Games with names

Nikos Tzevelekos

University of Oxford

What this talk is about

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

We present developments in game semantics which capture new resources

Computation with new resources

$\vdash$ `λx.ref(0)` : com $\to$ `intref`

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

Computation with new resources

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`

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

Computation with new resources

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references + CBV

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references + CBV

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references + CBV

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

$$\lambda y.\,\mathsf{ref}(0) \quad \not\cong \quad \mathsf{let}\, x = \mathsf{ref}(0)\,\mathsf{in}\,(\lambda y.\,x) \ : \mathsf{com} \to \mathsf{intref}$$

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references + CBV

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

$$\lambda y.\, \text{ref}(0) \quad \not\cong \quad \text{let}\, x = \text{ref}(0)\, \text{in}\, (\lambda y.\, x) \ : \text{com} \to \text{intref}$$

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

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

Example: Reduced ML

RedML = simply-typed lambda-calculus
+ integer references + CBV

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

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

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

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

Two ways to model references

Reynolds

- *Idealized Algol (1978)*

References are *pairs:*

```
intref =
 (com → int)x(int → 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:*

```
intref =
 (com → int)x(int → com)
```

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

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

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

```
intref =
 (com → int)x(int → com)

  ⟼  (1 → Z)x(Z → 1)
```

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

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

References are *names:*

`intref` = base type

⟼ **N** (names)

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

```
intref =
 (com → int)x(int → com)
```

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

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

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

References are *names:*

`intref` = 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

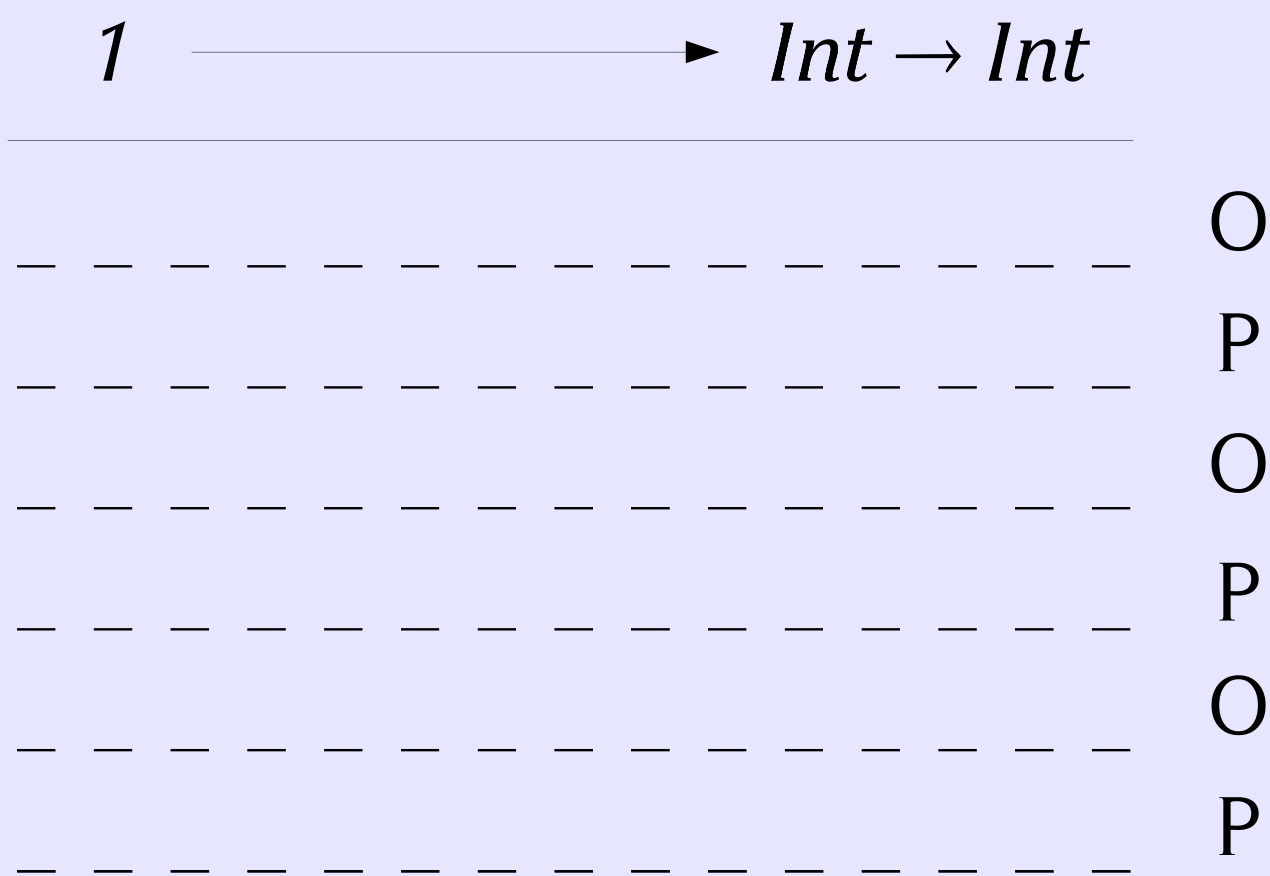
Game Semantics

Game Semantics

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

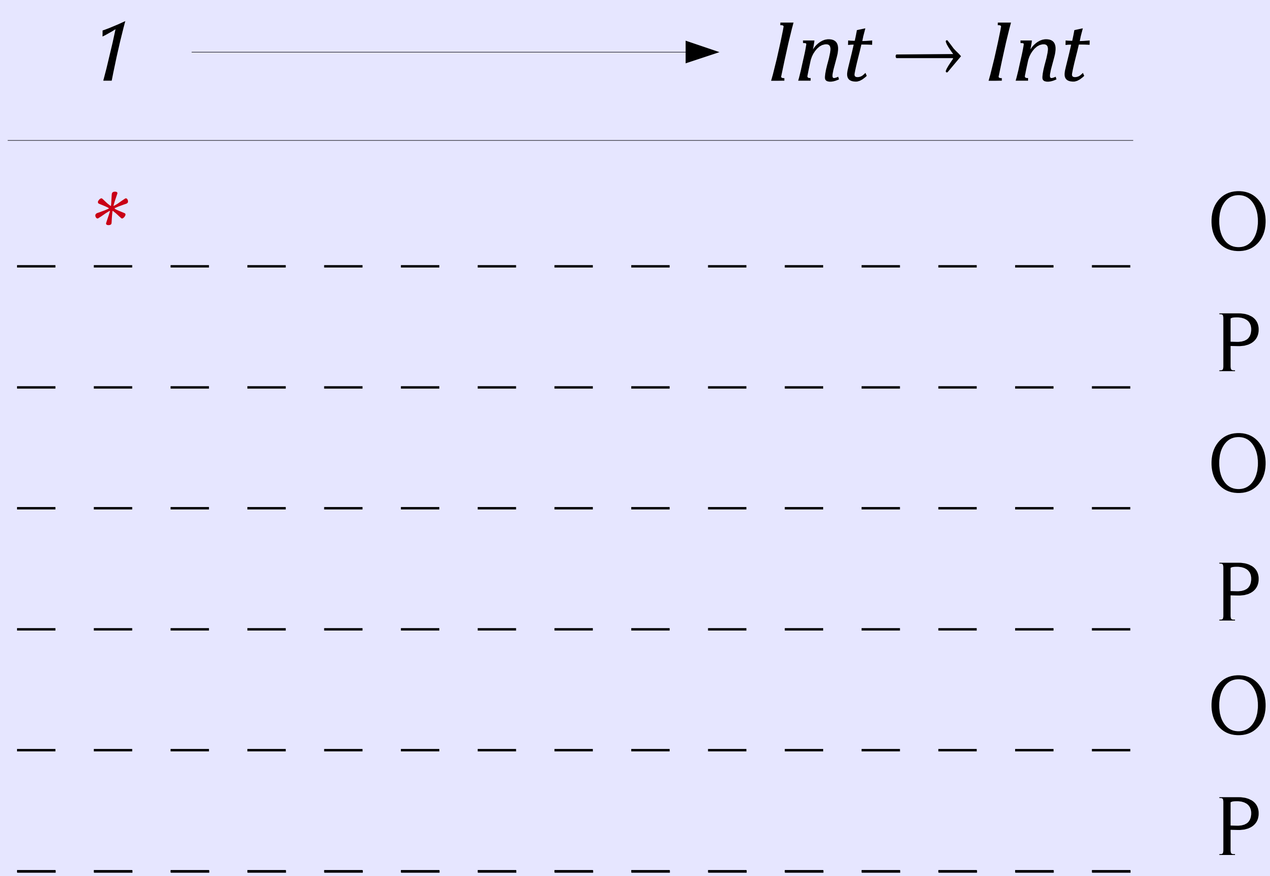
Example

$\vdash$ `λx.x+1 : int` $\rightarrow$ `int`



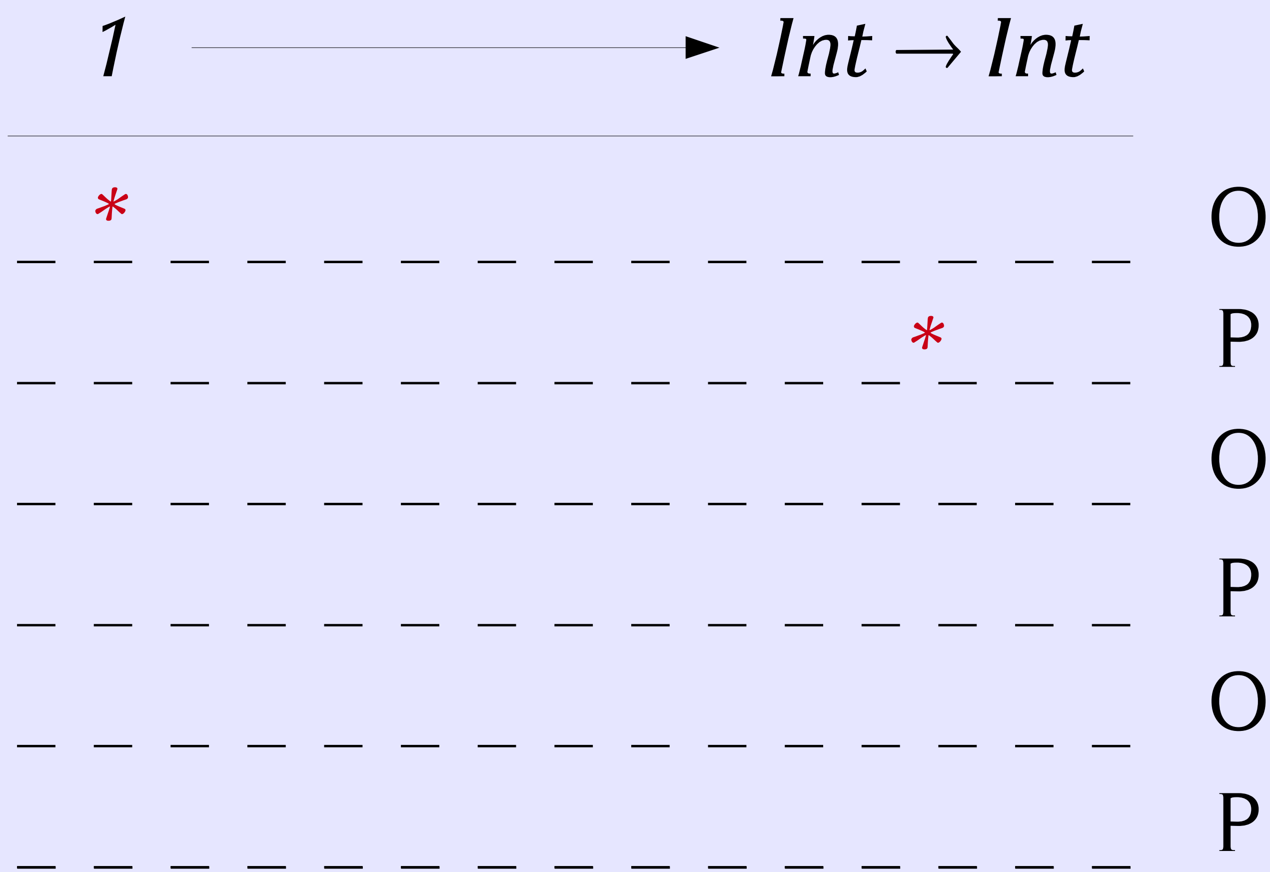
Example

$\vdash$ `λx.x+1 : int → int`



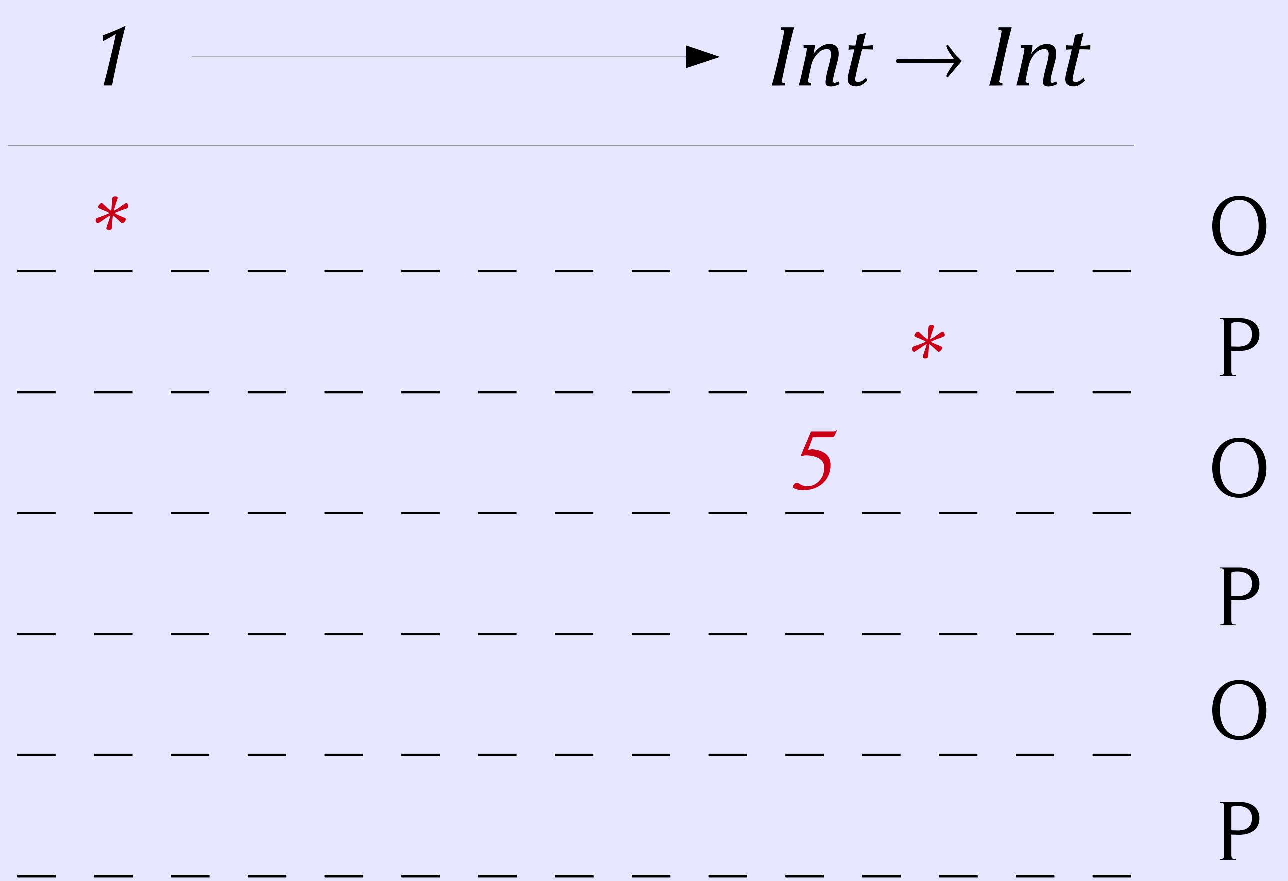
Example

$\vdash$ `λx.x+1 : int → int`



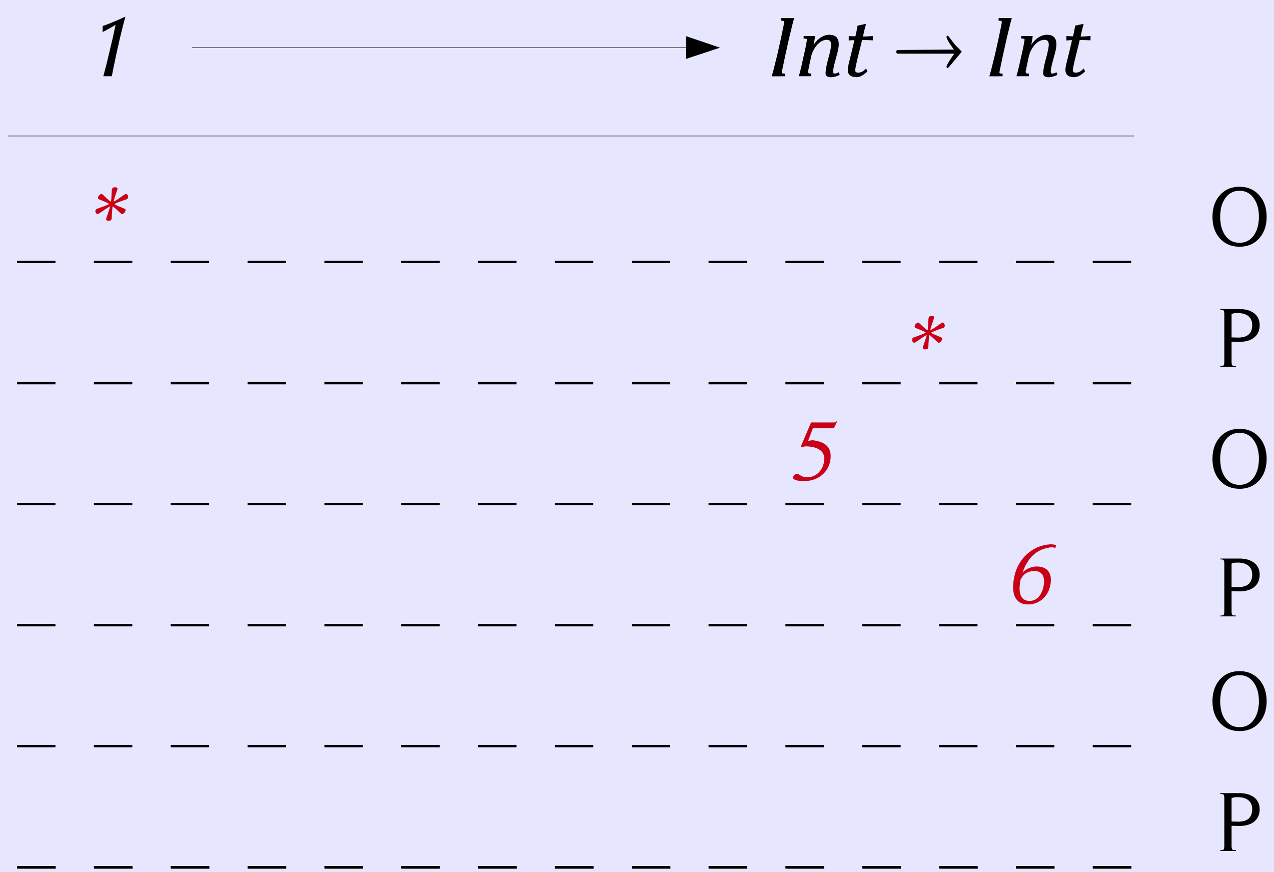
Example

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



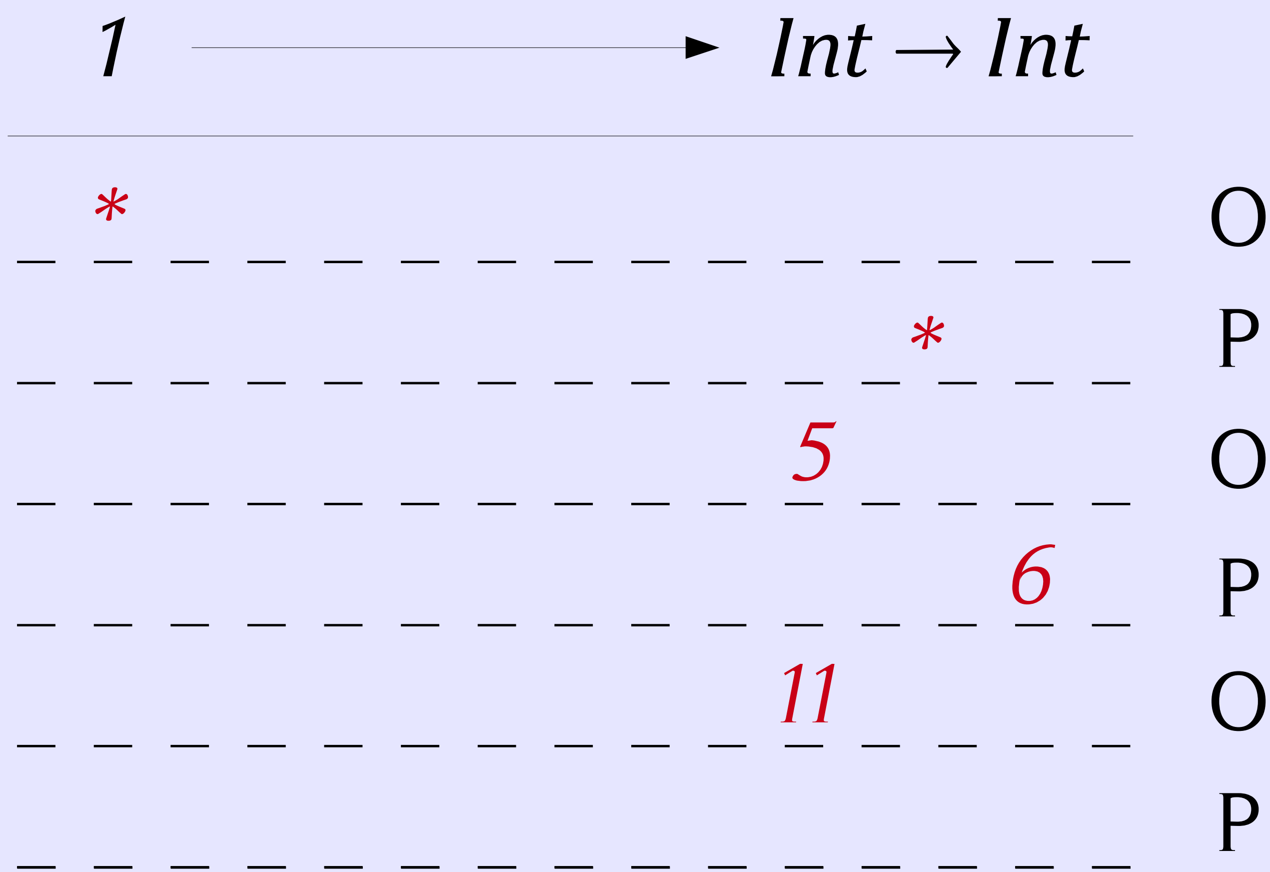
Example

$\vdash$ `λx.x+1 : int → int`



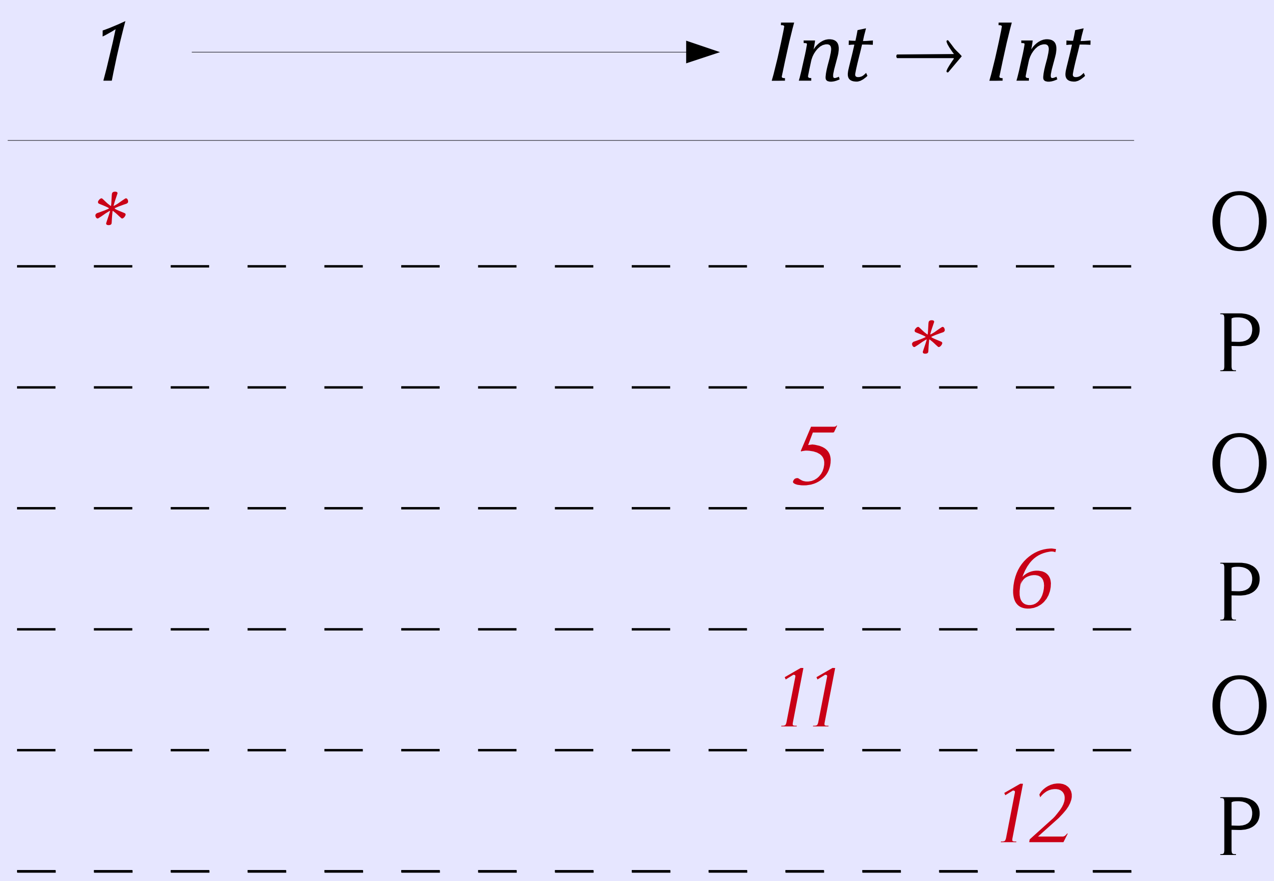
Example

$\vdash$ `λx.x+1 : int → int`



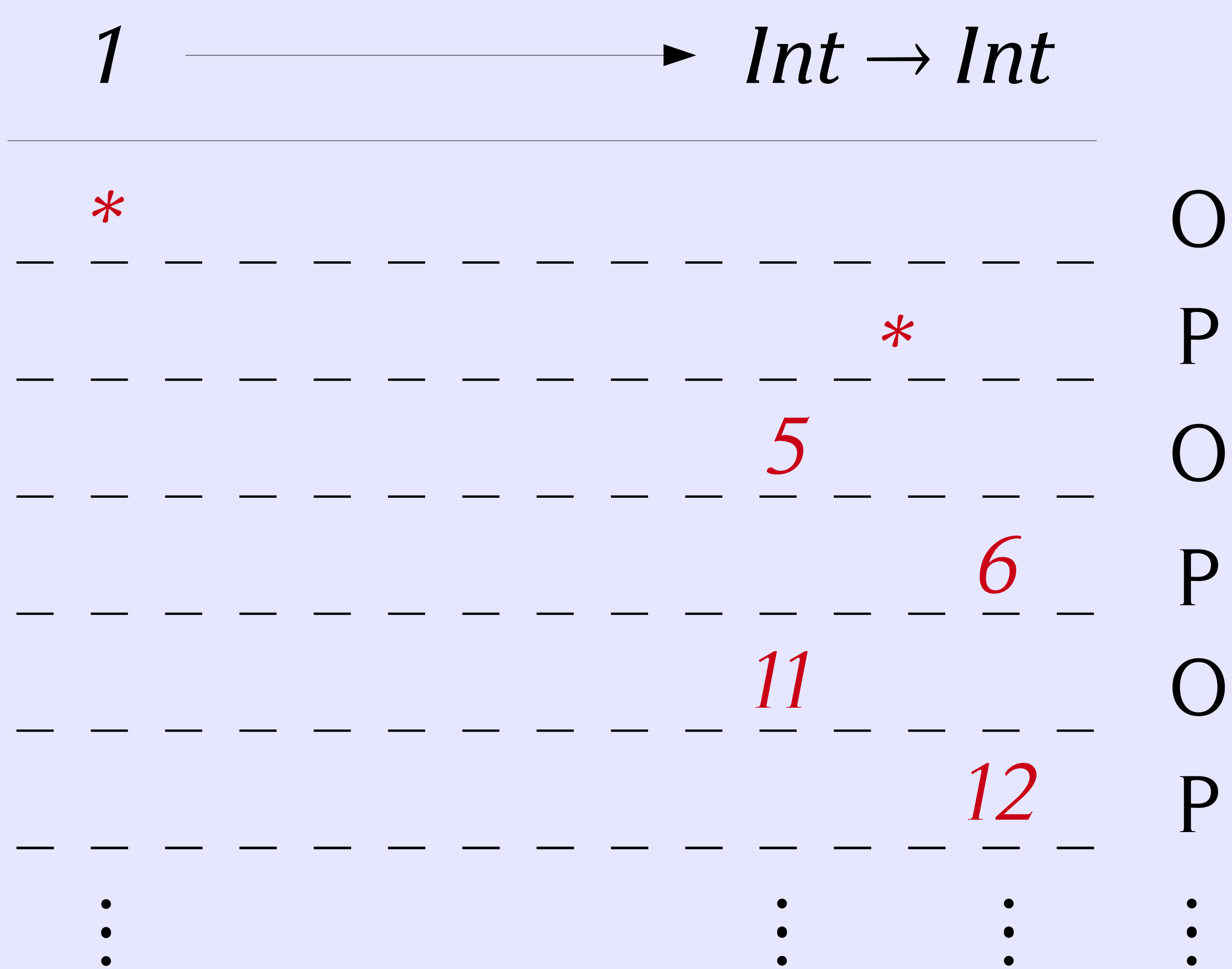
Example

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



Example

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



Example

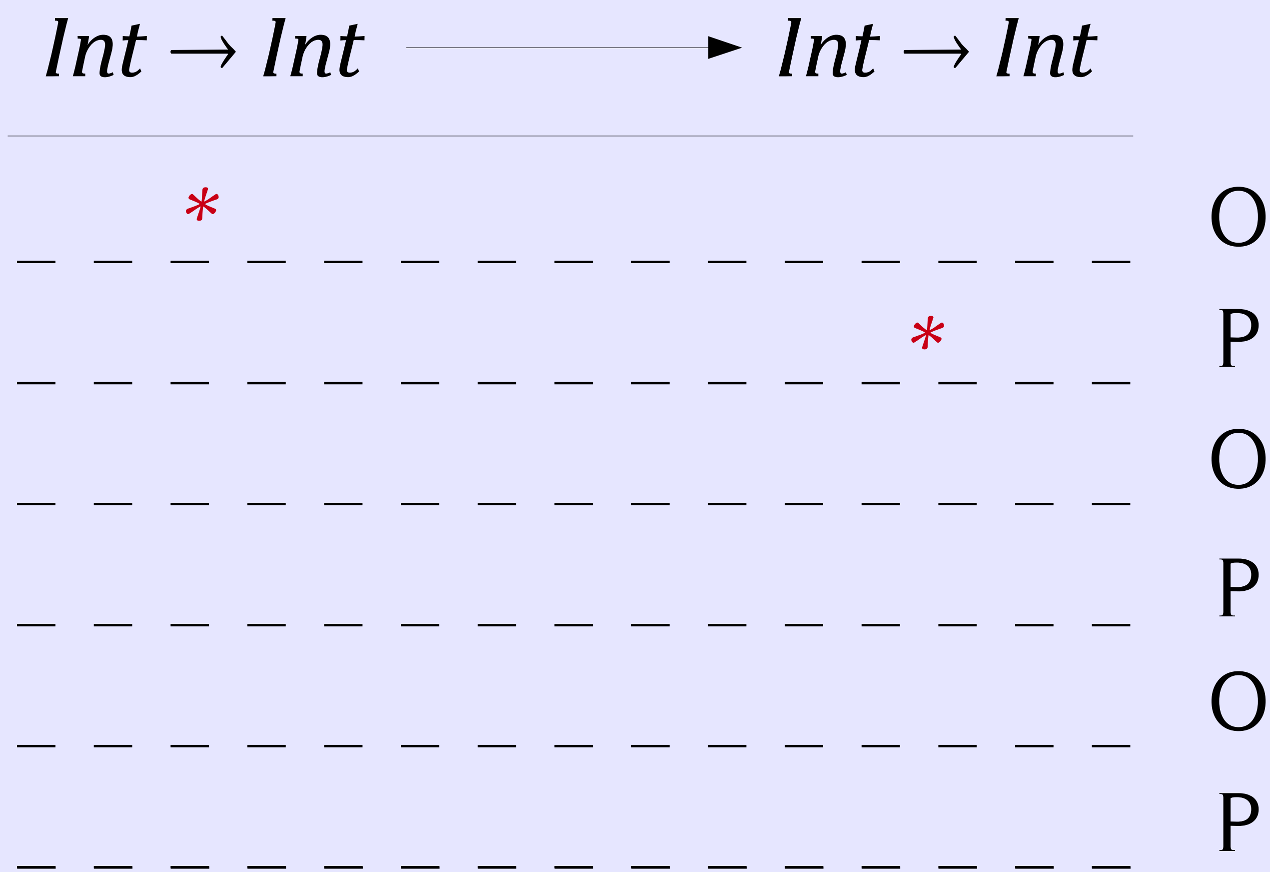
`f : int→int ⊢ λx.f(x)+1 : int→int`

$$Int \to Int \longrightarrow Int \to Int$$

_ _ * _ _ _ _ _ _ _ _ _ _ _ _ _	O
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _	P
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _	O
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _	P
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _	O
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _	P

Example

`f : int→int ⊢ λx.f(x)+1 : int→int`



Example

`f : int→int ⊢ λx.f(x)+1 : int→int`

$$Int \to Int \longrightarrow Int \to Int$$

_	_	*	_	_	_	_	_	_	_	_	_	_	_	_	O
_	_	_	_	_	_	_	_	_	_	_	_	*	_	_	P
_	_	_	_	_	_	_	_	_	_	5	_	_	_	_	O
_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	P
_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	O
_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	P

Example

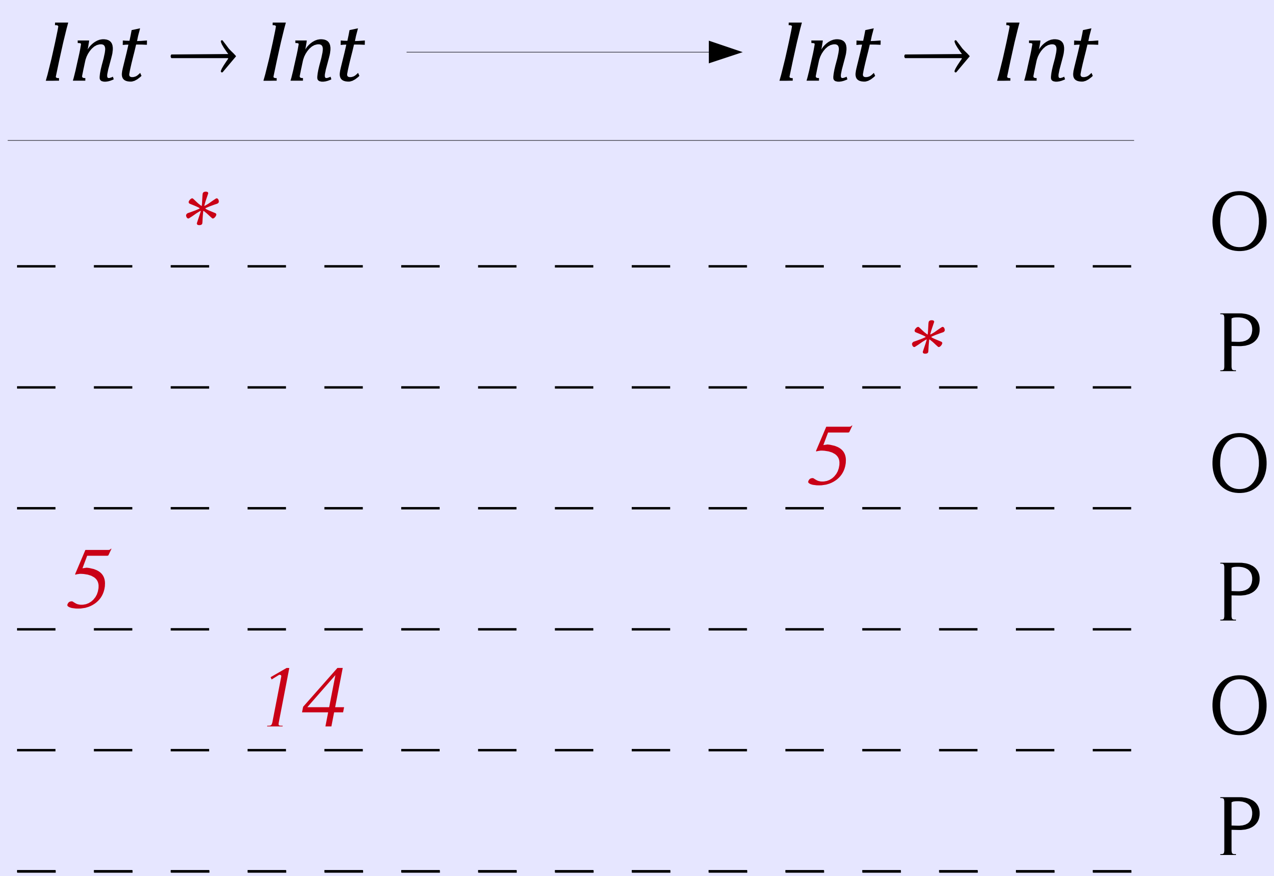
`f : int→int ⊢ λx.f(x)+1 : int→int`

$$Int \to Int \longrightarrow Int \to Int$$

$*$	O
$*$	P
5	O
5	P
	O
	P

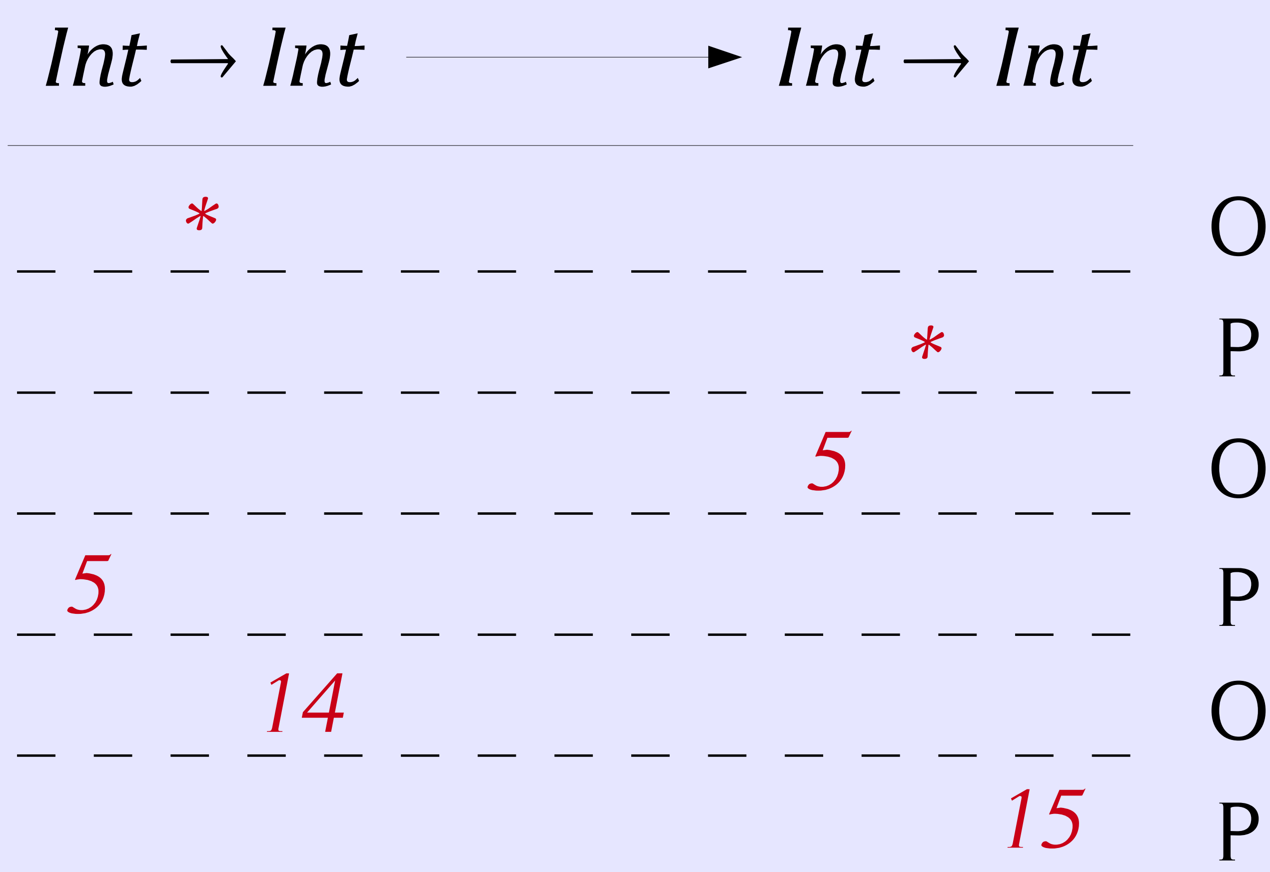
Example

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



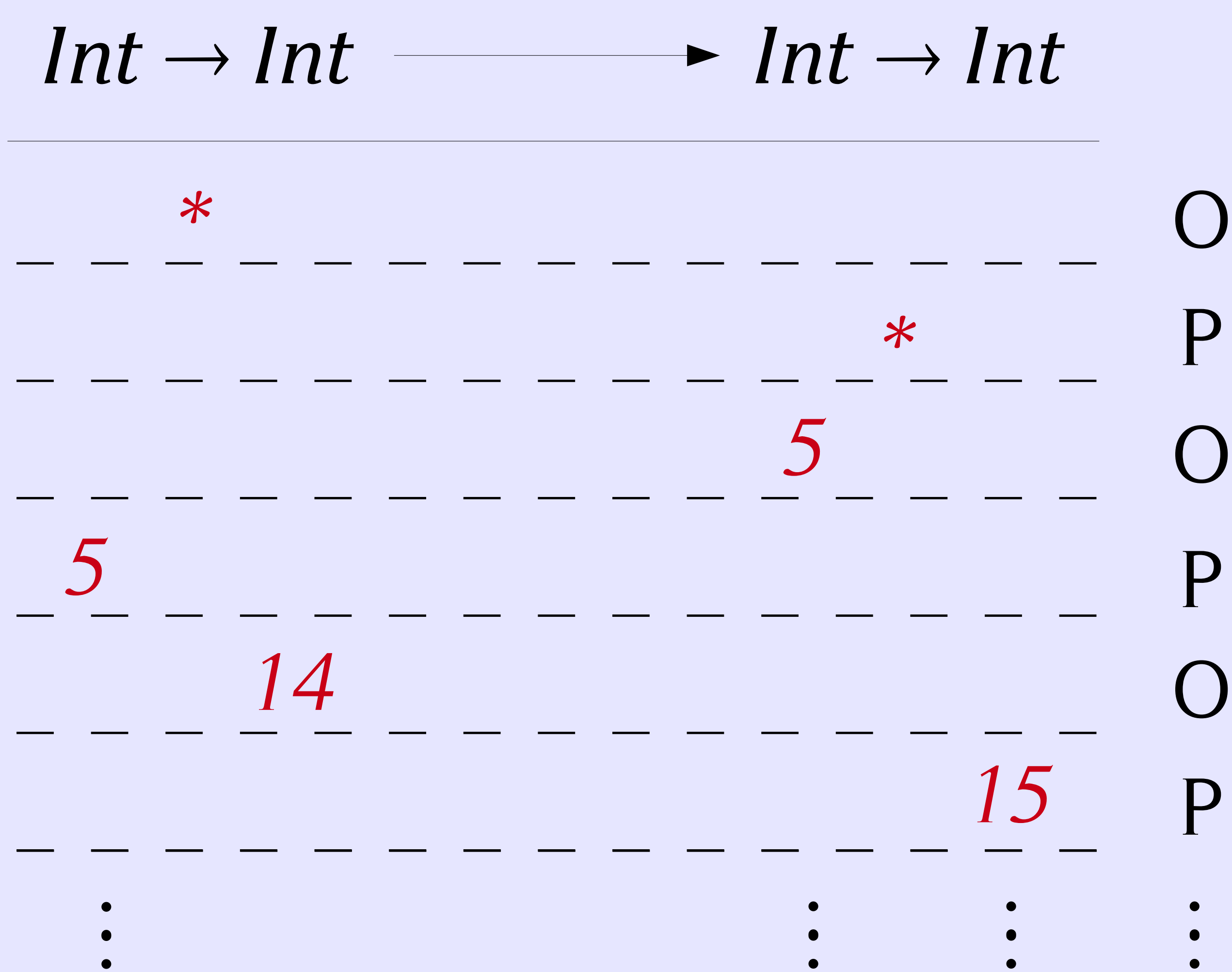
Example

`f : int→int ⊢ λx.f(x)+1 : int→int`

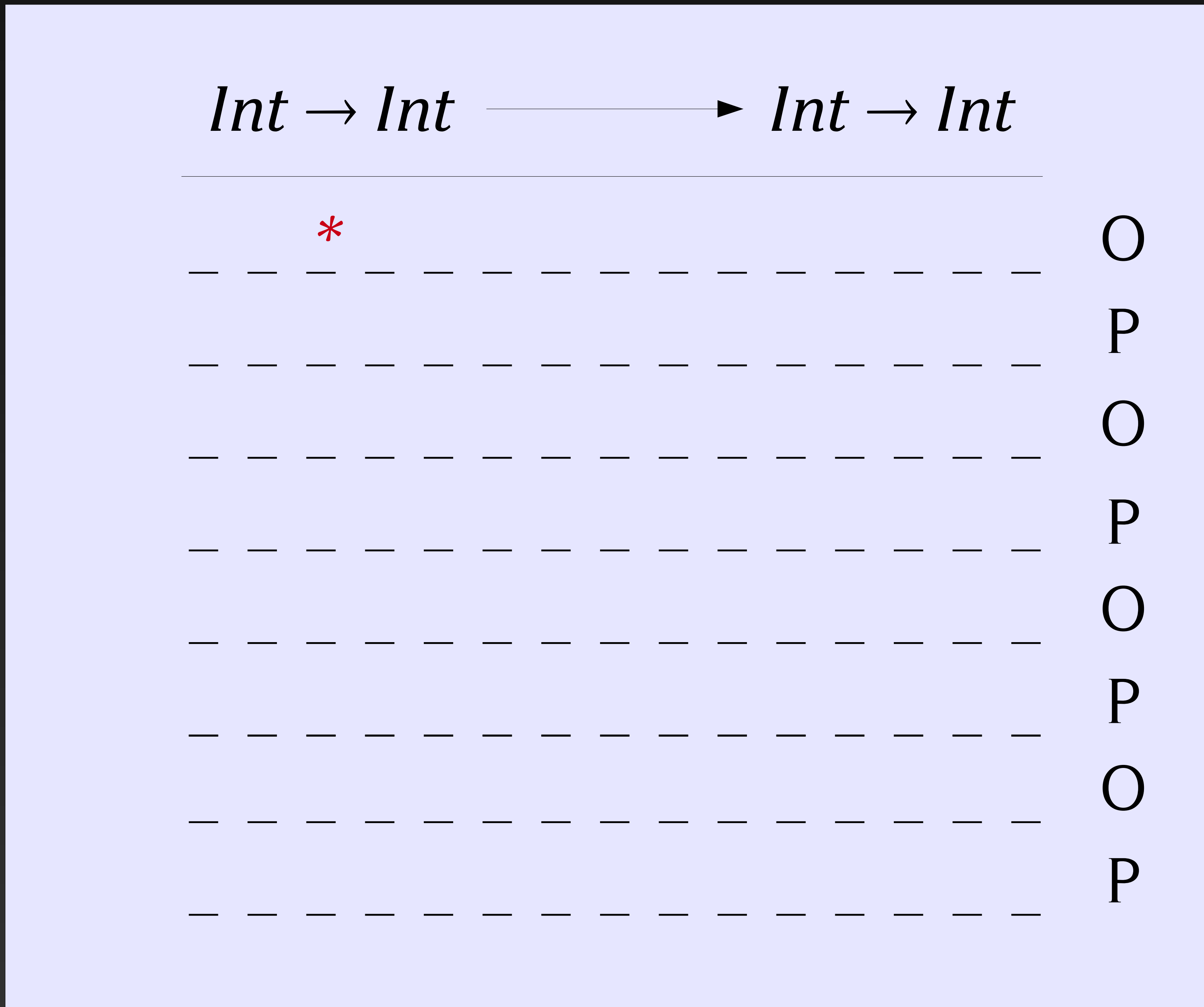


Example

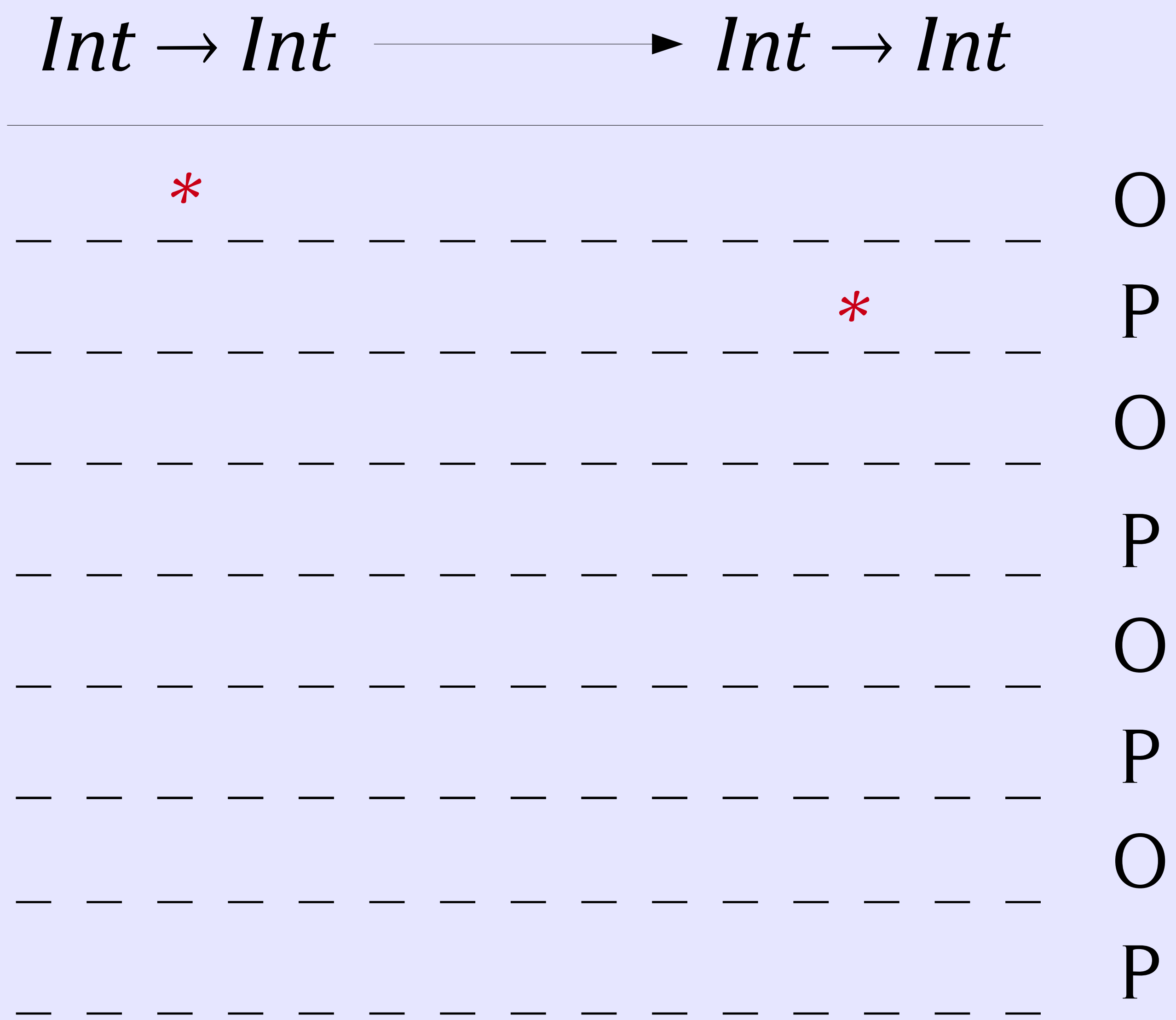
$$f : \mathtt{int} \to \mathtt{int} \vdash \lambda x.f(x)+1 : \mathtt{int} \to \mathtt{int}$$



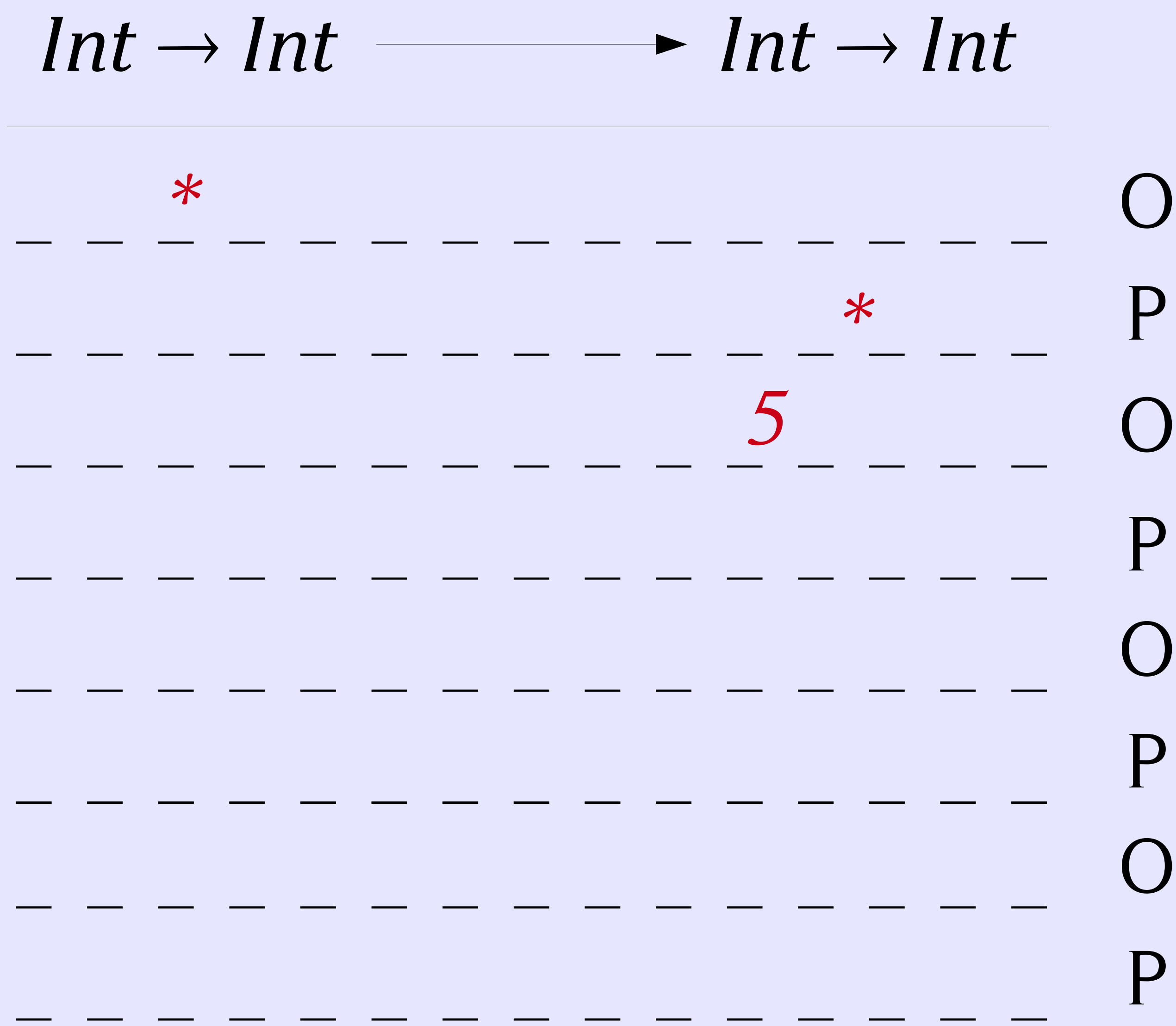
Example



Example

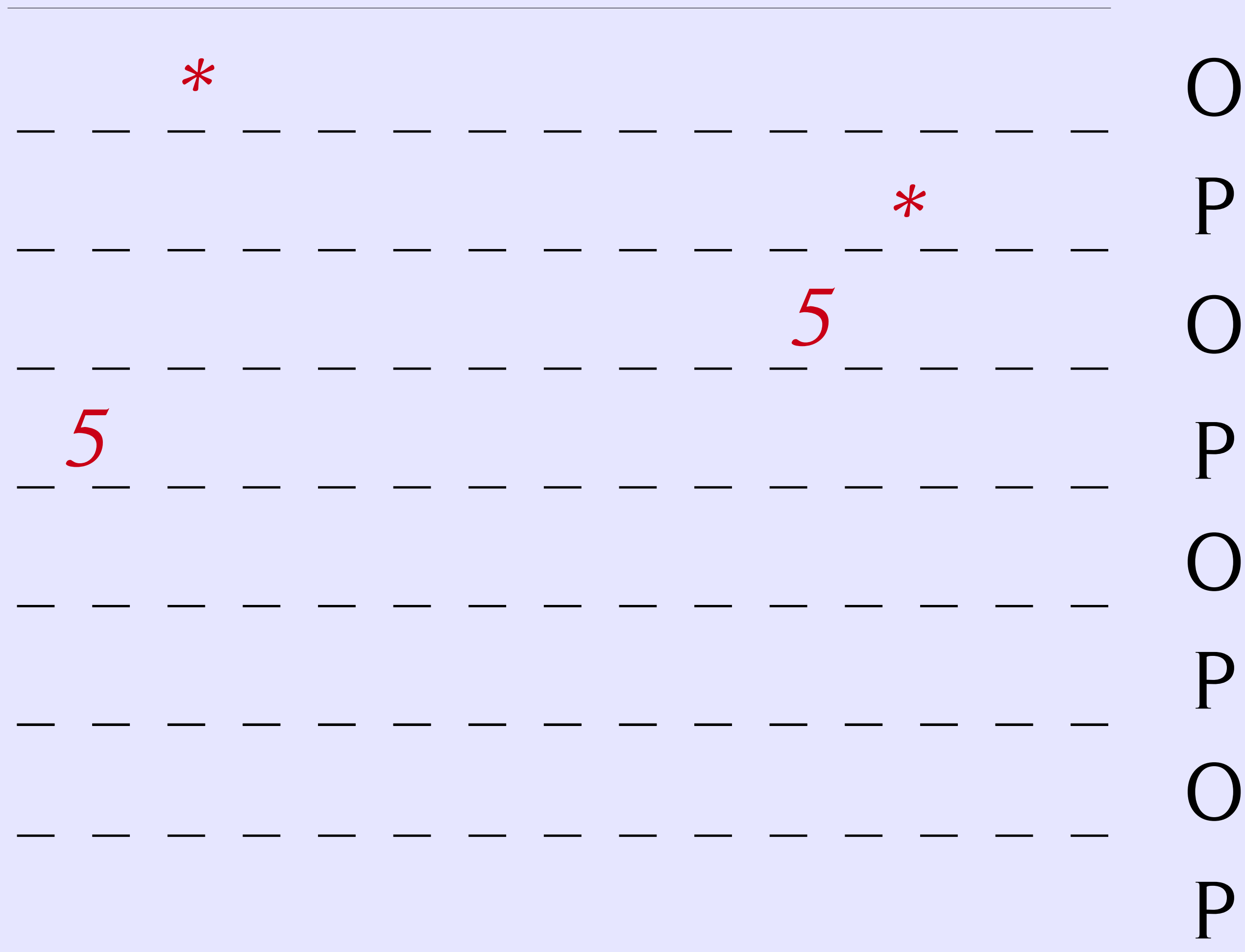


Example

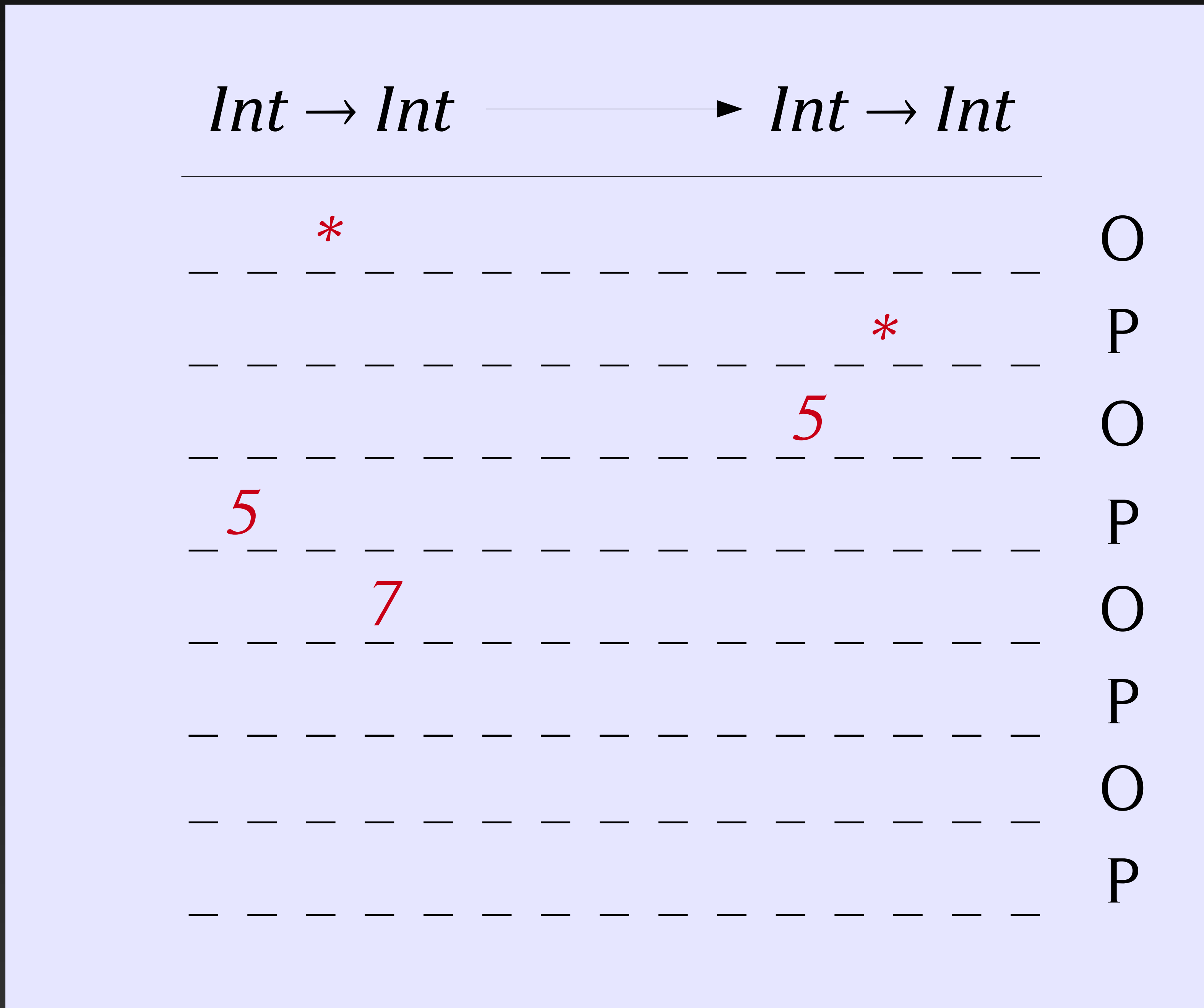


Example

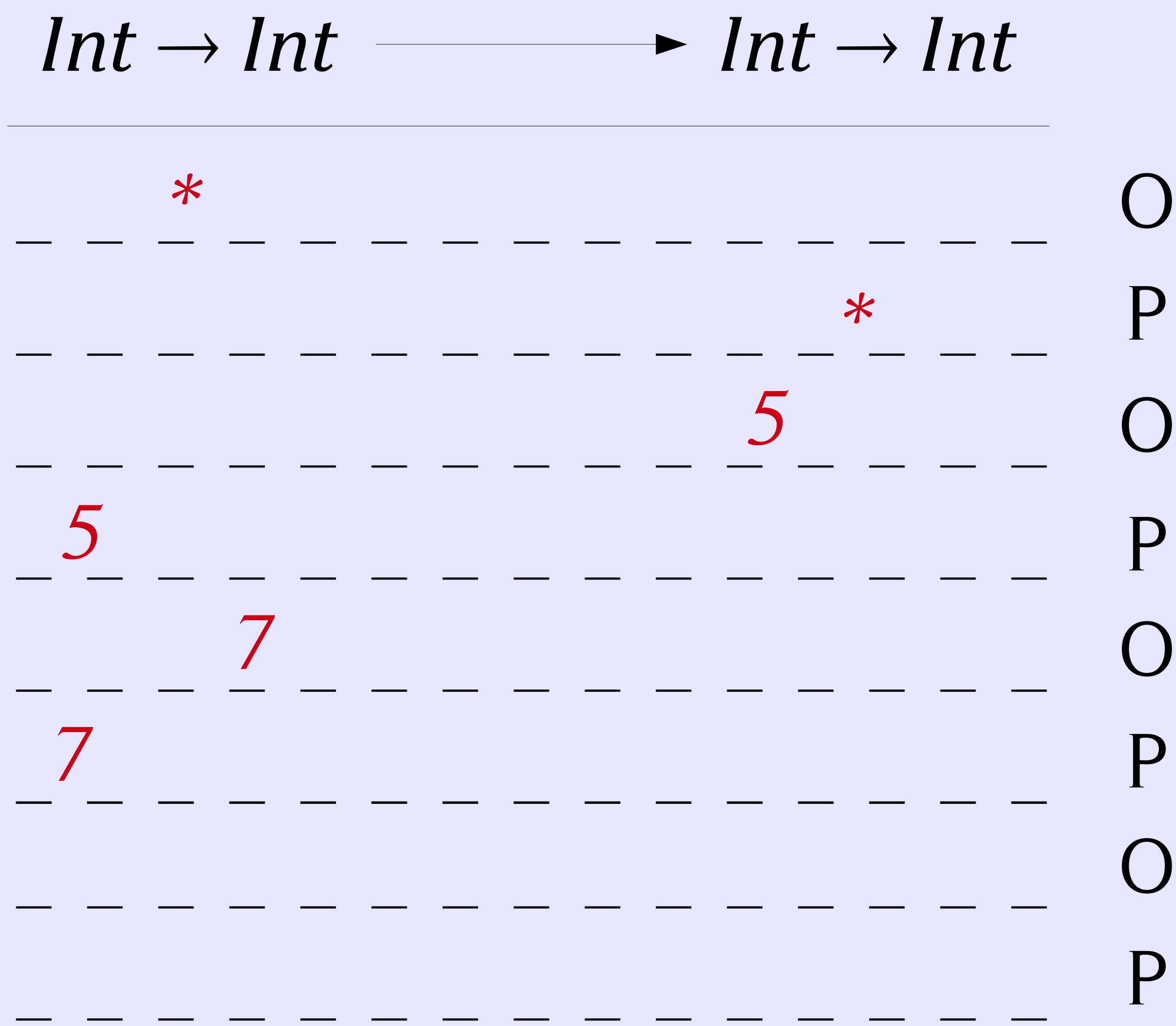
$$Int \to Int \longrightarrow Int \to Int$$



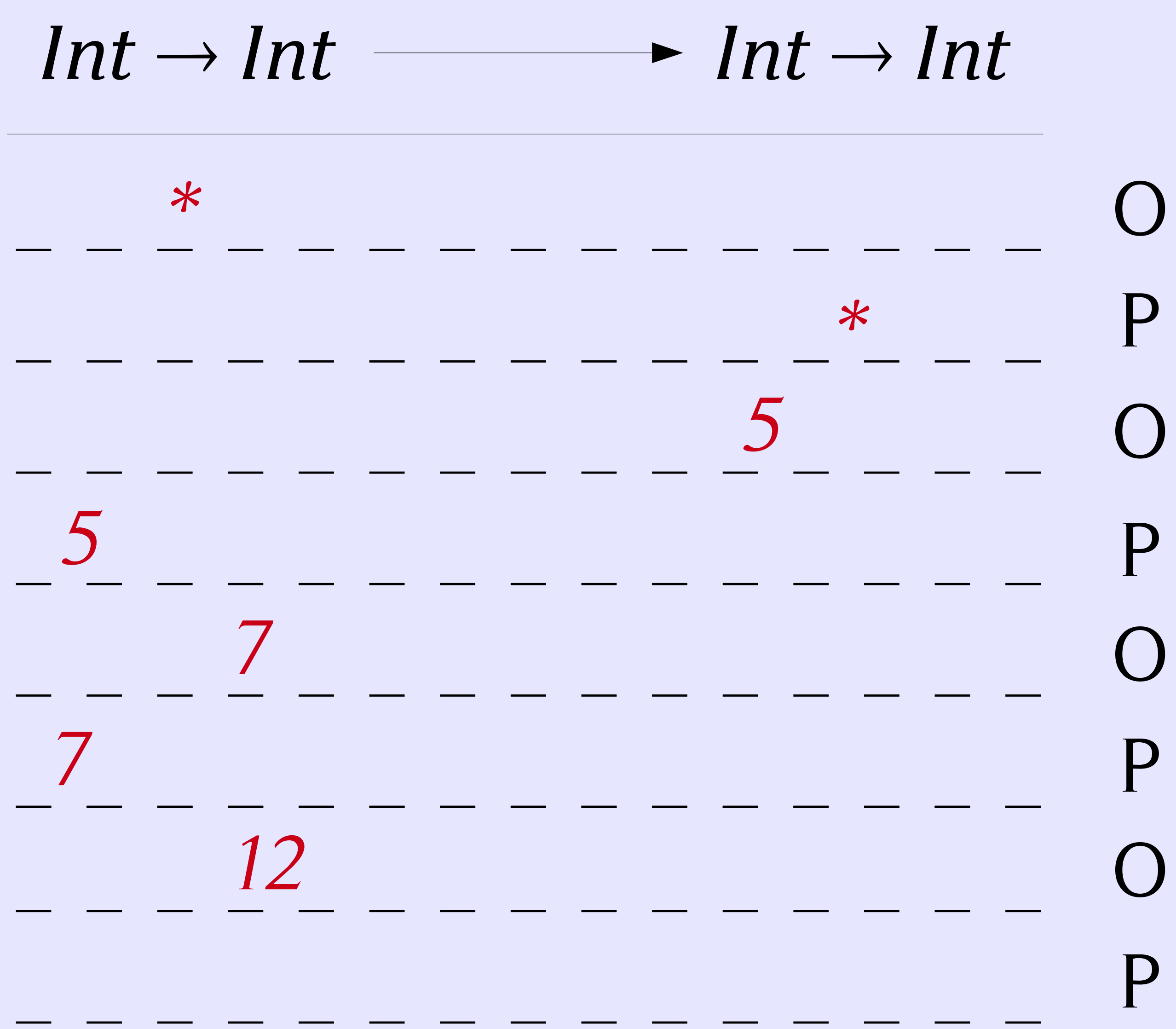
Example



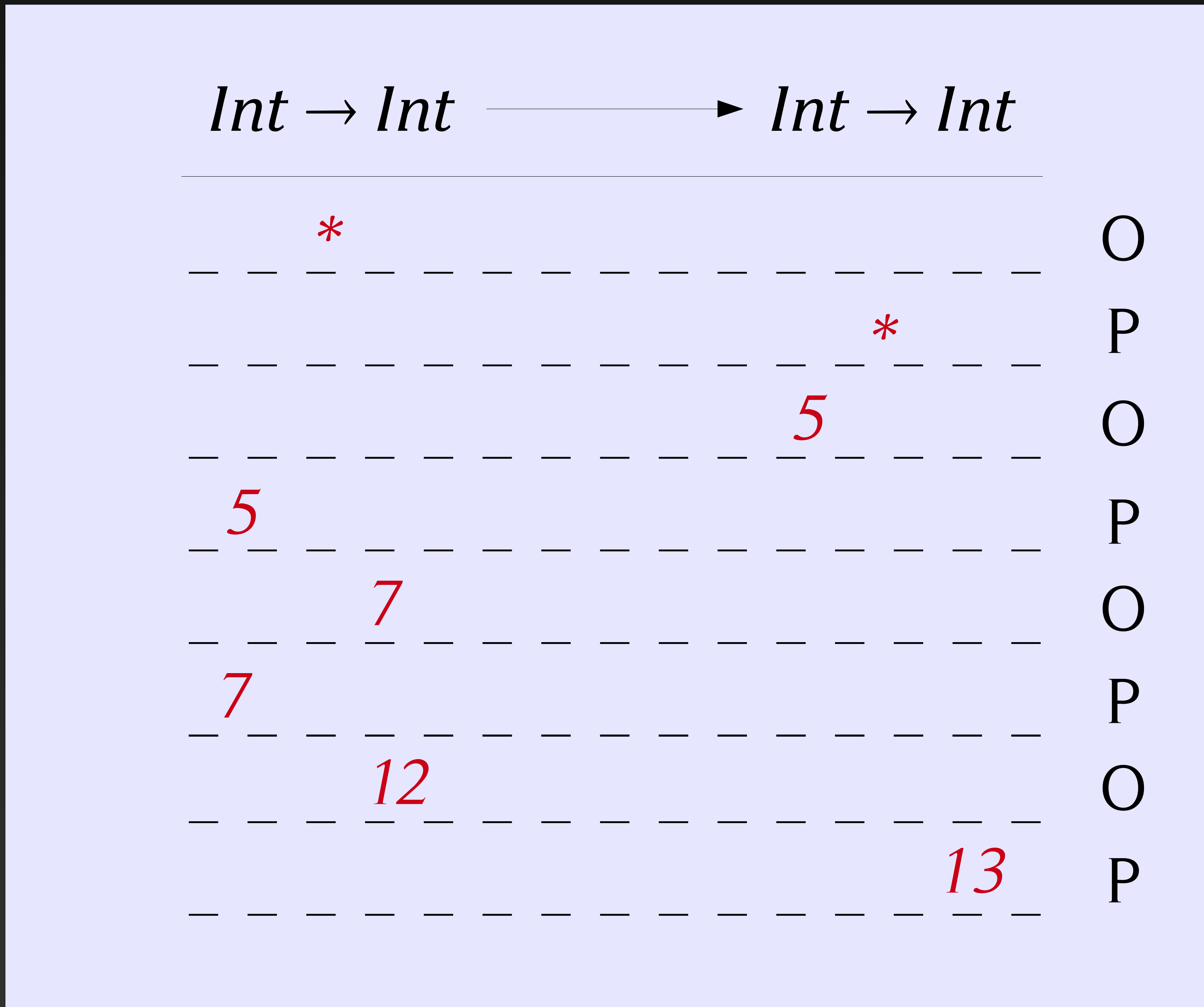
Example



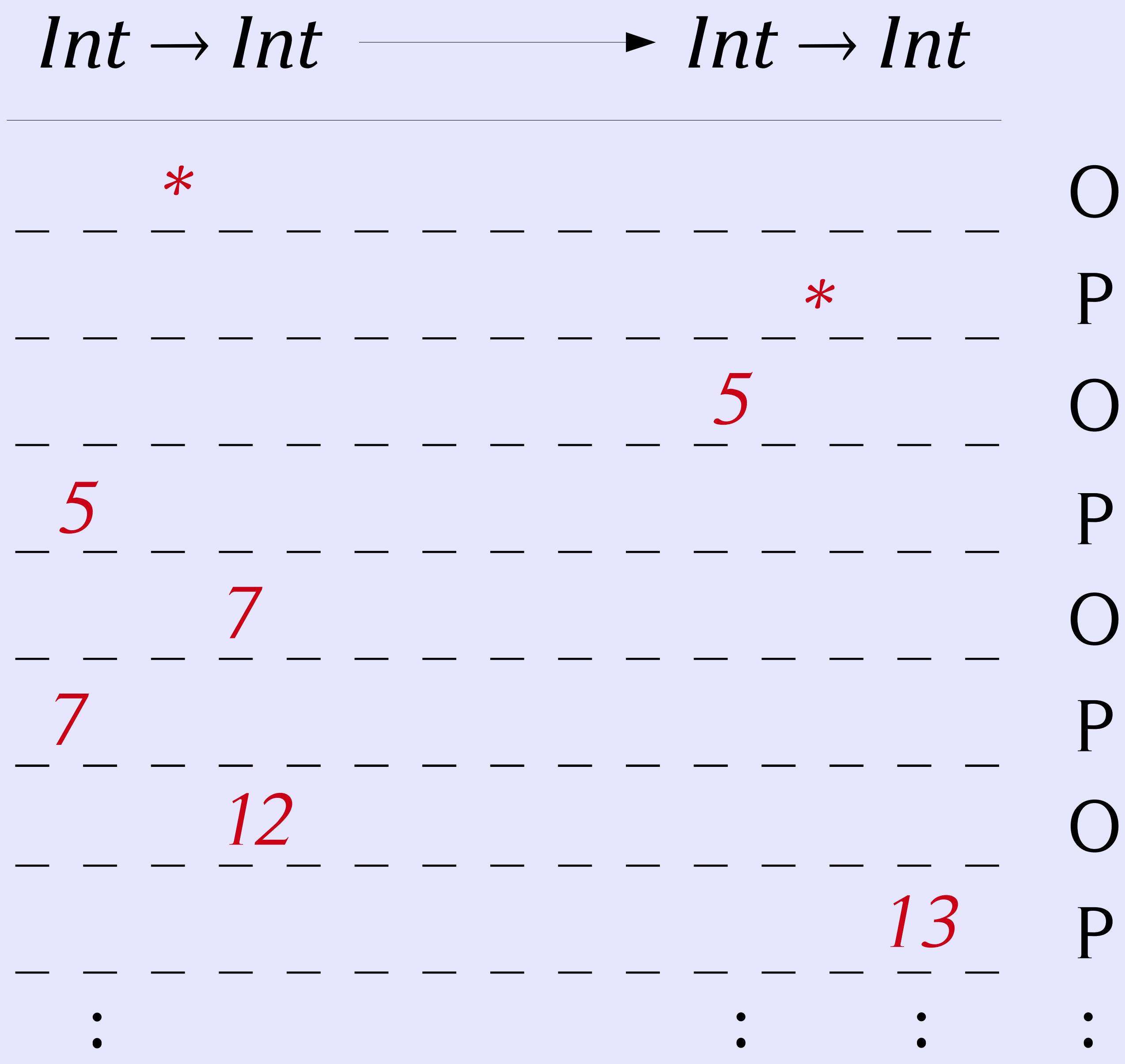
Example



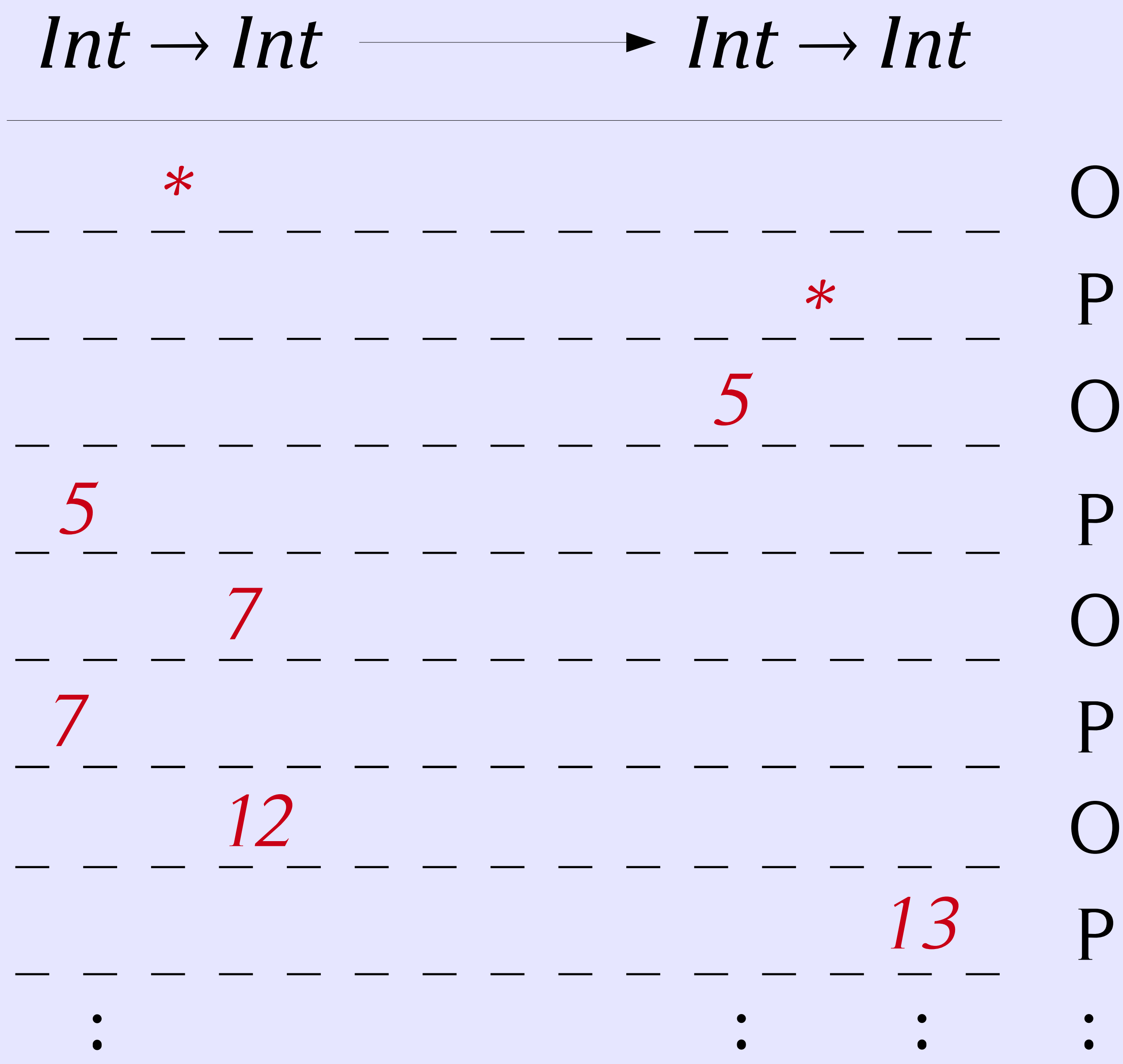
Example



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

$\vdash$ `λx.x+1 : int ⟶ int`

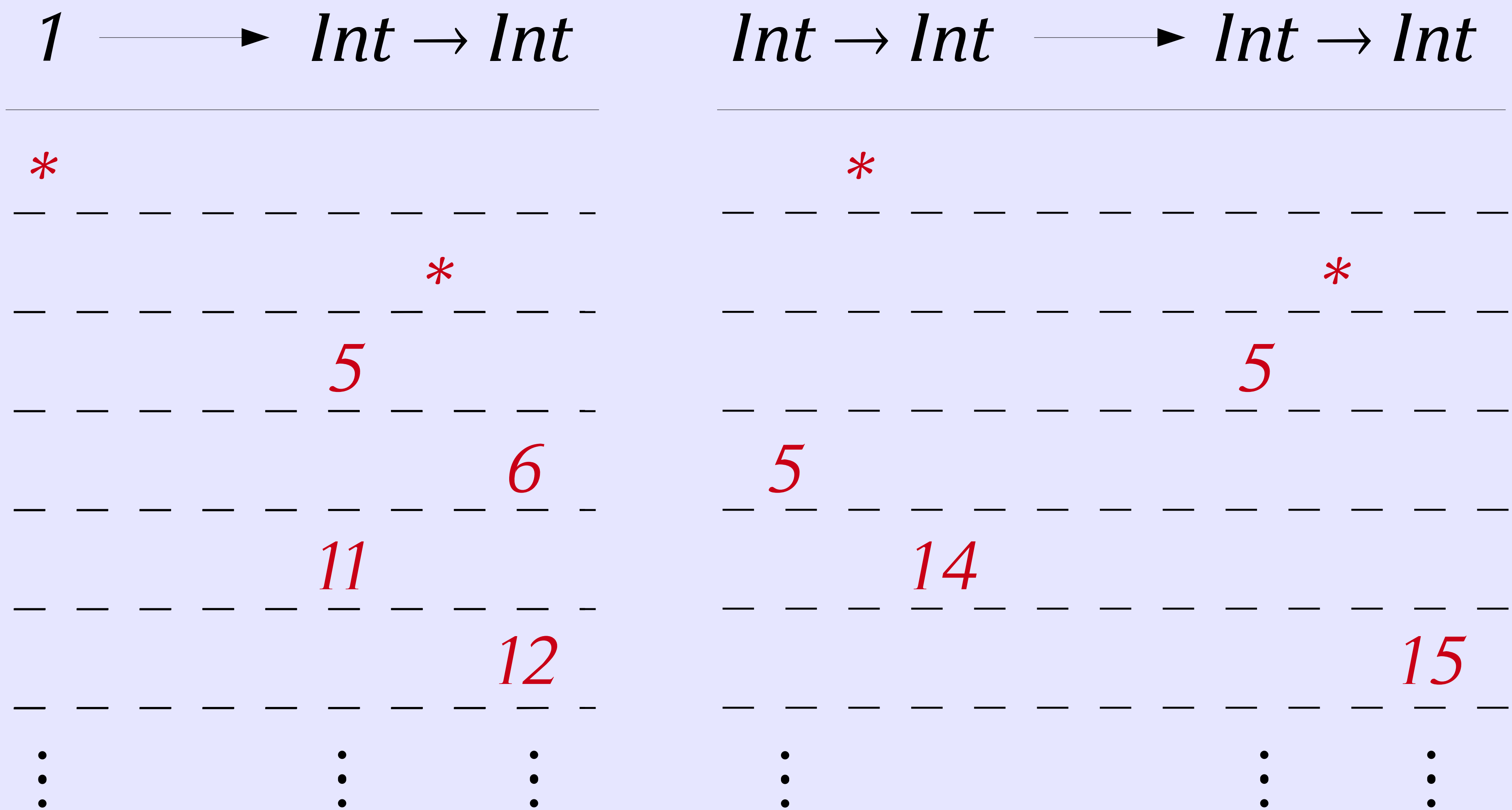
`f : int ⟶ int` $\vdash$ `λx.f(x)+1 : int ⟶ int`

Composition

$\vdash$ `λx.x+1 : int ⟶ int`

`f : int ⟶ int` $\vdash$ `λx.f(x)+1 : int ⟶ int`

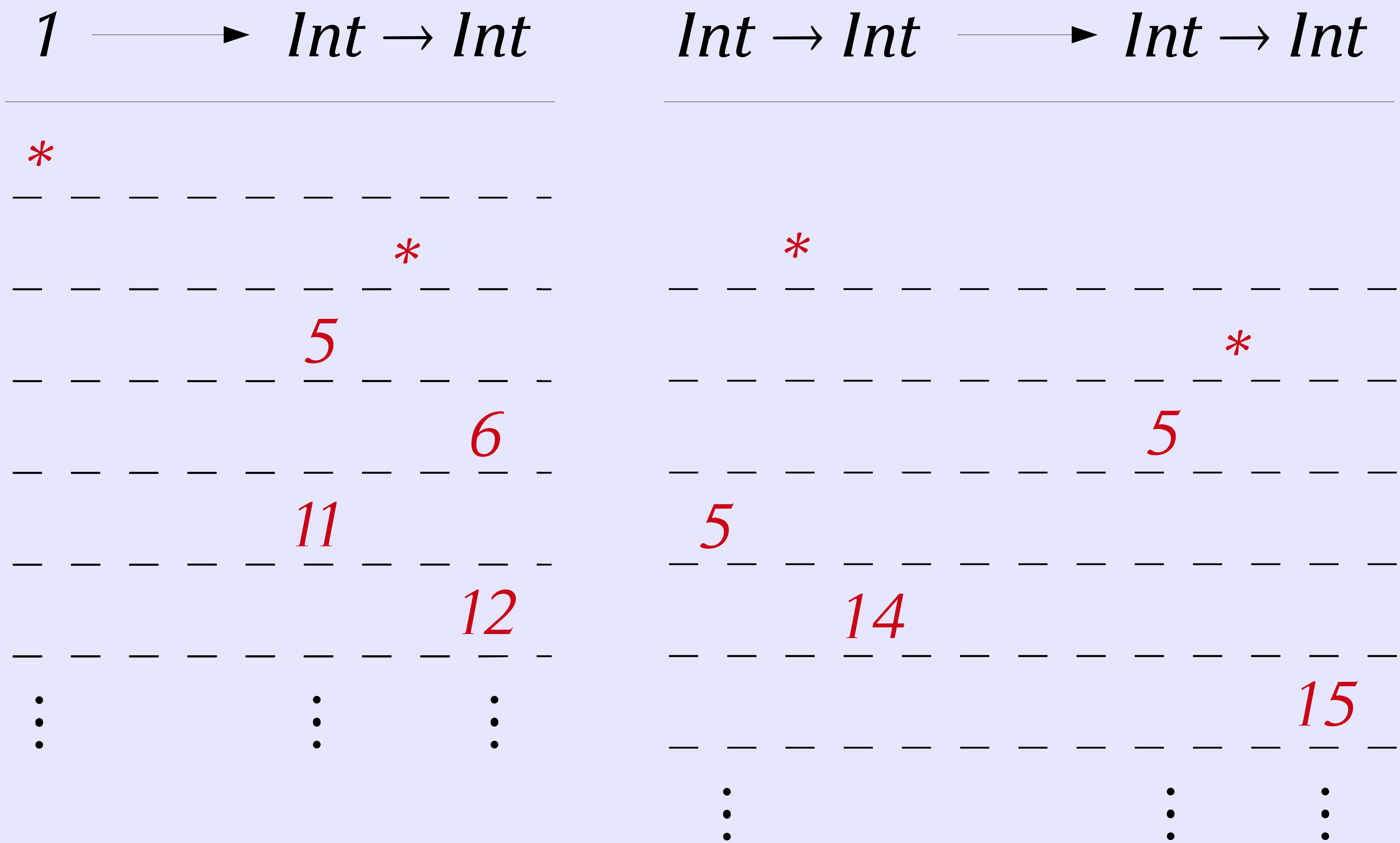
Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

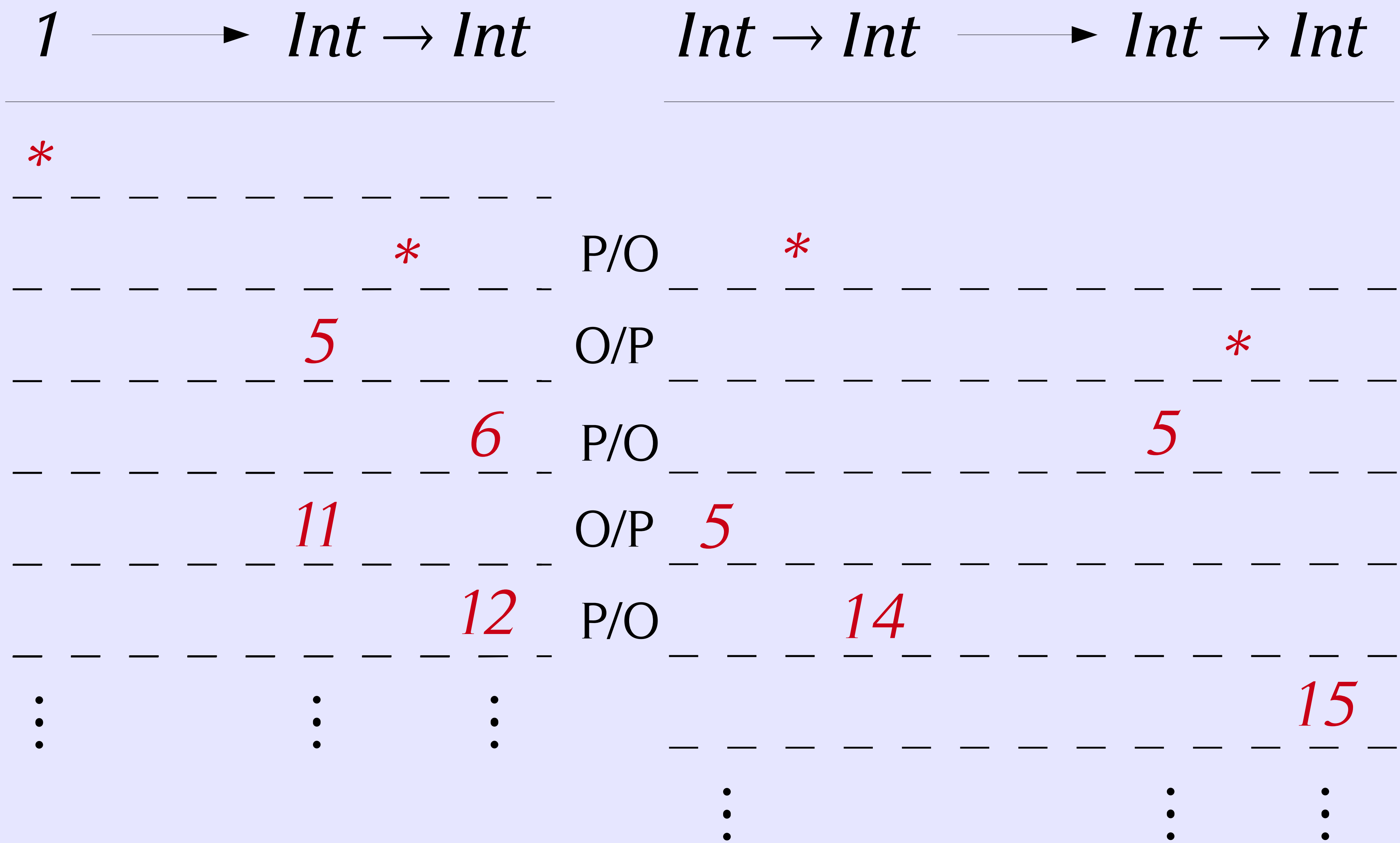
Composition



`⊢ λx.x+1 : int ⟶ int`

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

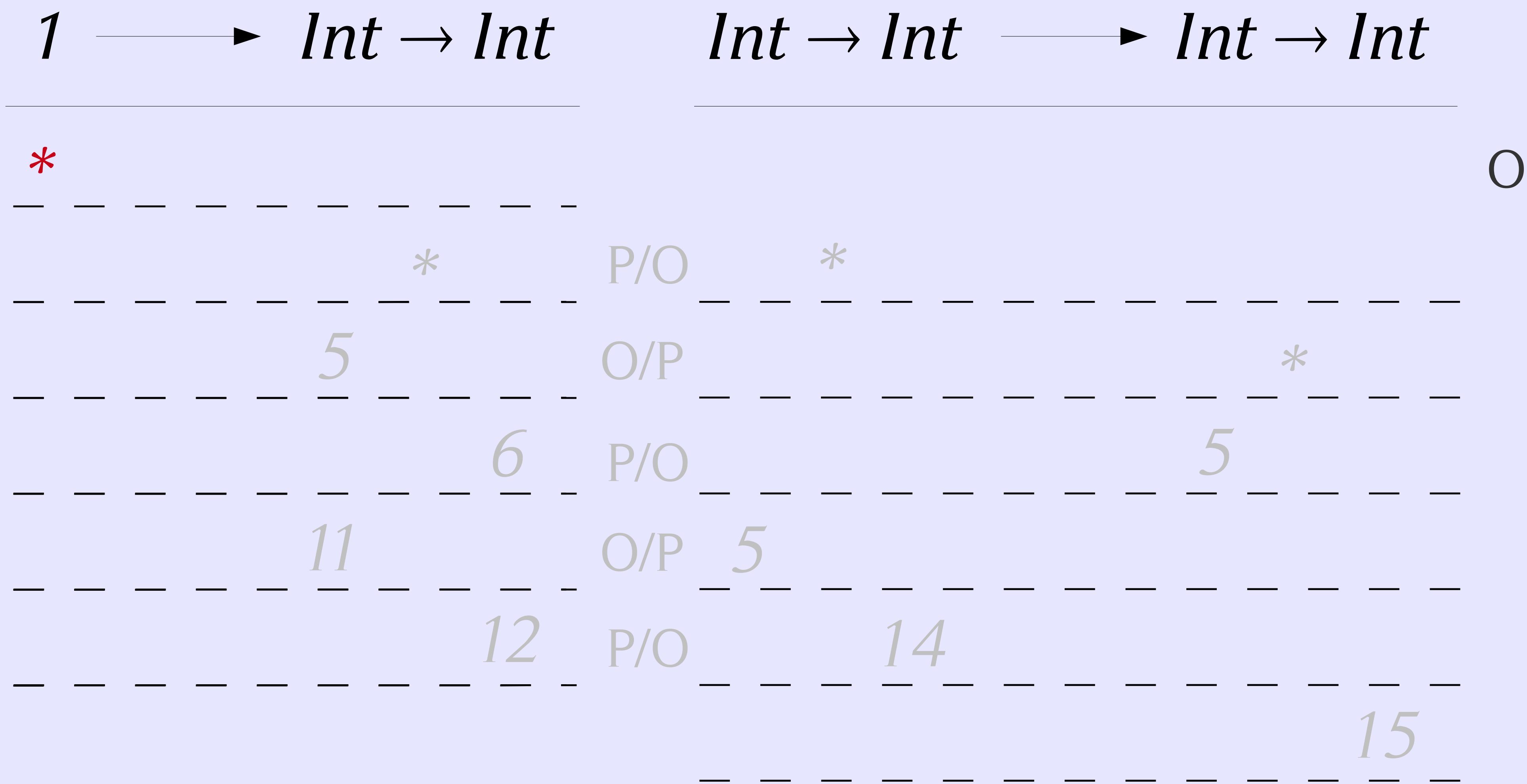
Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

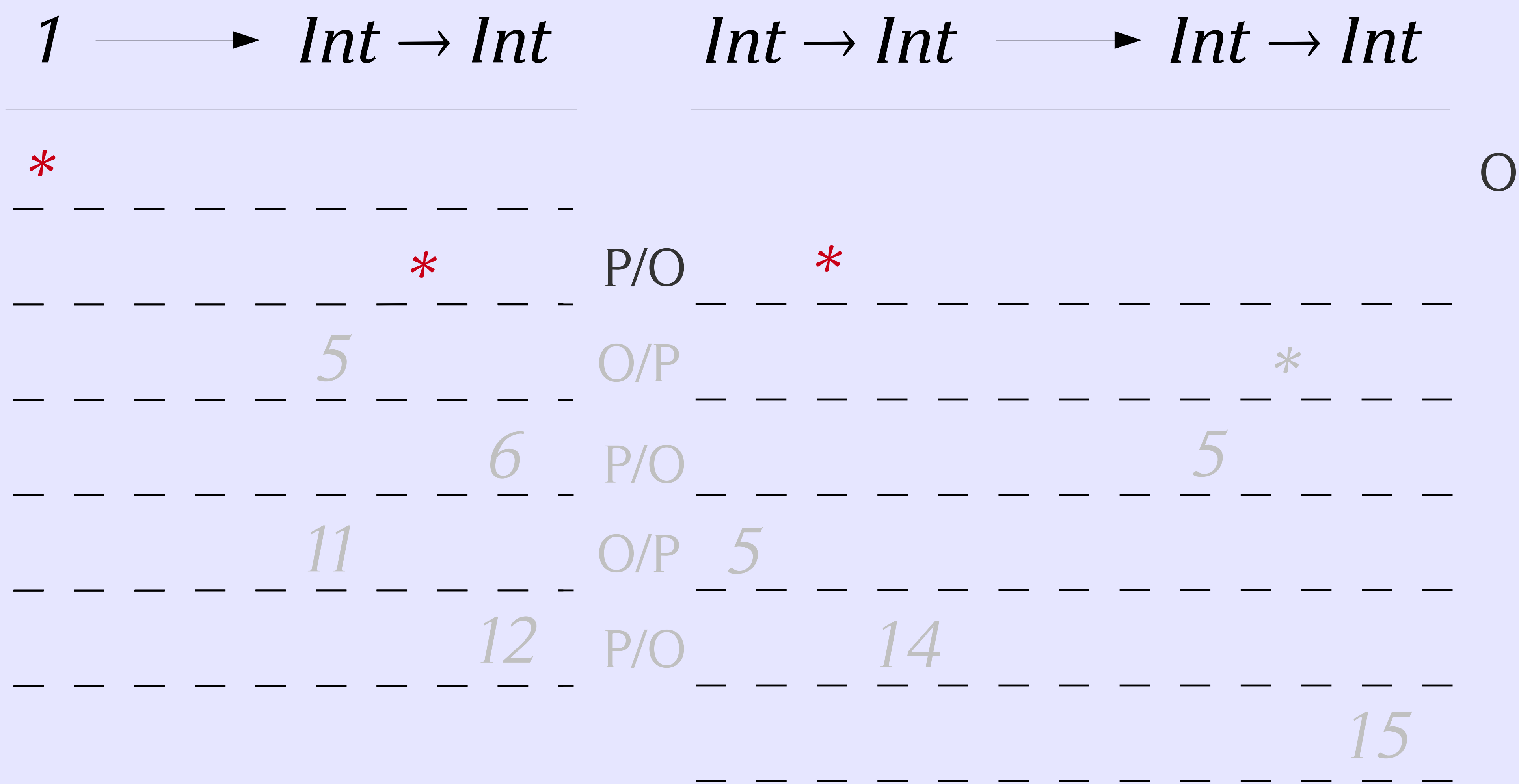
Composition



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

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

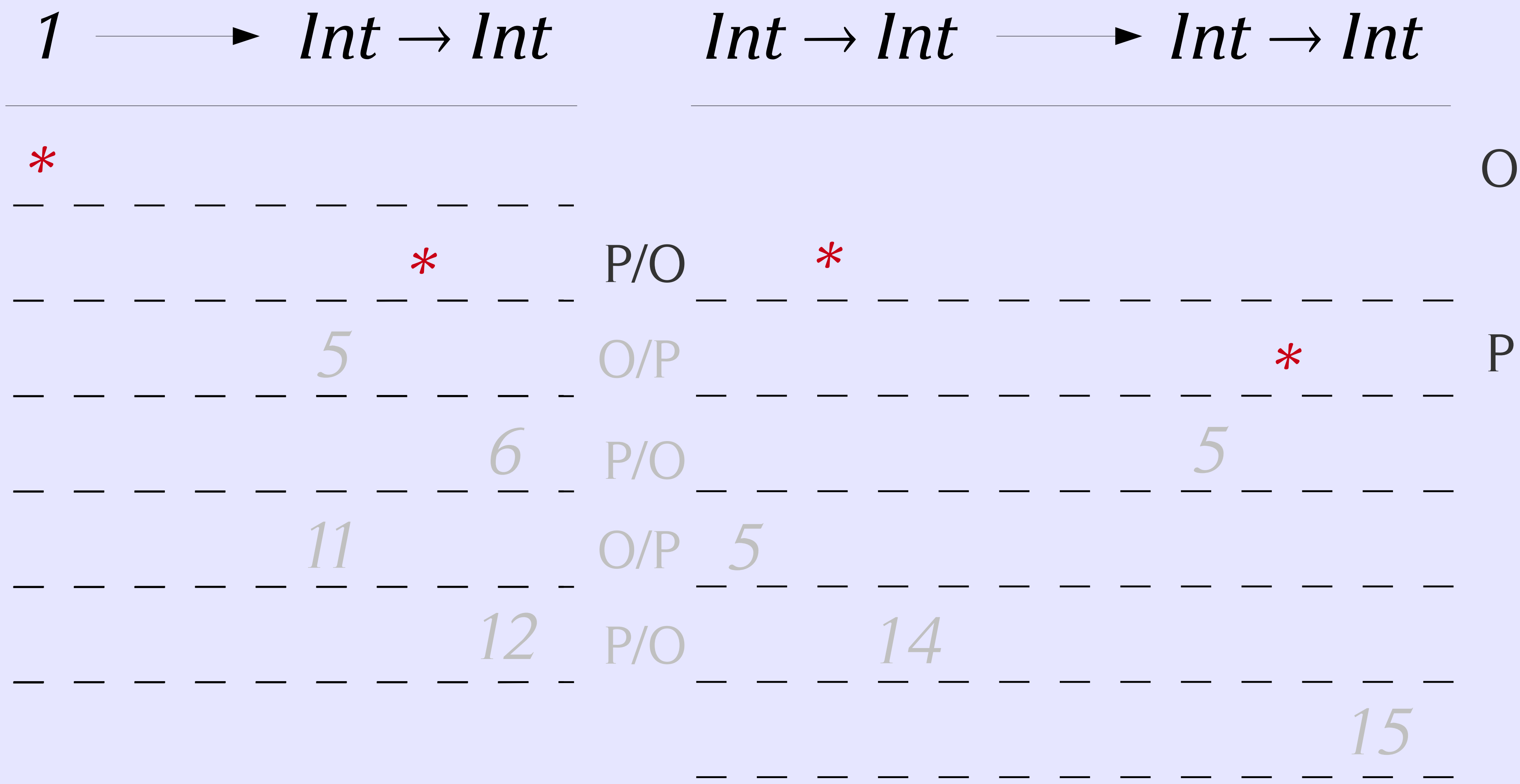
Composition



`⊢ λx.x+1 : int ⟶ int`

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

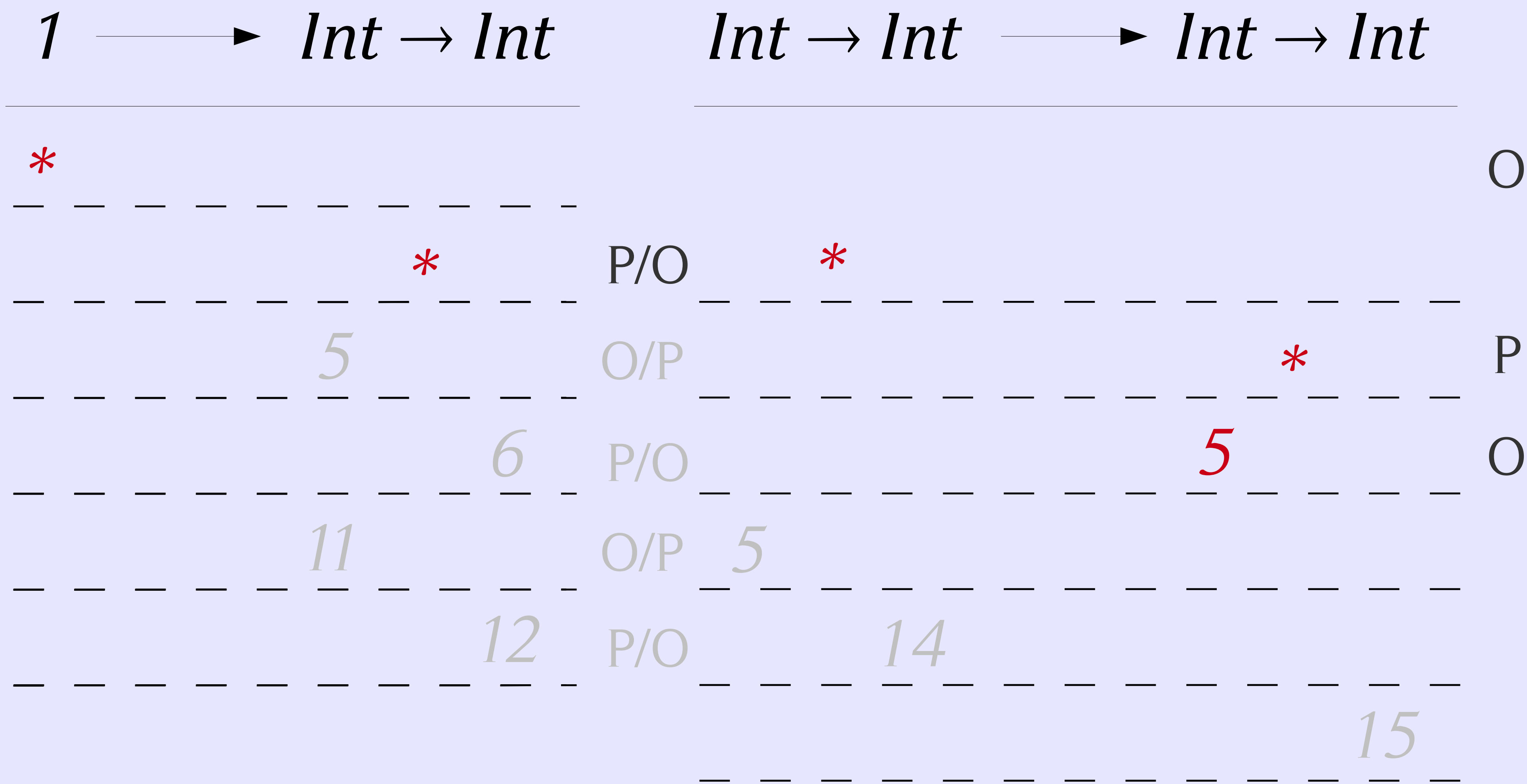
Composition



⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

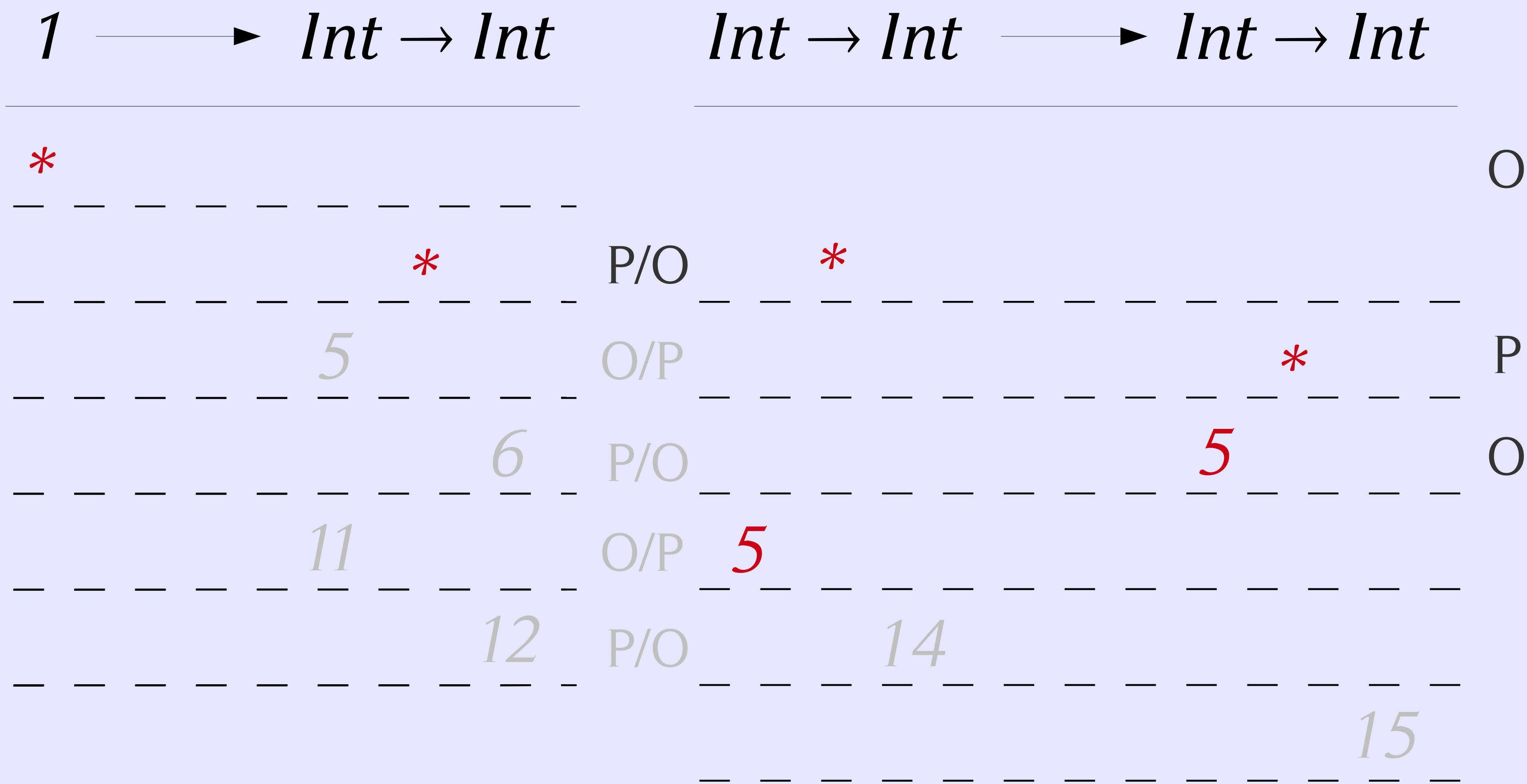
Composition



`⊢ λx.x+1 : int ⟶ int`

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

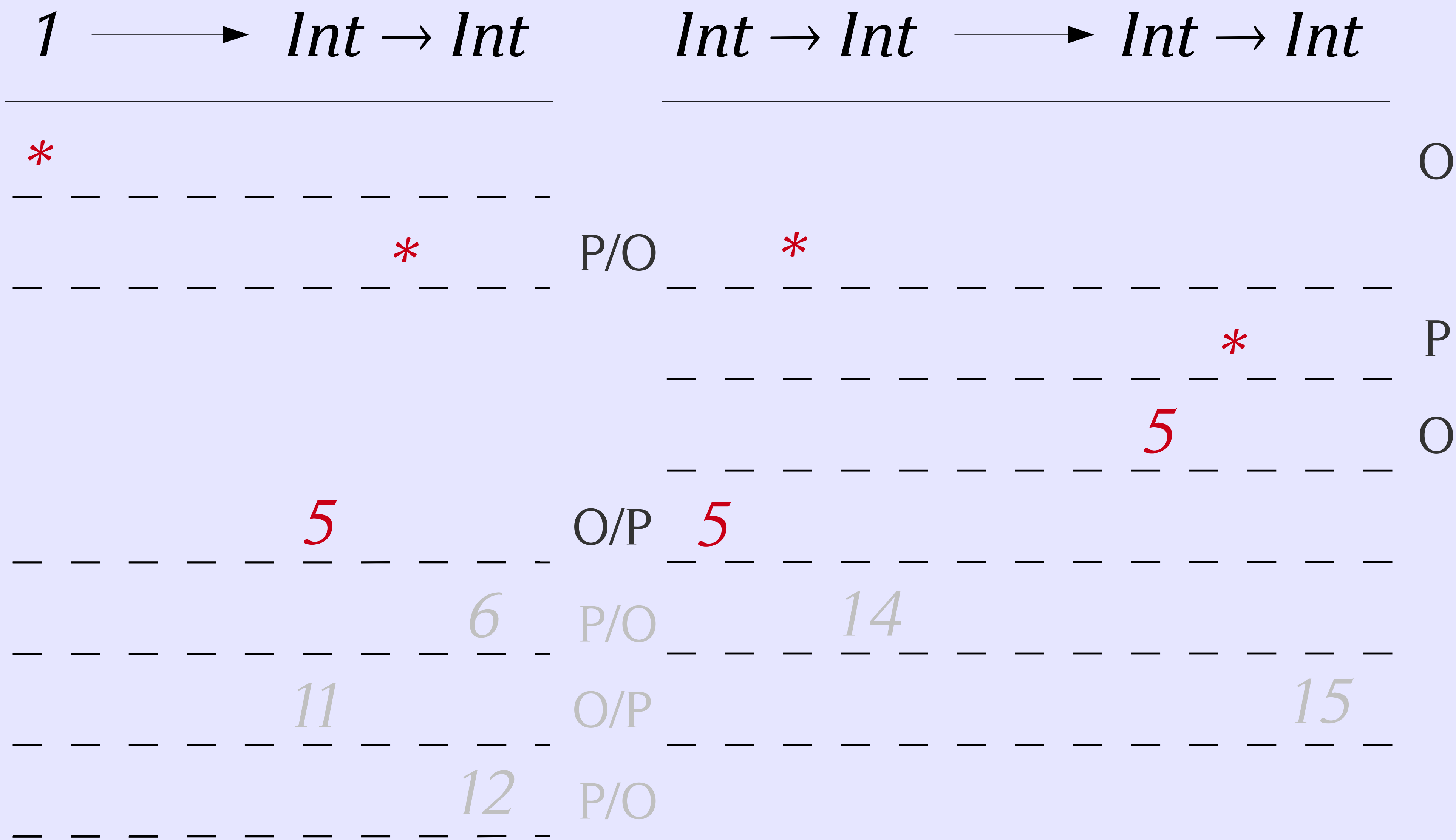
Composition



`⊢ λx.x+1 : int ⟶ int`

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

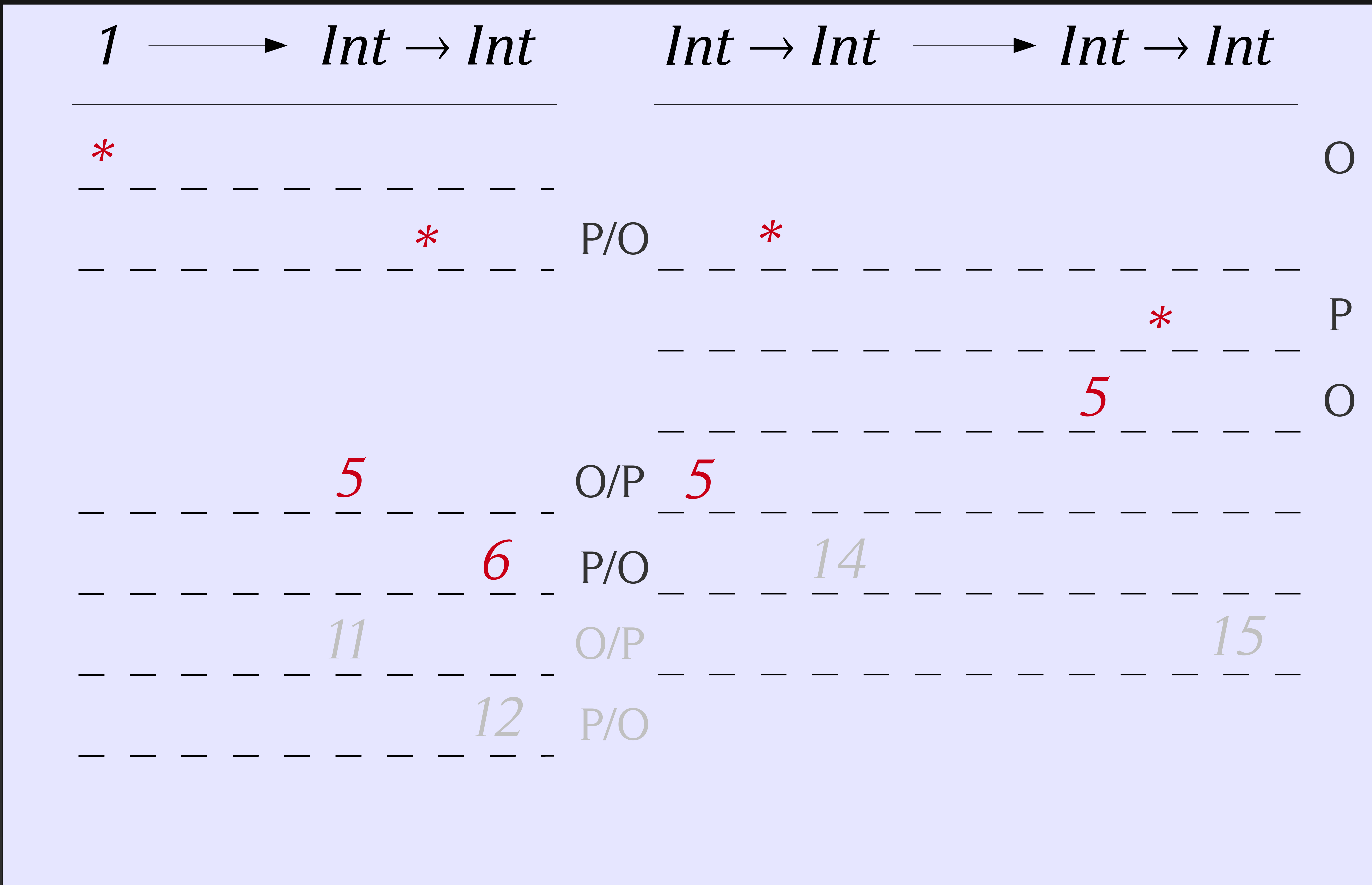
Composition



`⊢ λx.x+1 : int ⟶ int`

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

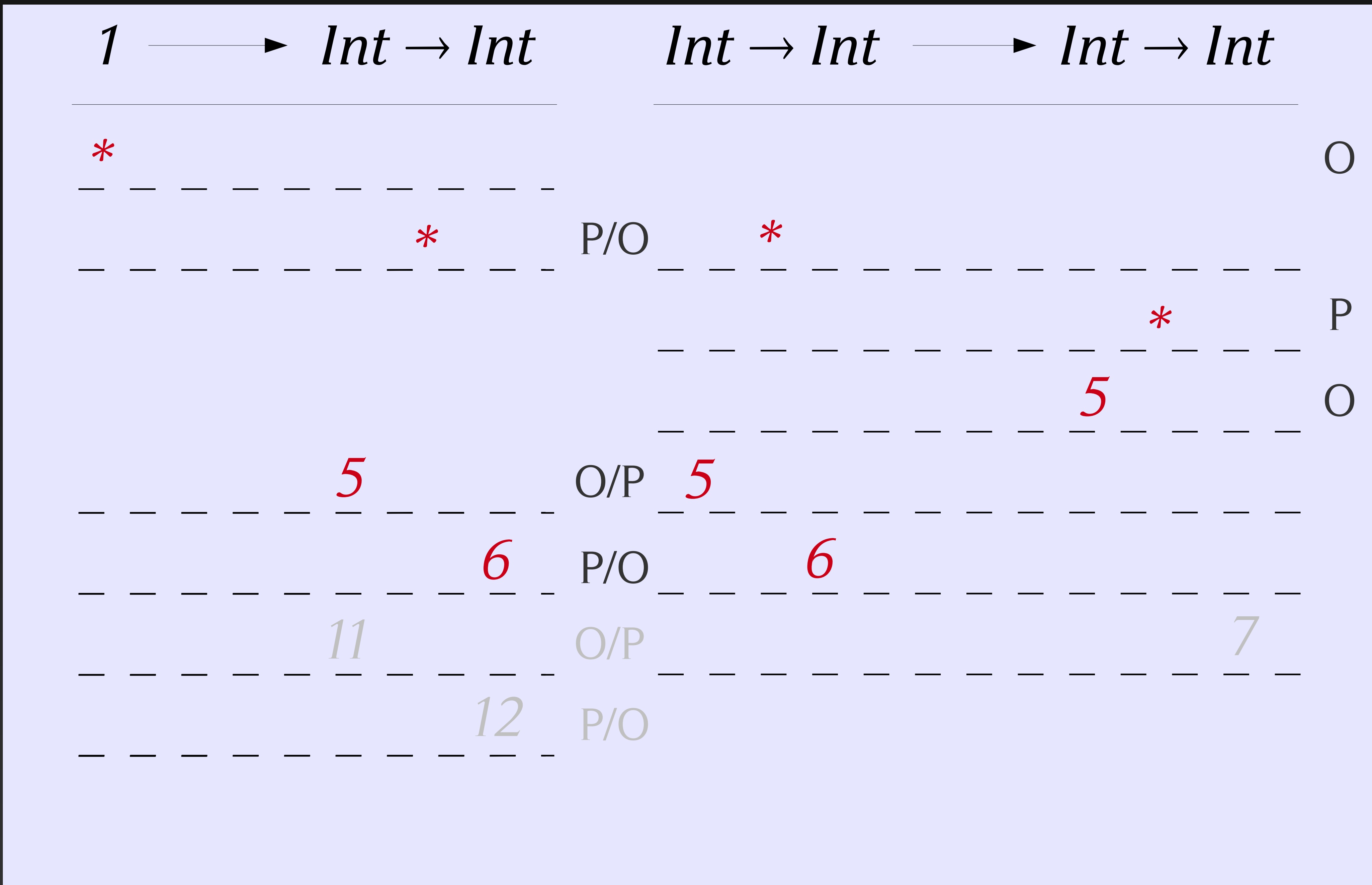
Composition



`⊢ λx.x+1 : int ⟶ int`

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

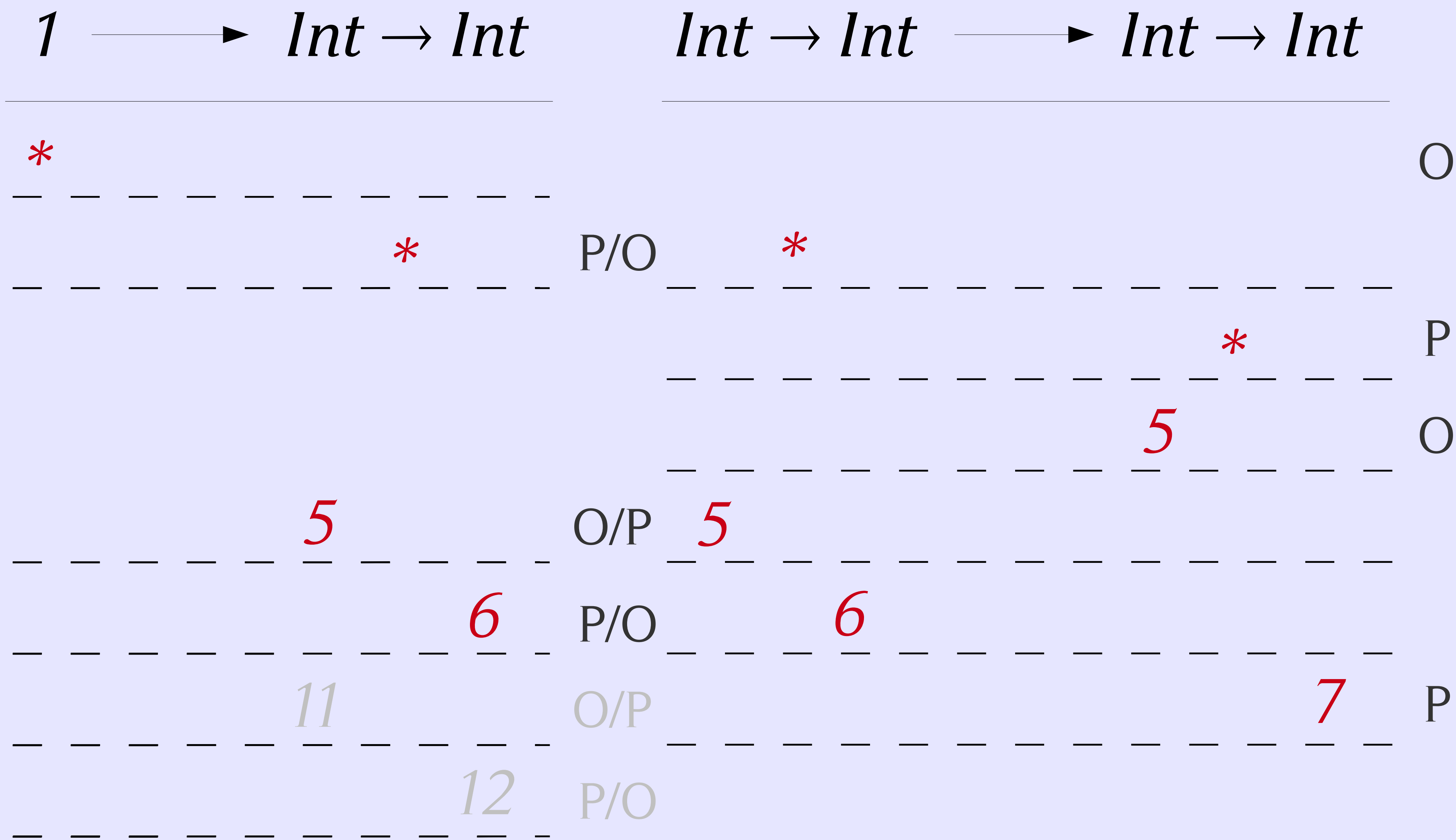
Composition



`⊢ λx.x+1 : int ⟶ int`

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

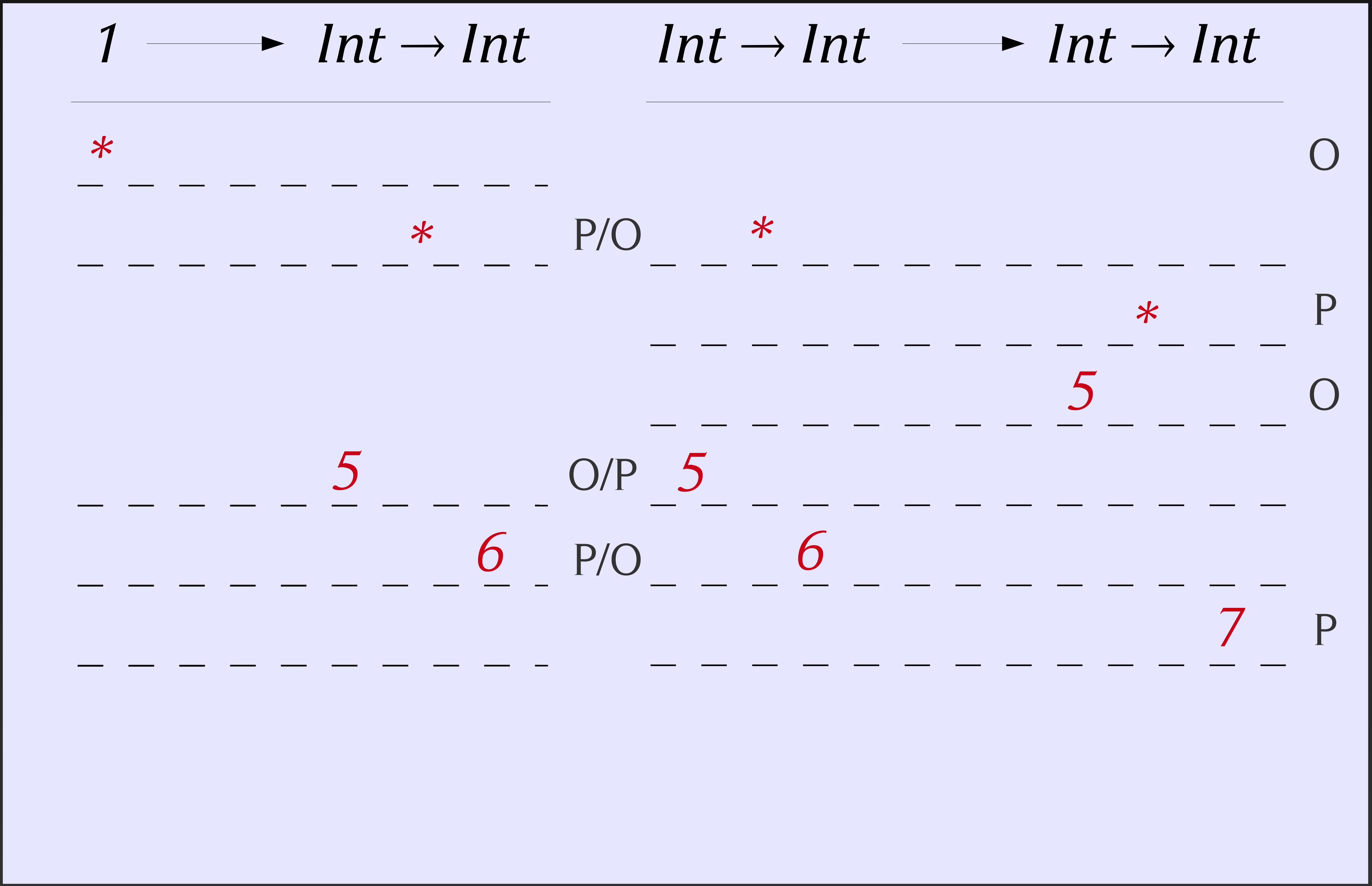
Composition



`⊢ λx.x+1 : int ⟶ int`

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

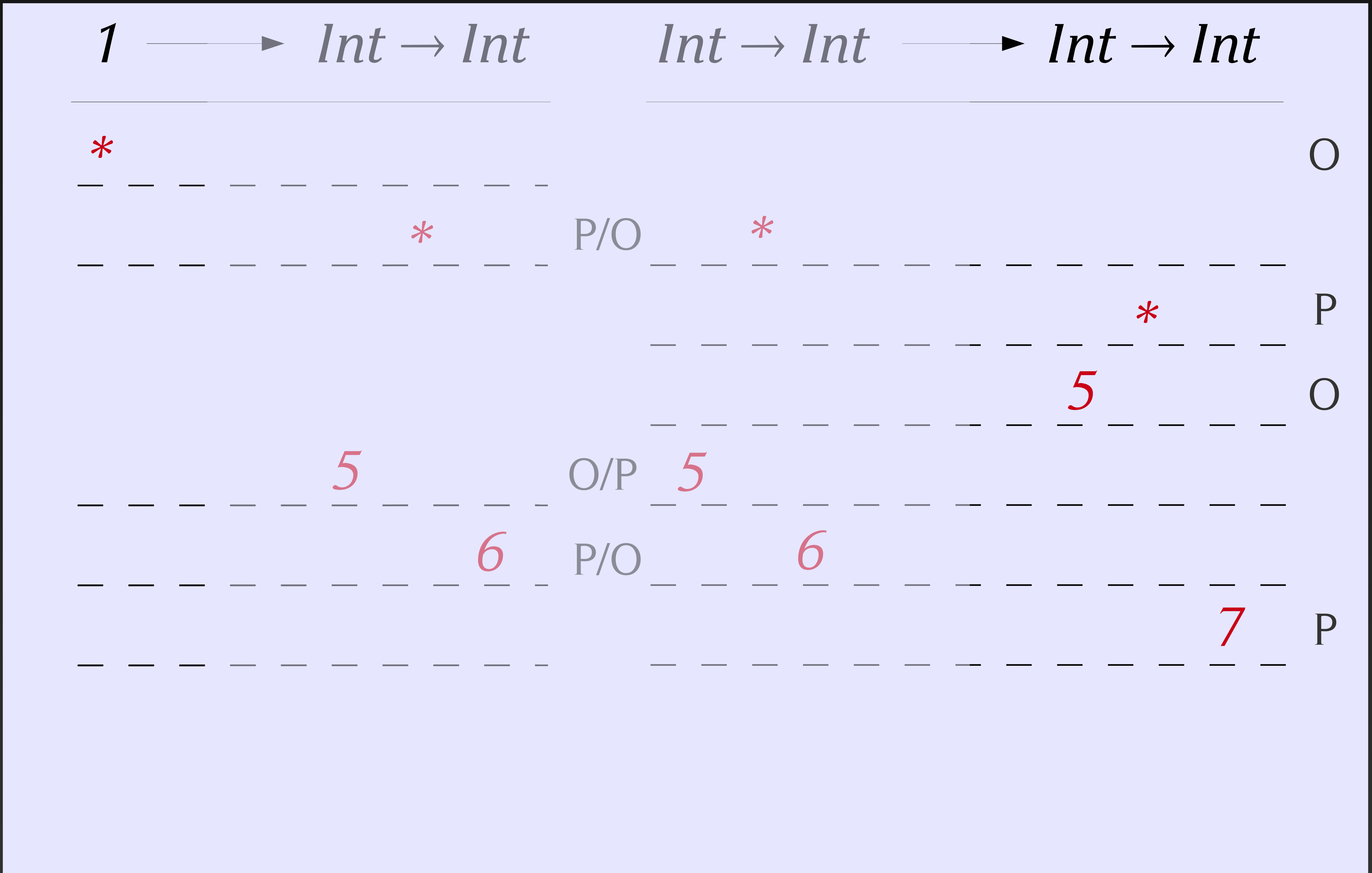
Composition



$\vdash$ `λx.x+1 : int ⟶ int`

`f : int ⟶ int` $\vdash$ `λx.f(x)+1 : int ⟶ int`

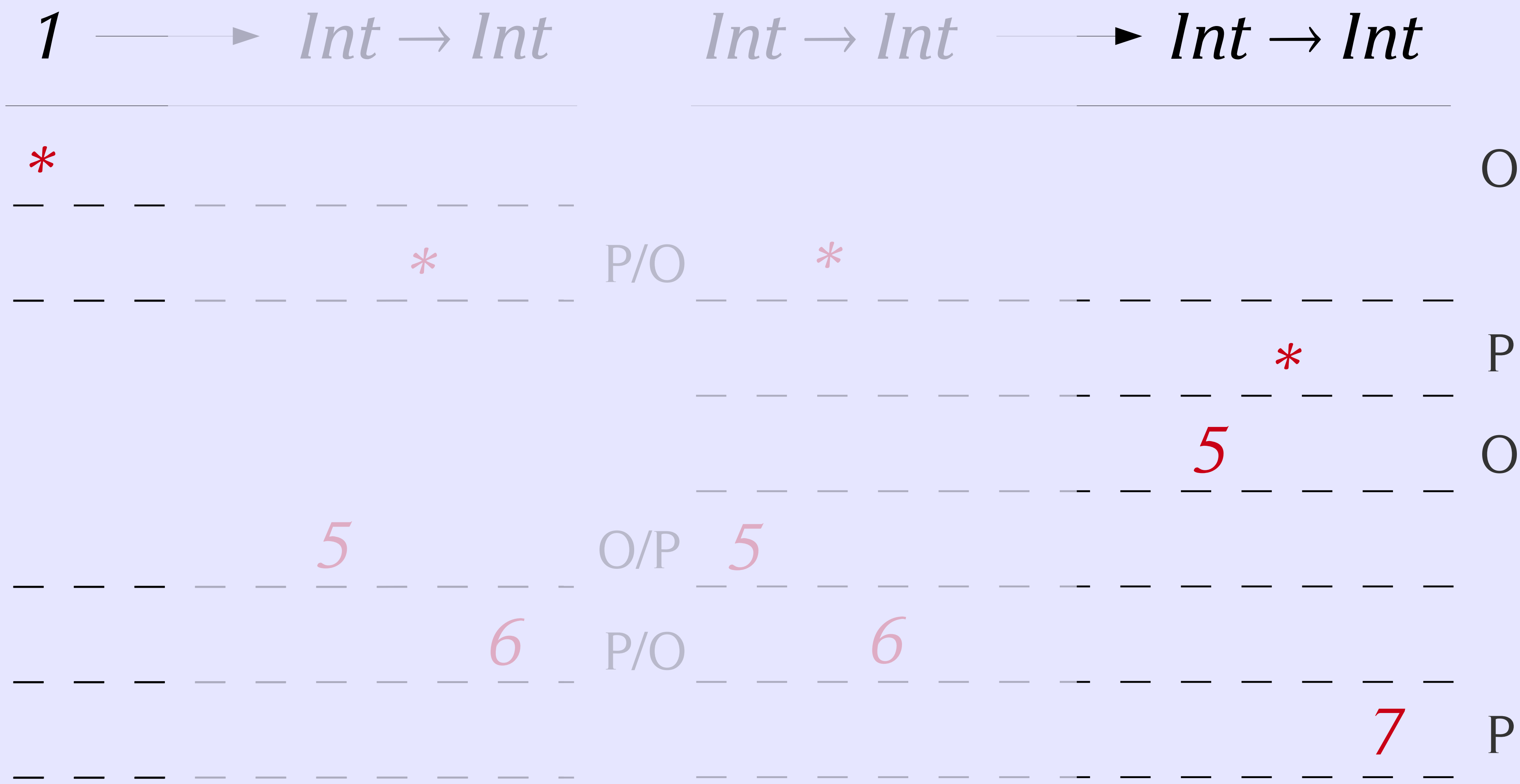
Composition



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

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

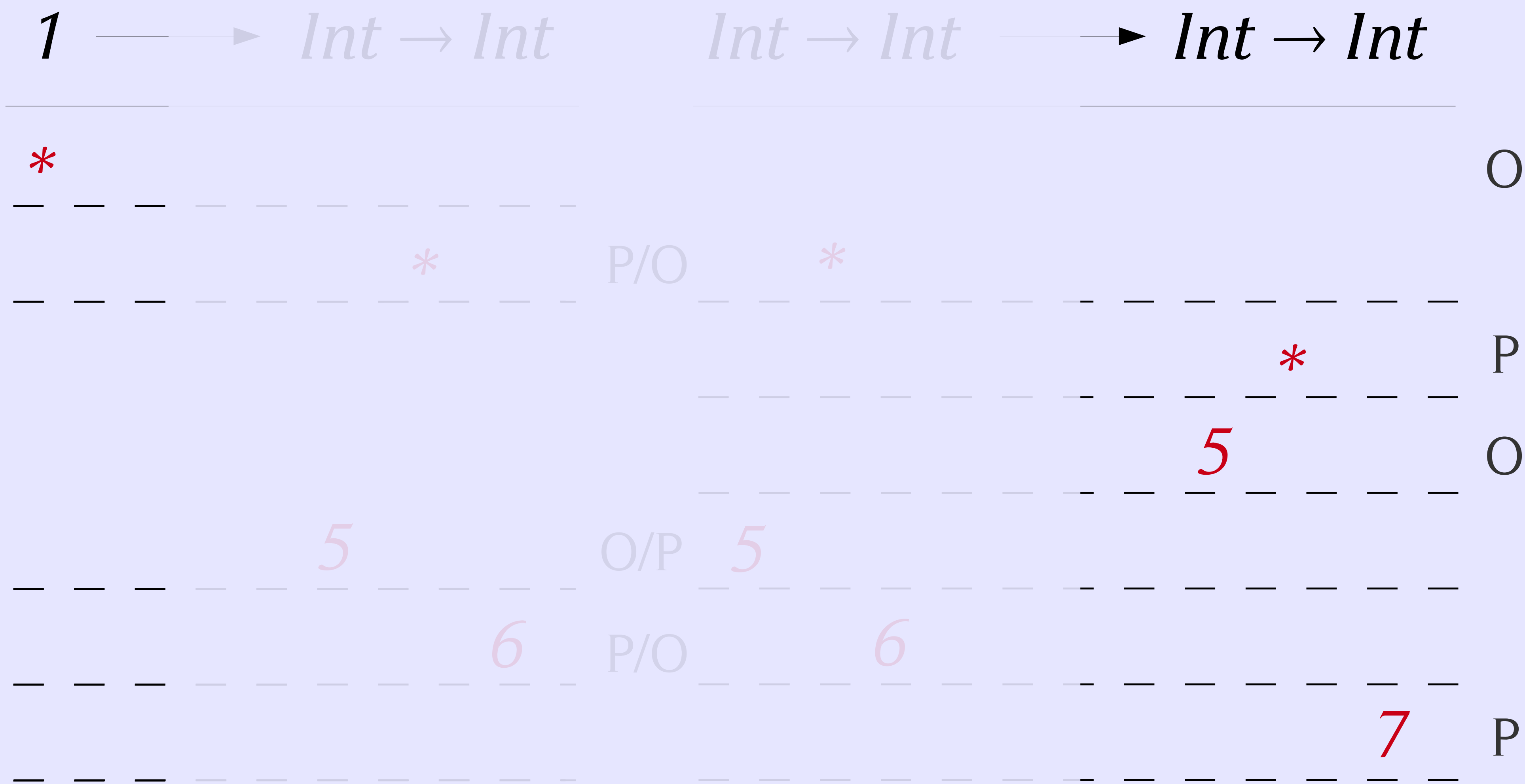
Composition



`⊢ λx.x+1 : int ⟶ int`

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

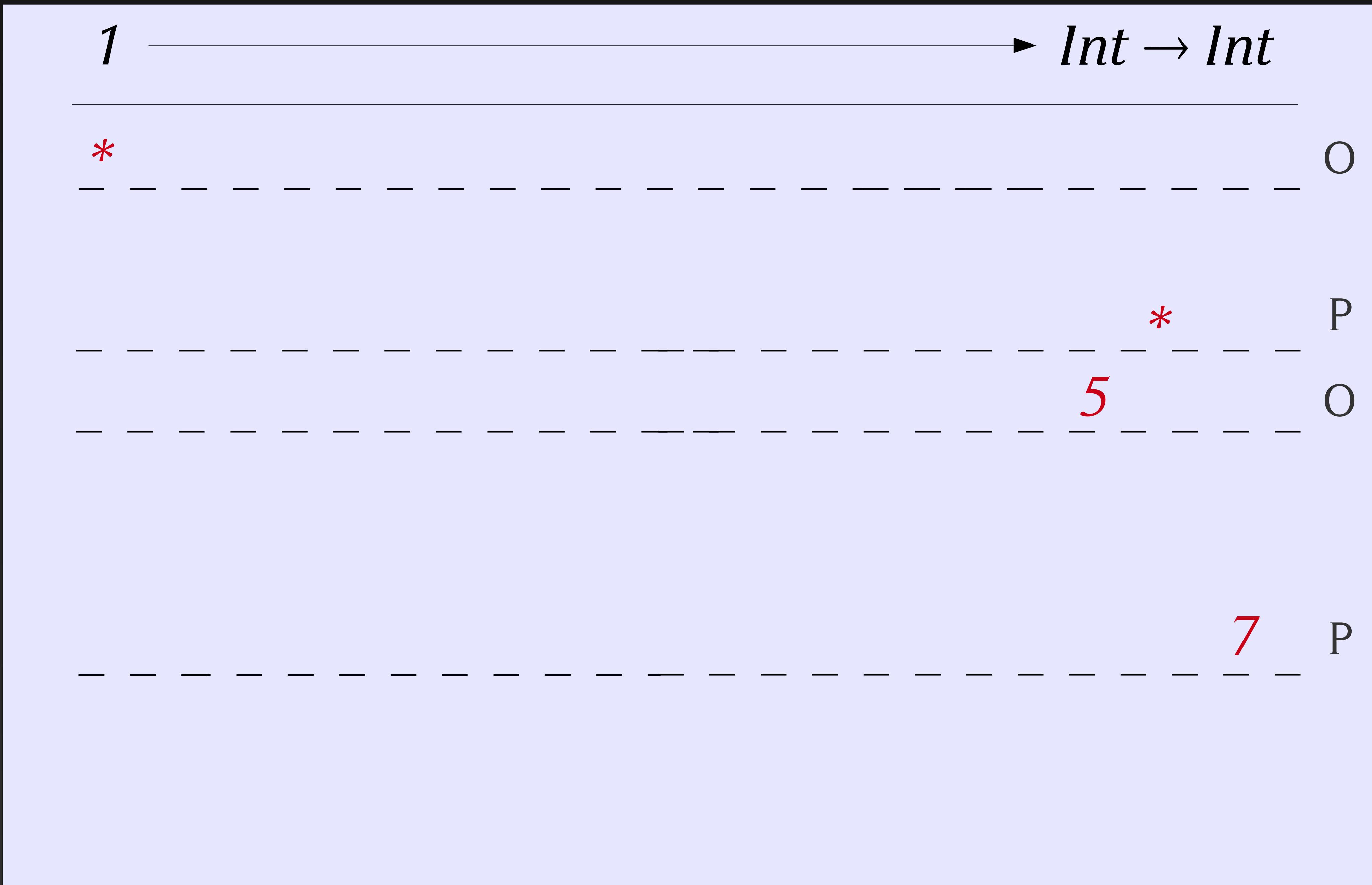
Composition



`⊢ λx.x+1 : int ⟶ int`

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

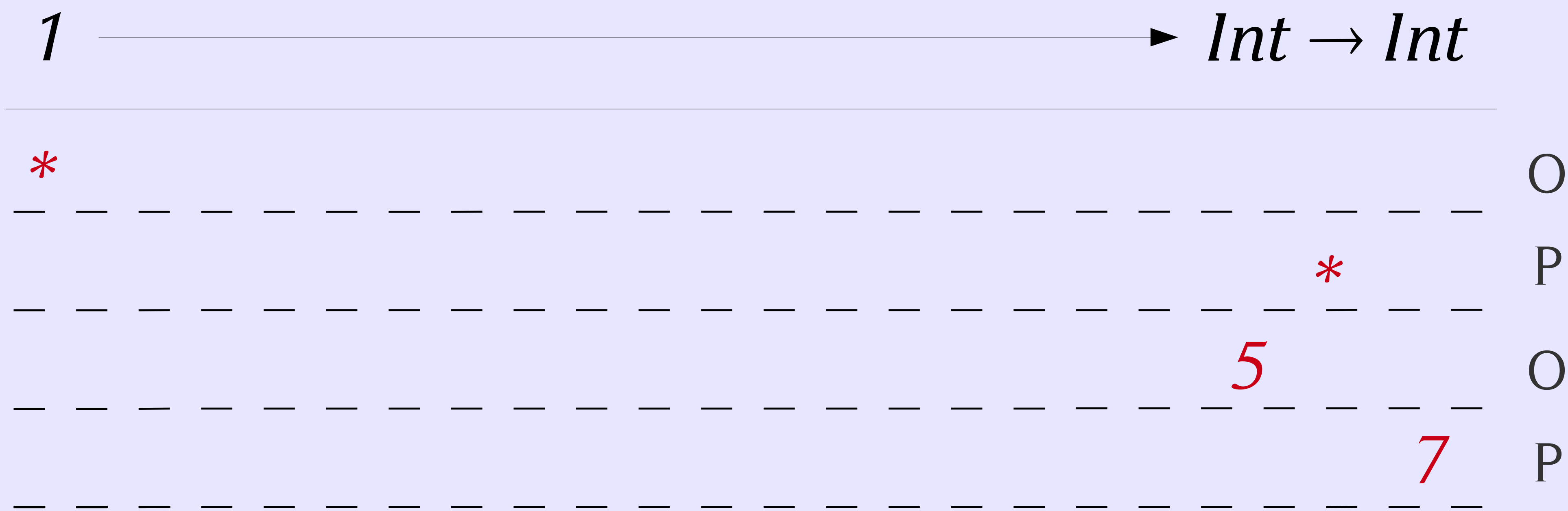
Composition



$\vdash$ λx.x+1 : int ⟶ int

f : int ⟶ int $\vdash$ λx.f(x)+1 : int ⟶ int

Composition



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

Composition

$$1 \longrightarrow Int \to Int$$

*		O
	*	P
	5	O
	7	P

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

= `⊢ λx.x+2 : int ⟶ int`

⊢ λx.x+1 : int ⟶ int

f : int ⟶ int ⊢ λx.f(x)+1 : int ⟶ int

Composition

$$1 \longrightarrow Int \to Int$$

1	$Int \to Int$	
*		O
	*	P
	5	O
	7	P

⊢ 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 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)

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

Road 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

Road 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?

Road to nominal games

Full Abstraction for PCF (early 90's)

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

First stage (1993-2004)

- Models for various programming features
- Program analysis

} realism?

Nominal game semantics (2004-)

Nominal sets

Foundation for doing maths with names and name-binding

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

Nominal sets

Foundation for doing maths with names and name-binding

- 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

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`

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

Nominal games

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`

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

Nominal games

Games in nominal sets

names

$\vdash$ `λx.ref(0)` : `com` → `intref`

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

Examples

$\vdash$ `λx.ref(0)` : `com` $\to$ `intref`

$$1 \longrightarrow 1 \to Ref$$

O

P

O

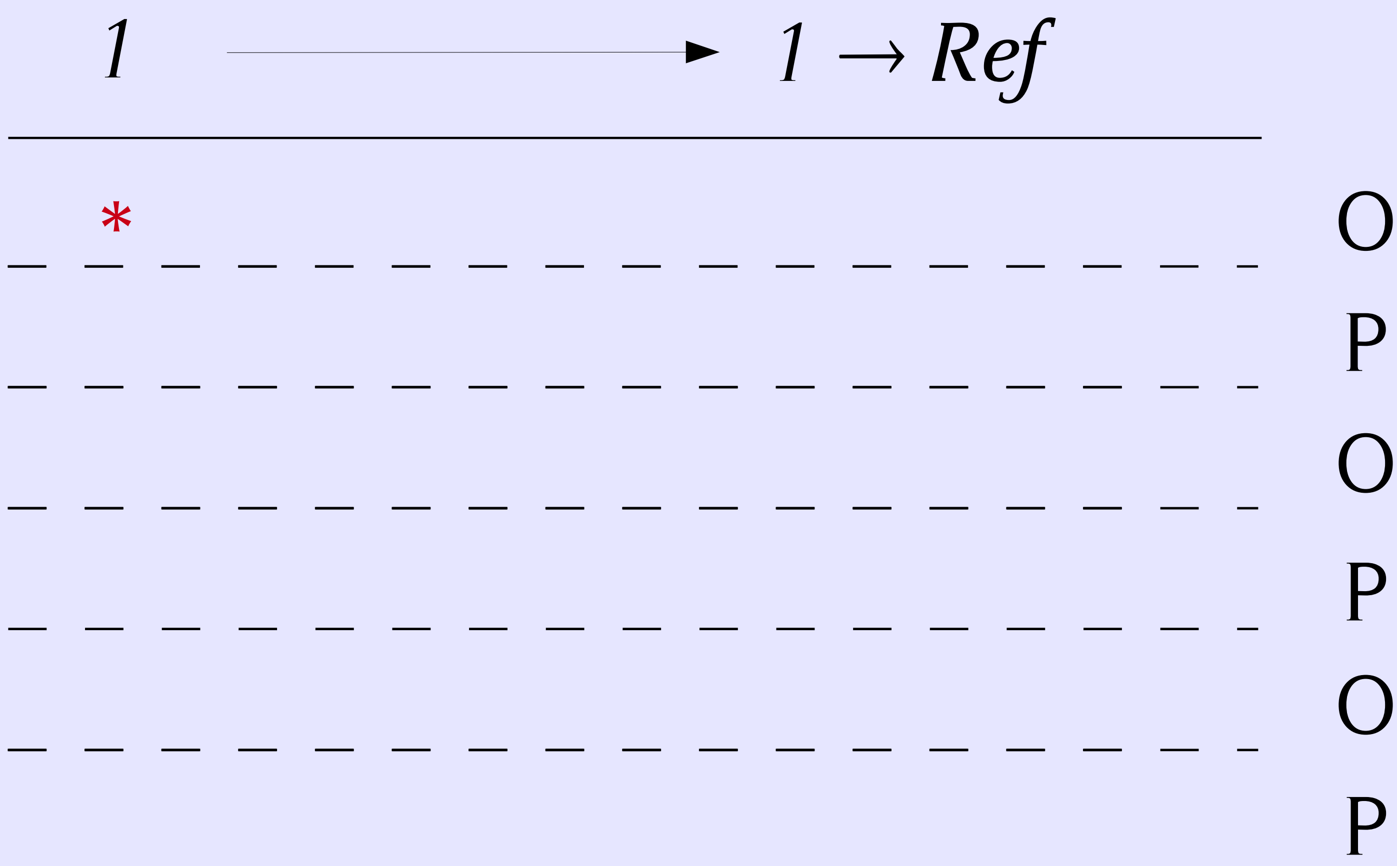
P

O

P

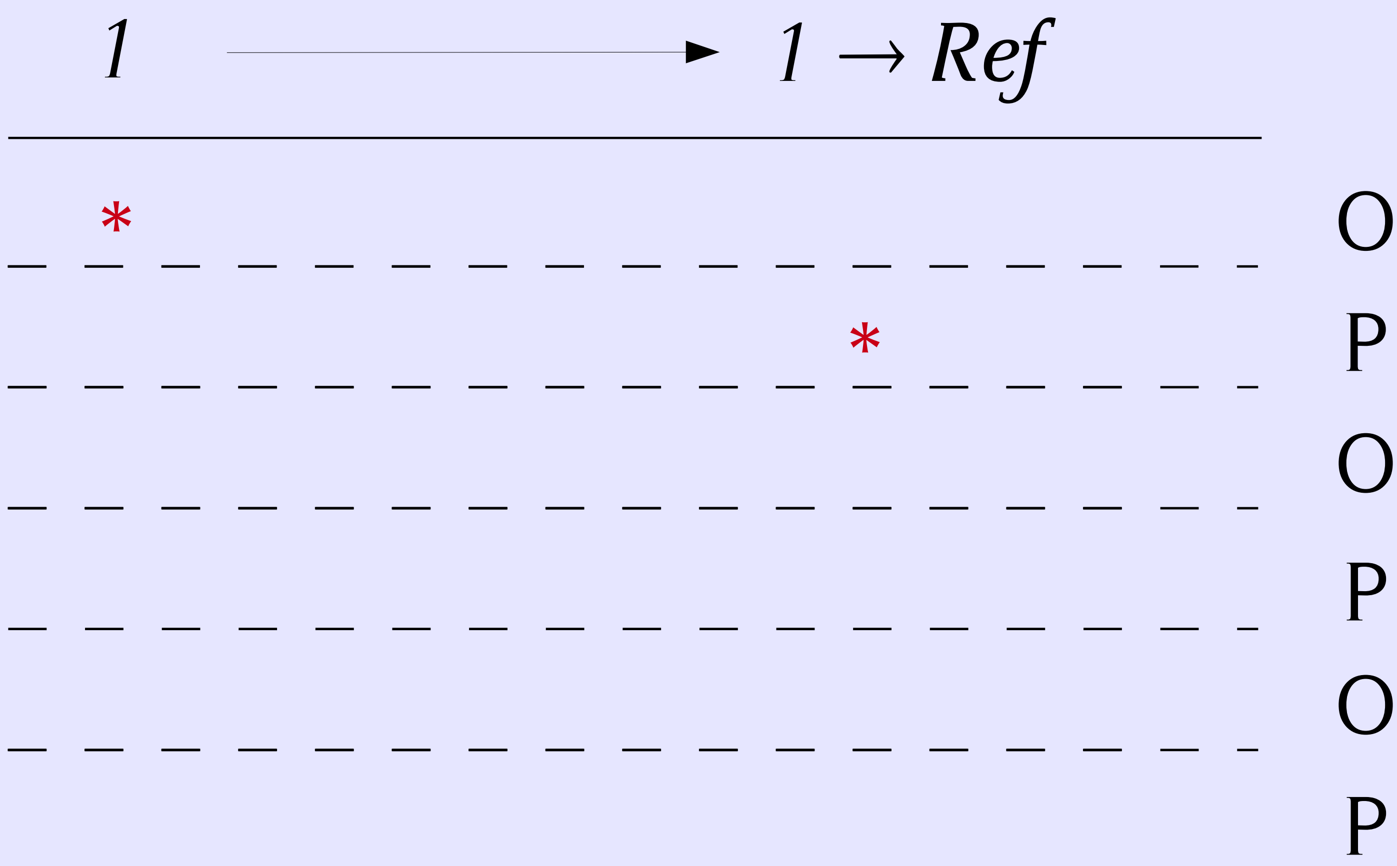
Examples

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`



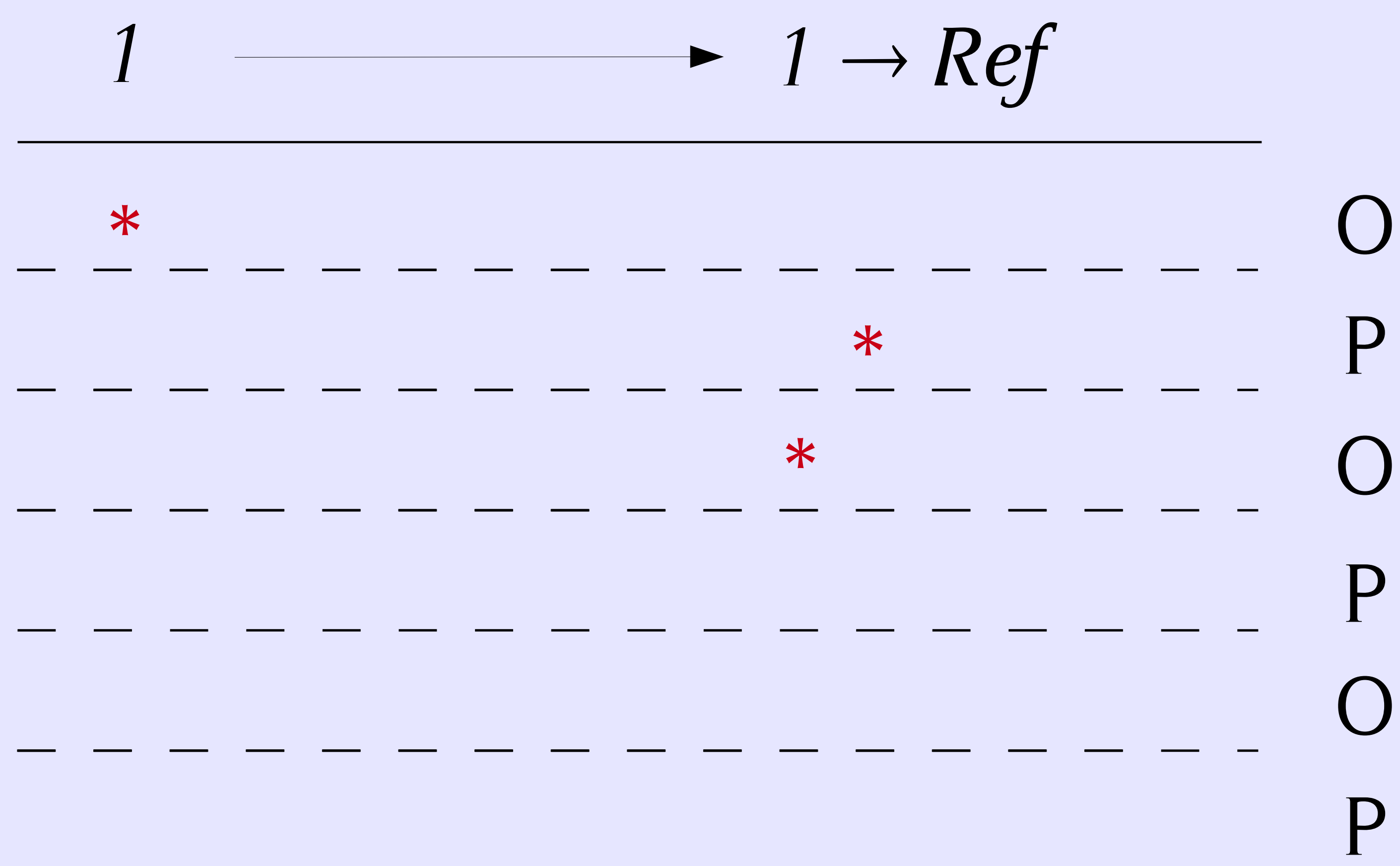
Examples

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`



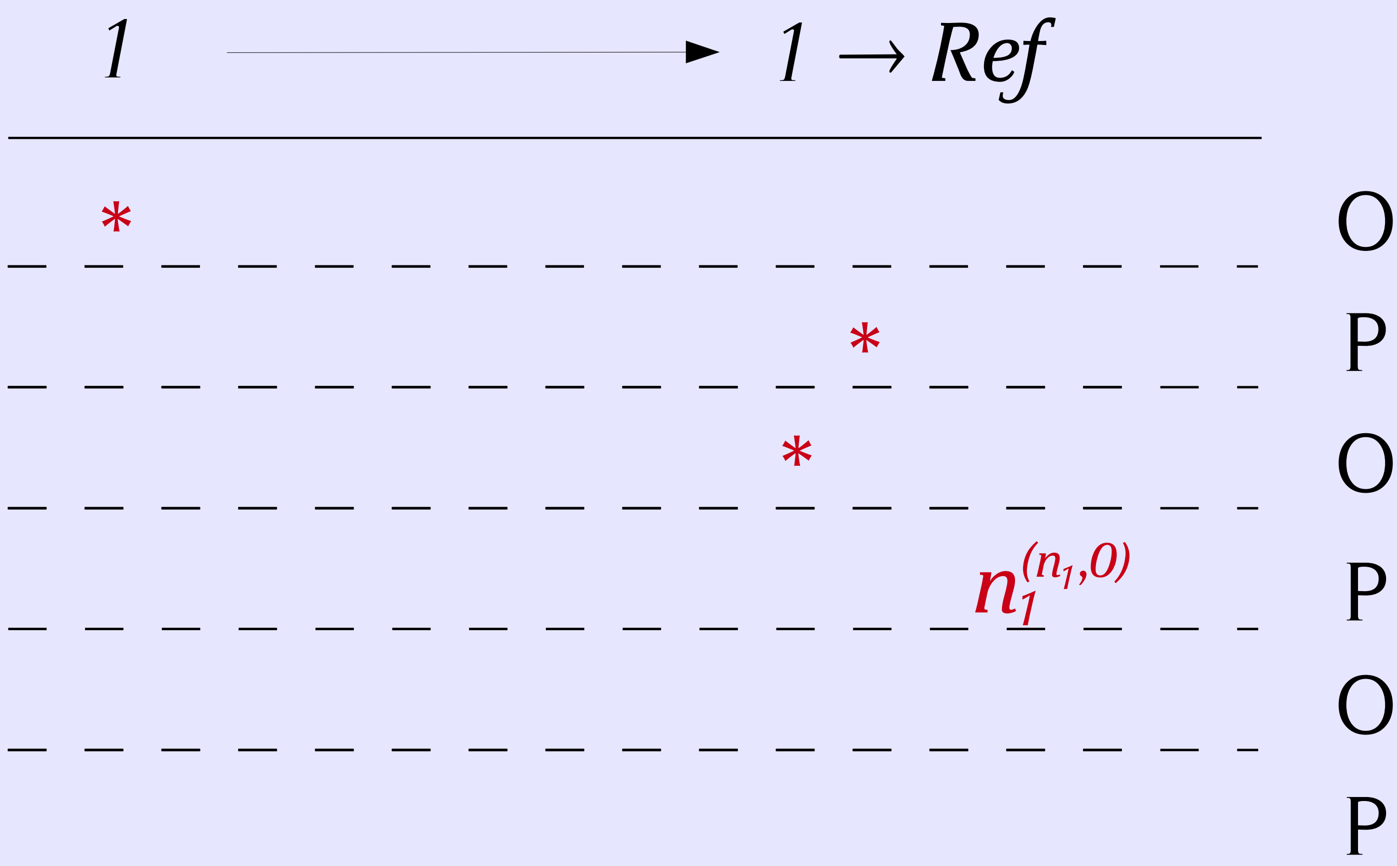
Examples

$\vdash$ `λx.ref(0)` : `com` → `intref`



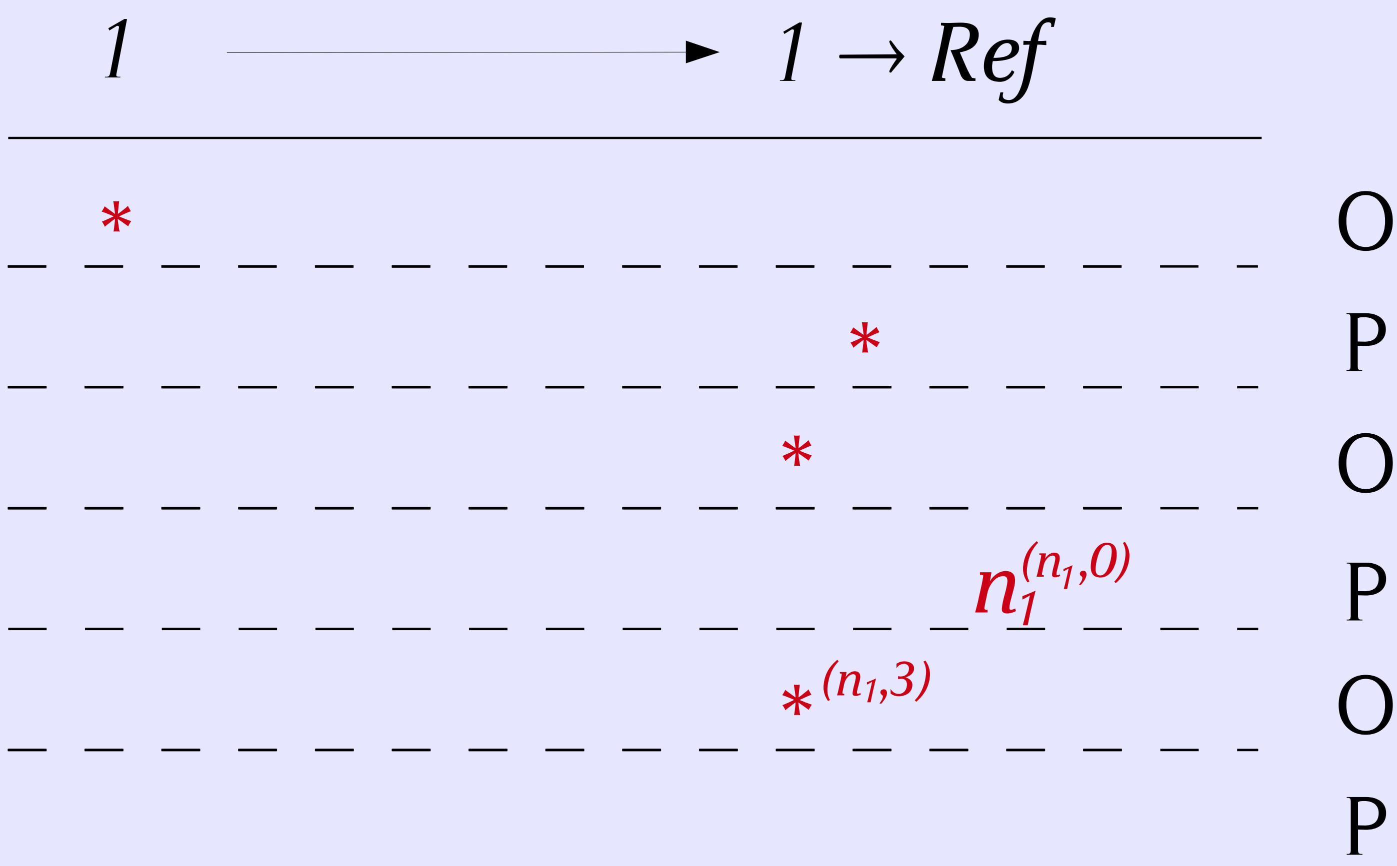
Examples

⊢ `λx.ref(0)` : `com` → `intref`



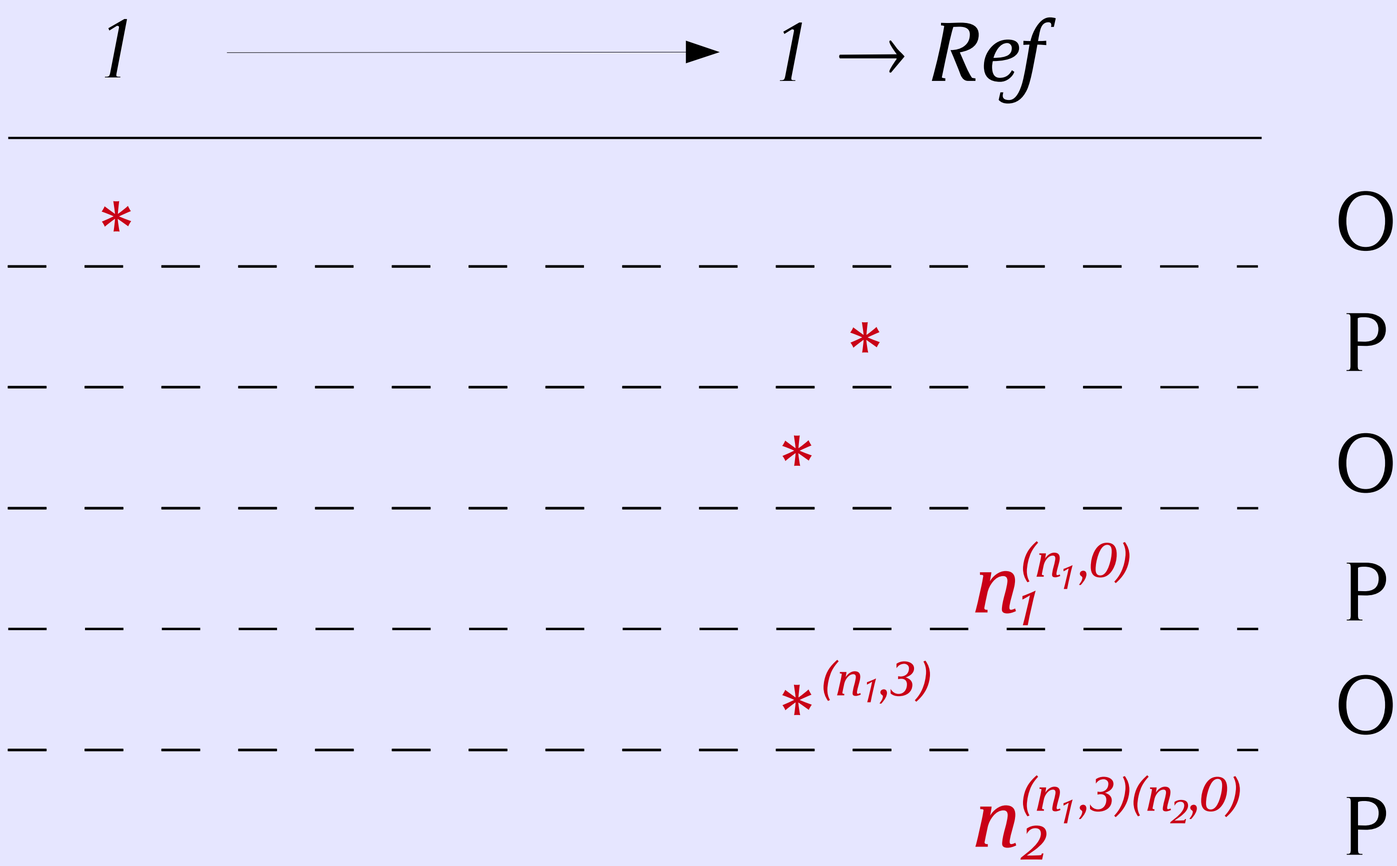
Examples

⊢ `λx.ref(0)` : com → intref



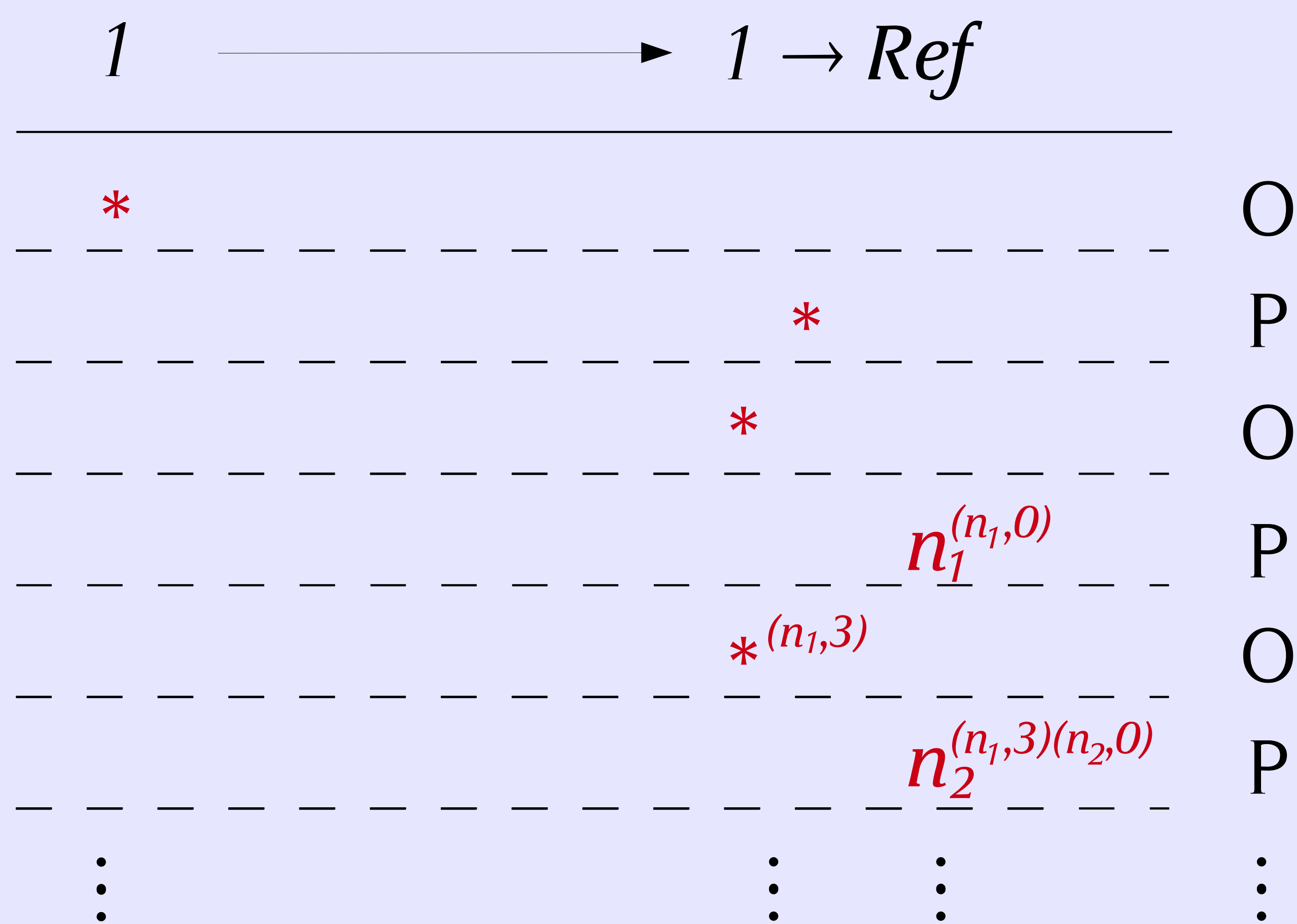
Examples

$\vdash \texttt{λx.ref(0)} : \texttt{com} \rightarrow \texttt{intref}$



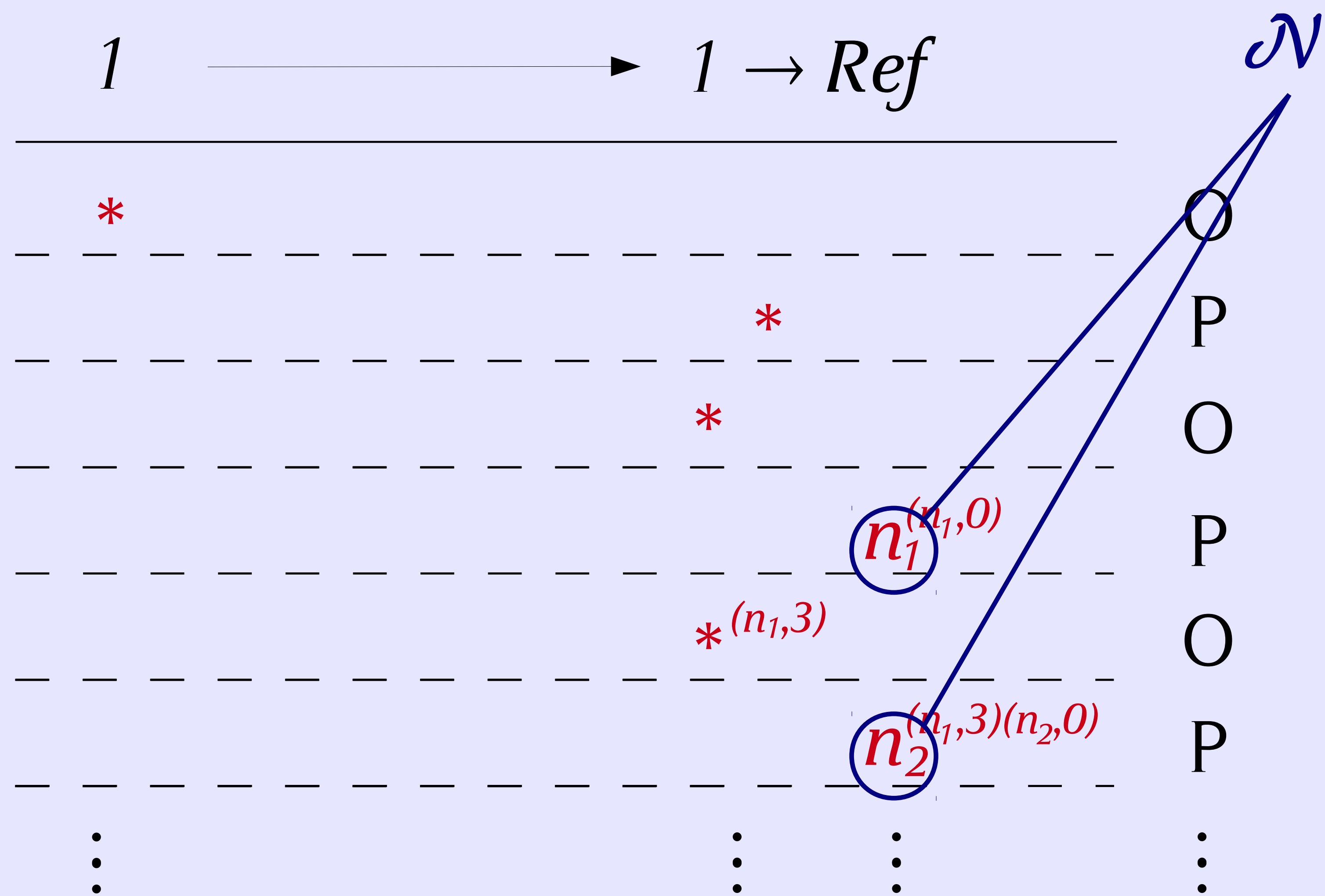
Examples

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`



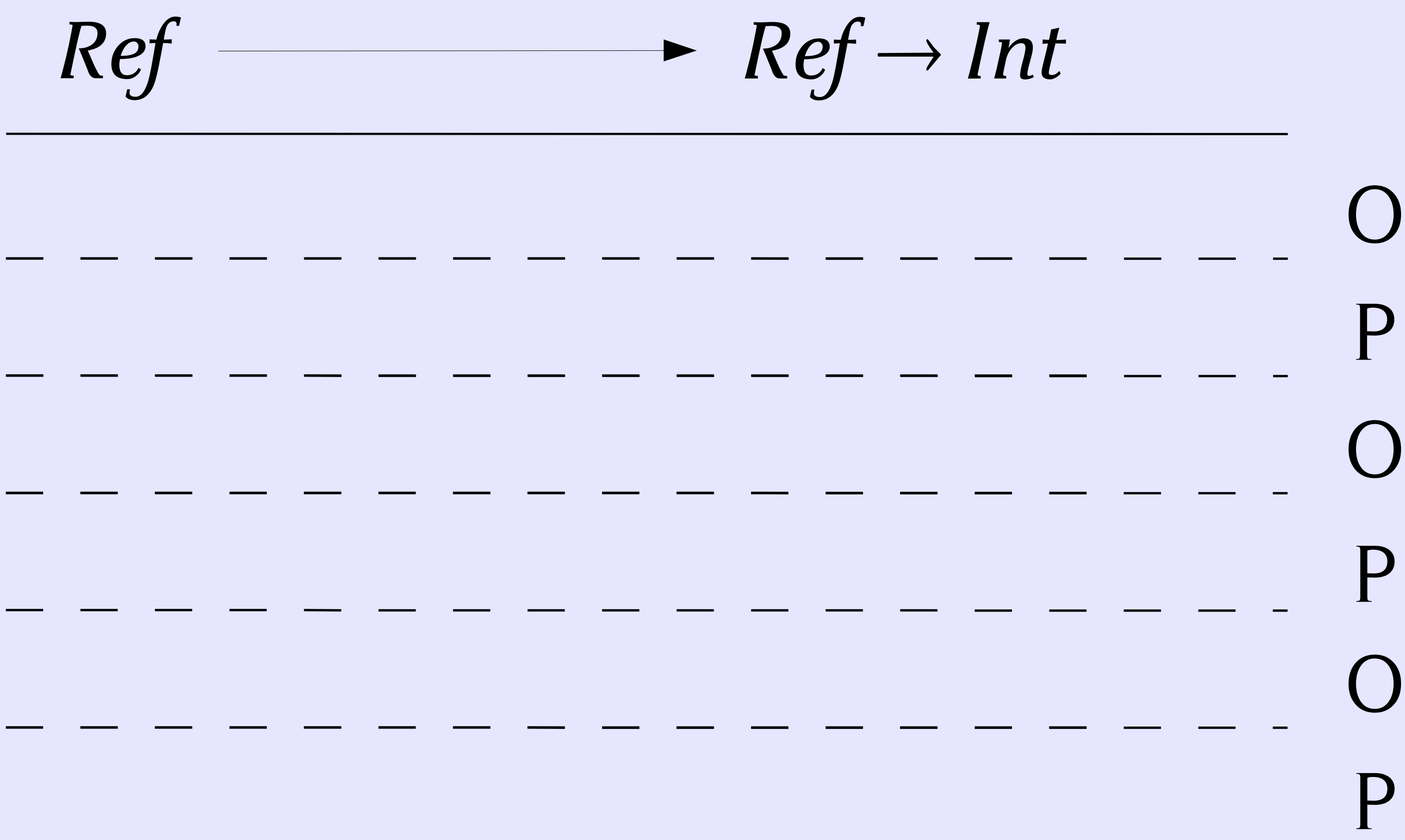
Examples

$\vdash$ `λx.ref(0)` : `com` $\rightarrow$ `intref`



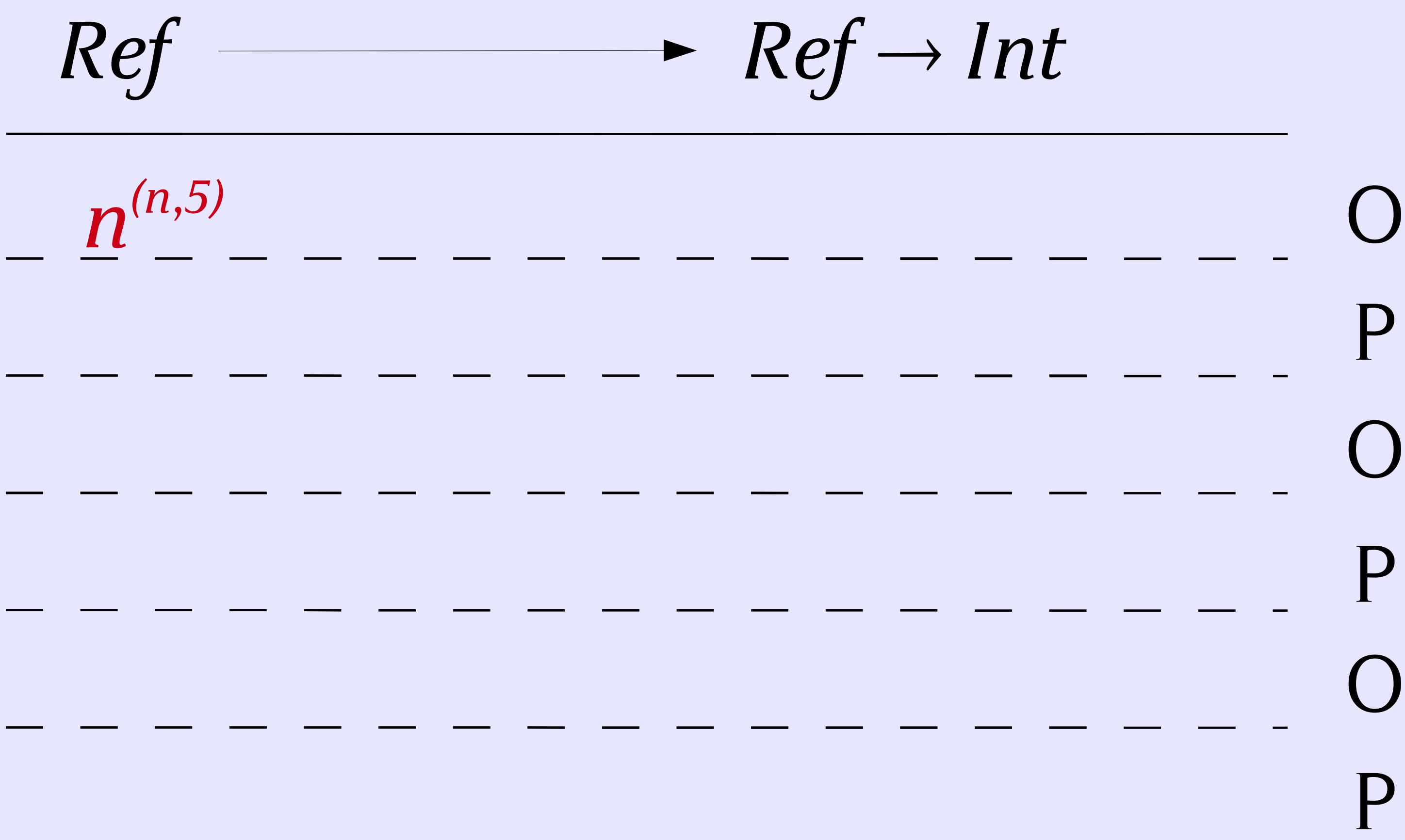
Examples

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



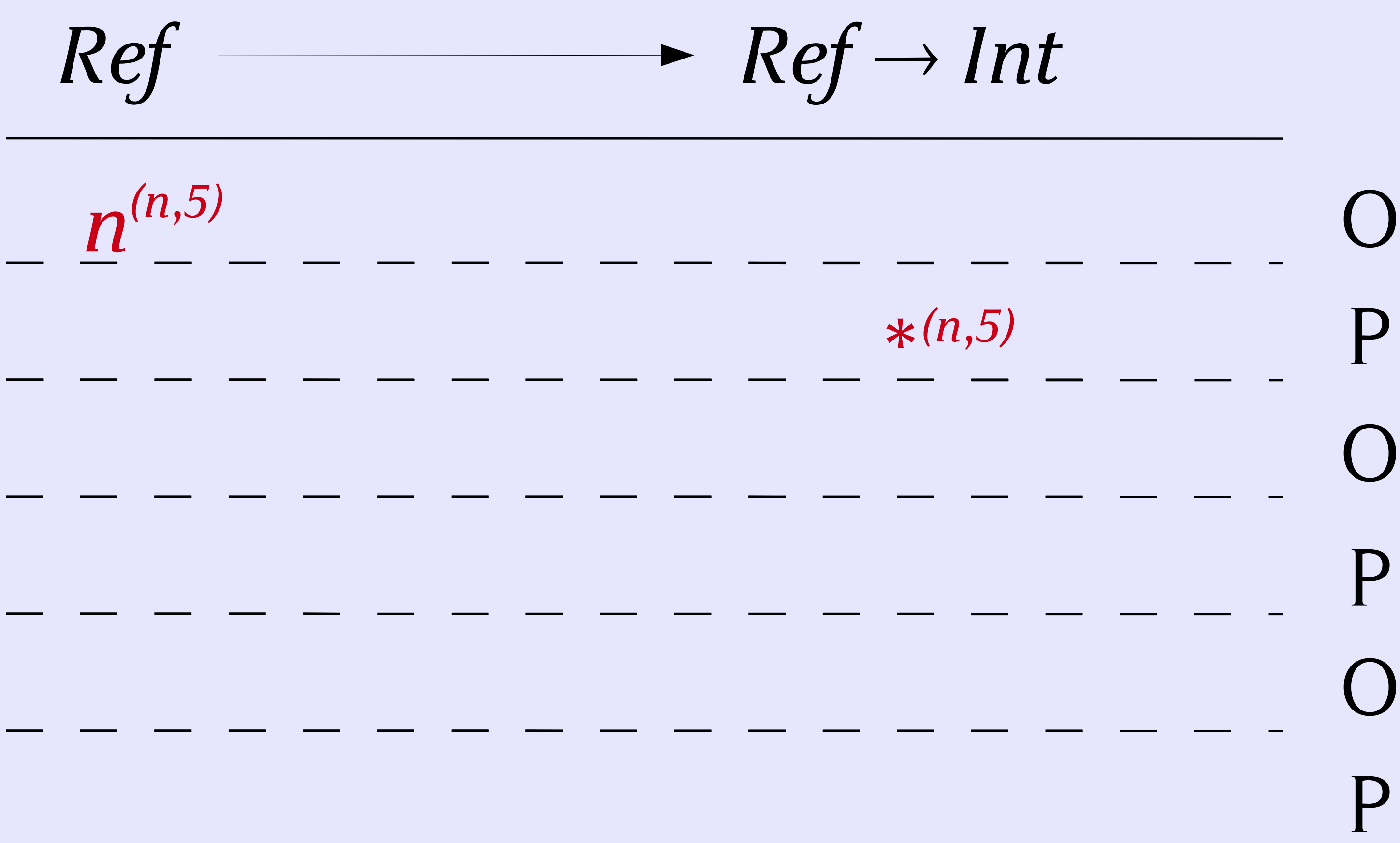
Examples

x : intref ⊢ λy.(x == y) : intref → int



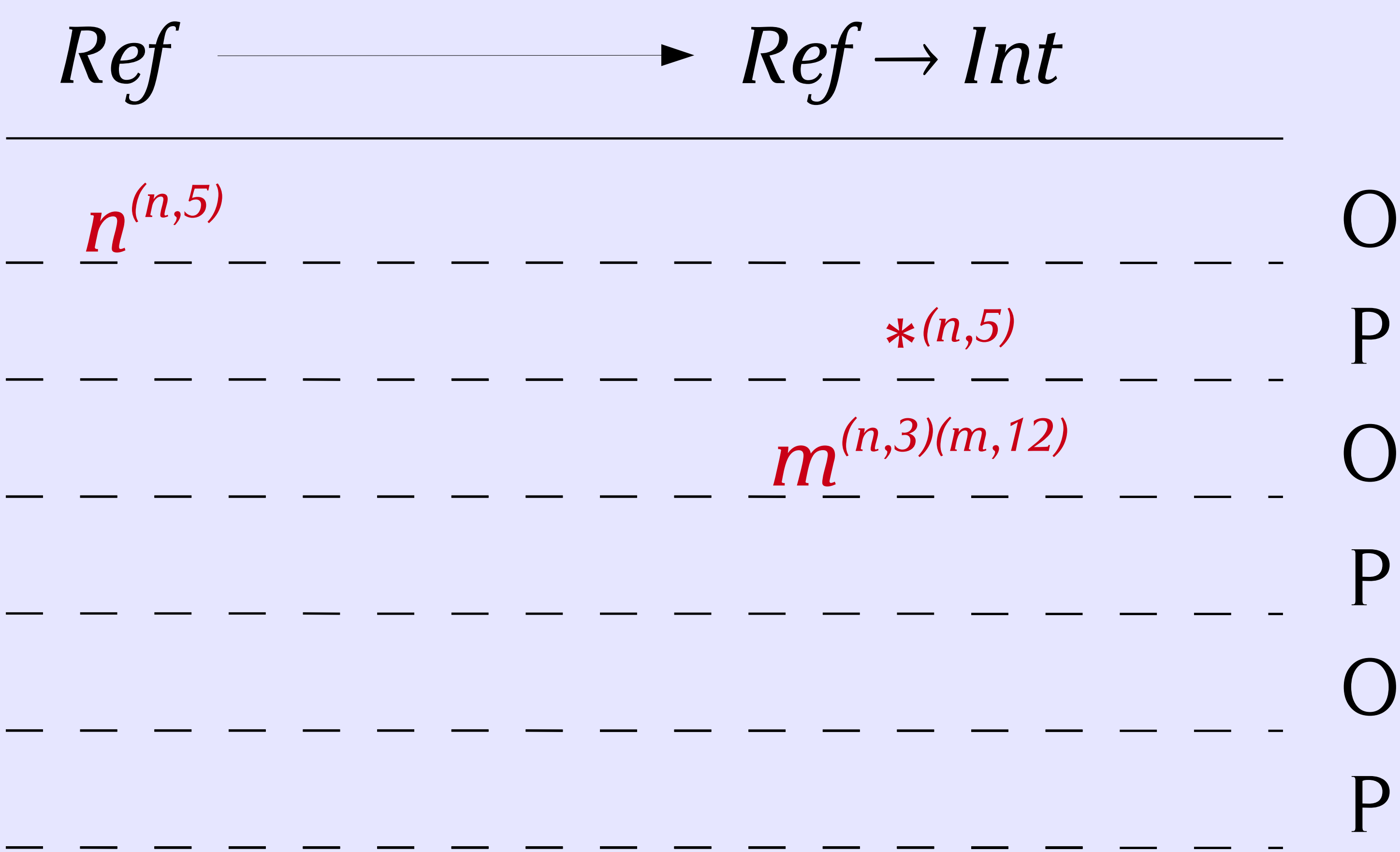
Examples

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



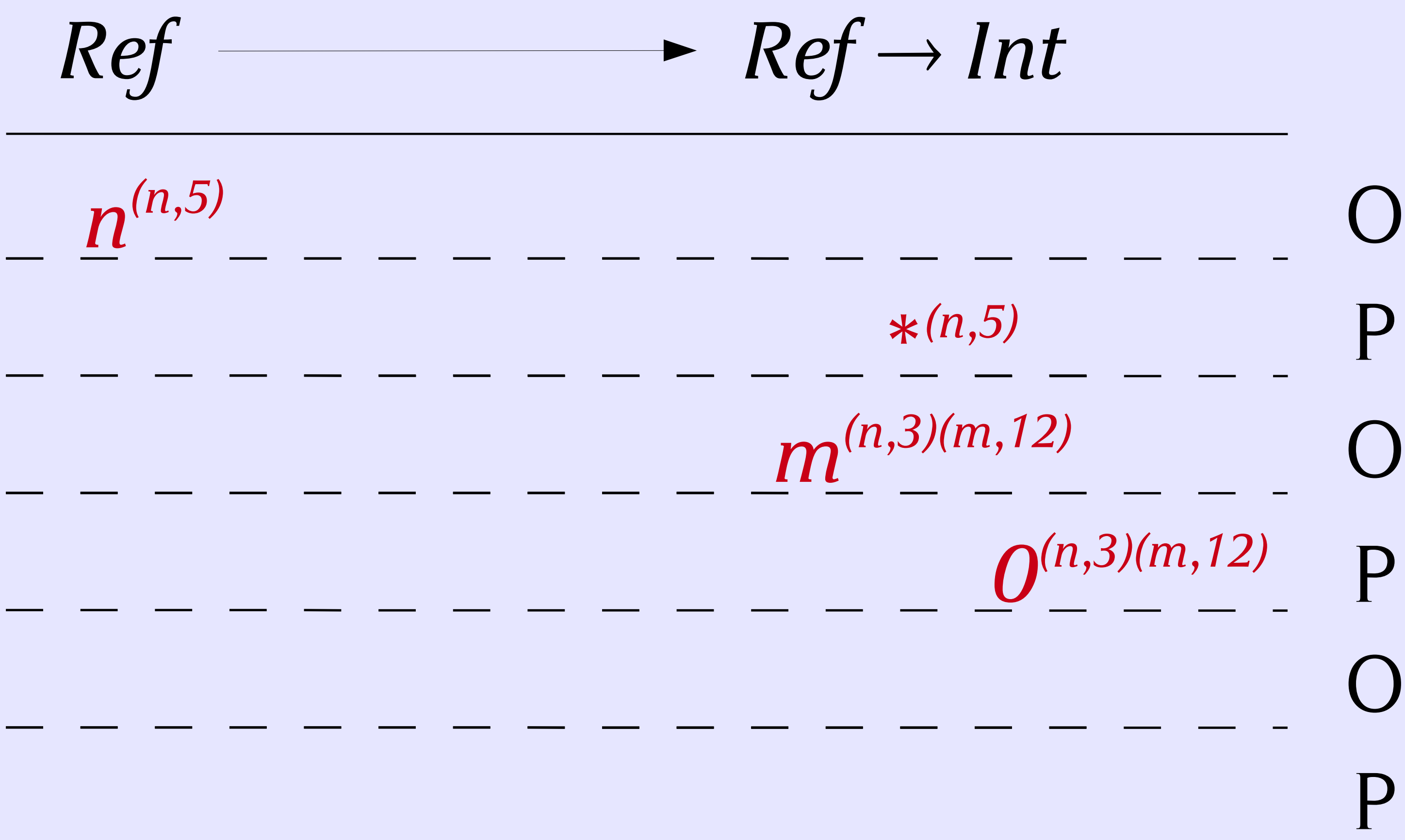
Examples

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



Examples

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



Examples

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

Ref	$\longrightarrow$ $Ref \to Int$	
$n^{(n,5)}$		O
	$*^{(n,5)}$	P
	$m^{(n,3)(m,12)}$	O
	$0^{(n,3)(m,12)}$	P
	$n^{(n,13)}$	O
		P

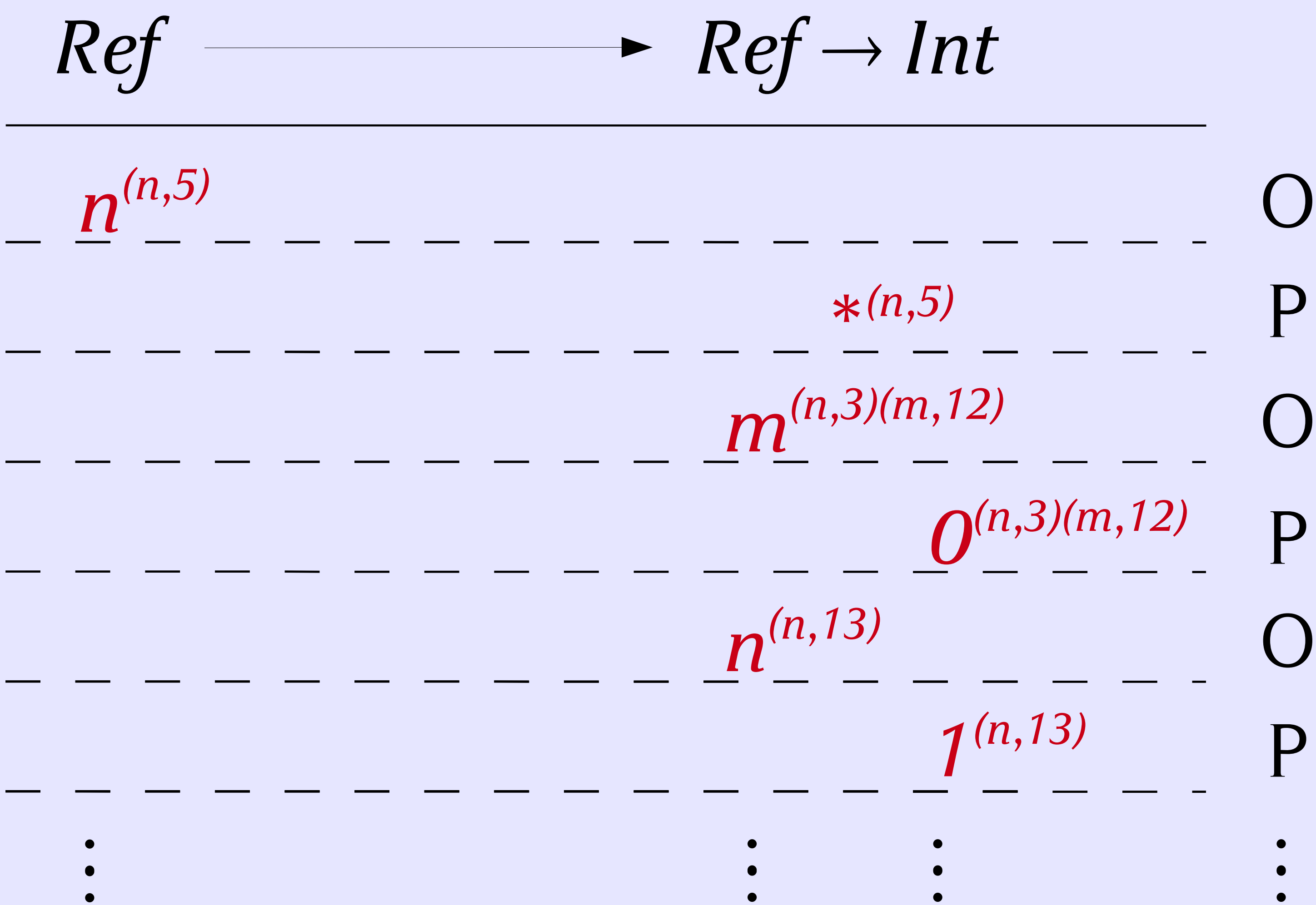
Examples

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

Ref	$\longrightarrow$	$Ref \to Int$	
$n^{(n,5)}$			O
		$*^{(n,5)}$	P
		$m^{(n,3)(m,12)}$	O
		$0^{(n,3)(m,12)}$	P
		$n^{(n,13)}$	O
		$1^{(n,13)}$	P

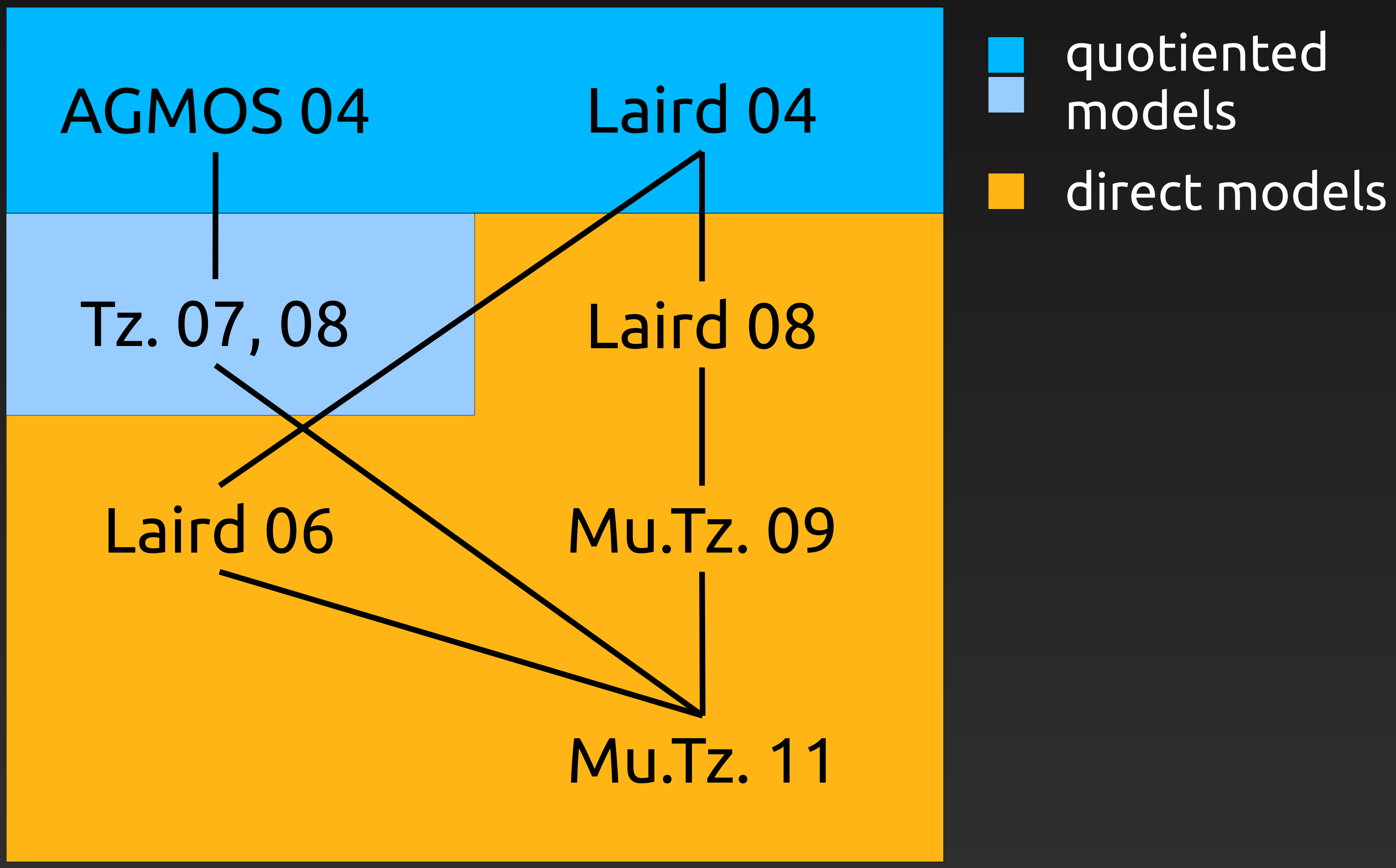
Examples

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

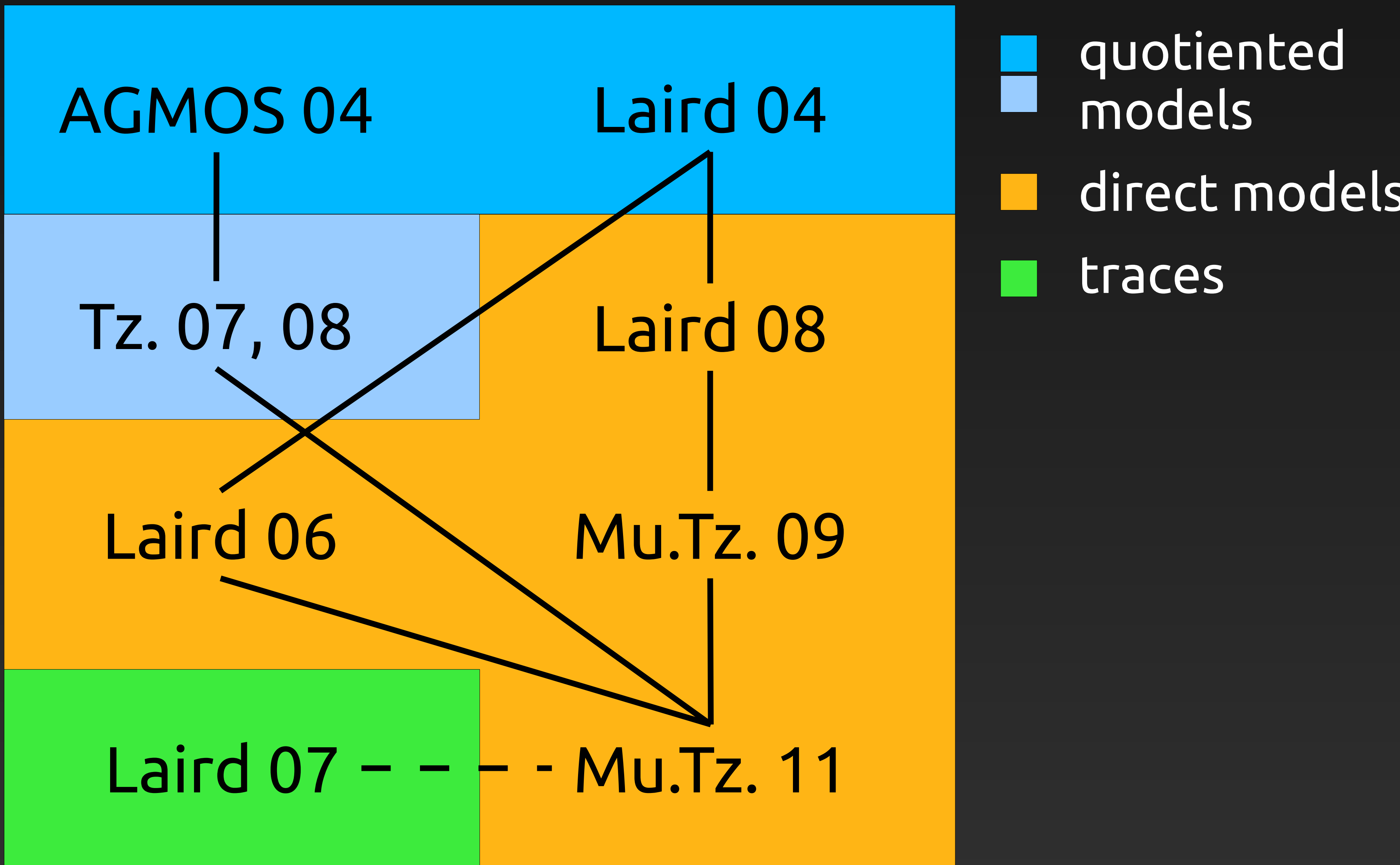


Achievements

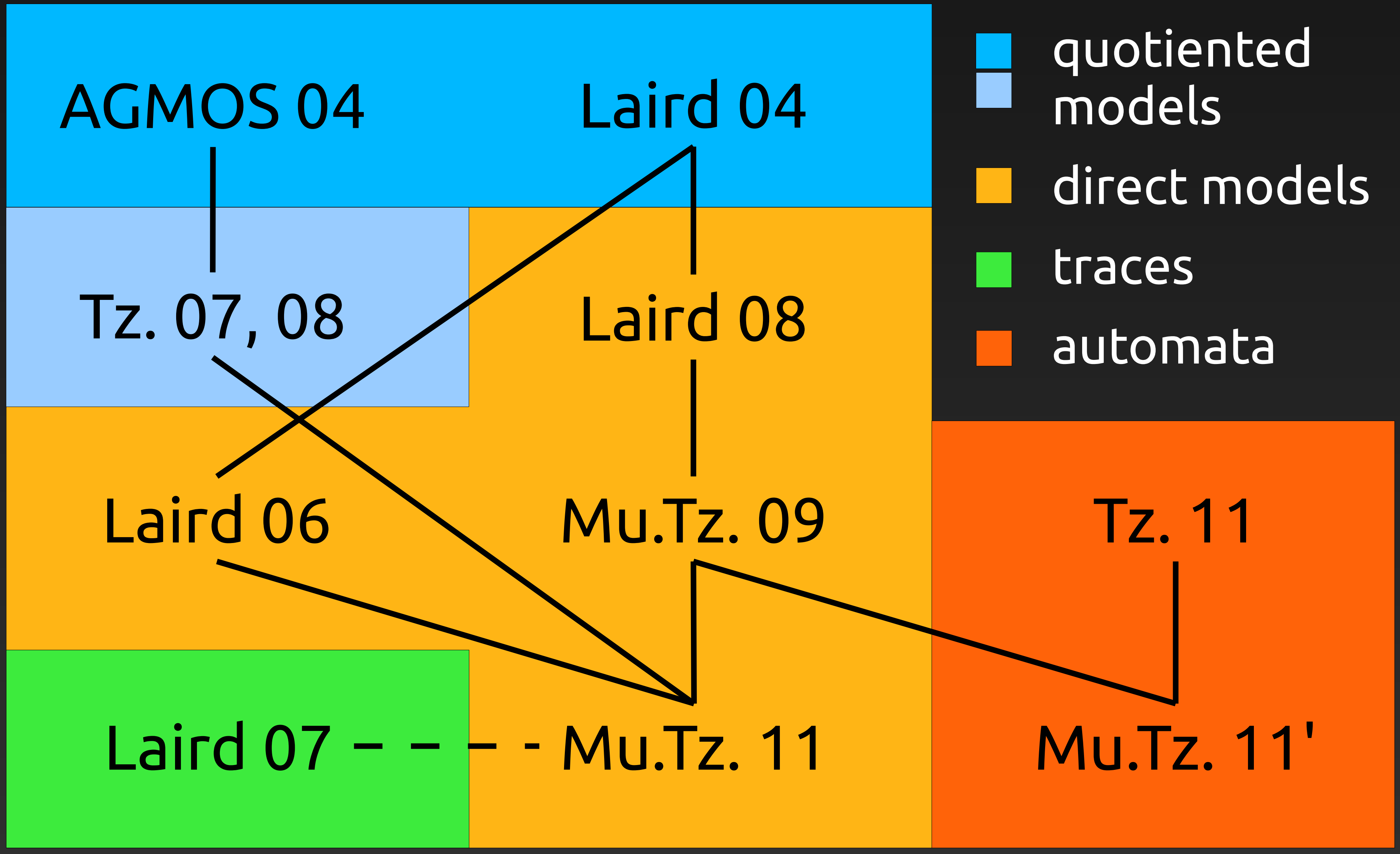
Achievements



Achievements



Achievements



Quotiented models

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

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

Quotiented models

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

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

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

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

Quotiented models

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

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

Characteristics:

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

Direct models

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

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

Direct models

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

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

Characteristics:

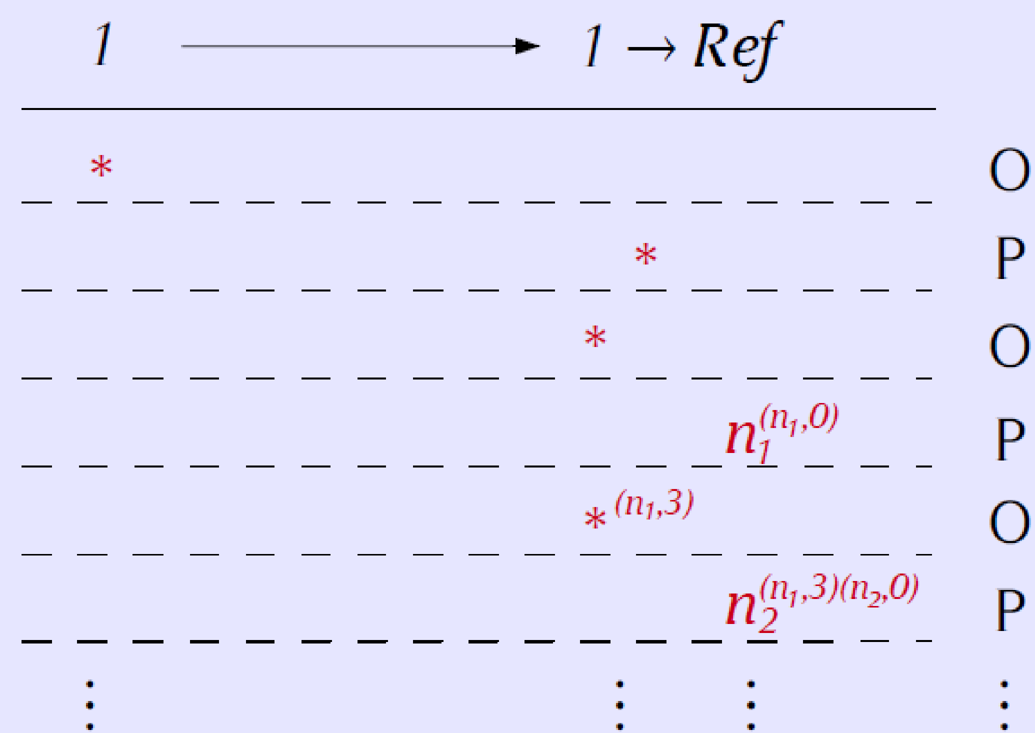
- moves involving names/ moves-with-store
- name-availability conditions/ “direct” effects

Direct models

- Laird 06: hig
  Laird 08: po
- Mu.Tz. 09: i
  Mu.Tz. 11: H

$M \cong N$

$\vdash$ `λx.ref(0) : com → intref`



Characteristics

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

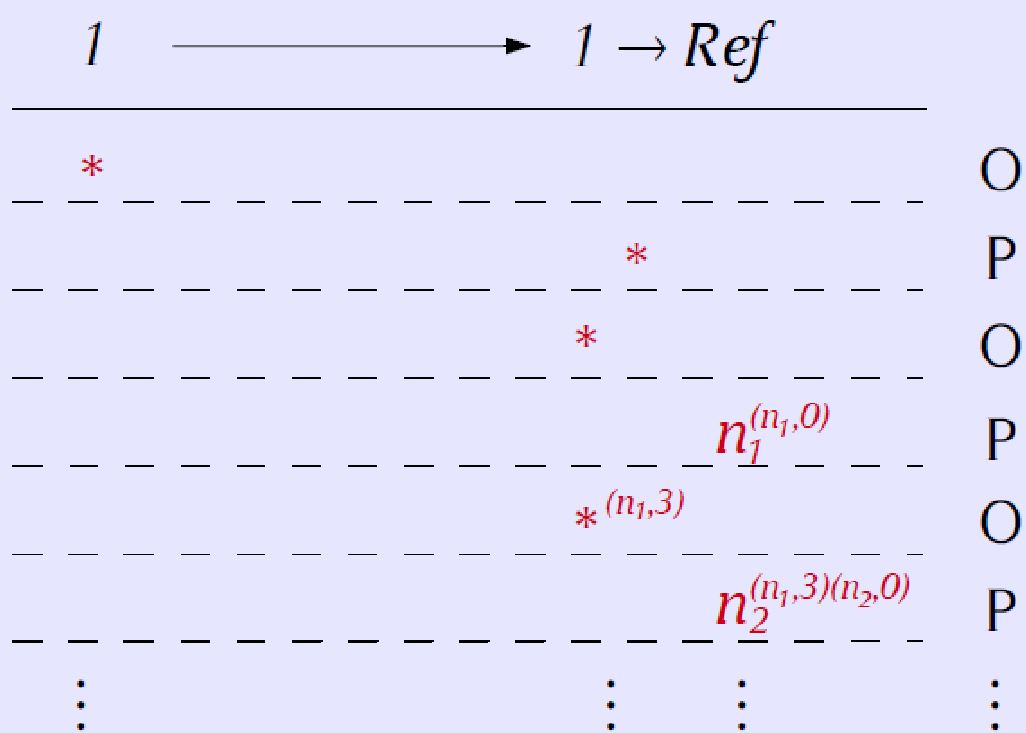
Direct models

- Laird 06: hi
  Laird 08: po
- Mu.Tz. 09: i
  Mu.Tz. 11: H

$M \cong N$

Characteristics

$\vdash$ `λx.ref(0) : com → intref`



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

Algorithmics

Fresh-register automata

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

Fresh-register automata

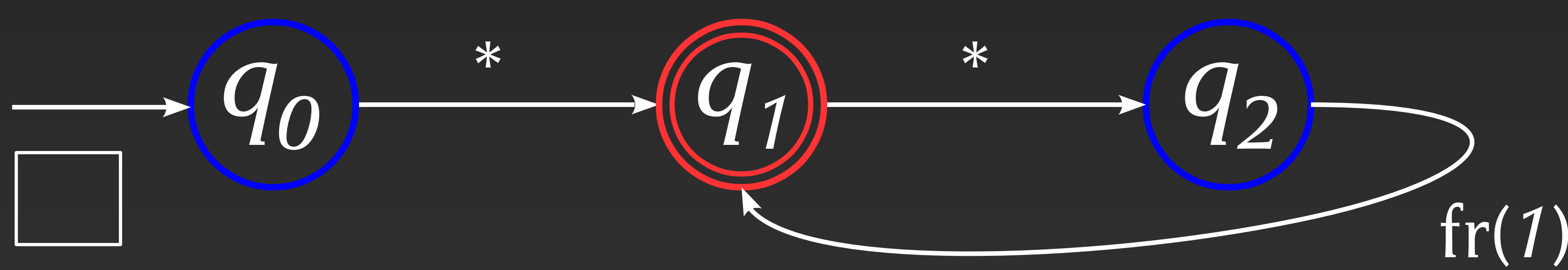
$\texttt{λz.ref(0)} \longmapsto \{ * * *n_1 * n_2 * n_3 \ldots \mid n_i\text{'s distinct} \}$

Automata with names

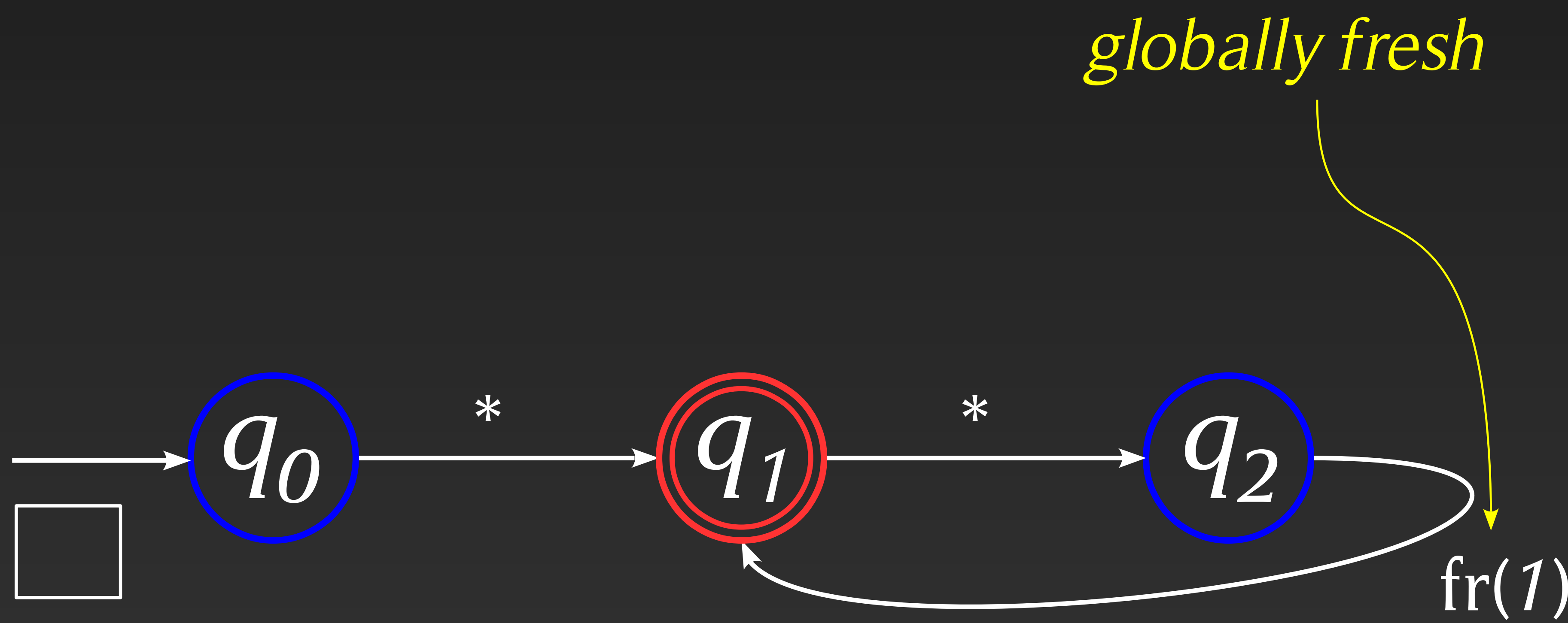
- Infinite alphabet $\mathcal{N}$
- Freshness recognition

Finite-state machines with registers

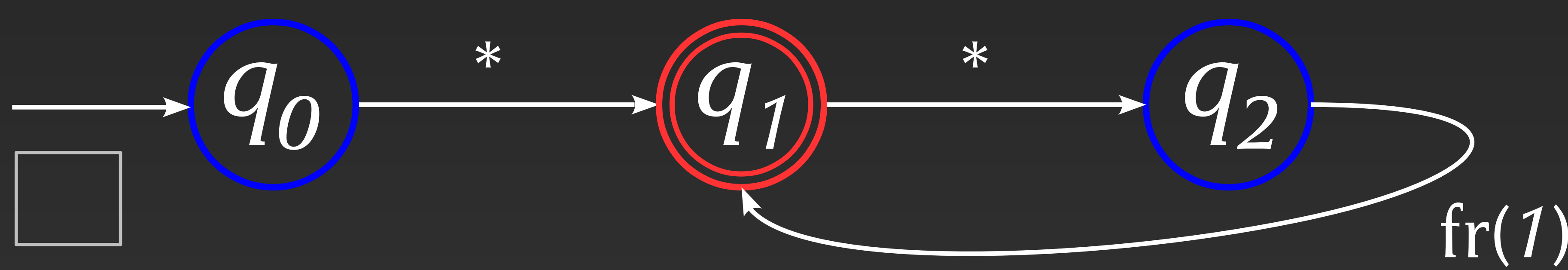
Fresh-register automata



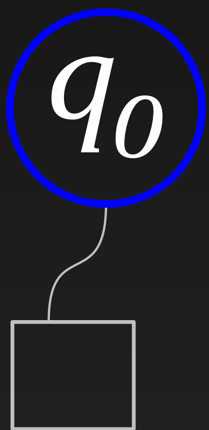
Fresh-register automata



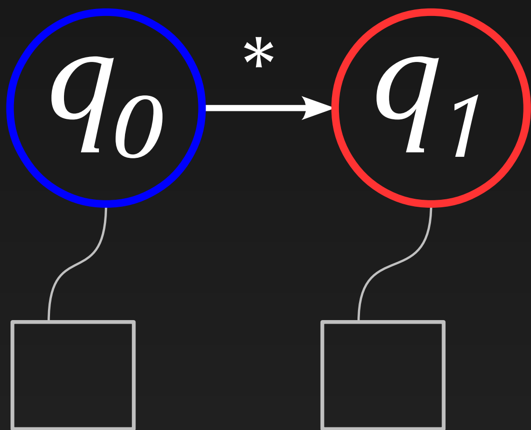
Fresh-register automata

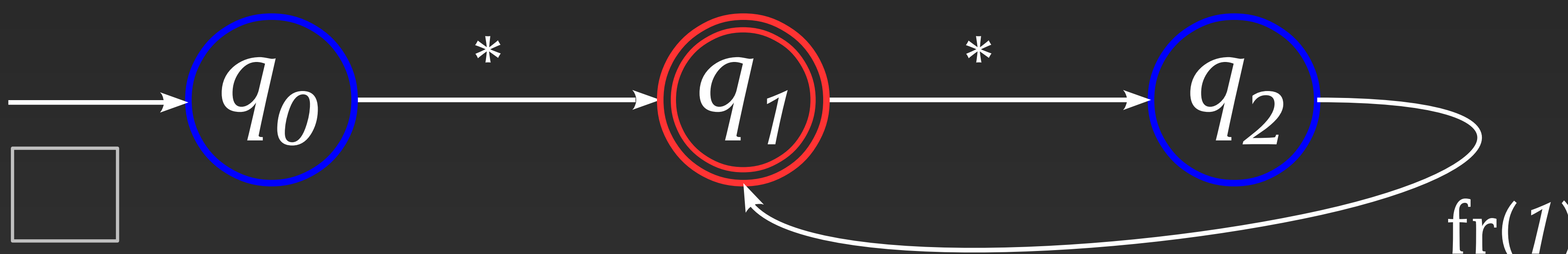


Fresh-register automata

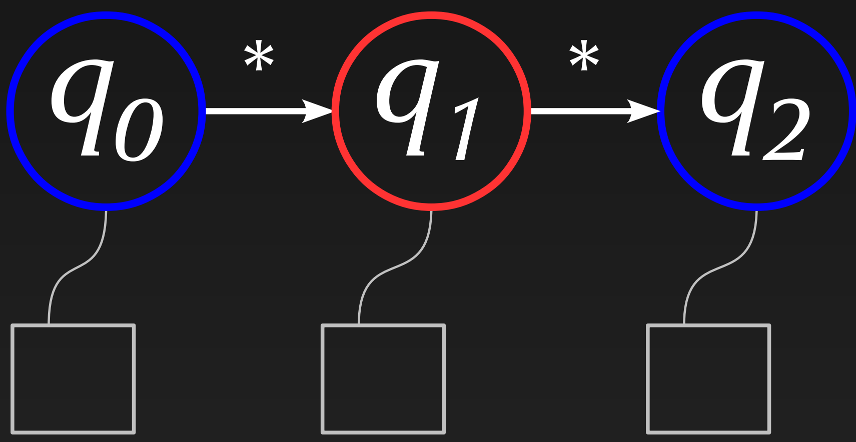


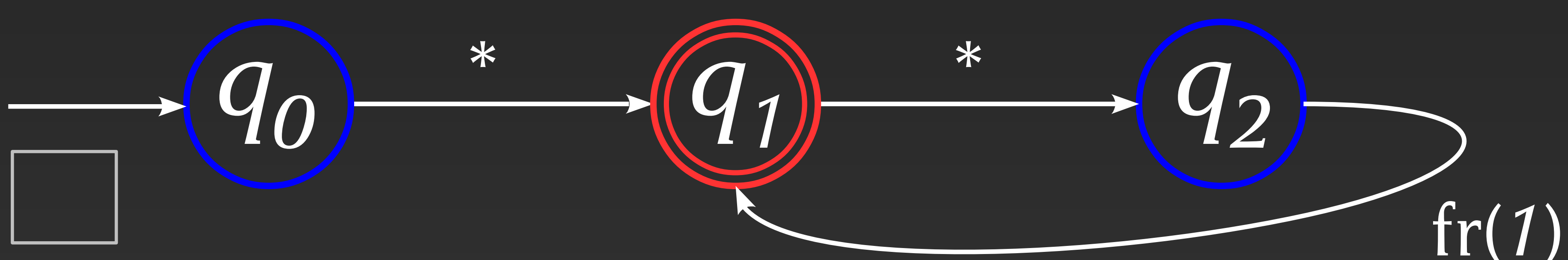
Fresh-register automata



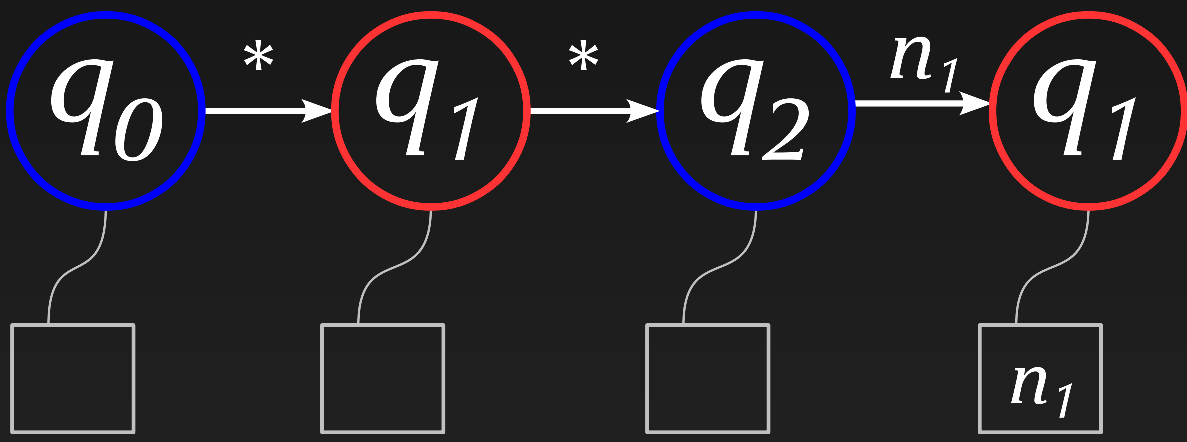


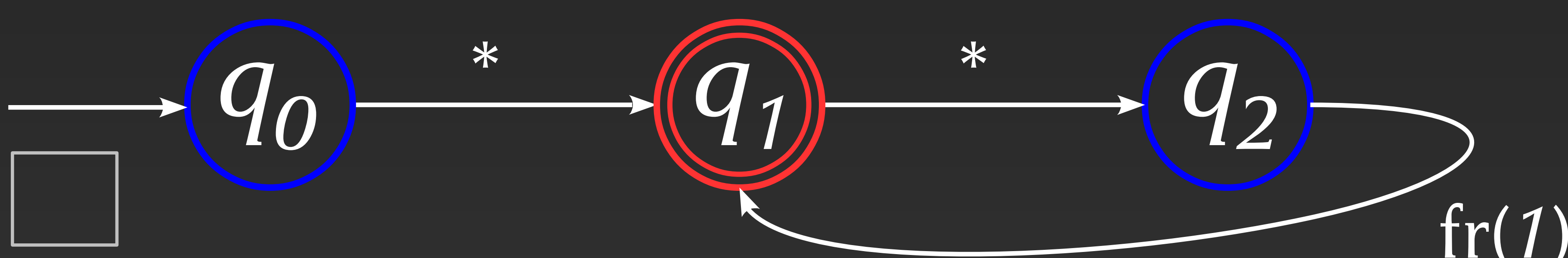
Fresh-register automata



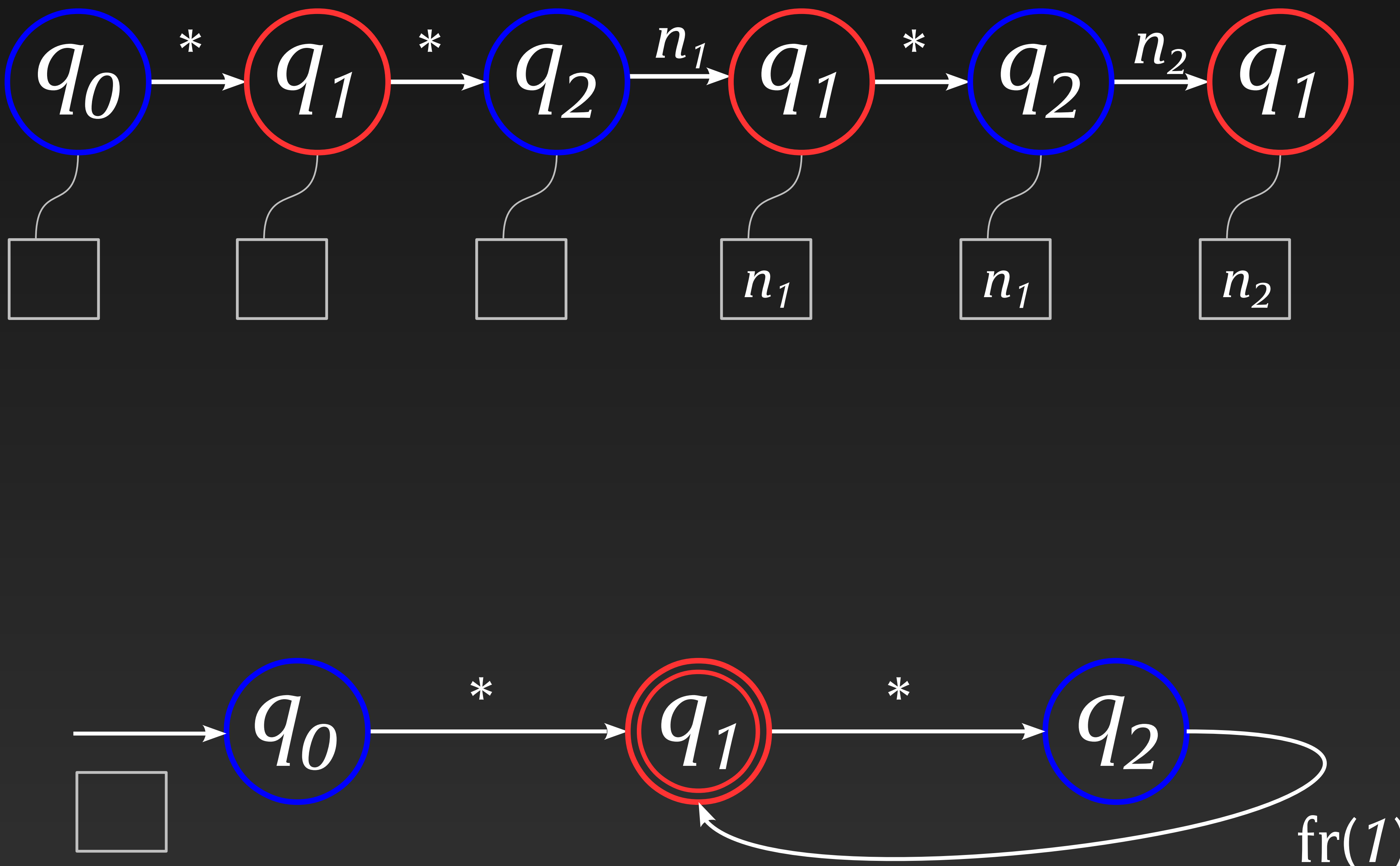


Fresh-register automata

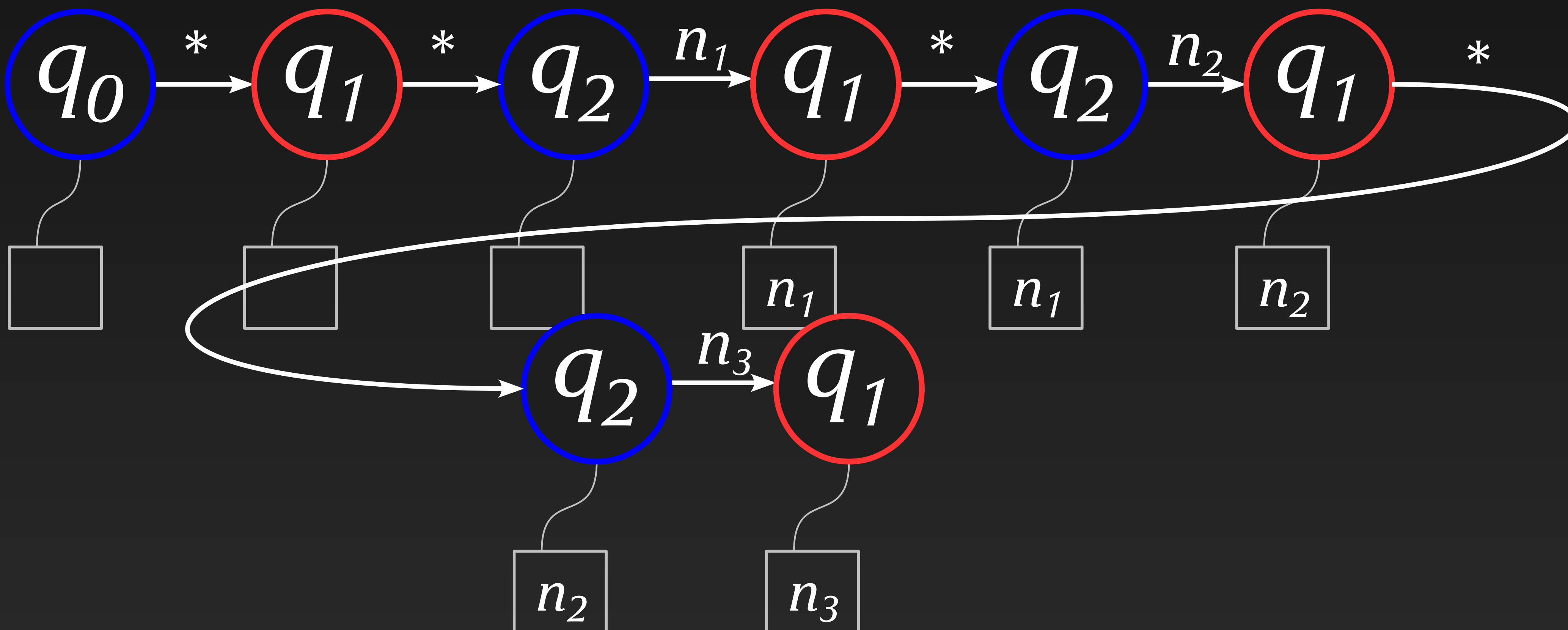




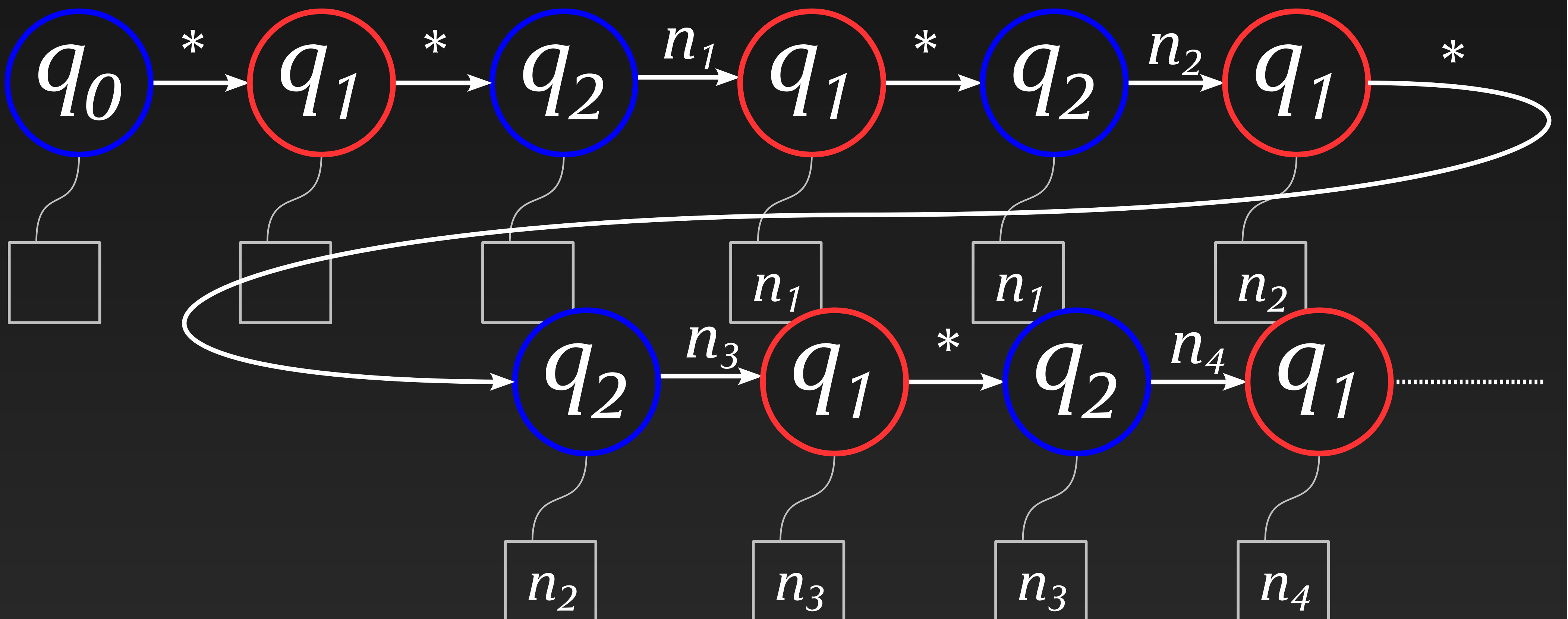
Fresh-register automata



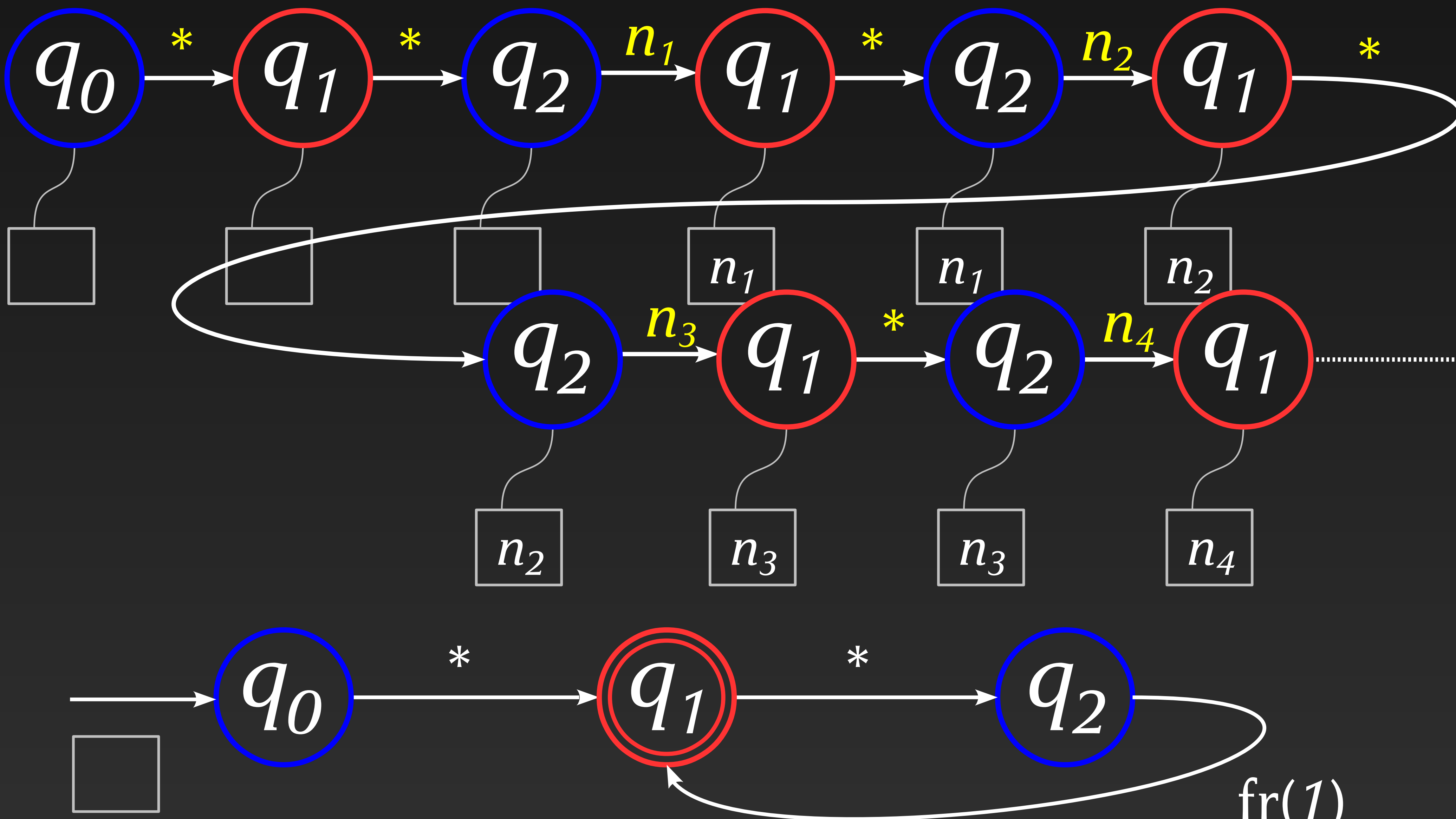
Fresh-register automata



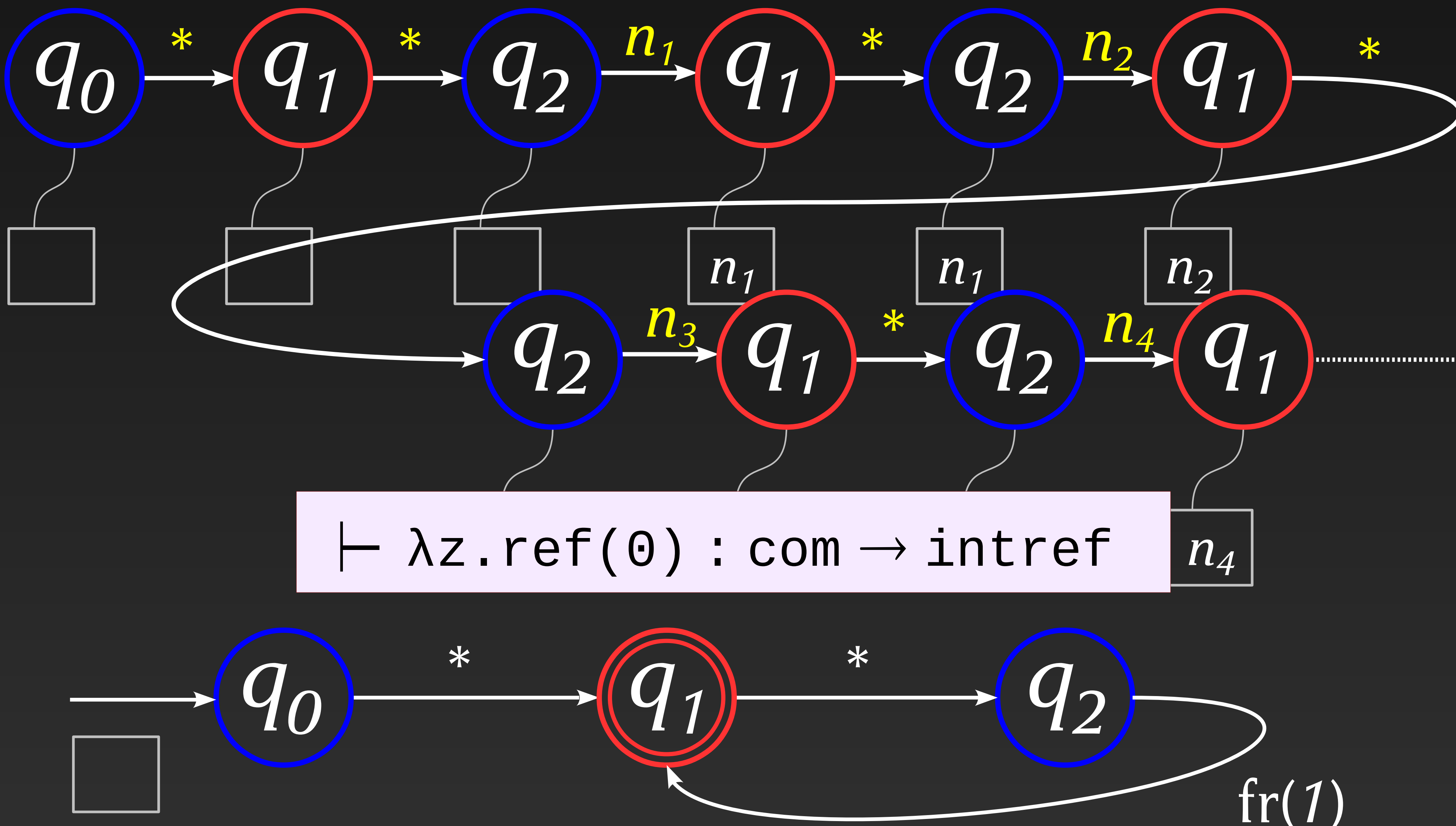
Fresh-register automata



Fresh-register automata

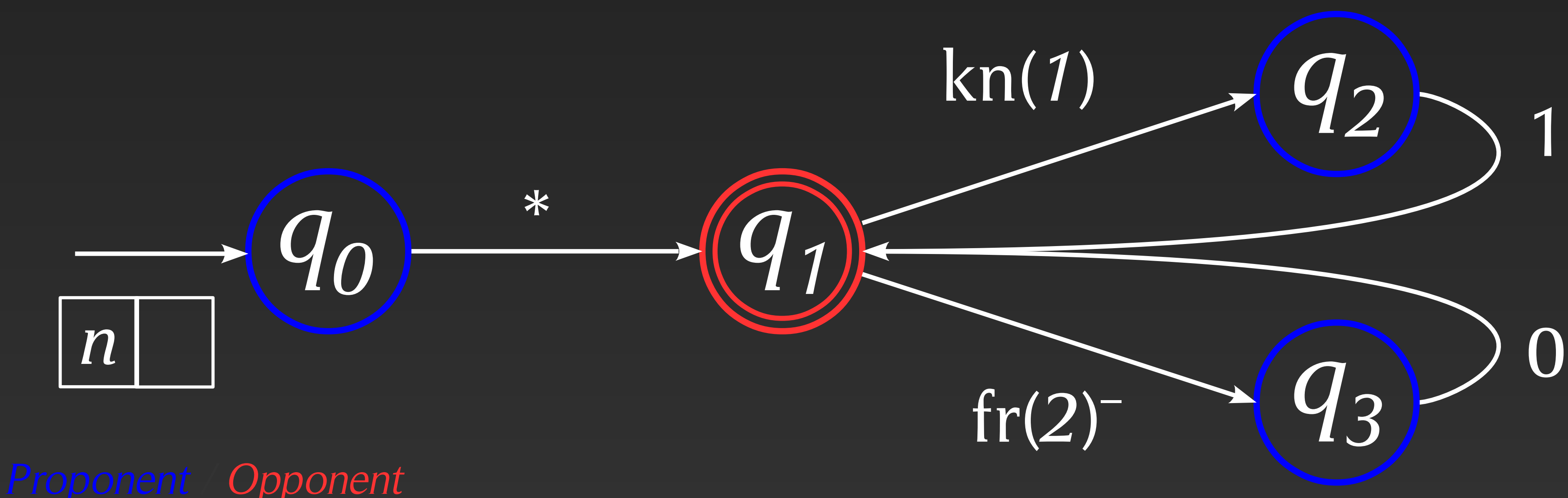


Fresh-register automata

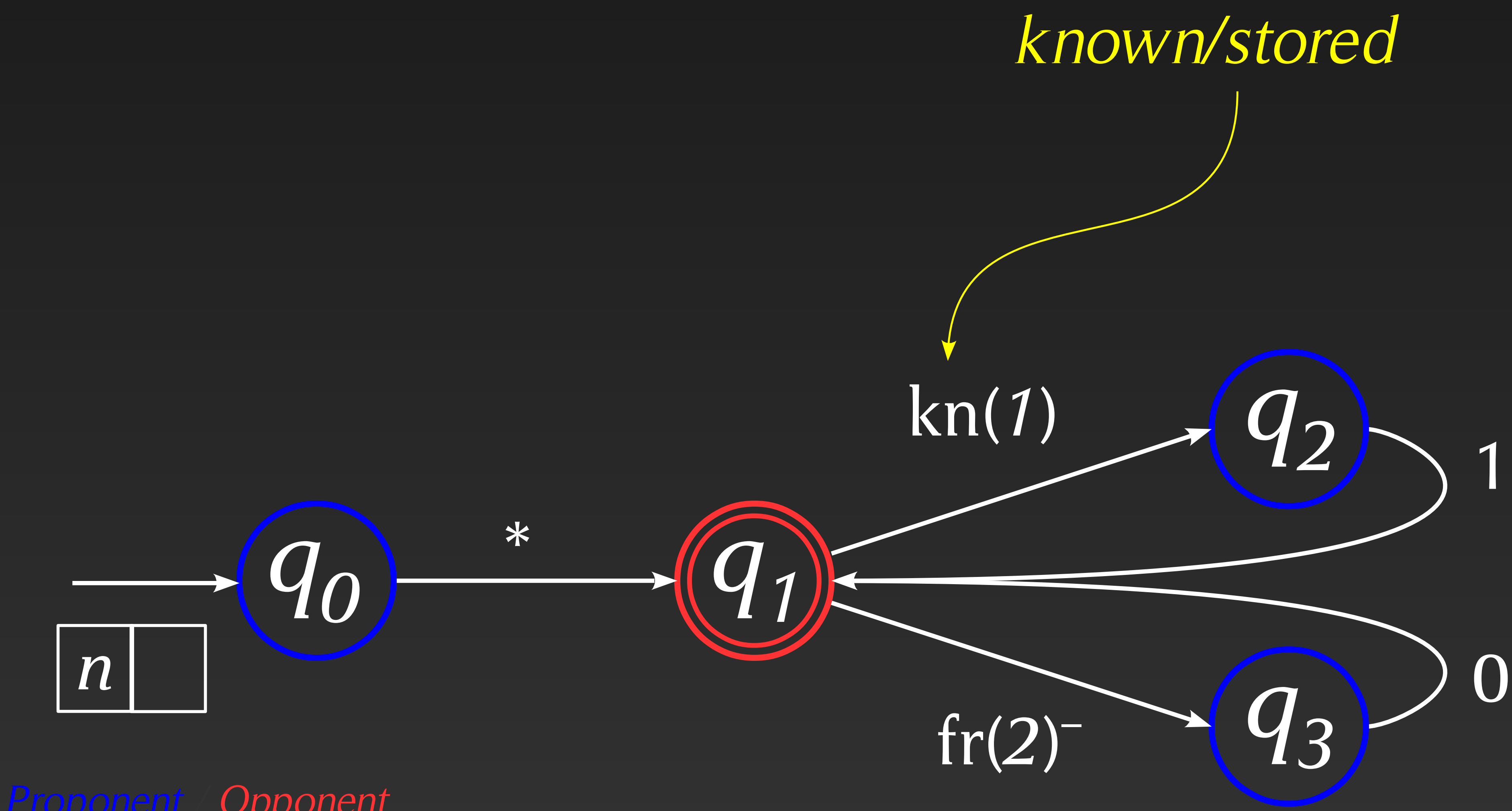


Proponent *Opponent*

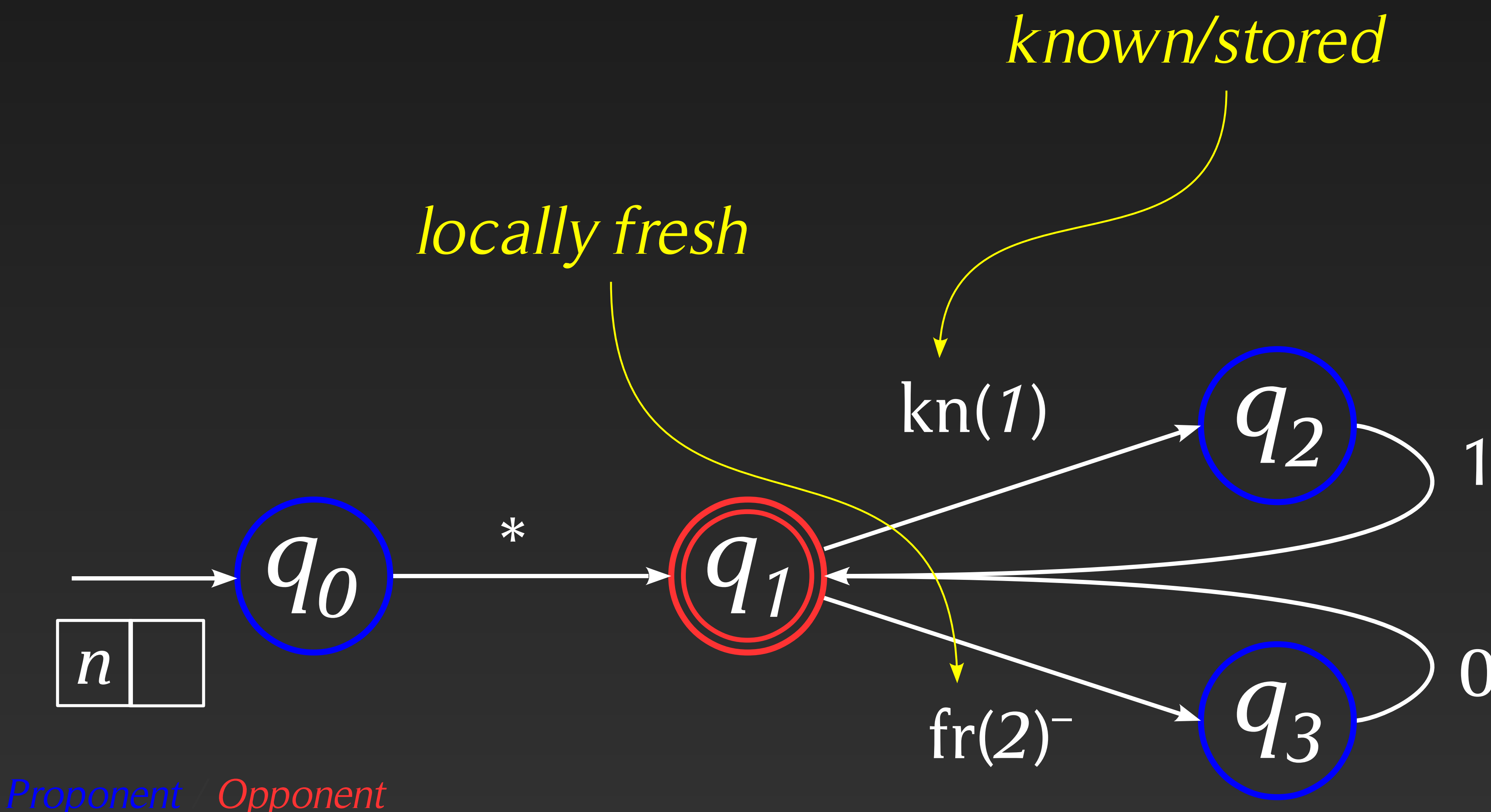
Fresh-register automata



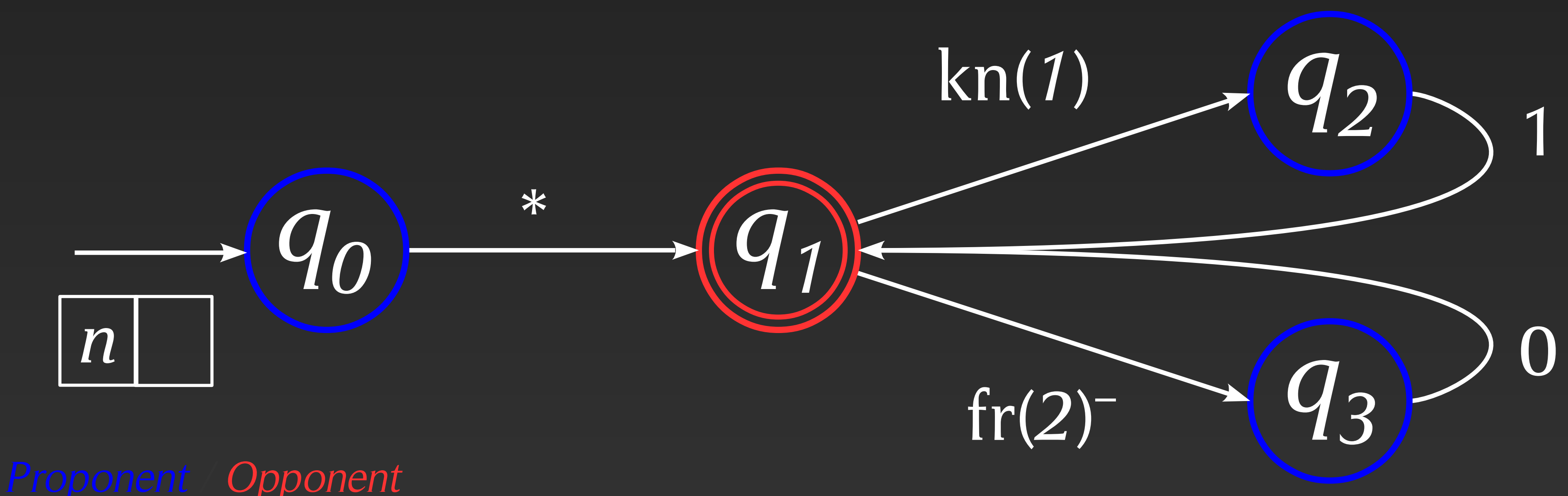
Fresh-register automata



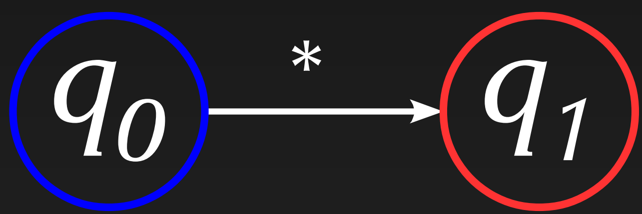
Fresh-register automata

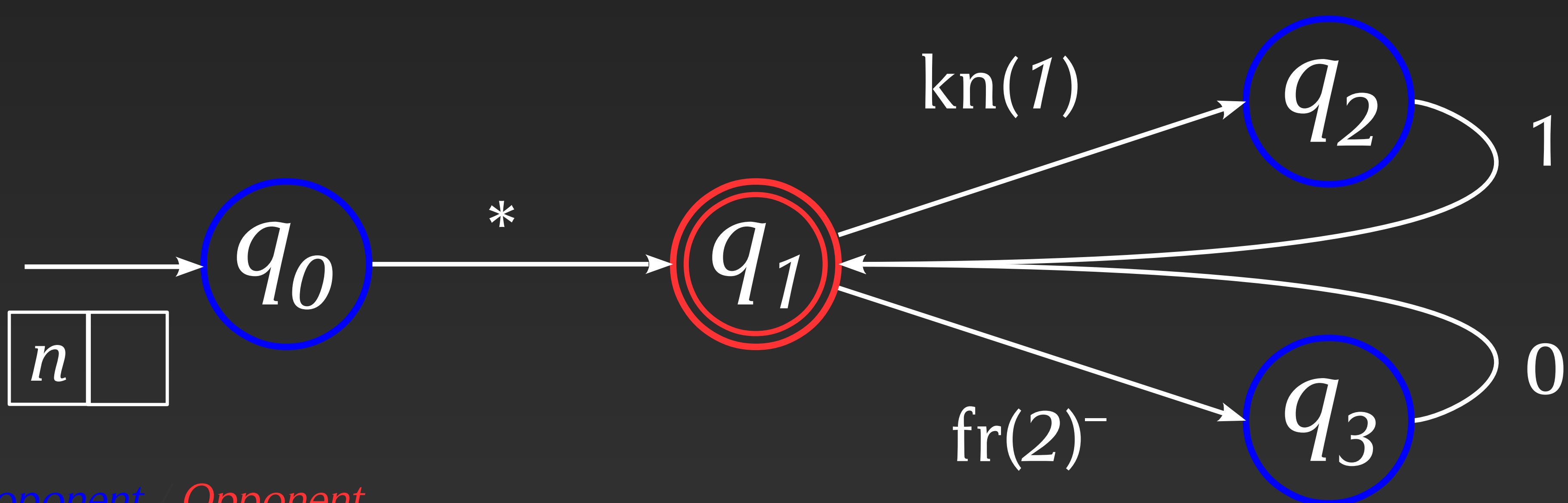


Fresh-register automata



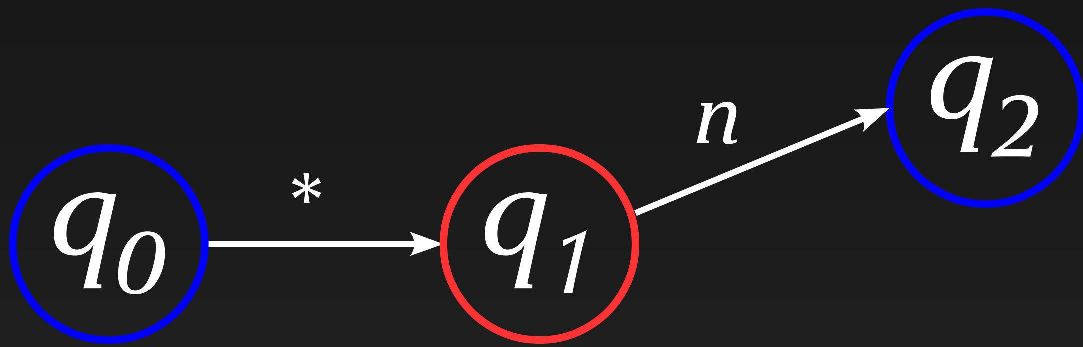
Fresh-register automata

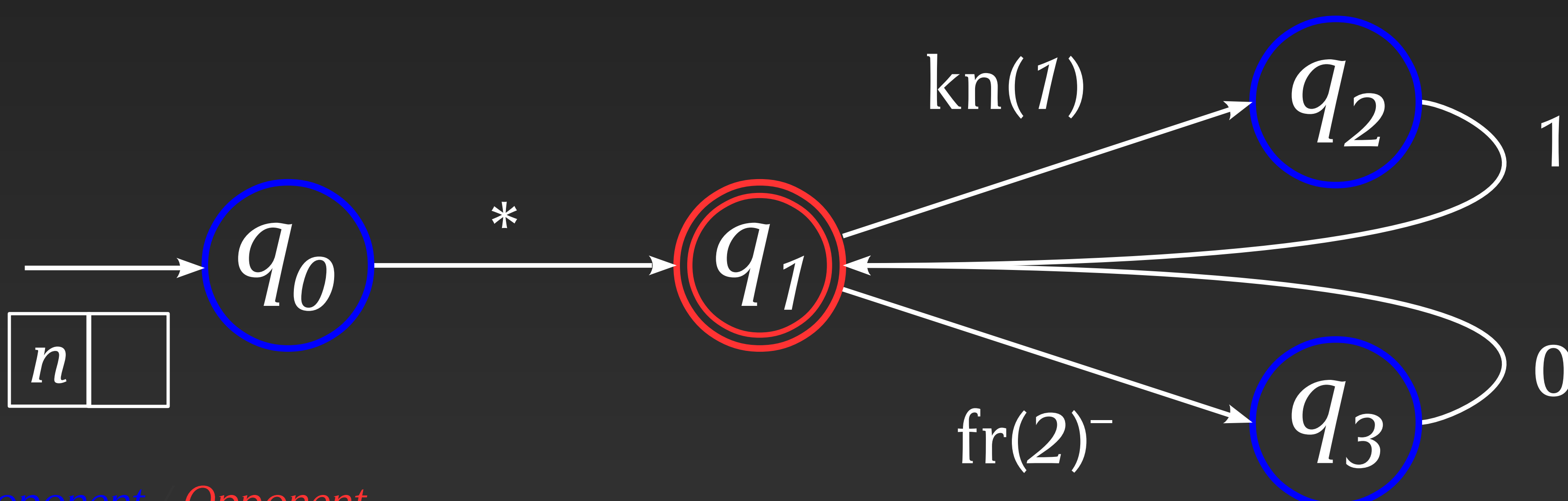




Proponent *Opponent*

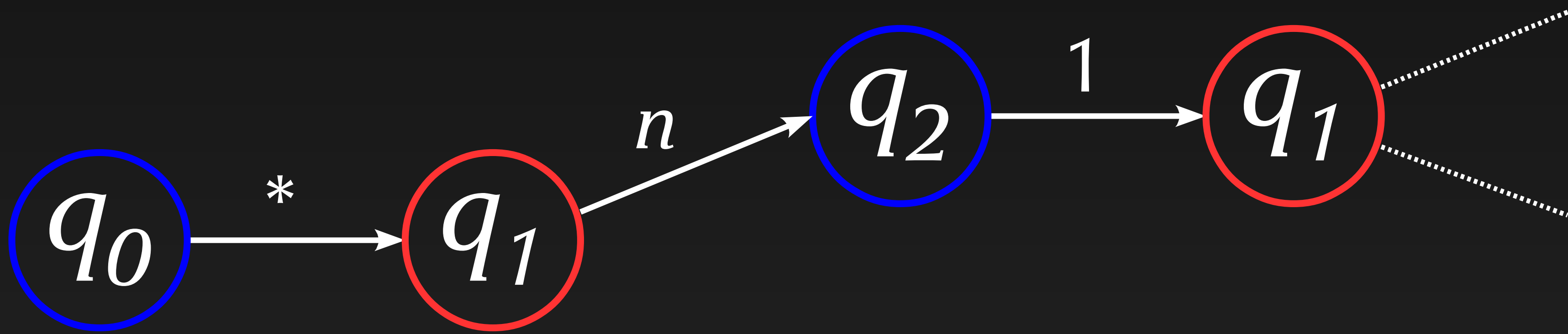
Fresh-register automata

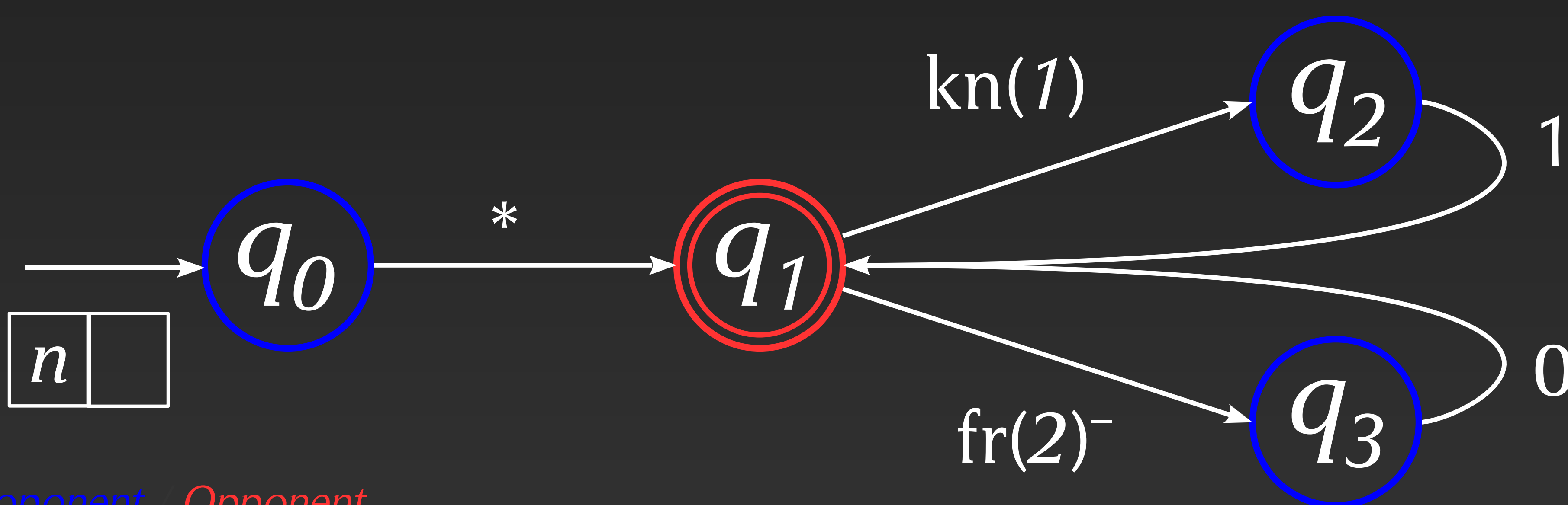




Proponent *Opponent*

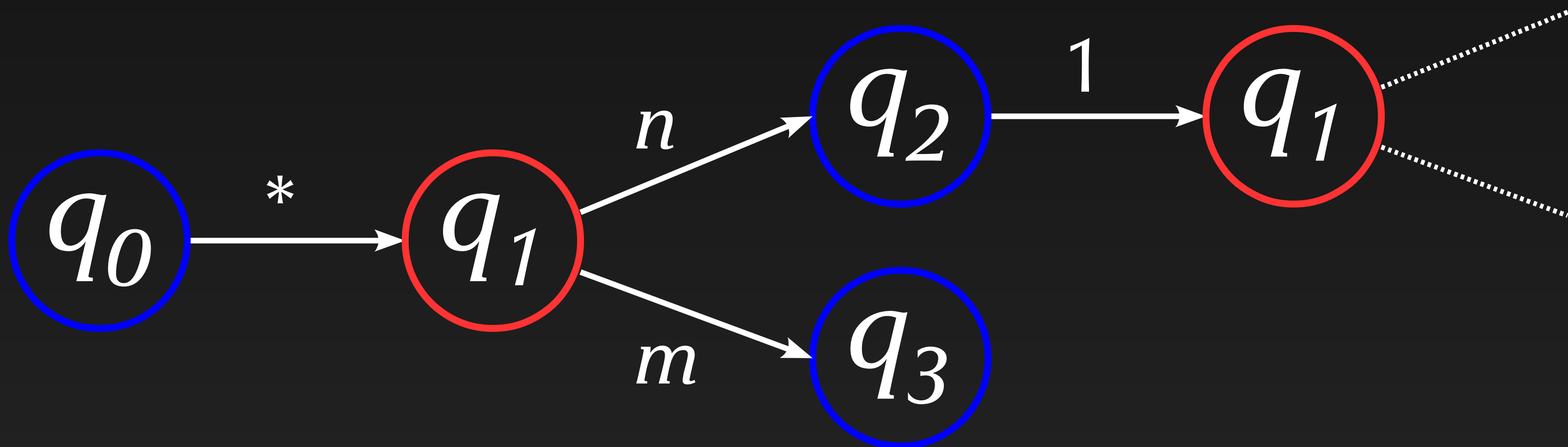
Fresh-register automata

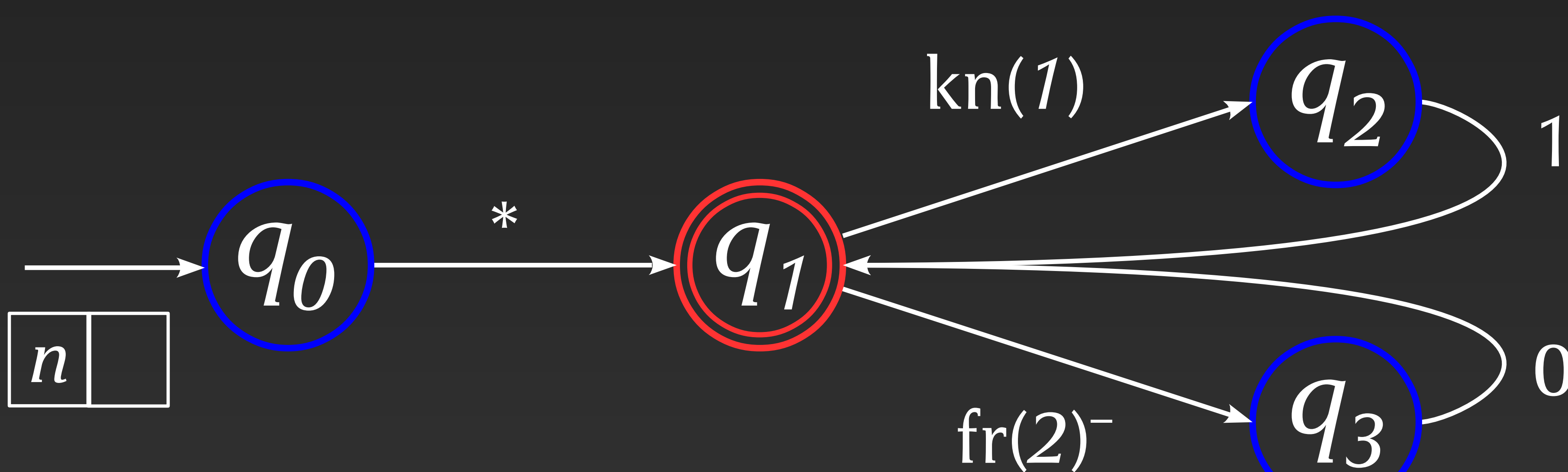




Proponent *Opponent*

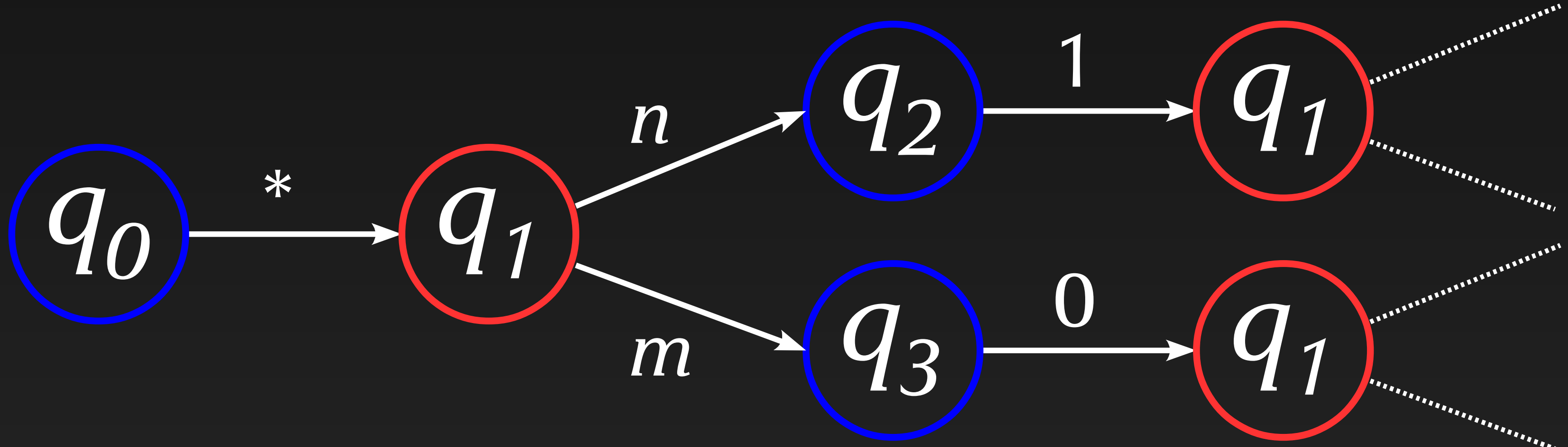
Fresh-register automata





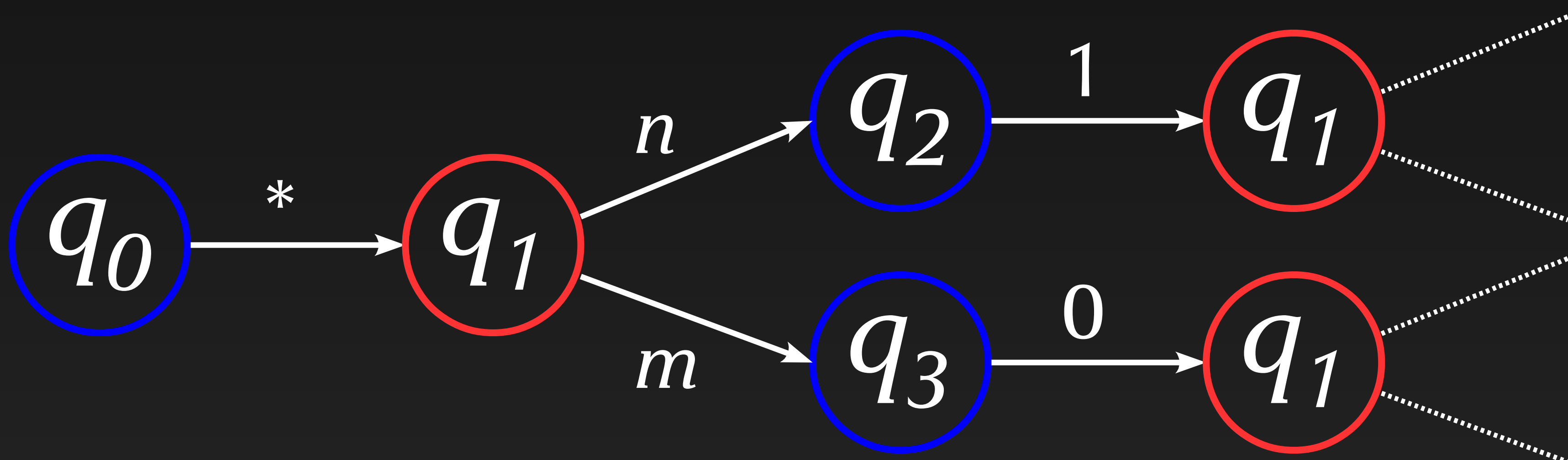
Proponent *Opponent*

Fresh-register automata

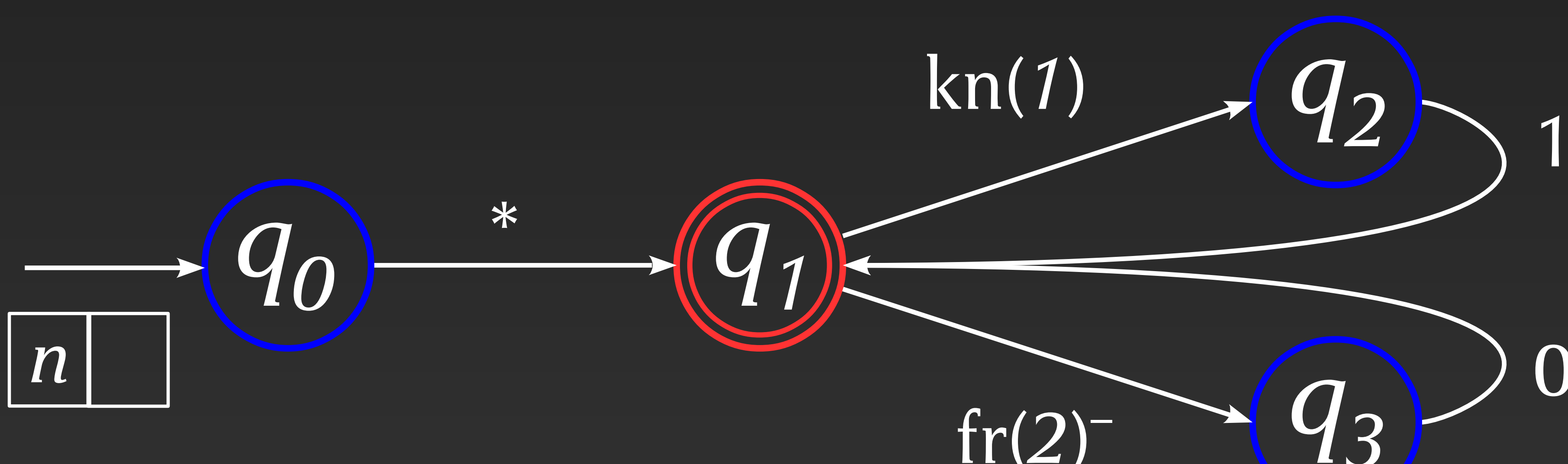


Proponent *Opponent*

Fresh-register automata



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



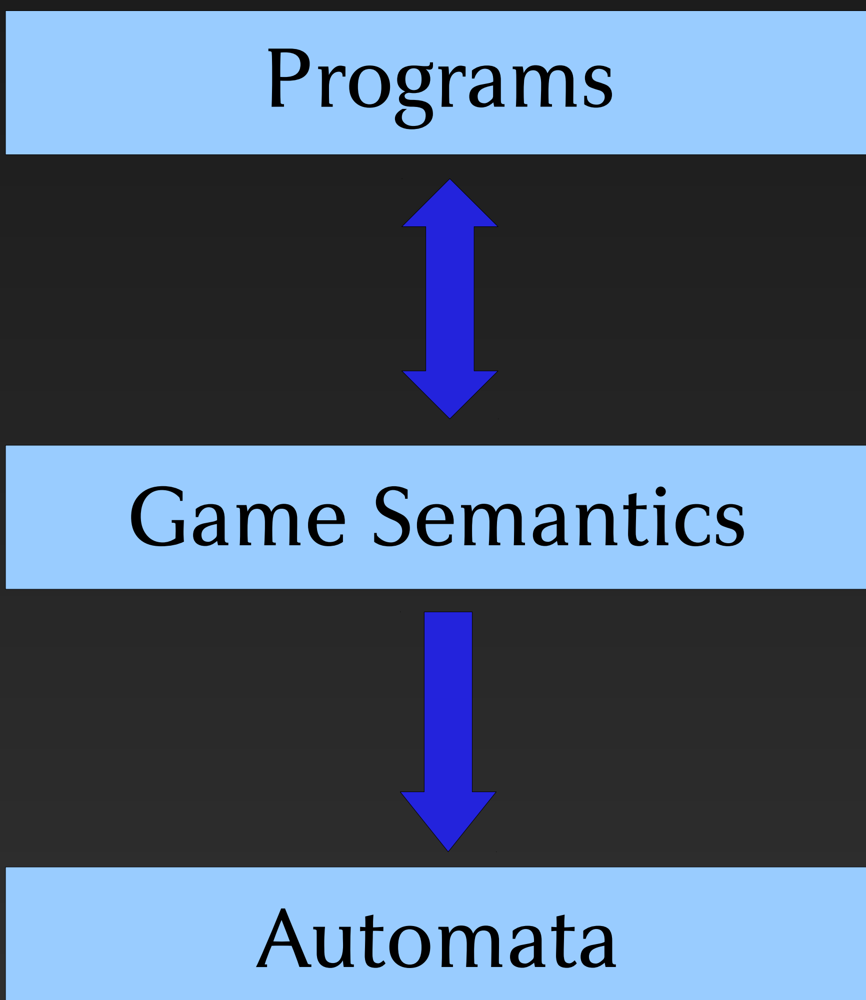
Proponent *Opponent*

Automata

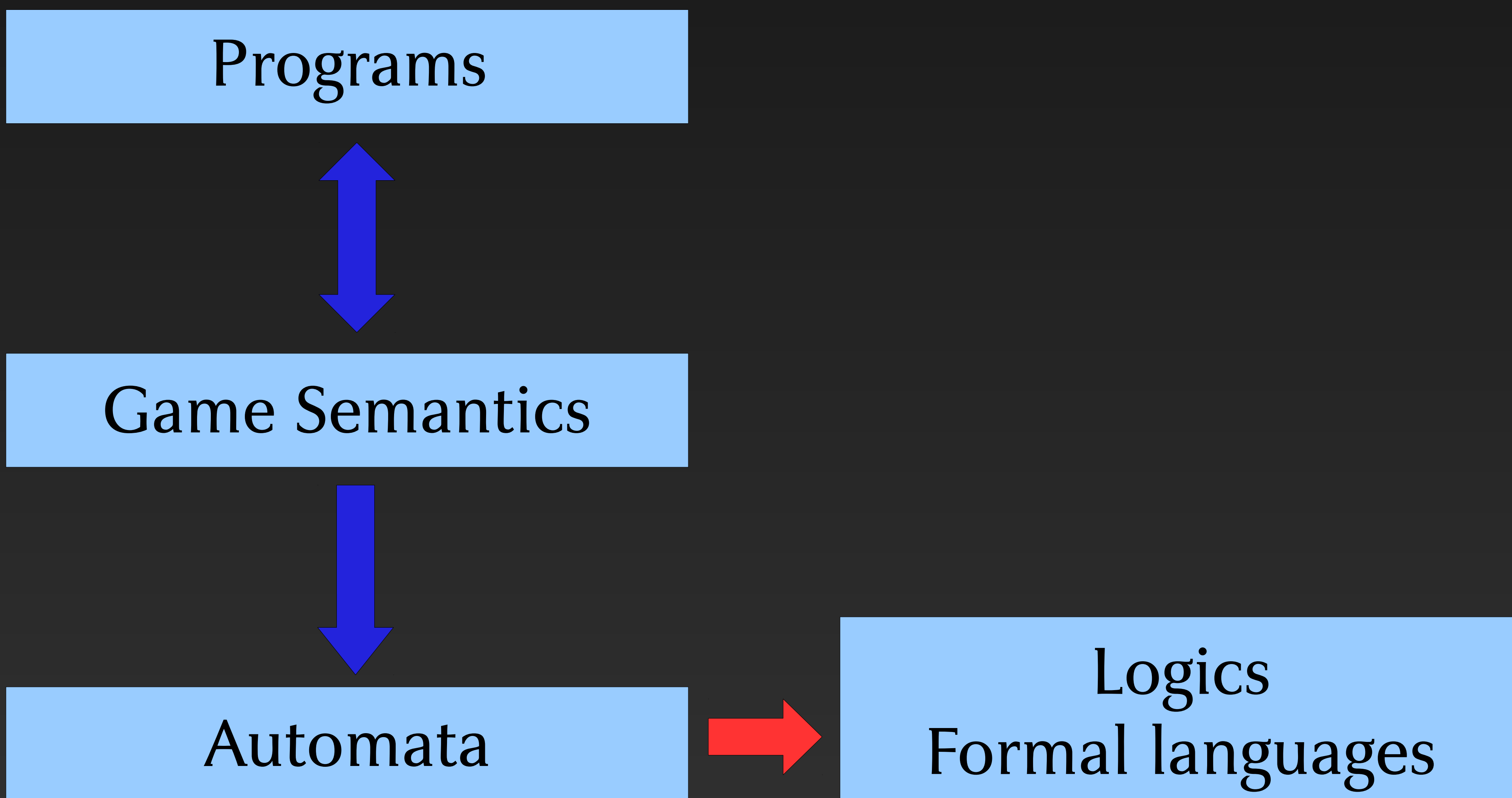
- Tz.11: Fresh-Register Automata
- Mu.Tz. 11': Algorithmic games for RedML*

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

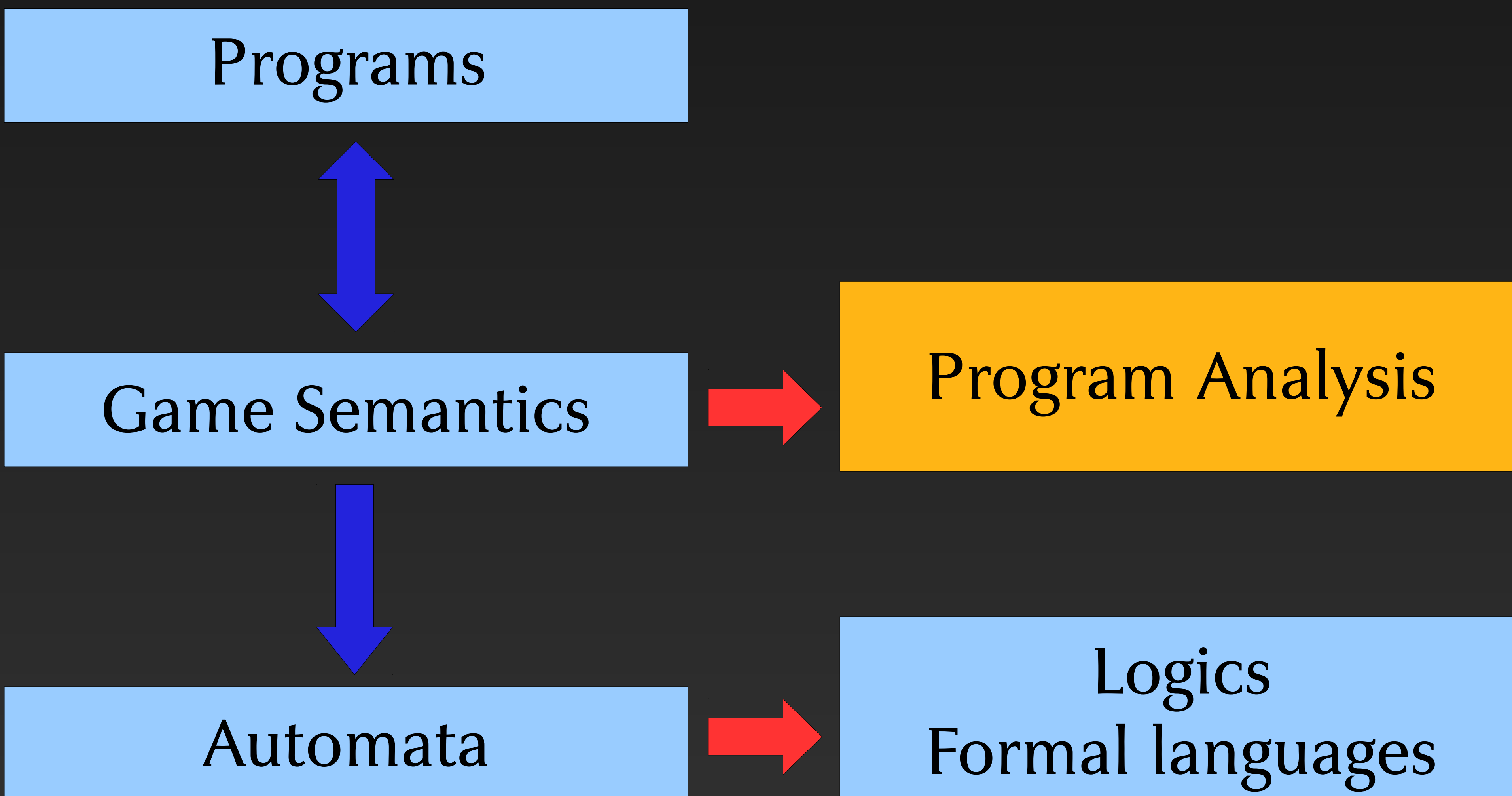
Reasoning about new resources



Reasoning about new resources



Reasoning about new resources



Game semantics for program analysis

Why games?

- Accurate models

Game semantics for program analysis

Why games?

- Accurate models
- Scalability

compositionality
modelling in isolation

Game semantics for program analysis

Why games?

- Accurate models
- Scalability — compositionality, modelling in isolation
- Automation — concrete representation

Game semantics for program analysis

Why games?

- Accurate models
- Scalability
- Automation

Directions:

- Model checking
- Control-flow analysis
- Verification logics

Game semantics for program analysis

thank you!

Why games?

- Accurate models
- Scalability
- Automation

Directions:

- Model checking
- Control-flow analysis
- Verification logics