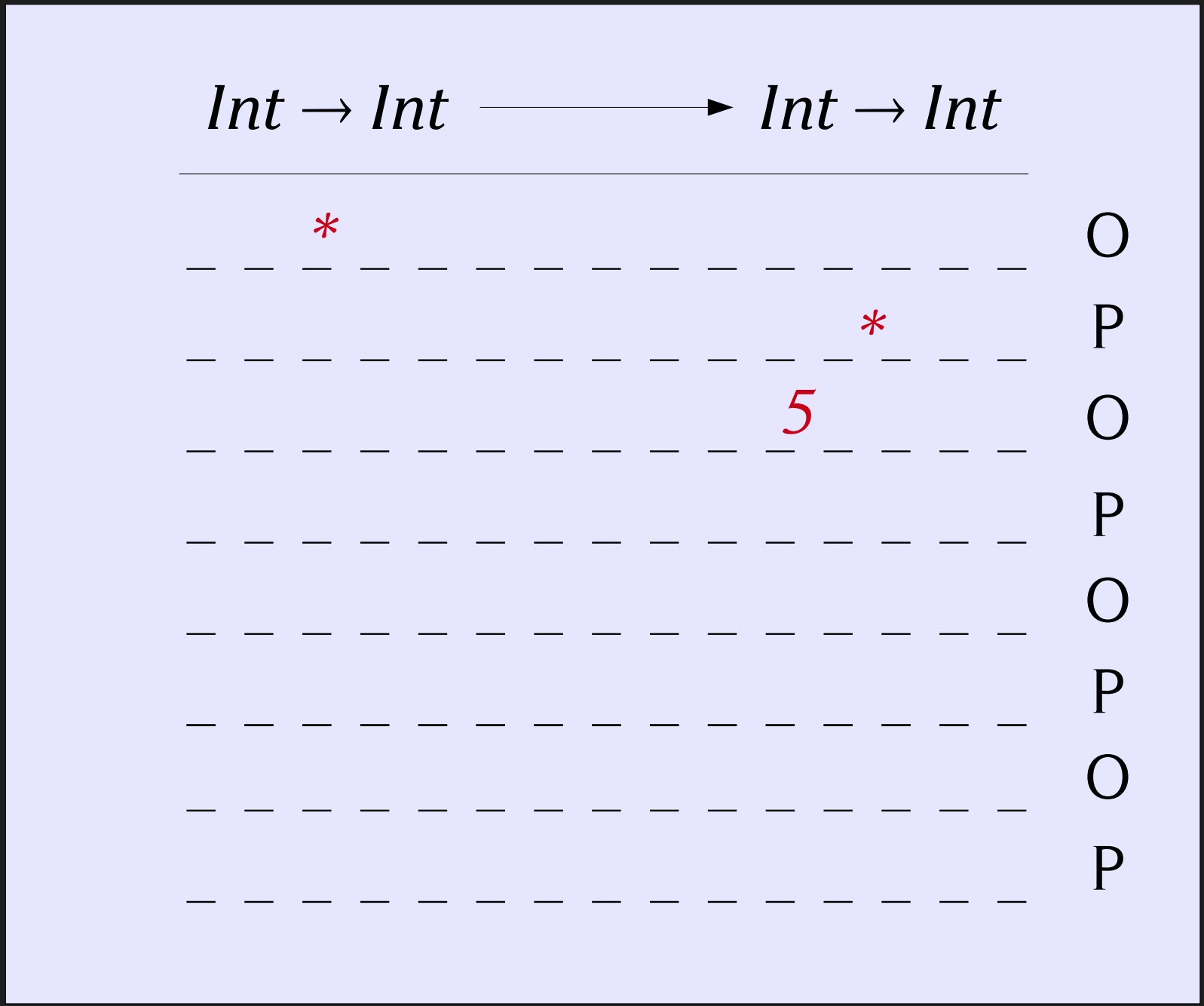
Example



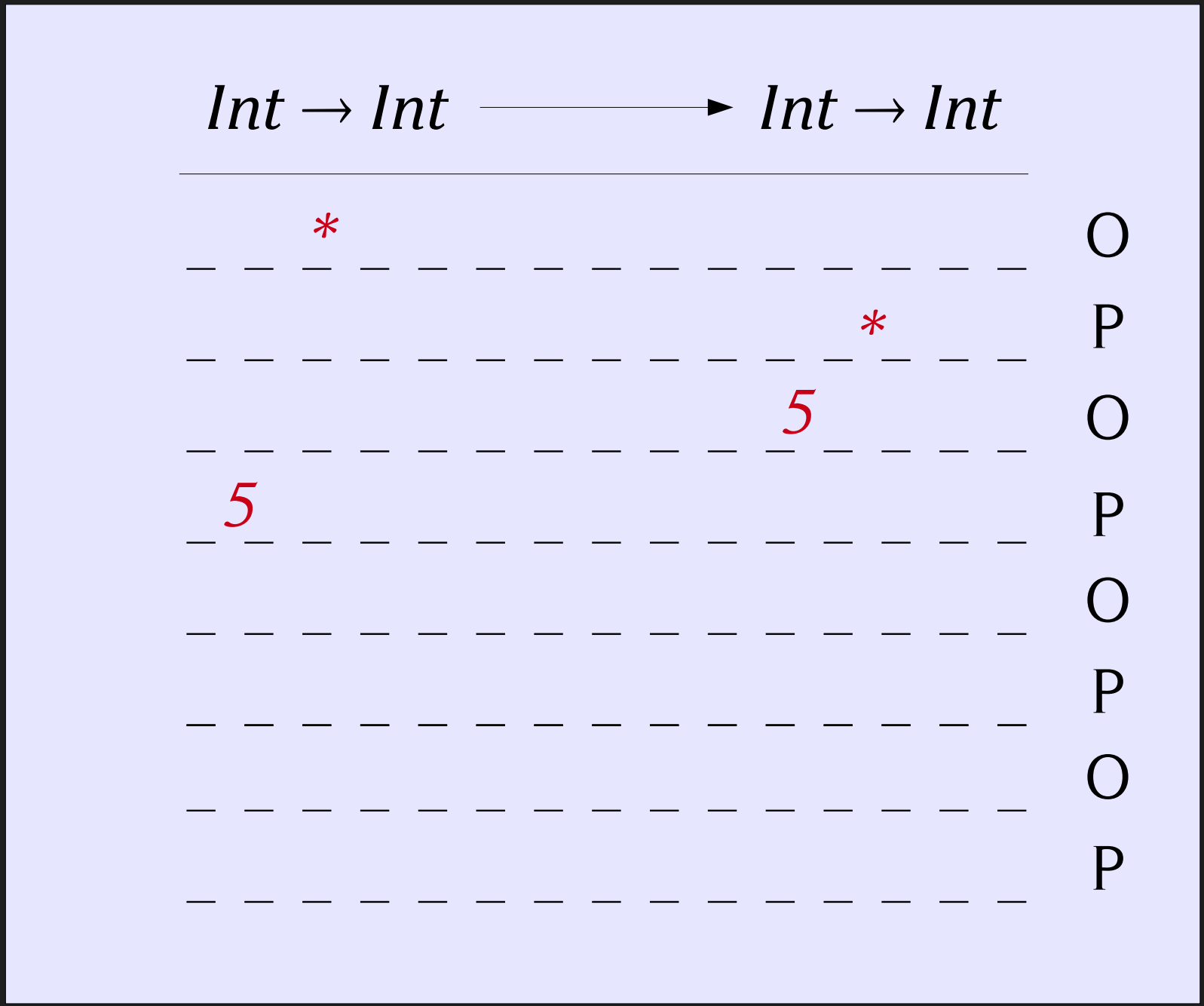
Example



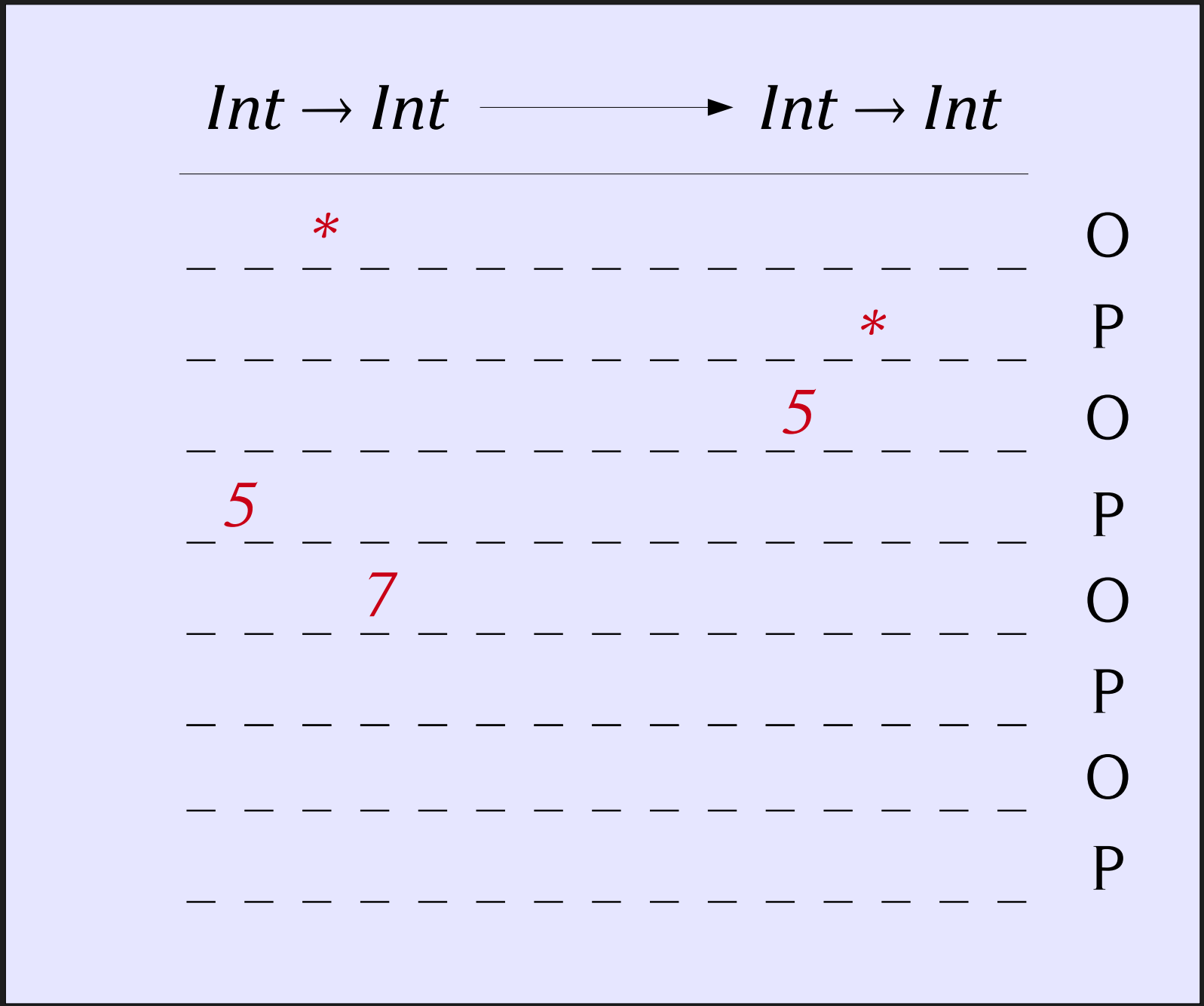
Example



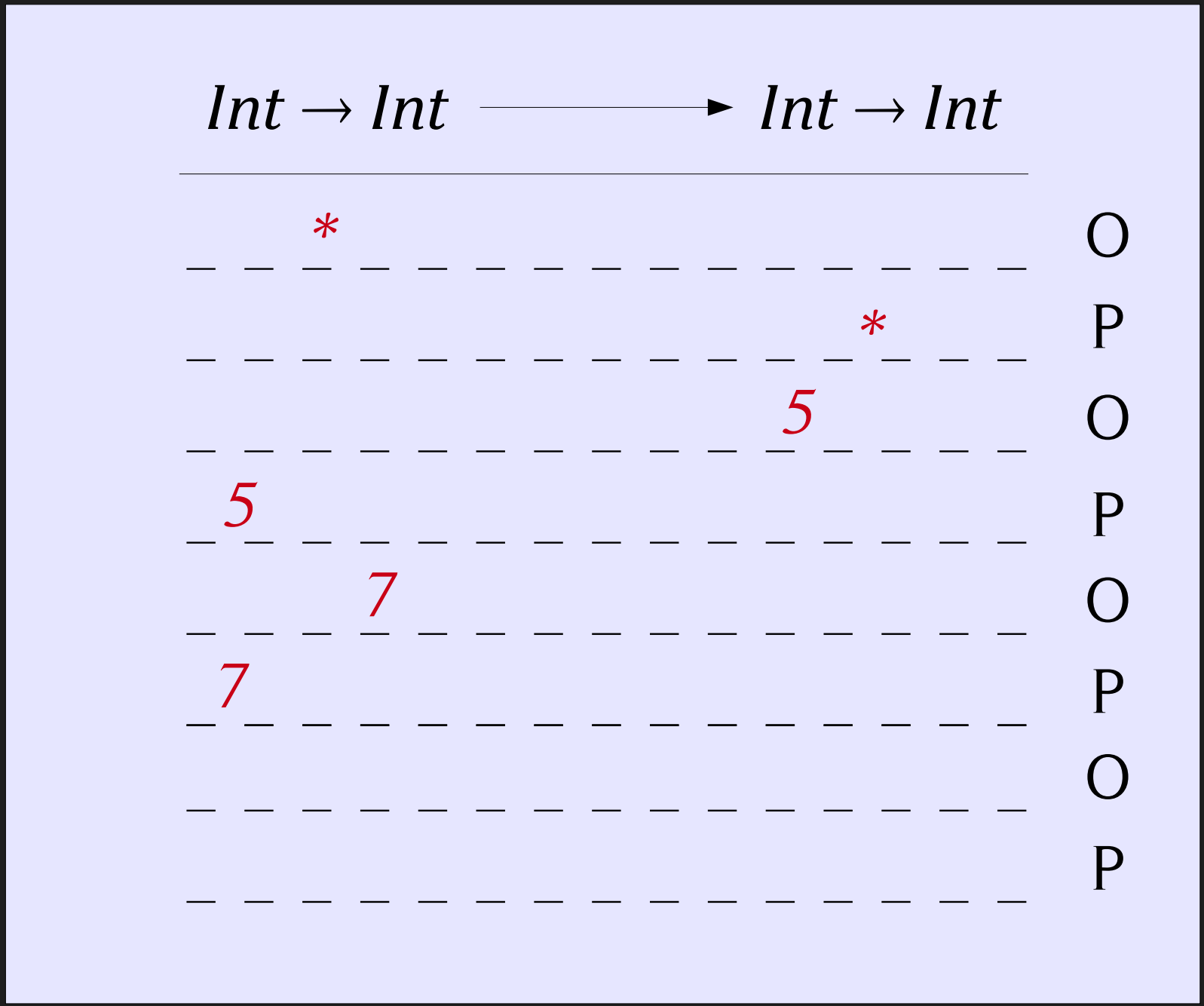
Example



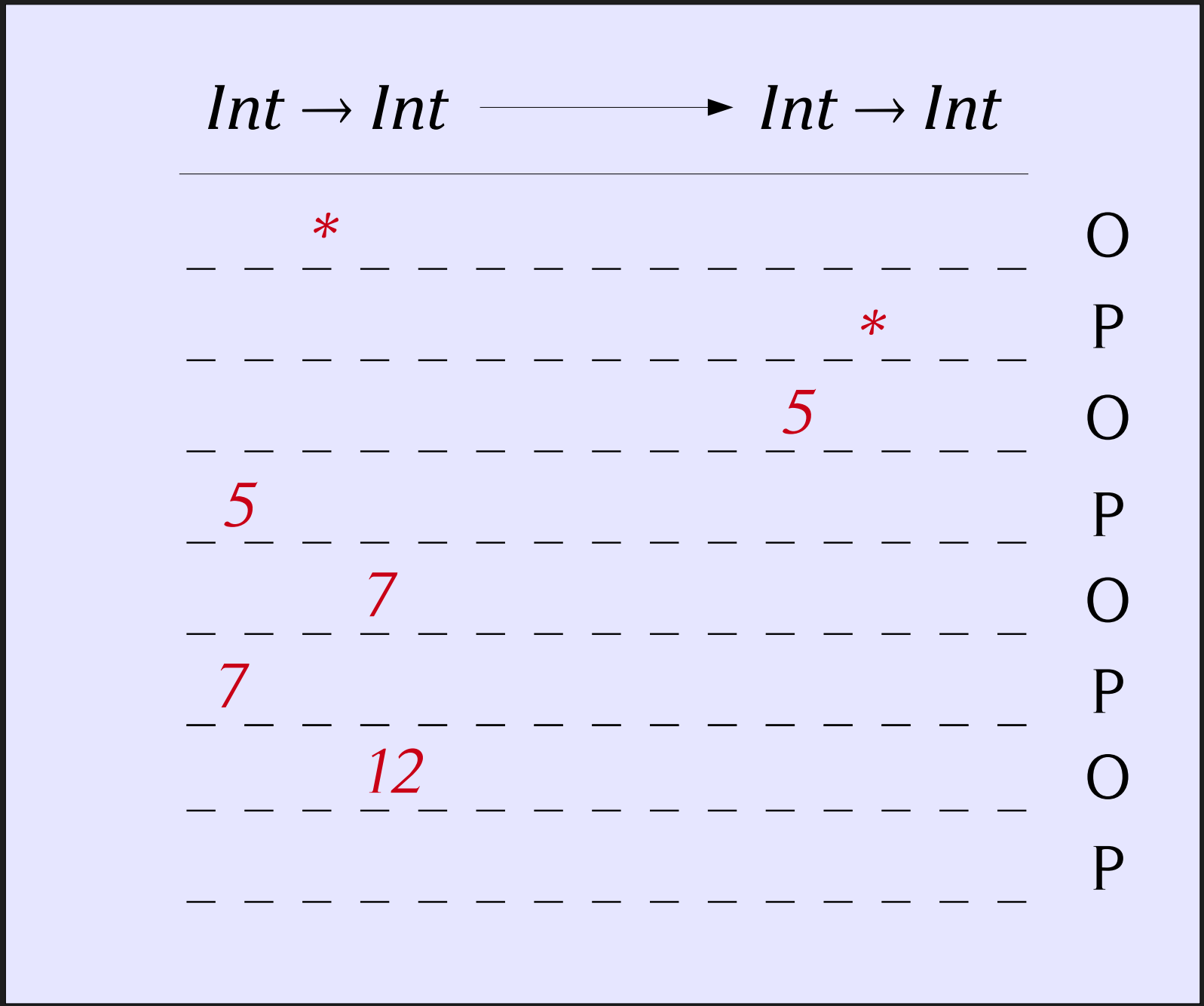
Example



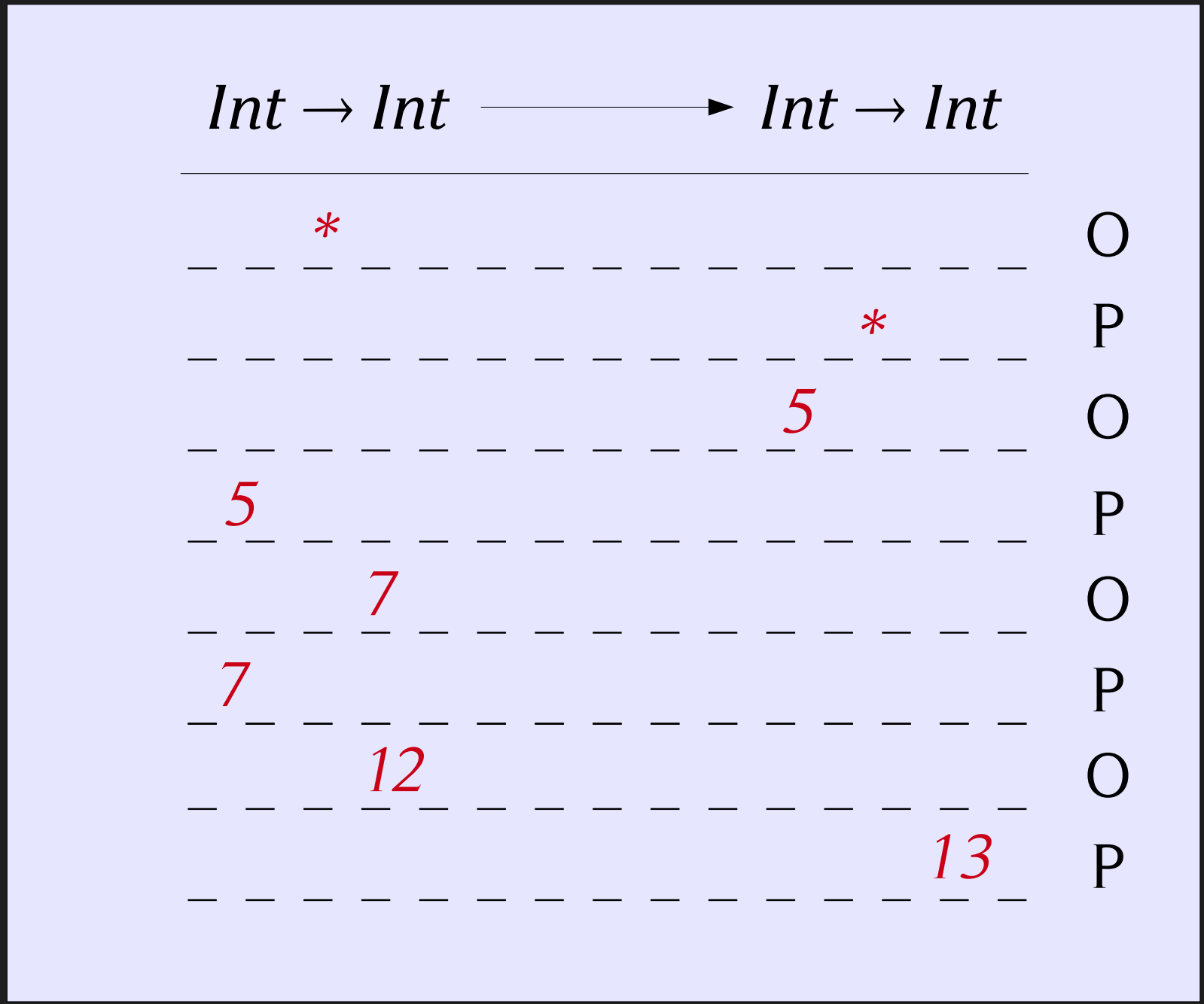
Example



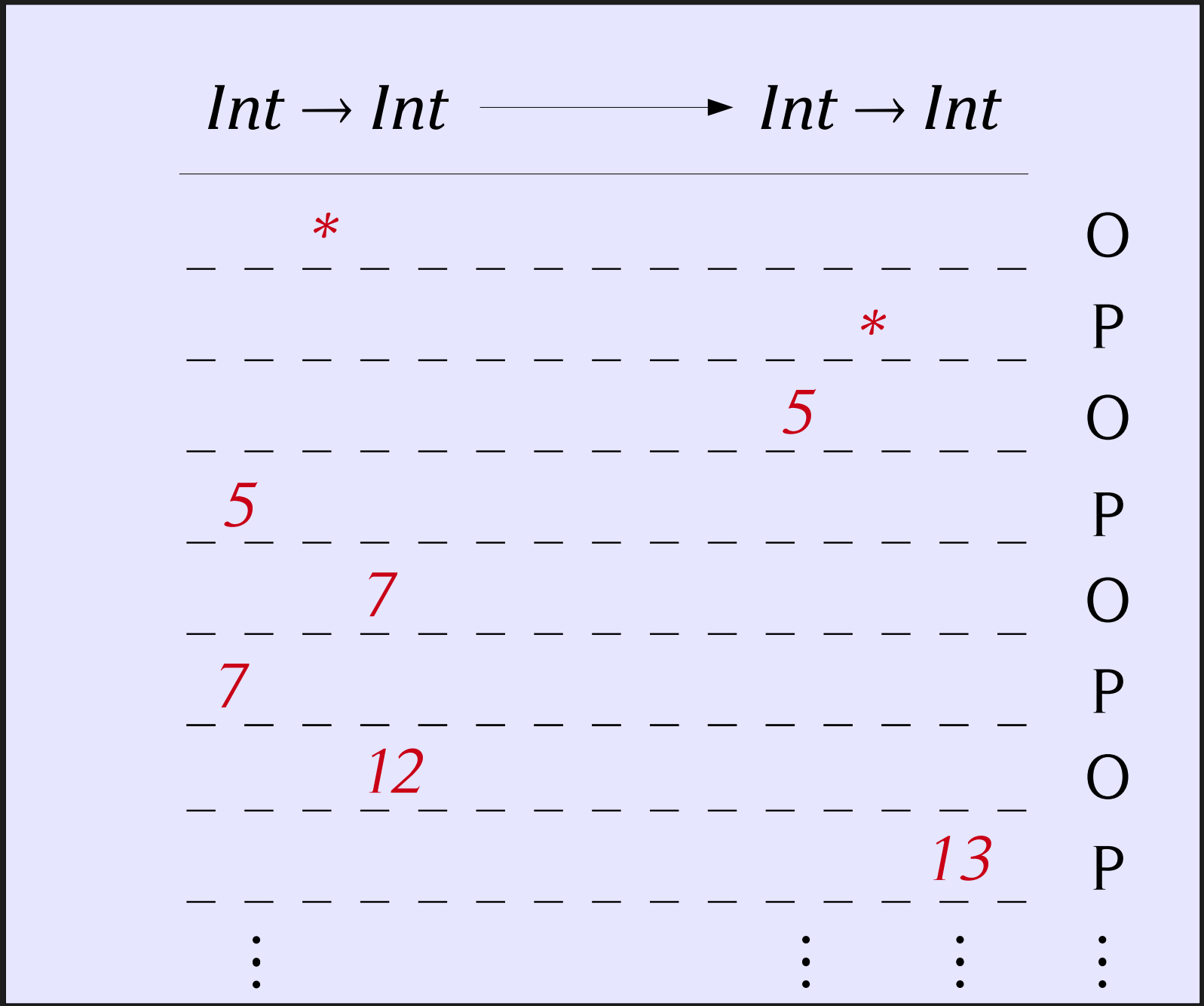
Example



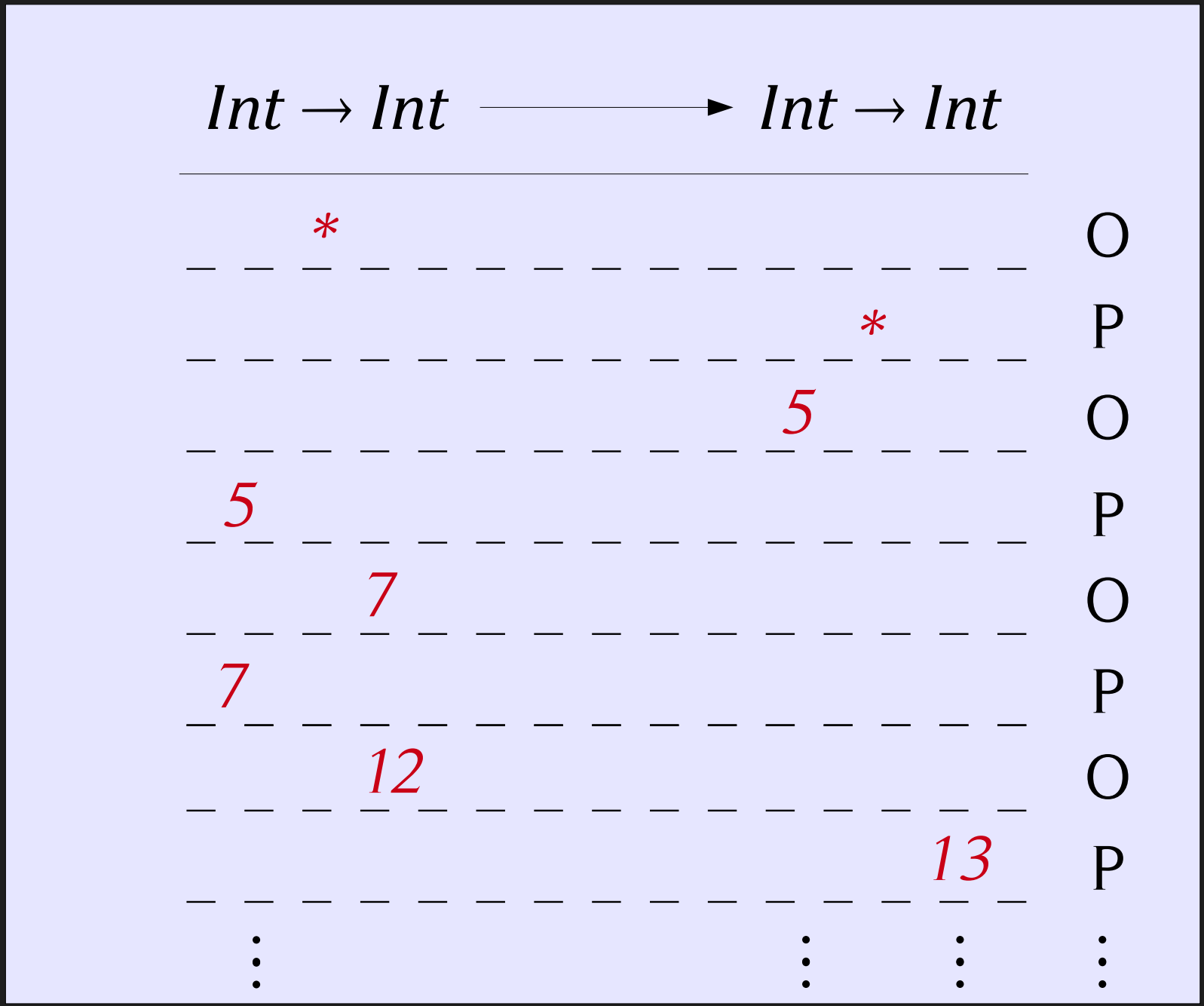
Example



Example



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



Game Semantics

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

Game Semantics

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

Composition

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

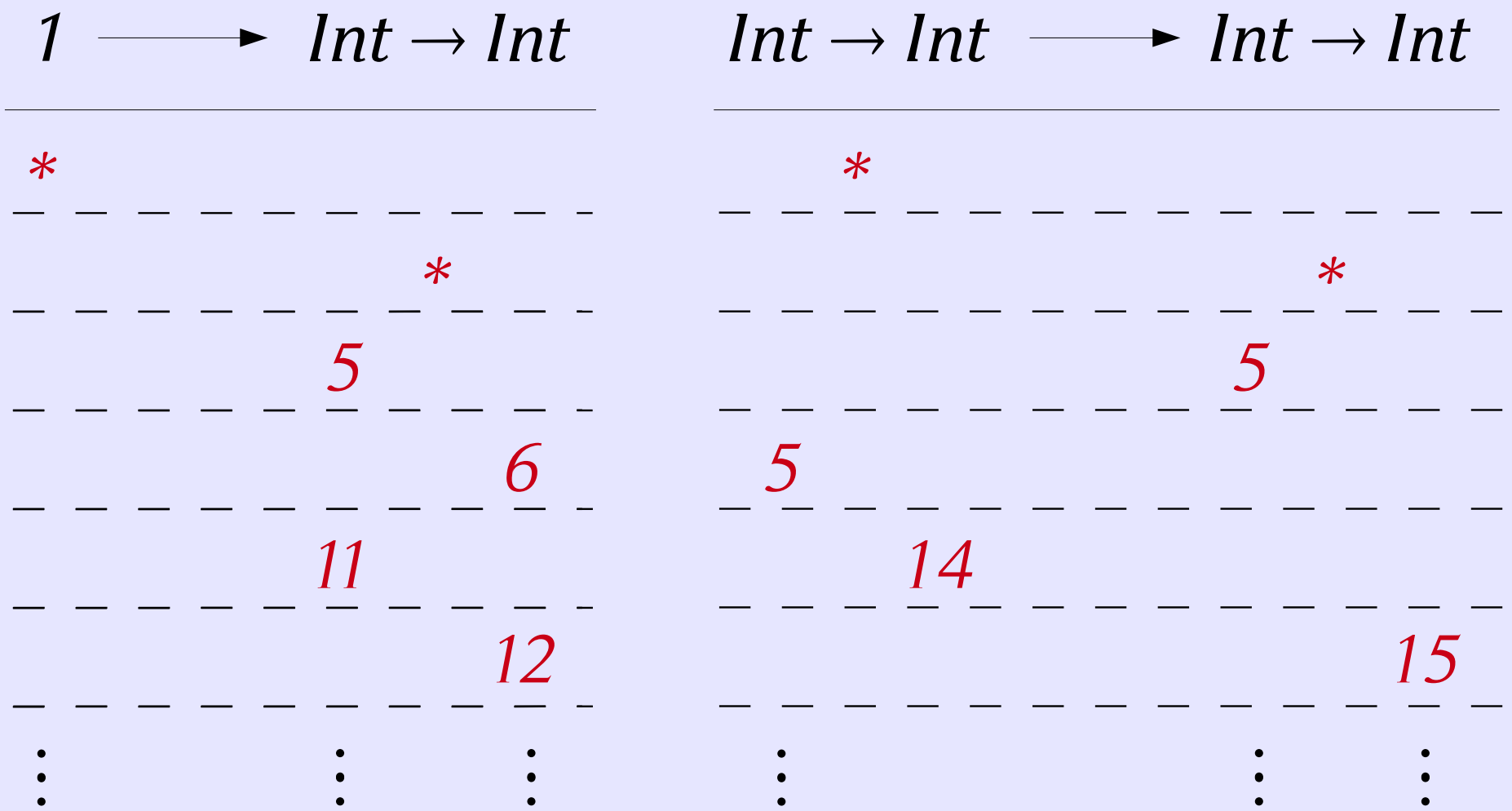
$f : \mathtt{int} \longrightarrow \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \longrightarrow \mathtt{int}$

Composition

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

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

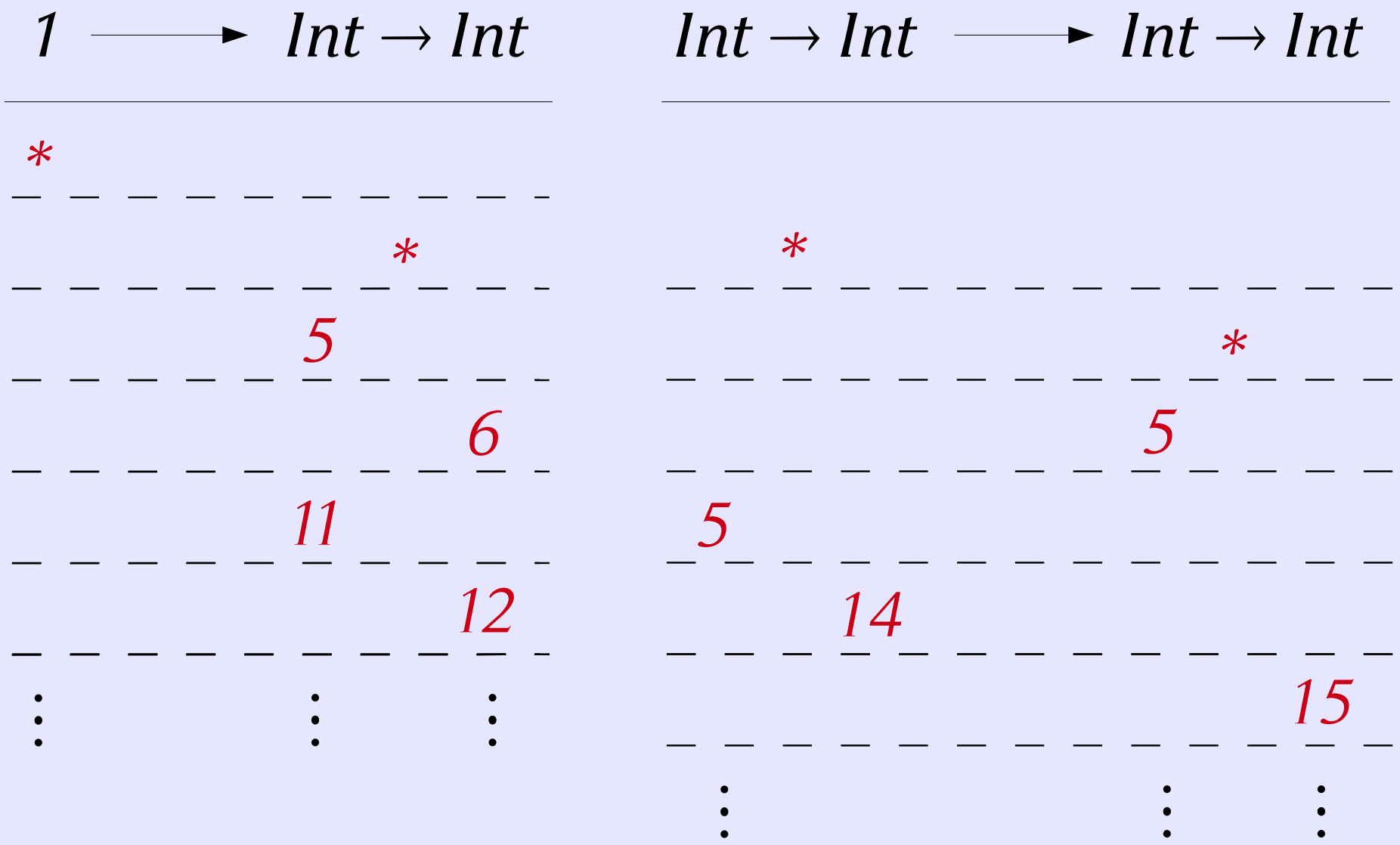
Composition



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

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

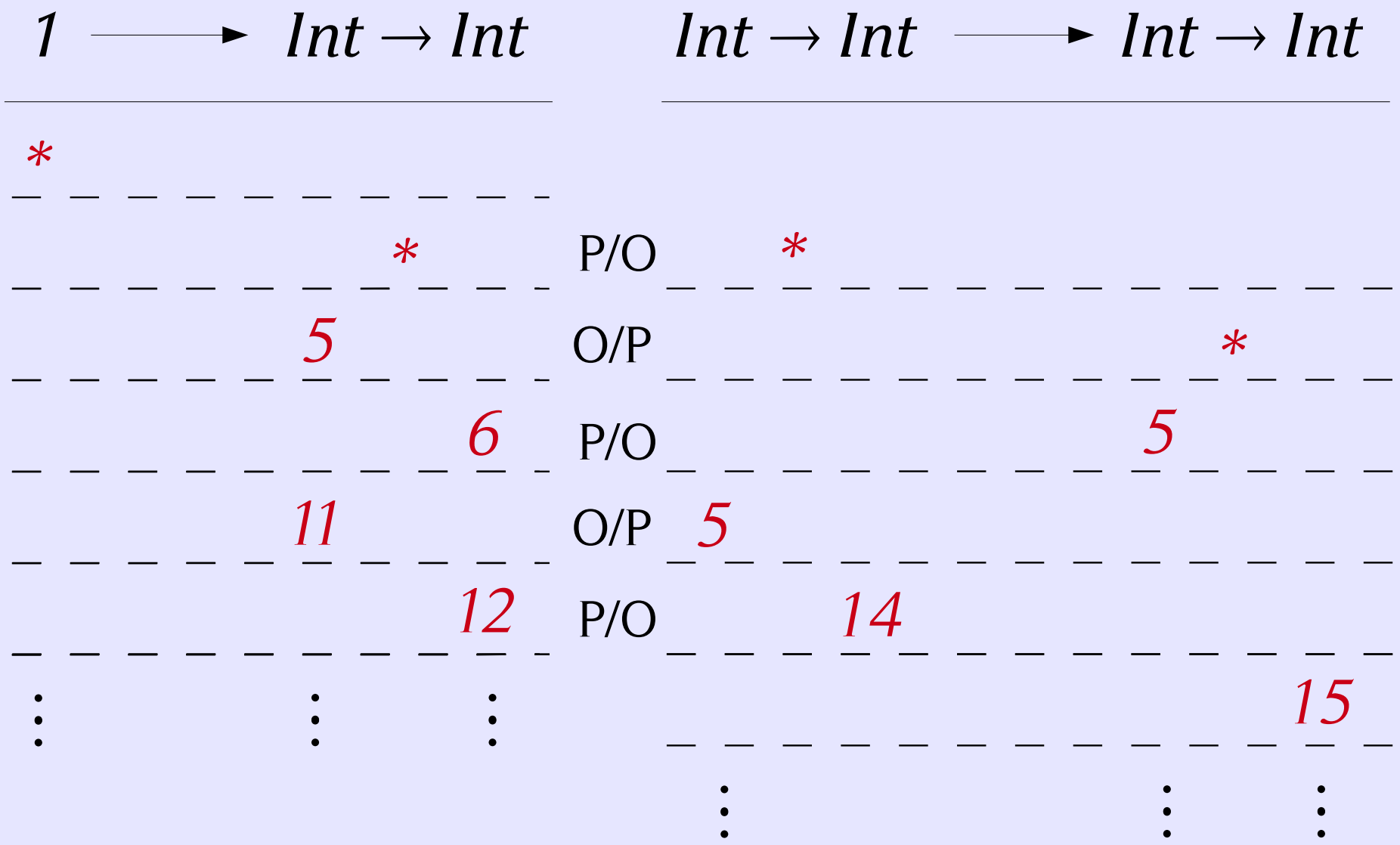
Composition



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

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

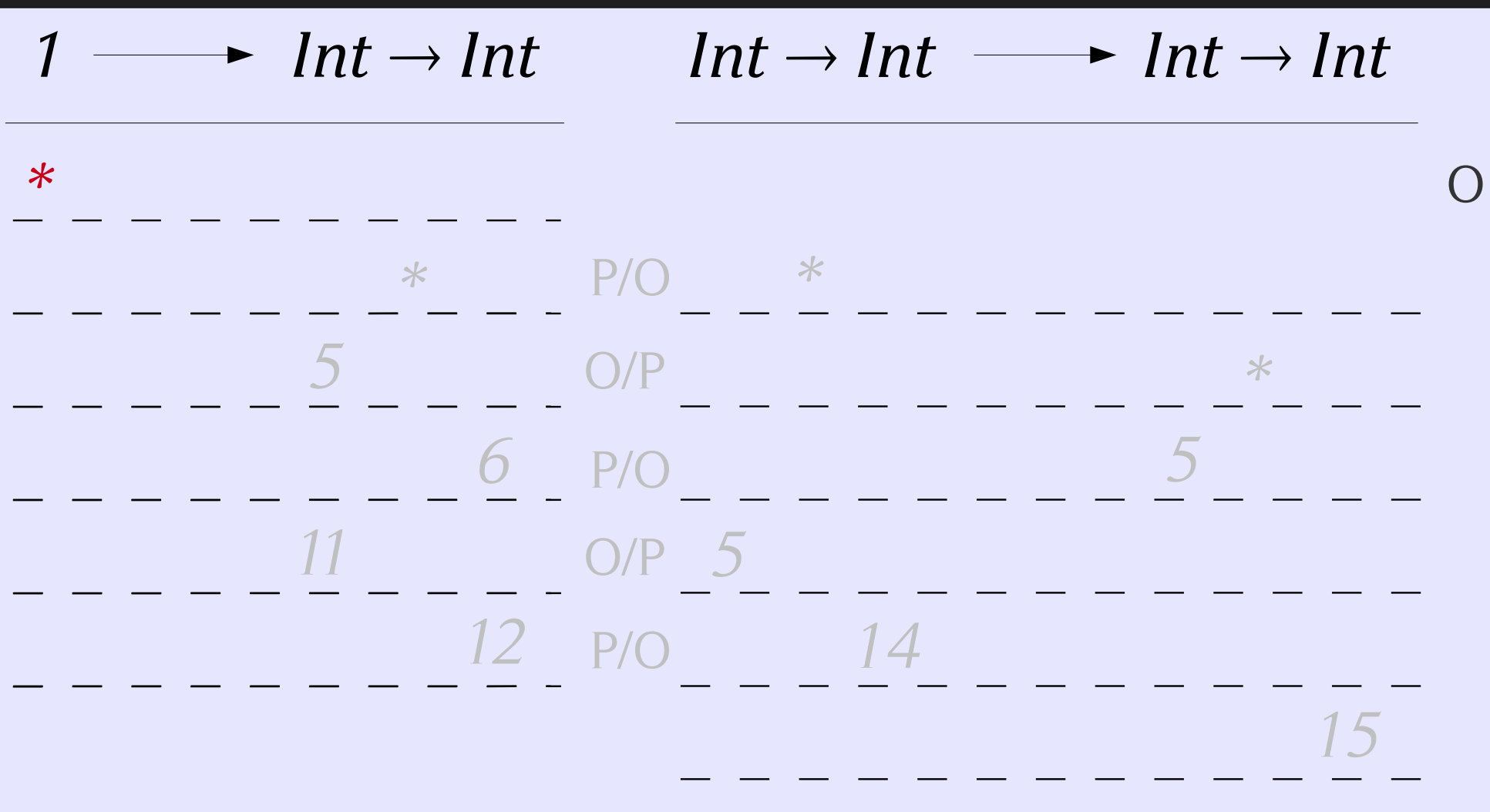
Composition



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

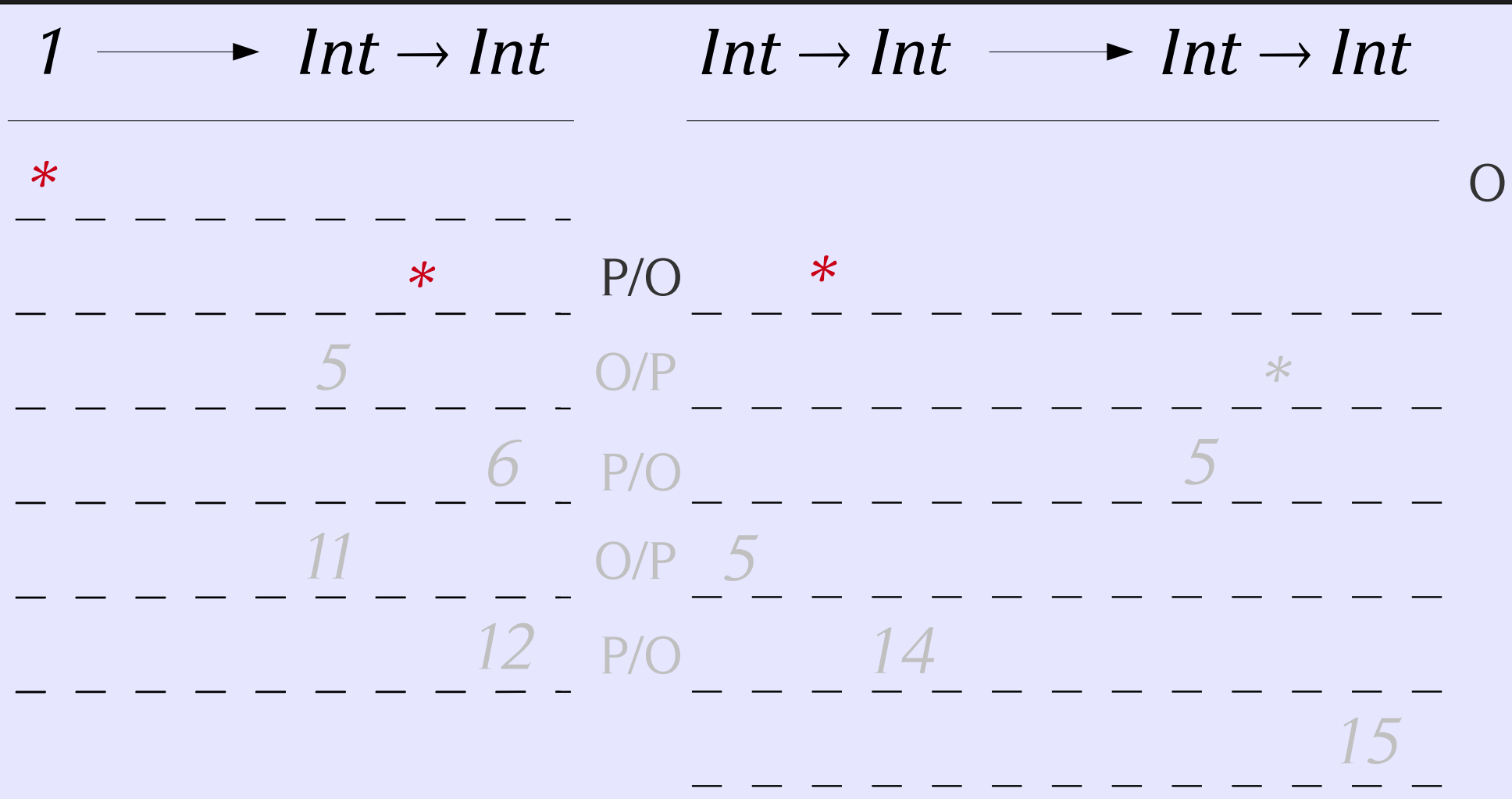
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



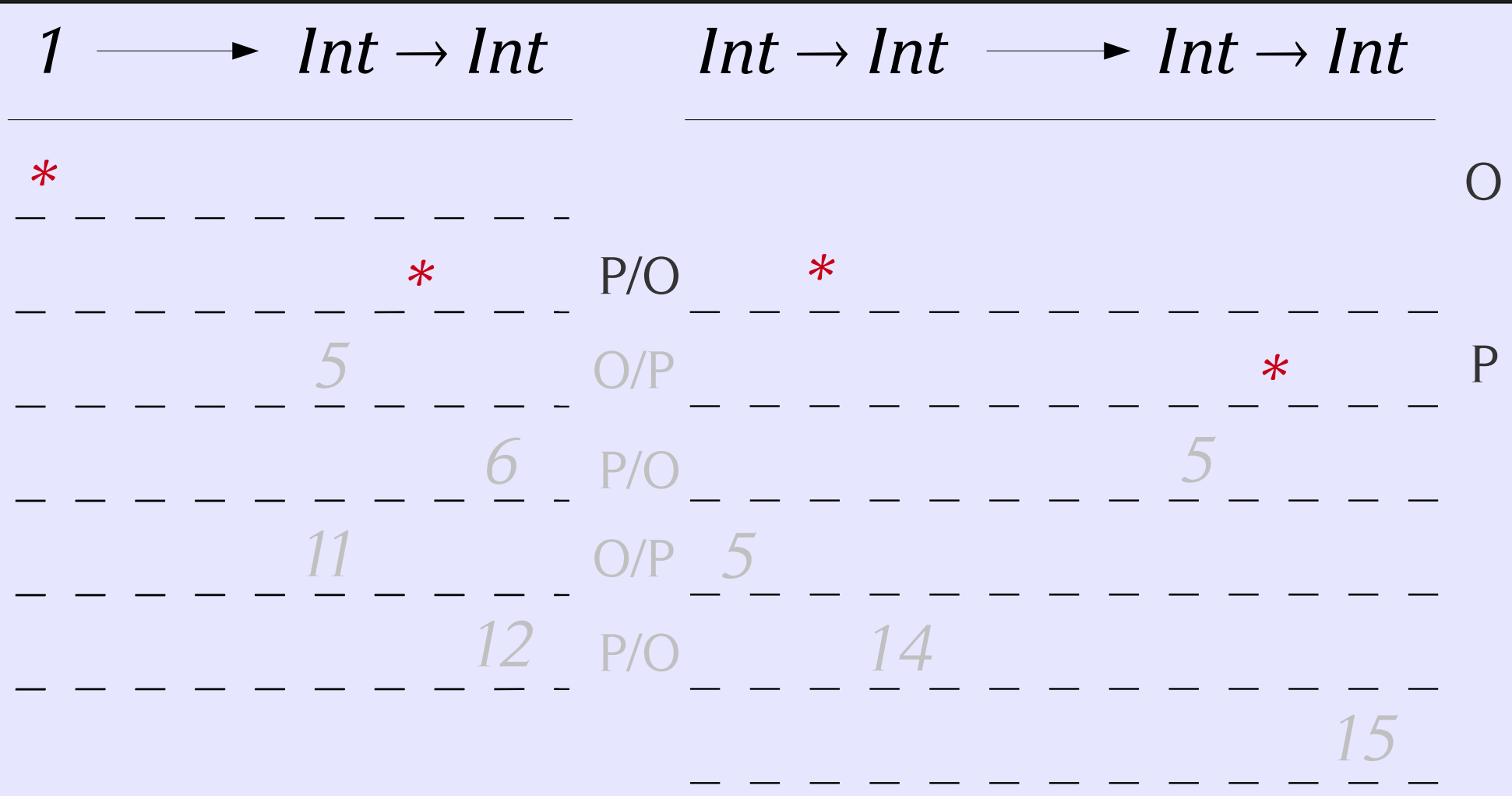
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

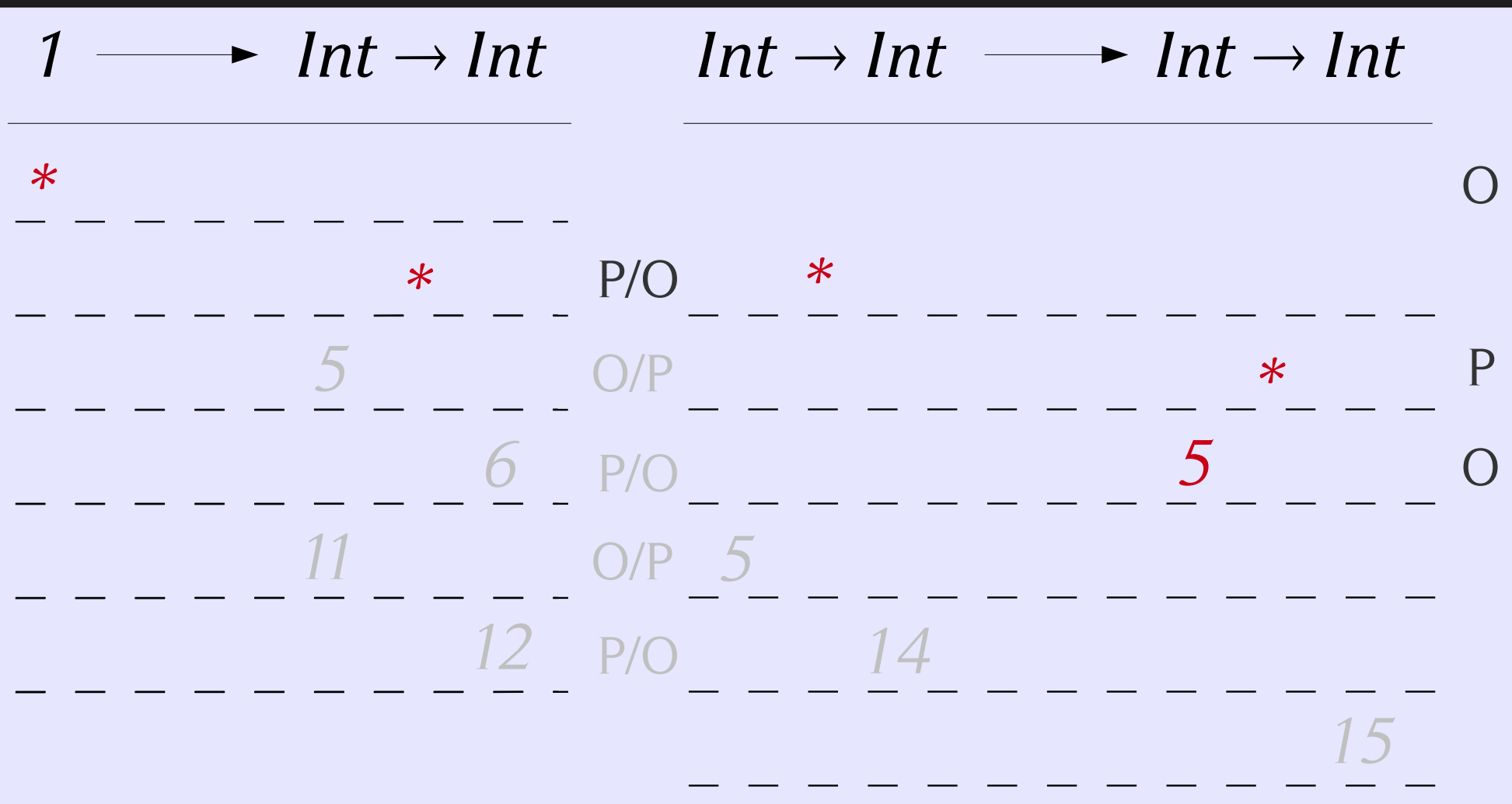
Composition



`⊢ λx.x+1 : int ⟶ int`

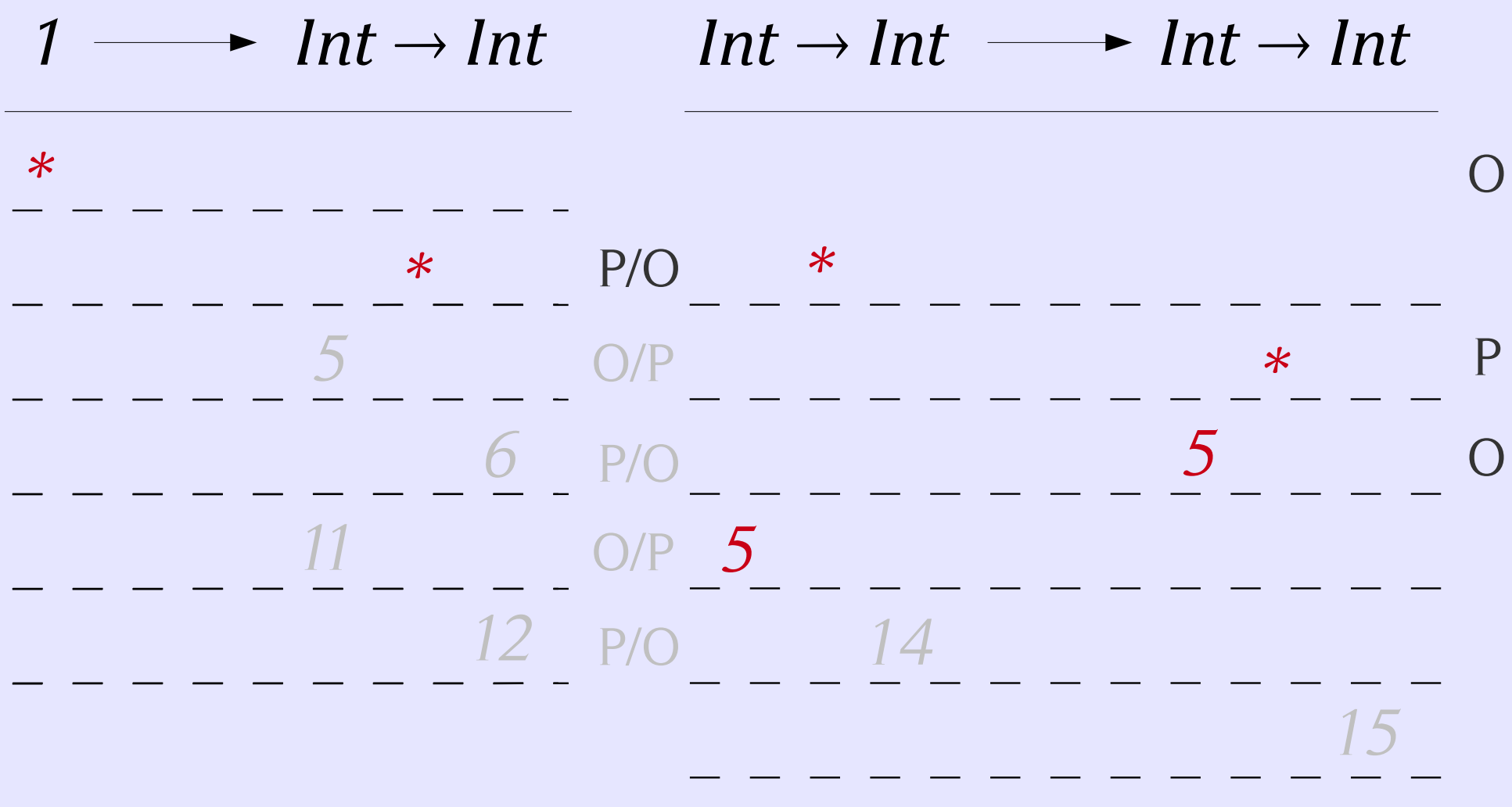
`f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int`

Composition



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

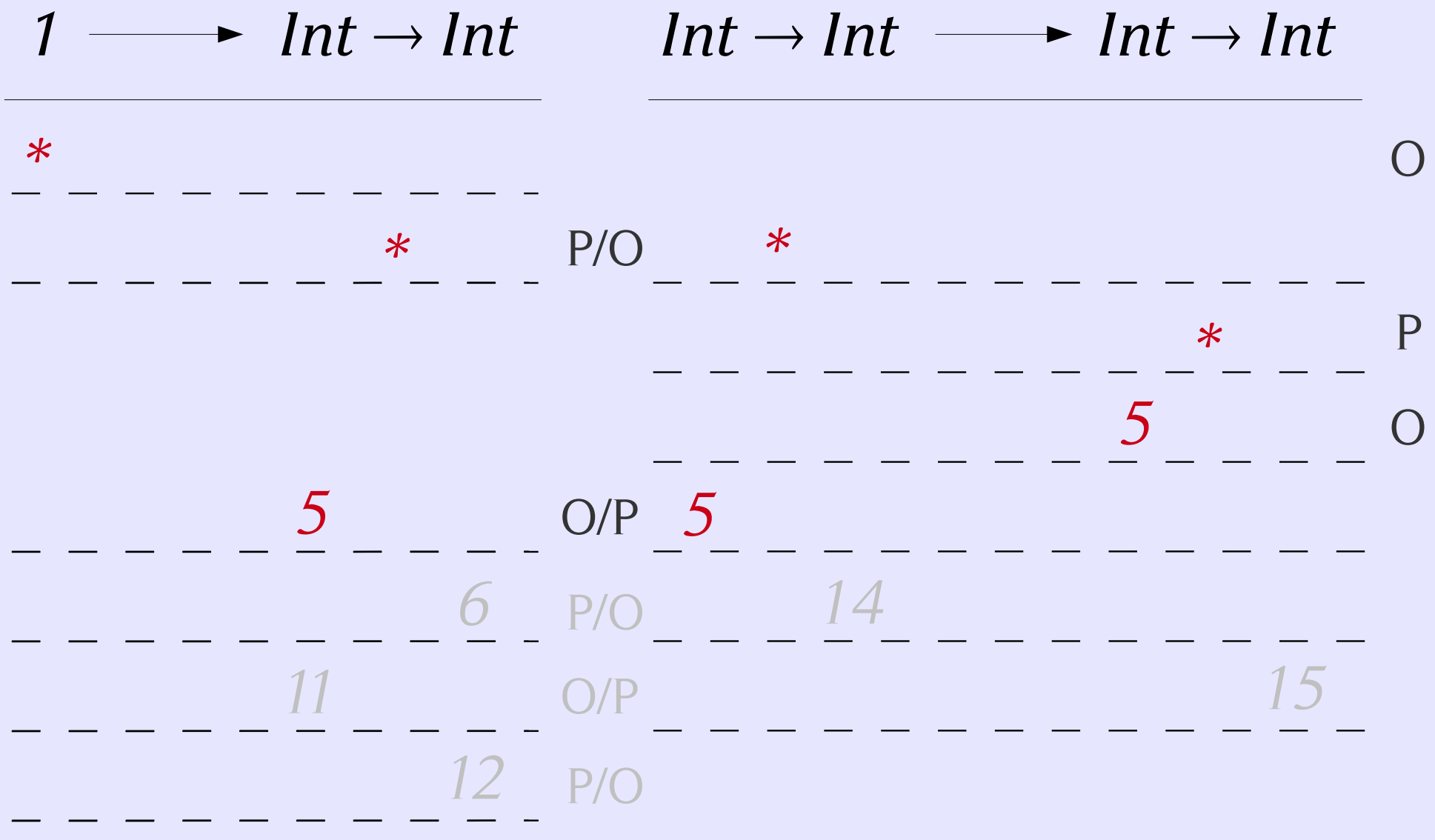
Composition



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

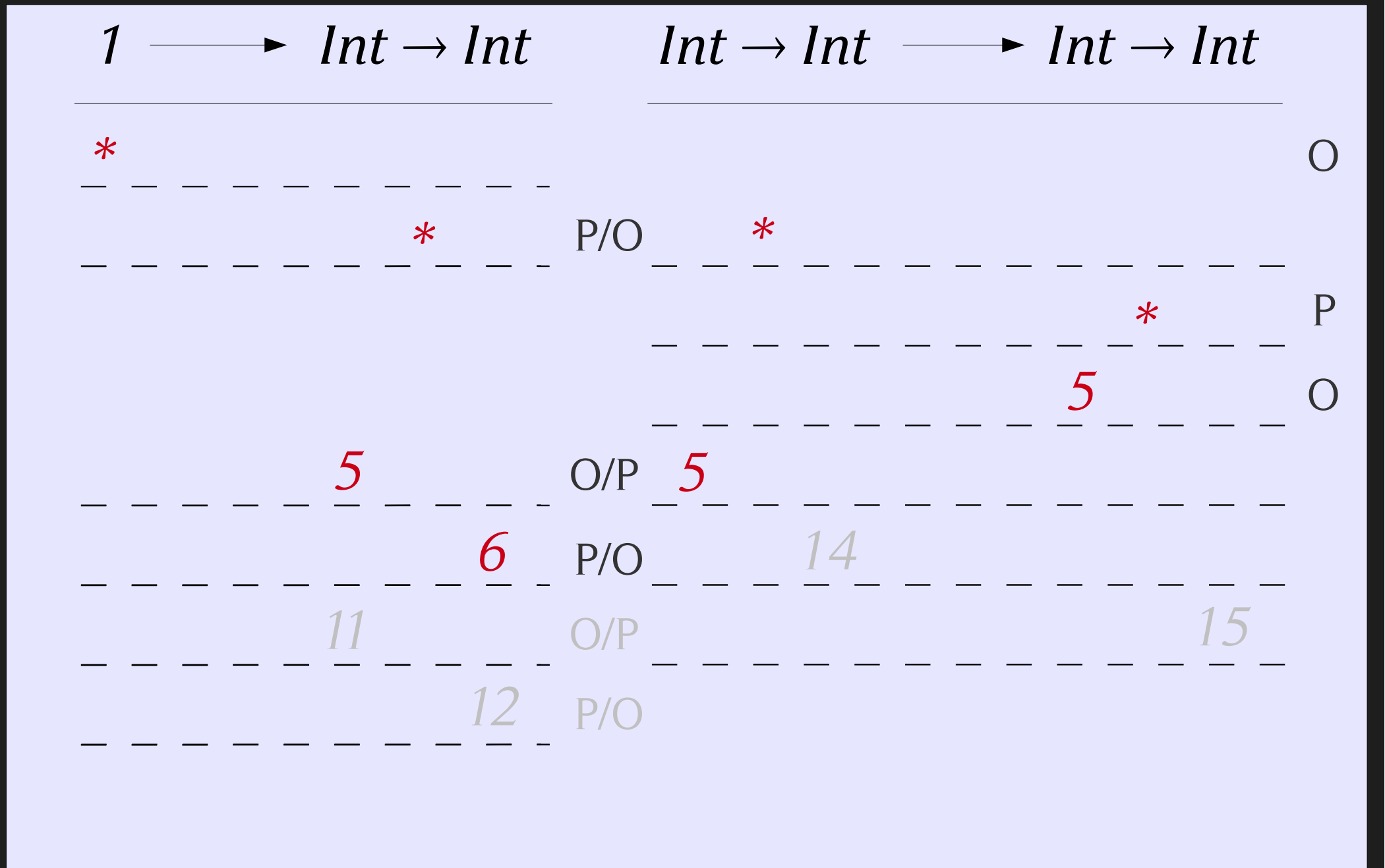
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

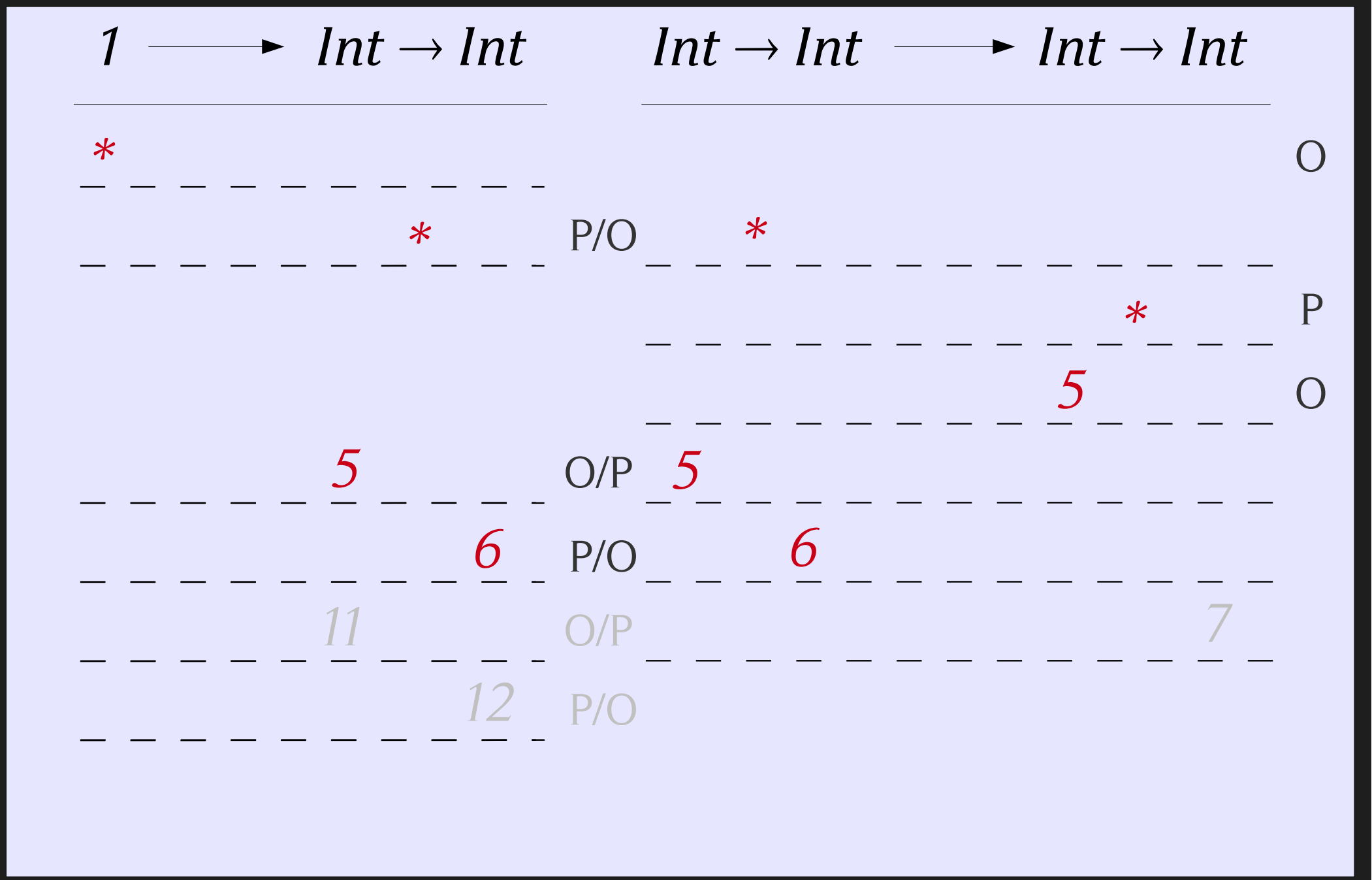
Composition



⊢ λx.x+1 : int ⟶ int

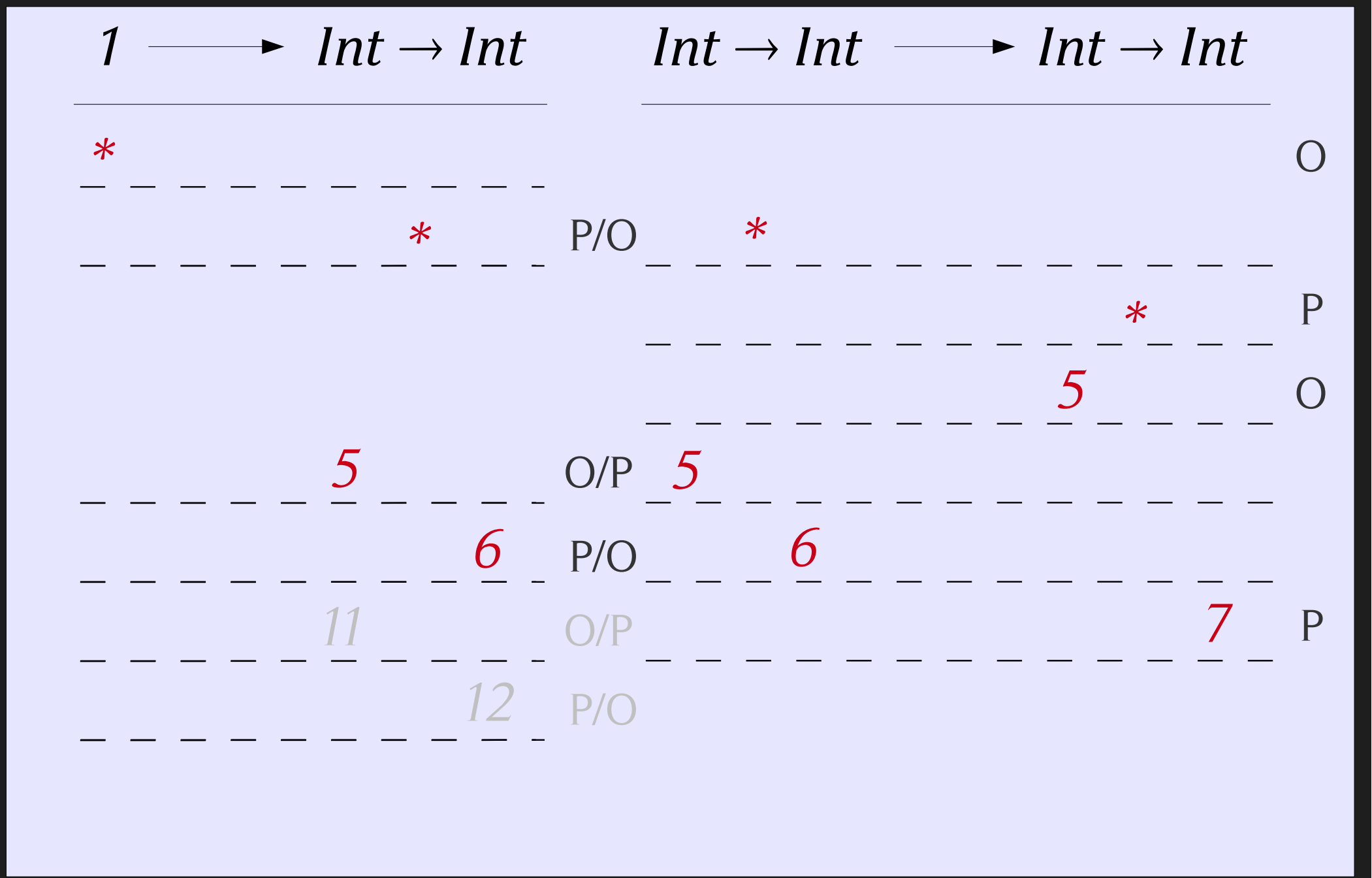
f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

Composition



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

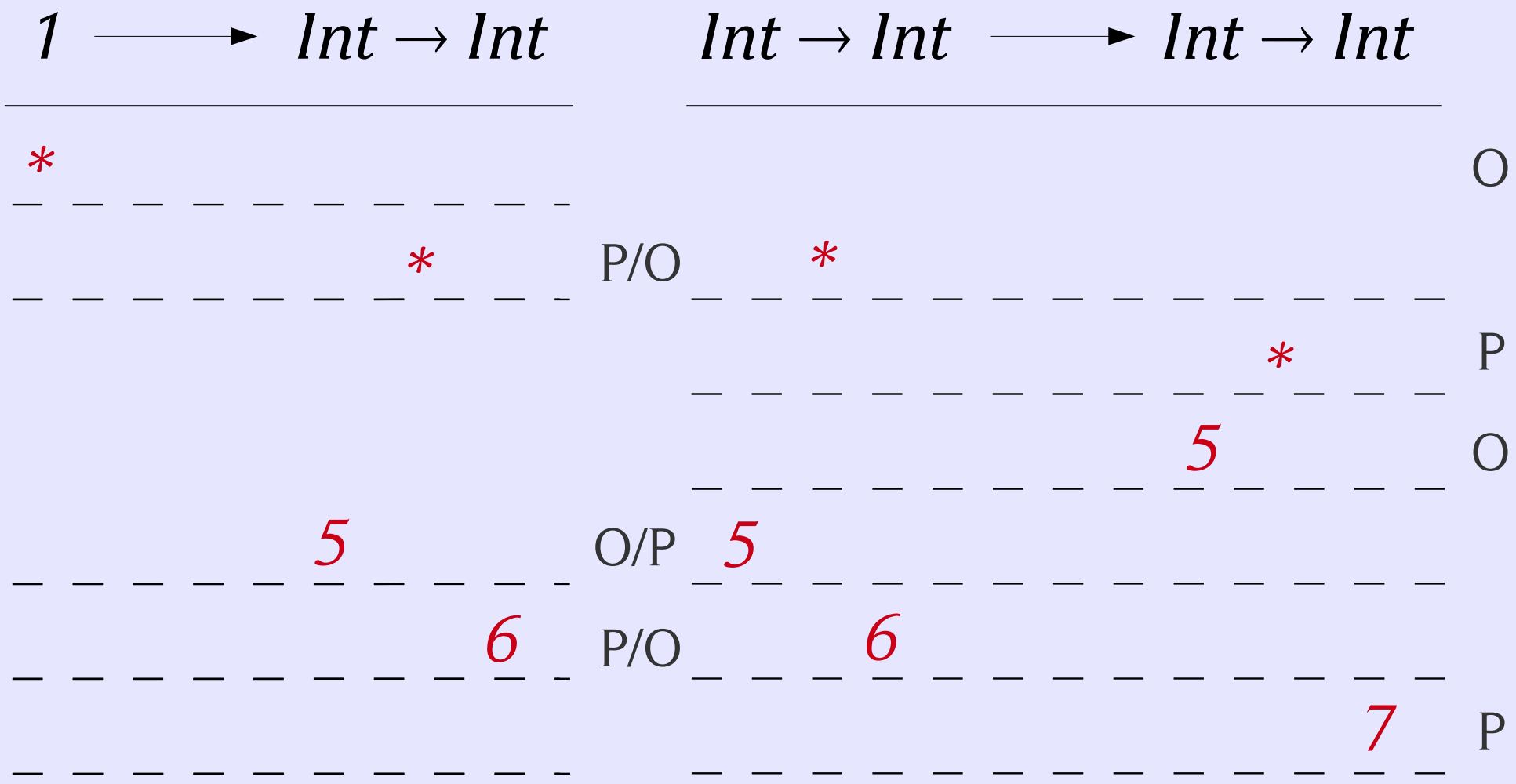
Composition



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

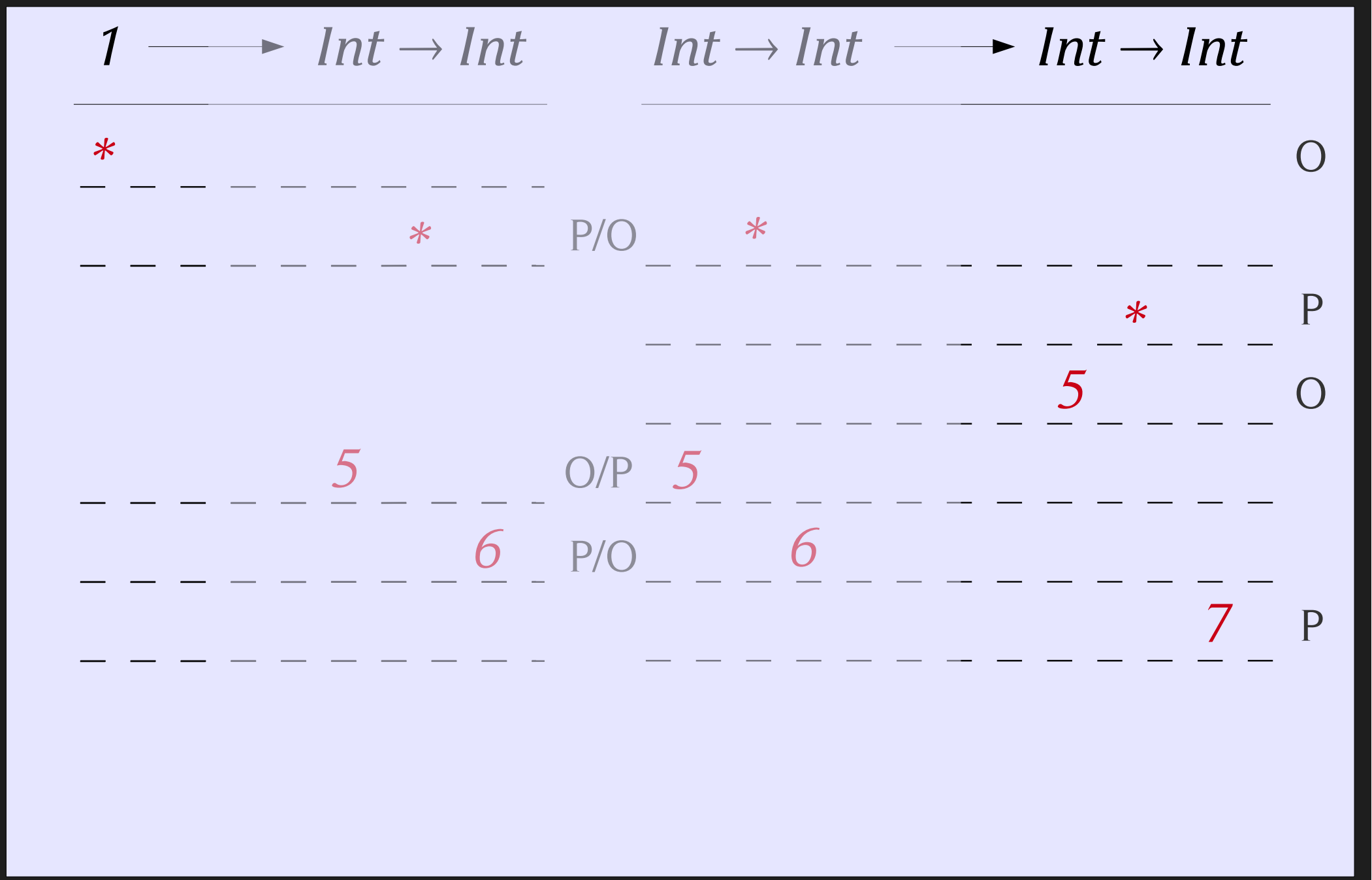
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

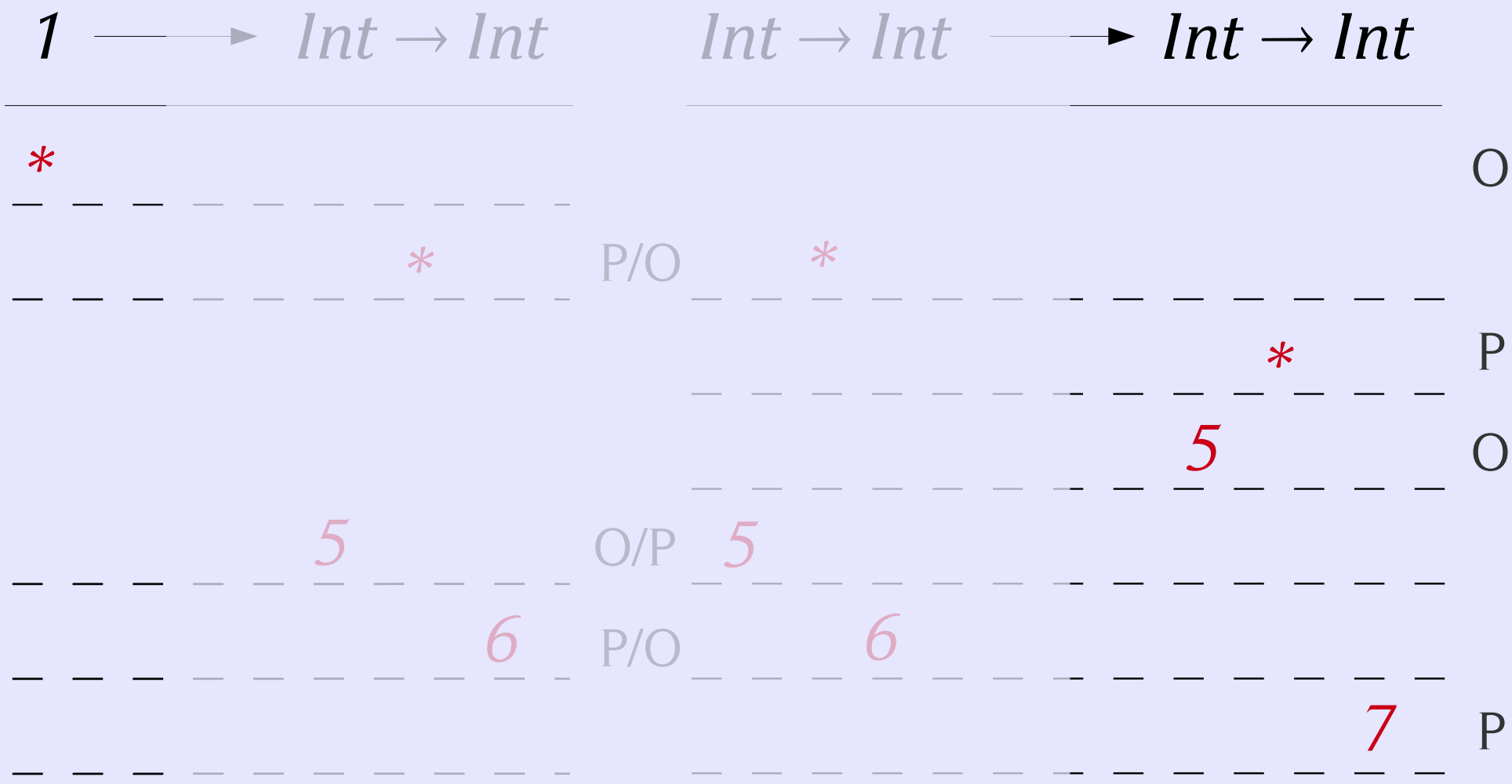
Composition



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

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

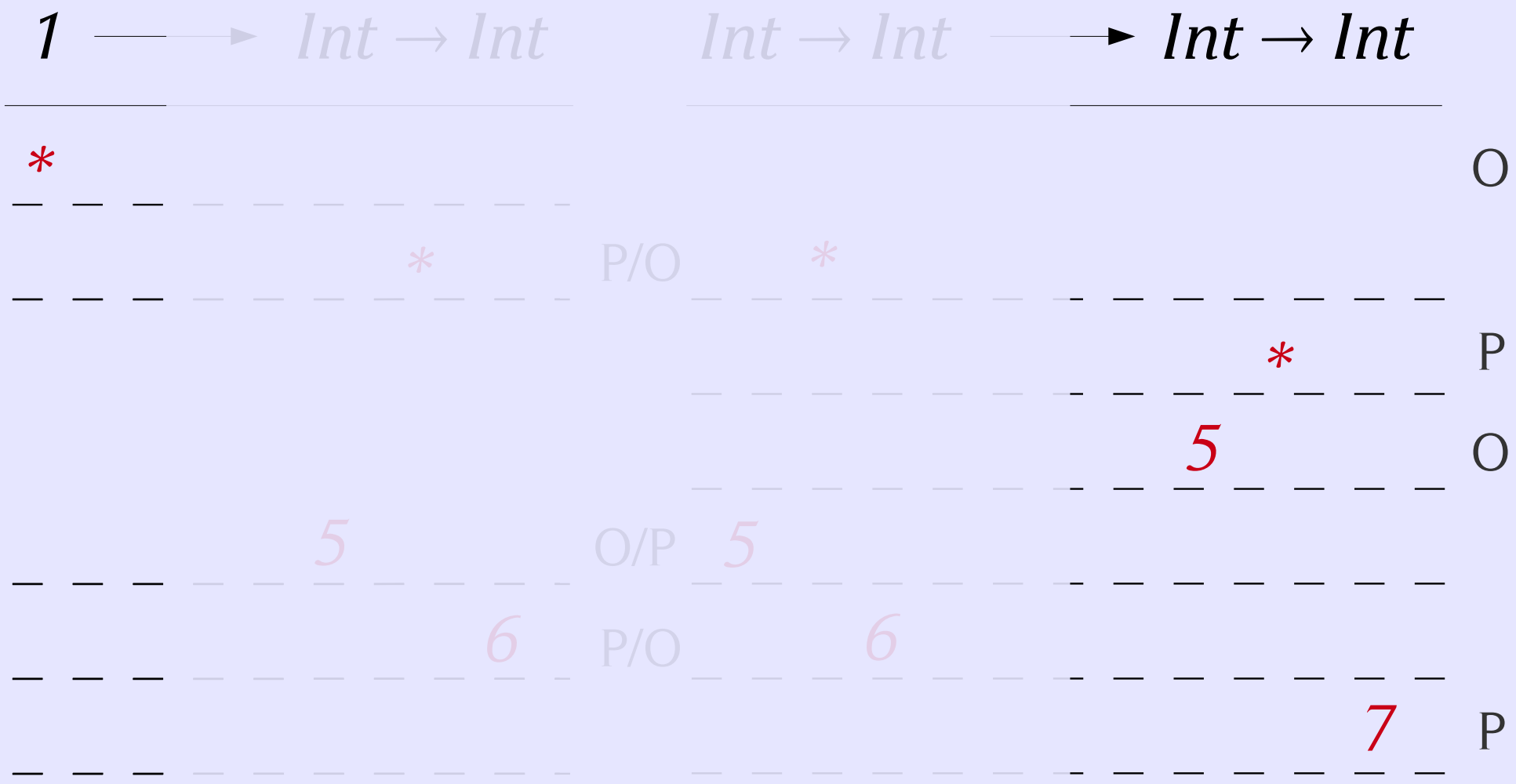
Composition



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

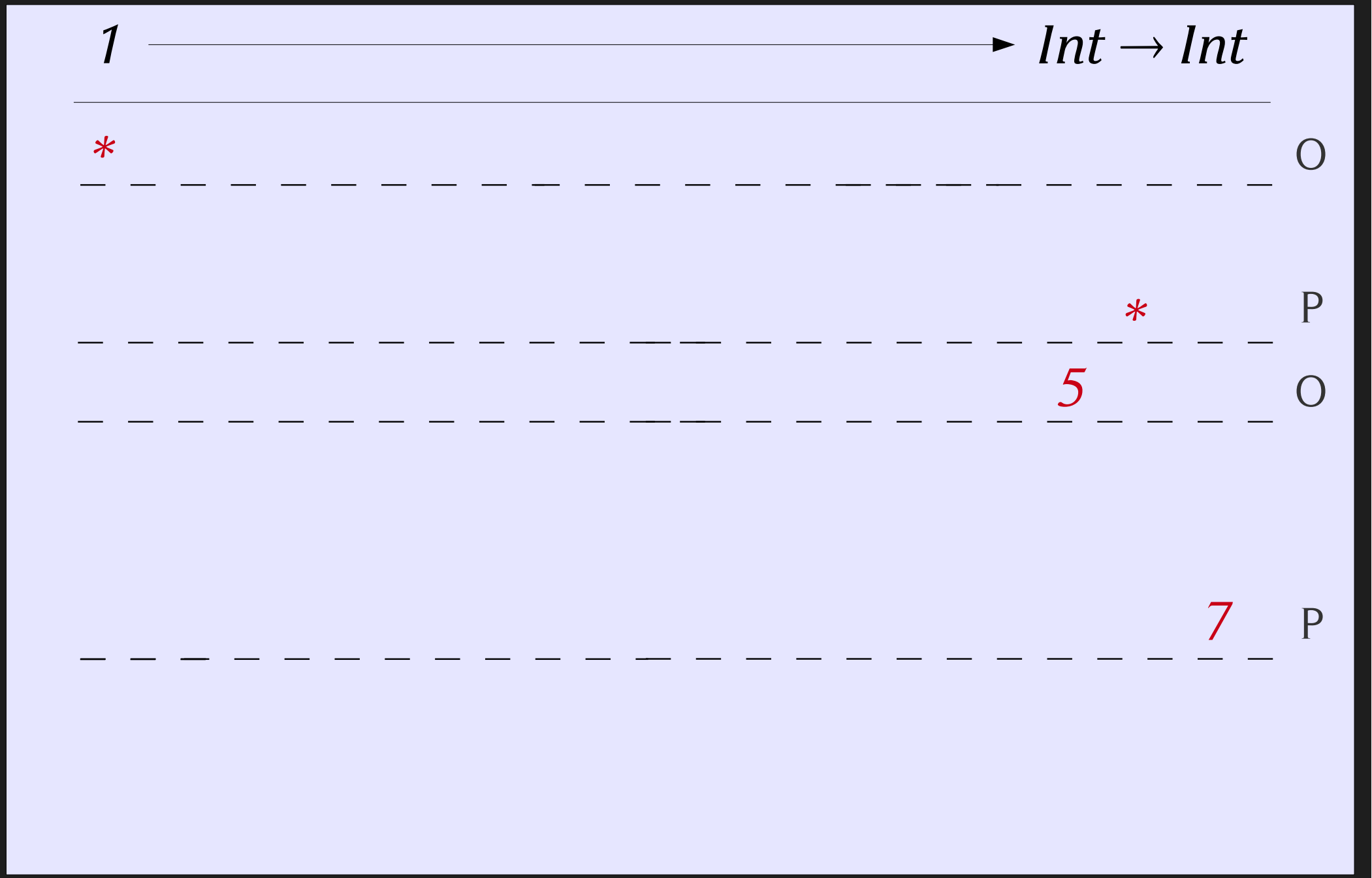
$f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



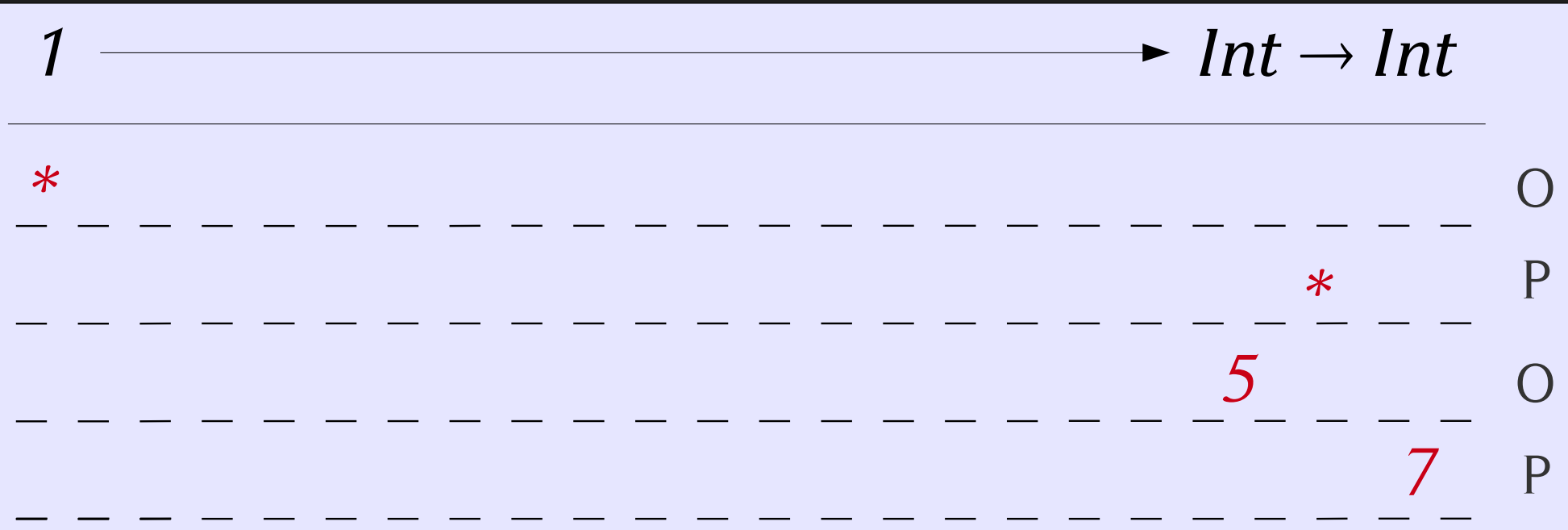
$\vdash \lambda x.x+1 : int \longrightarrow int$ $\quad$ $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



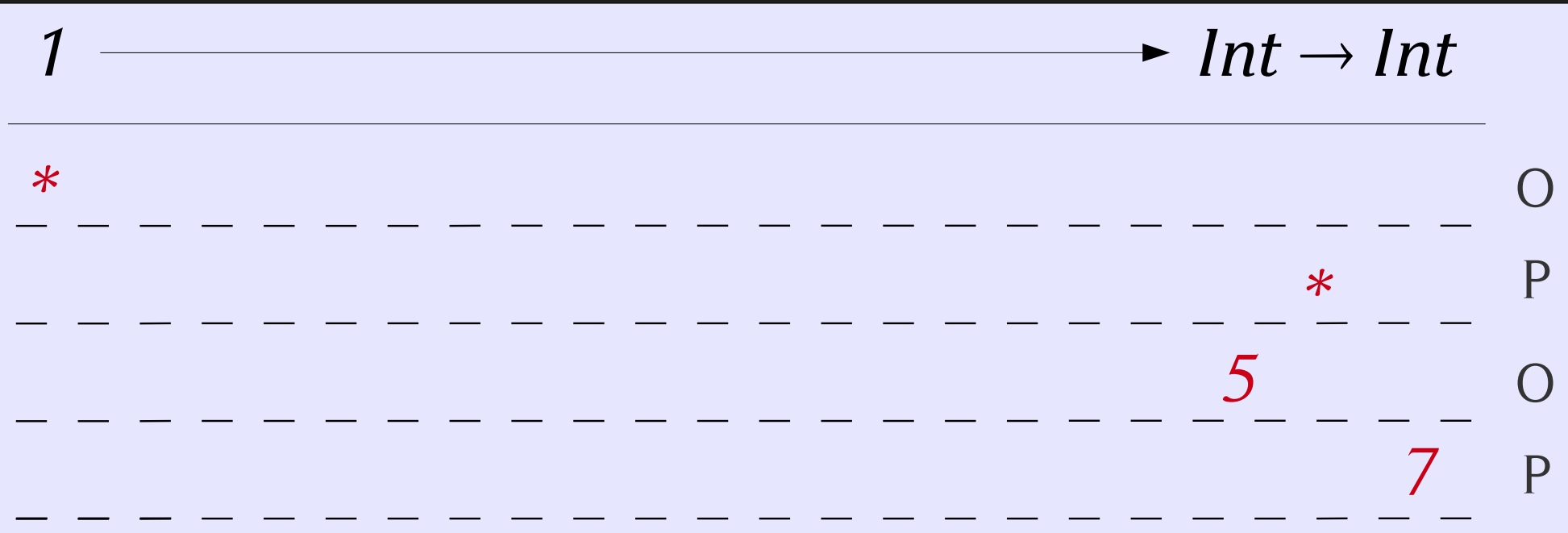
$\vdash \lambda x.x+1 : int \longrightarrow int$ | $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

Composition



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

Composition

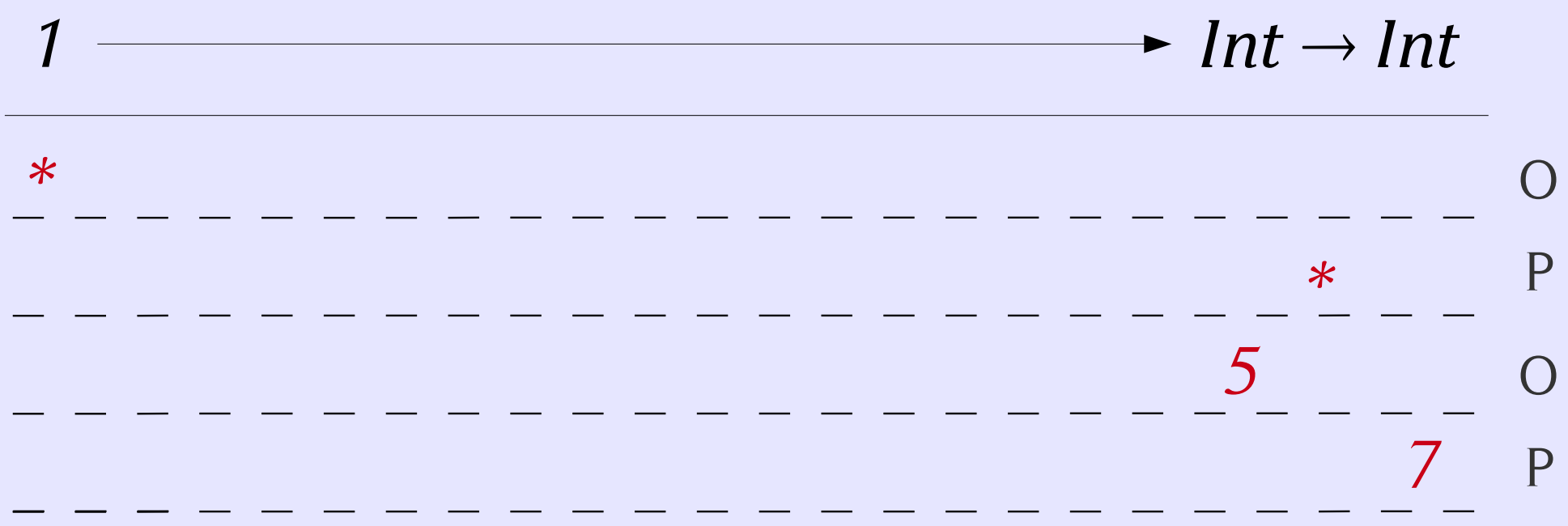


$\vdash \lambda x.x+1 : int \longrightarrow int$; $f : int \longrightarrow int \vdash \lambda x.f(x)+1 : int \longrightarrow int$

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

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

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

Quick storyline

Quick storyline

Introduced in early 90's

Full abstraction for PCF

- Two teams in the UK; one in Siegen
- Roots in Mathematical Logic

Quick storyline

Introduced in early 90's

- Two teams in the UK; one in Siegen
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

Quick storyline

Introduced in early 90's

- Two teams in the UK; one in Siegen
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

Quick storyline

Introduced in early 90's

- Two teams in the UK; one in Siegen
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} new resources?

Nominal game semantics (2004-)

Quick storyline

Introduced in early 90's

- Two teams in the UK; one in Siegen
- Roots in Mathematical Logic

First stage (1993-2004)

- Models for various programming features
- Program analysis

} new resources?

Nominal game semantics (2004-)

- Models of realistic programs (2006-)
- Algorithmic games (2011-)

Computation with new resources

RedML = simply-typed lambda-calculus
+ integer variables (*names*)

Computation with new resources

RedML = simply-typed lambda-calculus
+ integer variables (*names*)

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

Computation with new resources

RedML = simply-typed lambda-calculus
+ integer variables (*names*)

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

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

Computation with new resources

RedML = simply-typed lambda-calculus
+ integer variables (*names*)

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

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

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

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

Computation with new resources

RedML = simply-typed lambda-calculus
+ integer variables (*names*)

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

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

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

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

Nominal techniques

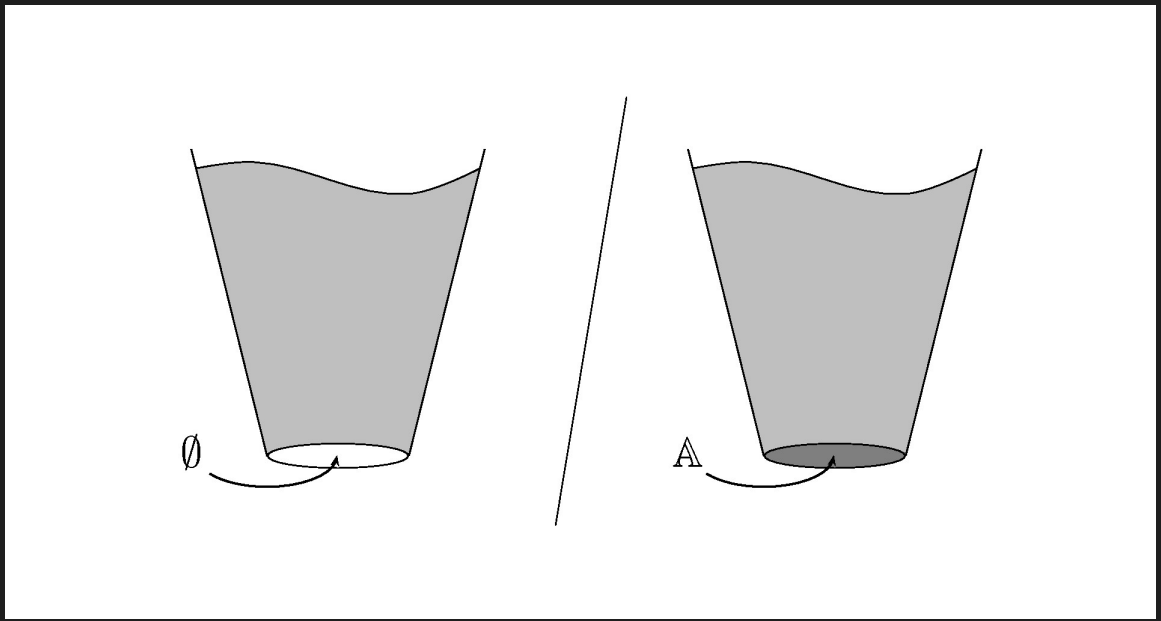
- A foundational theory for doing maths with names and name-binding

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

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 Logic

Example axioms (note sorts should match):

$$\text{И}a.\ \phi(\vec{x}) \iff \exists a.\ a\#\vec{x} \wedge \phi(\vec{x}) \qquad \text{(Q)}$$

$$(a\ a') \cdot (b\ b') \cdot x = ((a\ a') \cdot b\ (a\ a') \cdot b') \cdot (a\ a') \cdot x \qquad \text{(E1)}$$

$$b\#x \implies (a\ a') \cdot b \# (a\ a') \cdot x \qquad \text{(E2)}$$

$$a\#x \wedge a'\#x \implies (a\ a') \cdot x = x \qquad \text{(F1)}$$

$$a.x = a'.x' \iff (a = a' \vee a'\#x) \wedge x' = (a\ a') \cdot x \qquad \text{(A1)}$$

Nominal Logic

Example axioms (note sorts should match):

$$\text{И}a.\ \phi(\vec{x}) \iff \exists a.\ a\#\vec{x} \wedge \phi(\vec{x}) \qquad \text{(Q)}$$

$$(a\ a') \cdot (b\ b') \cdot x = ((a\ a') \cdot b\ (a\ a') \cdot b') \cdot (a\ a') \cdot x \qquad \text{(E1)}$$

$$b\#x \implies (a\ a') \cdot b \# (a\ a') \cdot x \qquad \text{(E2)}$$

$$a\#x \wedge a'\#x \implies (a\ a') \cdot x = x \qquad \text{(F1)}$$

$$a.x = a'.x' \iff (a = a' \vee a'\#x) \wedge x' = (a\ a') \cdot x \qquad \text{(A1)}$$

Nominal Logic

Example axioms (note sorts should match):

$$И a.\ \phi(\vec{x}) \iff \exists a.\ a\#\vec{x} \wedge \phi(\vec{x}) \qquad \text{(Q)}$$

$$(a\ a') \cdot (b\ b') \cdot x = ((a\ a') \cdot b\ (a\ a') \cdot b') \cdot (a\ a') \cdot x \qquad \text{(E1)}$$

$$b\#x \implies (a\ a') \cdot b \# (a\ a') \cdot x \qquad \text{(E2)}$$

$$a\#x \wedge a'\#x \implies (a\ a') \cdot x = x \qquad \text{(F1)}$$

$$a.x = a'.x' \iff (a = a' \vee a'\#x) \wedge x' = (a\ a') \cdot x \qquad \text{(A1)}$$

NL gives us a strong handle on names. For example:

- $\phi(\vec{x}) \iff \phi((a\ a') \cdot \vec{x})$
- $(\exists a.\ a\#\vec{x} \wedge \phi(\vec{x})) \iff (\forall a.\ a\#\vec{x} \implies \phi(\vec{x}))$
- $b \# a.x \iff b = a \vee b \# x$
- $a.x = a'.x' \iff И b.\ (a\ b) \cdot x = (a'\ b) \cdot x'$

Nominal games

Nominal games

$\vdash \lambda x.\texttt{newvar}(0) : \texttt{com} \to \texttt{intvar}$

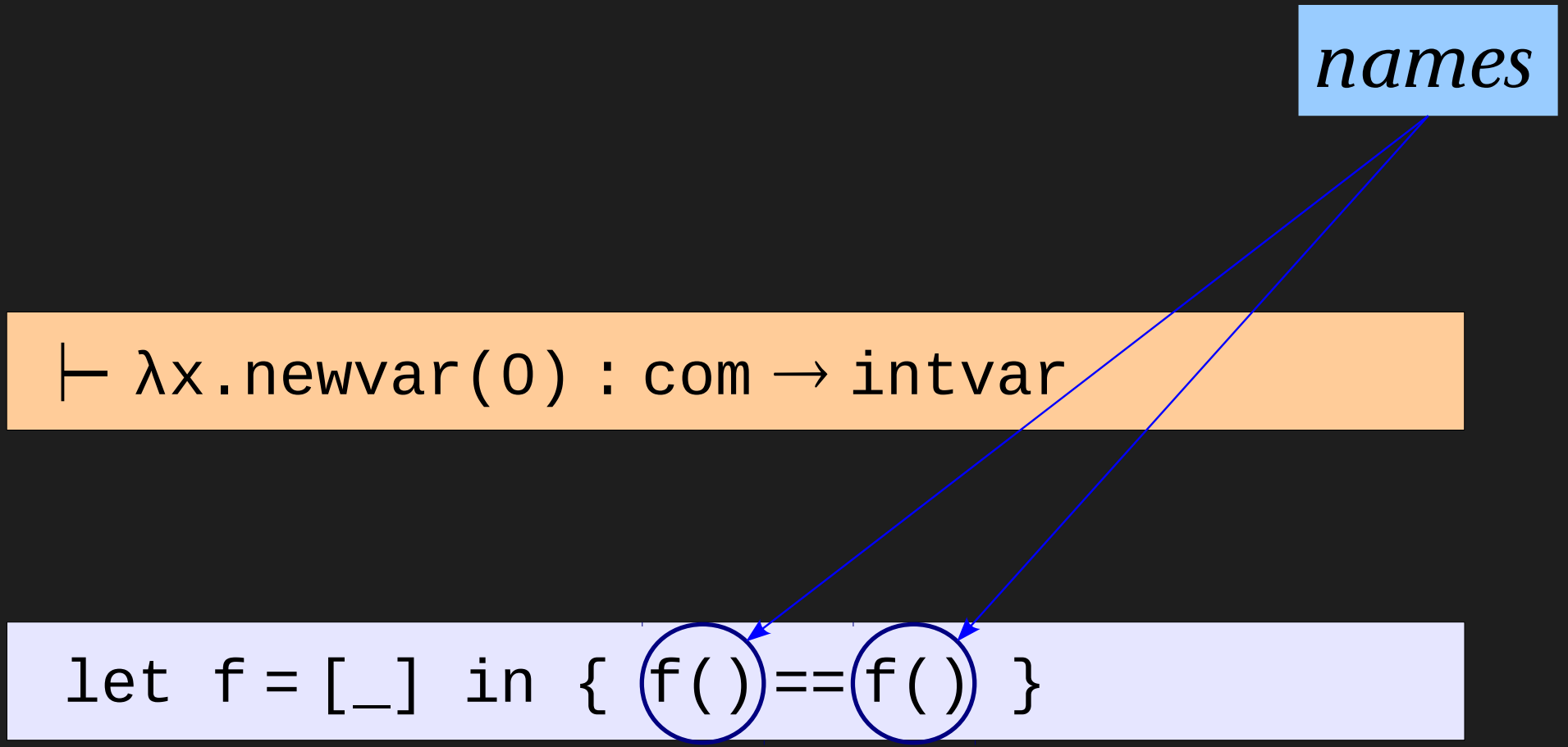
```
let f = [_] in { f() == f() }
```

Nominal games

$\vdash \lambda x.\texttt{newvar(0)} : \texttt{com} \to \texttt{intvar}$

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

Nominal games



Nominal games

Games in nominal sets

names

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$

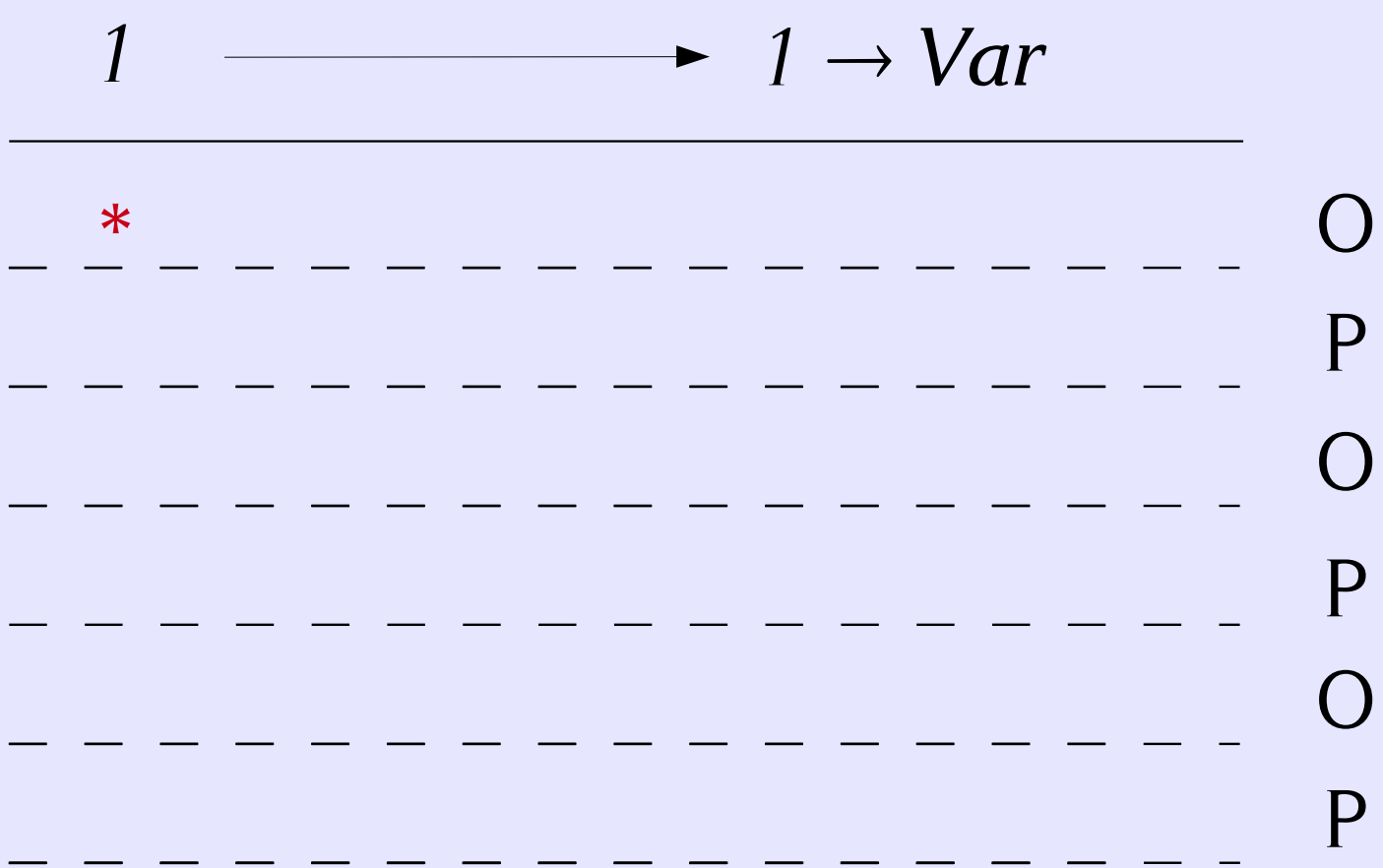
```
let f = [_] in { f() == f() }
```

Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$

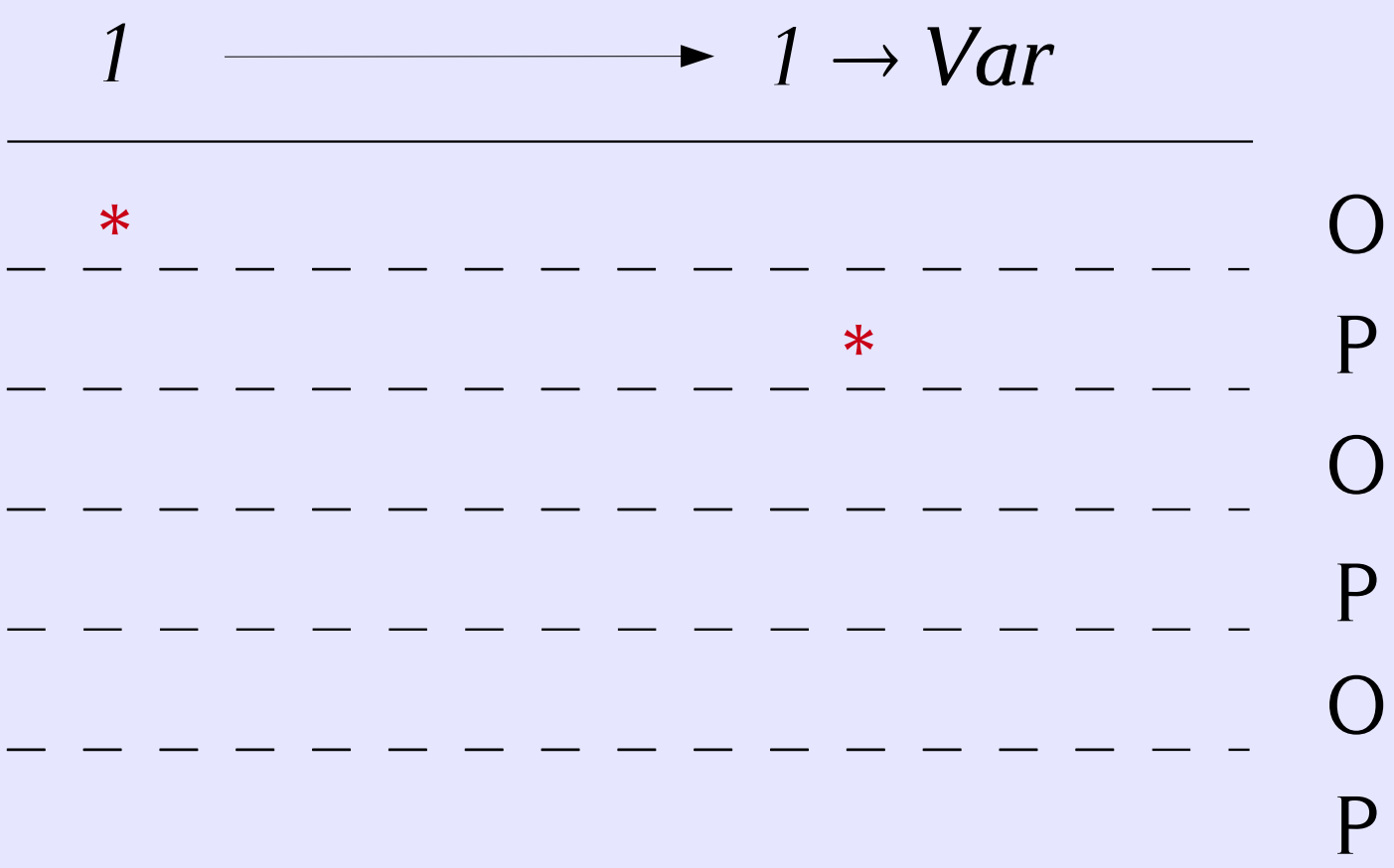
Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$



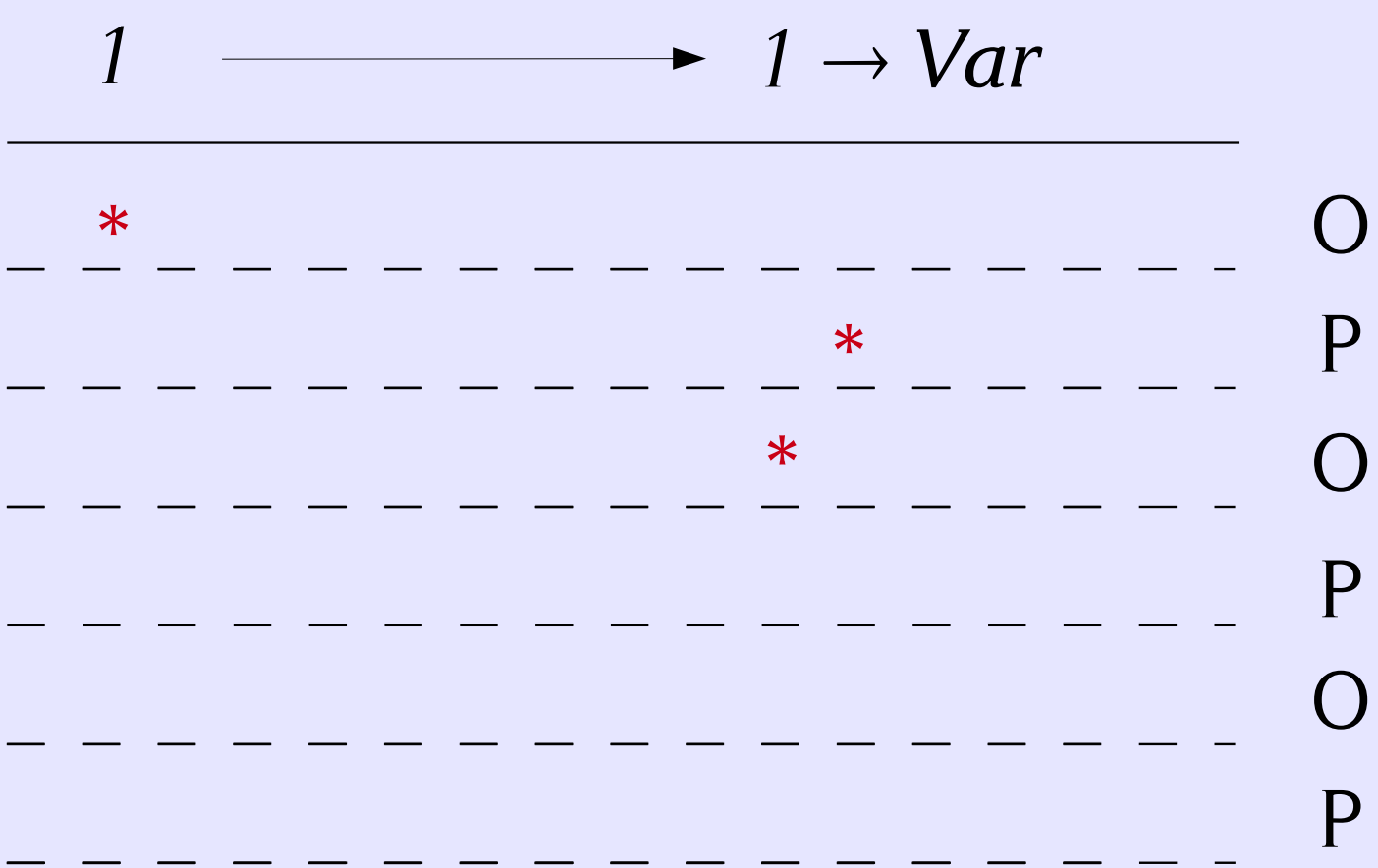
Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$



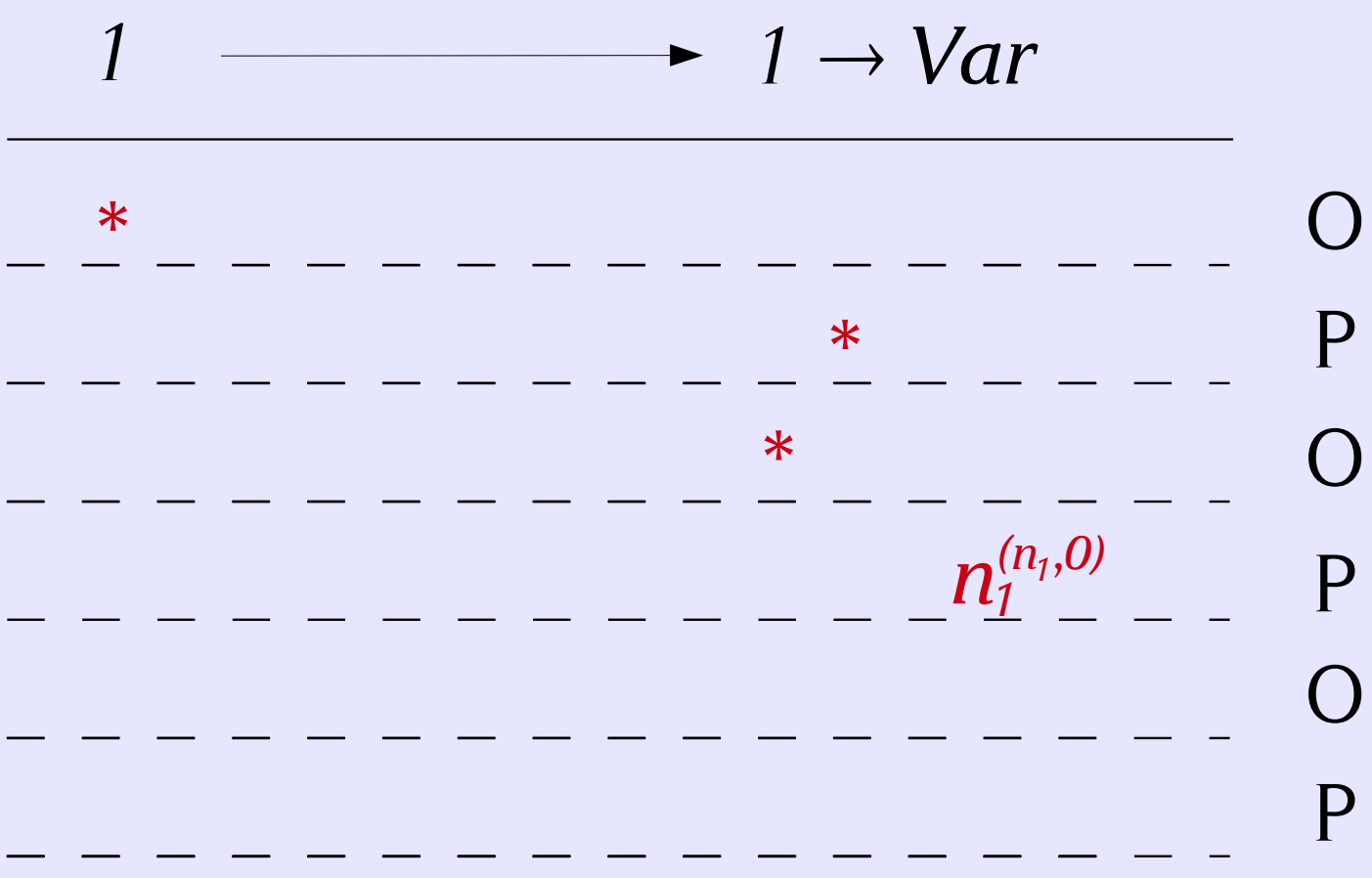
Examples

$\vdash$ `λx.newvar(0) : com → intvar`



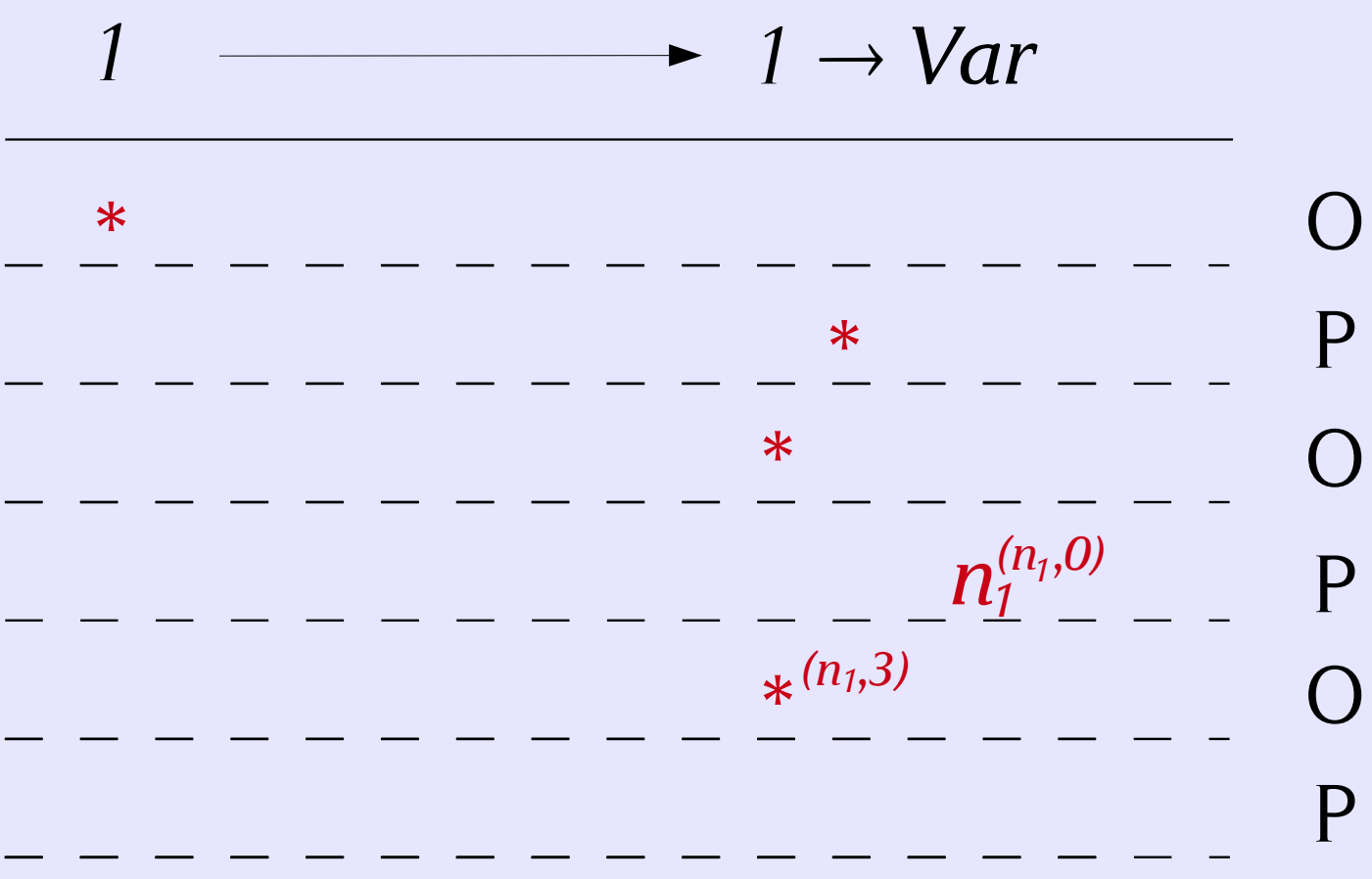
Examples

$\vdash$ `λx.newvar(0) : com → intvar`



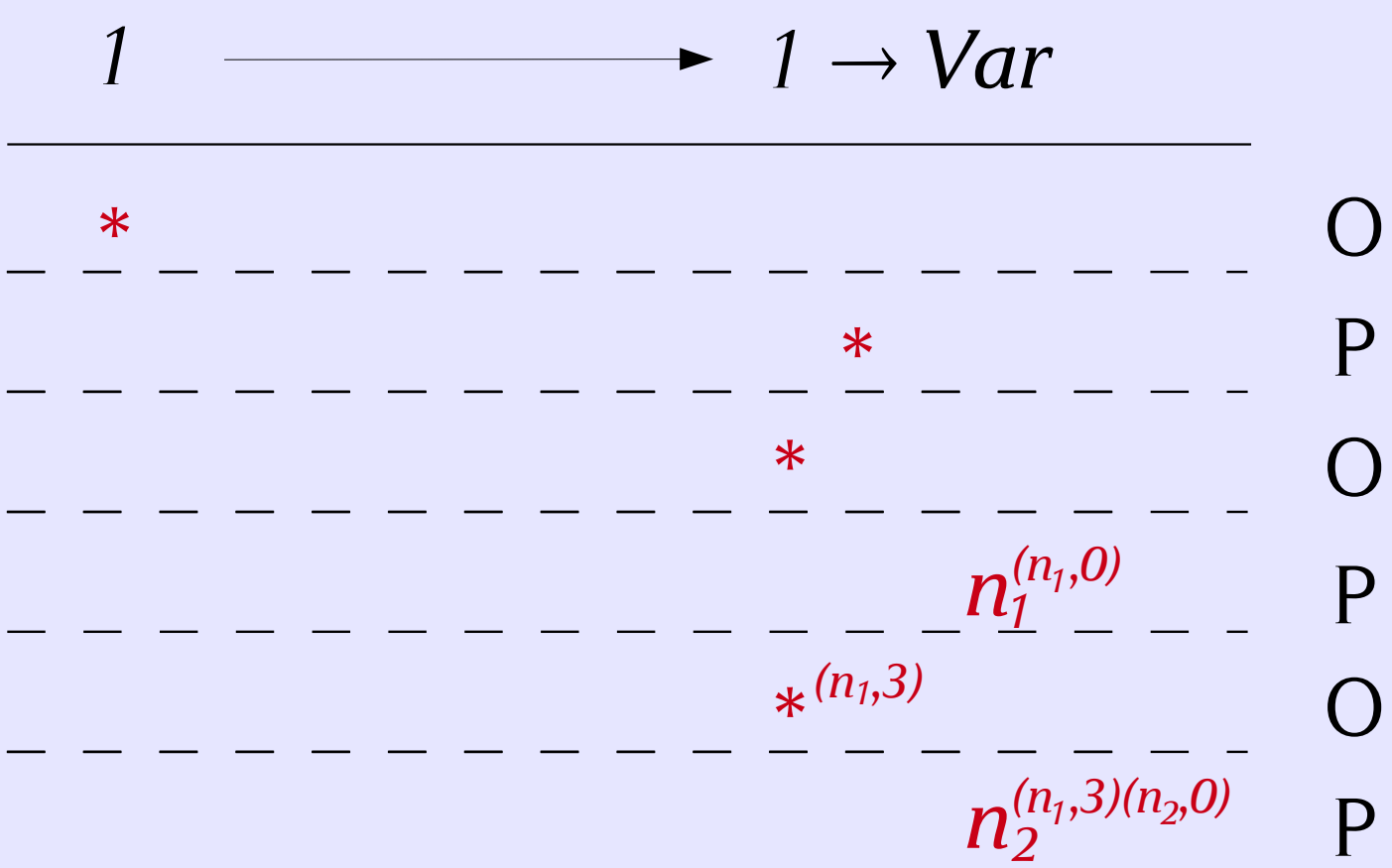
Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$



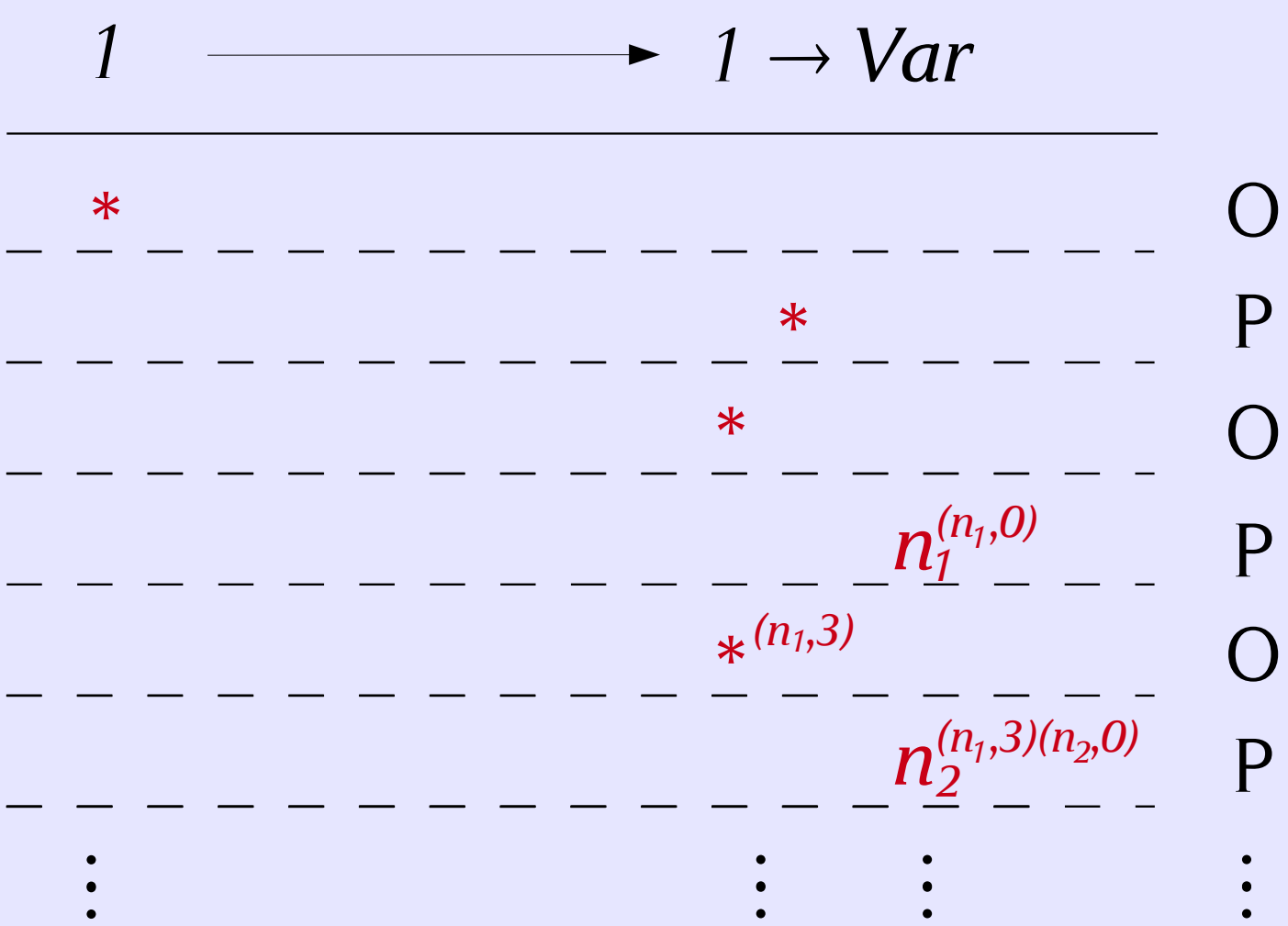
Examples

$\vdash$ `λx.newvar(0) : com → intvar`



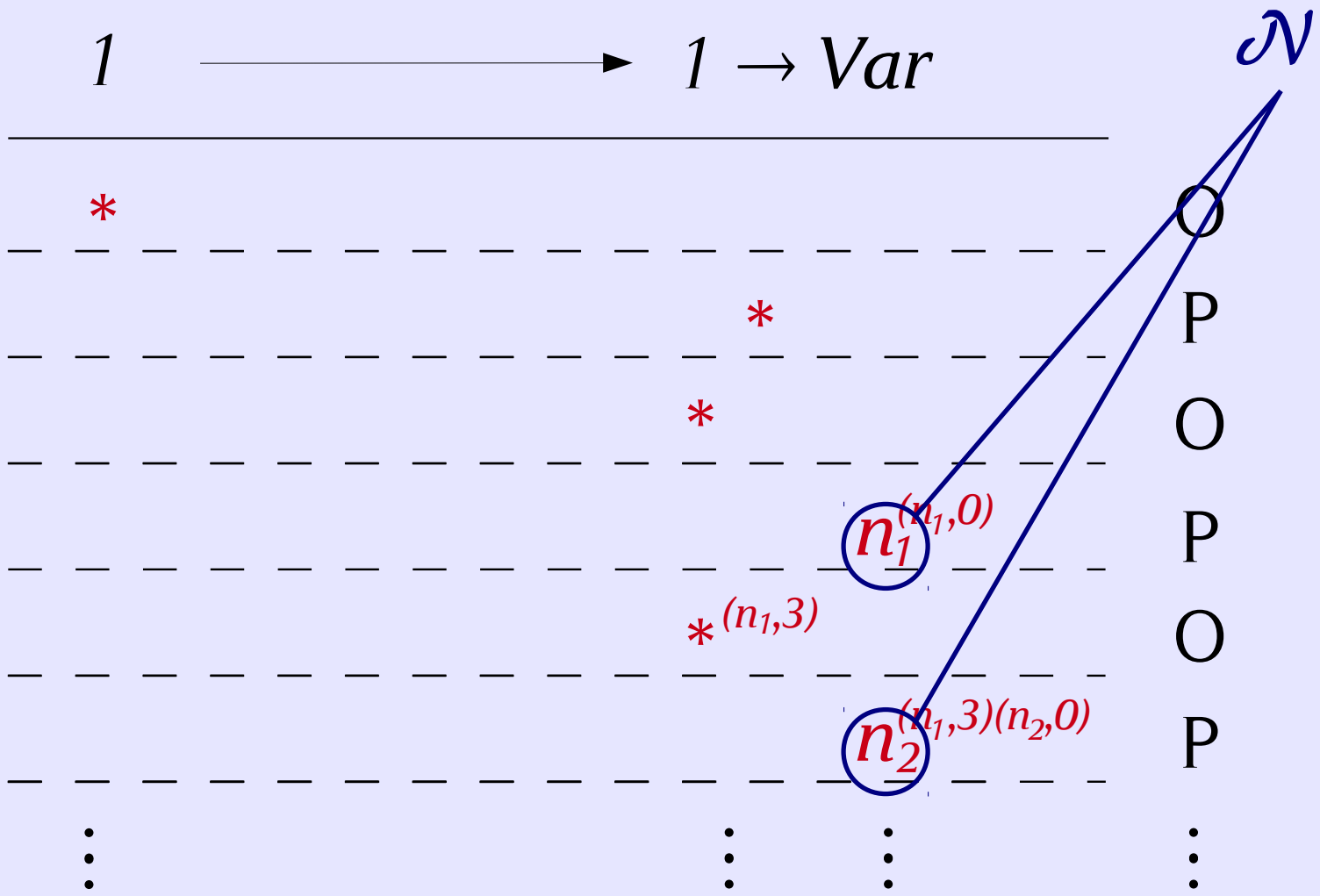
Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$



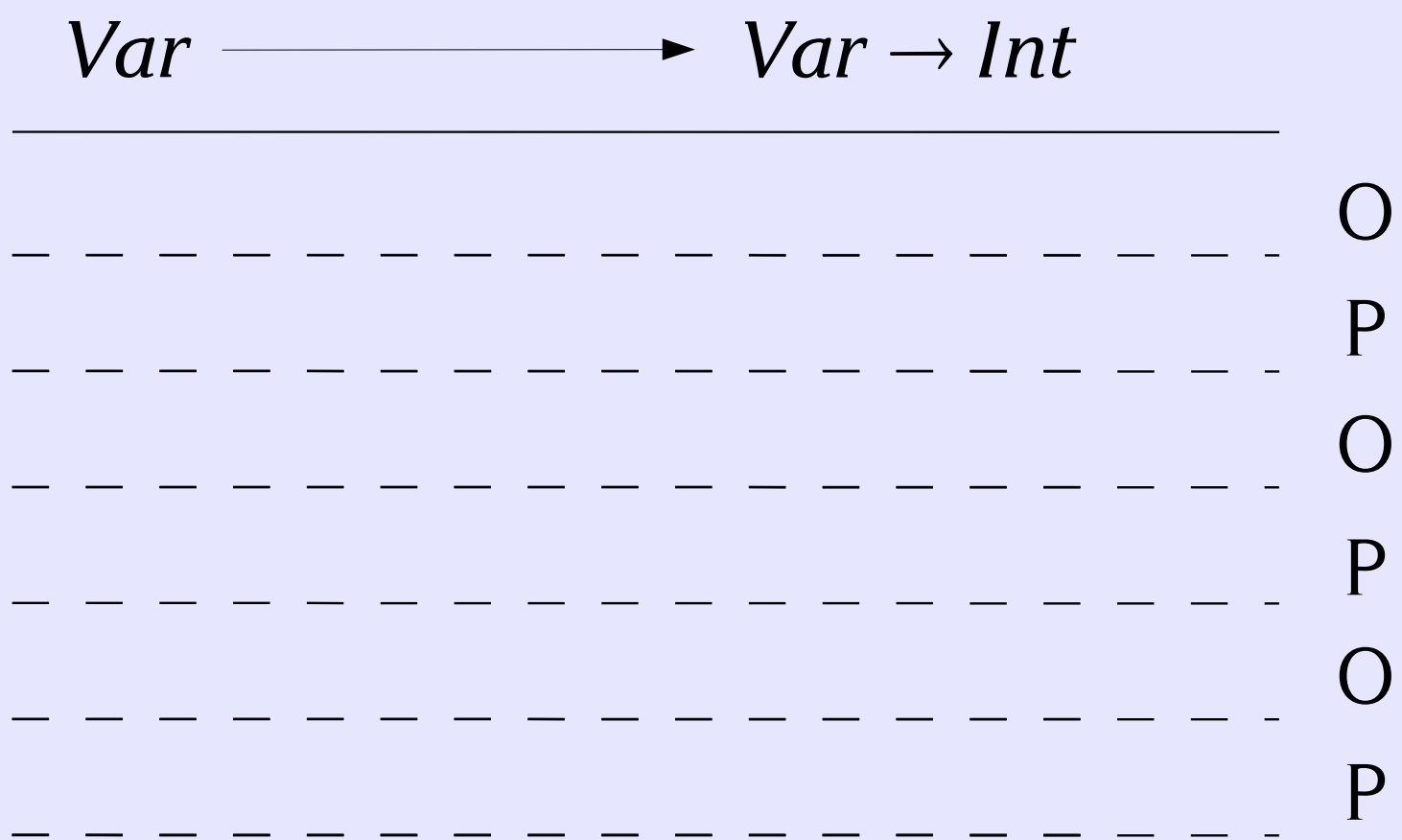
Examples

$\vdash \lambda x.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$



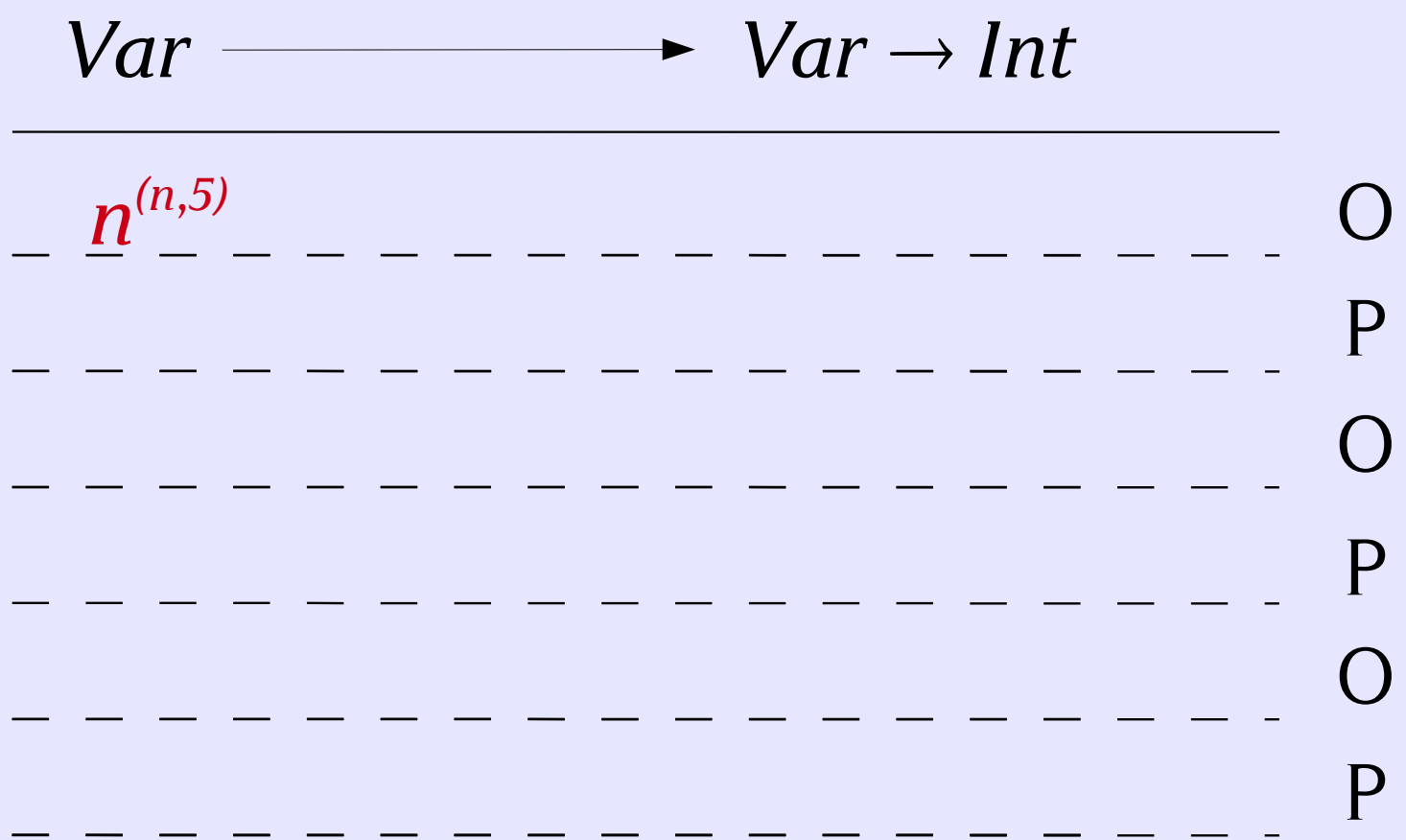
Examples

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



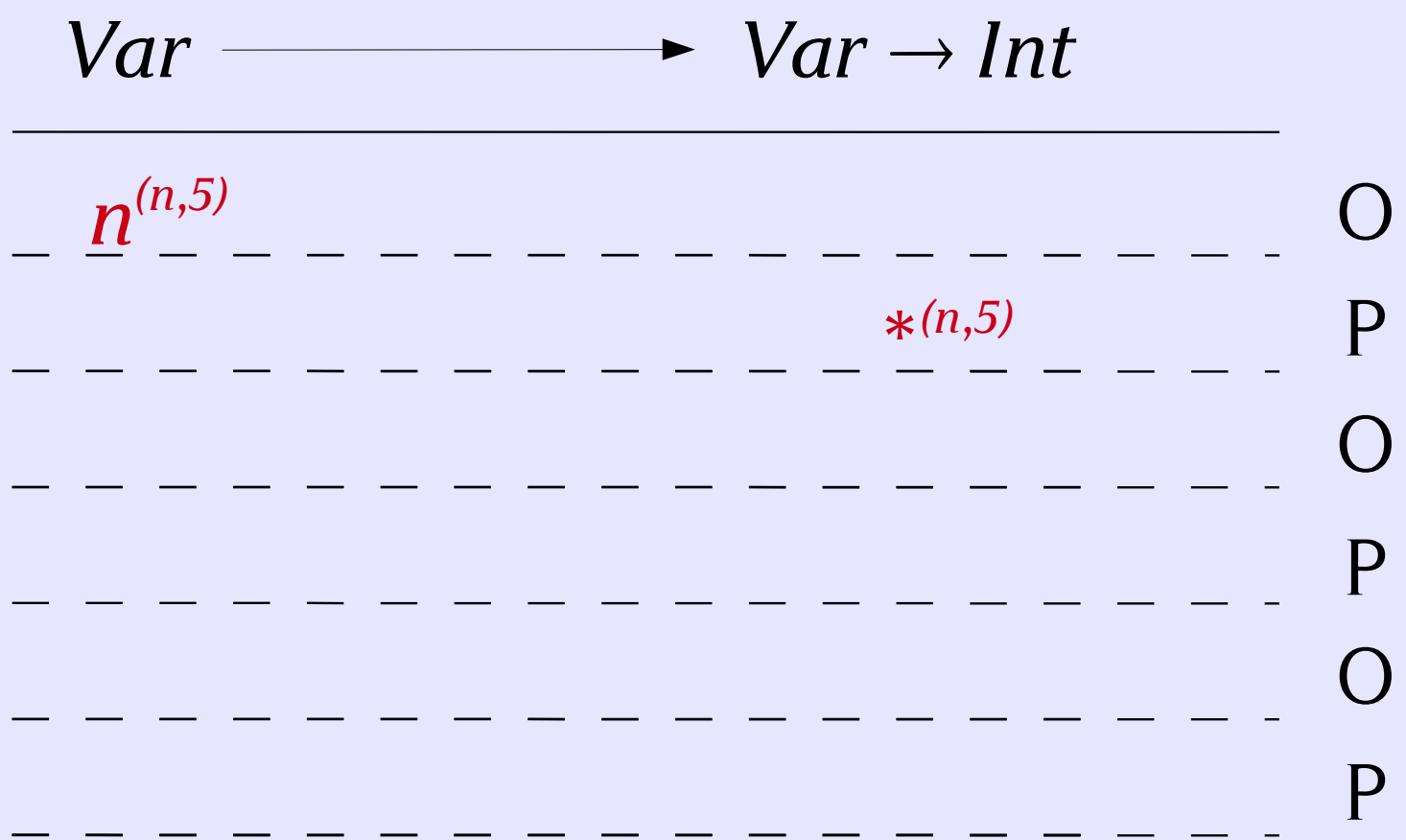
Examples

`x : intvar ⊢ λy.(x == y) : intvar → int`



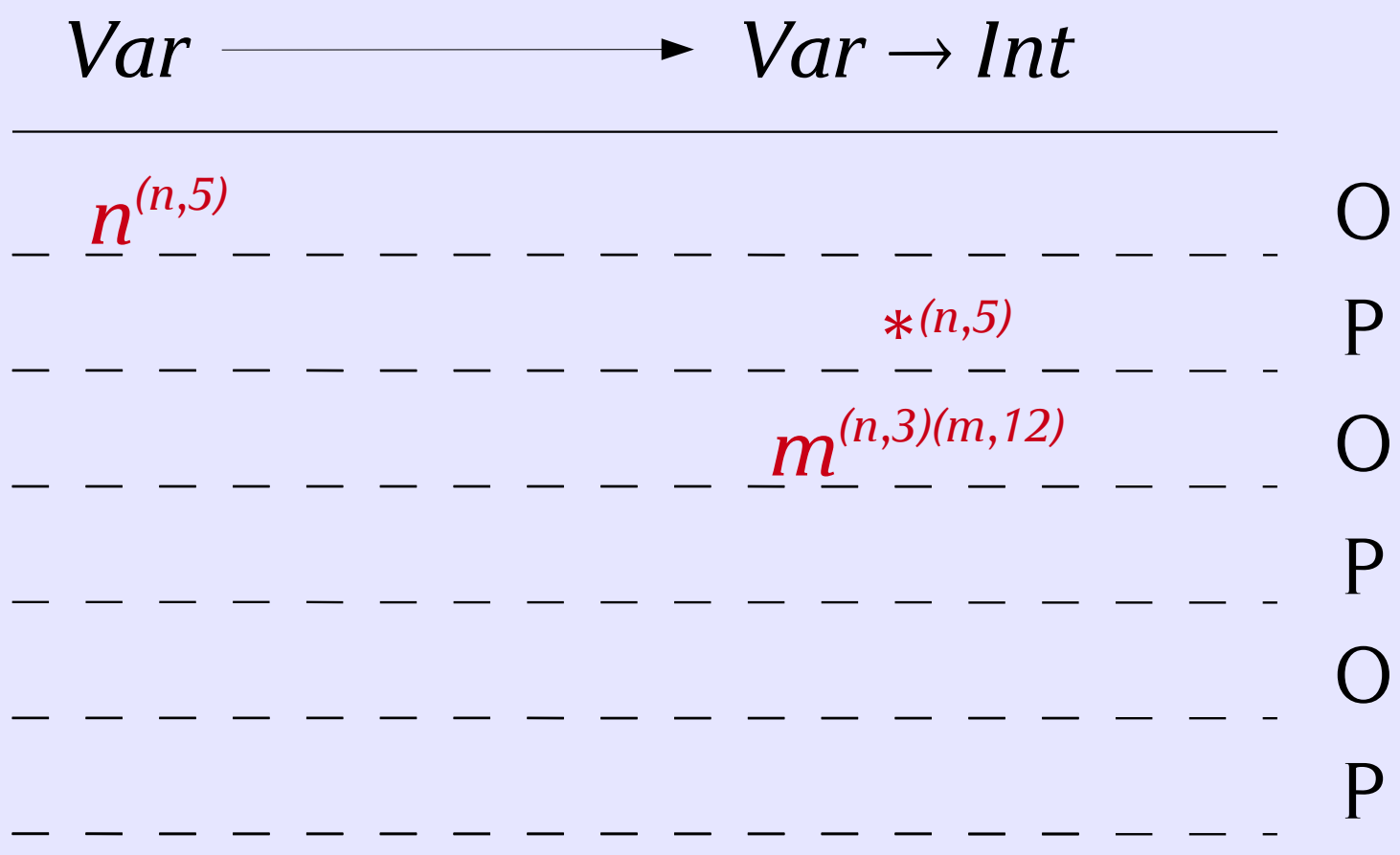
Examples

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



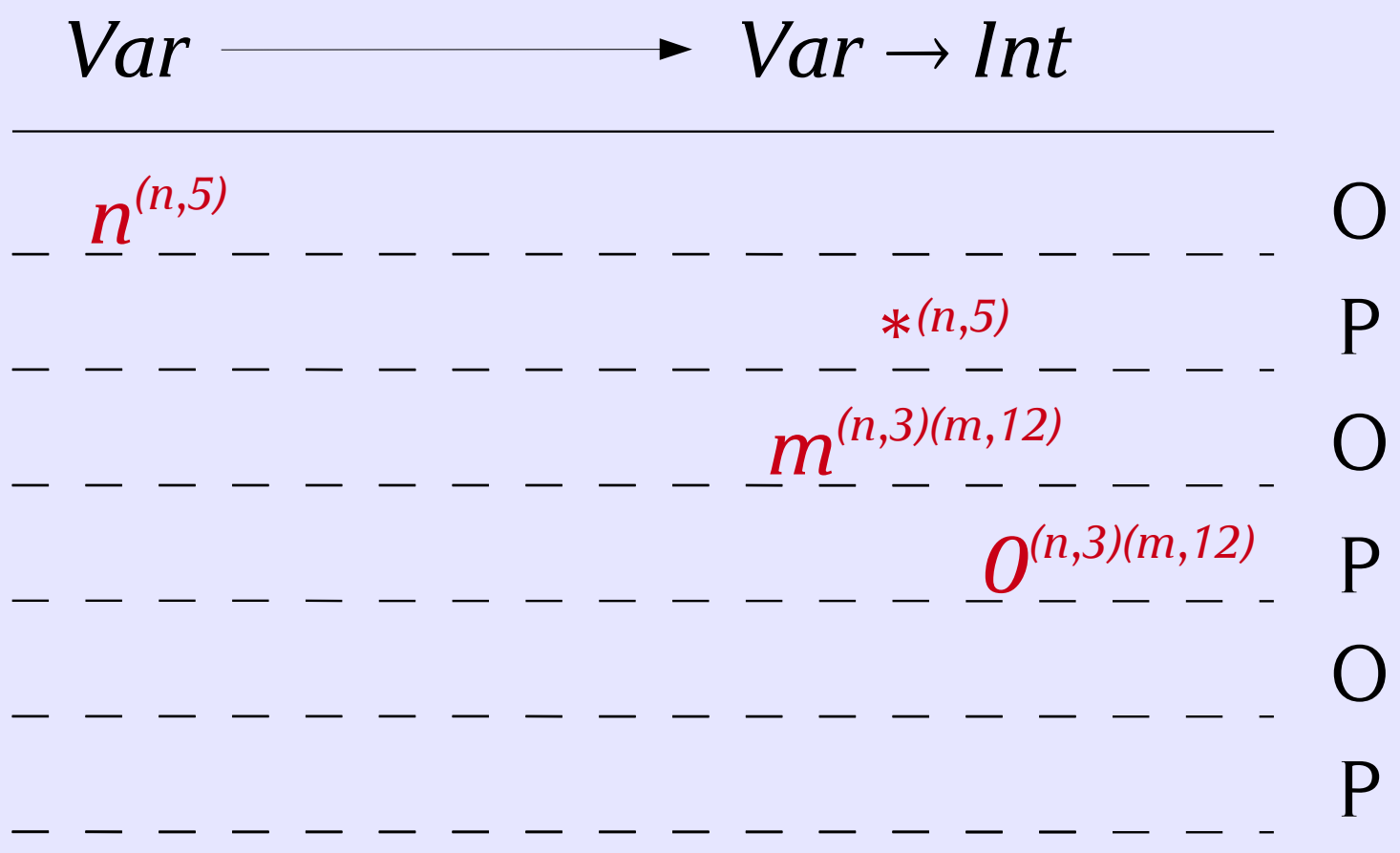
Examples

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



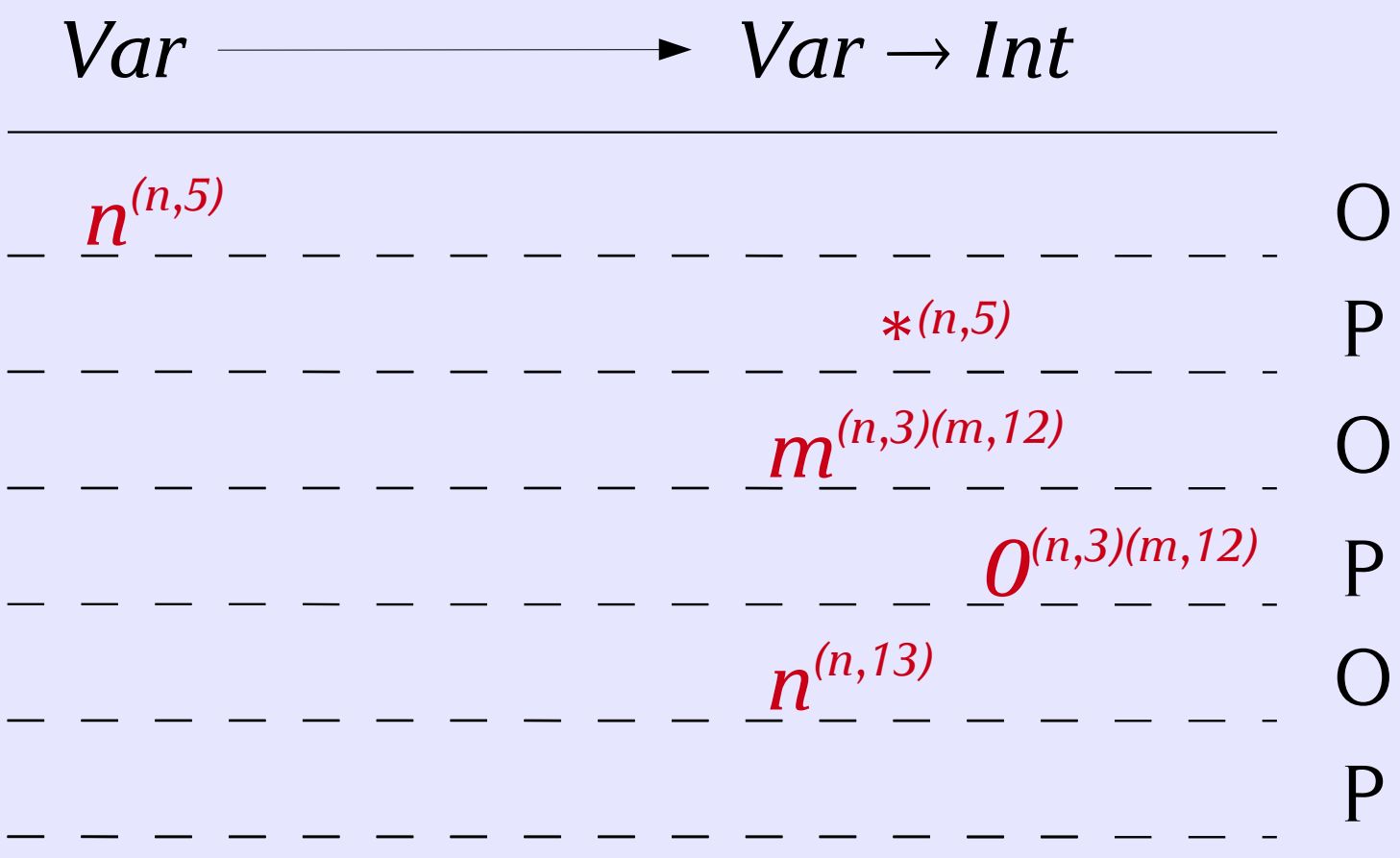
Examples

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



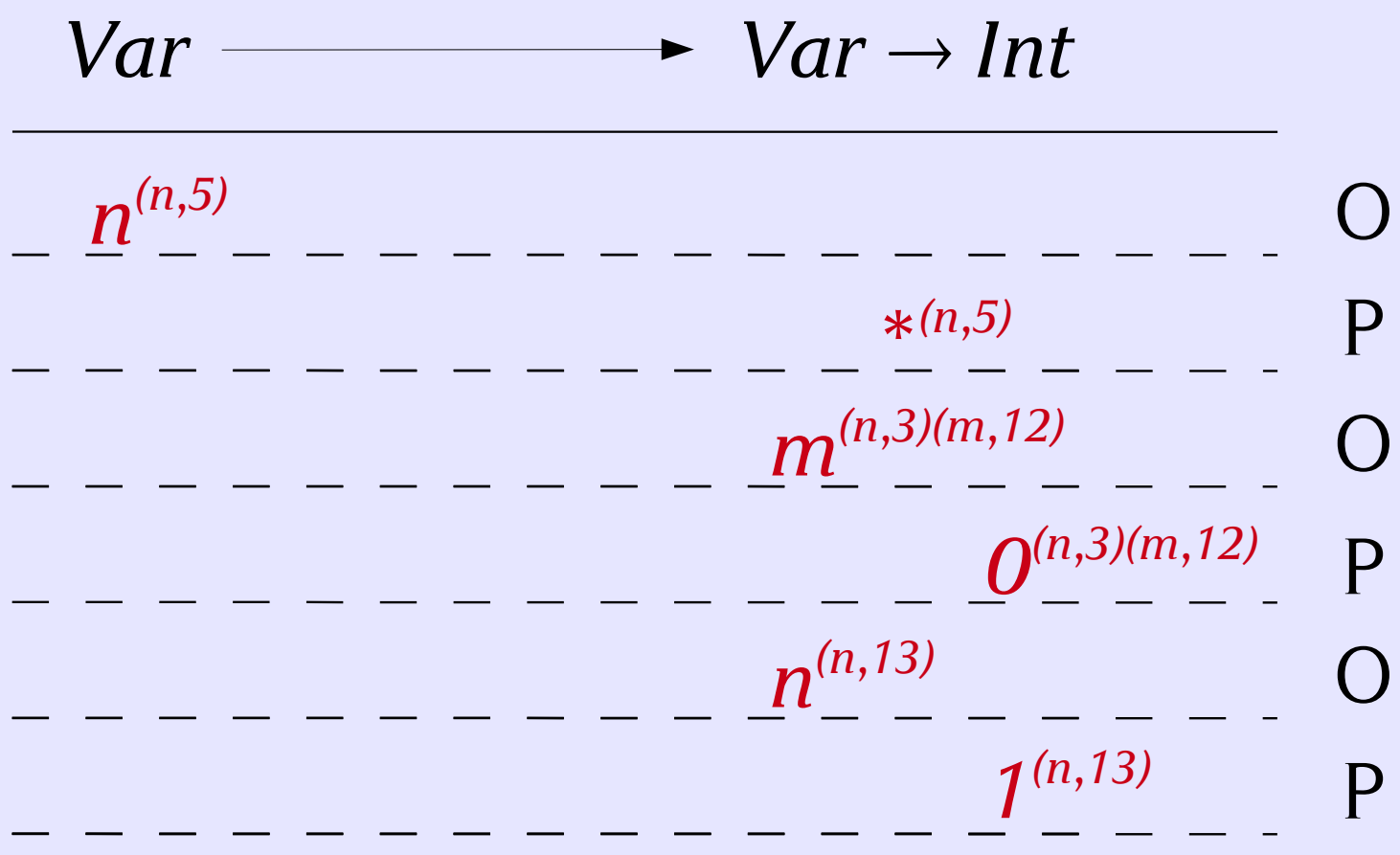
Examples

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



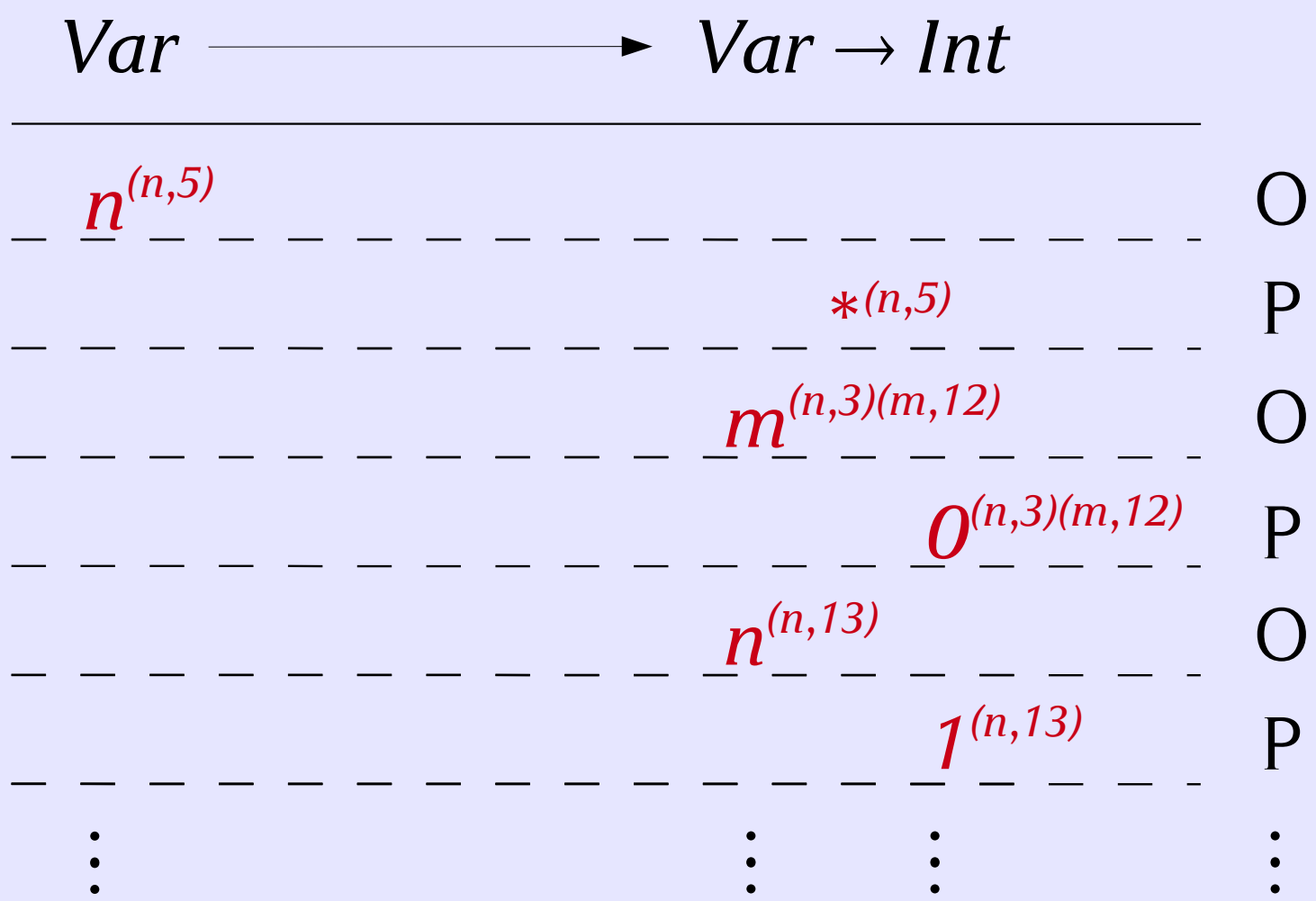
Examples

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



Examples

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



Games with new resources

Models of realistic programs

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]

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

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

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]

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

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

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

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

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

Games with new resources

Models of realistic programs

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

Algorithmic games

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

Fresh-register automata [*POPL'11*]

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

Fresh-register automata [*POPL'11*]

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

Automata with names

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

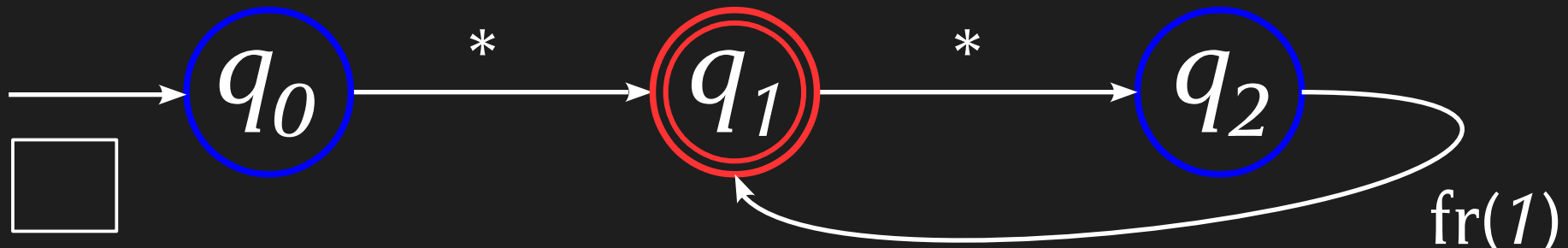
Fresh-register automata [*POPL'11*]

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

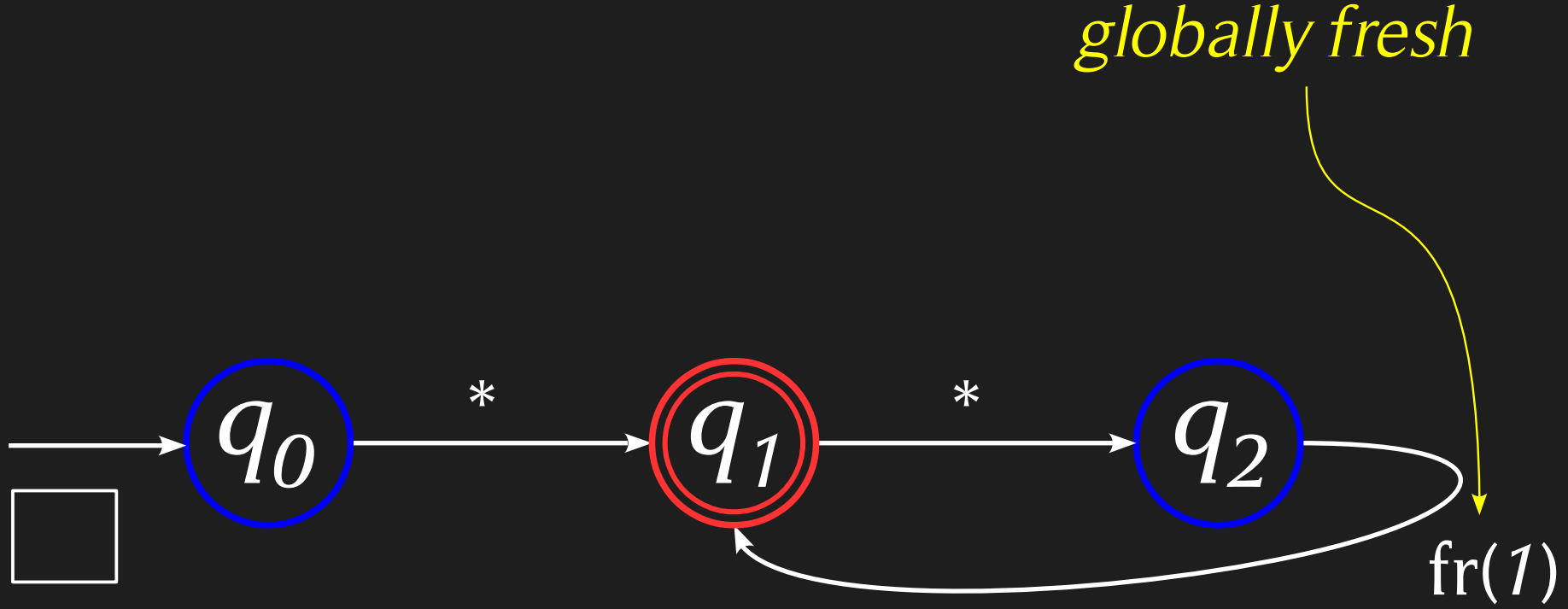
Automata with names

- Finite-state machines with registers
- Infinite alphabet $\mathcal{N}$
- *Freshness* recognition

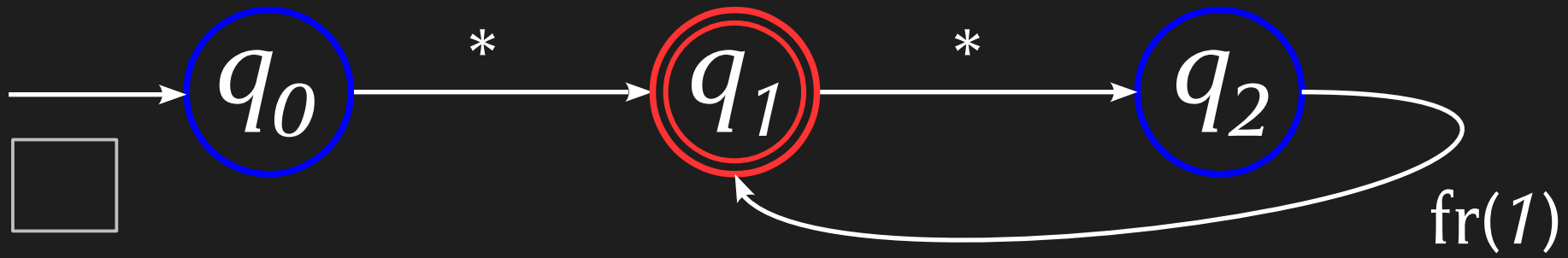
Fresh-register automata



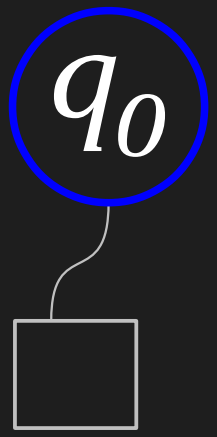
Fresh-register automata

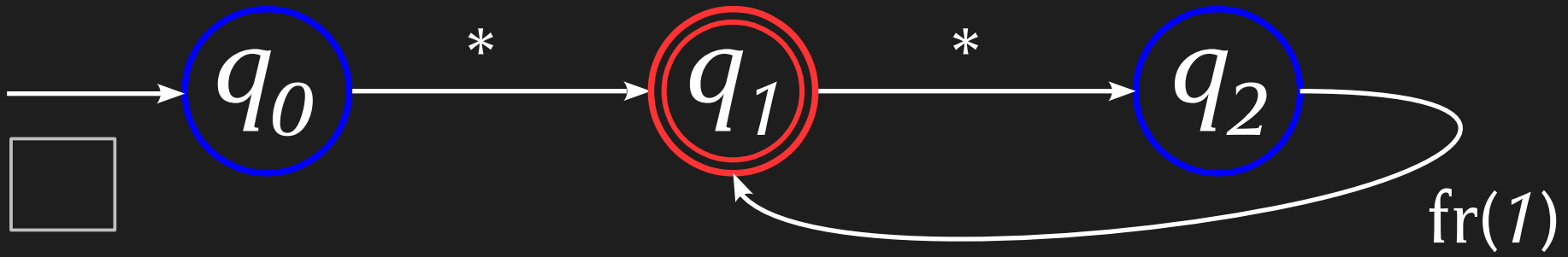


Fresh-register automata

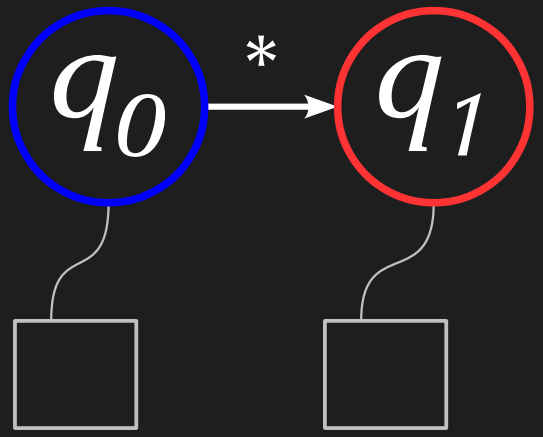


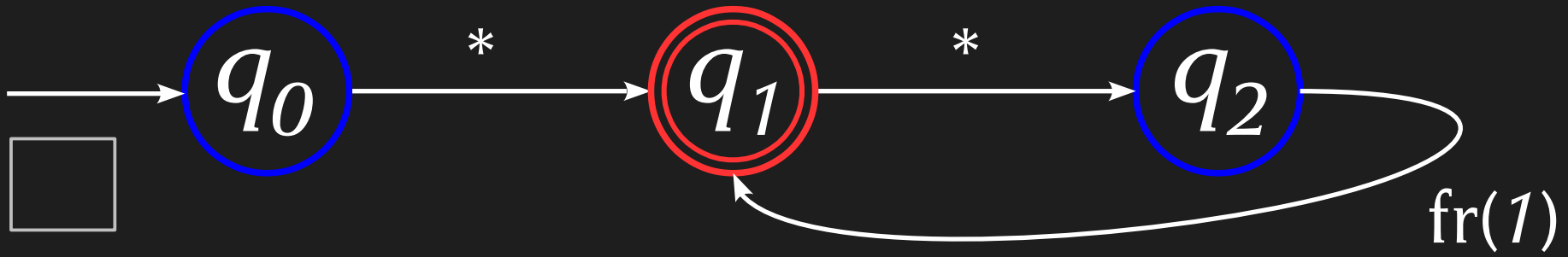
Fresh-register automata



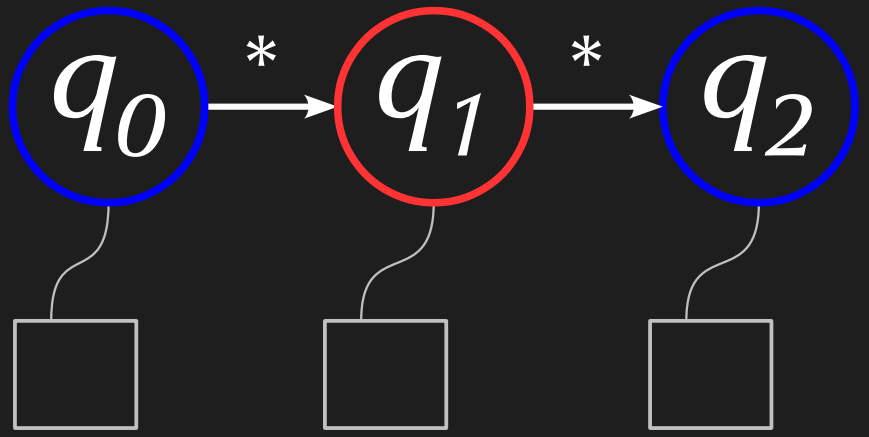


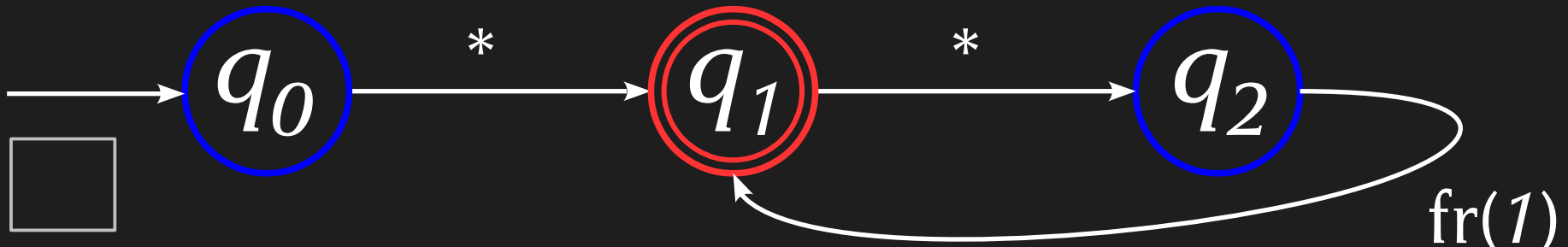
Fresh-register automata



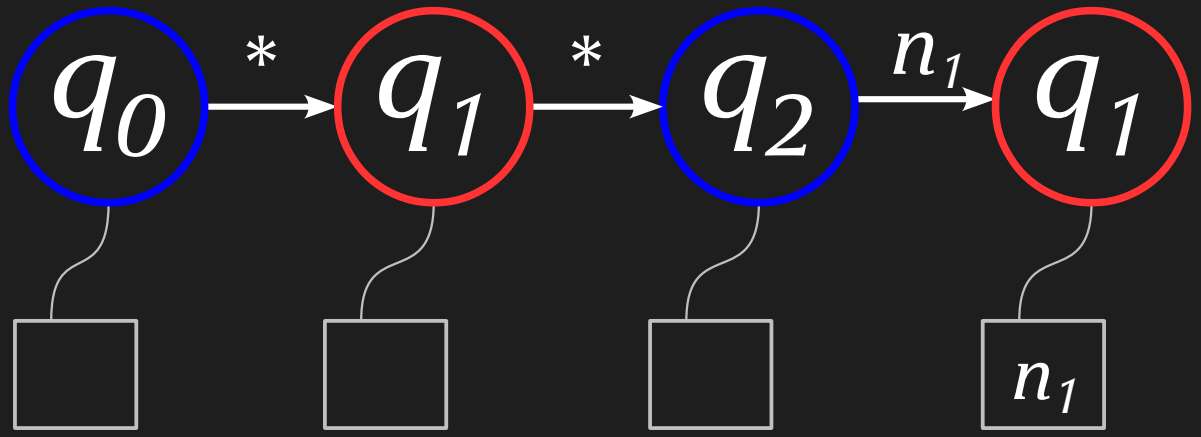


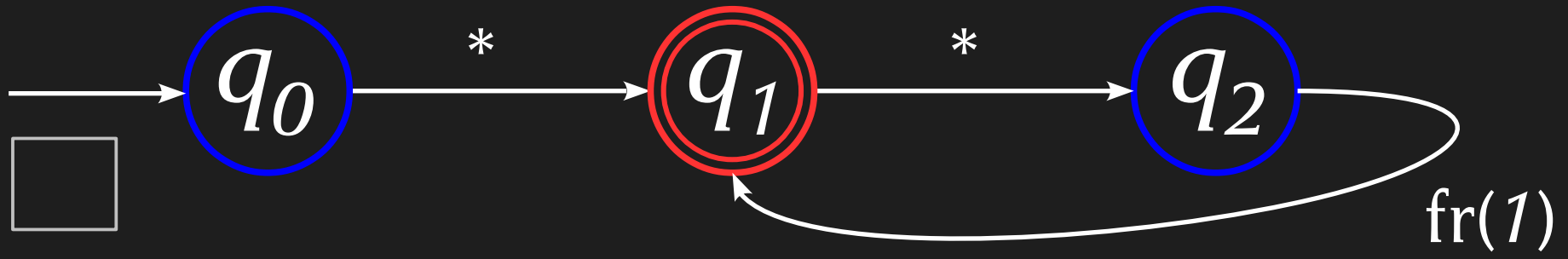
Fresh-register automata



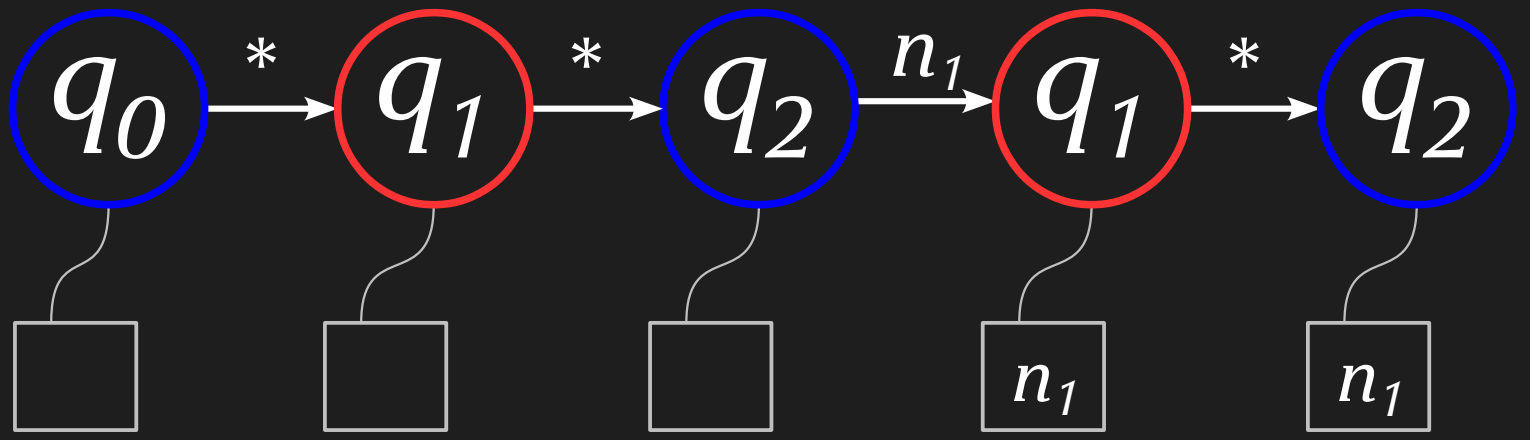


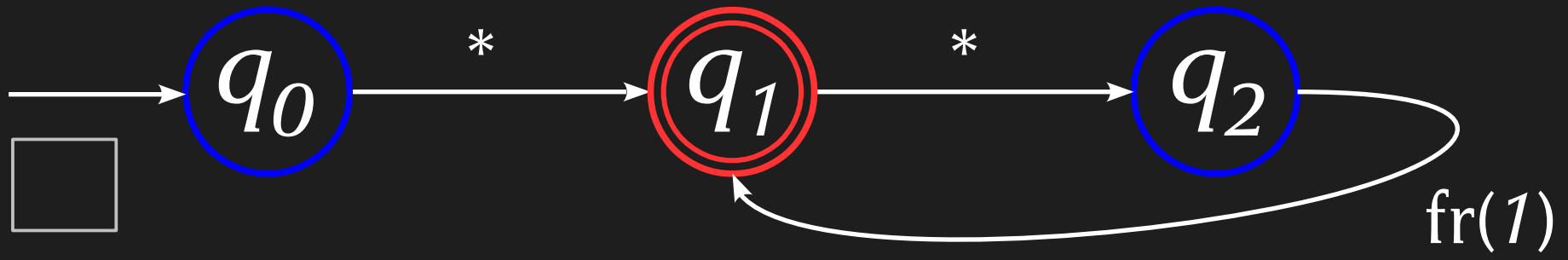
Fresh-register automata



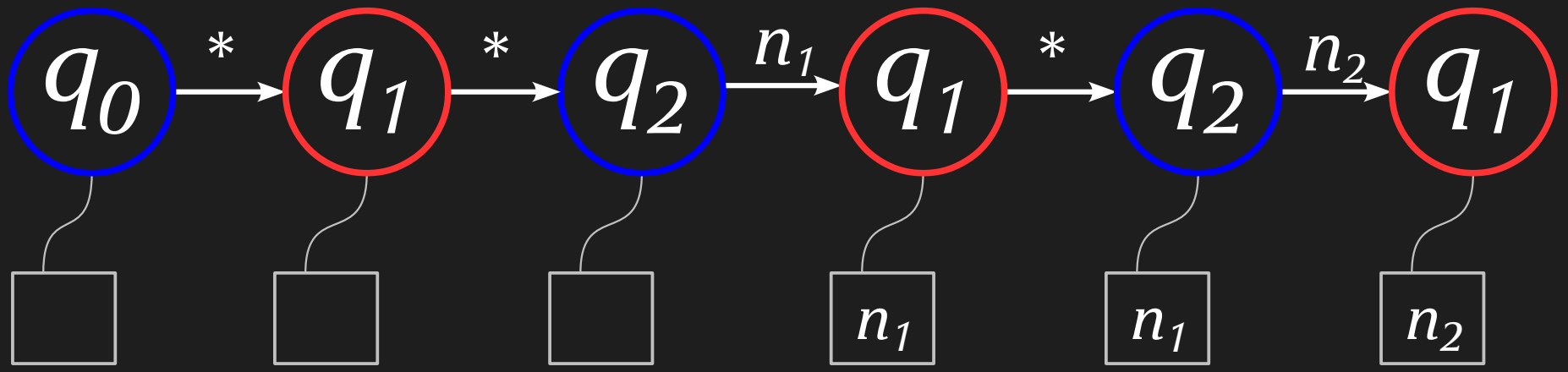


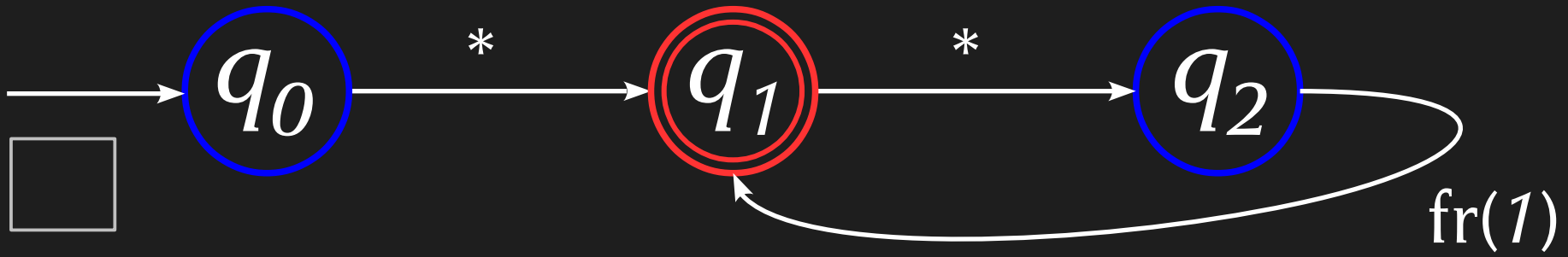
Fresh-register automata



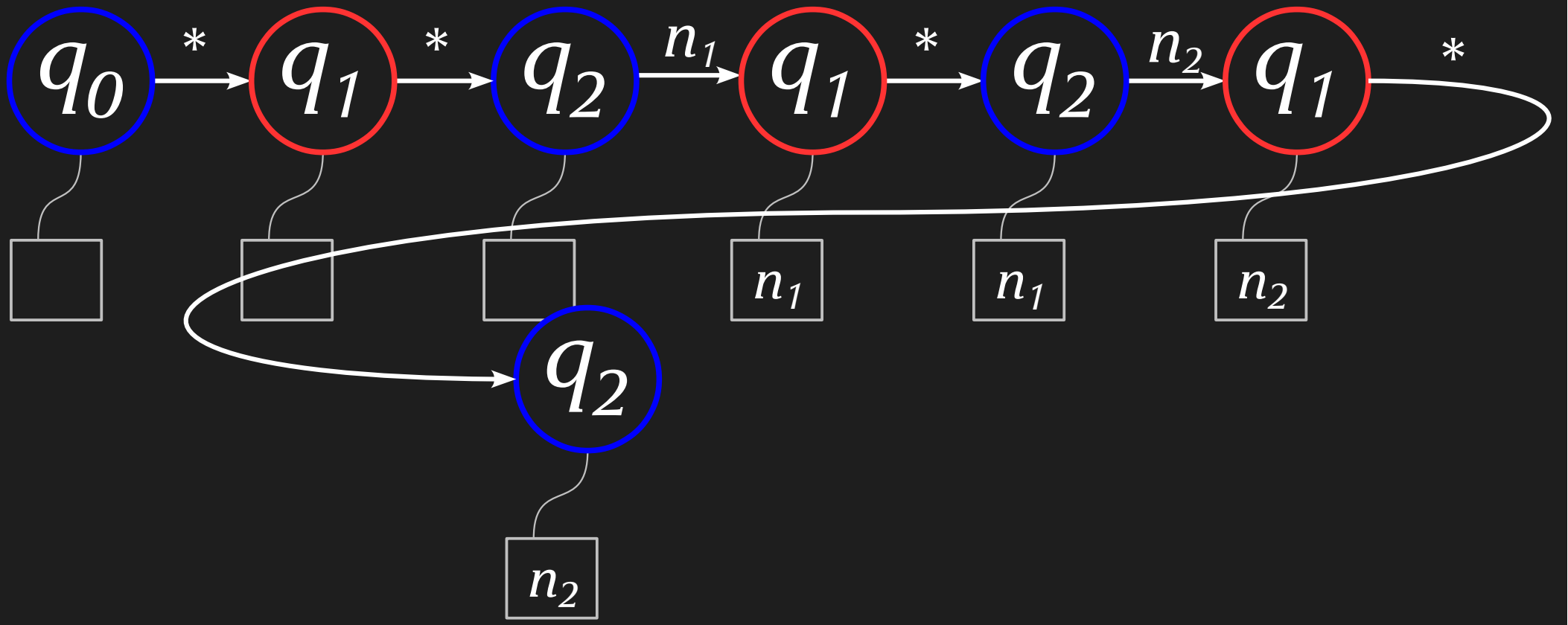


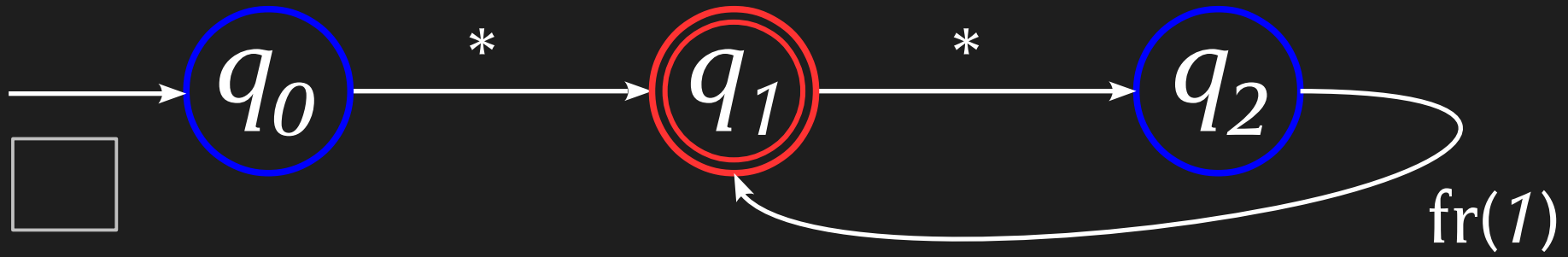
Fresh-register automata



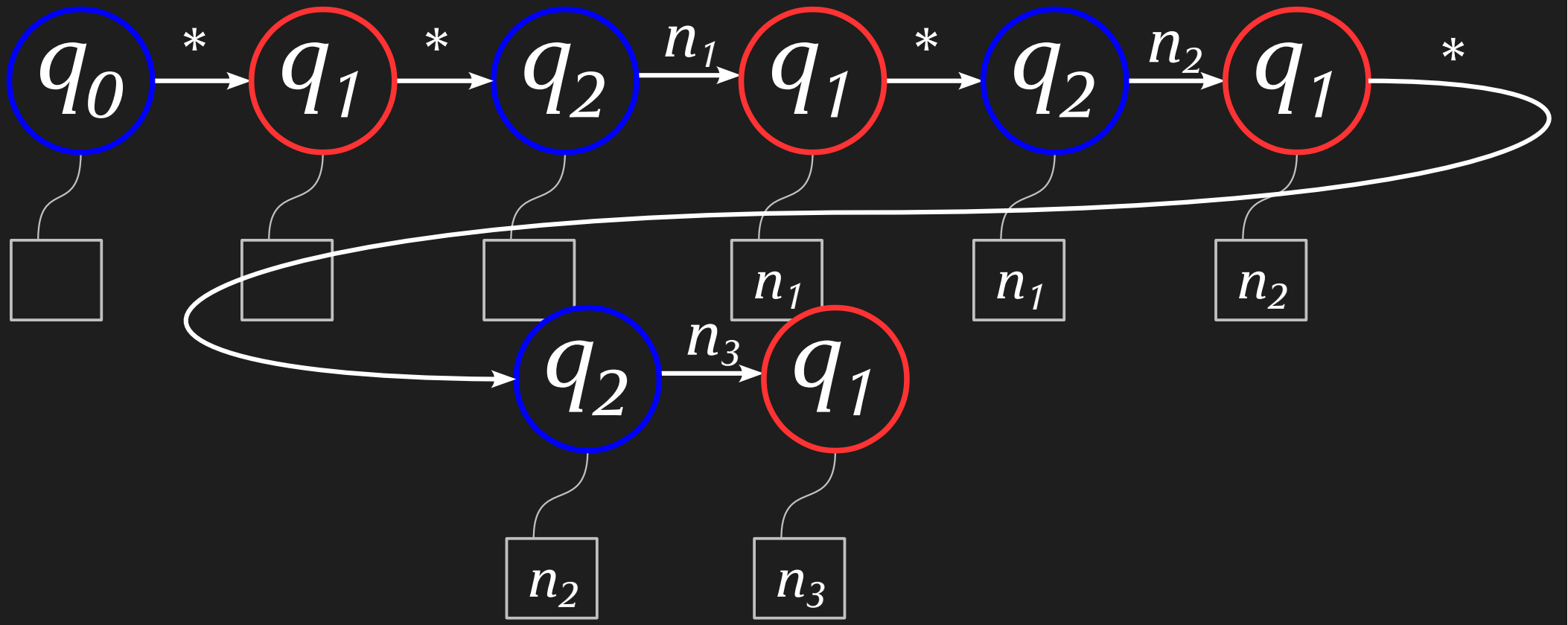


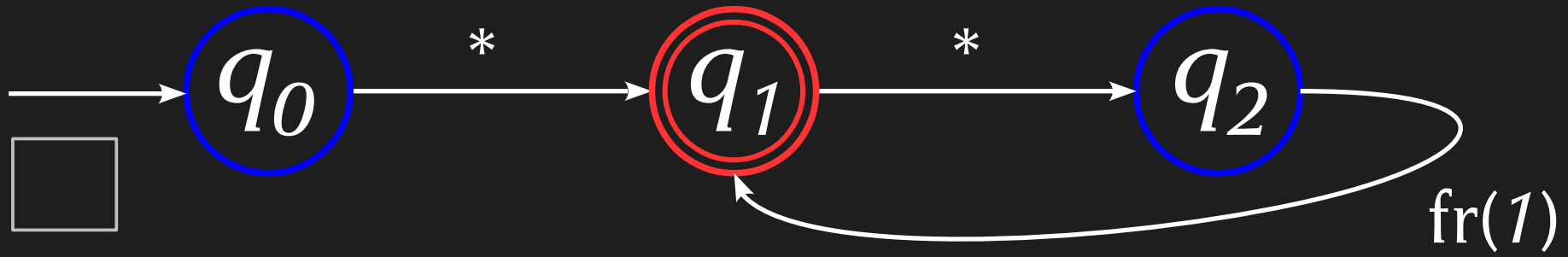
Fresh-register automata



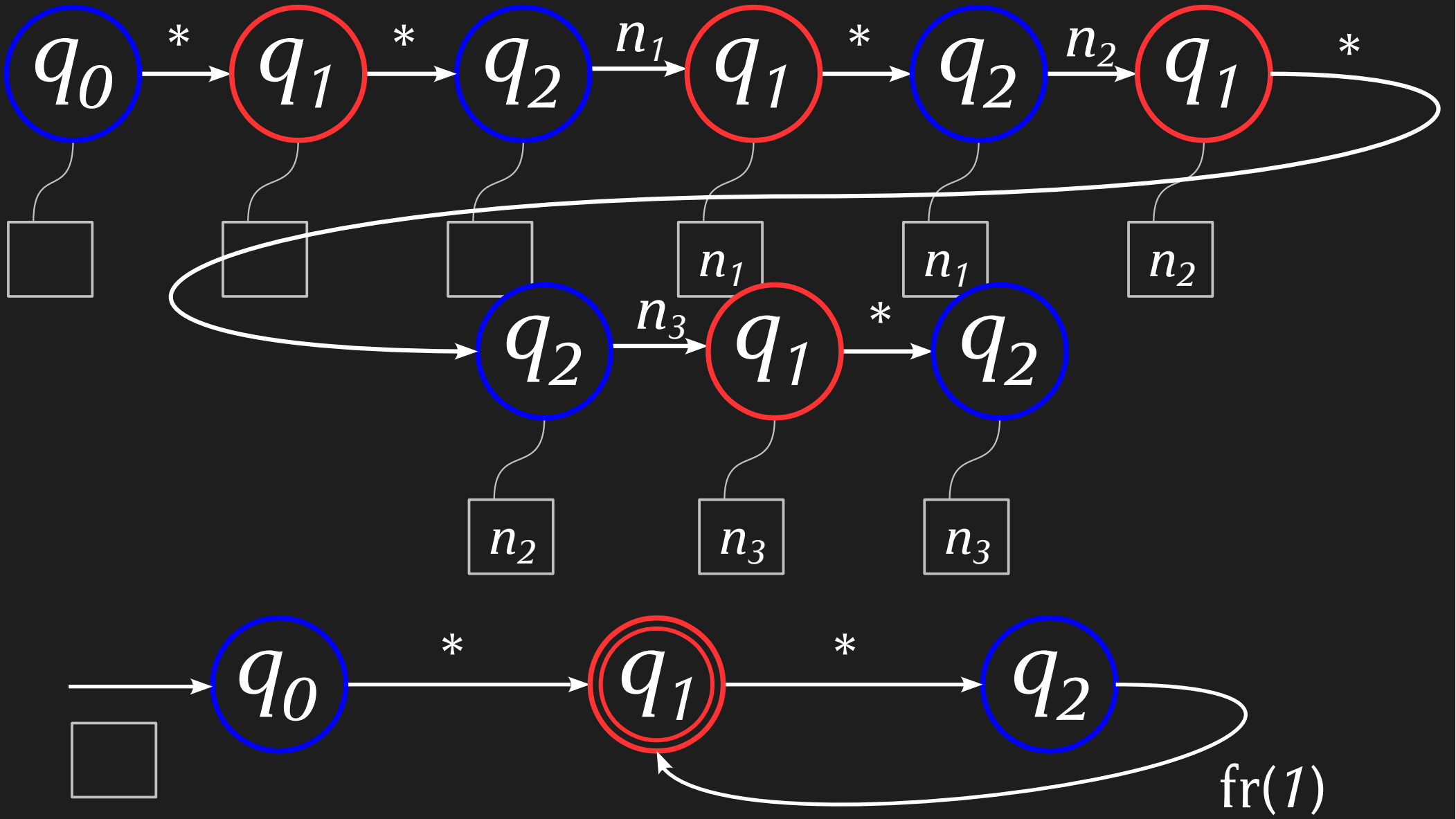


Fresh-register automata

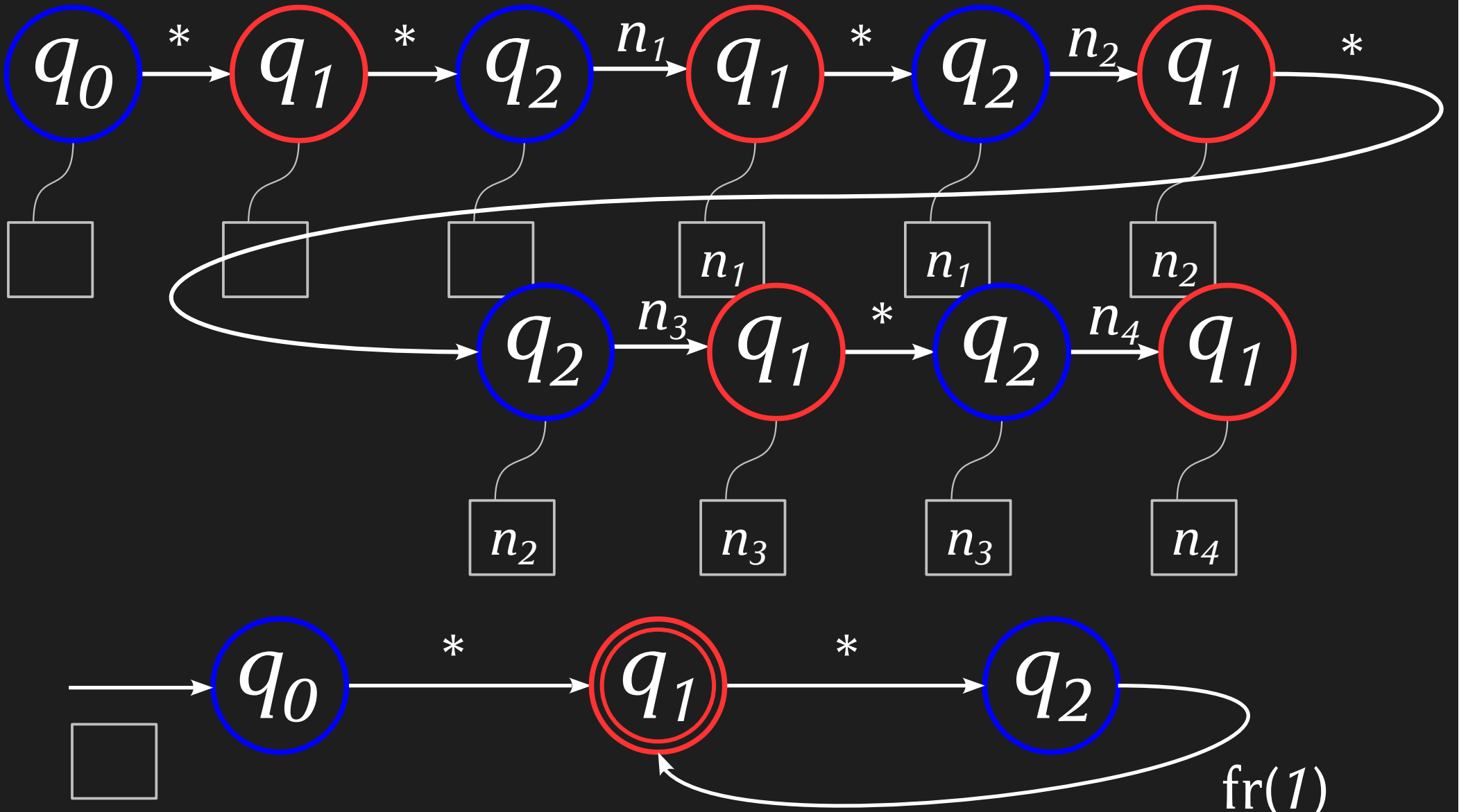




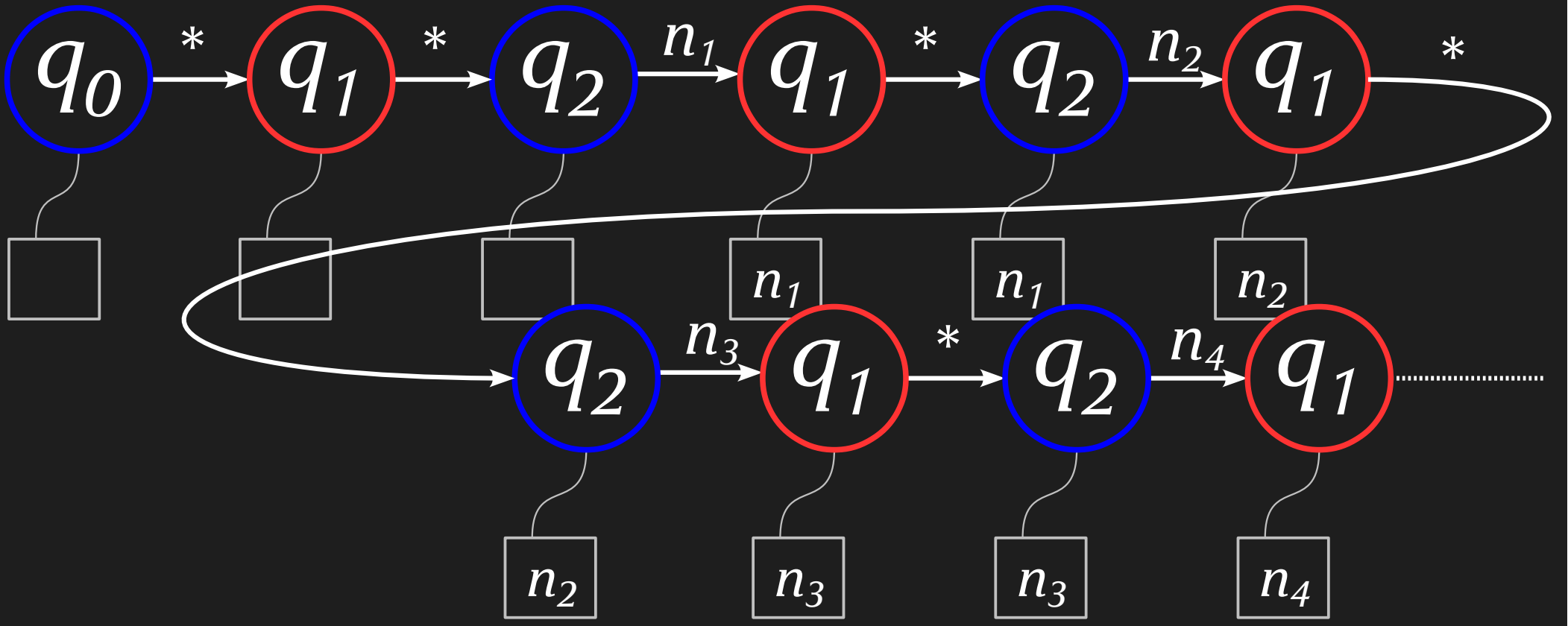
Fresh-register automata

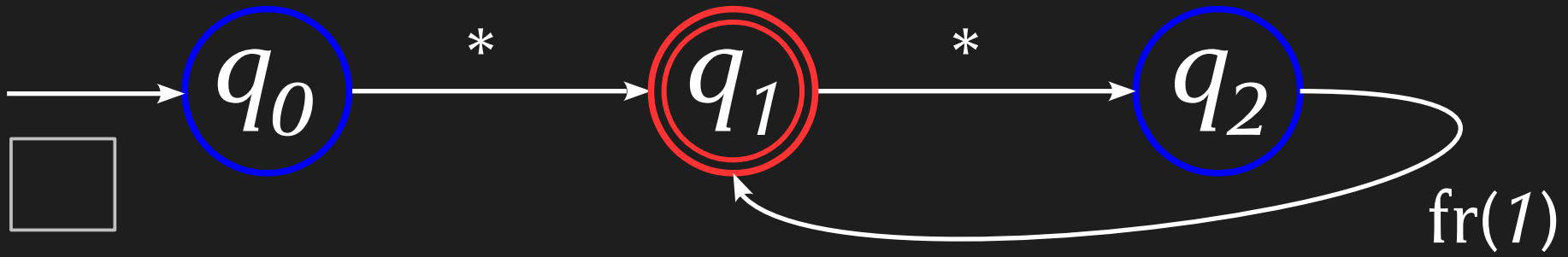


Fresh-register automata

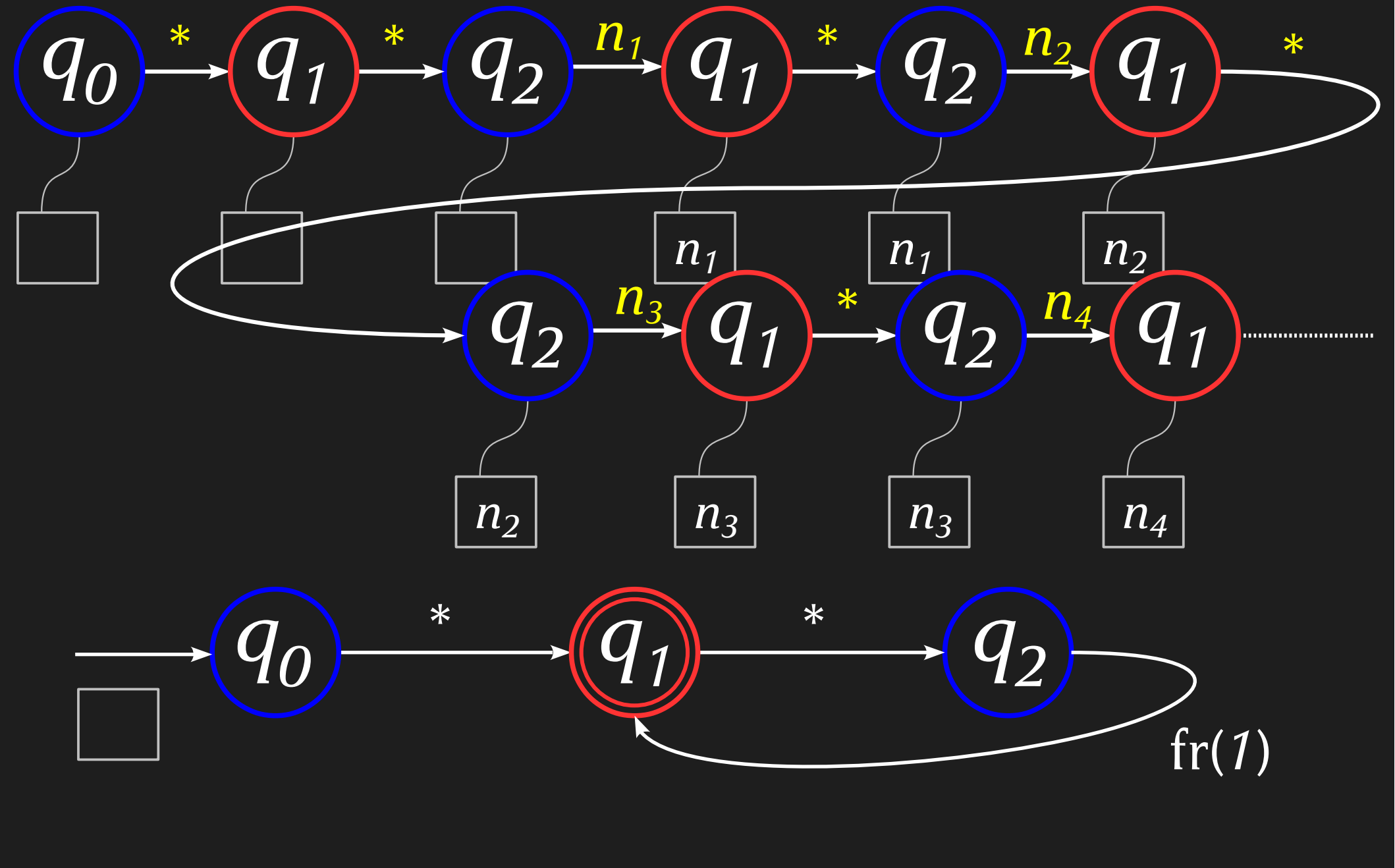


Fresh-register automata

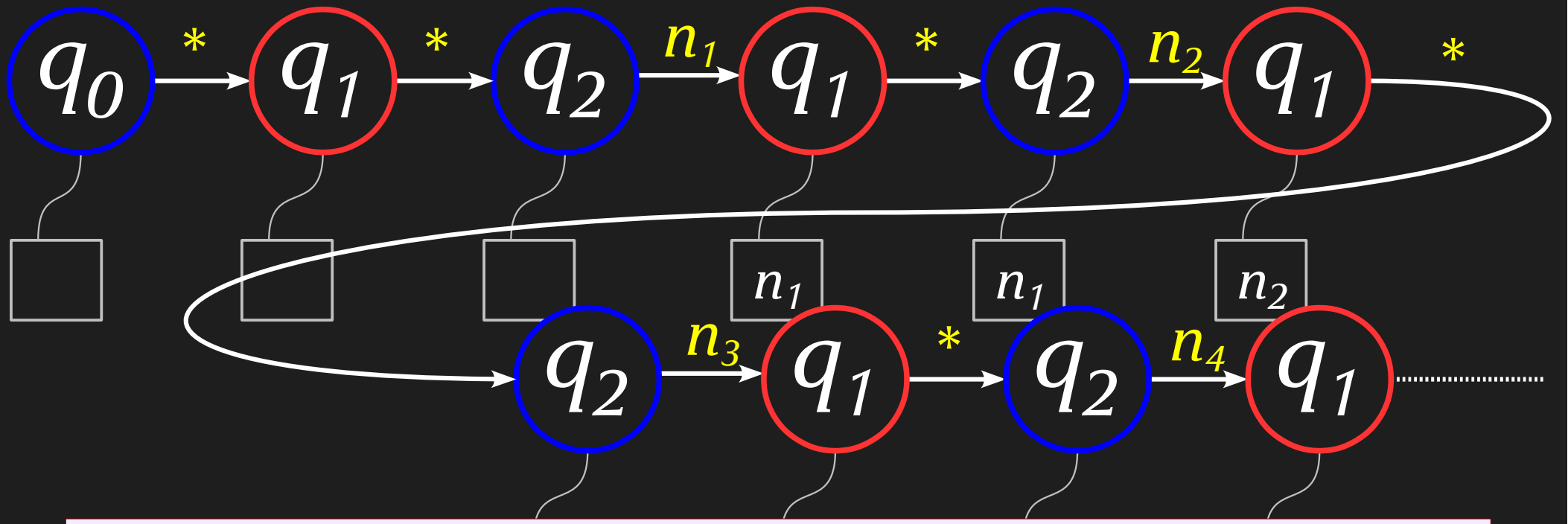




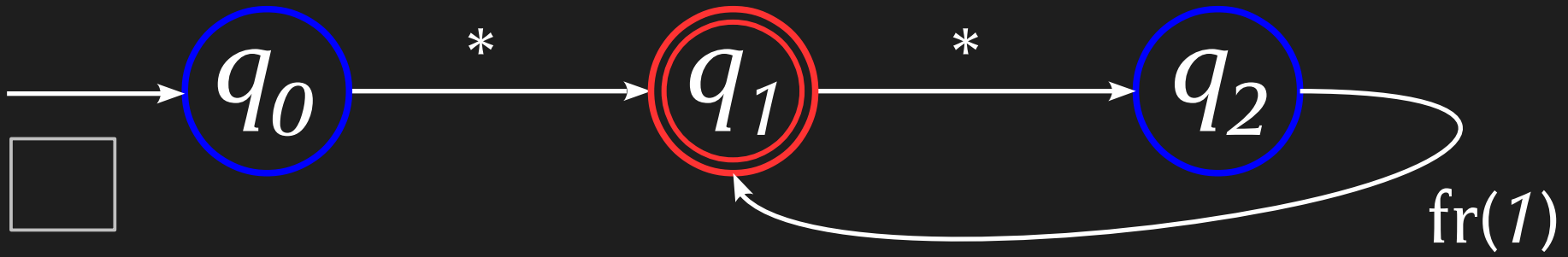
Fresh-register automata



Fresh-register automata

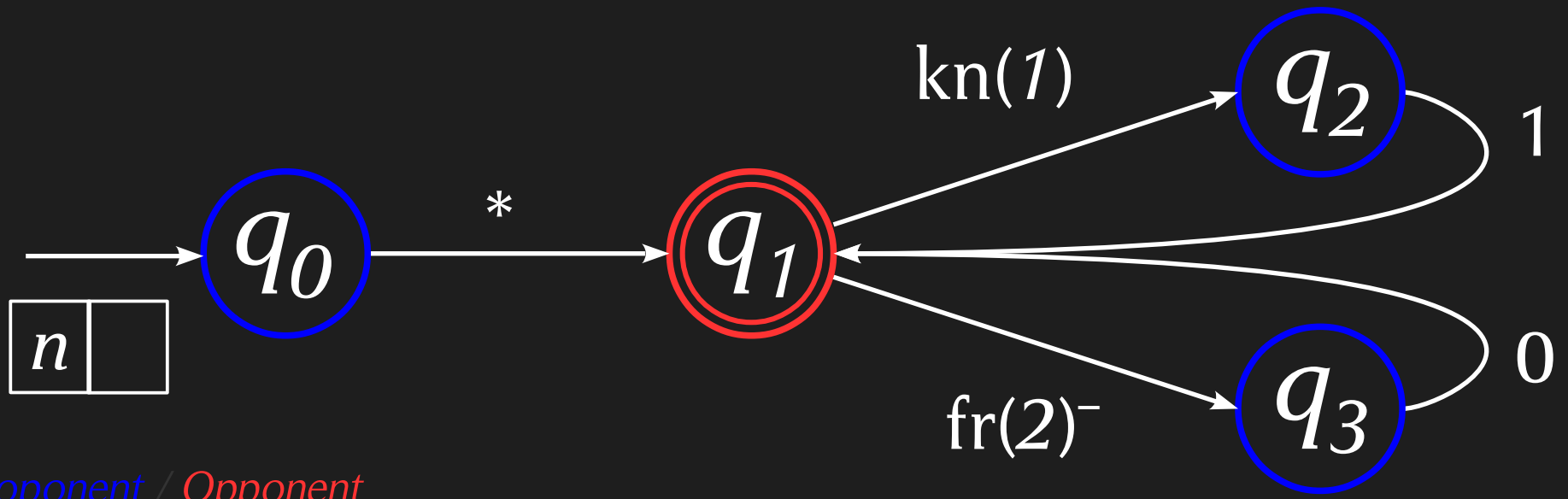


$$\vdash \lambda z.\mathtt{newvar}(0) : \mathtt{com} \to \mathtt{intvar}$$



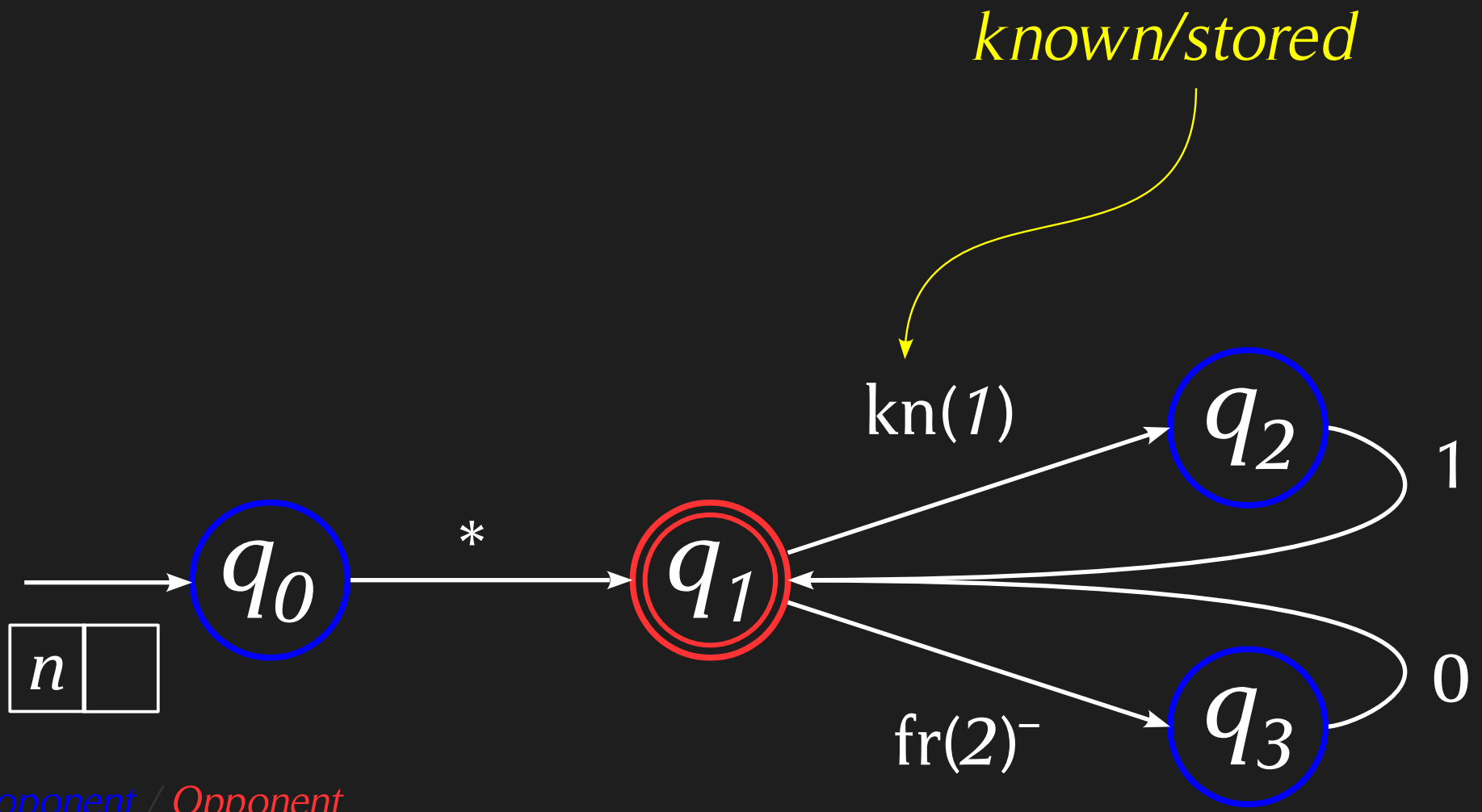
Proponent / *Opponent*

Fresh-register automata



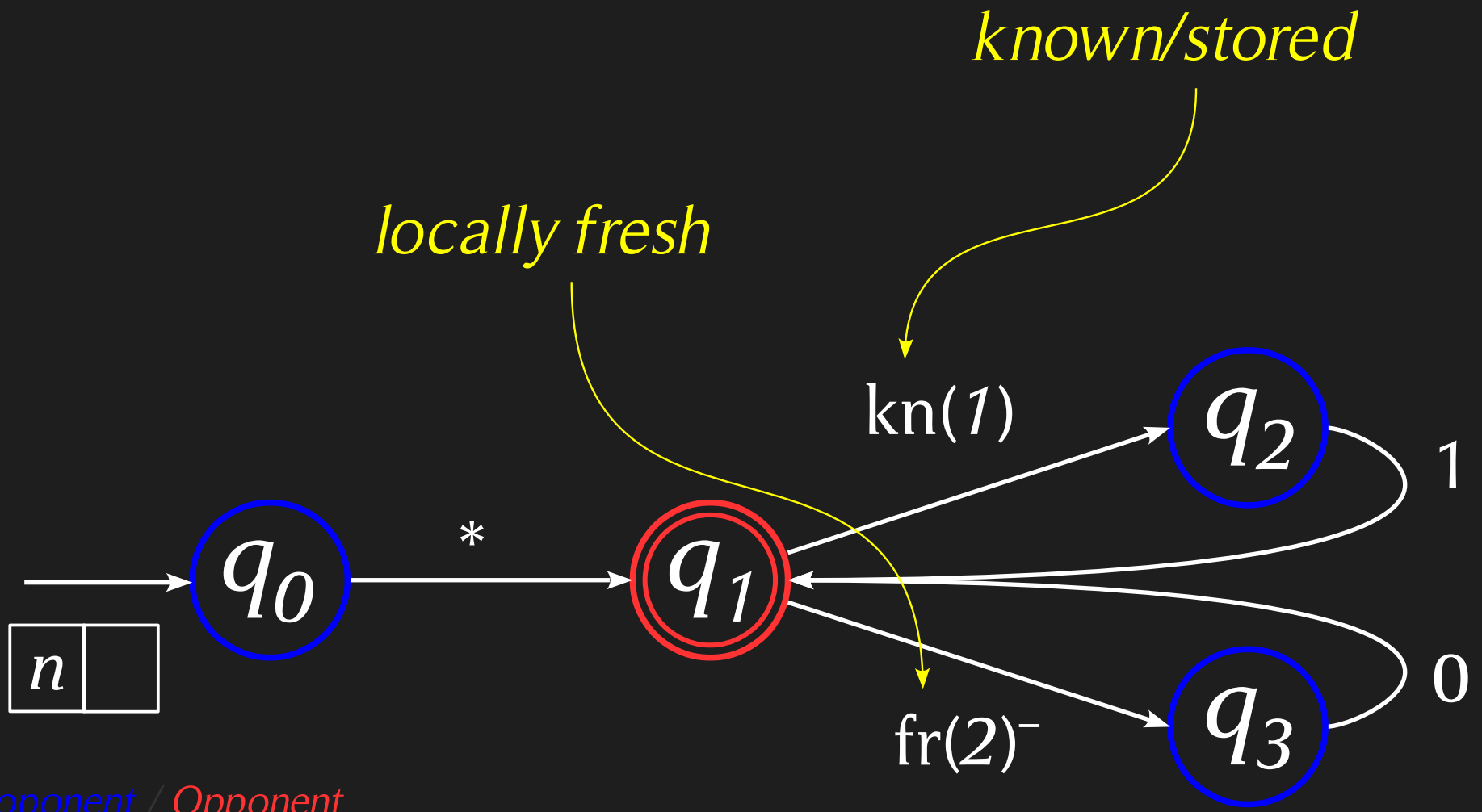
Proponent / *Opponent*

Fresh-register automata



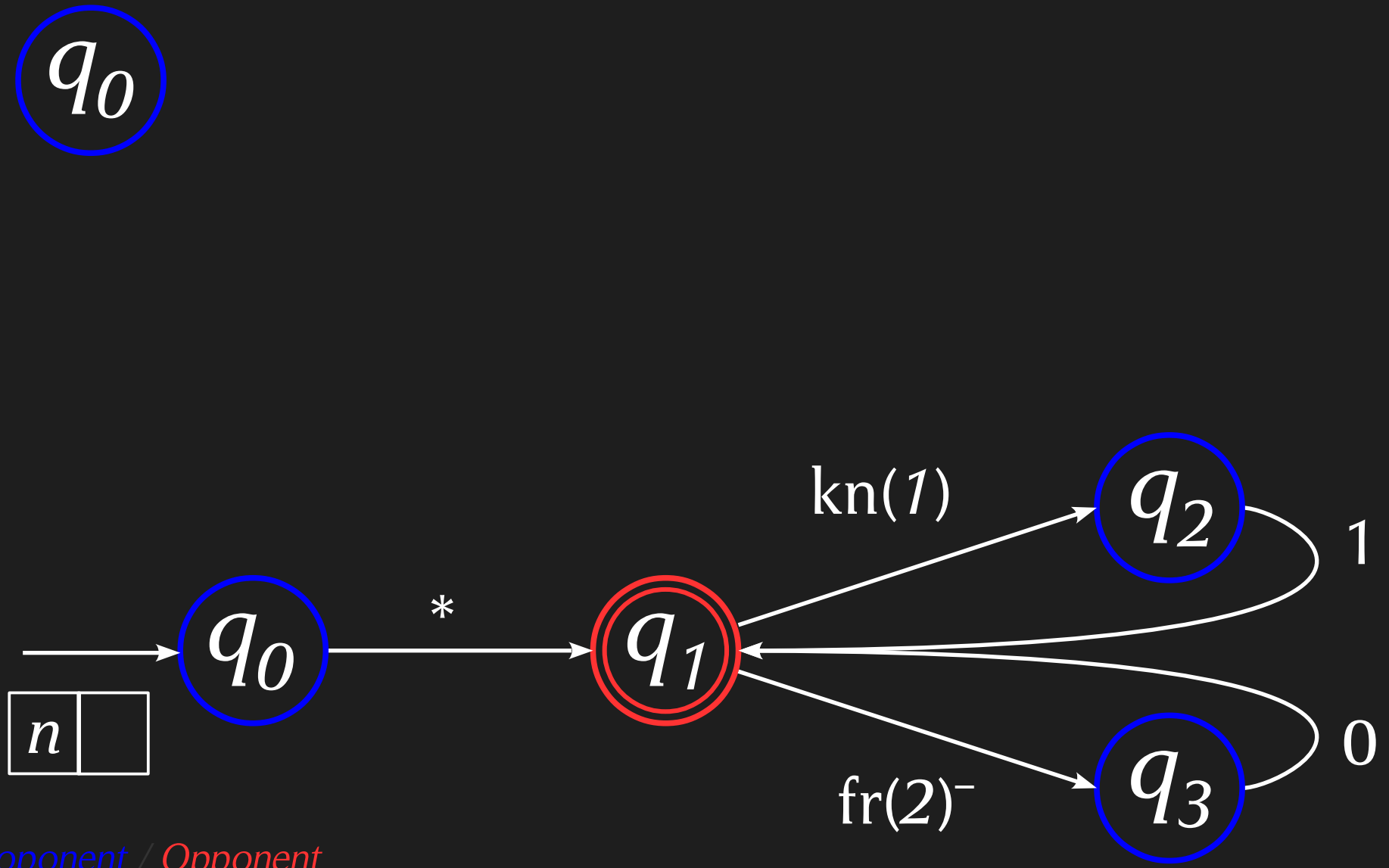
Proponent / *Opponent*

Fresh-register automata

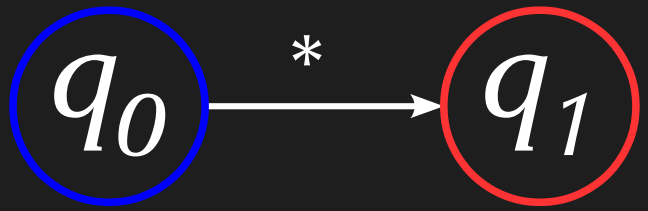


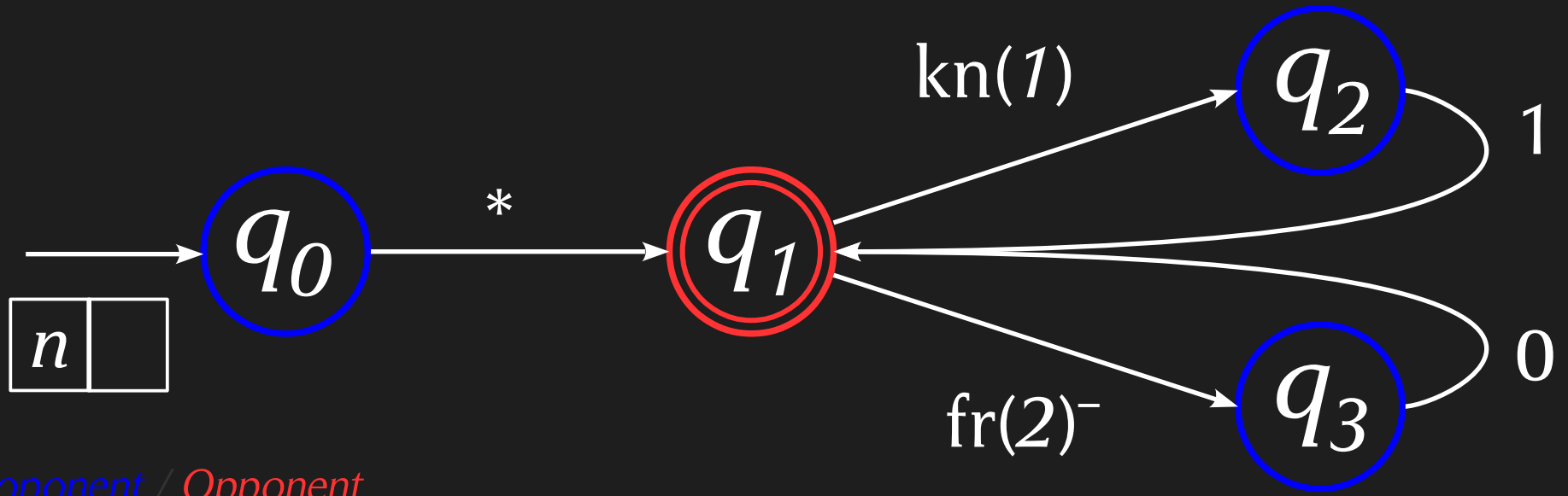
Proponent / *Opponent*

Fresh-register automata



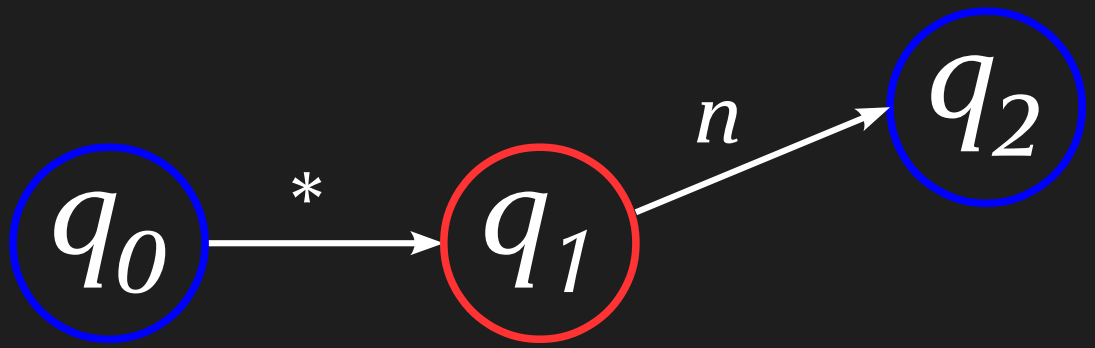
Fresh-register automata

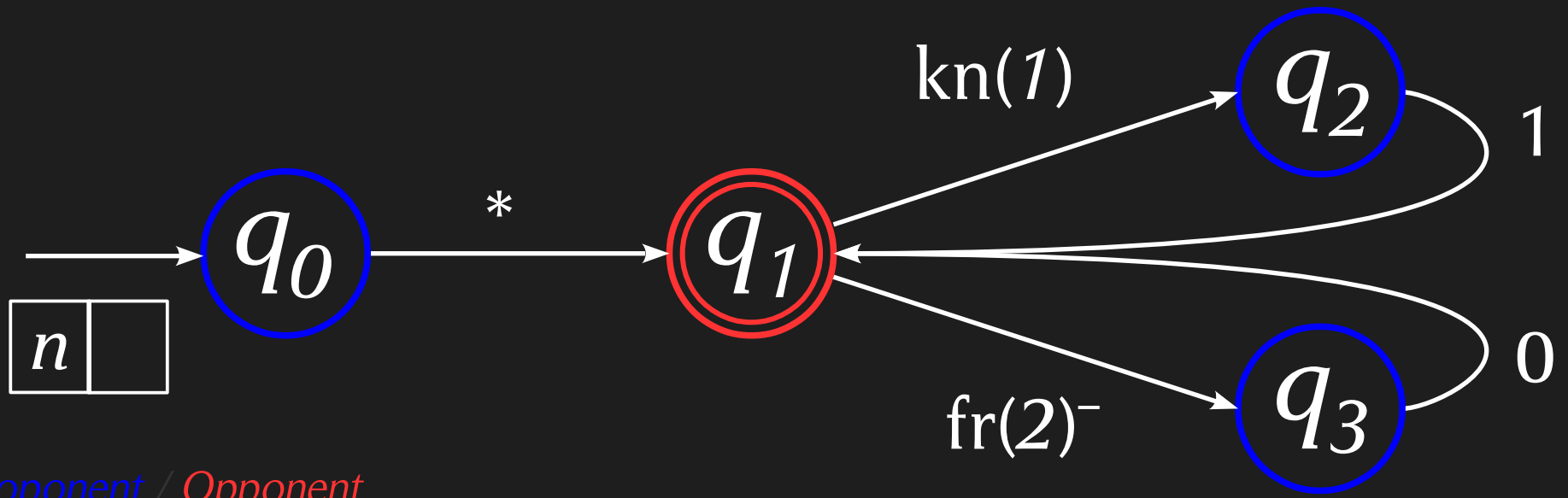




Proponent / *Opponent*

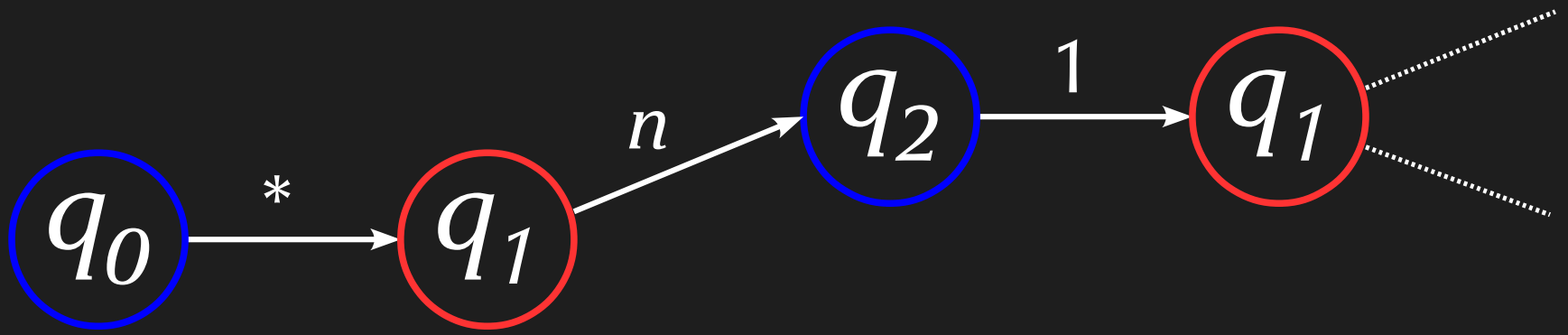
Fresh-register automata

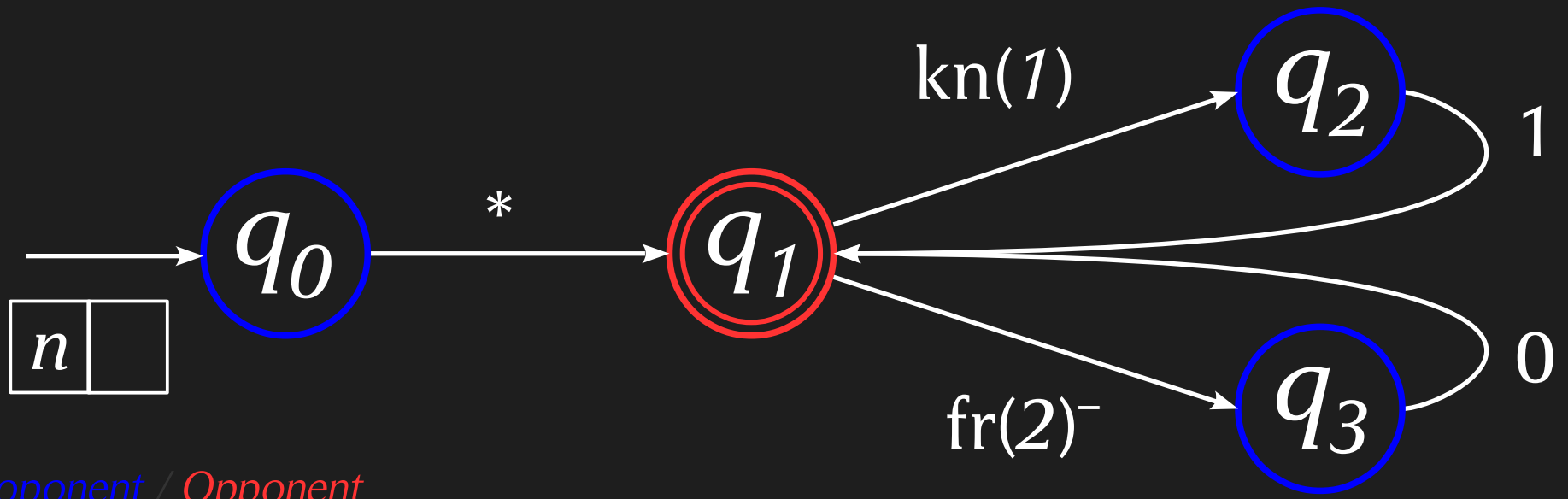




Proponent / *Opponent*

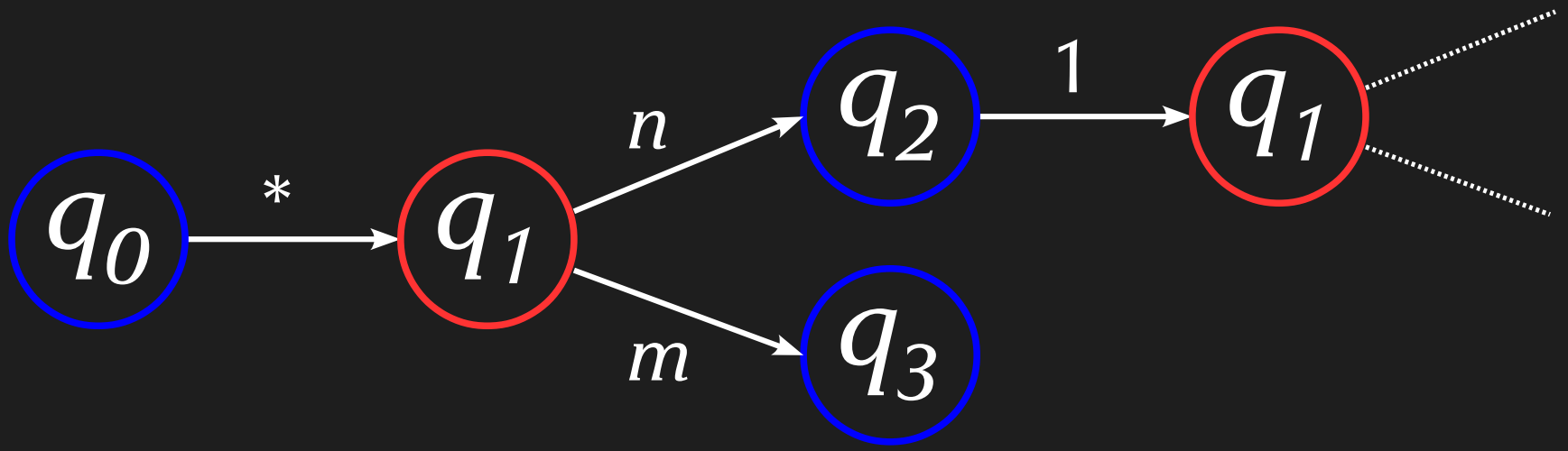
Fresh-register automata

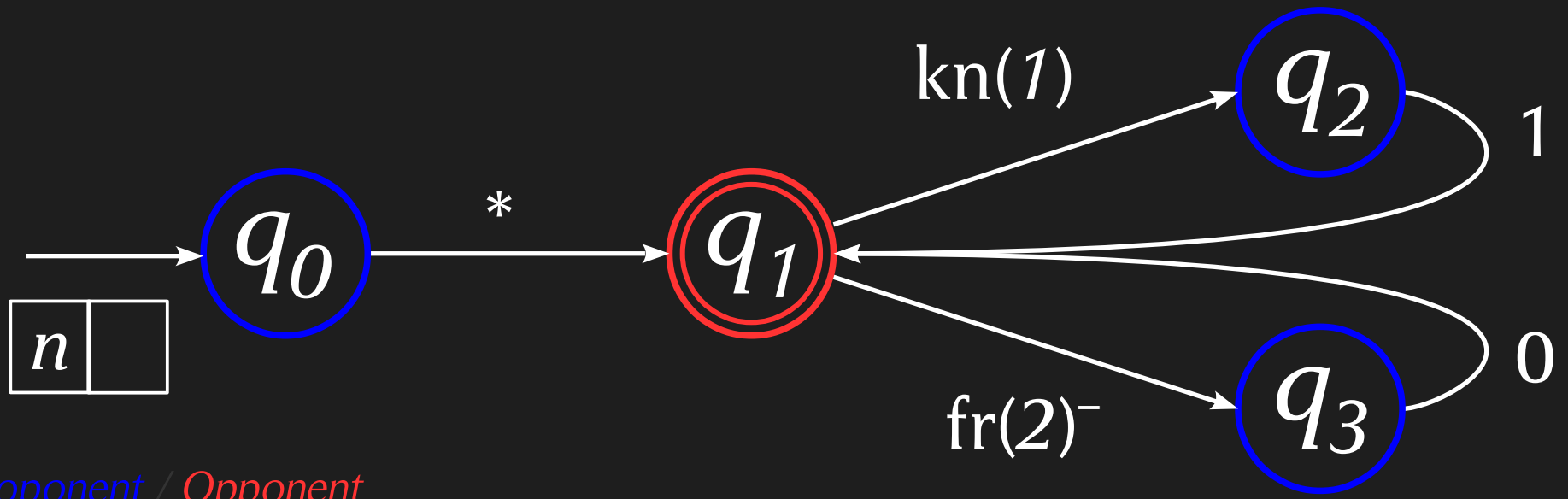




Proponent / *Opponent*

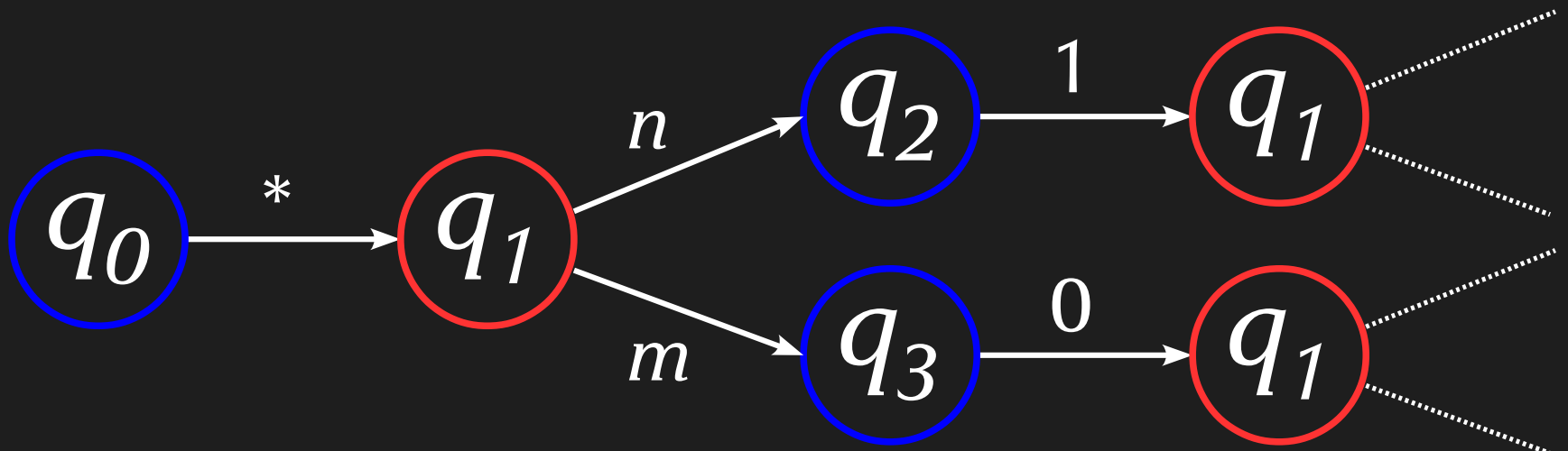
Fresh-register automata

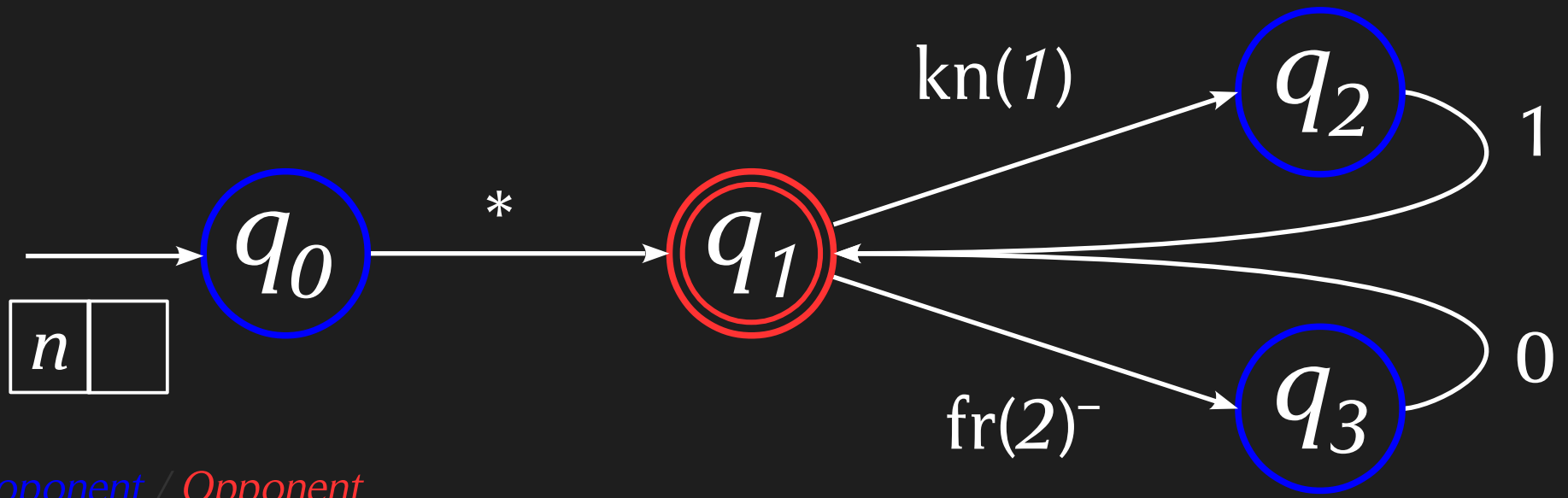




Proponent / *Opponent*

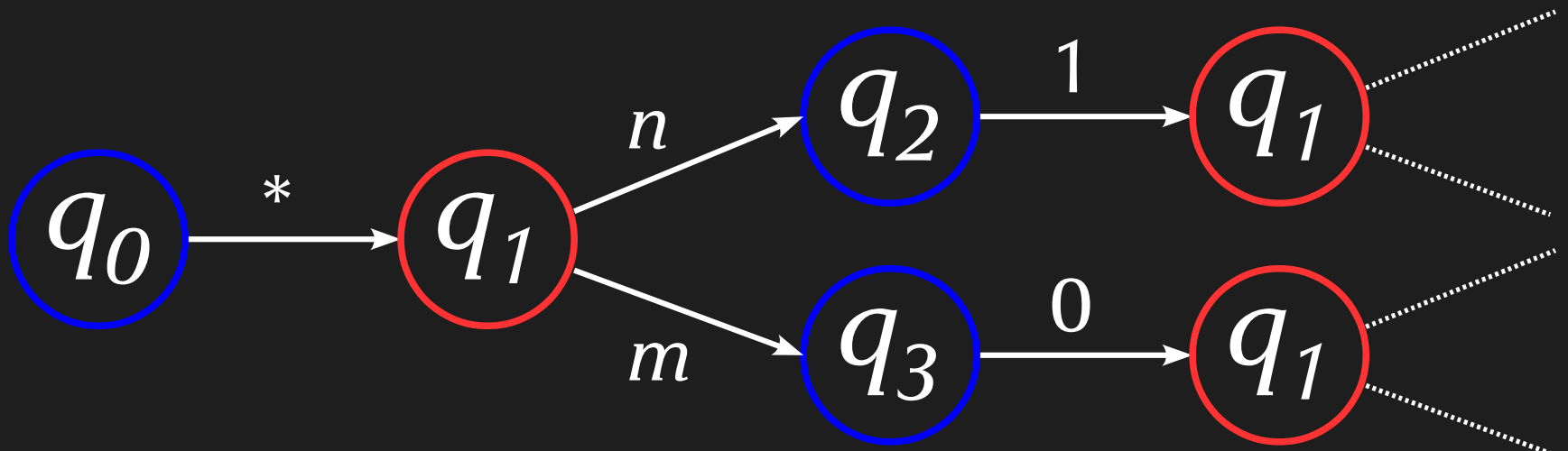
Fresh-register automata



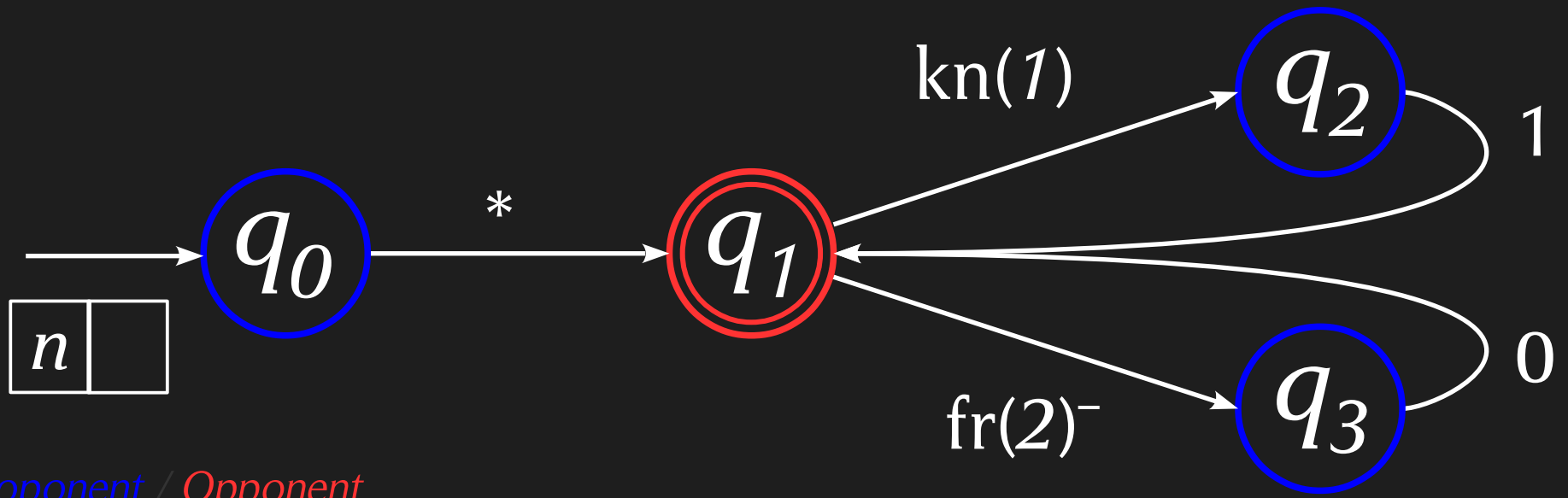


Proponent / *Opponent*

Fresh-register automata




```
x : intvar ⊢ λy.(x == y) : intvar → int
```



Proponent / *Opponent*

Games with new resources

Models of realistic programs (2007-)

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

Algorithmic games

- Fresh-register automata [Tz.11]

Games with new resources

Models of realistic programs (2007-)

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

Algorithmic games

- Fresh-register automata [Tz.11]
- Algorithmic model for mini-RedML [Mur.&Tz.11']

Games with new resources

Models of realistic programs (2007-)

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

Algorithmic games

- Fresh-register automata [Tz.11]
- Algorithmic model for mini-RedML [Mur.&Tz.11']

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

Games with new resources

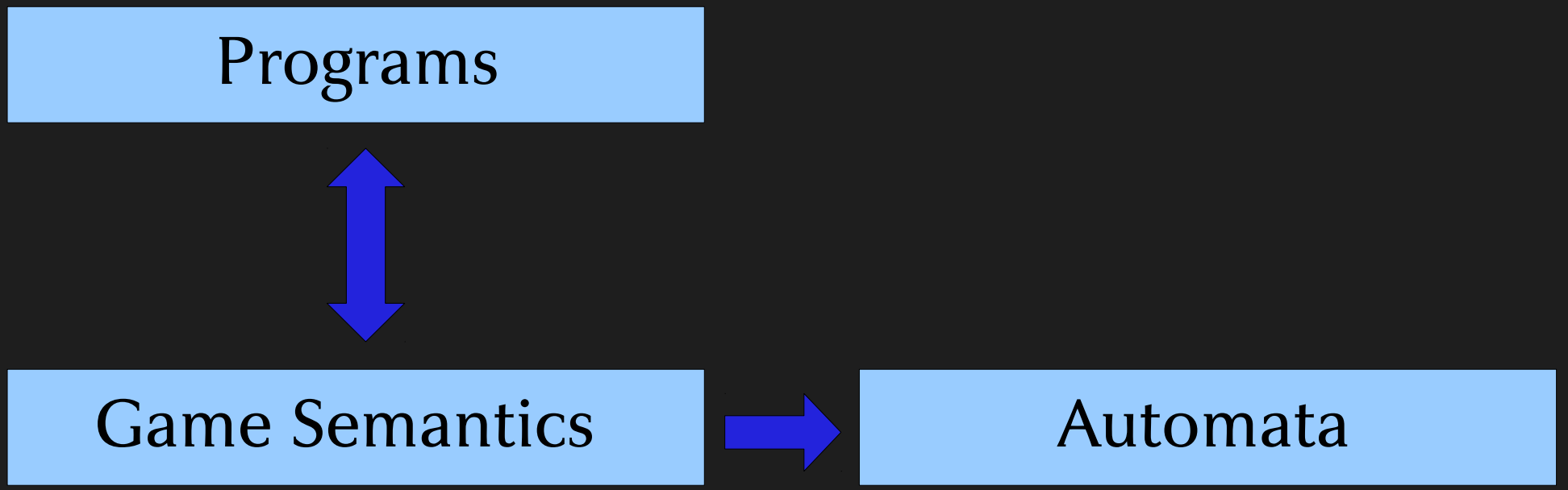
Models of realistic programs (2007-)

- General name-features [Tz.07, Tz.08]
- Direct models [Laird 06, Laird 08, Murawski&Tz.09, Mur.&Tz.11]

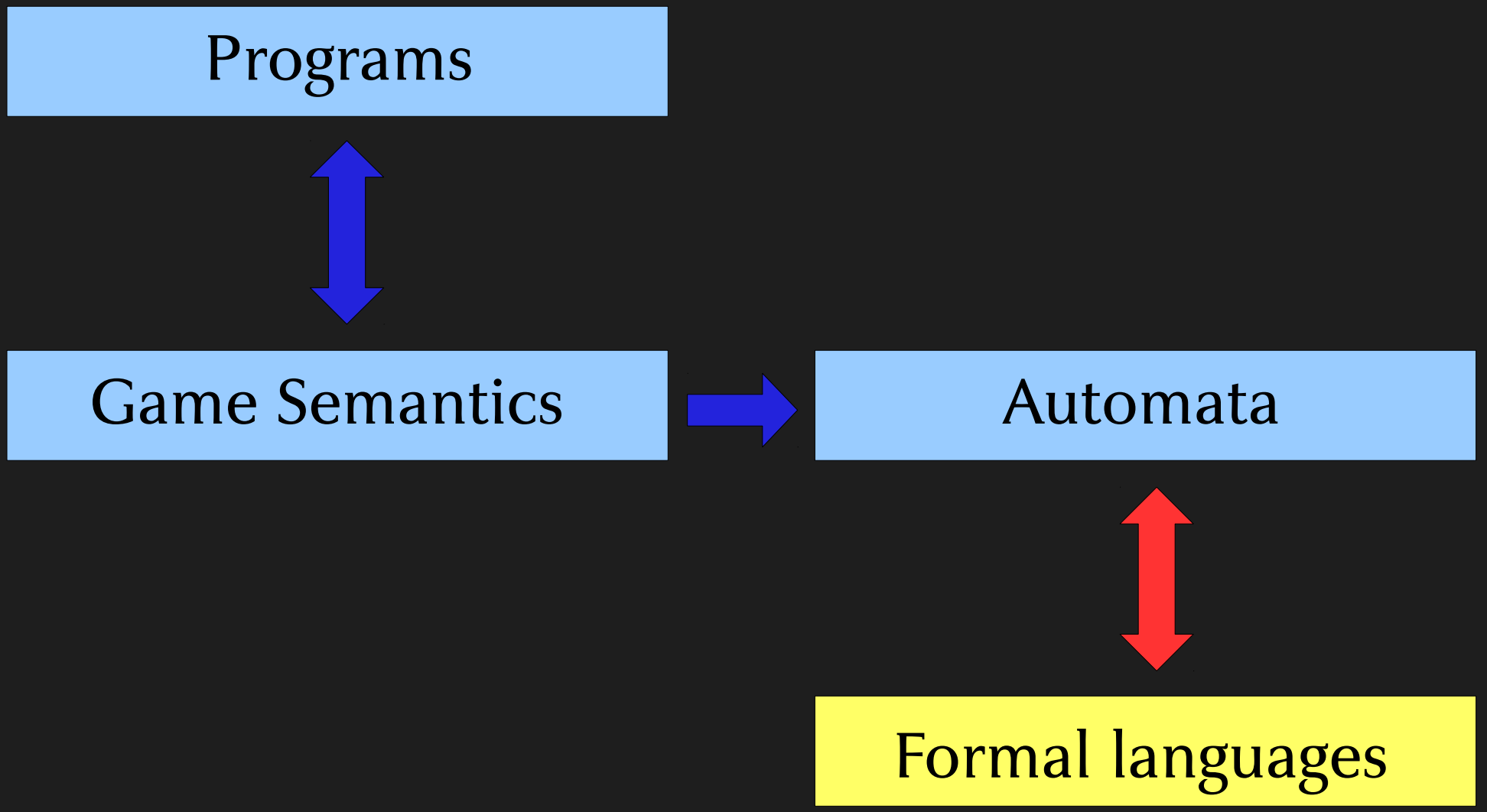
Algorithmic games

- Fresh-register automata [Tz.11]
- Algorithmic model for mini-RedML [Mur.&Tz.11']

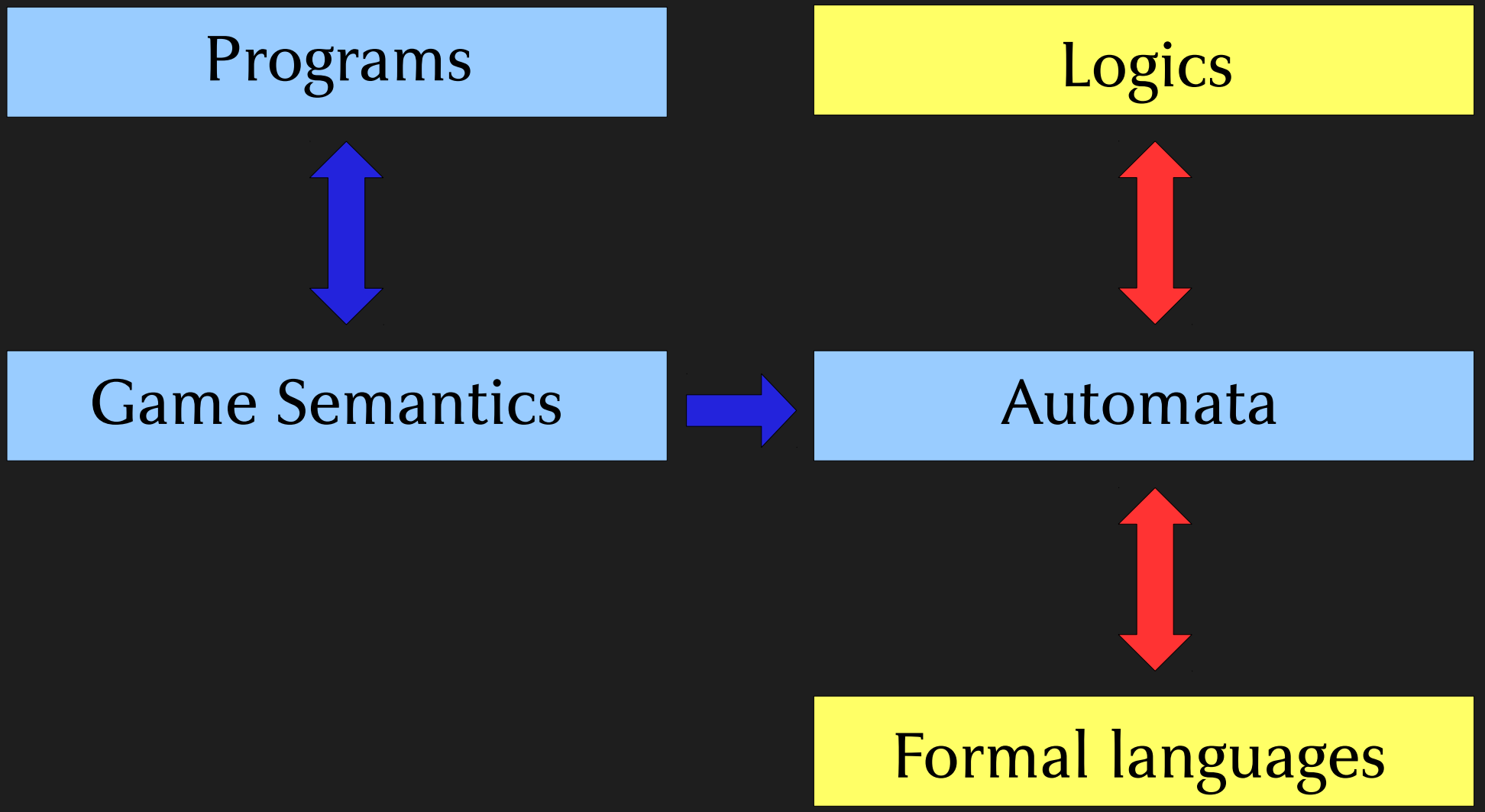
Reasoning about new resources



Reasoning about new resources



Reasoning about new resources



thank you!