

# Games with names

Nikos Tzevelekos

Queen Mary, University of London

# What this talk is about

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

We call these **names**

We present **game semantics** for computation with names


# Computation with names

```
 $\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$ 
```

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

# Computation with names

```
λx.ref(0) : com → intref
```



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

# Computation with names

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

$\text{let } f = [\_ ] \text{ in } \{ f() == f() \} \mapsto \text{false}$

# Example: Reduced ML

RedML = simply-typed lambda-calculus  
+ integer references

$$P \cong P'$$

# Example: Reduced ML

RedML = simply-typed lambda-calculus  
+ integer references

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

# Example: Reduced ML

RedML = simply-typed lambda-calculus  
+ integer references

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

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



# Example: Reduced ML

RedML = simply-typed lambda-calculus  
+ integer references

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

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

$f : \text{intref} \rightarrow \text{int} \vdash \lambda y. f(\text{ref}(0))$

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

# Example: Reduced ML

RedML = simply-typed lambda-calculus  
+ integer references

$$\text{let } x = \text{ref}(0) \text{ in } \lambda y. (x == y) \quad \cong \quad \lambda y. \text{false} : \text{intref} \rightarrow \text{int}$$
$$\lambda y. \text{ref}(0) \quad \not\cong \quad \text{let } x = \text{ref}(0) \text{ in } (\lambda y. x) : \text{com} \rightarrow \text{intref}$$
$$f : \text{intref} \rightarrow \text{int} \vdash \lambda y. f(\text{ref}(0)) \\ \cong \quad \text{let } x = \text{ref}(0) \text{ in } \lambda y. x := 0; f(x) : \text{com} \rightarrow \text{int}$$
$$f : \text{intref} \rightarrow \text{com} \vdash \text{let } x, y = \text{ref}(0) \text{ in } f(x); (y := !x); y \\ \cong \quad \text{let } x = \text{ref}(0) \text{ in } f(x); x : \text{intref}$$

# Two ways to model references

Reynolds

- *Idealized Algol (1978)*

References are *pairs*:

`intref =`  
`(com  $\rightarrow$  int)  $\times$  (int  $\rightarrow$  com)`

$\longmapsto$  `(1  $\rightarrow$  Z)  $\times$  (Z  $\rightarrow$  1)`

# Two ways to model references

## Reynolds

- *Idealized Algol (1978)*

References are *pairs*:

$\text{intref} =$   
 $(\text{com} \rightarrow \text{int}) \times (\text{int} \rightarrow \text{com})$

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

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

# Two ways to model references

## Reynolds

- *Idealized Algol (1978)*

References are *pairs*:

`intref =`  
`(com  $\rightarrow$  int)  $\times$  (int  $\rightarrow$  com)`  
 $\longmapsto$  `(1  $\rightarrow$  Z)  $\times$  (Z  $\rightarrow$  1)`

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

## Pitts & Stark

- *nu-calculus (1993)*

References are *names*:

`intref = base type`  
 $\longmapsto$  `N (names)`

# Two ways to model references

## Reynolds

- *Idealized Algol (1978)*

References are *pairs*:

$\text{intref} =$   
 $(\text{com} \rightarrow \text{int}) \times (\text{int} \rightarrow \text{com})$   
 $\longmapsto (\mathbf{1} \rightarrow \mathbf{Z}) \times (\mathbf{Z} \rightarrow \mathbf{1})$

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

## Pitts & Stark

- *nu-calculus (1993)*

References are *names*:

$\text{intref} = \text{base type}$   
 $\longmapsto \mathbf{N}$  (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( $\lambda x. 3, \lambda x. ()$ )`

- 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`

# Game Semantics



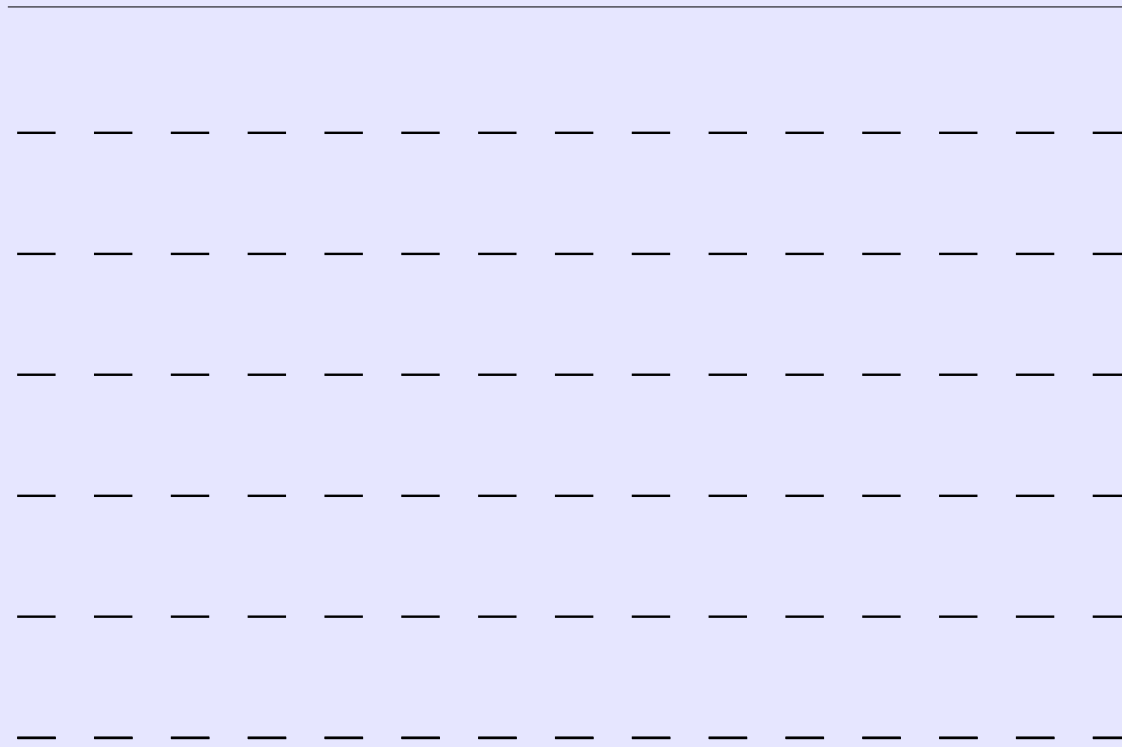
# Game Semantics

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

# Example

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

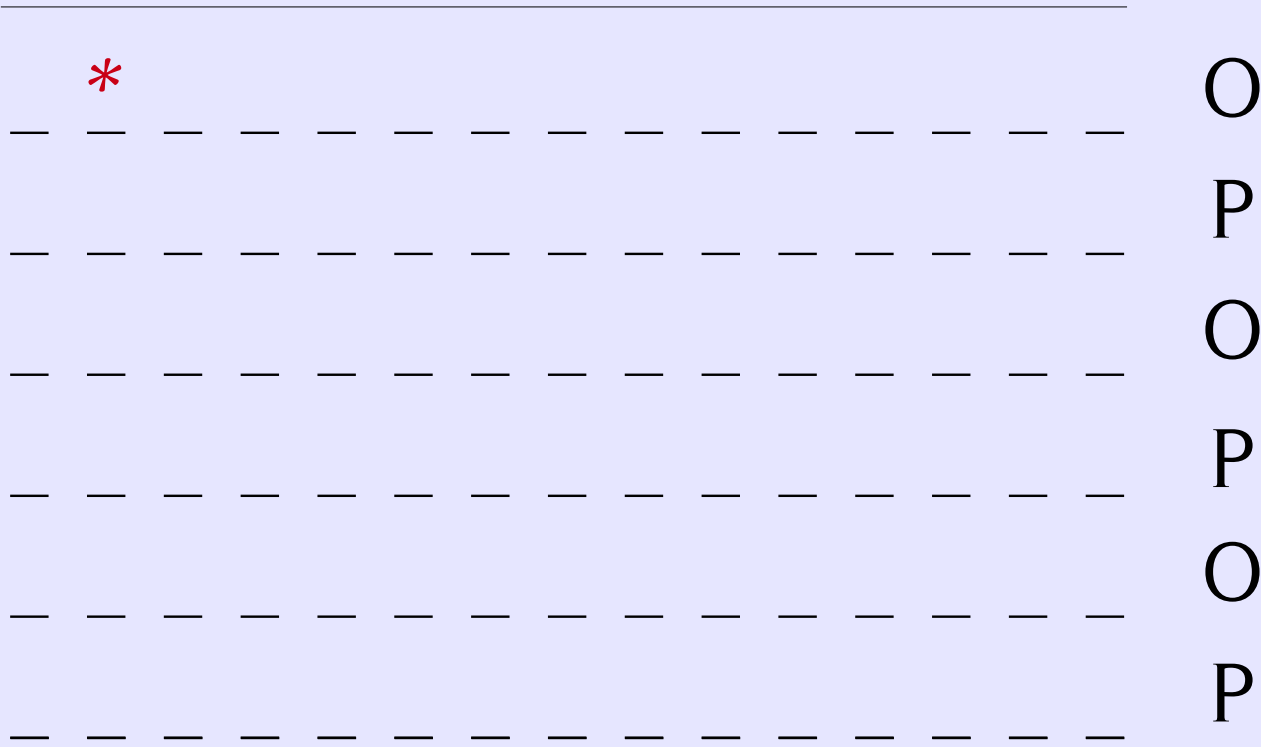
$1 \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

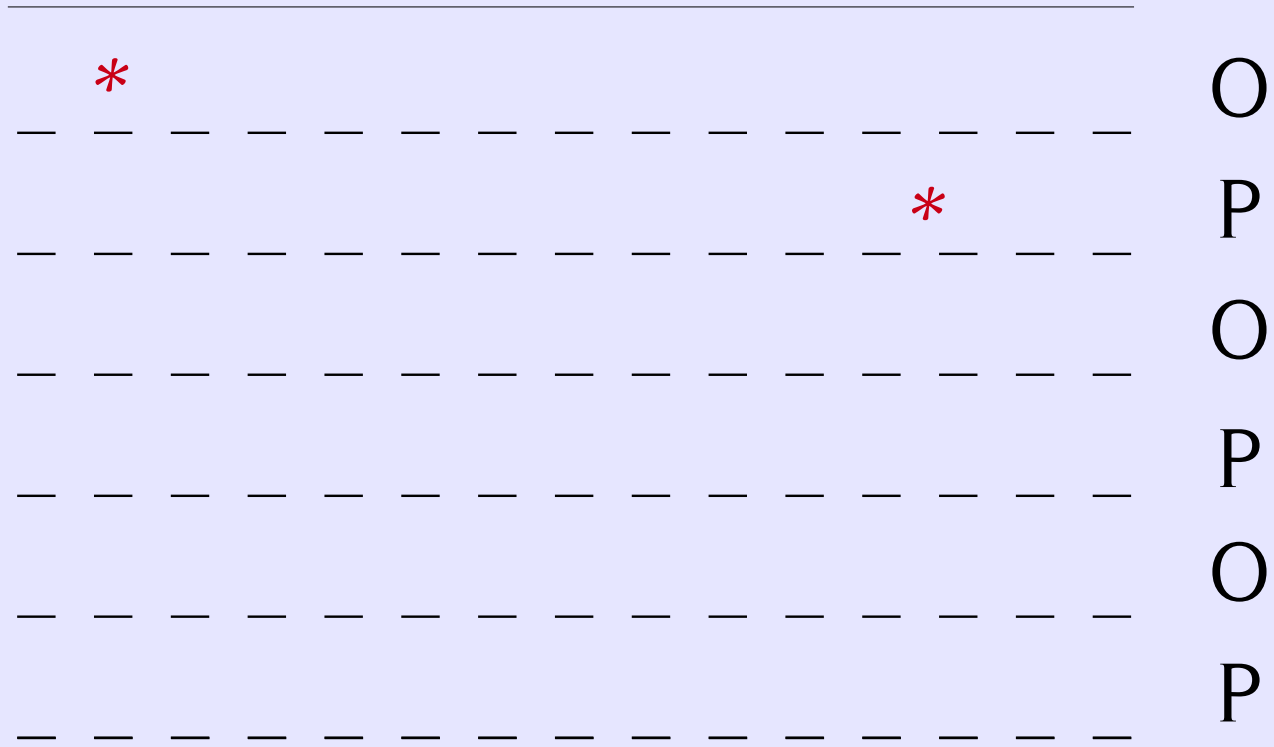
$1 \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

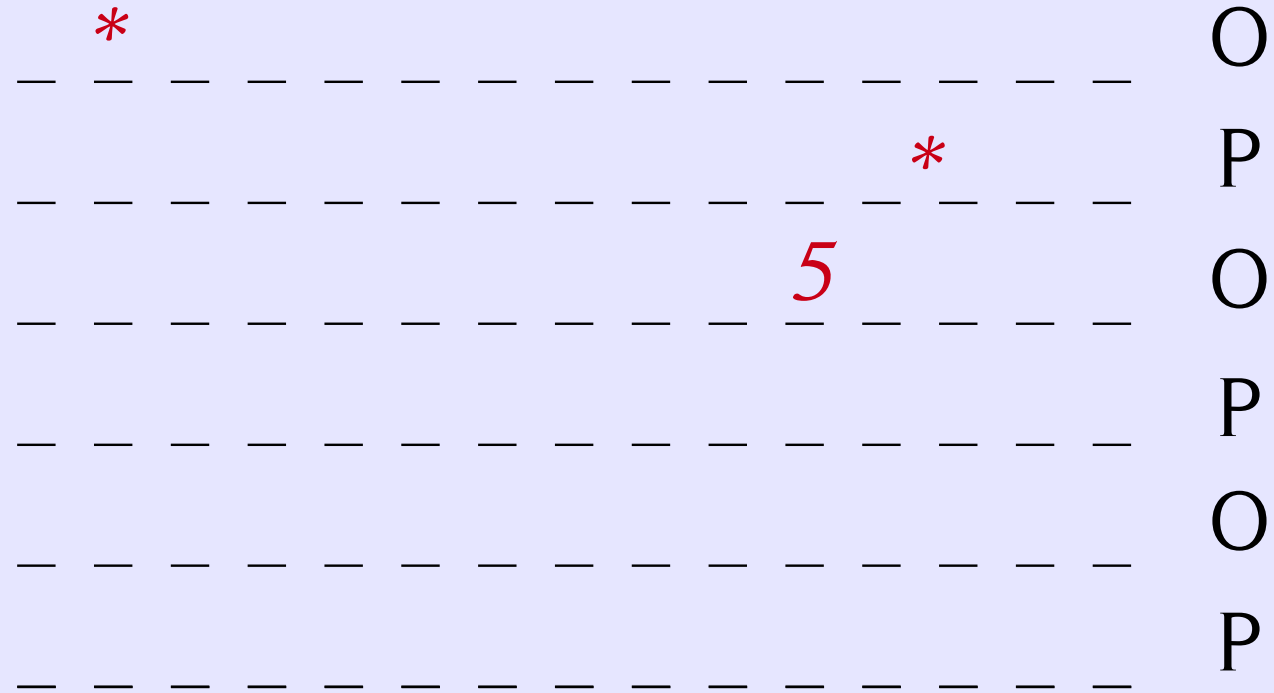
$1 \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

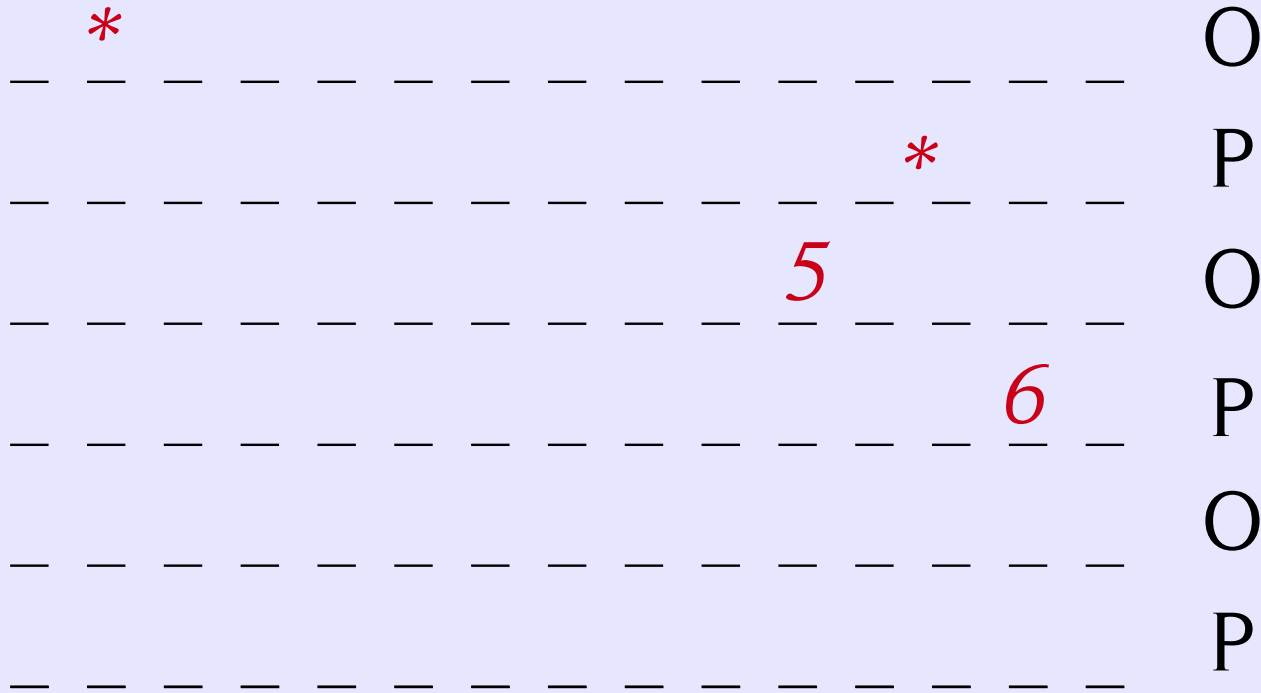
$1 \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$





# Example

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

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

---

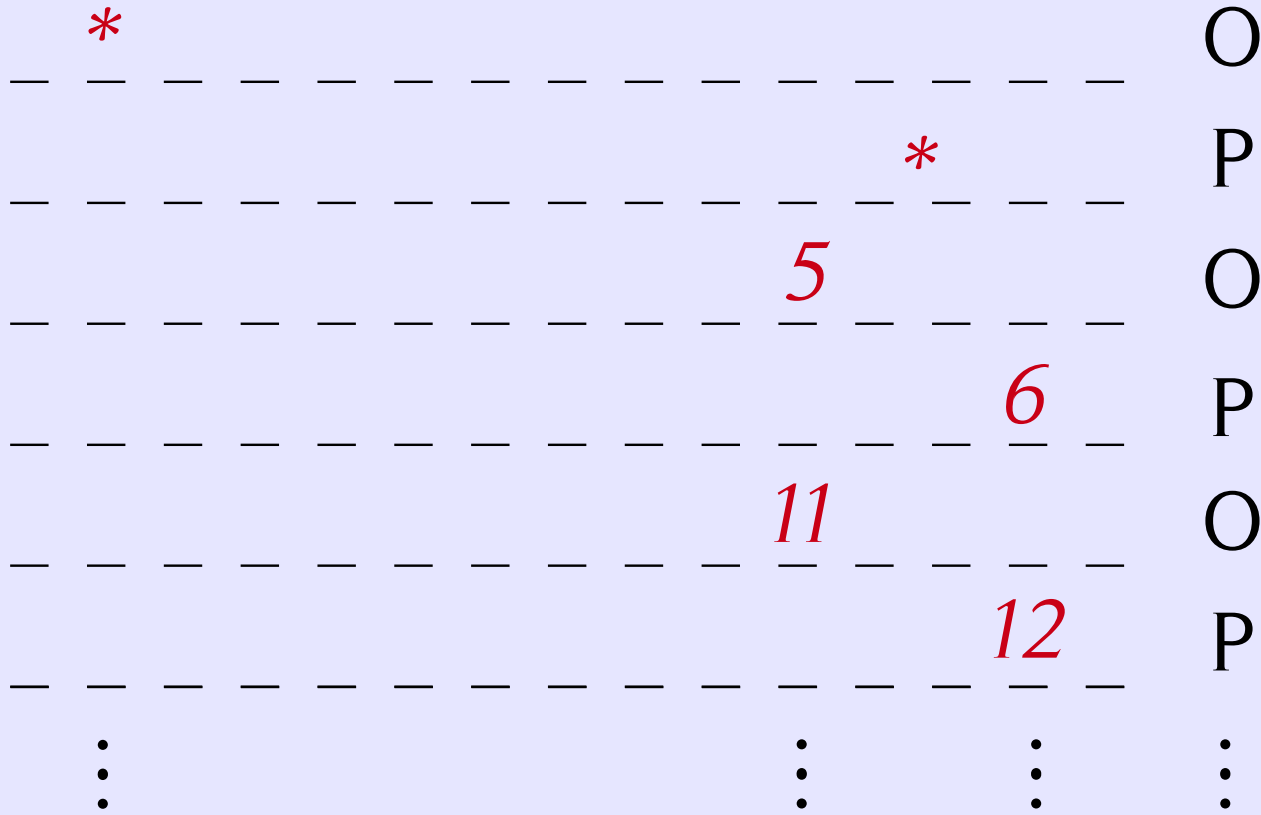
*												O
								*				P
							5					O
									6			P
							11					O
									12			P



# Example

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

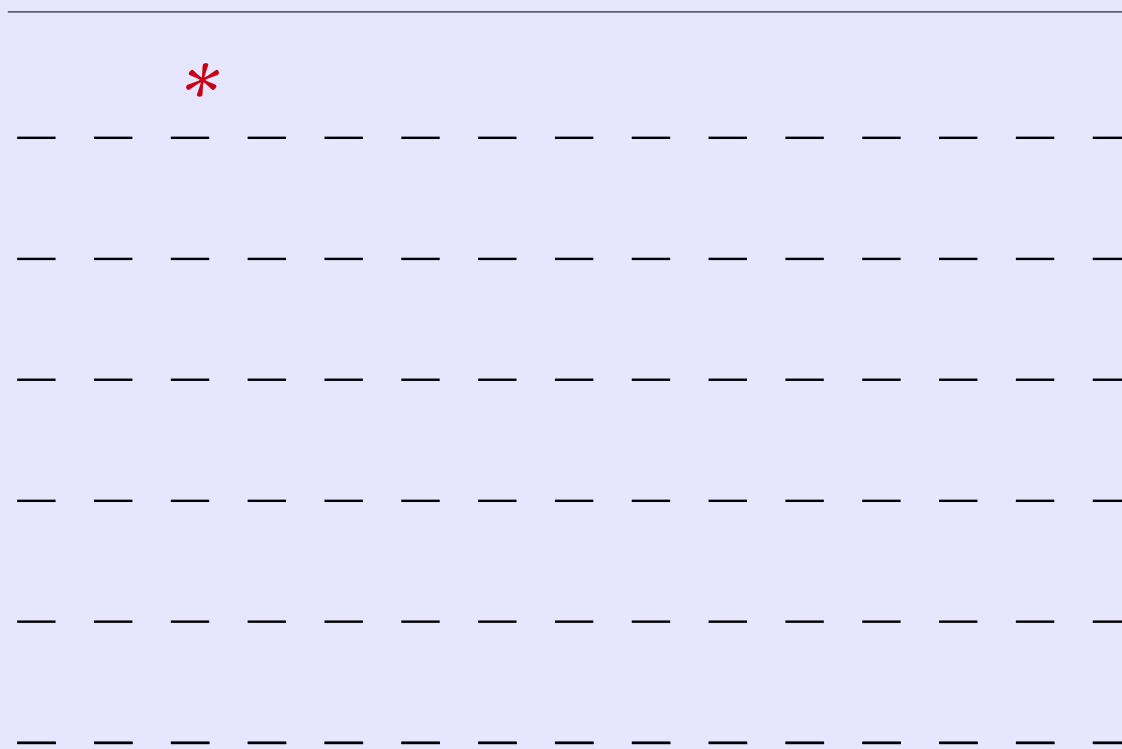
$1 \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$

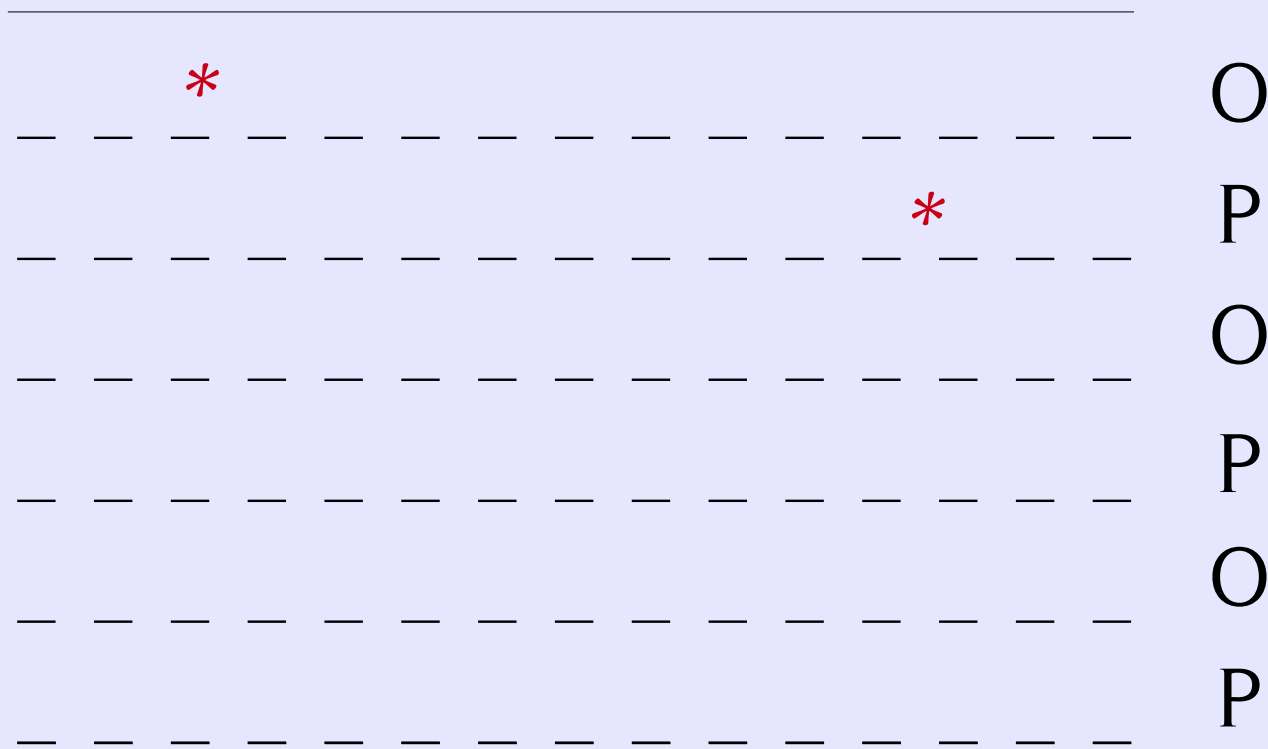


O  
P  
O  
P  
O  
P

# Example

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

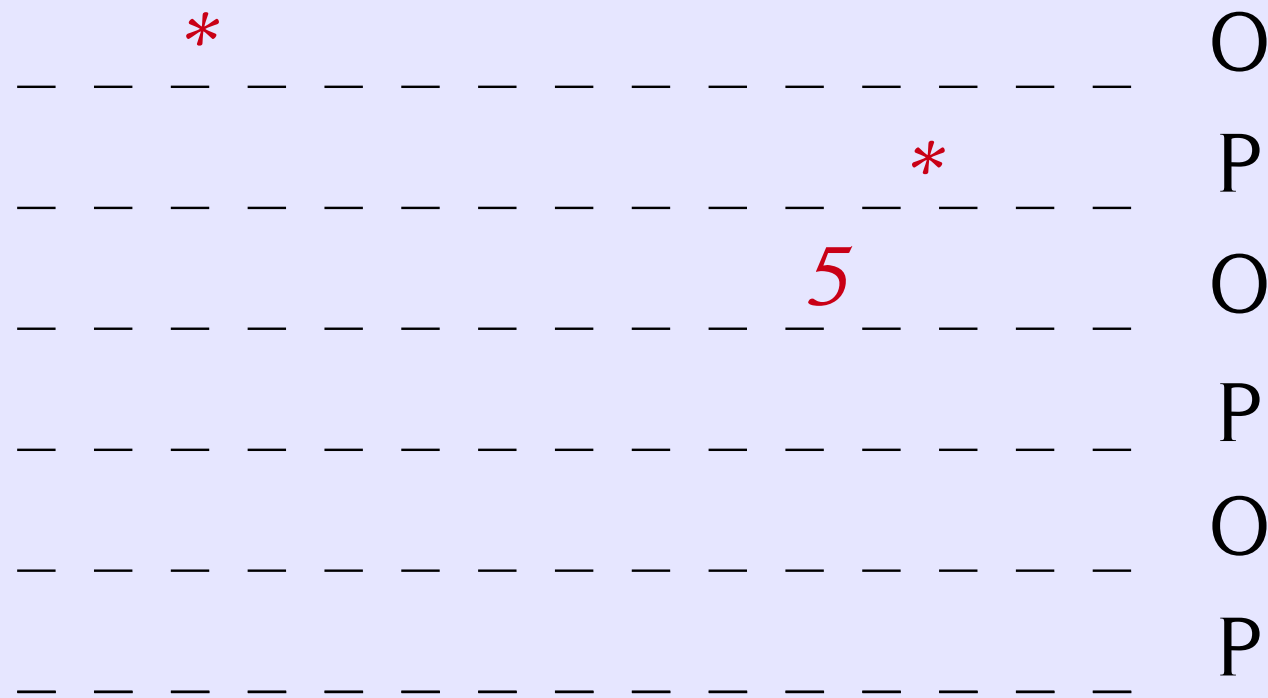
$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

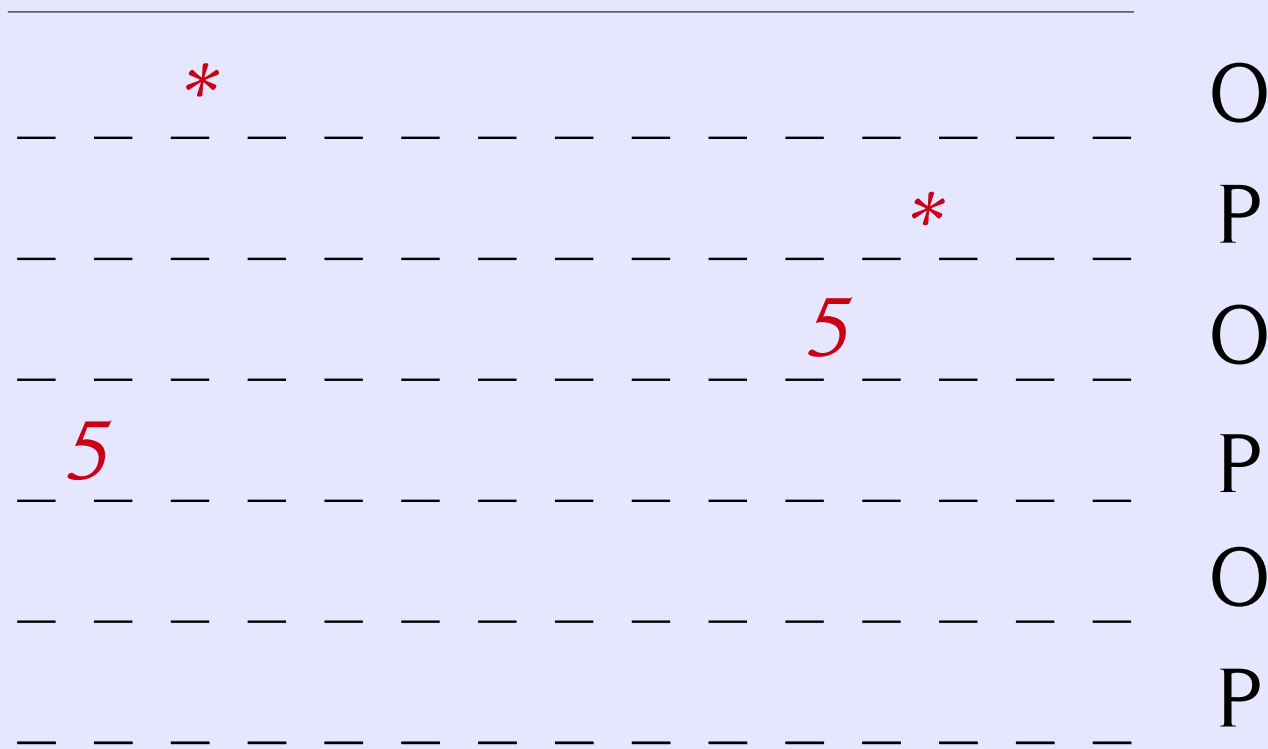
$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

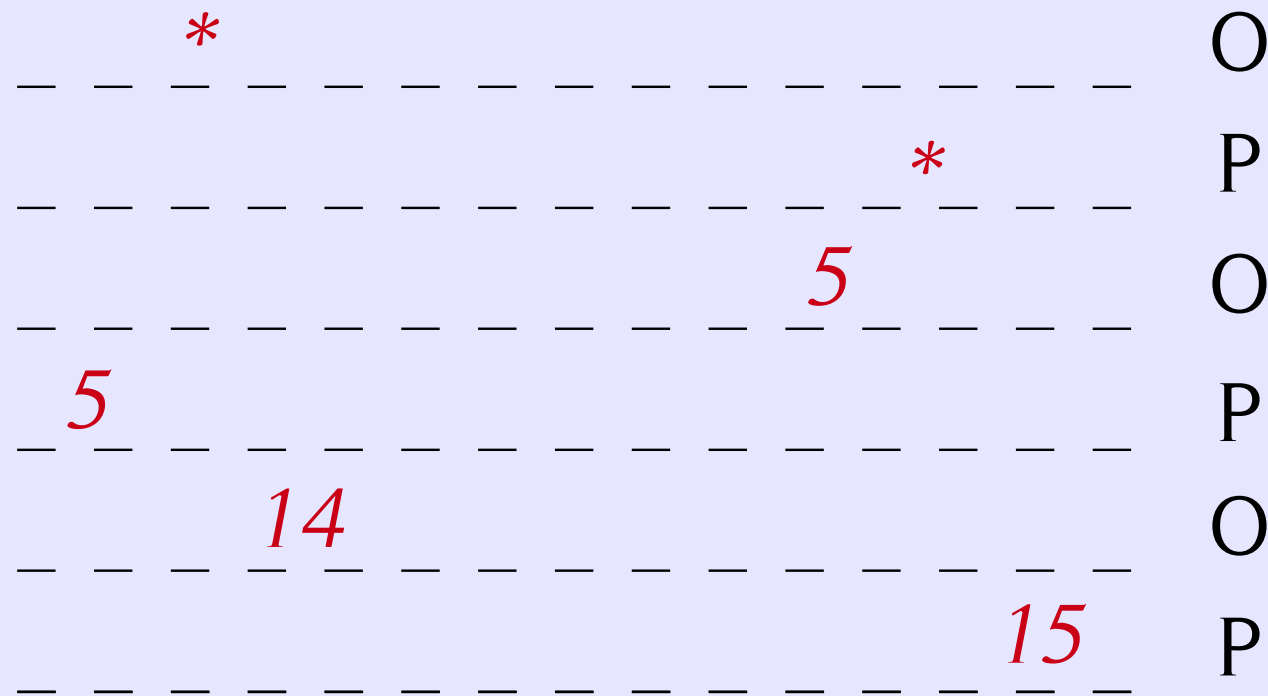
$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Example

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

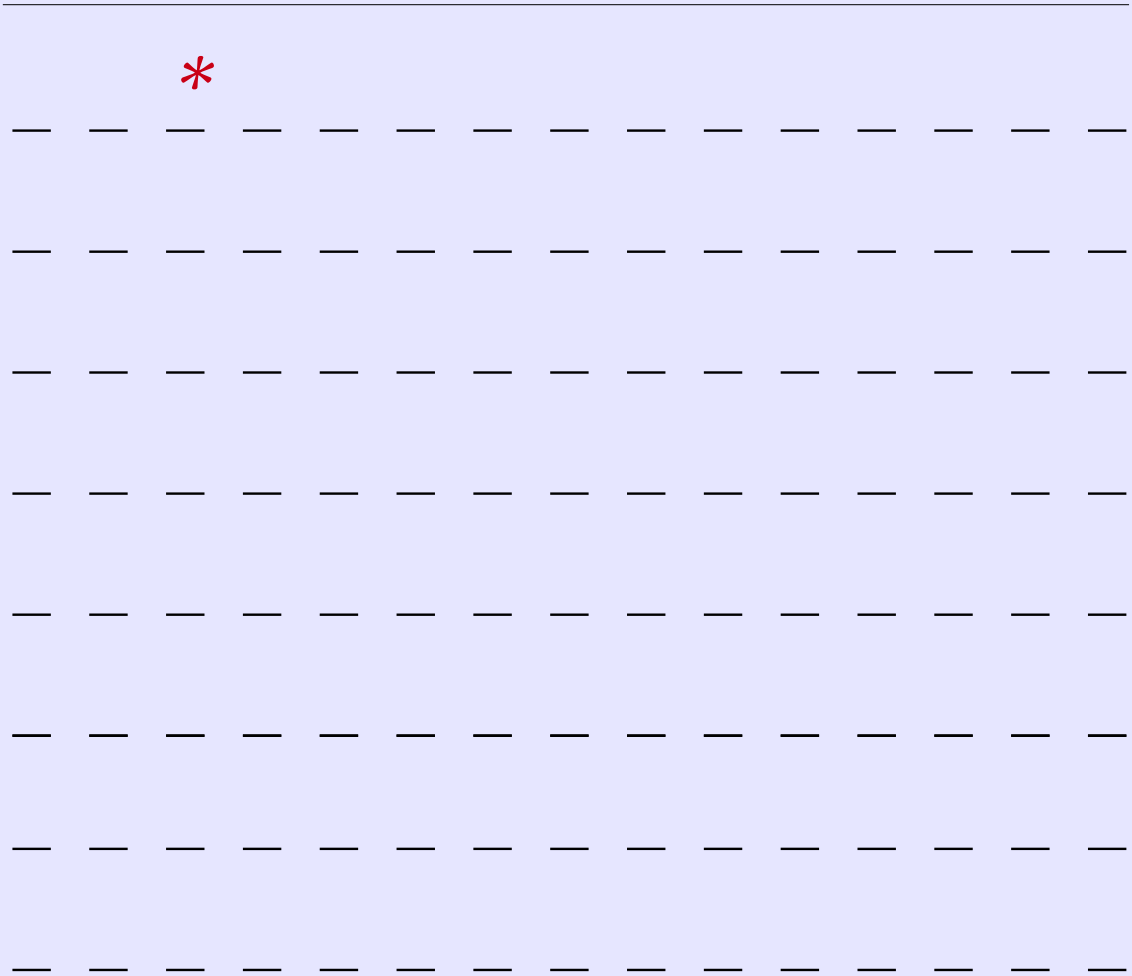
$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$





# Example

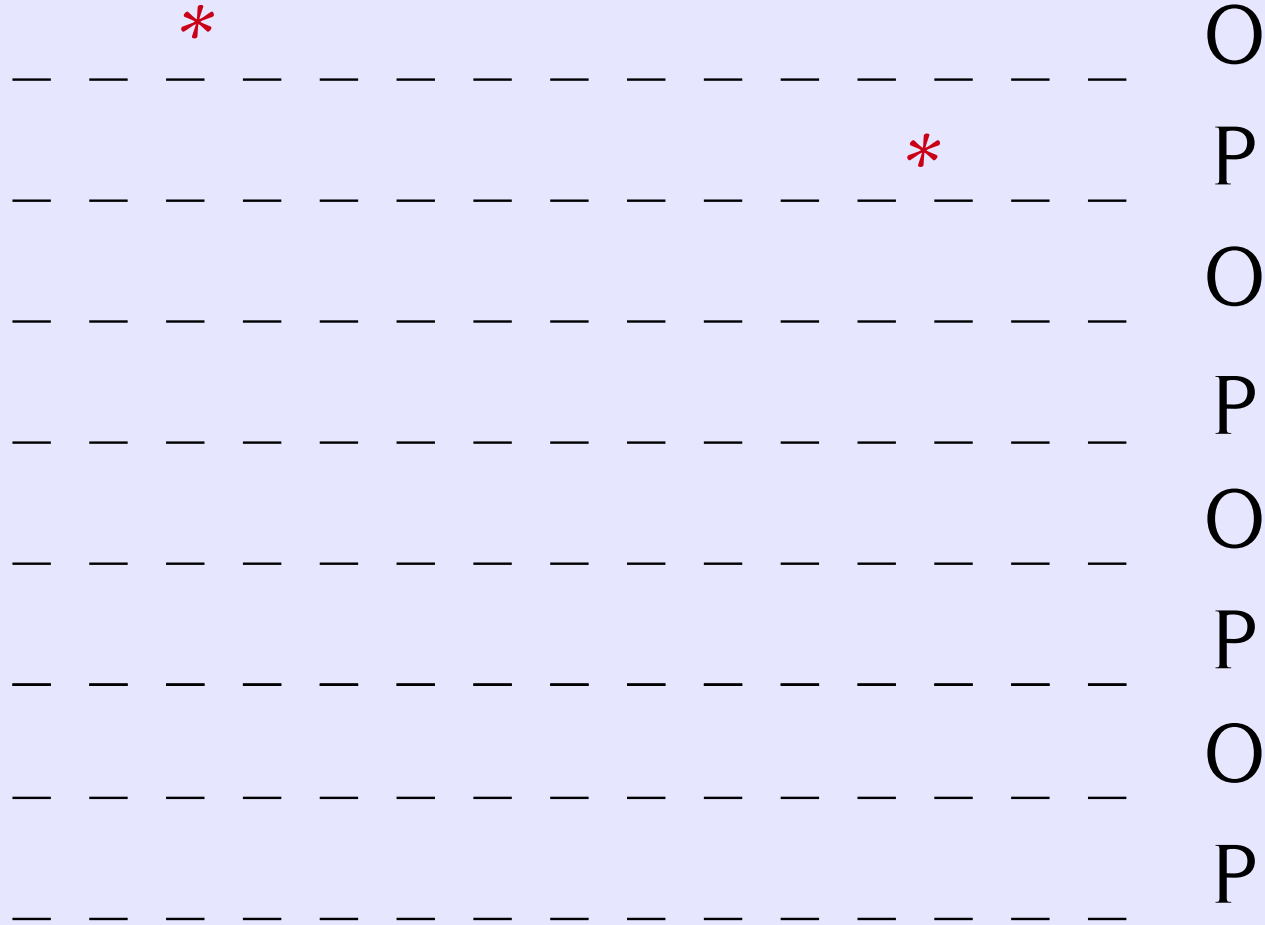
$Int \rightarrow Int \longrightarrow Int \rightarrow Int$



O  
P  
O  
P  
O  
P  
O  
P

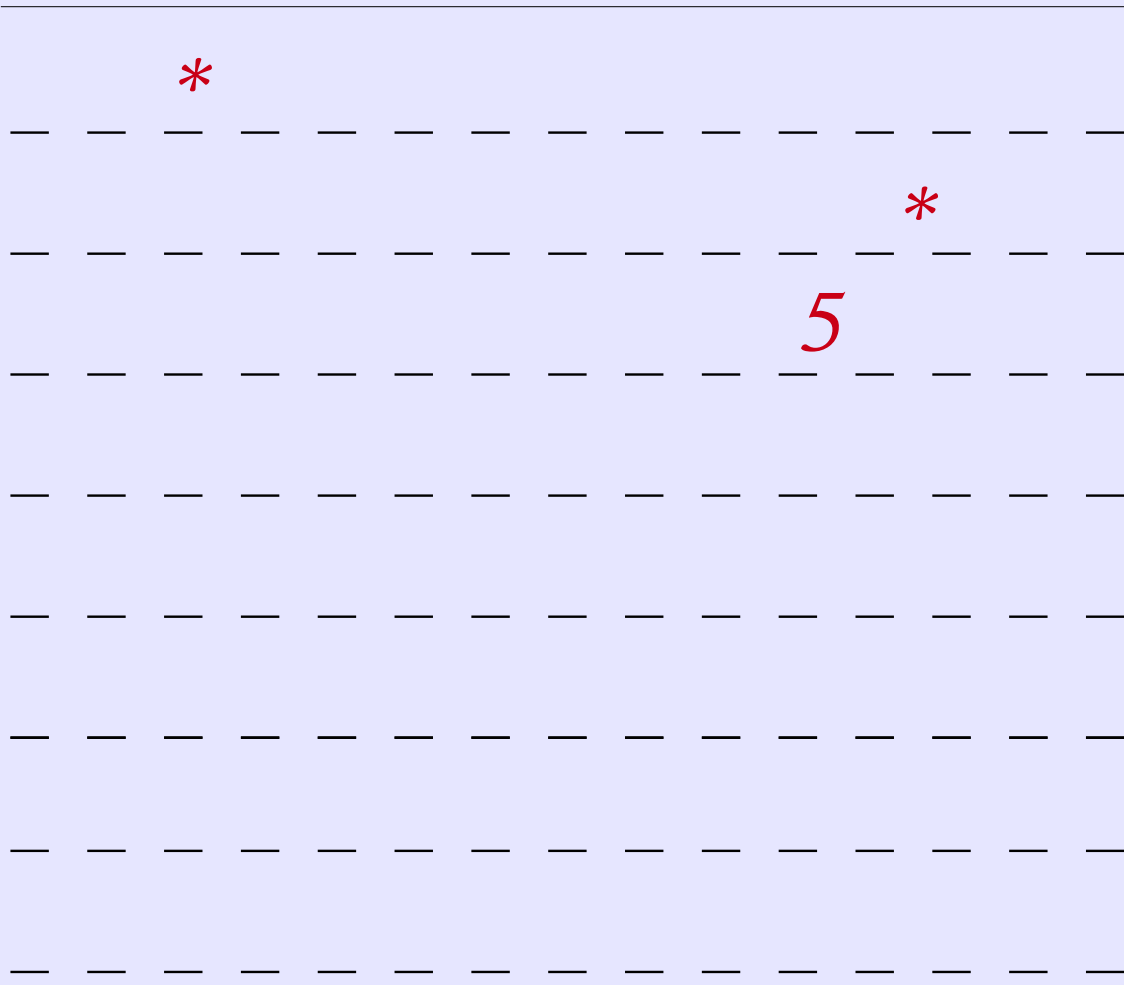
# Example

$Int \rightarrow Int \longrightarrow Int \rightarrow Int$



# Example

$Int \rightarrow Int \longrightarrow Int \rightarrow Int$



O  
P  
O  
P  
O  
P  
O  
P

# Example

$Int \rightarrow Int \longrightarrow Int \rightarrow Int$





# Example

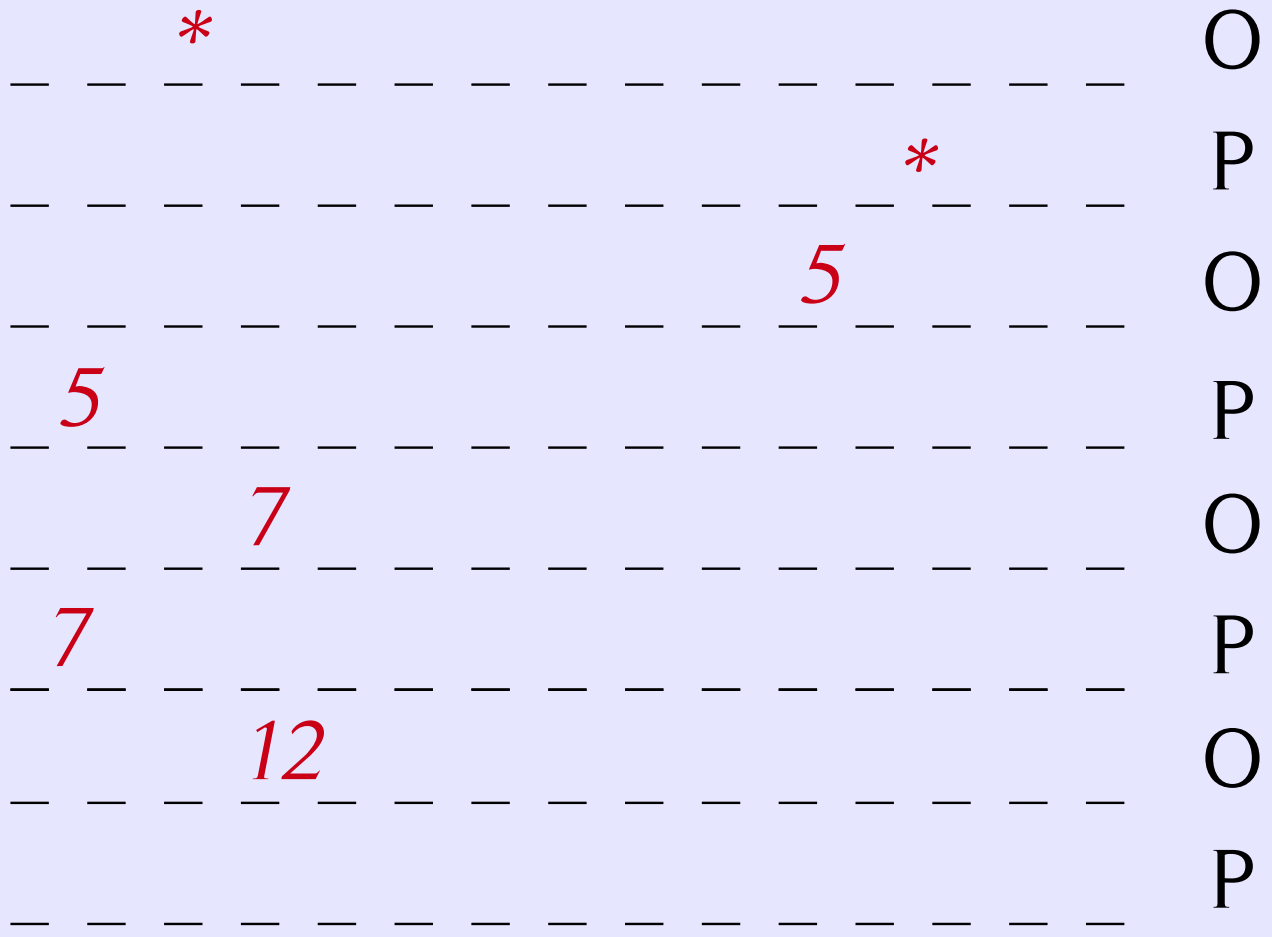
$Int \rightarrow Int \longrightarrow Int \rightarrow Int$



# Example

$Int \rightarrow Int \longrightarrow Int \rightarrow Int$

---

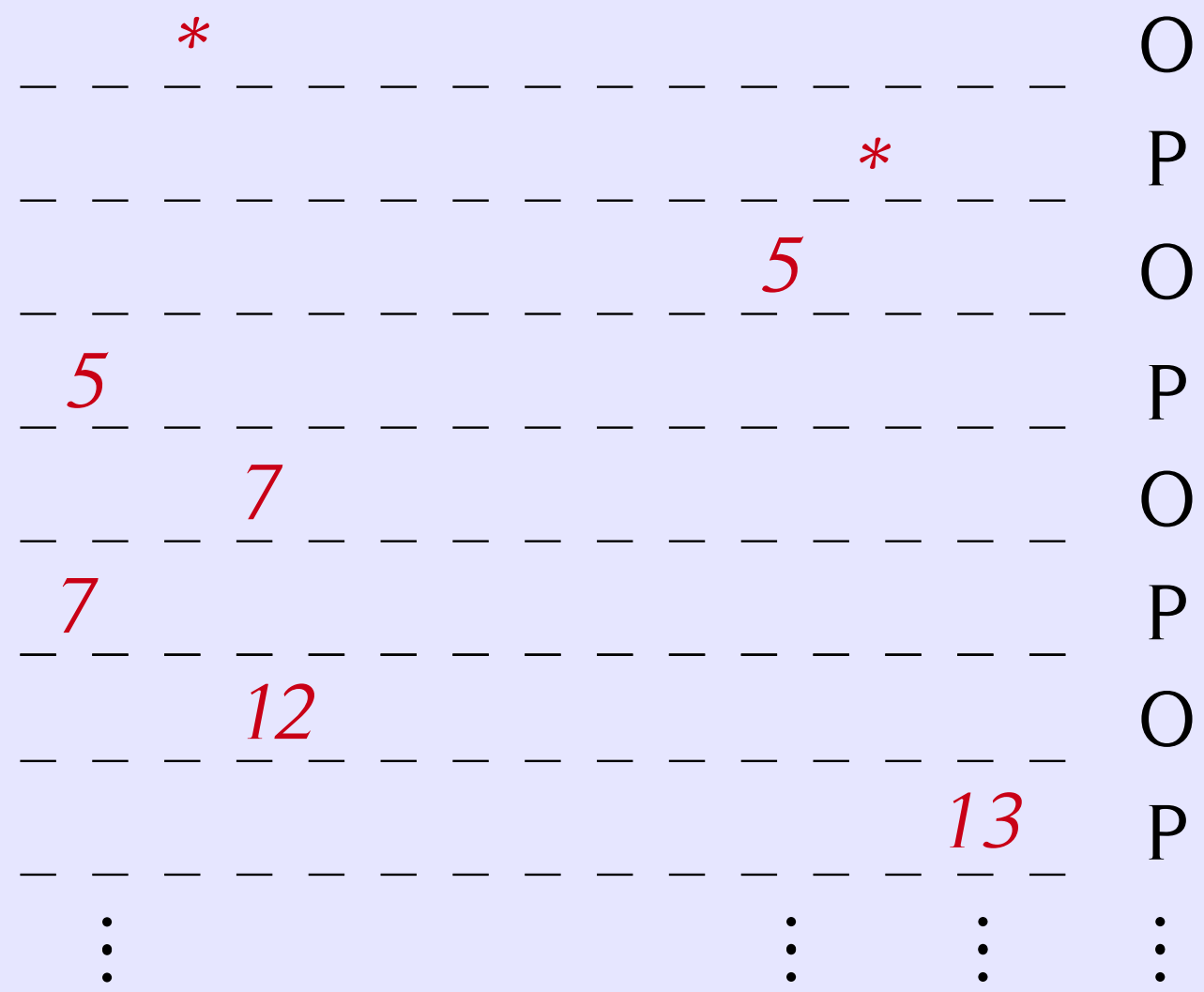






# Example

$Int \rightarrow Int \longrightarrow Int \rightarrow Int$



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

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



# Game Semantics

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

# Composition

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

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

# Composition

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

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$

*\**

*\**

*5*

*6*

*11*

*12*

$\vdots$

$\vdots$

$\vdots$

*\**

*\**

*5*

*5*

*14*

*15*

$\vdots$

$\vdots$

$\vdots$

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

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$

*\**

*\**

*5*

*6*

*11*

*12*

*⋮*   *⋮*   *⋮*

*\**

*\**

*5*

*5*

*14*

*15*

*⋮*   *⋮*   *⋮*

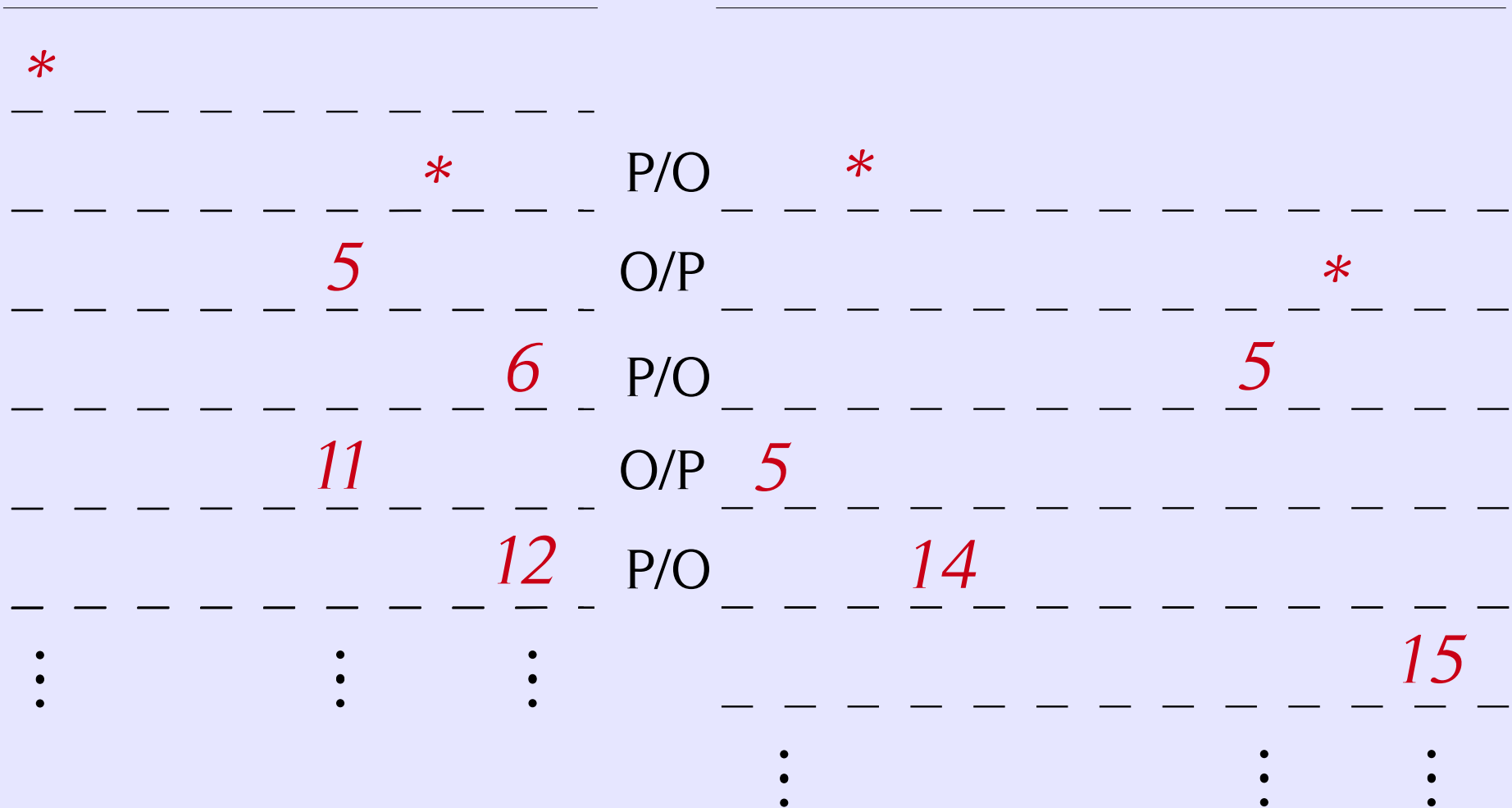
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$









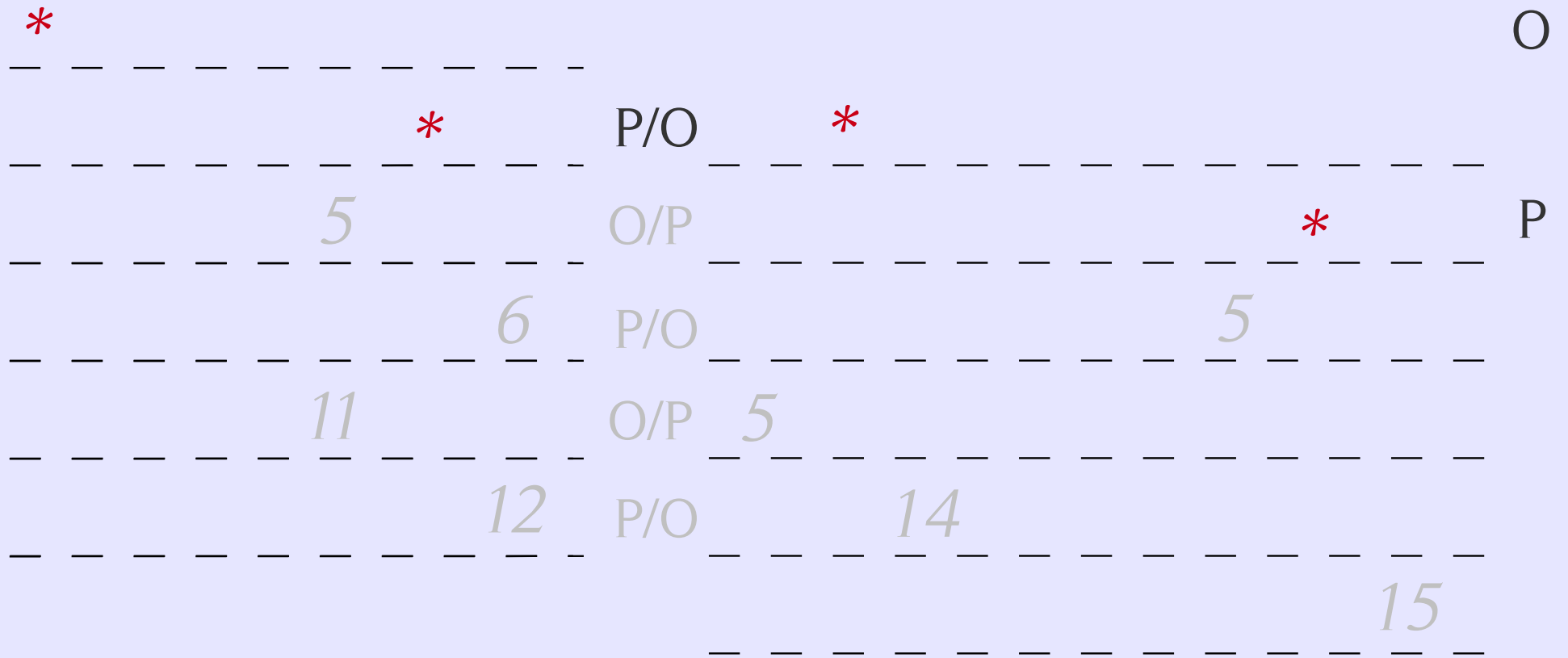
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$







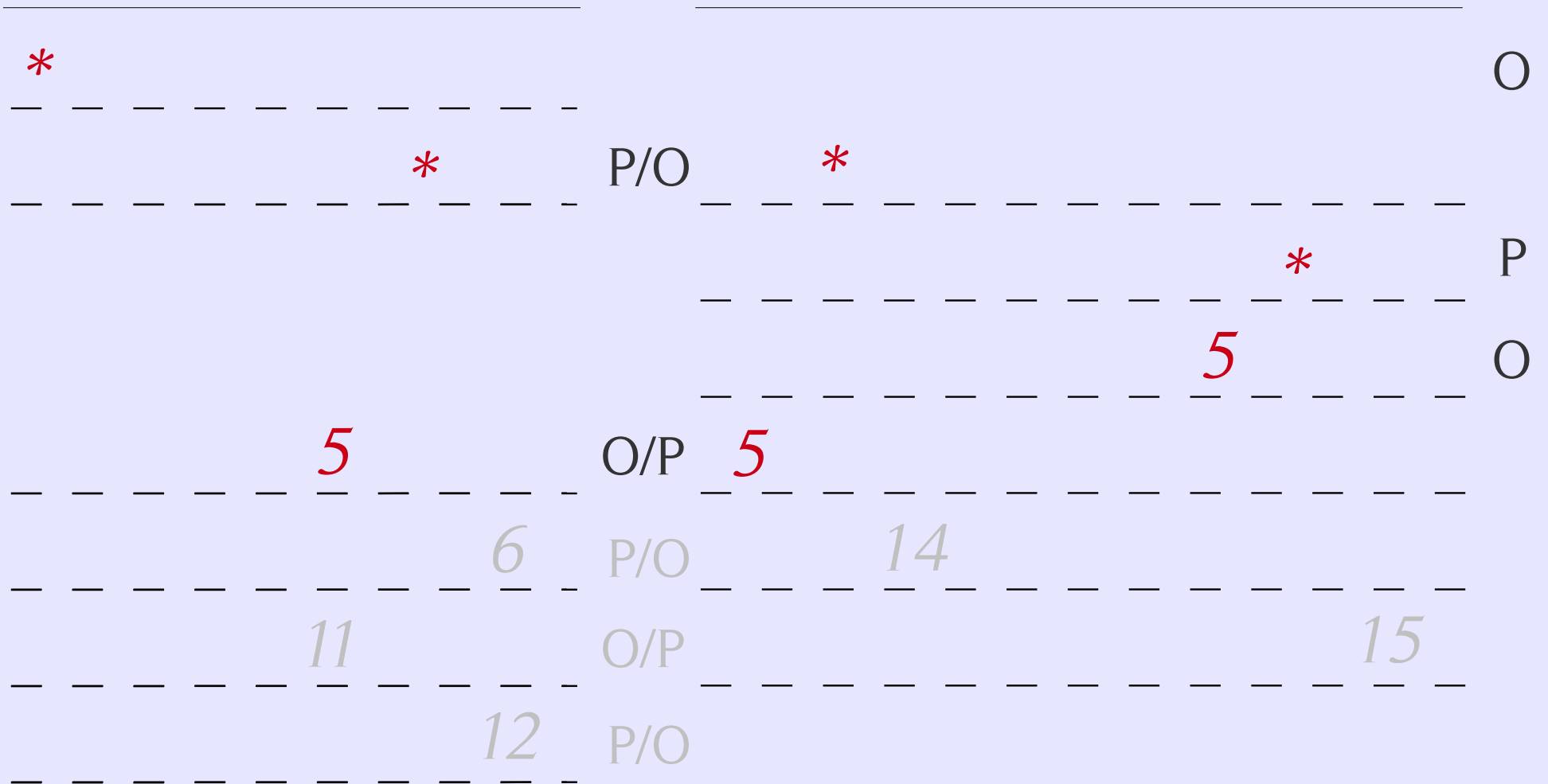
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



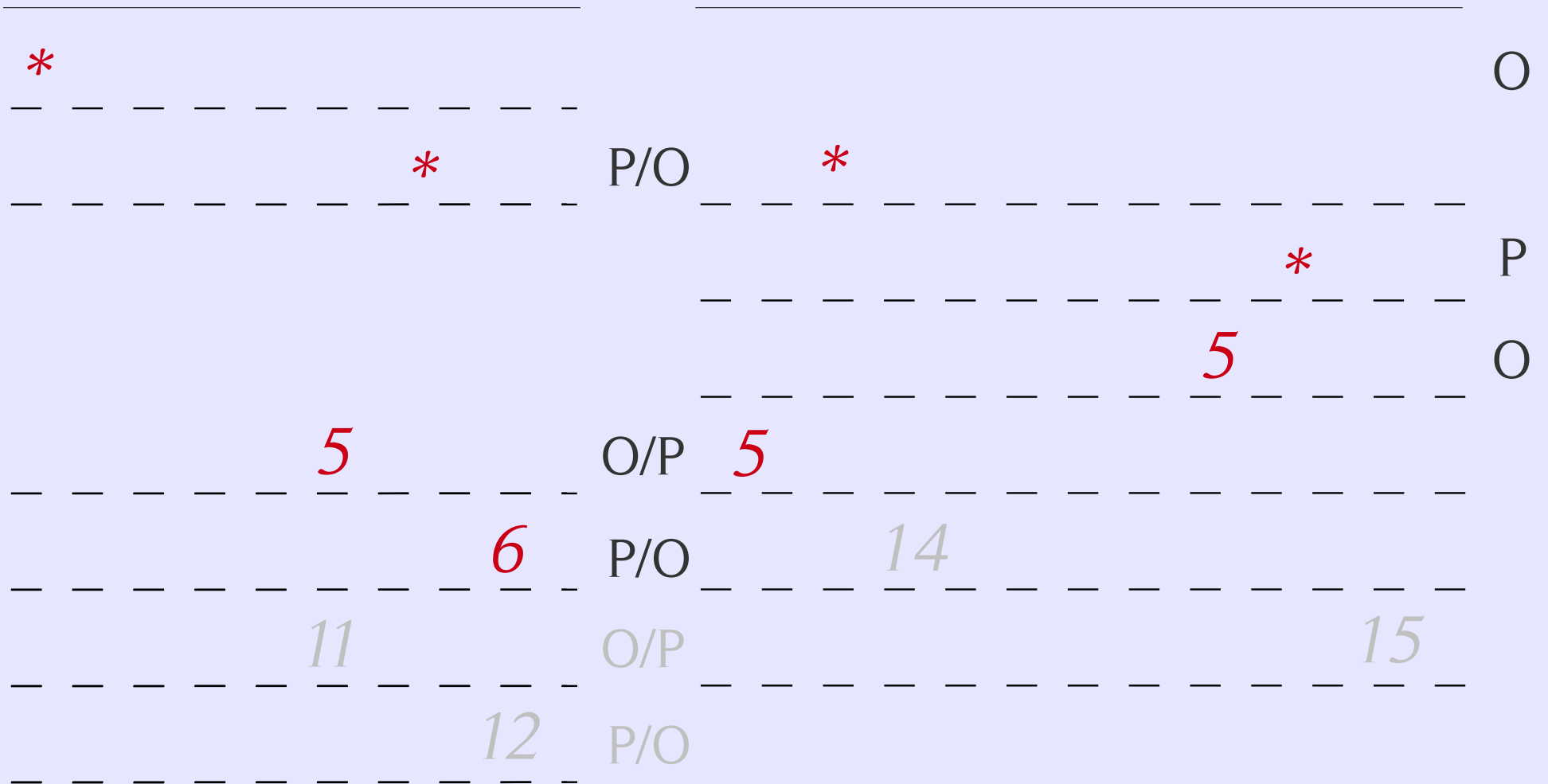
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



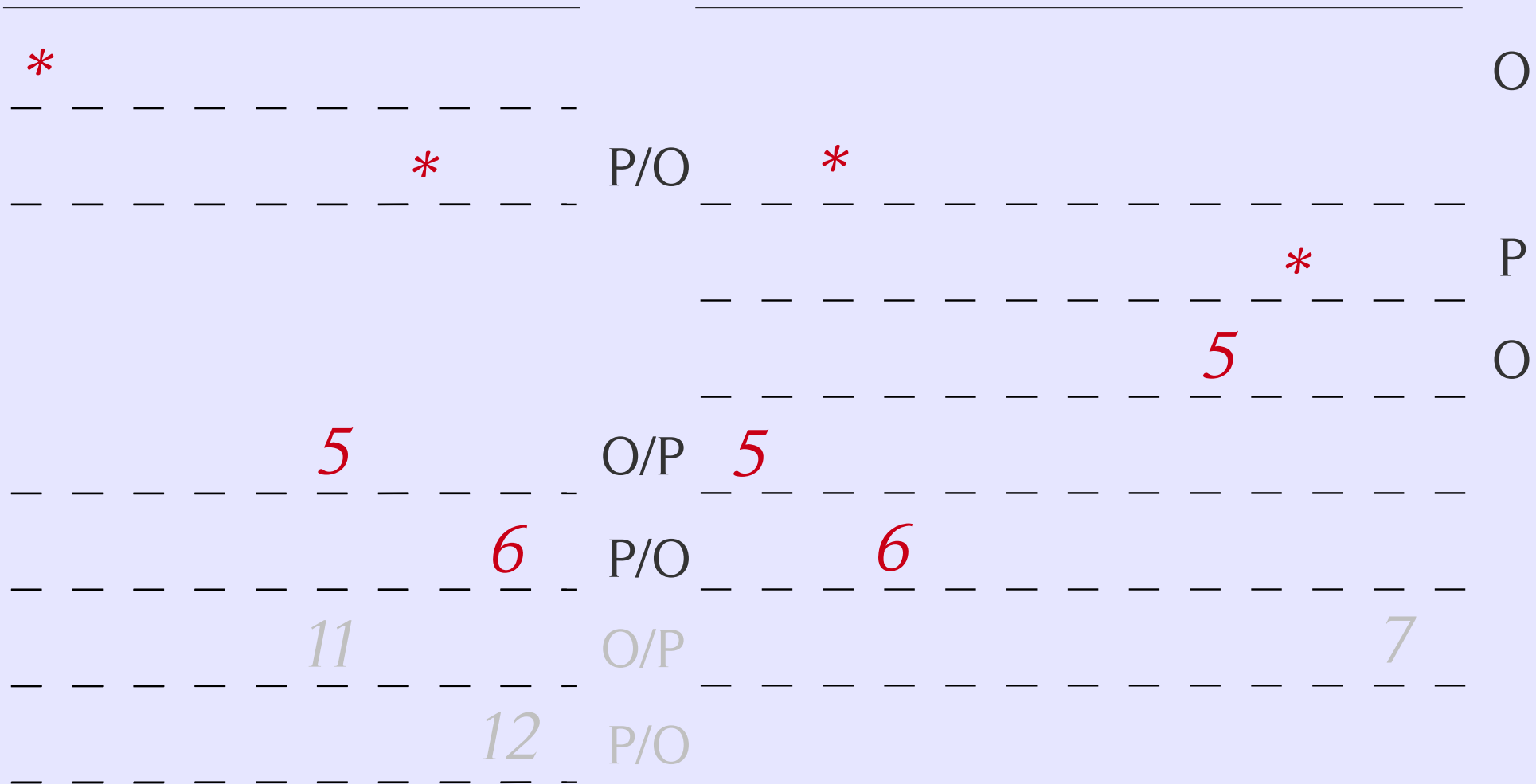
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$







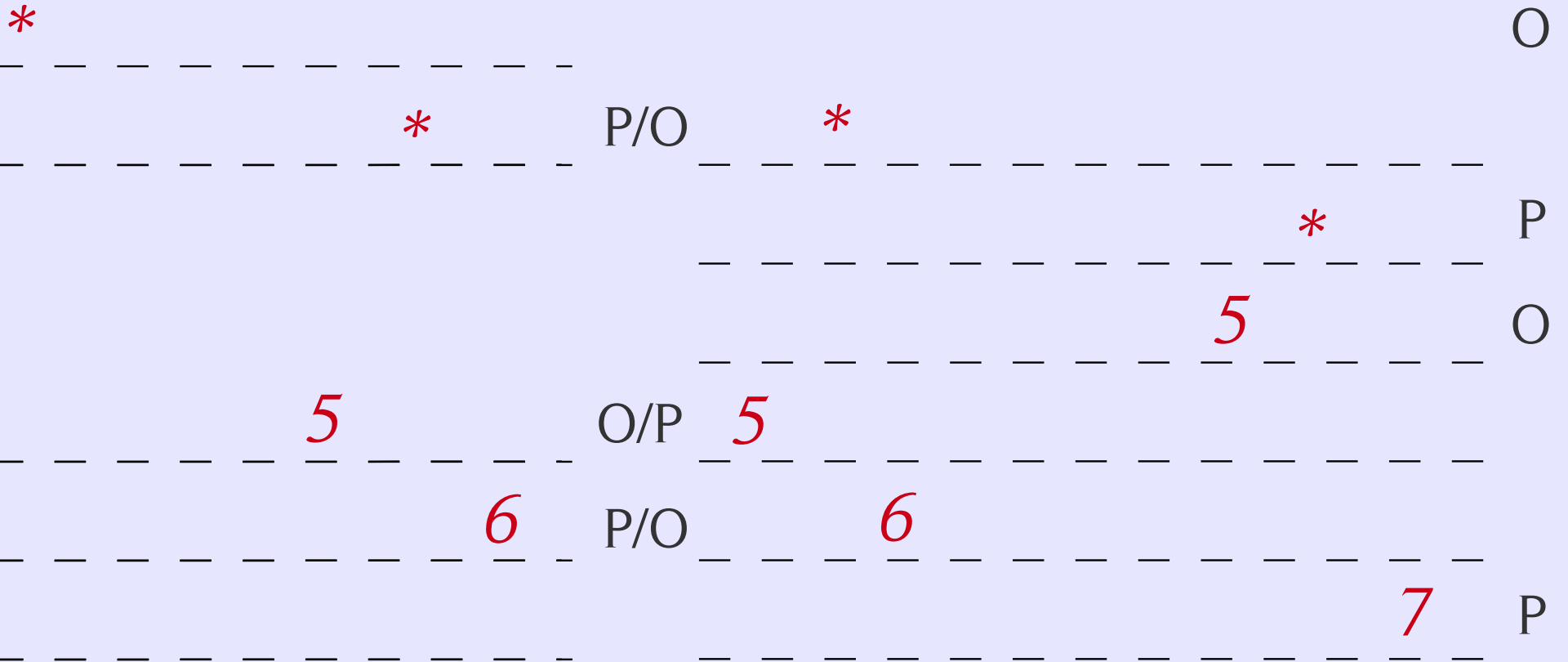
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



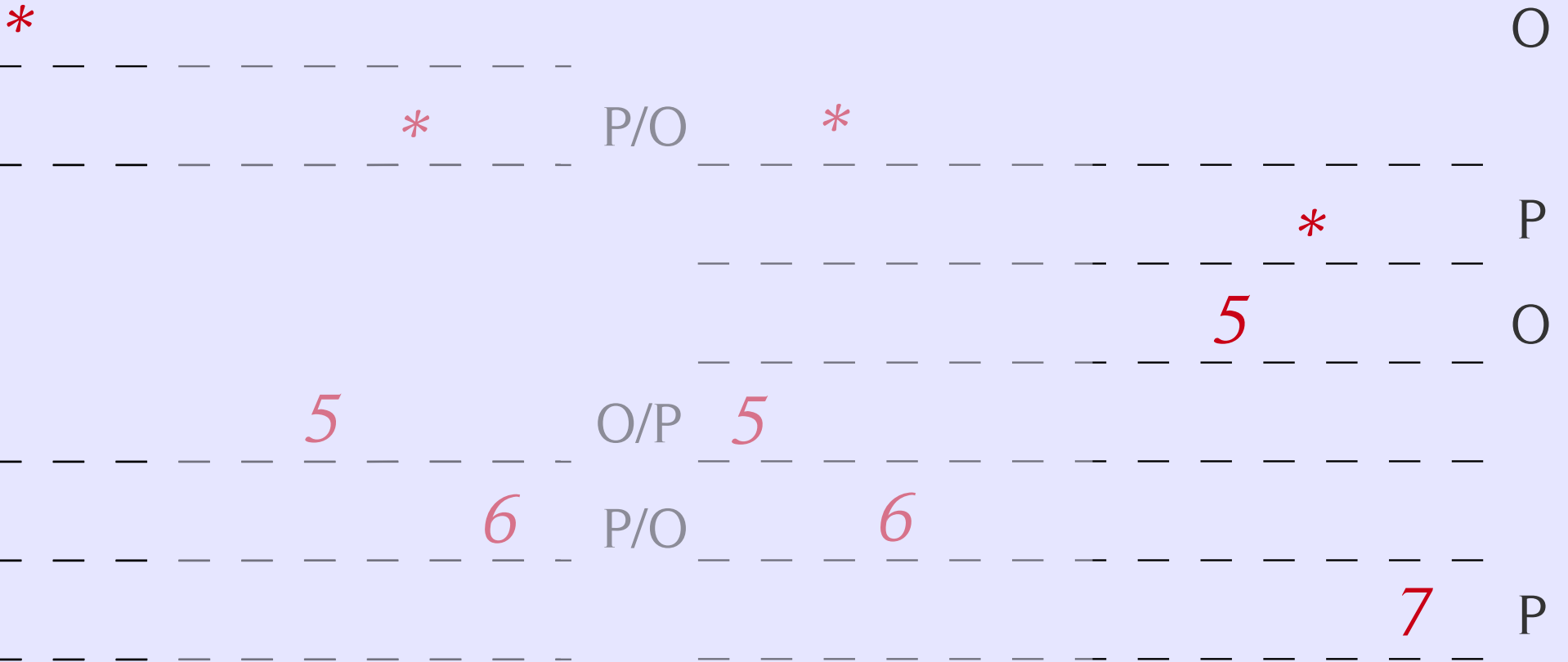
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



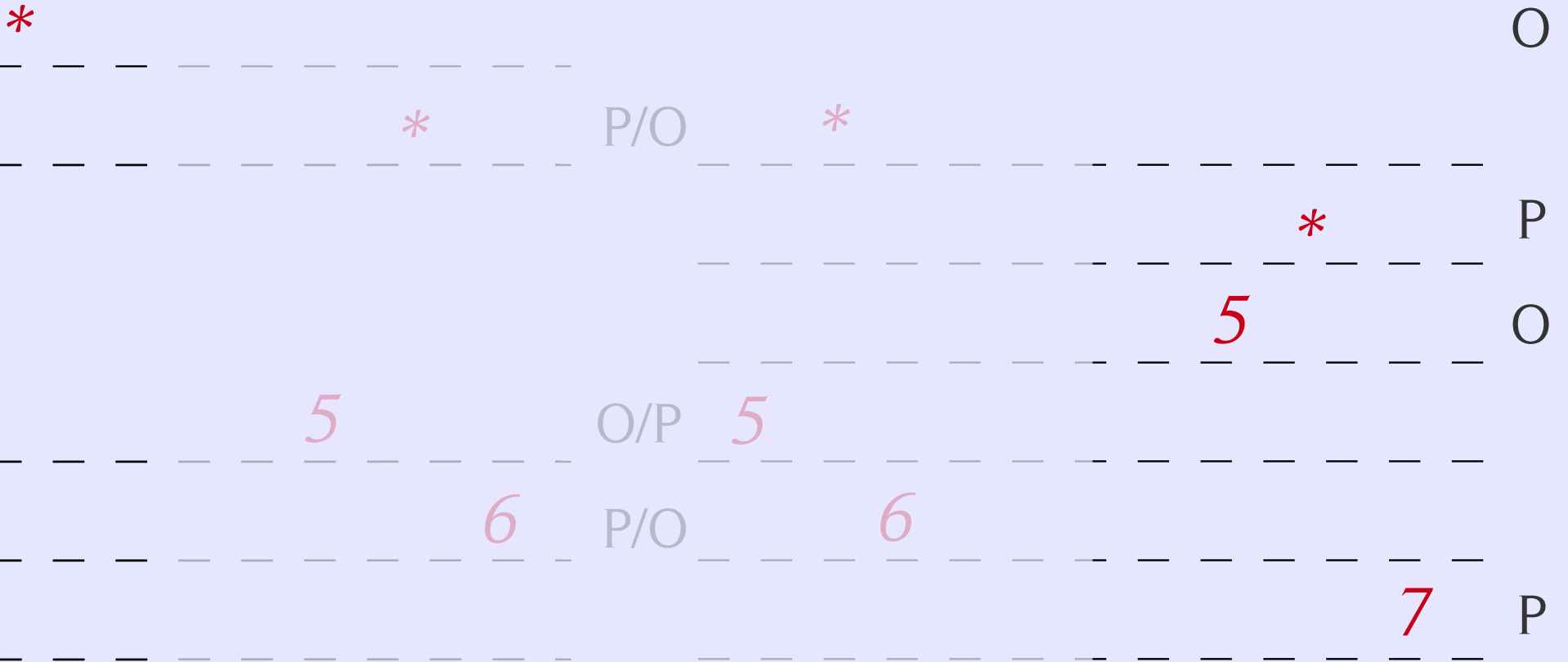
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



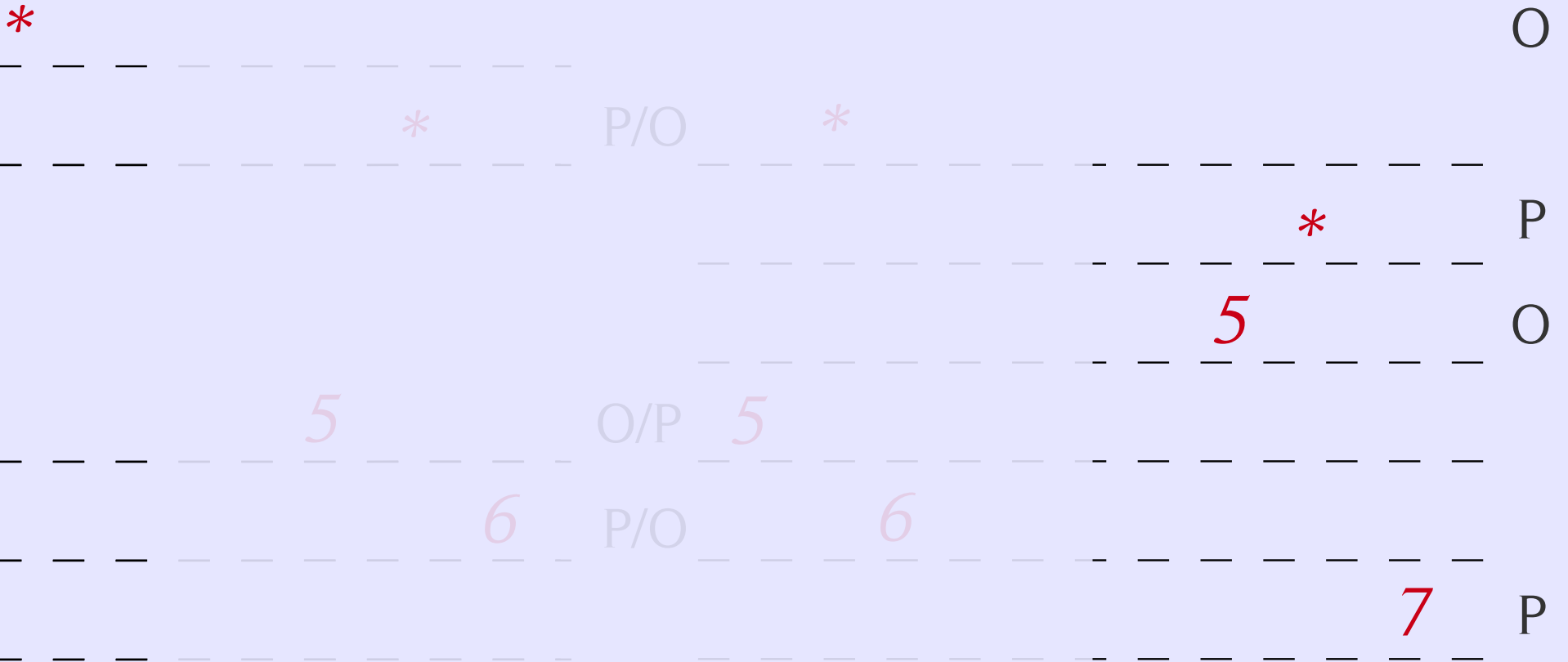
$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

$1 \longrightarrow \text{Int} \rightarrow \text{Int}$

$\text{Int} \rightarrow \text{Int} \longrightarrow \text{Int} \rightarrow \text{Int}$



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

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

# Composition

*1*  $\longrightarrow$  *Int*  $\rightarrow$  *Int*

*\** O

*\** P

*5* O

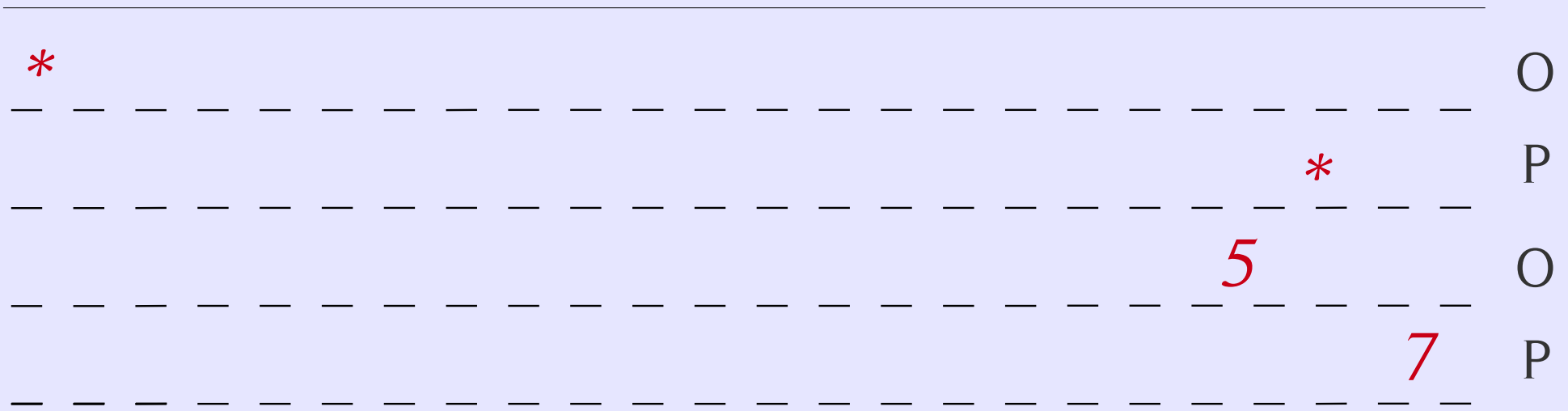
*7* P

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

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

# Composition

*1*  $\longrightarrow$  *Int*  $\rightarrow$  *Int*

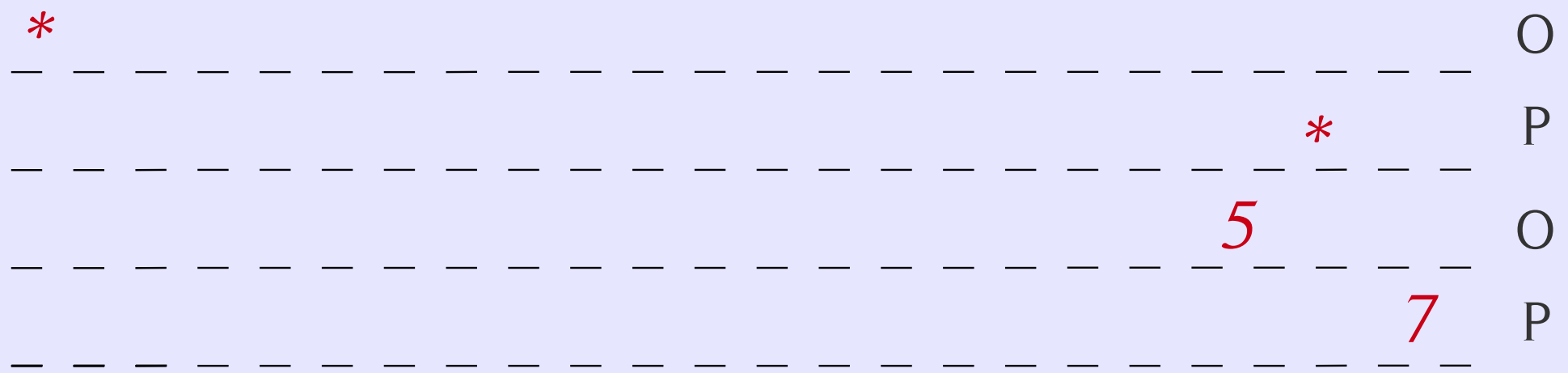


$\vdash \lambda x. x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

1  $\longrightarrow$   $\text{Int} \rightarrow \text{Int}$



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

$=$   $\vdash \lambda x. x+2 : \text{int} \rightarrow \text{int}$

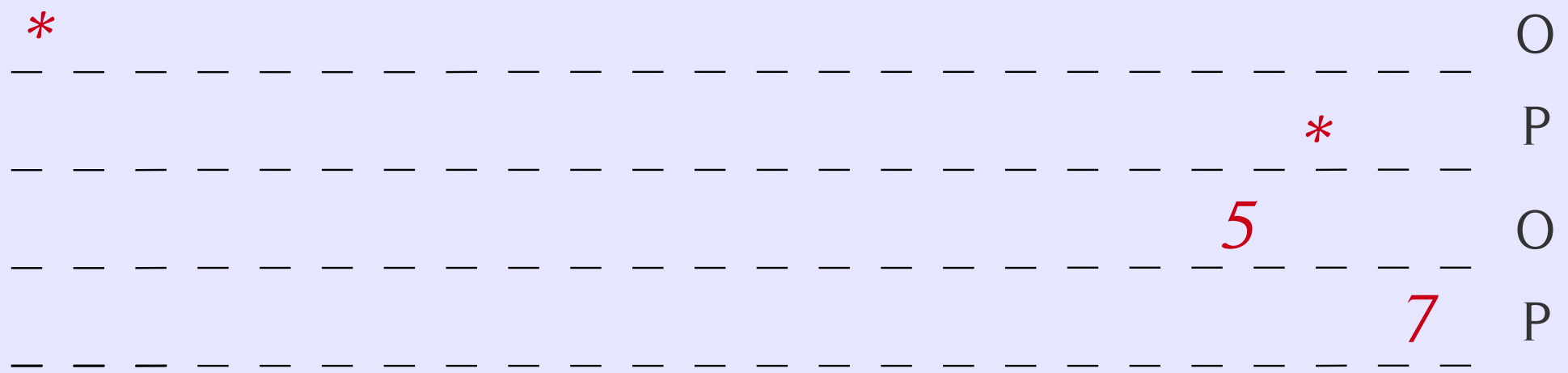


$\vdash \lambda x.x+1 : \text{int} \rightarrow \text{int}$

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

# Composition

1  $\longrightarrow$   $\text{Int} \rightarrow \text{Int}$



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

$= \vdash \lambda x.x+2 : \text{int} \rightarrow \text{int}$

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

# 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  $\bullet : \text{PERM}(\mathcal{N}) \times X \longrightarrow X$   
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 (strong) support**
- NOM: nominal sets and equivariant functions



# Nominal games

# Nominal games

```
 $\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$ 
```

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

# Nominal games

```
 $\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$ 
```

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

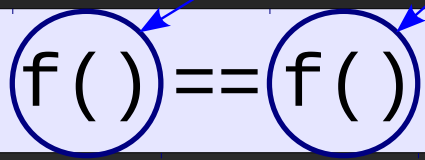
# Nominal games

## Games in nominal sets

names

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

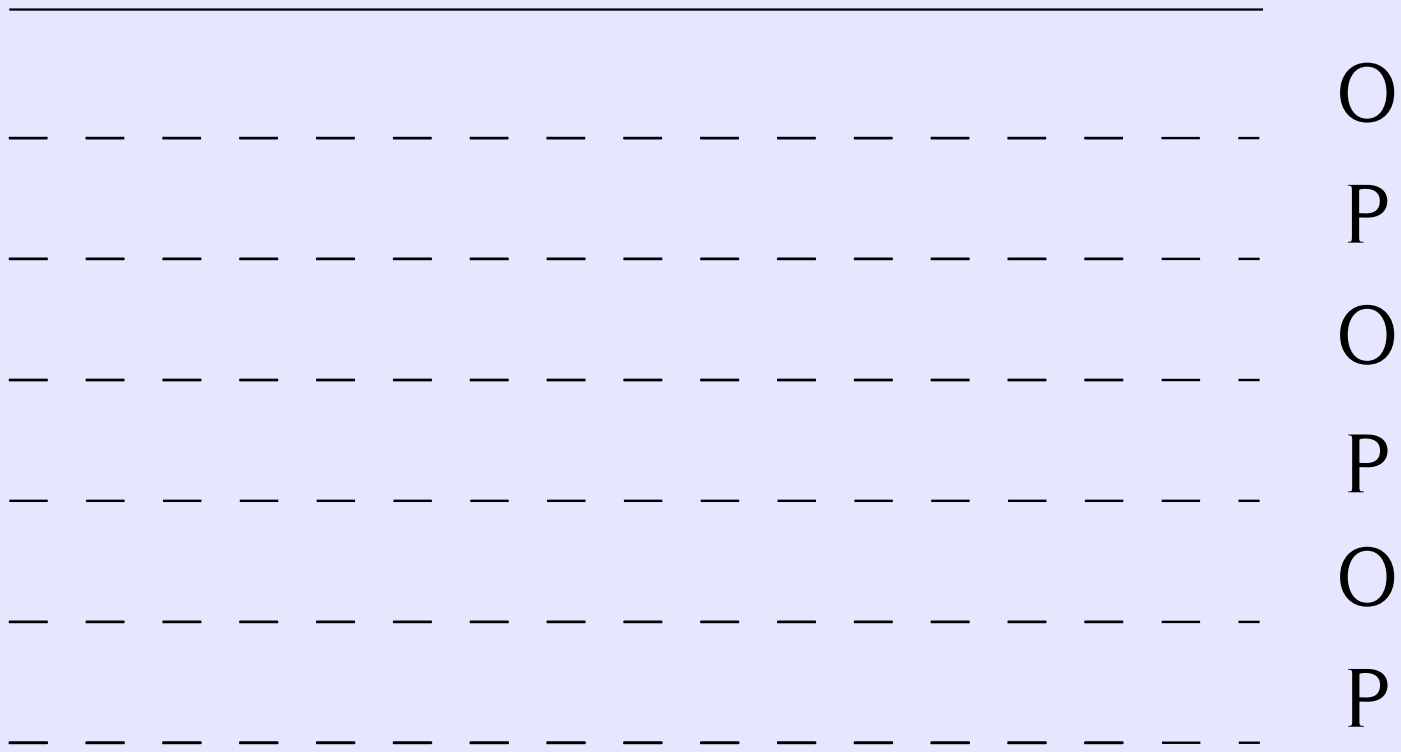
$\text{let } f = [\_ ] \text{ in } \{ \textcircled{f()} == \textcircled{f()} \}$

The diagram consists of a blue box labeled 'names' at the top right. Two blue arrows originate from the bottom corners of this box. One arrow points to the left 'f()' term in the expression below, and the other points to the right 'f()' term. Both 'f()' terms are enclosed in blue circles.

# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

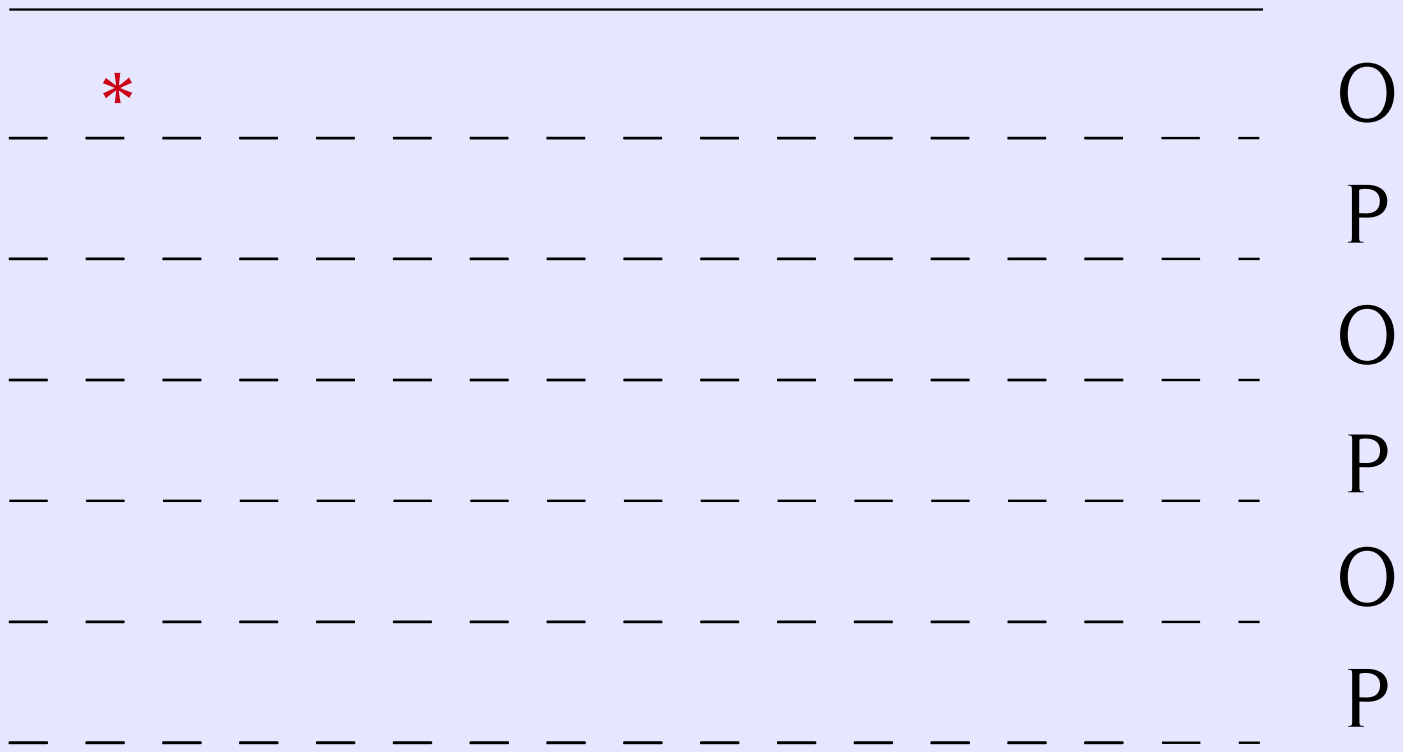
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

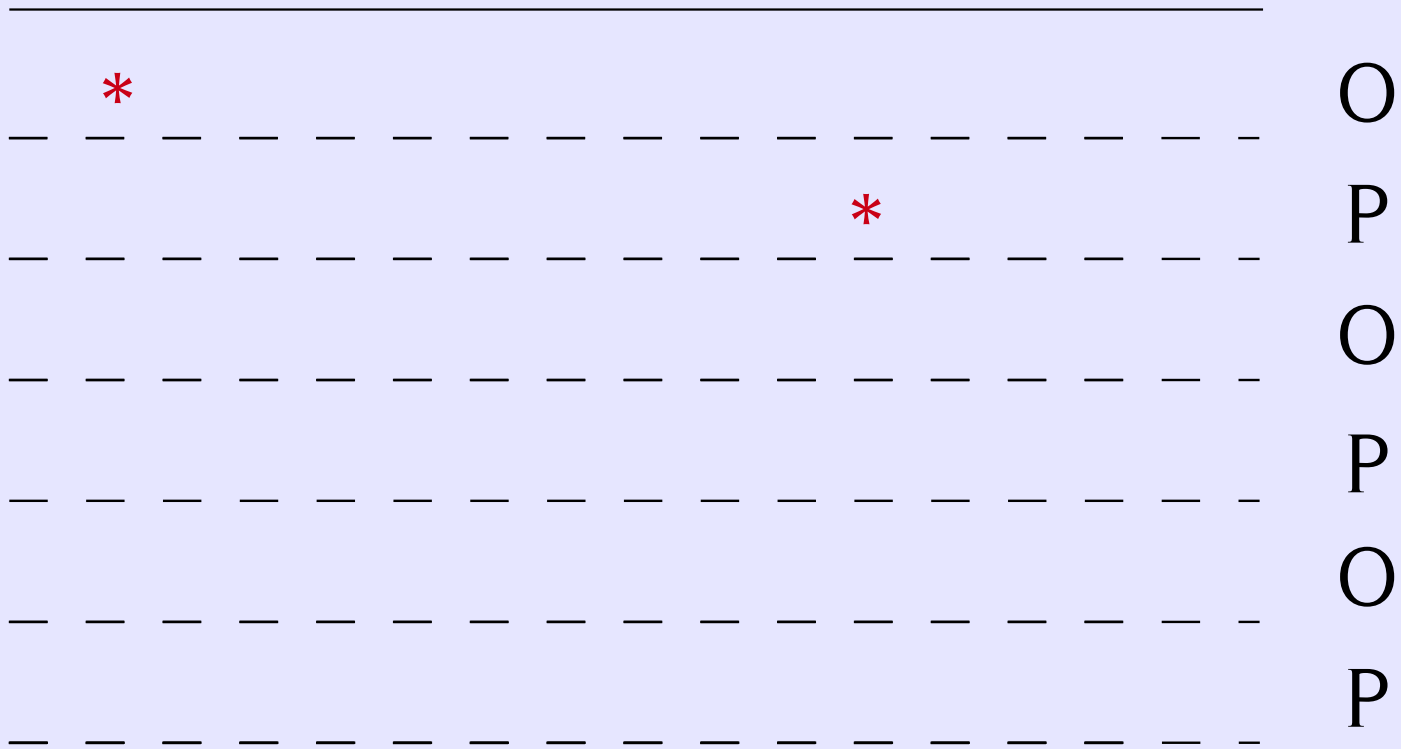
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

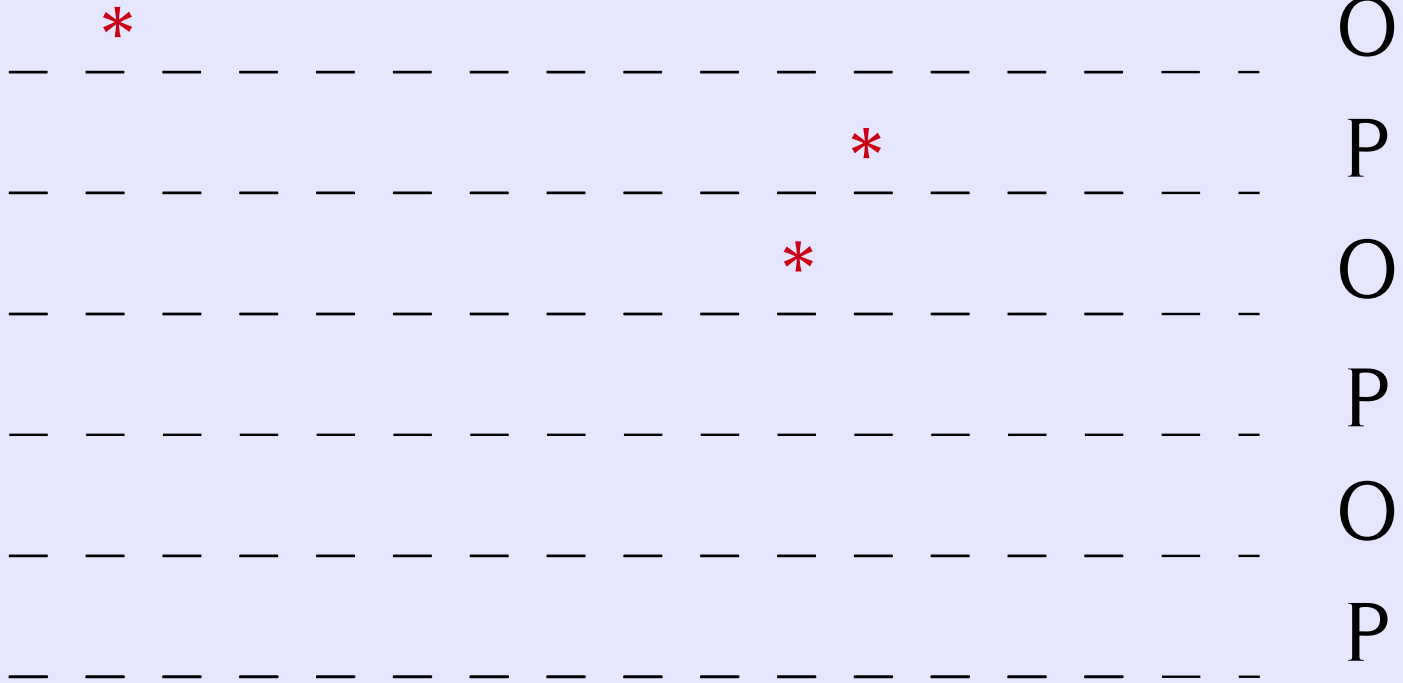
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

$\lambda x.\text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

$1 \longrightarrow 1 \rightarrow \text{Ref}$

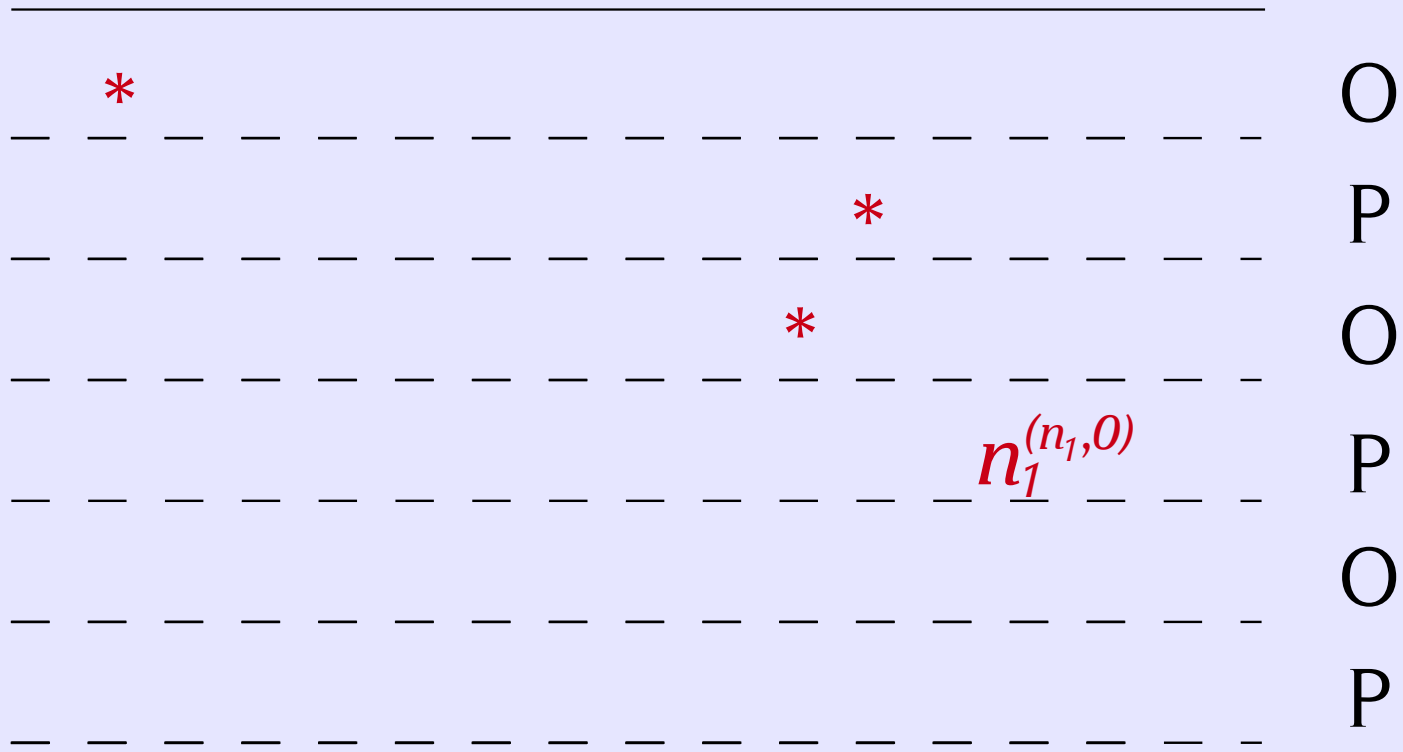




# Examples

$\lambda x.\text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

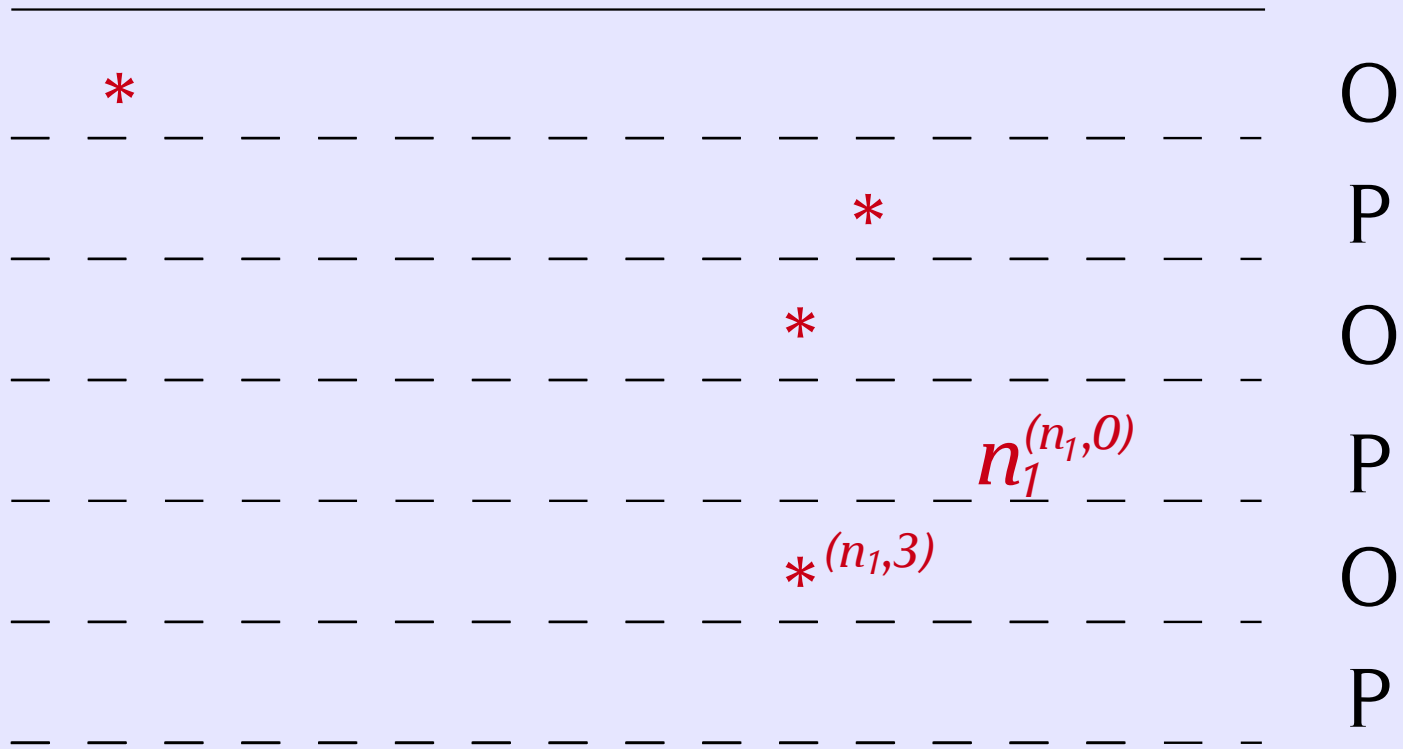
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

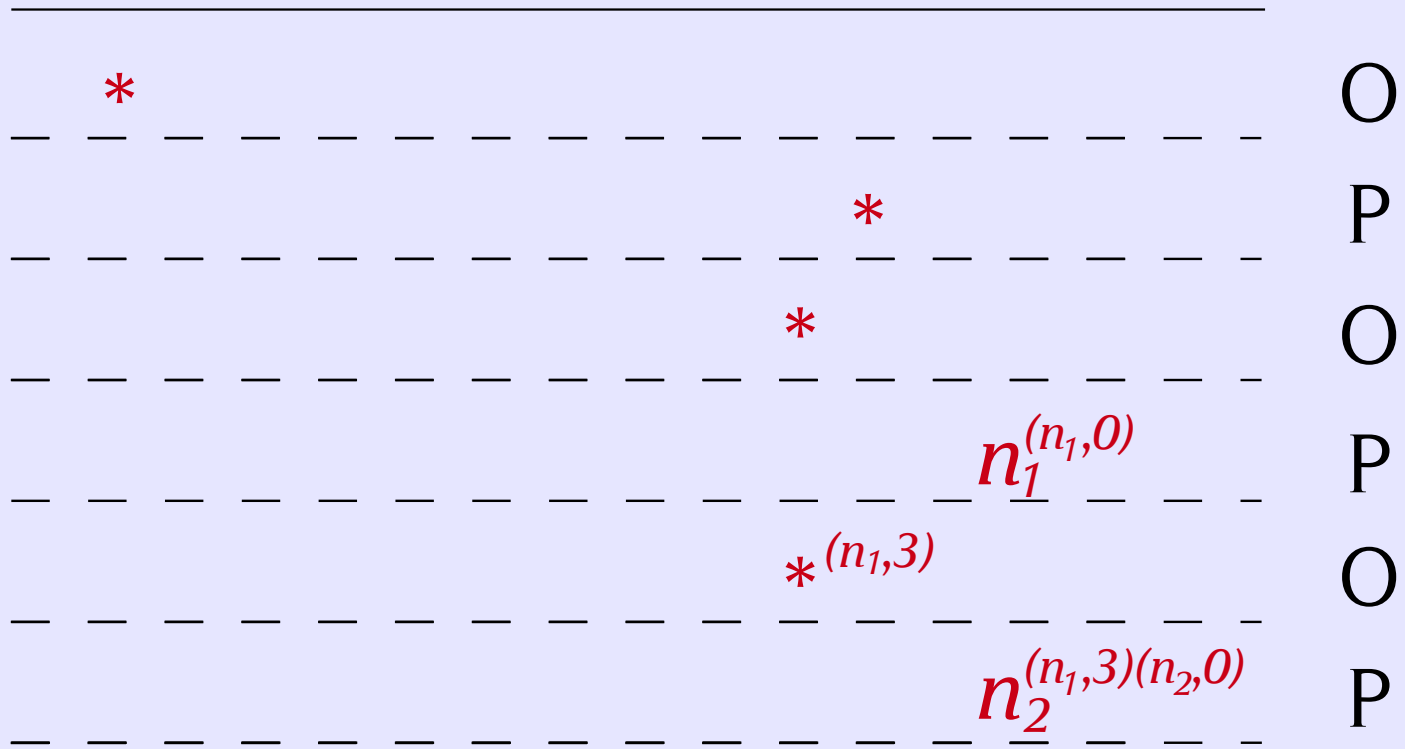
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

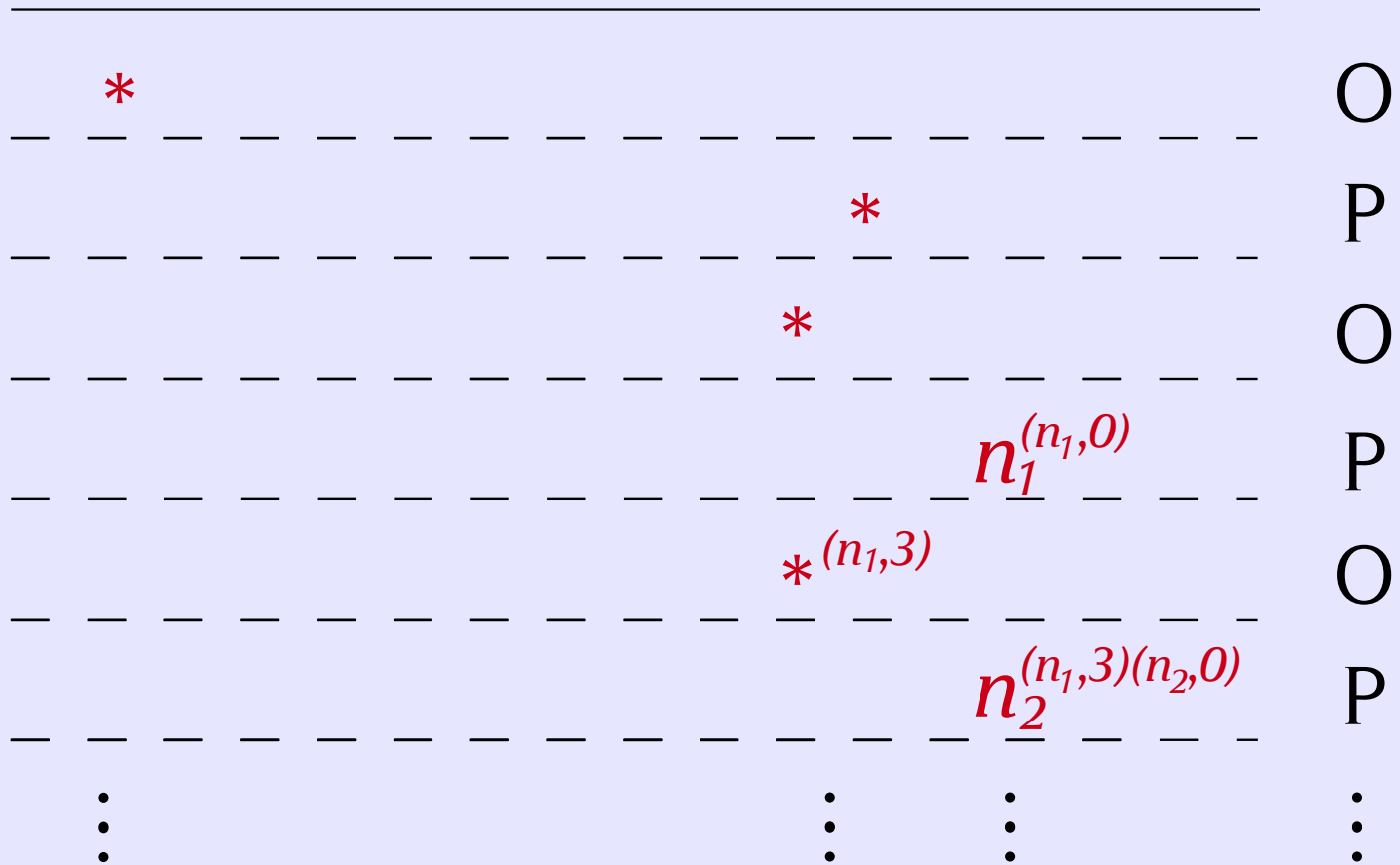
$1 \longrightarrow 1 \rightarrow \text{Ref}$



# Examples

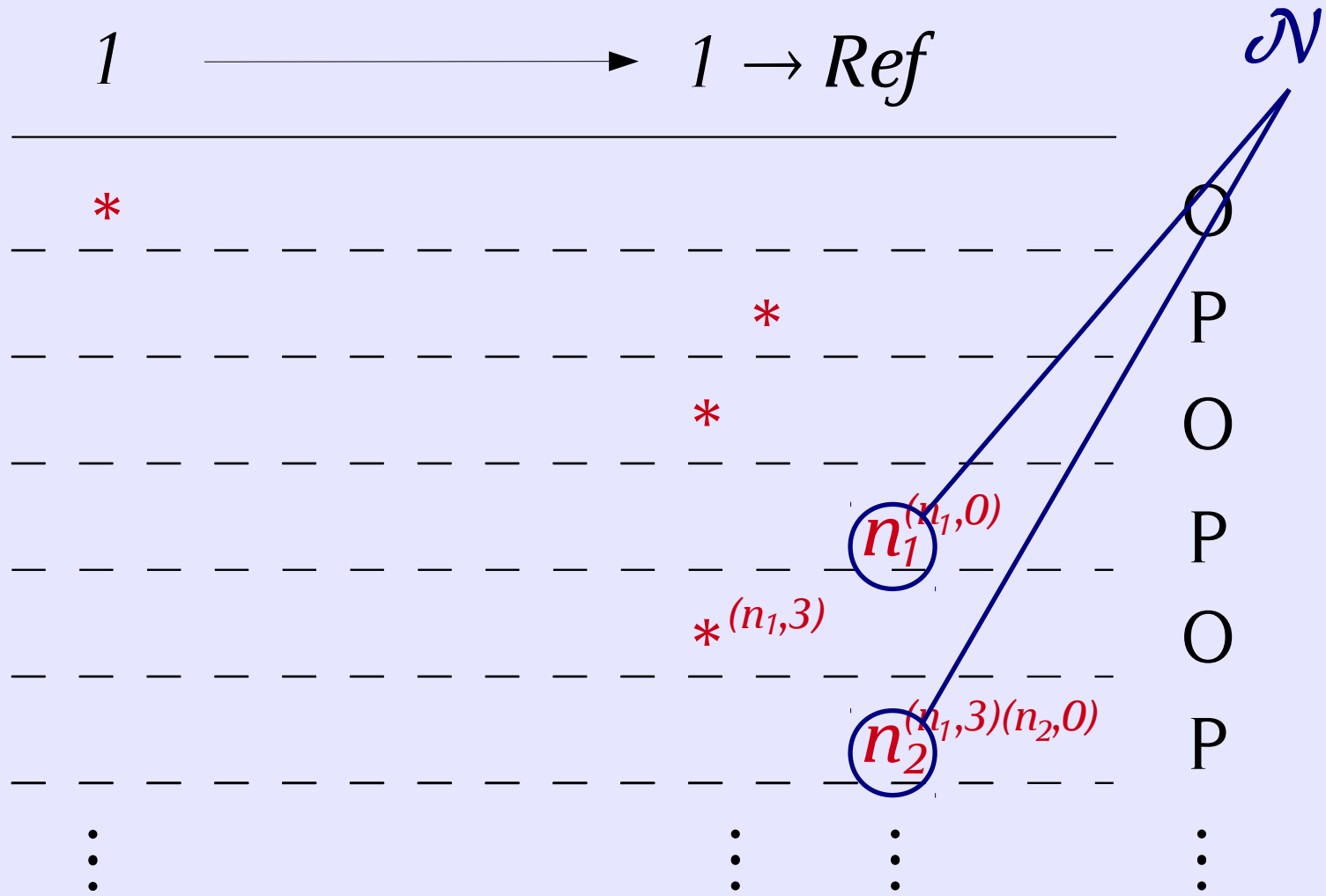
$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$

$1 \longrightarrow 1 \rightarrow \text{Ref}$



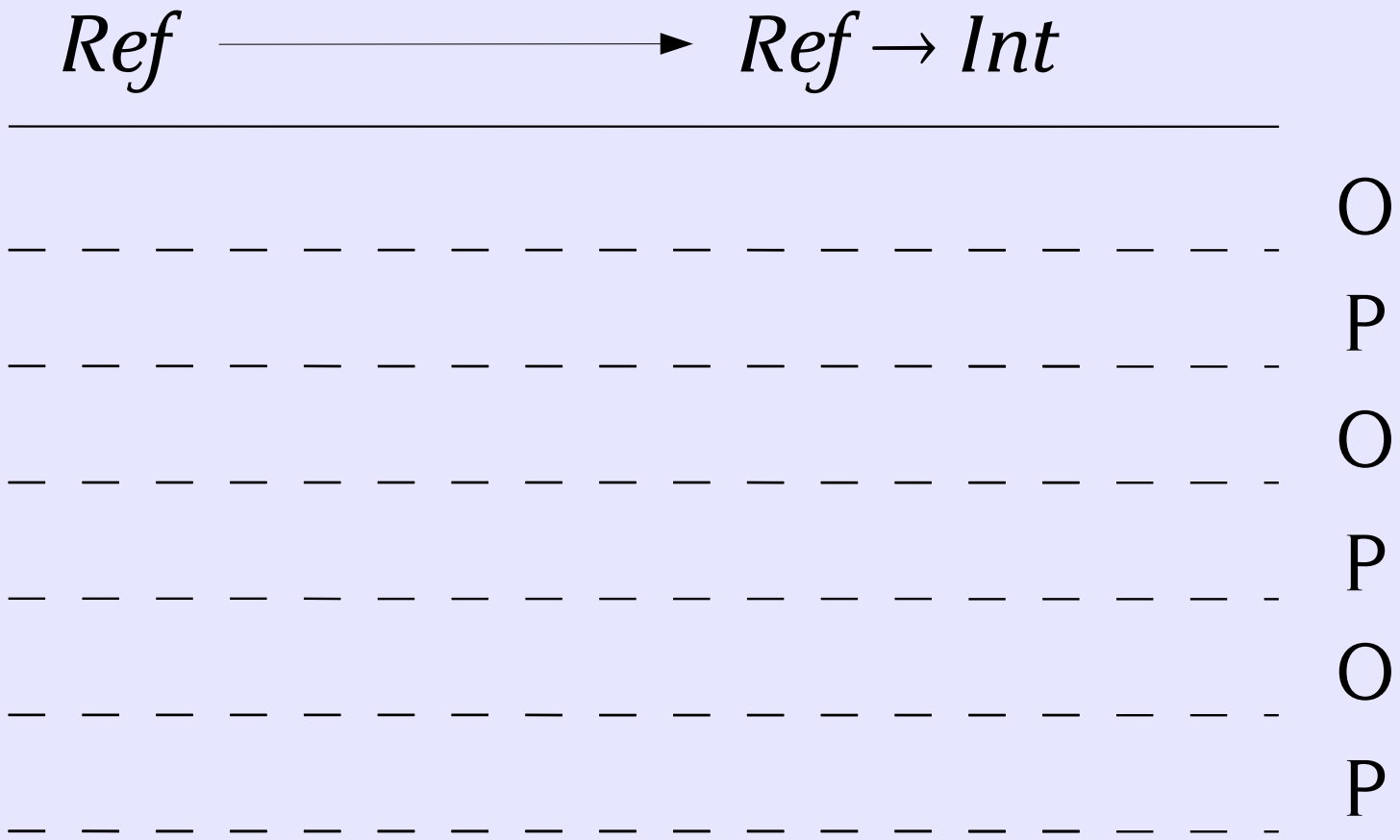
# Examples

$\lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$



# Examples

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

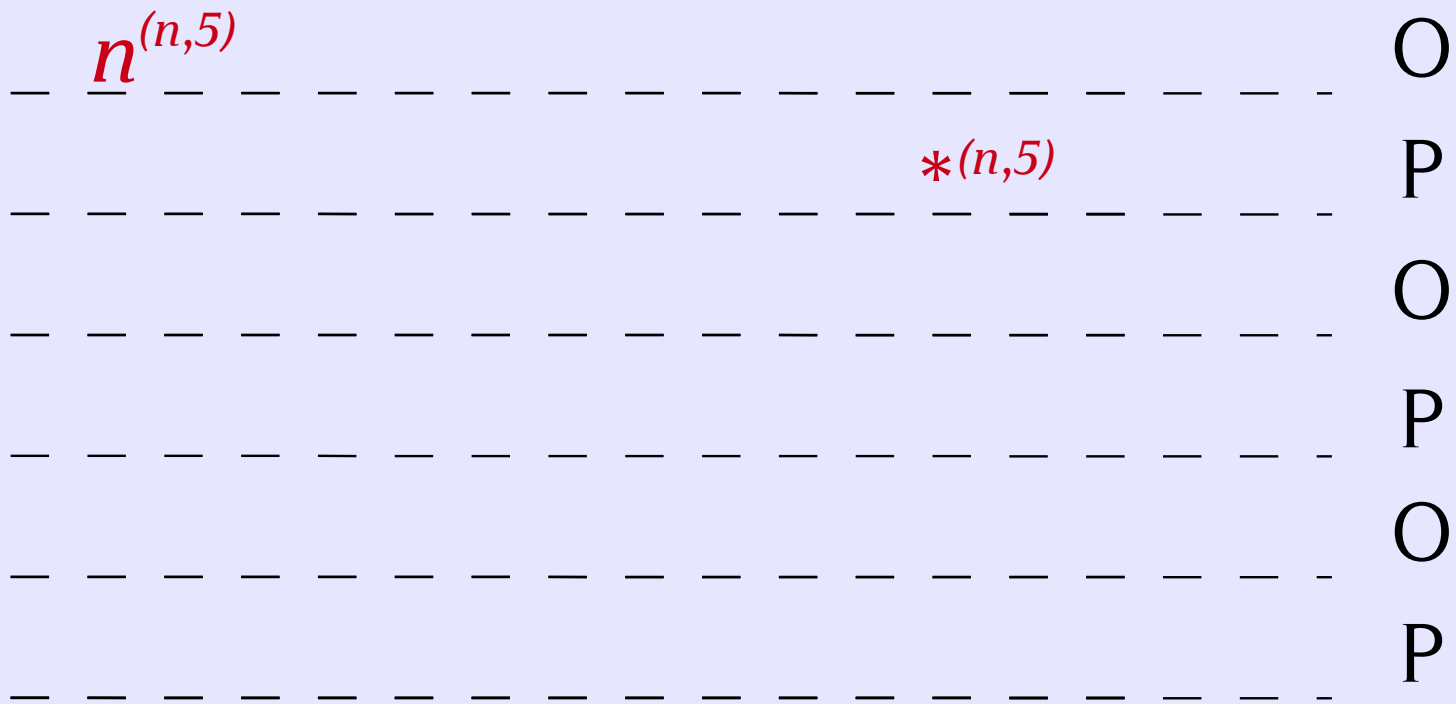




# Examples

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

$Ref \longrightarrow Ref \rightarrow Int$





# Examples

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

$Ref \longrightarrow Ref \rightarrow Int$



# Examples

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

$Ref \longrightarrow Ref \rightarrow Int$

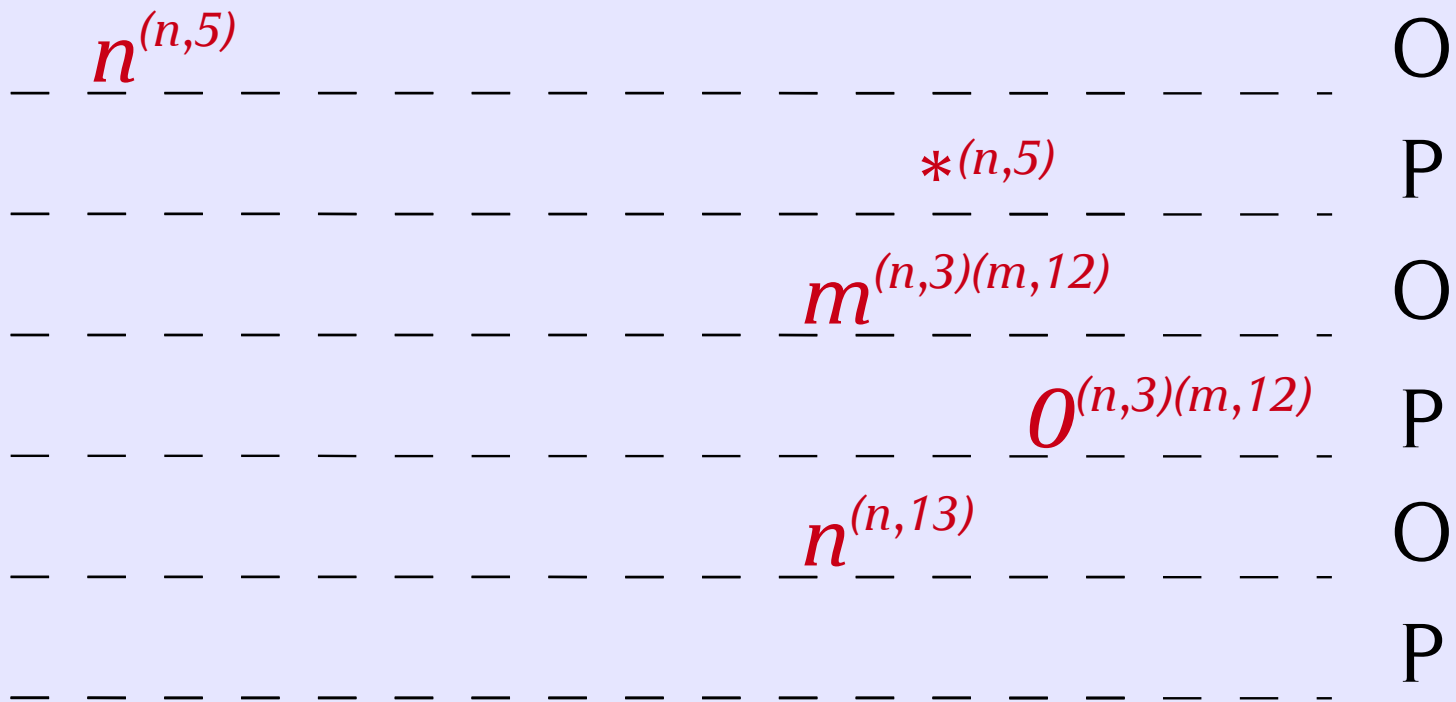
---

$n^{(n,5)}$		O
		P
	$*^{(n,5)}$	O
		P
	$m^{(n,3)(m,12)}$	O
		P
	$o^{(n,3)(m,12)}$	O
		P

# Examples

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

$Ref \longrightarrow Ref \rightarrow Int$



# Examples

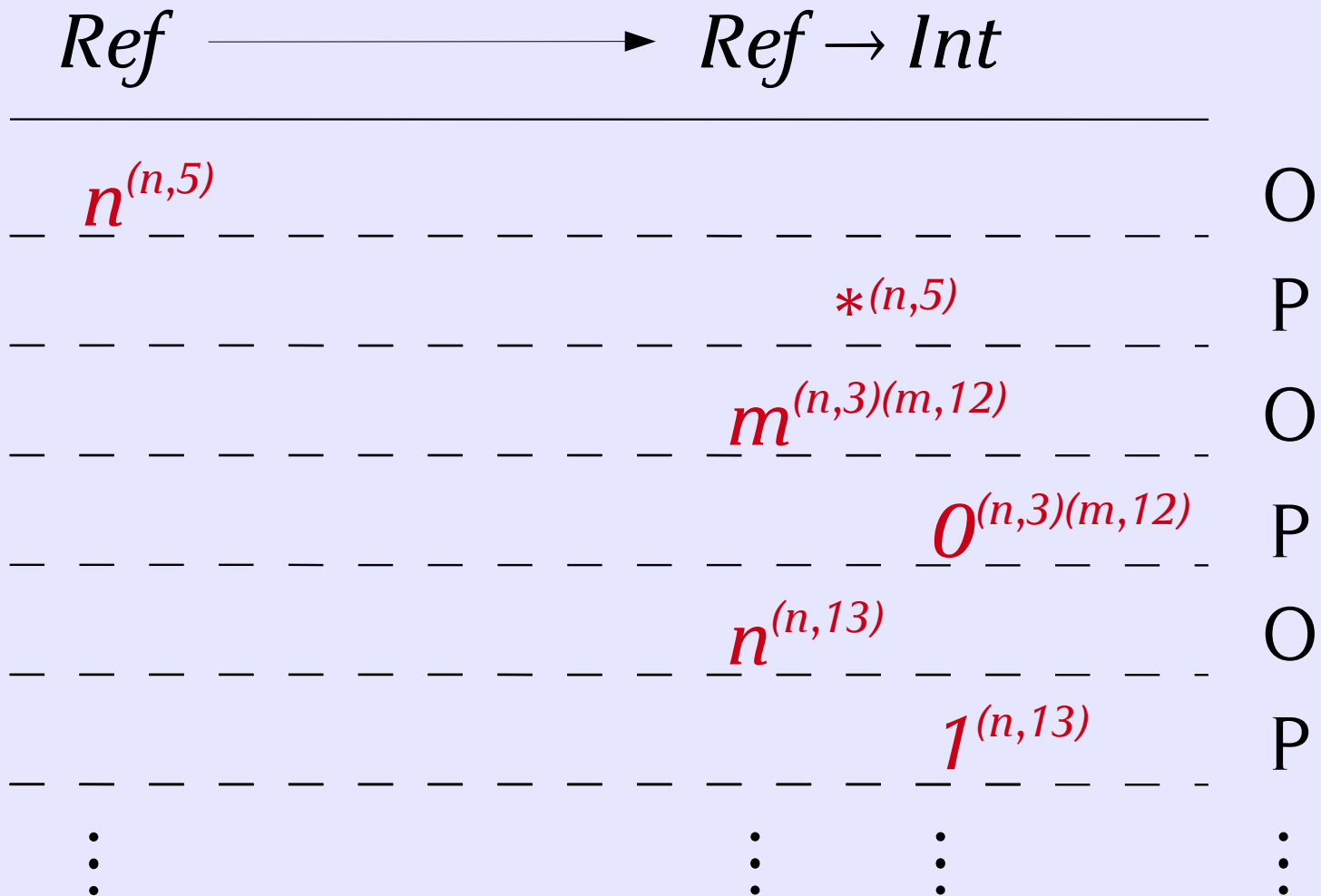
$x : \text{intref} \vdash \lambda y. (x == y) : \text{intref} \rightarrow \text{int}$

$Ref \longrightarrow Ref \rightarrow Int$

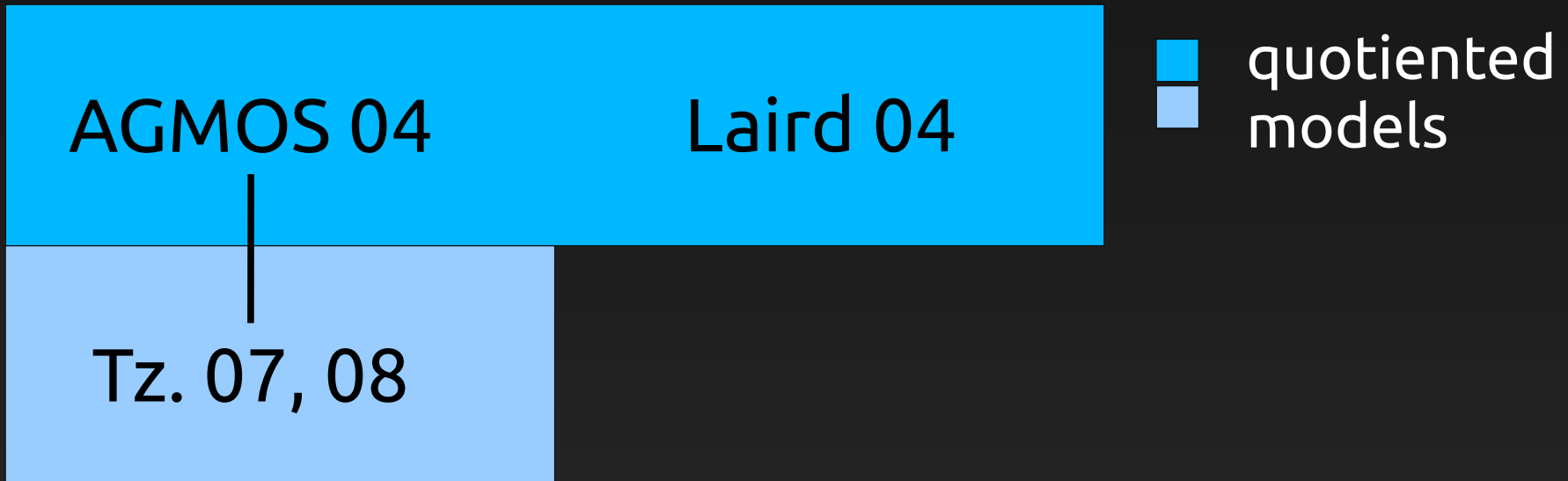
$n^{(n,5)}$		O
	$*^{(n,5)}$	P
	$m^{(n,3)(m,12)}$	O
	$o^{(n,3)(m,12)}$	P
	$n^{(n,13)}$	O
	$\uparrow^{(n,13)}$	P

# Examples

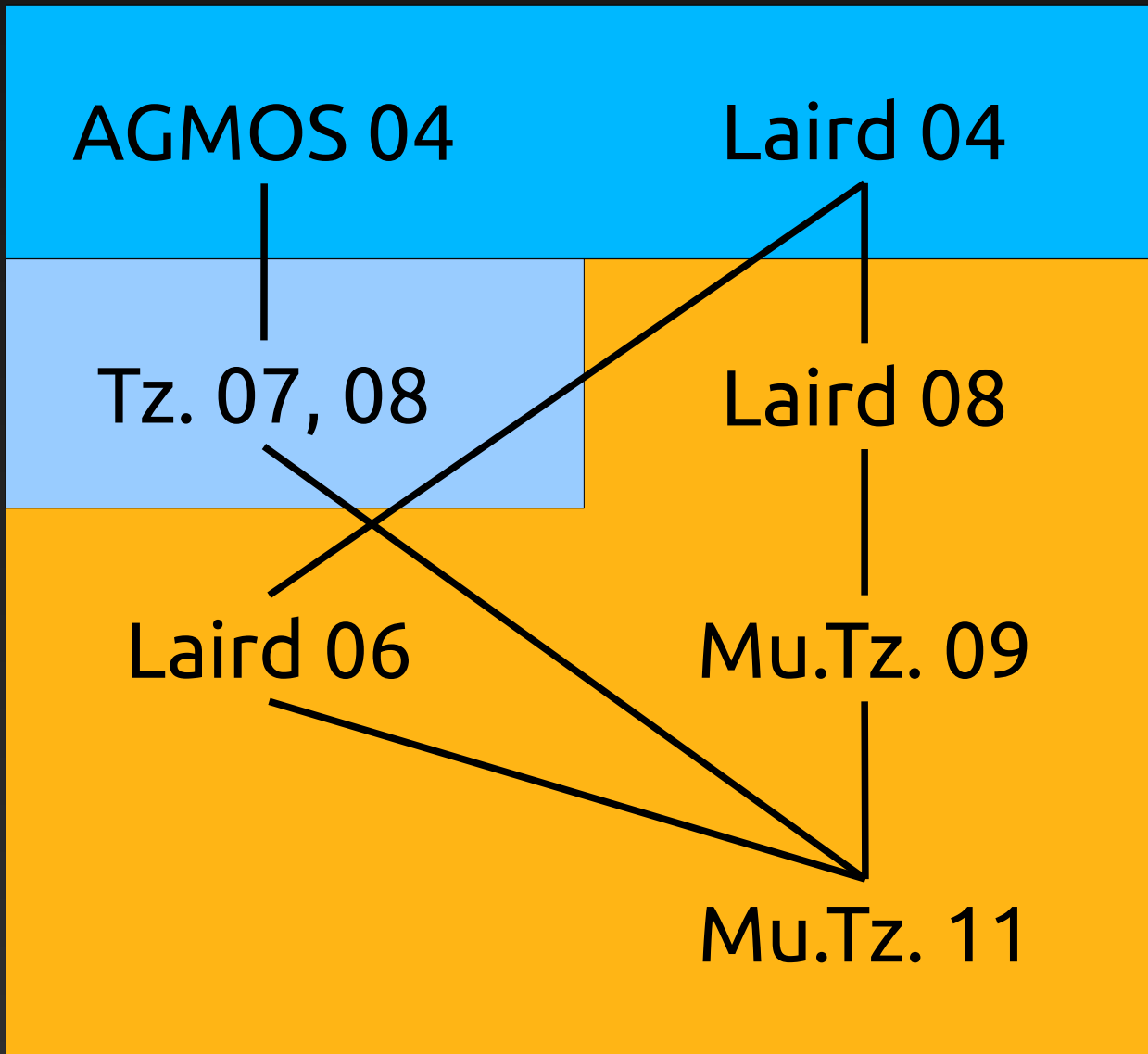
$x : \text{intref} \vdash \lambda y. (x == y) : \text{intref} \rightarrow \text{int}$



# Achievements

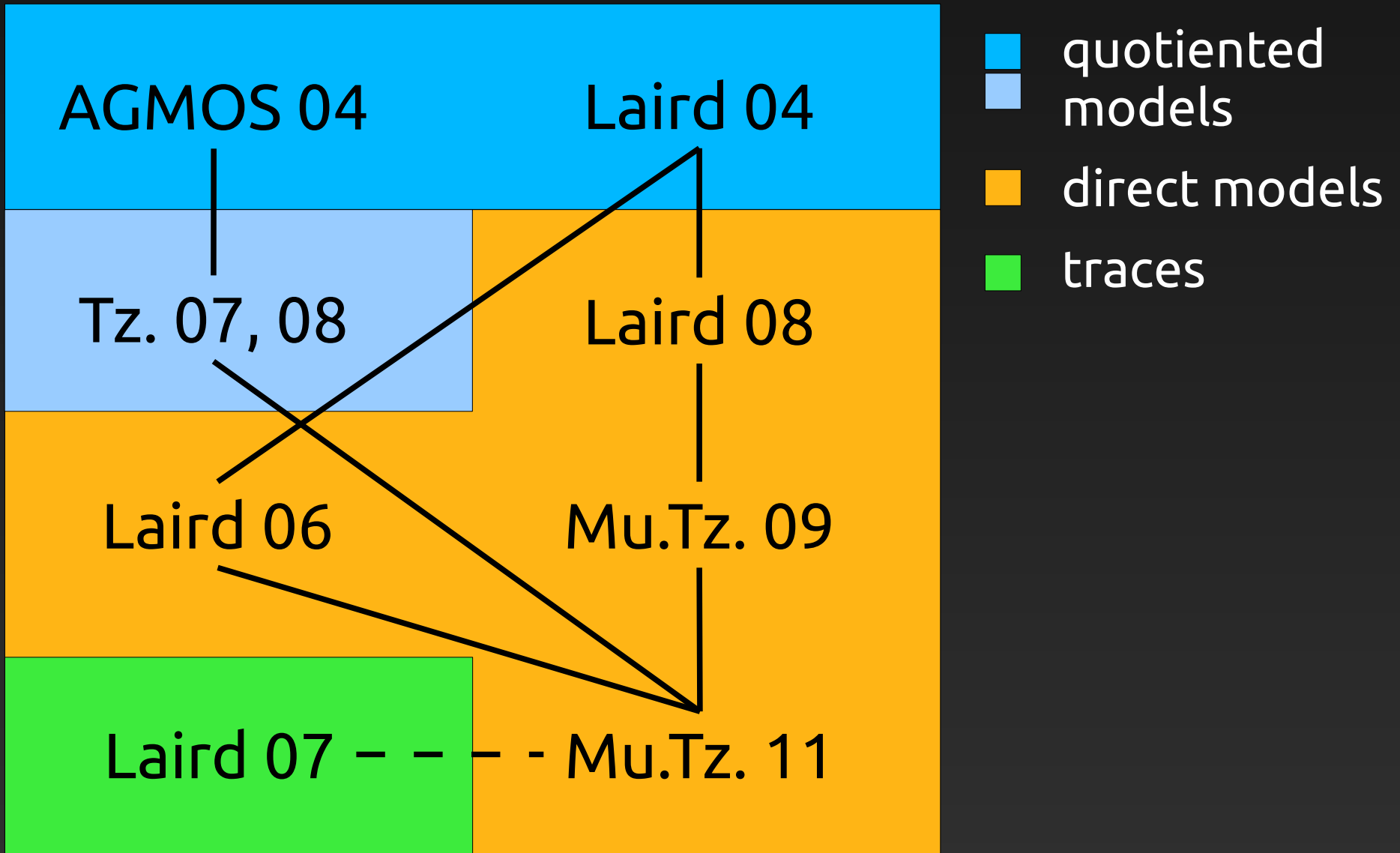


# Achievements



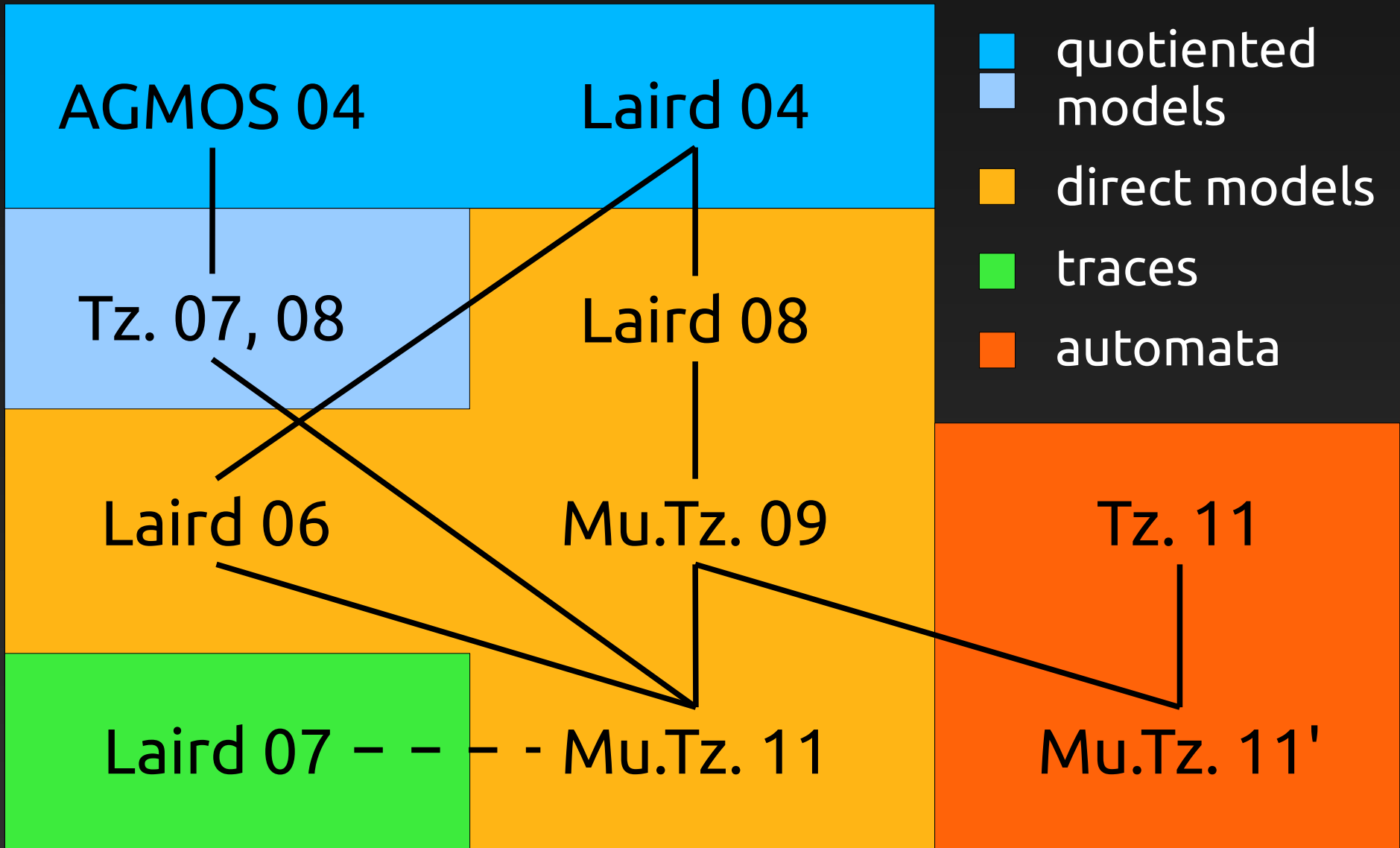
- quotiented models
- models
- direct models

# Achievements





# Achievements



# Quotiented models

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

$$M \cong N \iff \llbracket M \rrbracket \cong \llbracket N \rrbracket$$

# Quotiented models

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

$$M \cong N \iff \llbracket M \rrbracket \cong \llbracket N \rrbracket$$

$$\llbracket A \rightarrow B \rrbracket = \llbracket A \rrbracket \Rightarrow (S \Rightarrow \llbracket B \rrbracket \otimes S)$$

$$S = \bigotimes_A (N_A \Rightarrow \llbracket A \rrbracket)$$

# Quotiented models

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

$$M \cong N \iff \llbracket M \rrbracket \cong \llbracket N \rrbracket$$

## 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

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

# Direct models

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

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

## Characteristics:

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

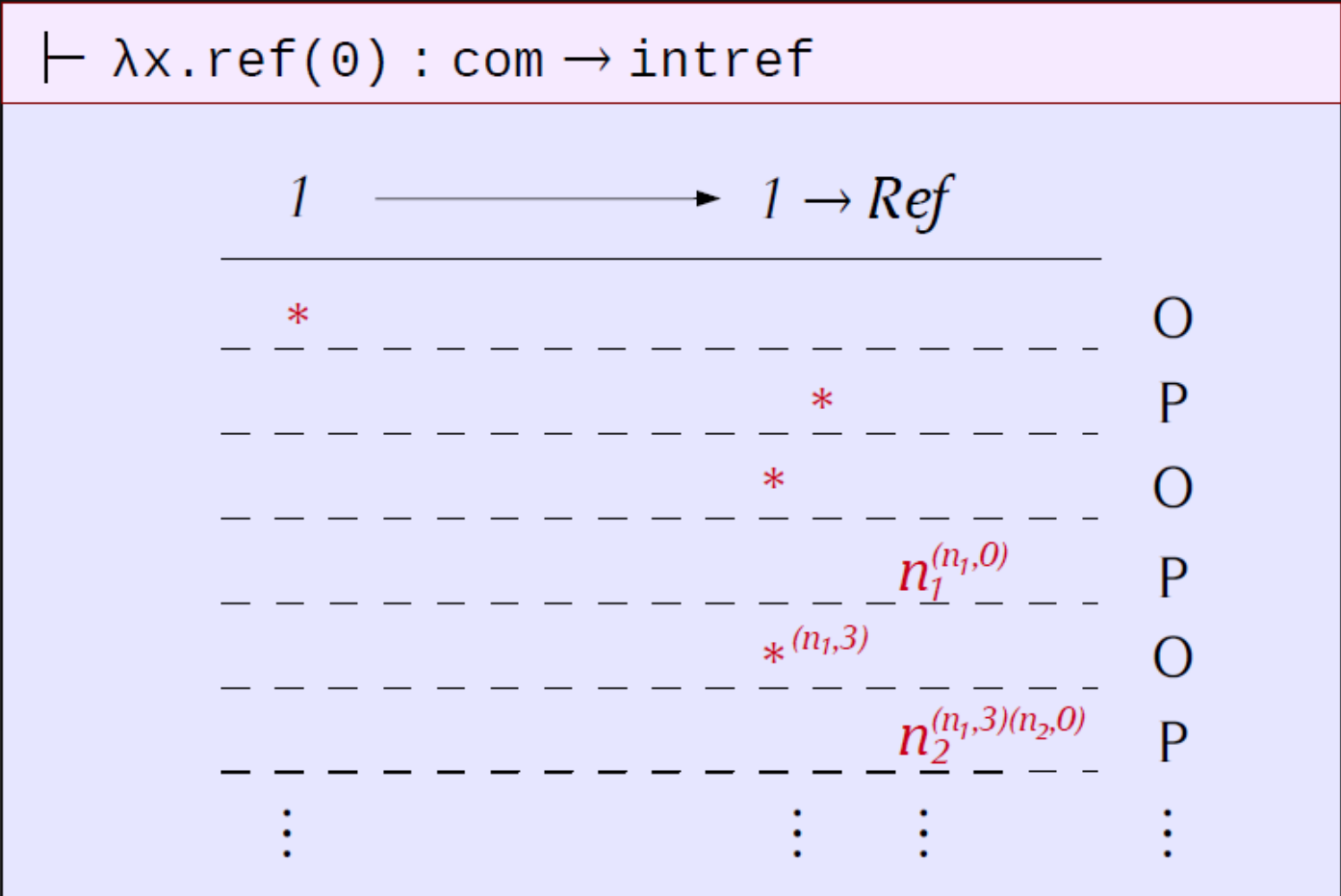
# Direct models

- Laird 06: high-level
- Laird 08: pointer
- Mu.Tz. 09: invariants
- Mu.Tz. 11:  $\vdash$

$$M \cong N$$

## Characteristics

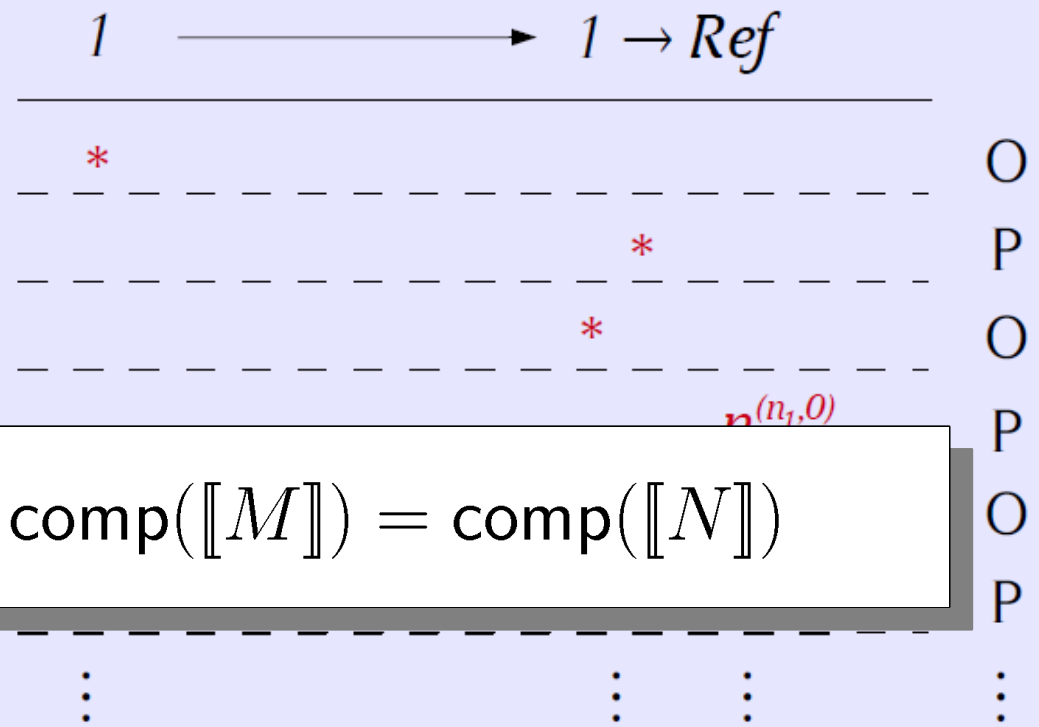
- moves involving names/ moves-with-store
- name-availability conditions/ “direct” effects



# Direct models

- Laird 06: high
- Laird 08: po
- Mu.Tz. 09: in
- Mu.Tz. 11:  $\vdash$

$\vdash \lambda x. \text{ref}(\theta) : \text{com} \rightarrow \text{intref}$



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

## Characteristics

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

Algorithmics



# Fresh-register automata

$$\lambda z. \text{ref}(\Theta) \mapsto \{ * * * n_1 * n_2 * n_3 \dots \mid n_i \text{'s distinct} \}$$

# Fresh-register automata

$$\lambda z. \text{ref}(\emptyset) \mapsto \{ * * * n_1 * n_2 * n_3 \dots \mid n_i \text{'s distinct} \}$$

## Automata with names

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

# Fresh-register automata

$$\lambda z. \text{ref}(\emptyset) \mapsto \{ * * * n_1 * n_2 * n_3 \dots \mid n_i \text{'s distinct} \}$$

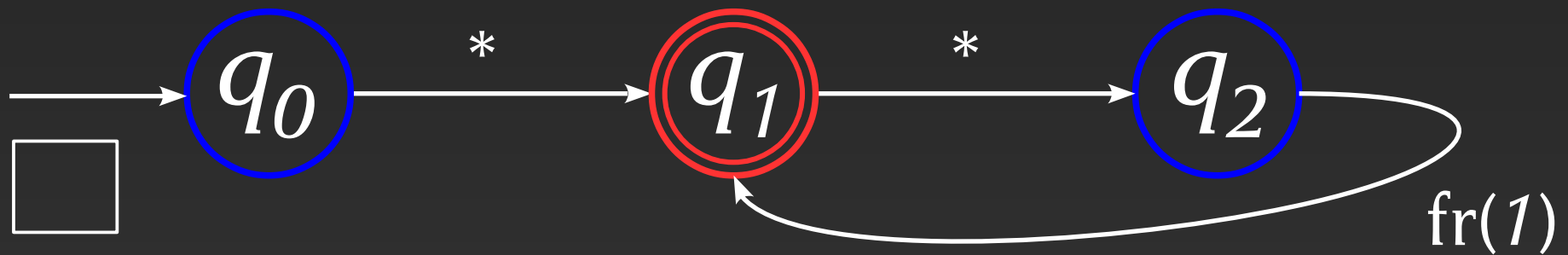
## Automata with names

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

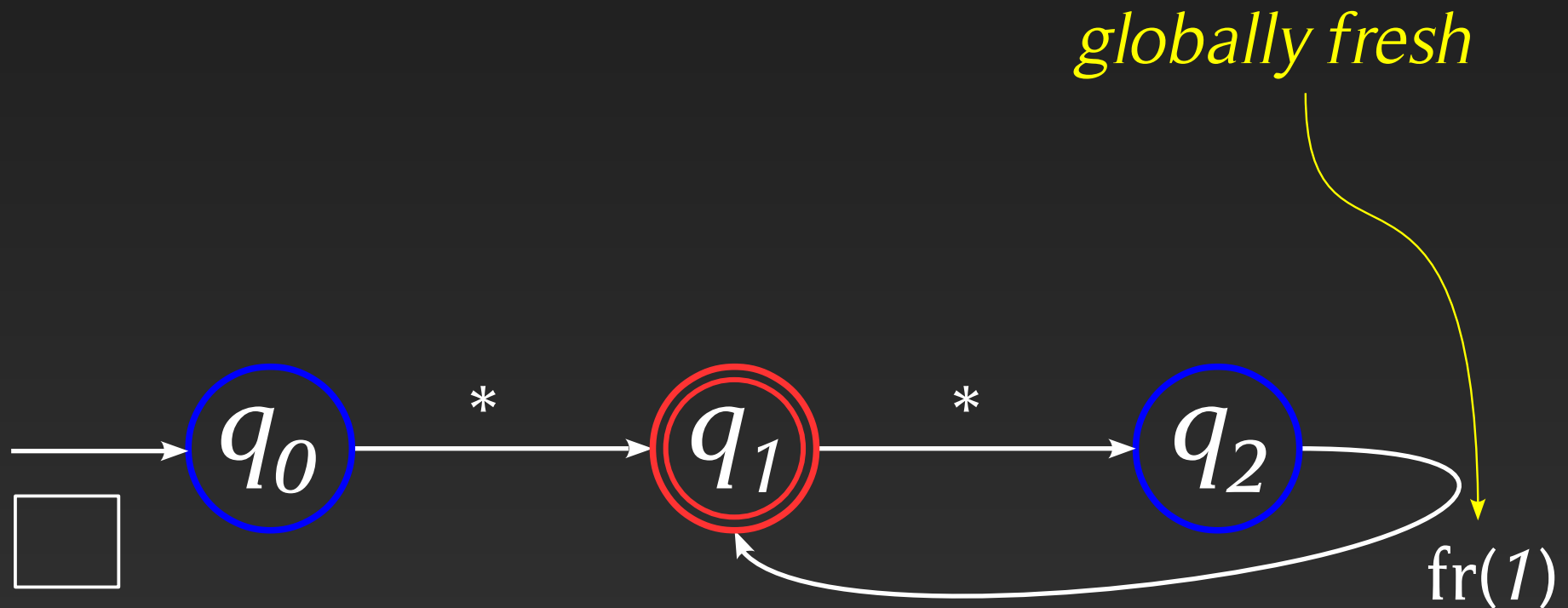
## Finite-state machines with registers

- *Kaminski & Francez (1994)*

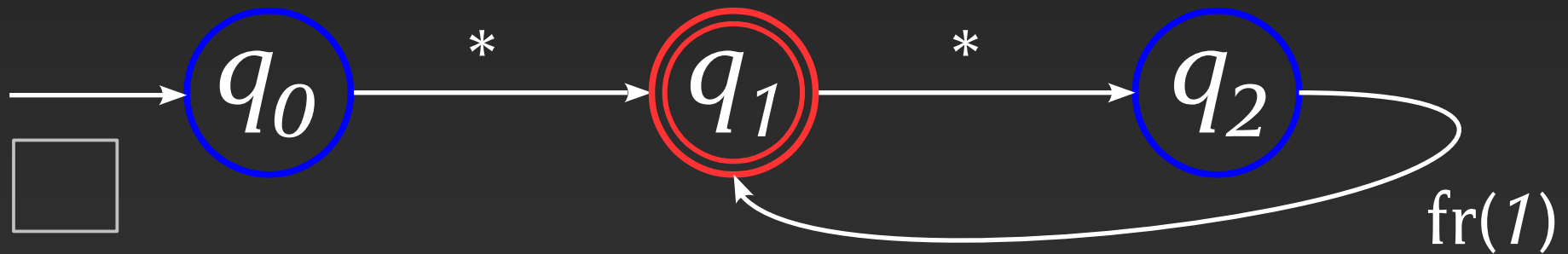
# Fresh-register automata



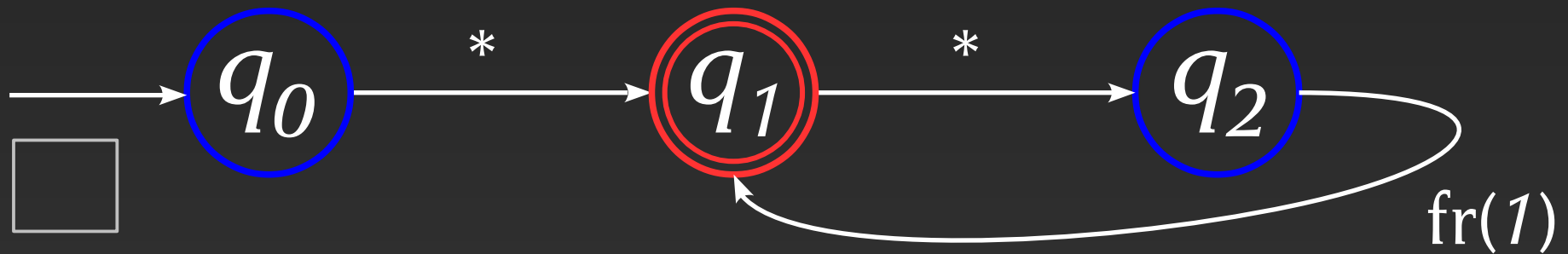
# Fresh-register automata



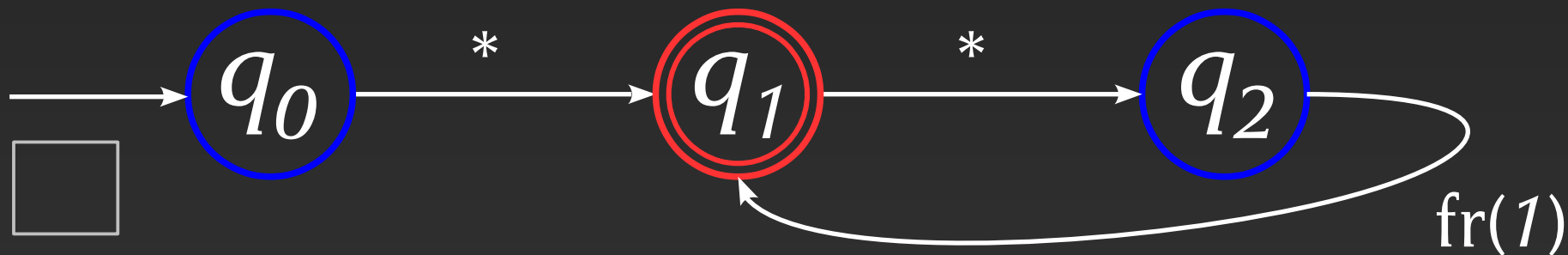
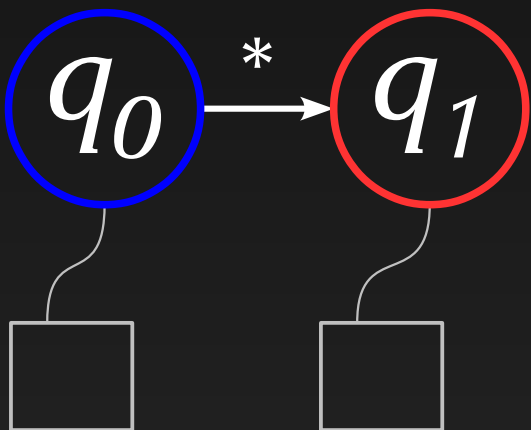
# Fresh-register automata



# Fresh-register automata

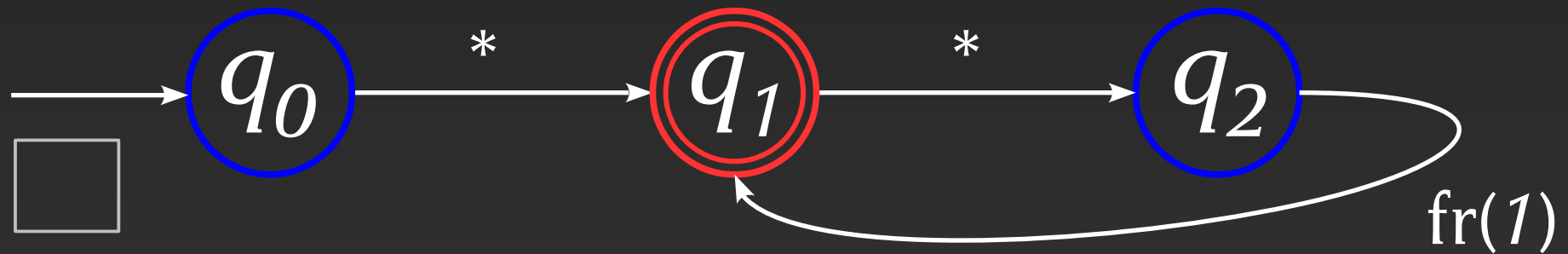
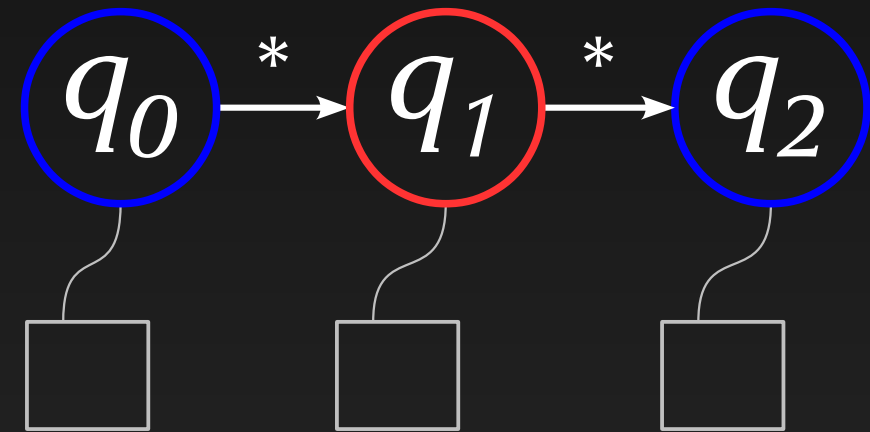


# Fresh-register automata

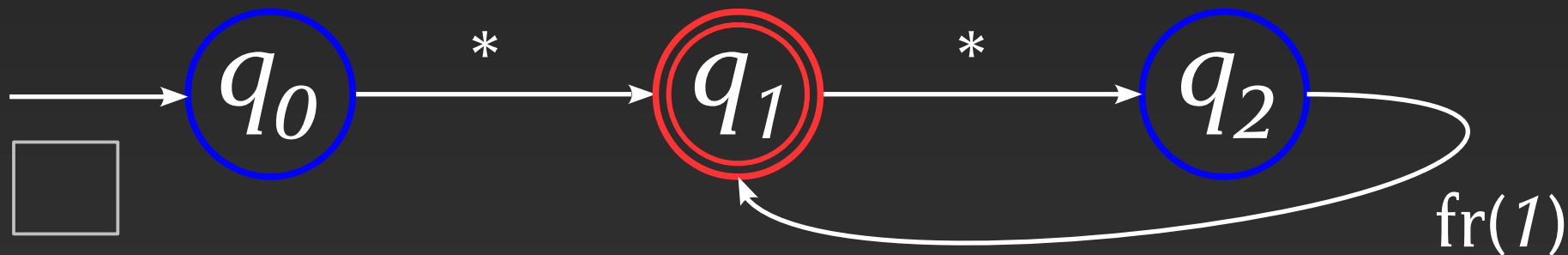
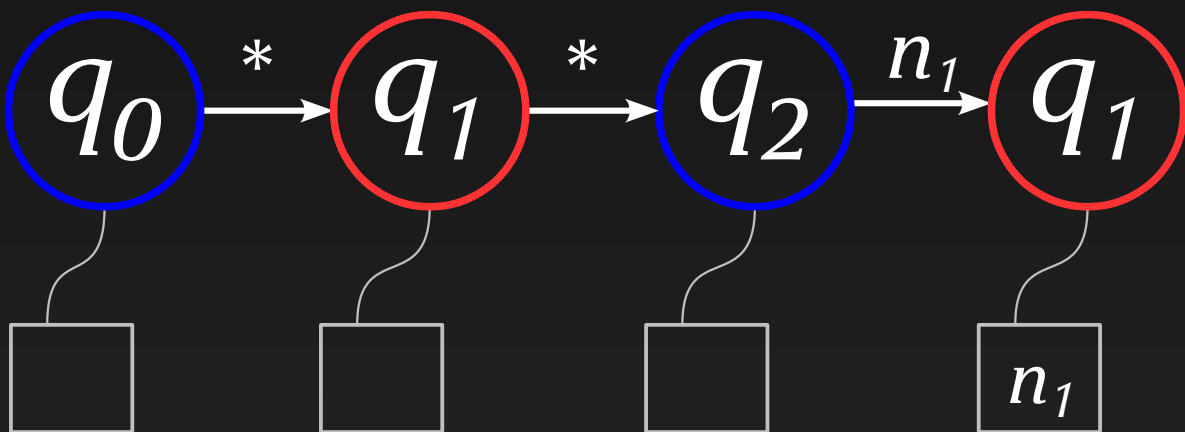




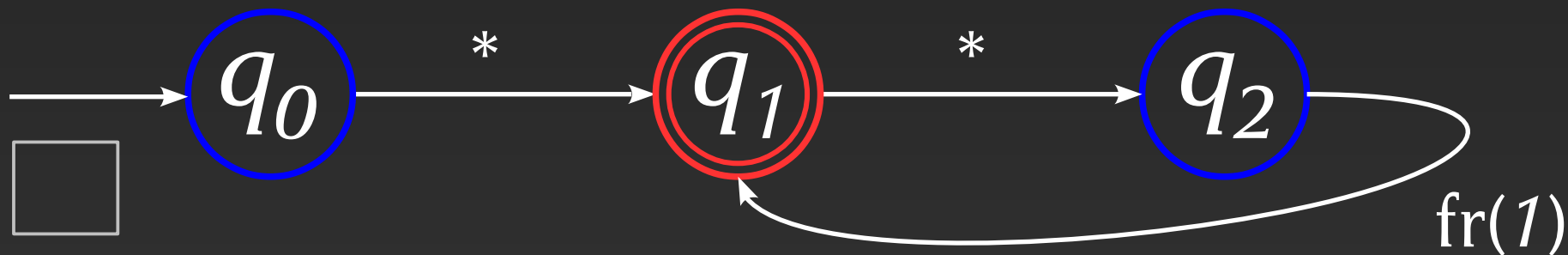
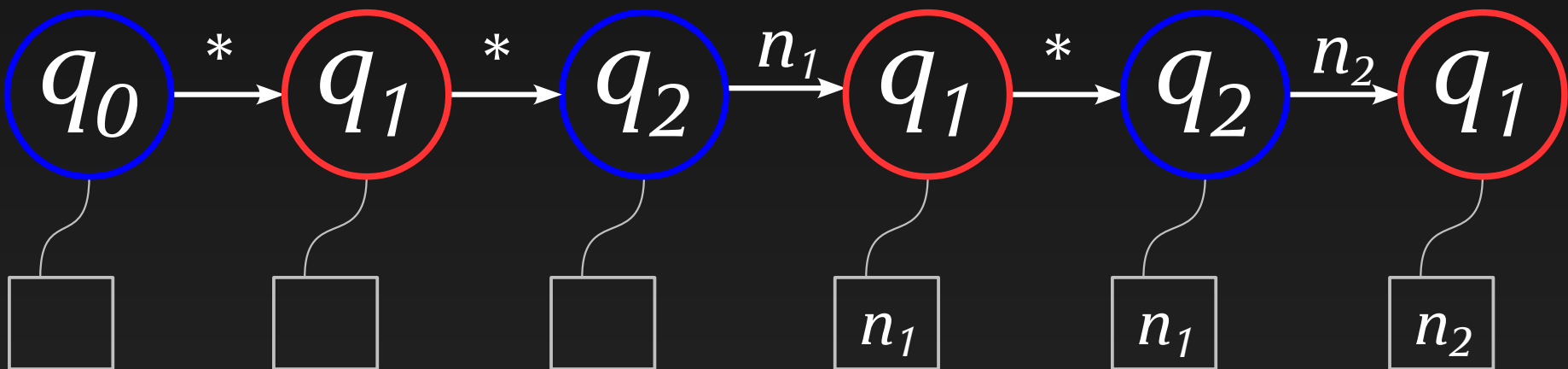
# Fresh-register automata



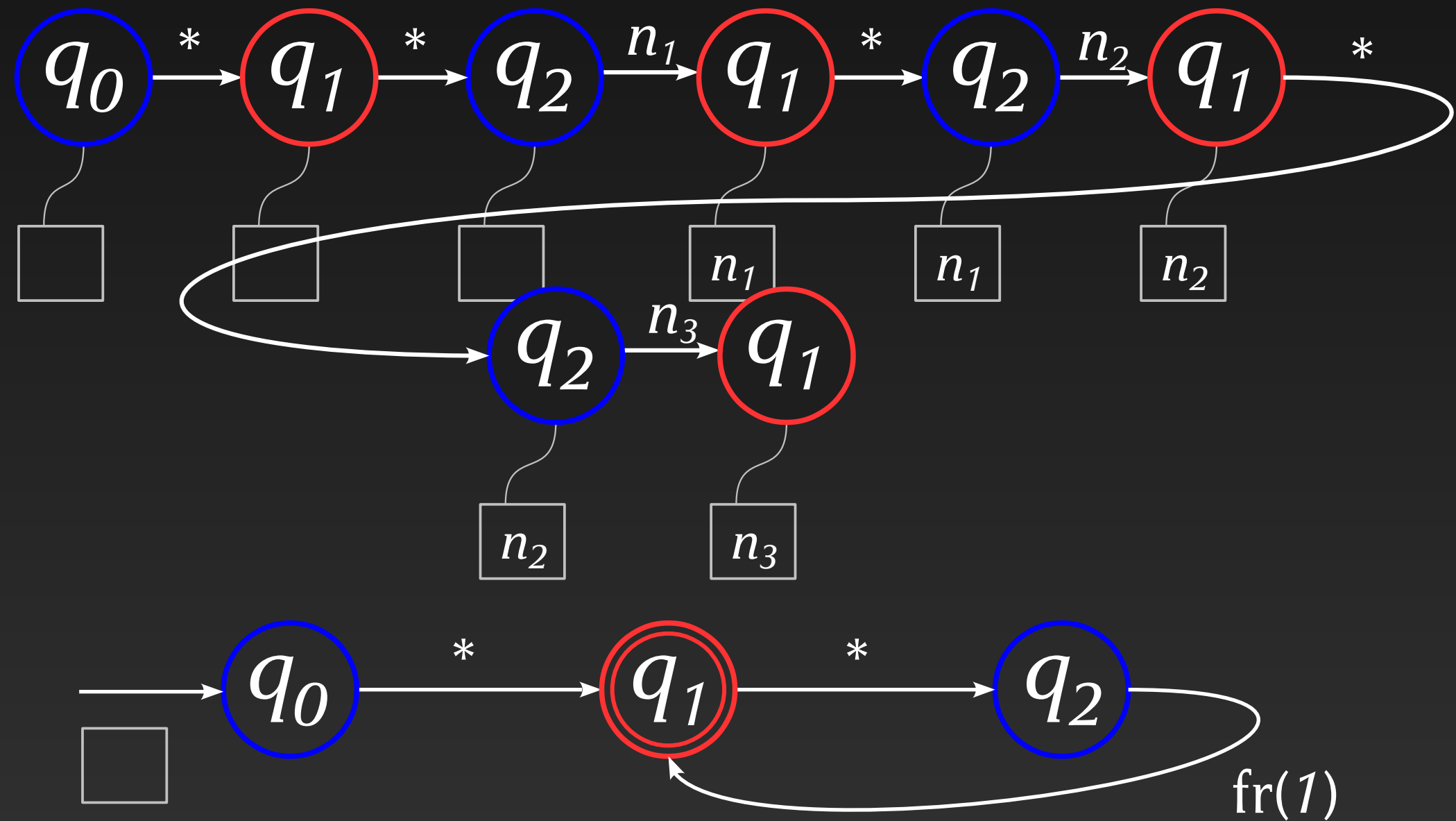
# Fresh-register automata



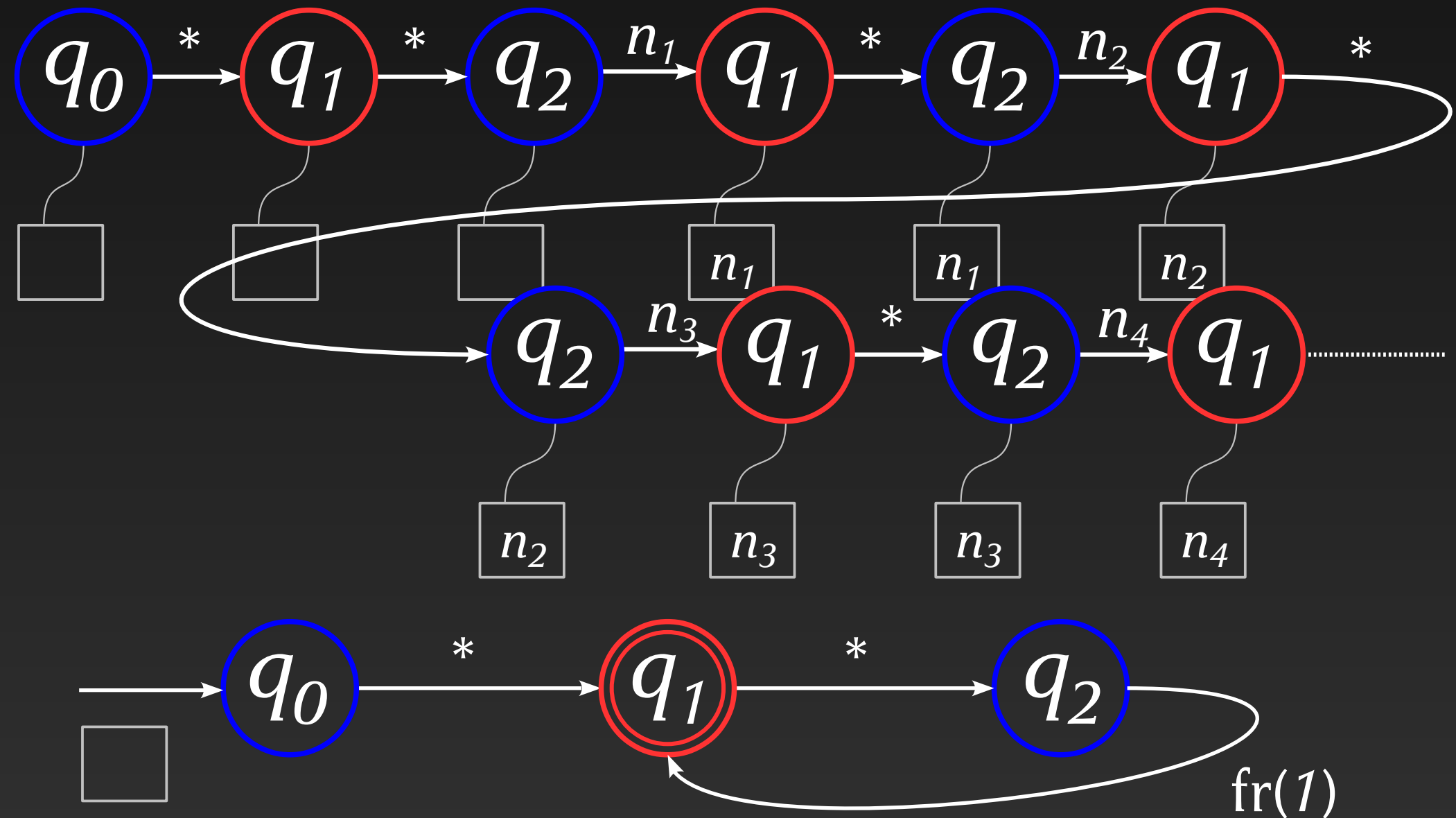
# Fresh-register automata



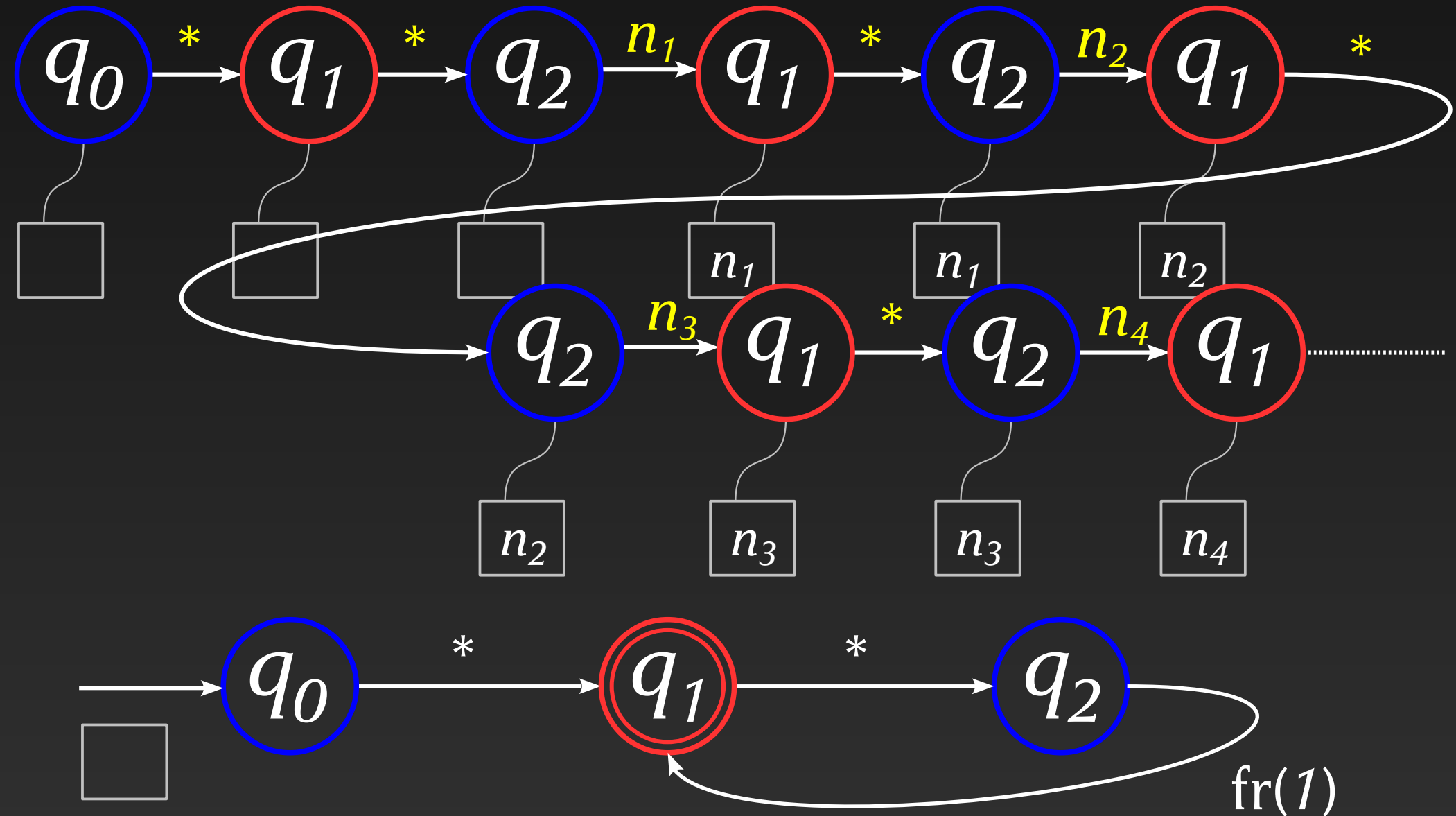
# Fresh-register automata



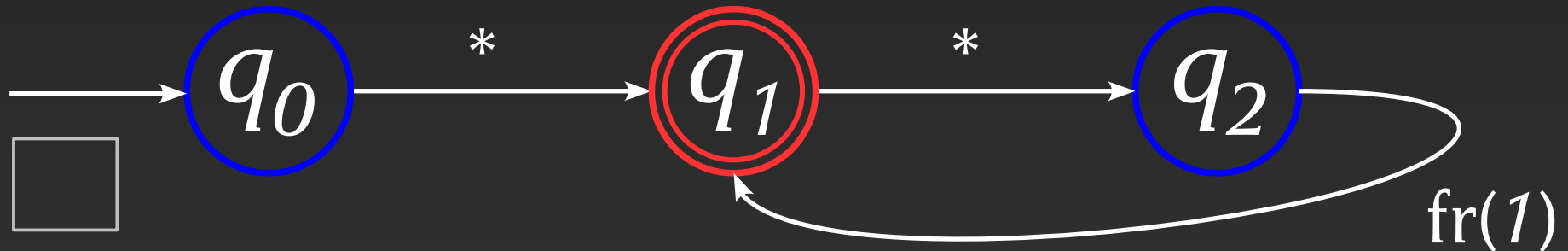
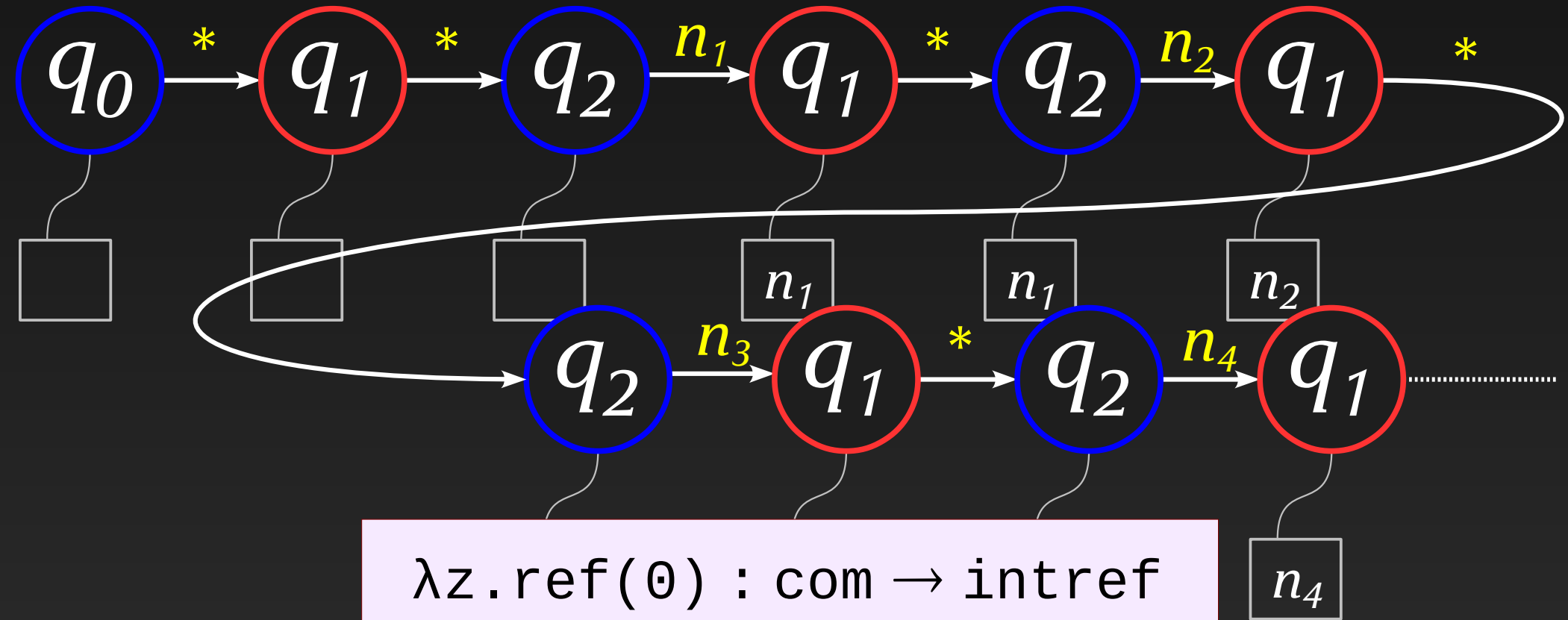
# Fresh-register automata



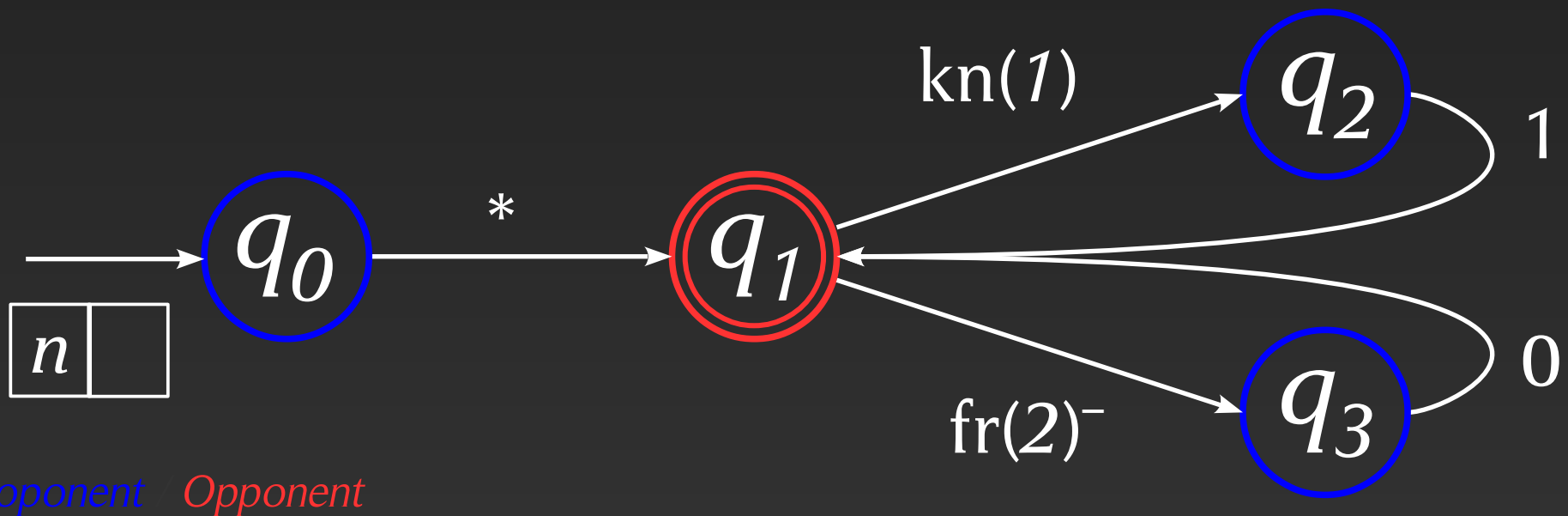
# Fresh-register automata



# Fresh-register automata

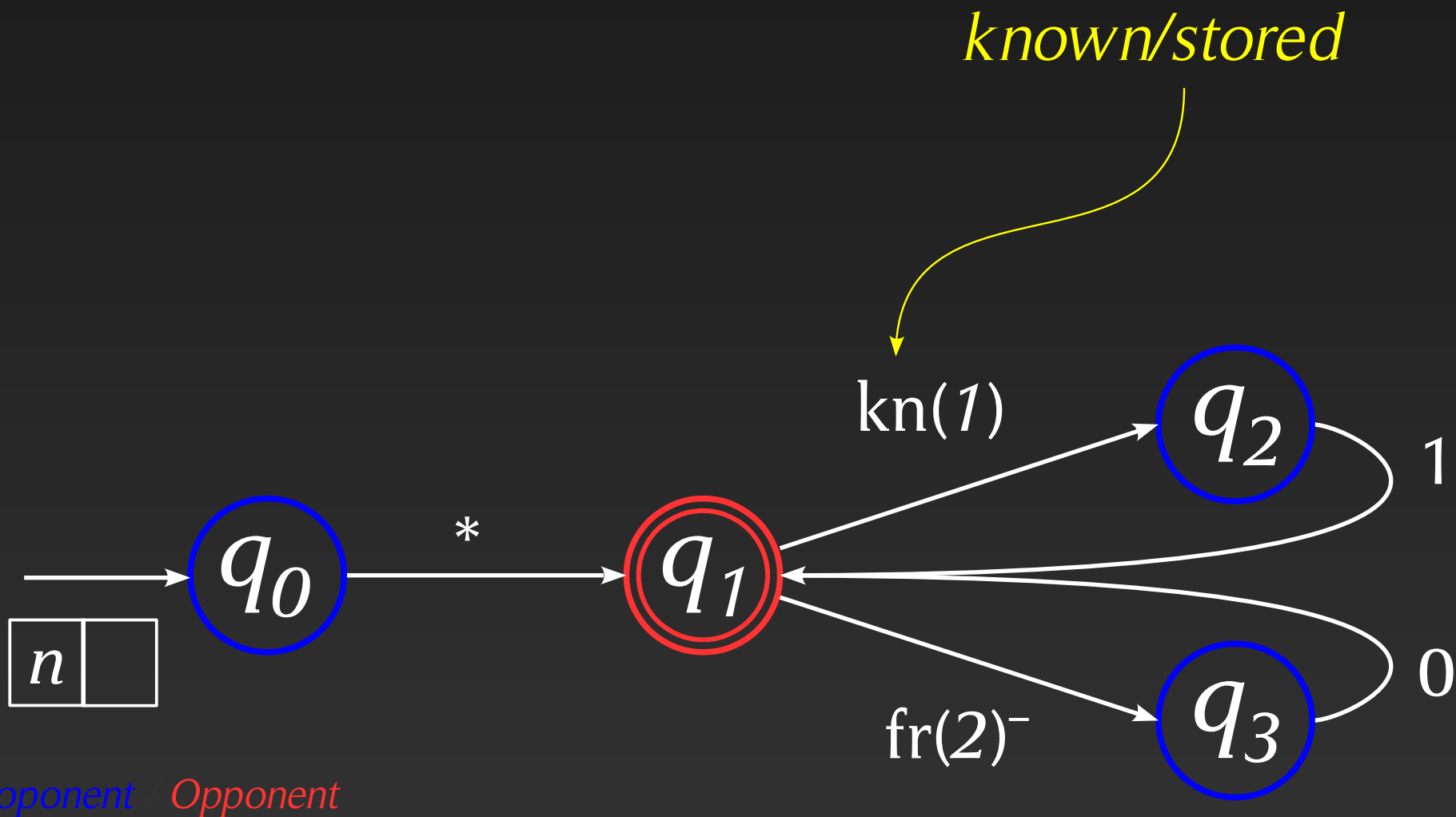


# Fresh-register automata

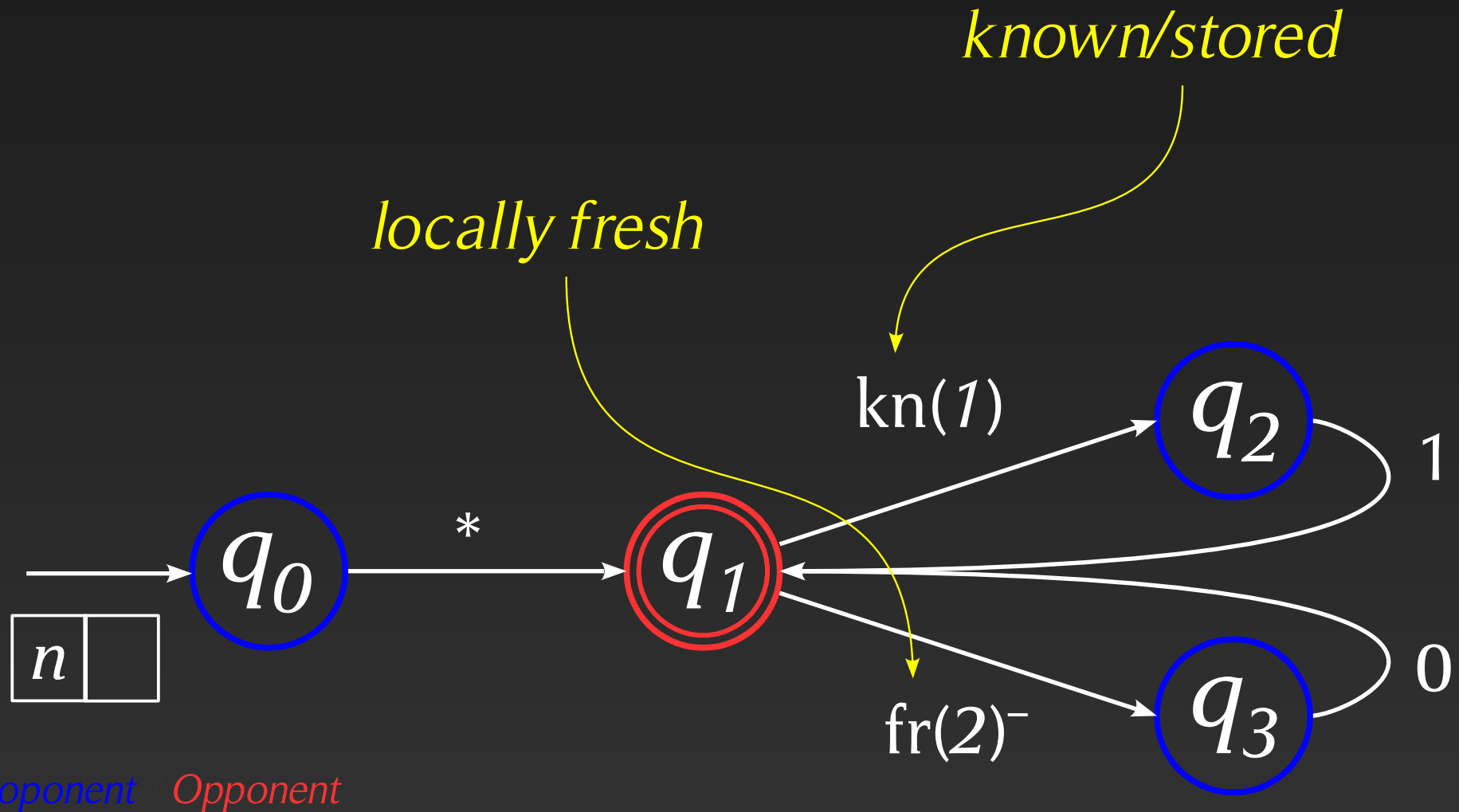




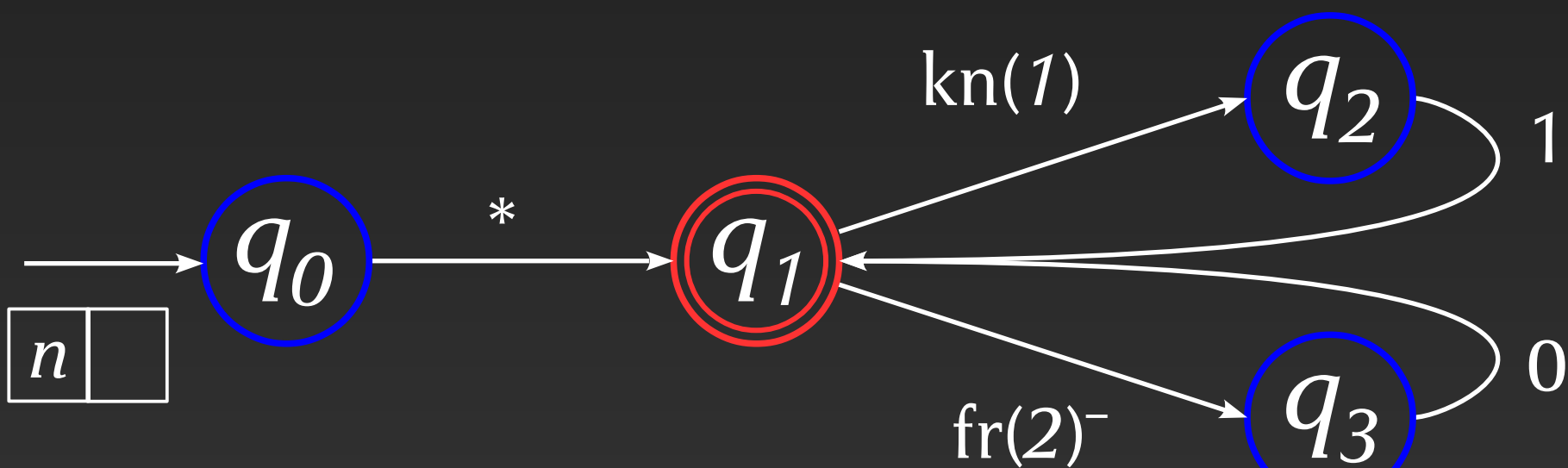
# Fresh-register automata



# Fresh-register automata

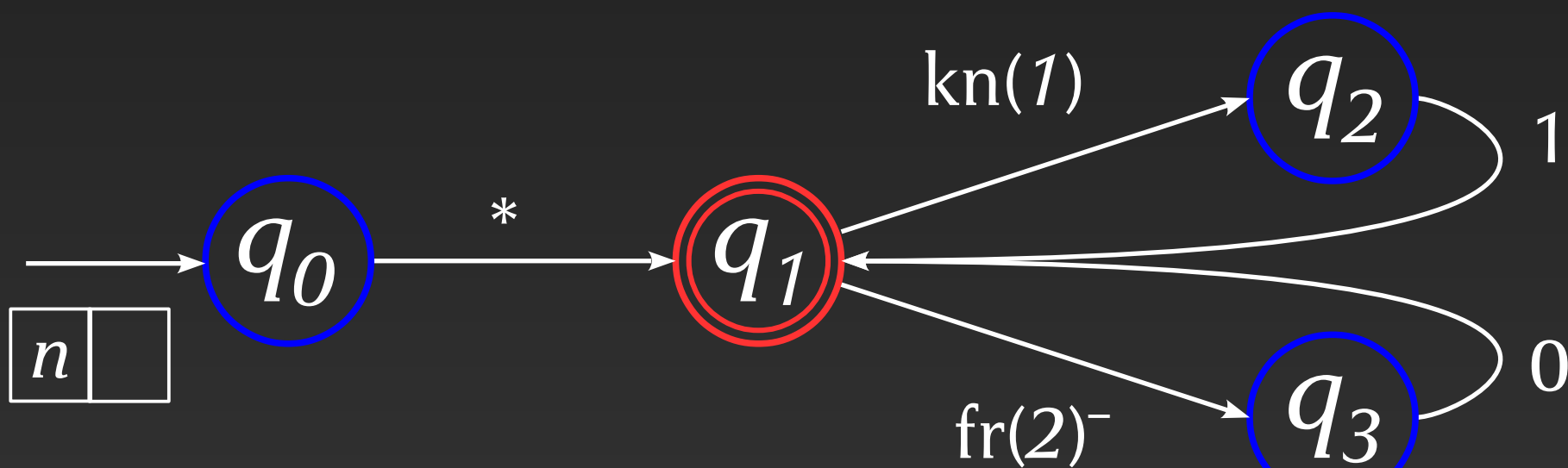
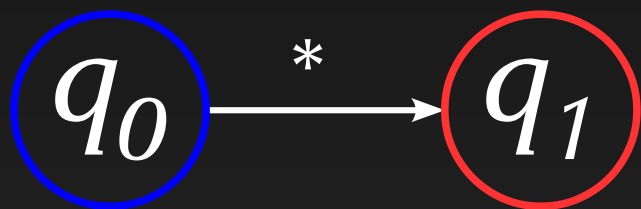


# Fresh-register automata



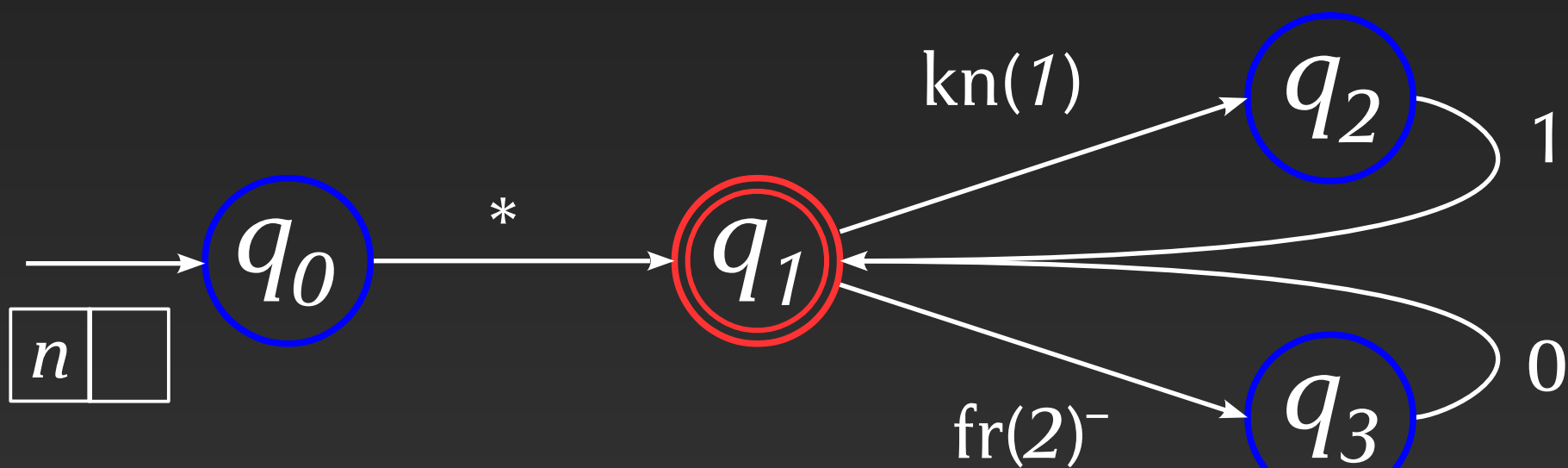
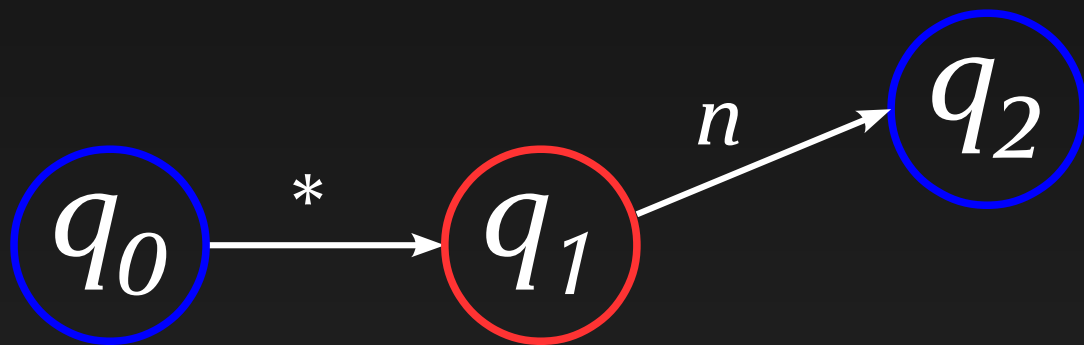
*Proponent* *Opponent*

# Fresh-register automata



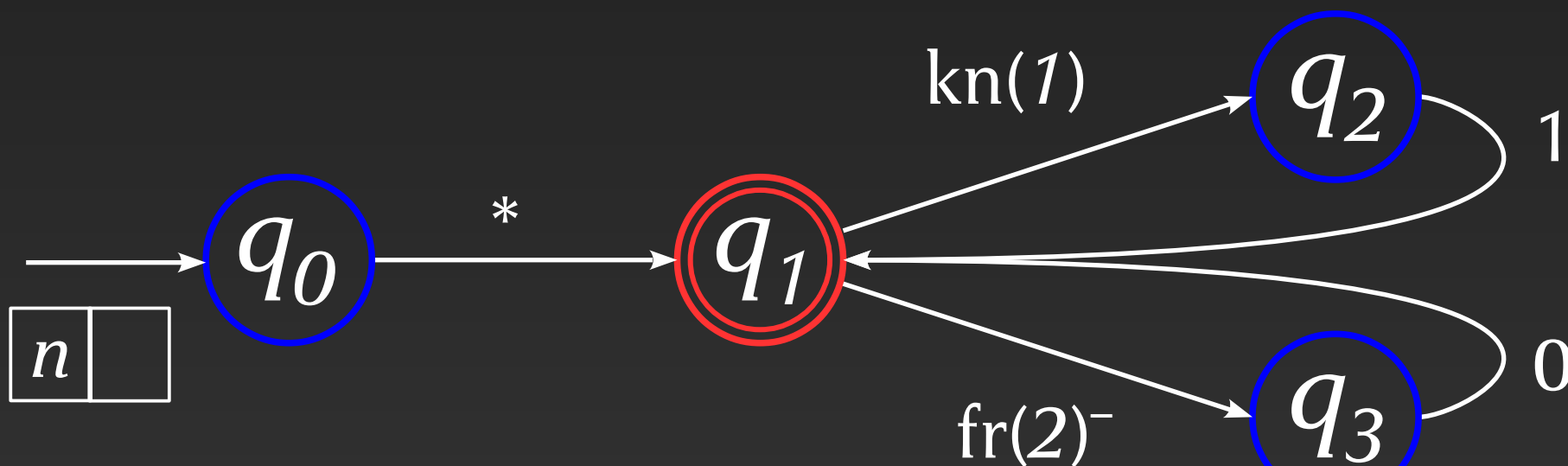
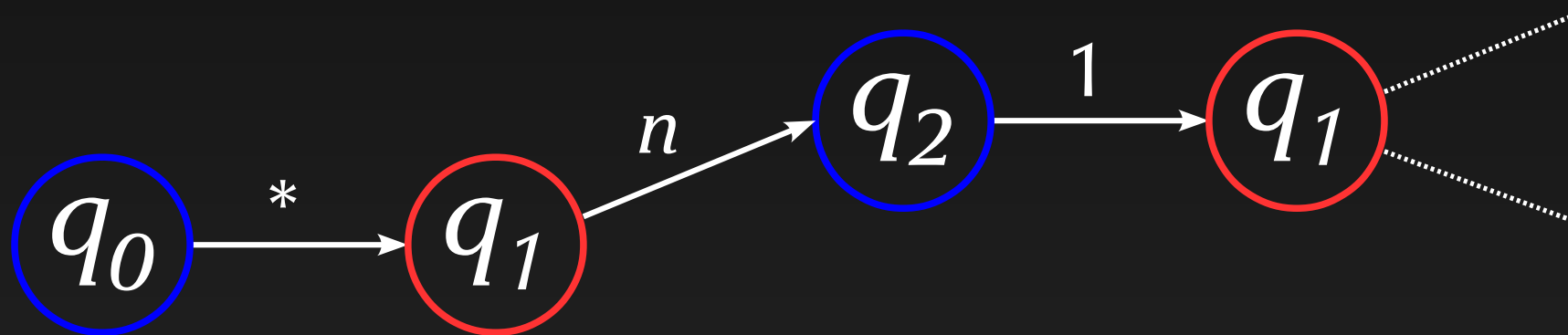
*Proponent* *Opponent*

# Fresh-register automata



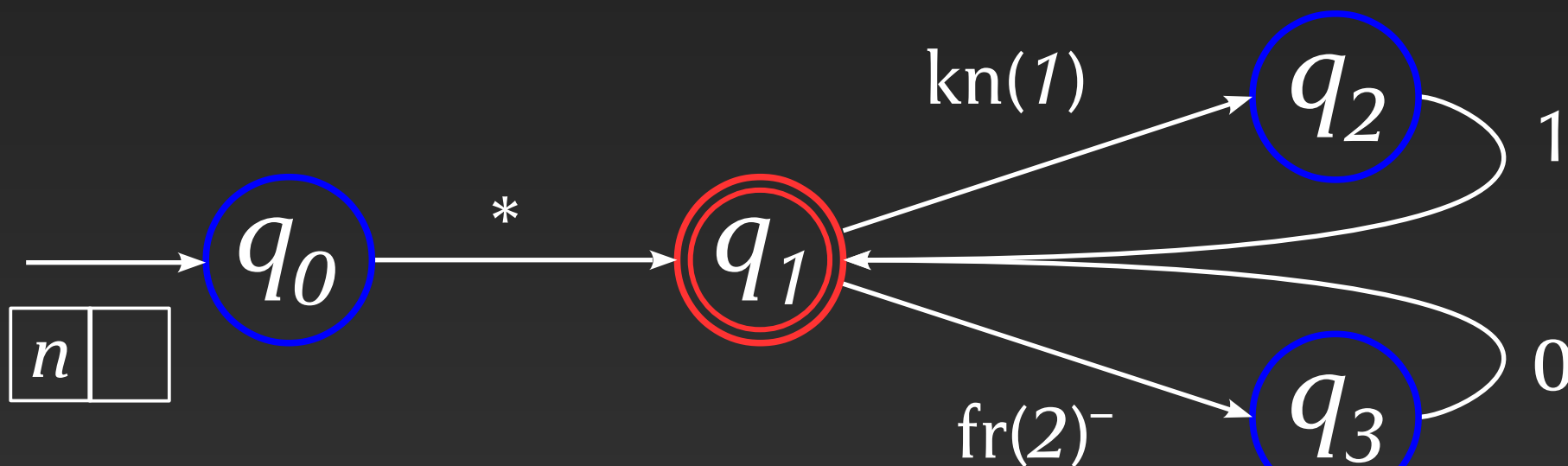
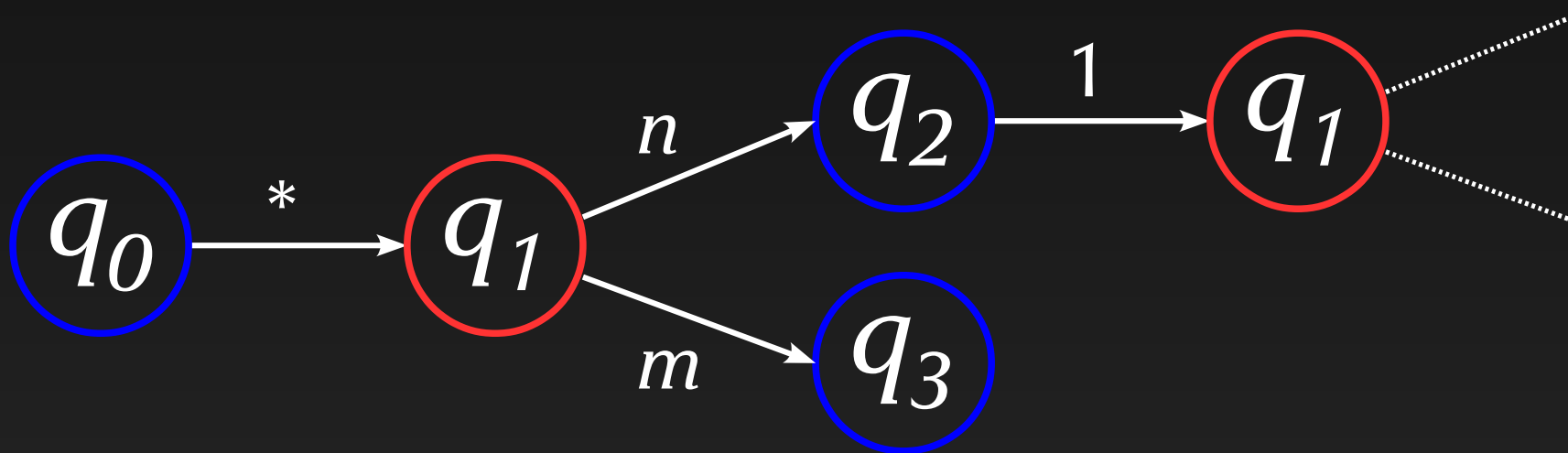
Proponent Opponent

# Fresh-register automata



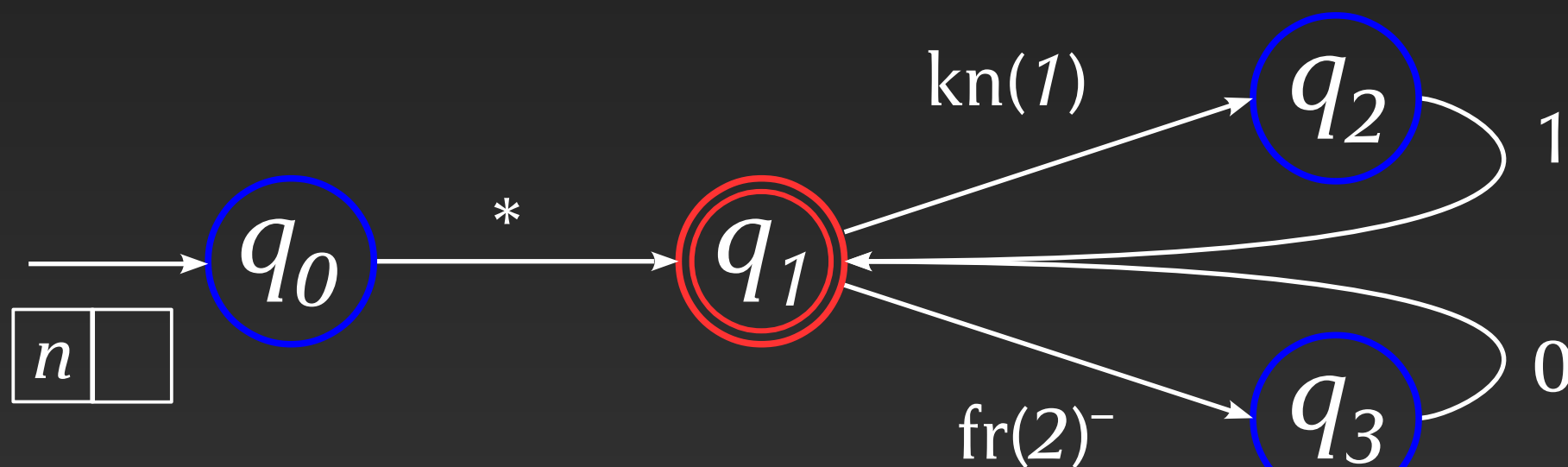
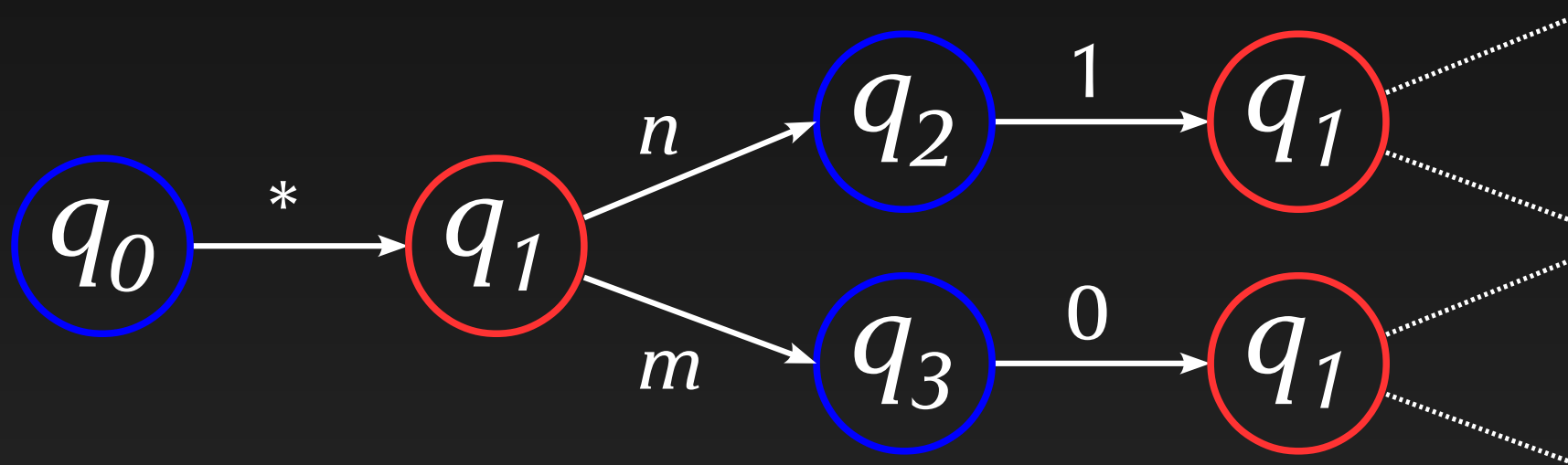
Proponent Opponent

# Fresh-register automata



Proponent Opponent

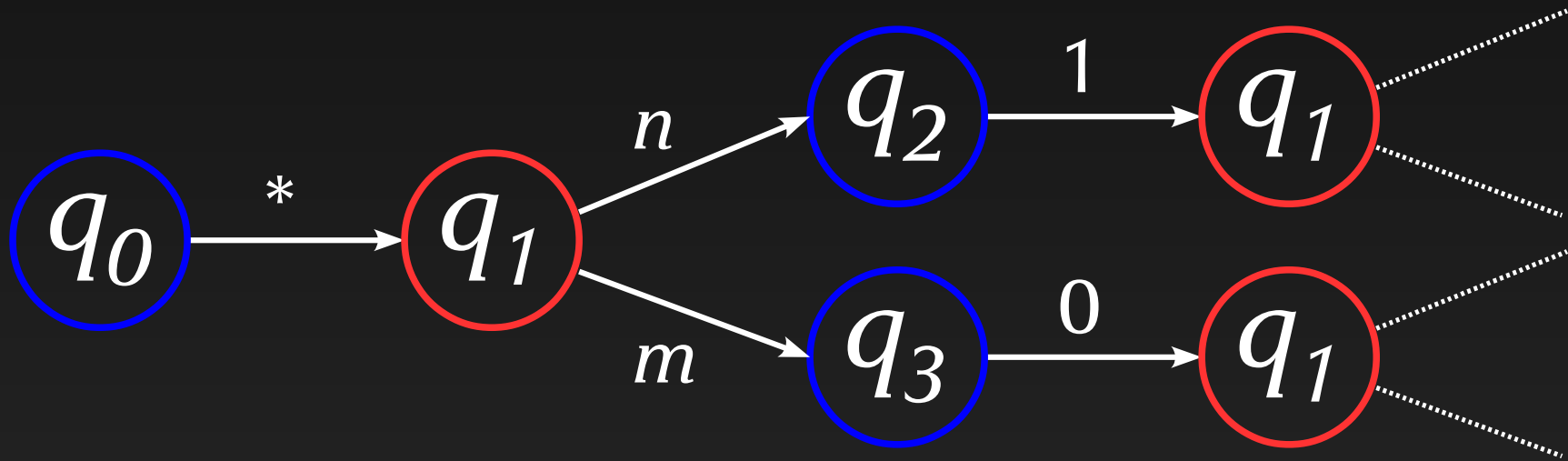
# Fresh-register automata



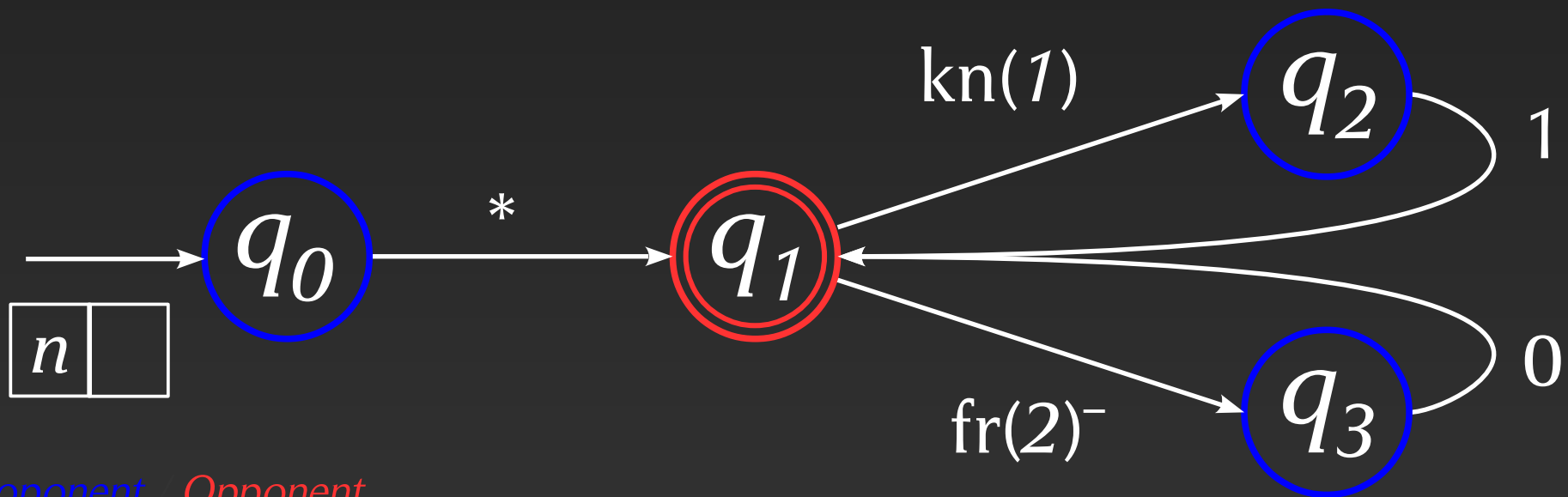
Proponent Opponent



# Fresh-register automata



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

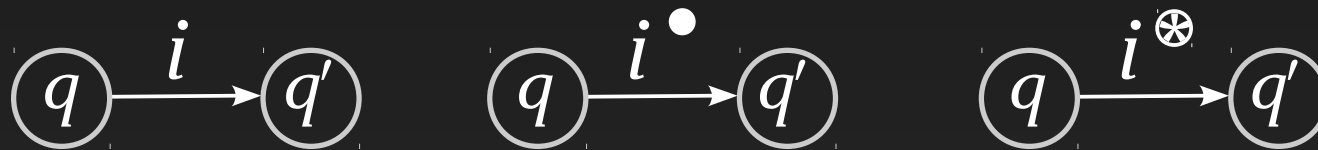


Proponent Opponent

# Automata

- Tz.11: Fresh-Register Automata

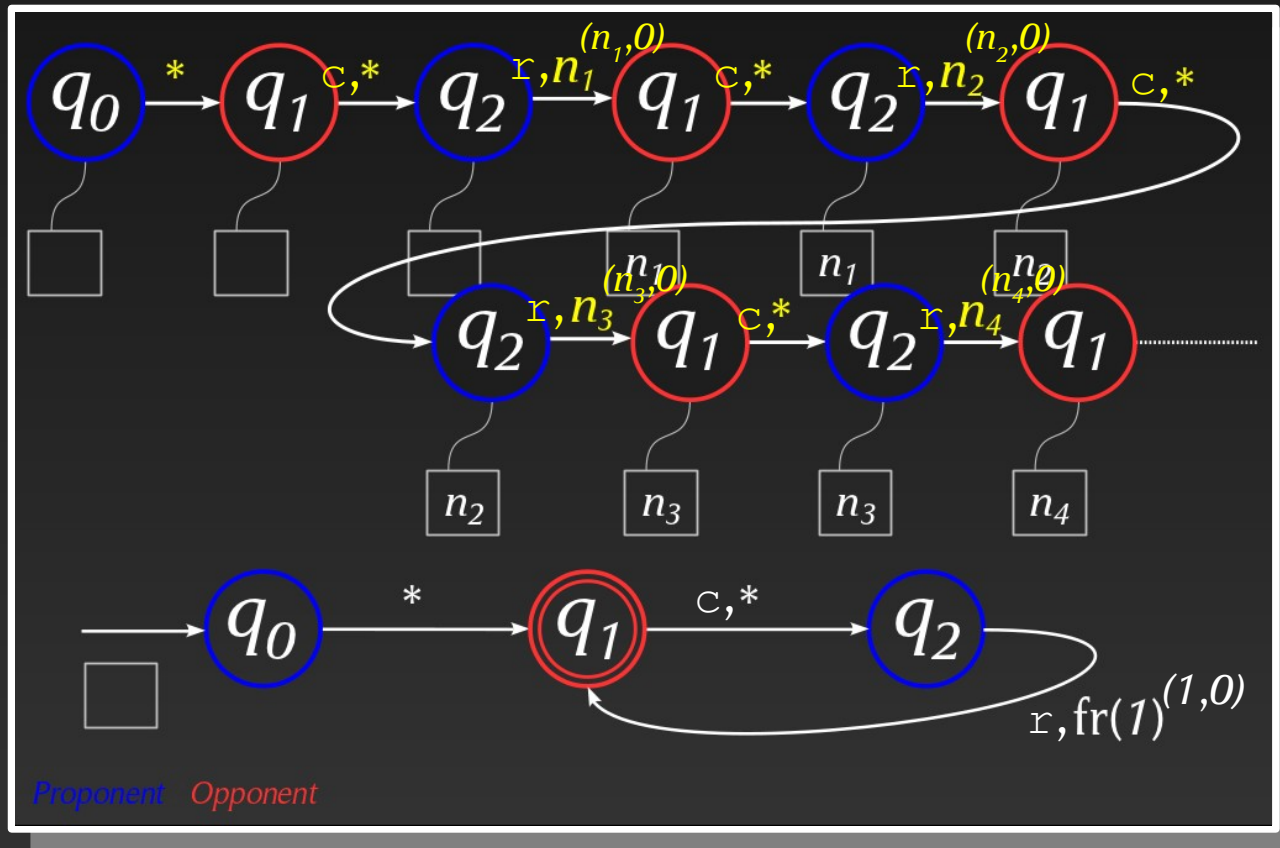
- Basic automata for names



- Closure ( $\cap, \cup$ ), non-closure ( $\bullet, *, -$ )
- Decidability (emptiness, bisimilarity)
- Undecidability (universality, containment)

# Automata

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



# Automata

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

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

# Automata

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

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

- Mu.Tz. 12\* : Pushdown FRA's

# Automata

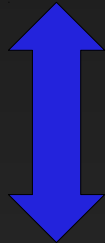
- Tz.11: Fresh-Register Automata
- Mu.Tz. 11': Algorithmic games for RedML\*

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

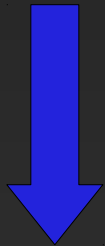
- Mu.Tz. 12\* : Pushdown FRA's
- Mu.Tz. ... : Algorithmic games for Java

# Games with names

Programs

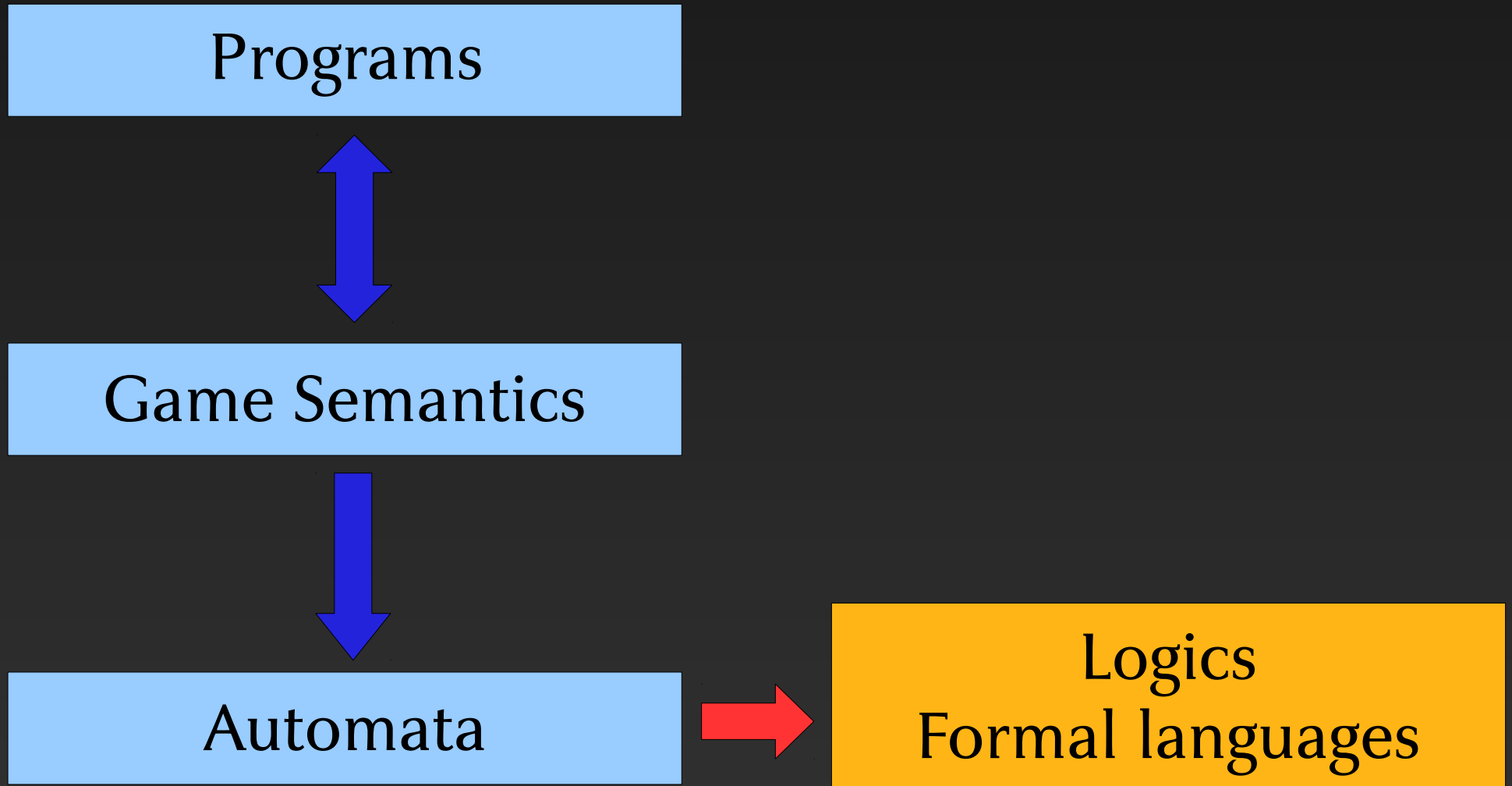


Game Semantics



Automata

# Games with names





# Games with names

