Nominal game semantics and automata

Nikos Tzevelekos

Queen Mary University of London

Supported by an RAEng research fellowship
Semantics of proofs and programs, IHP, June 2014

What this talk is about

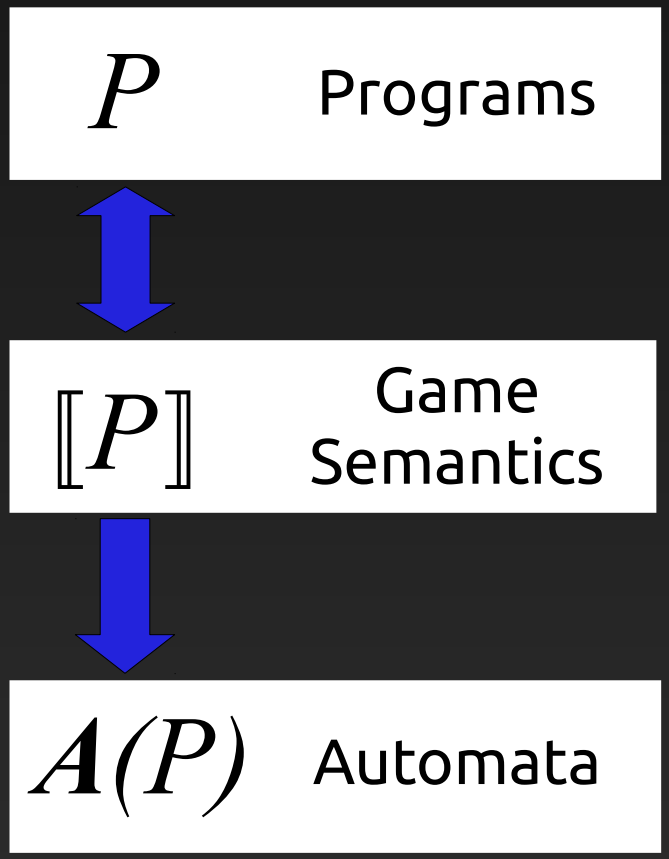
We give an overview of techniques from game semantics and automata for modelling and analysing programs

A distinctive feature of the approach is the handling of dynamic resources:

references, objects, channels, exceptions…

Resources are modelled as names with associated effects → nominal games & automata

Overview



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

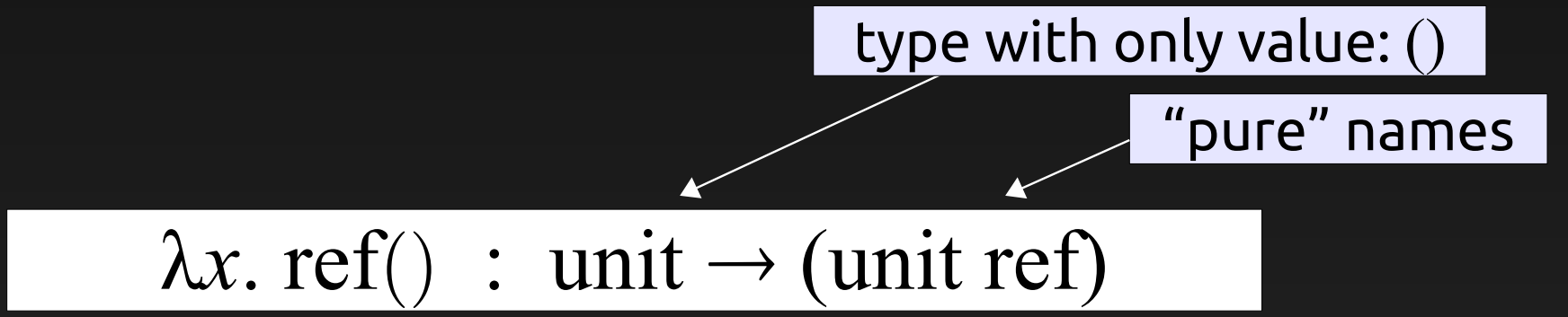
$$P \vdash \varphi \;\Leftrightarrow\; A(P) \vdash \varphi$$

Computation with names (Java)

```
...
  public void foo() {
    // Create new list
    List x = new ArrayList();

    x.add(1); x.add(2);
    Iterator i = x.iterator();
    Iterator j = x.iterator();
    i.next(); i.remove(); j.next();
  }
...
```

Computation with names (ML)



Computation with names (ML)

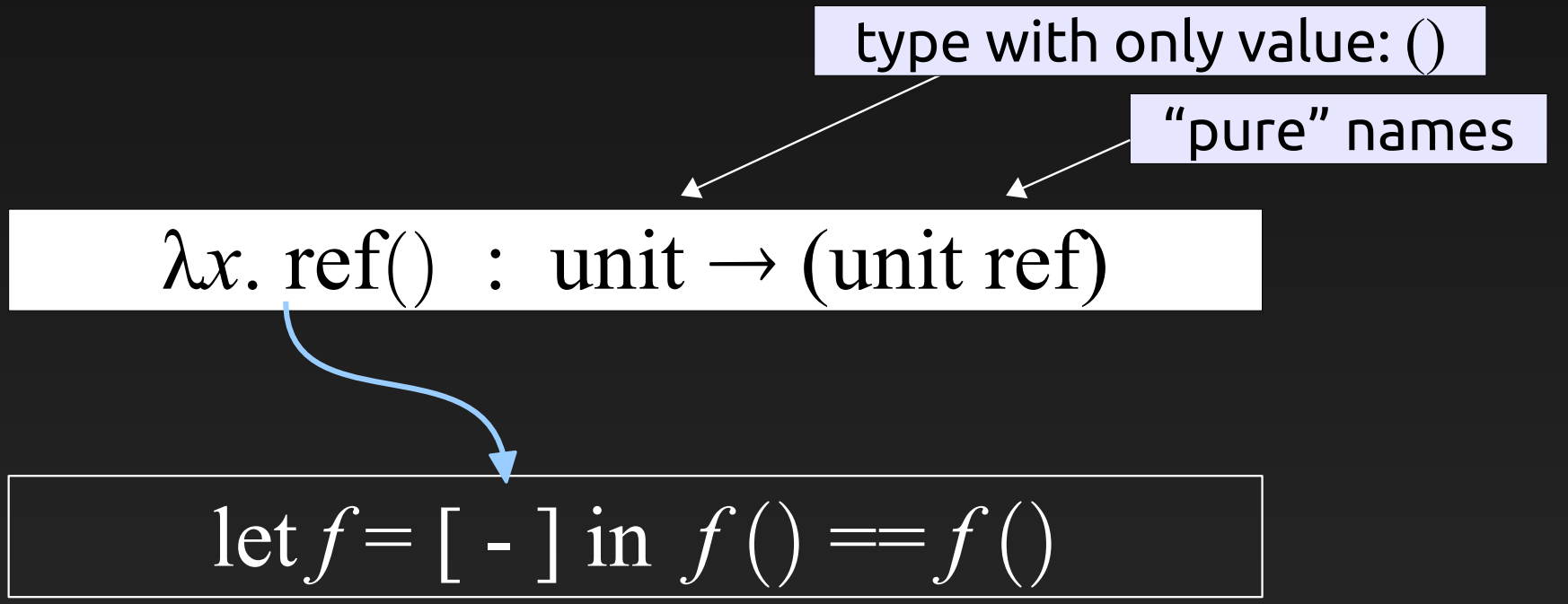
type with only value: ()

“pure” names

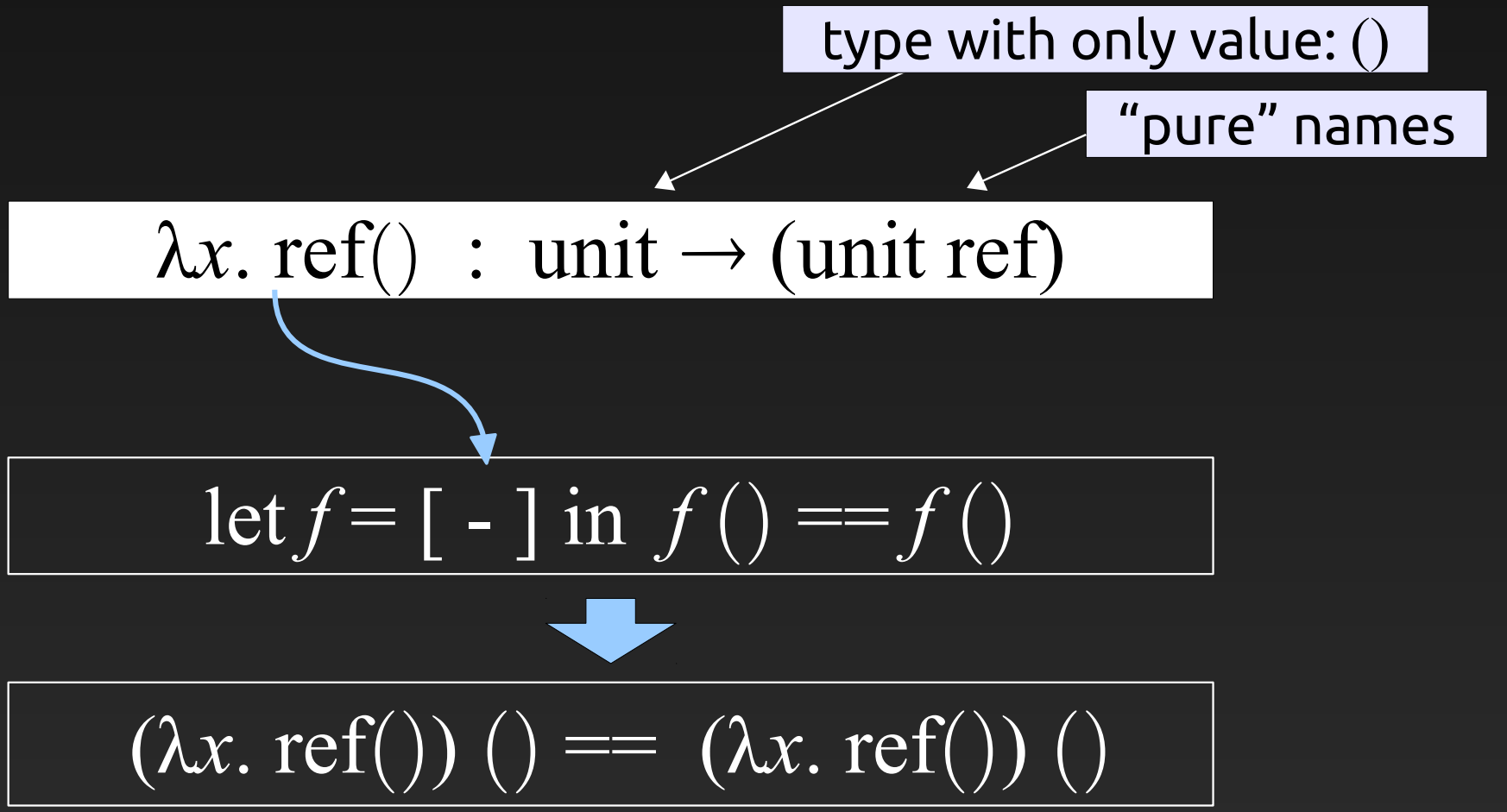
$$\lambda x.\ \mathrm{ref}()\ :\ \mathrm{unit} \to (\mathrm{unit\ ref})$$

$$\mathrm{let}\ f = [\ \text{-}\ ]\ \mathrm{in}\ \ f() == f()$$

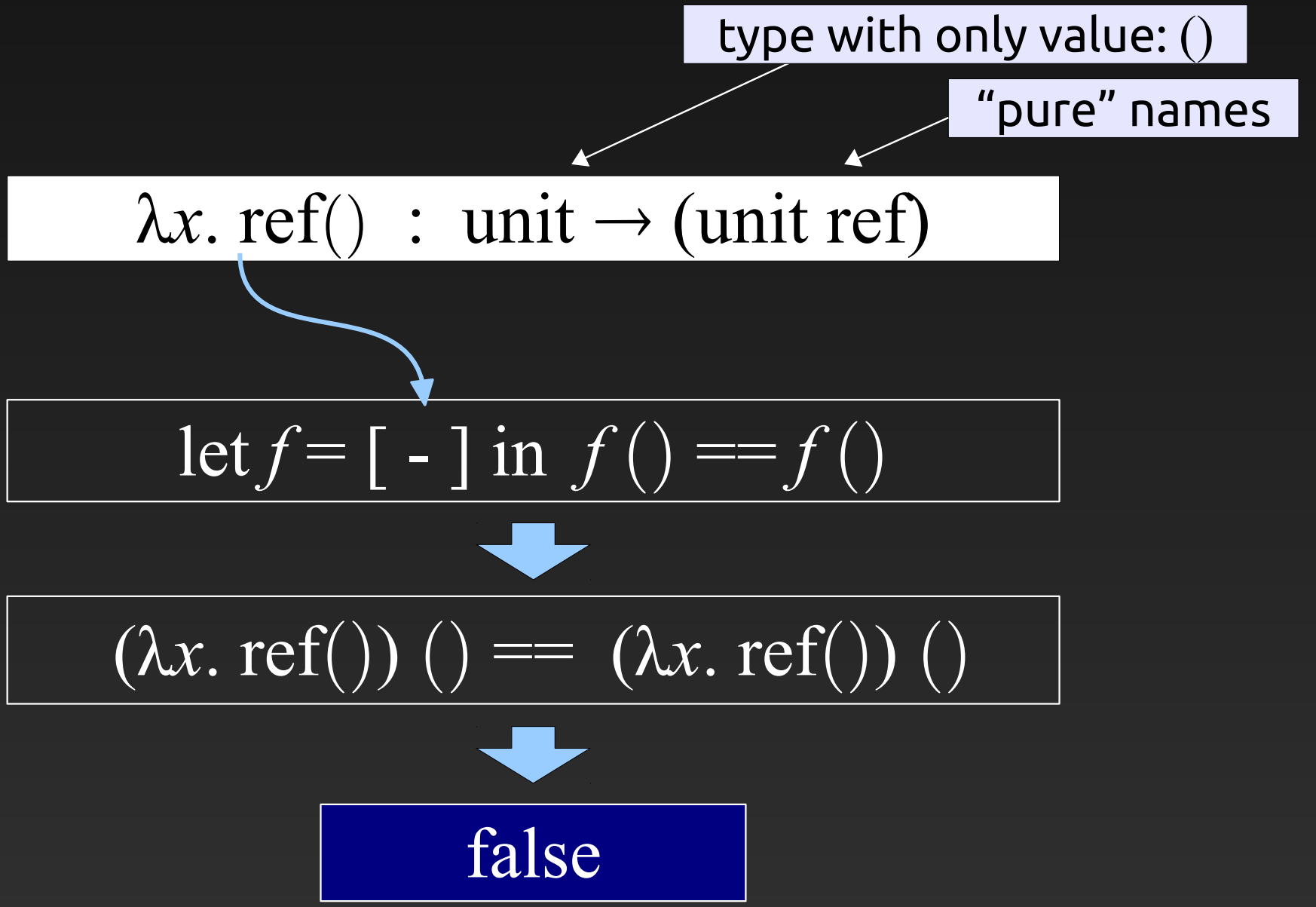
Computation with names (ML)



Computation with names (ML)



Computation with names (ML)



Example (ν-calculus)

Call-by-value language with unit references

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

$$P \cong P'$$

same observable behaviour in every context

Example (ν-calculus)

Call-by-value language with unit references

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

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

$$P \cong P'$$

same observable behaviour in every context

Example (ν-calculus)

Call-by-value language with unit references

"Create a *fresh reference* ; then *compare* input references with the fresh one and return the result of the comparison"

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

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

$$P \cong P'$$

same observable behaviour in every context

Full Abstraction

How to assign denotations to programs,

$$[\![\_]\!] : \text{Syntax} \longrightarrow \mathcal{M}$$

such that:

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

Full Abstraction

How to assign denotations to programs,

$$[\![\_]\!] : \text{Syntax} \longrightarrow \mathcal{M}$$

such that:

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

Here the Syntax will be some fragment of ML,
i.e. call-by-value PCF + references + exceptions + …

$$T ::= \text{unit} \mid \text{int} \mid T\,\text{ref} \mid T \to T \mid \ldots$$

$$M ::= () \mid i \mid a \mid x \mid \lambda x.M \mid M\,M \mid \text{ref}\,M \mid M := M \mid !M \mid \boxed{M == M \mid \ldots}$$

Game Semantics

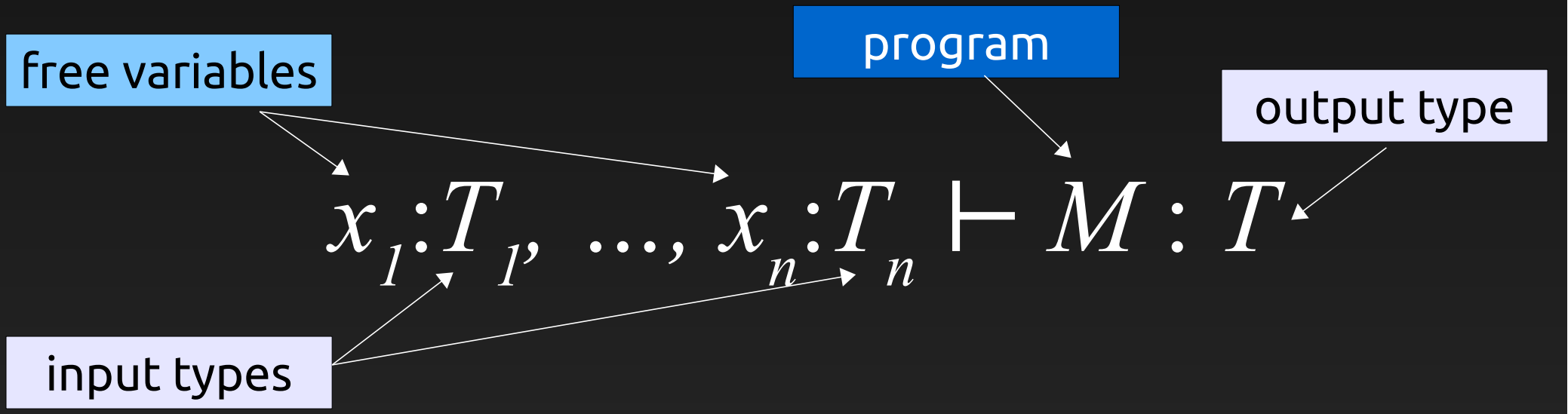
Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment), aka *O*
  - *Proponent* (the program), aka *P*
- Qualitative games (≠ Game Theory)
- Programs = *strategies* for *P*
- *Categories* of games

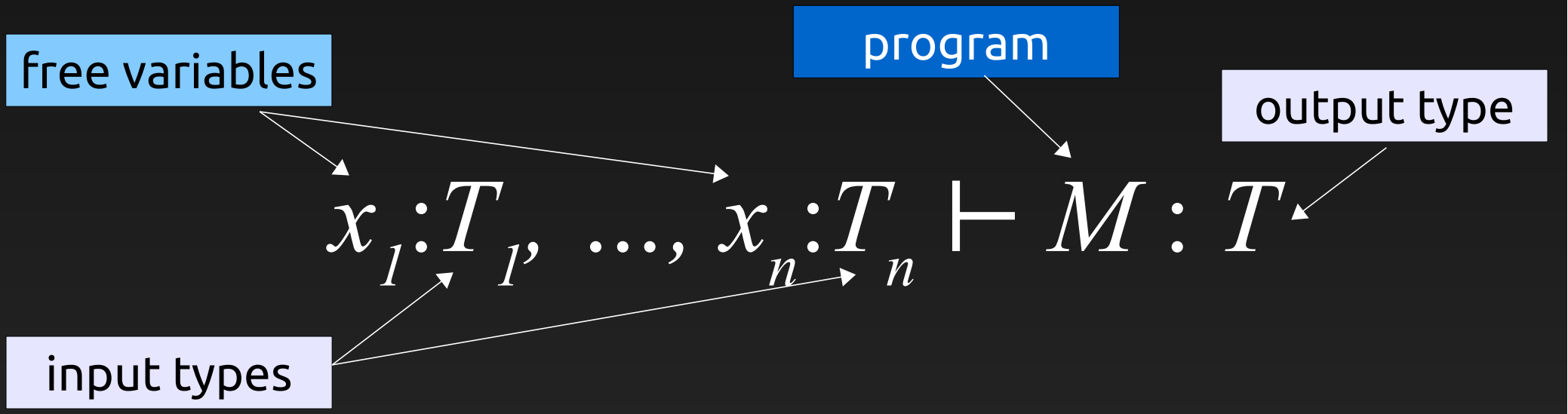
Games played in arenas

$$x_1 : T_1,\ \ldots,\ x_n : T_n \vdash M : T$$

Games played in arenas

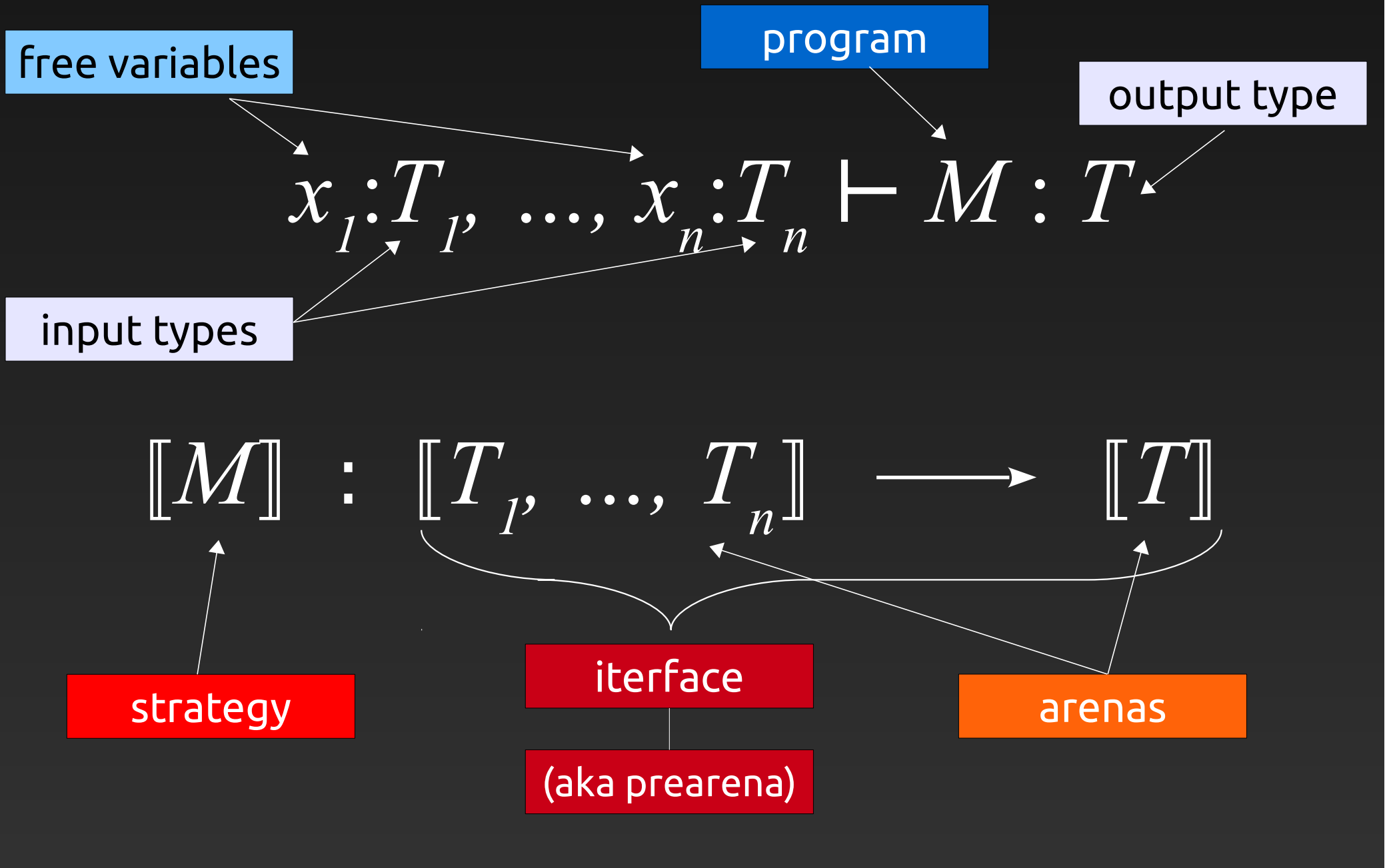


Games played in arenas

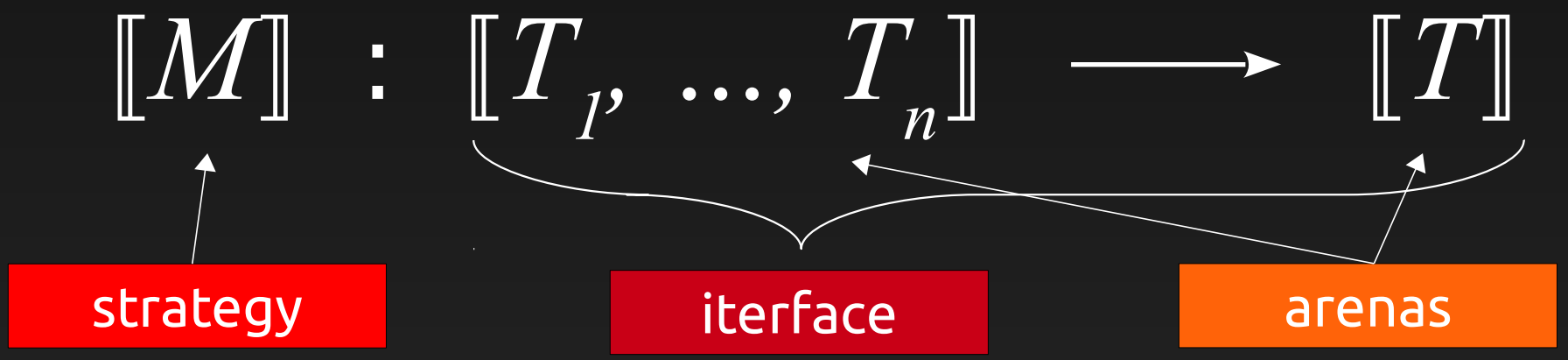


$$[\![M]\!] : [\![T_1, \ldots, T_n]\!] \longrightarrow [\![T]\!]$$

Games played in arenas

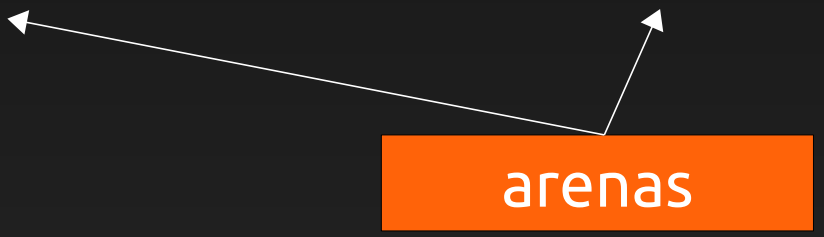


Arenas of moves



Arenas of moves

$$[\![M]\!] : [\![T_1, \ldots, T_n]\!] \longrightarrow [\![T]\!]$$



$[\![\text{unit}]\!] = \{ * \}$

$[\![\text{int}]\!] = \{ 0, 1, -1, \ldots \}$

$[\![T \text{ ref}]\!] = \{ a, b, \ldots \}$ $\quad a, b, \ldots \in \mathcal{N}_T$

...

Arenas of moves

$$[\![M]\!] : [\![T_1, \ldots, T_n]\!] \longrightarrow [\![T]\!]$$

arenas

$\mathcal{N}_T$ a set of *names*:
- Infinitely many
- Comparable for equality only

$$[\![\text{unit}]\!] = \{ * \}$$

$$[\![\text{int}]\!] = \{ 0, 1, -1, \ldots \}$$

$$[\![T \text{ ref}]\!] = \{ a, b, \ldots \}$$

$$a, b, \ldots \in \mathcal{N}_T$$

Arenas of moves

$$[\![M]\!] : [\![T_1, \ldots, T_n]\!] \longrightarrow [\![T]\!]$$

arenas

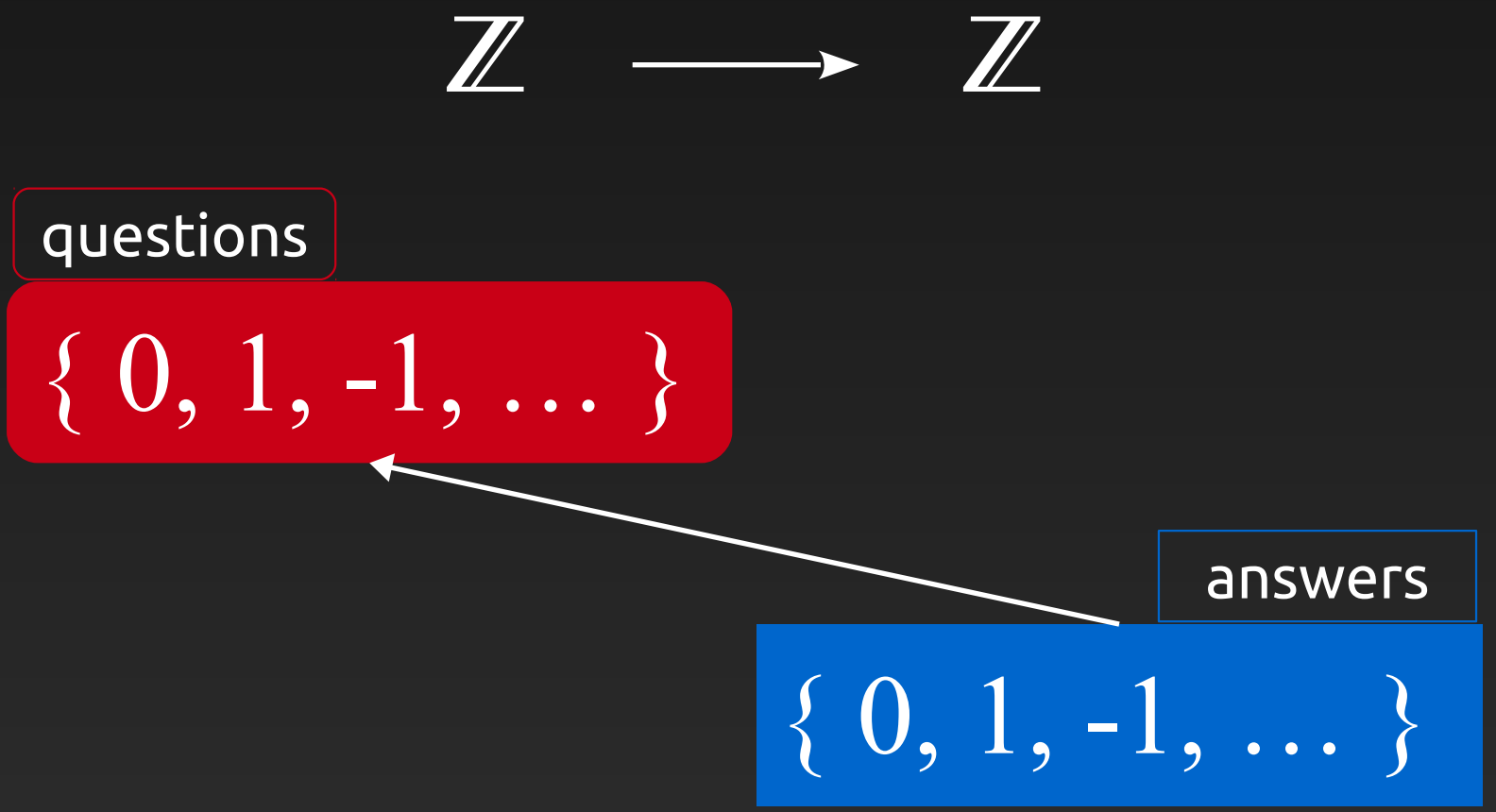
$\mathcal{N}_T$ a set of *names*:
- Infinitely many
- Comparable for equality only

$$[\![\text{unit}]\!] = \{ * \} = \mathbf{1}$$

$$[\![\text{int}]\!] = \{ 0, 1, -1, \ldots \} = \mathbb{Z}$$

$$[\![T \text{ ref}]\!] = \{ a, b, \ldots \} = \mathbb{A}_T \qquad a, b, \ldots \in \mathcal{N}_T$$

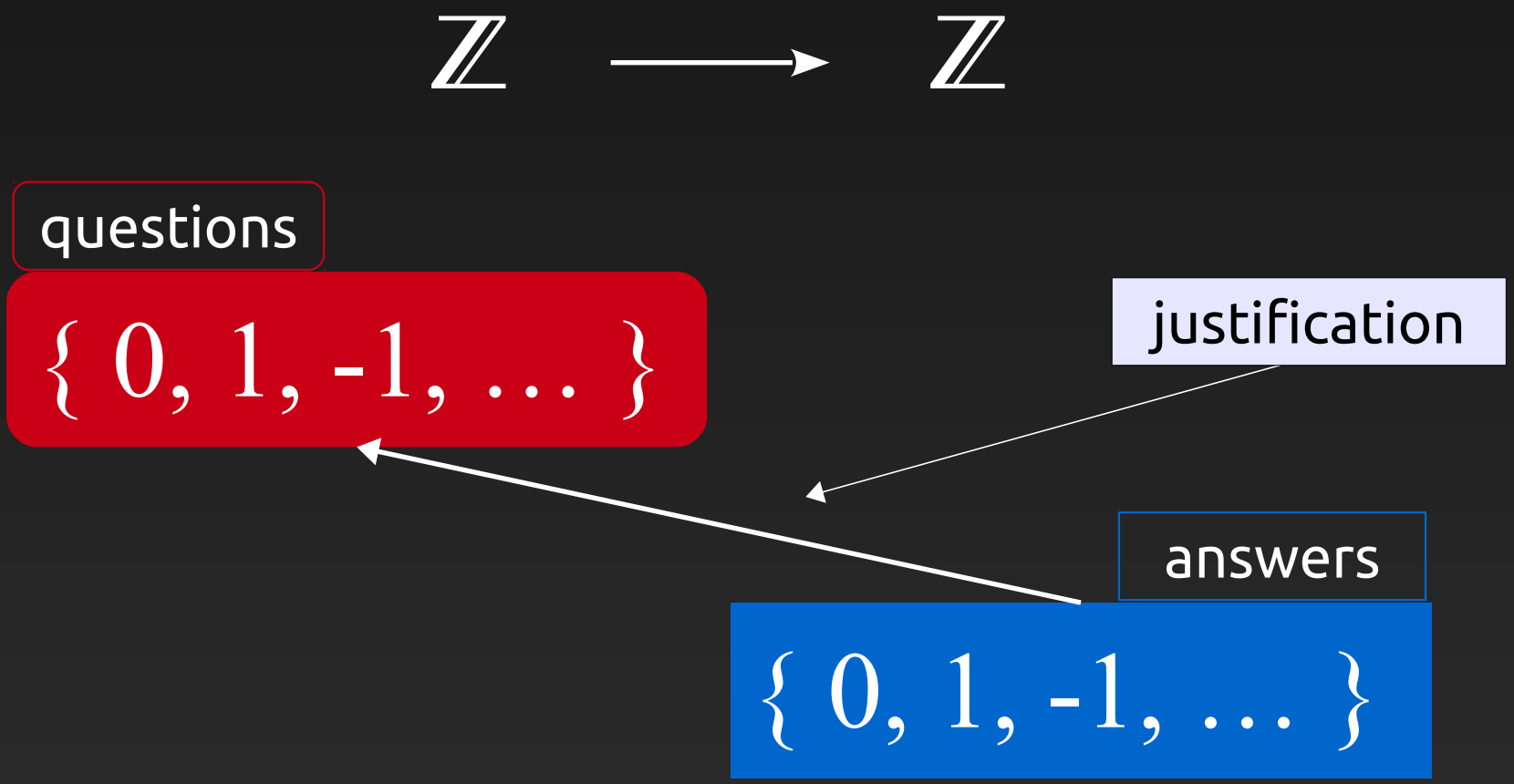
A simple interface



O : Opponent

P : Proponent

A simple interface

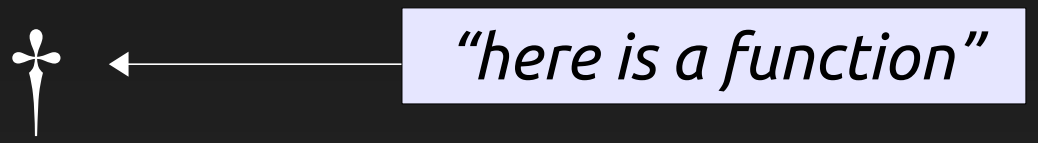


O : Opponent

P : Proponent

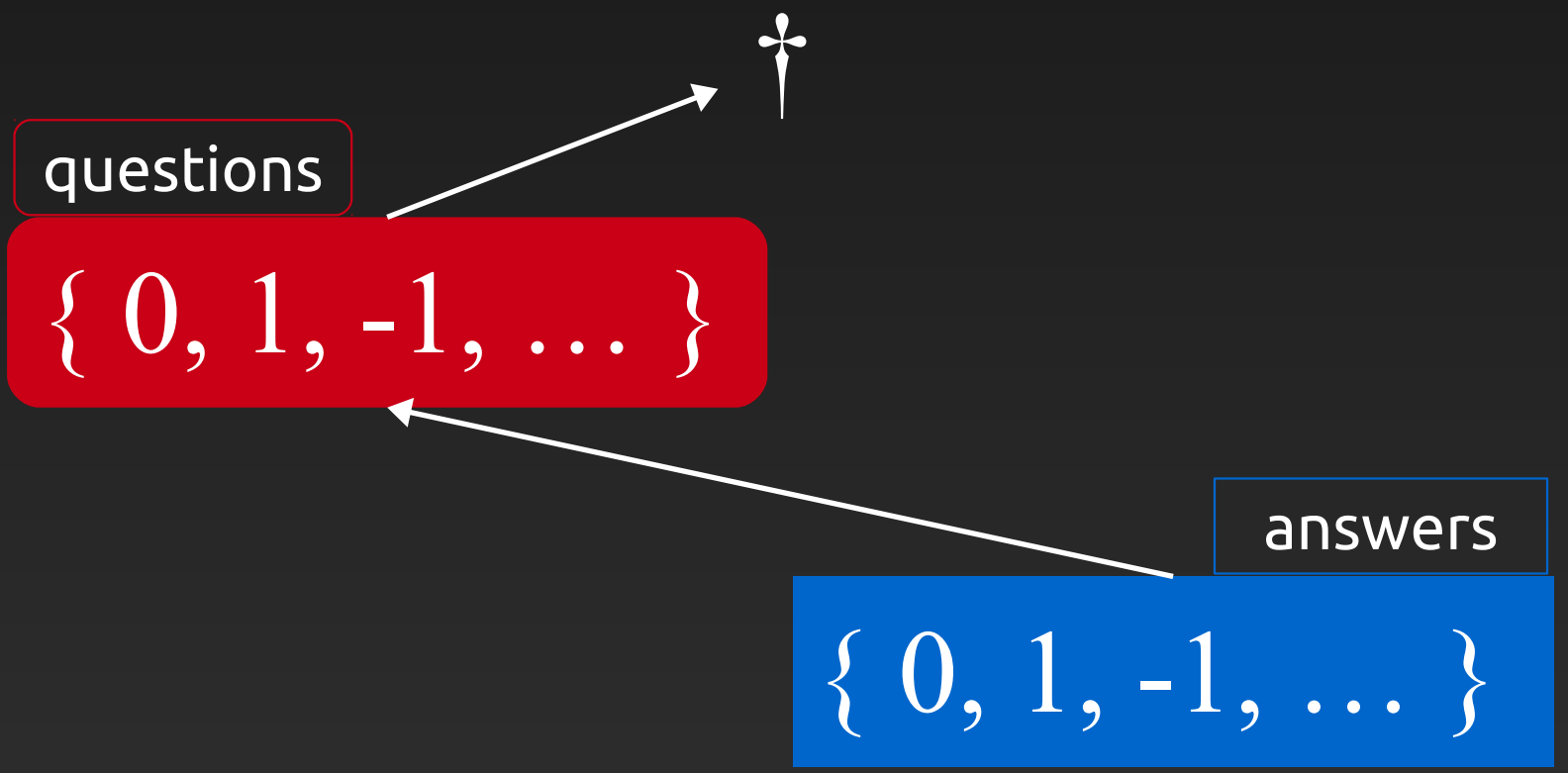
A higher-order arena

$$[\![\text{int} \to \text{int}]\!]$$



A higher-order arena

$$[\![\text{int} \to \text{int}]\!]$$

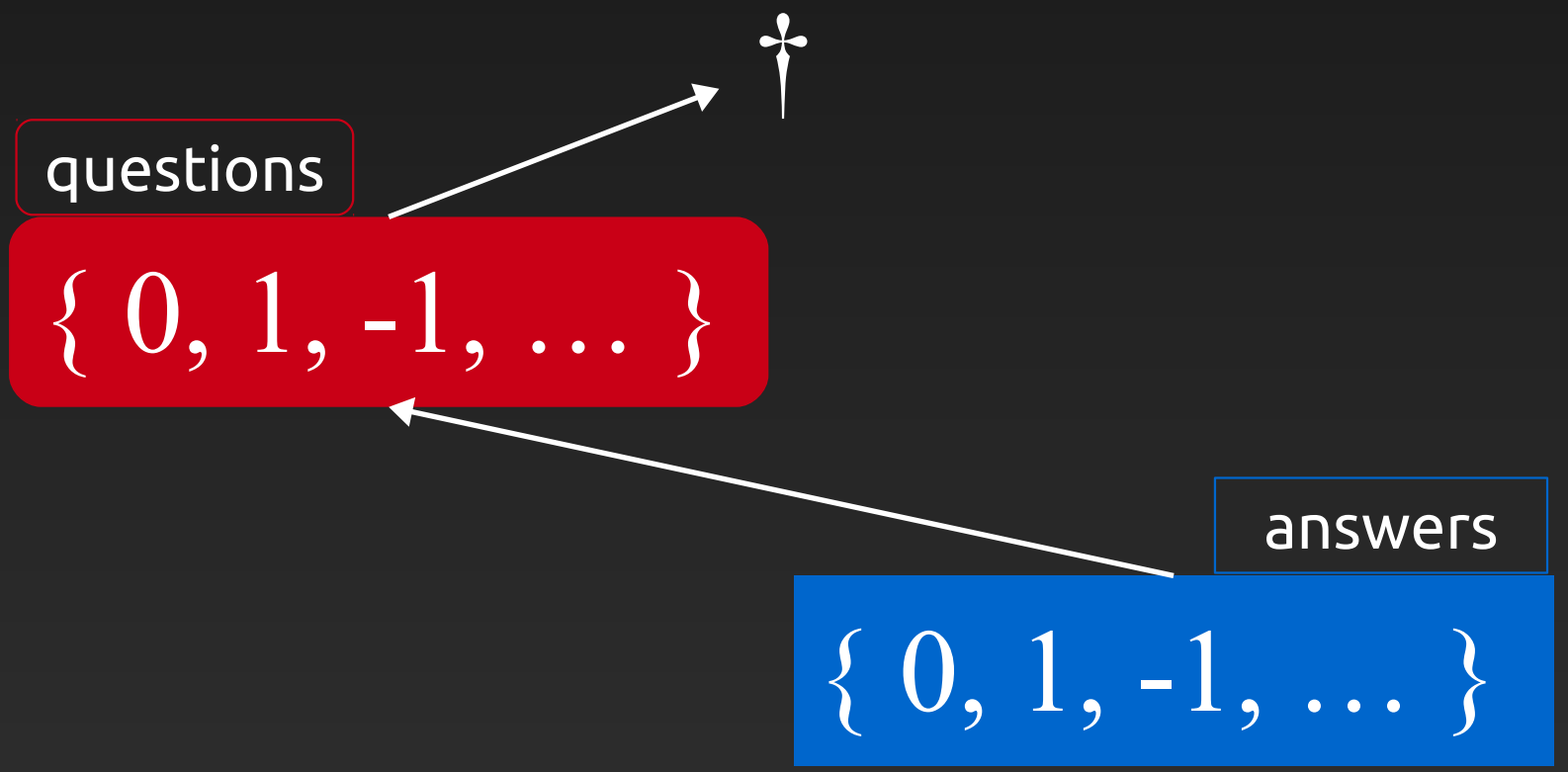


O : Opponent

P : Proponent

A higher-order arena

$$[\![\text{int} \to \text{int}]\!] = \mathbb{Z} \Rightarrow \mathbb{Z}$$

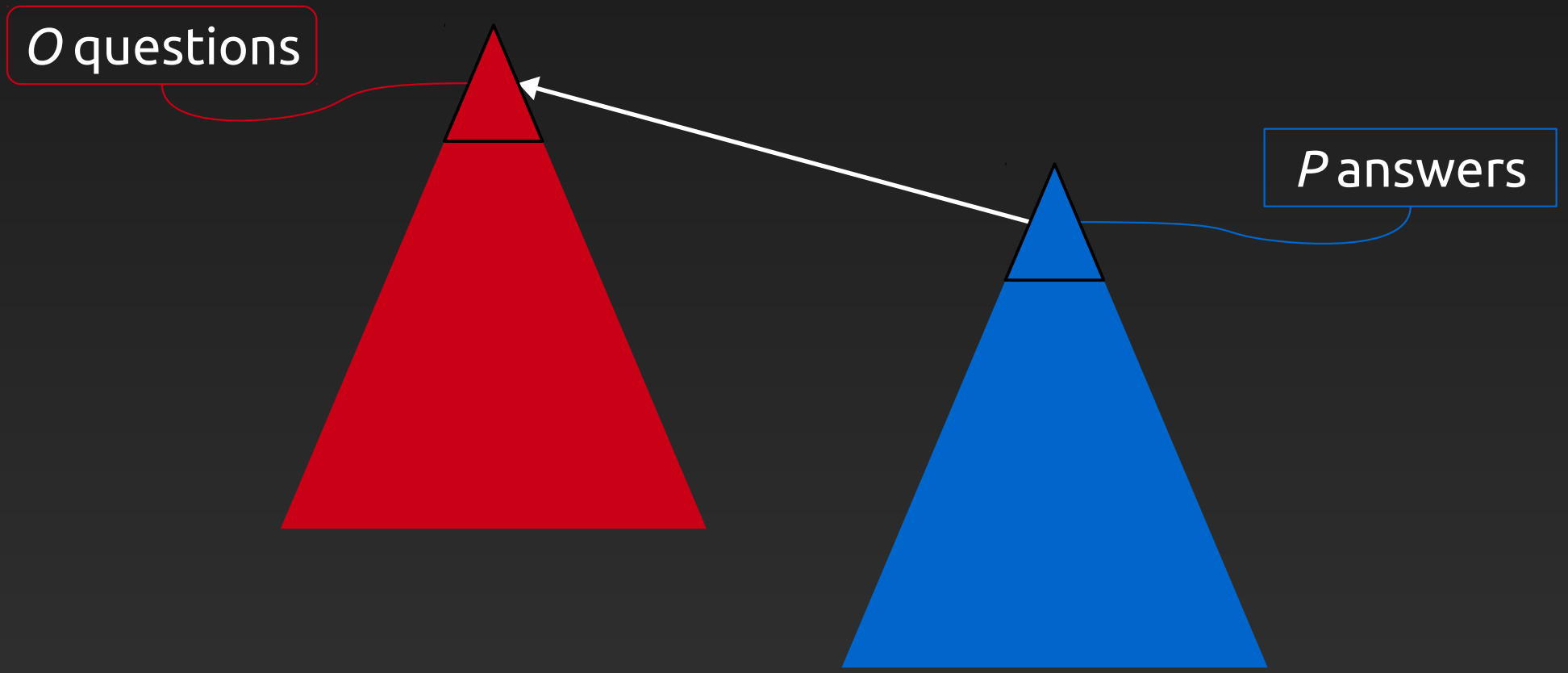


O : Opponent

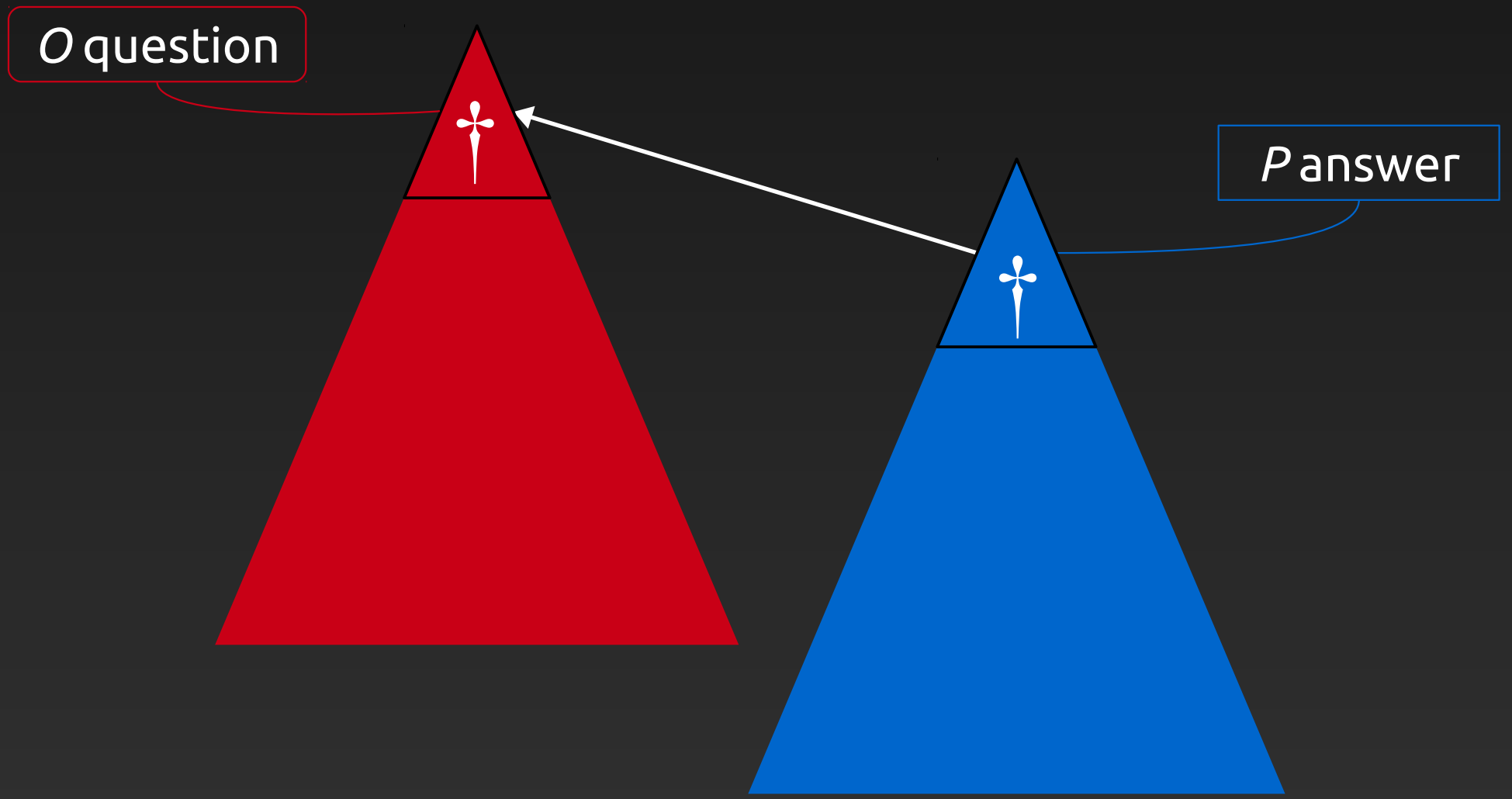
P : Proponent

A higher-order interface

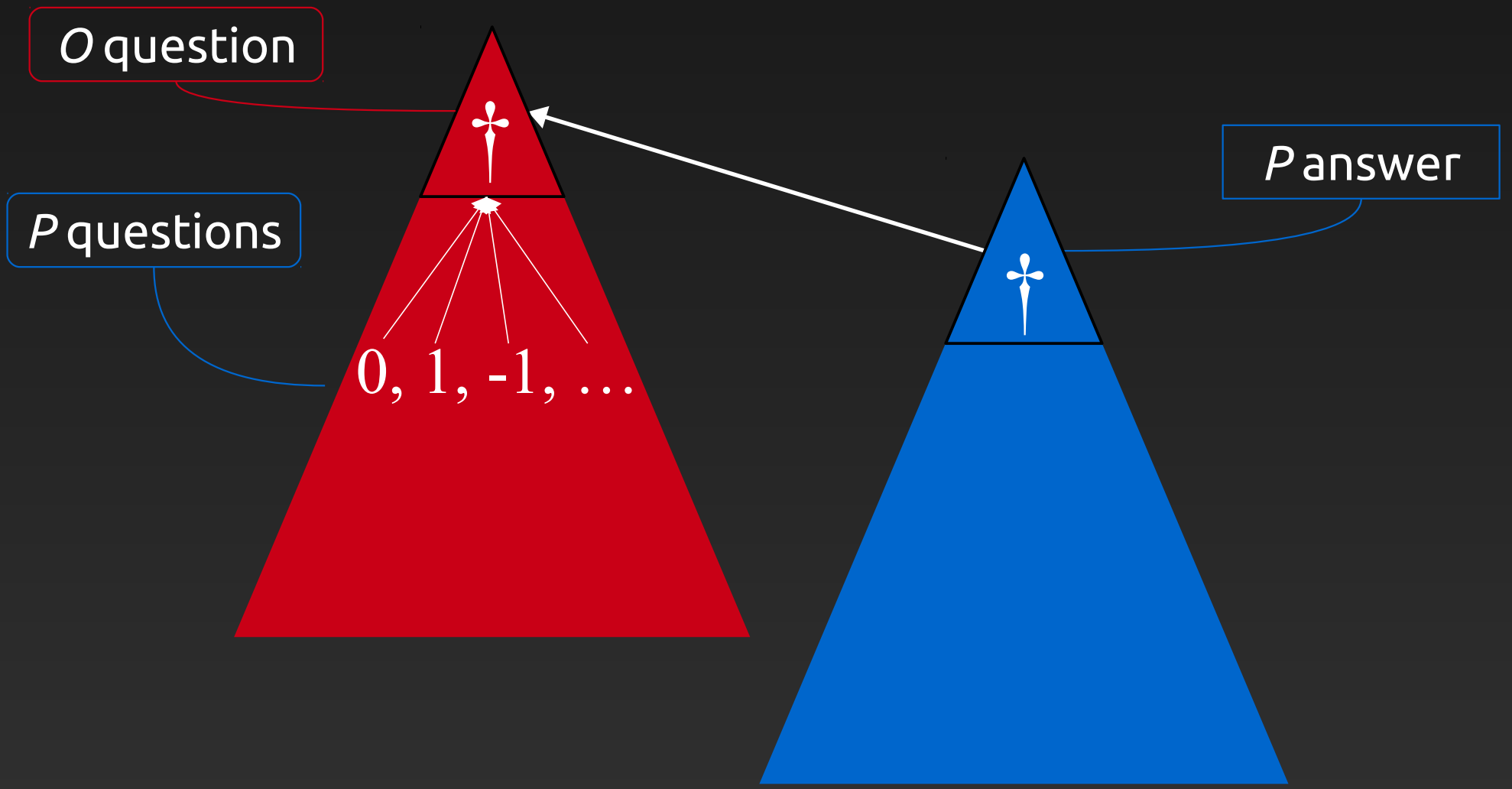
$$[\![T]\!] \longrightarrow [\![T']\!]$$



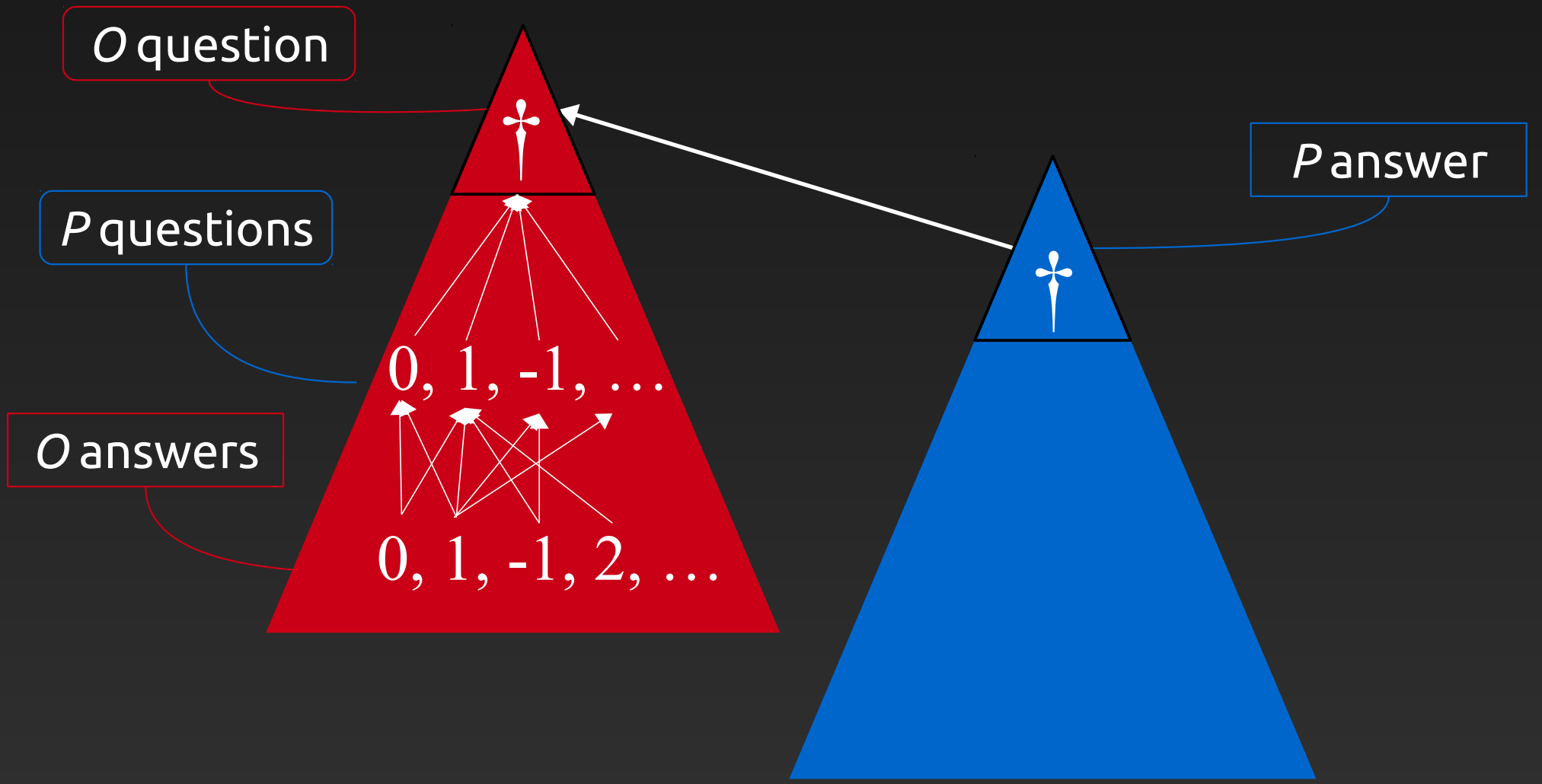
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



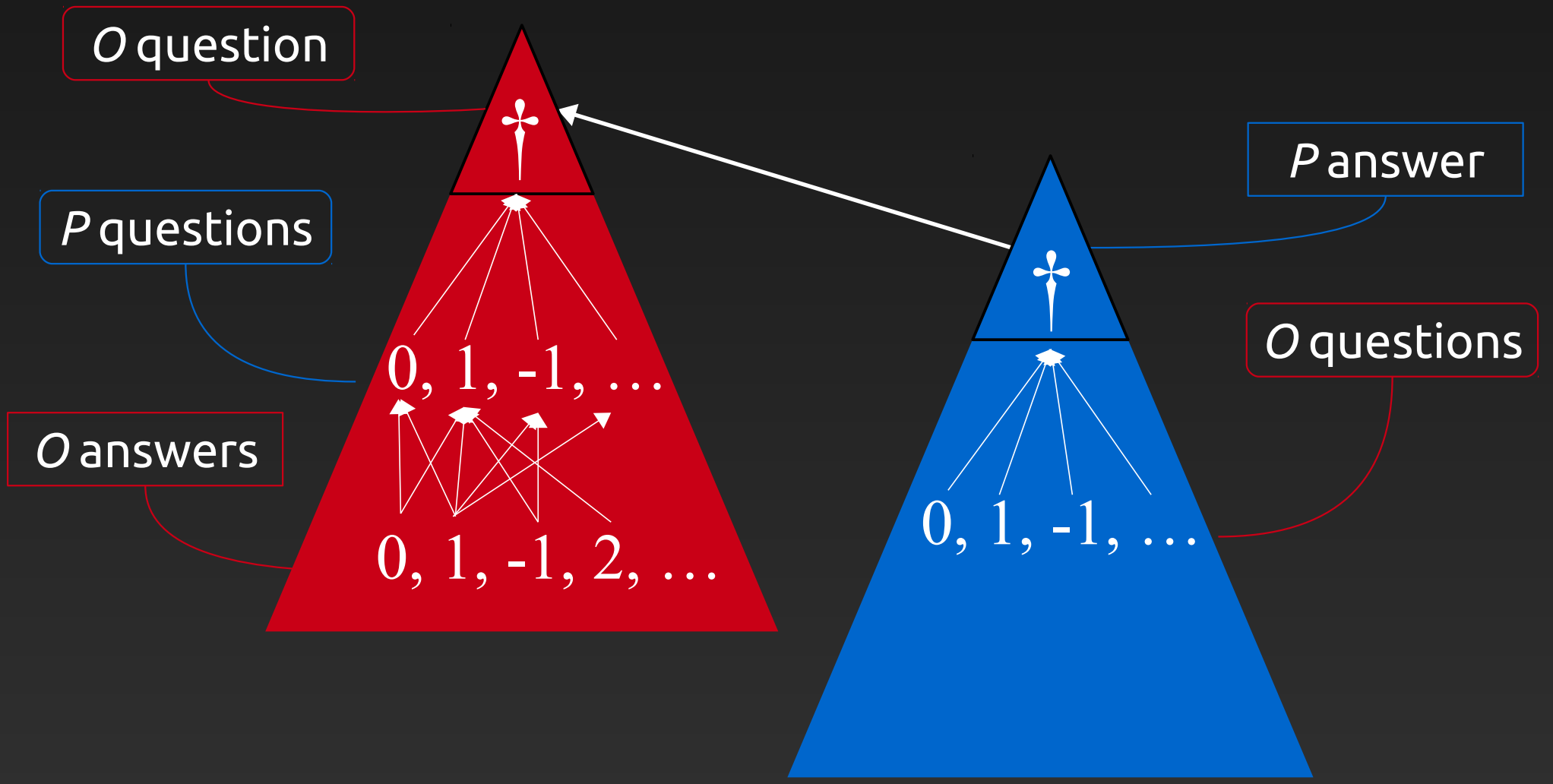
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



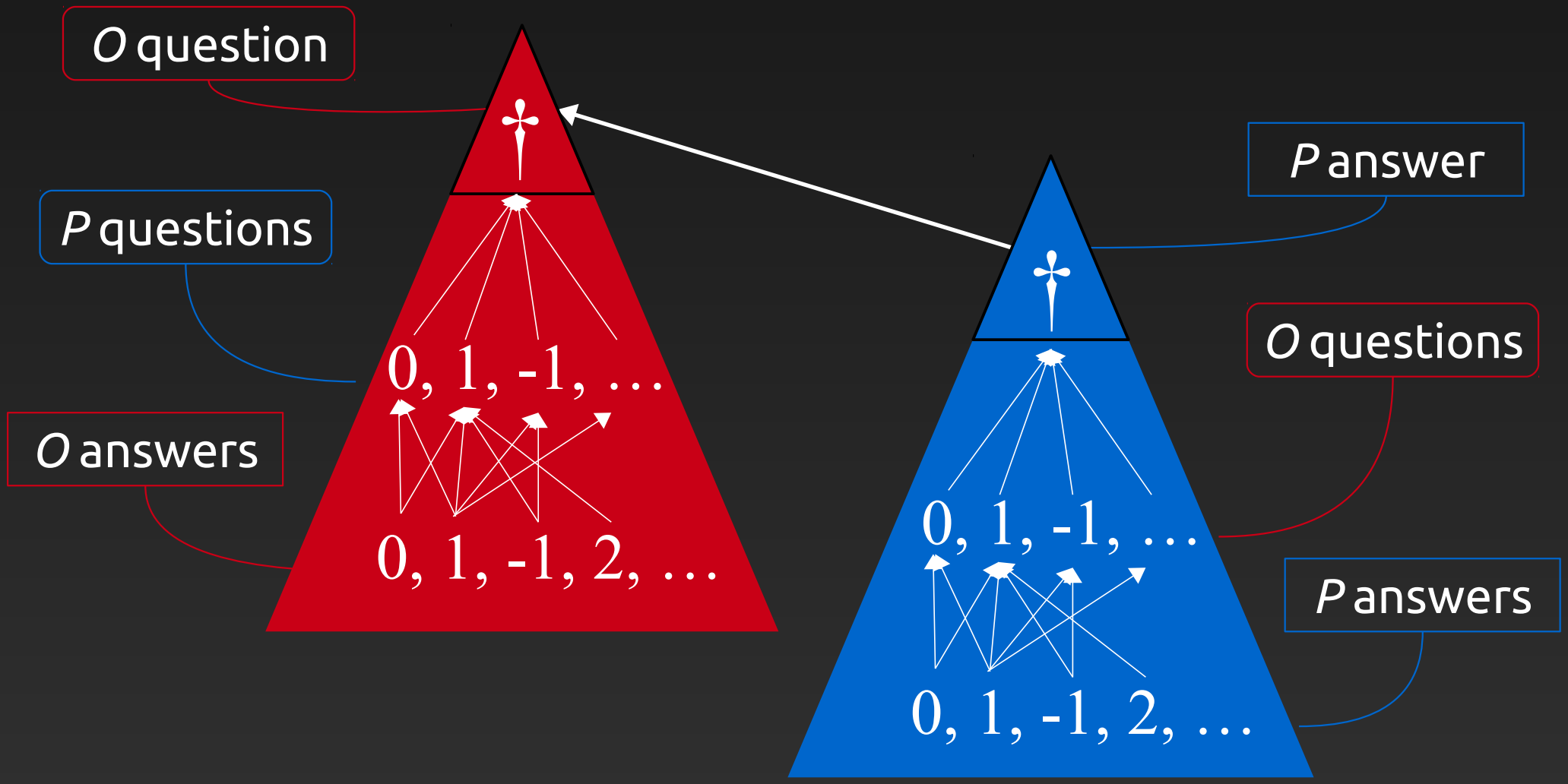
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



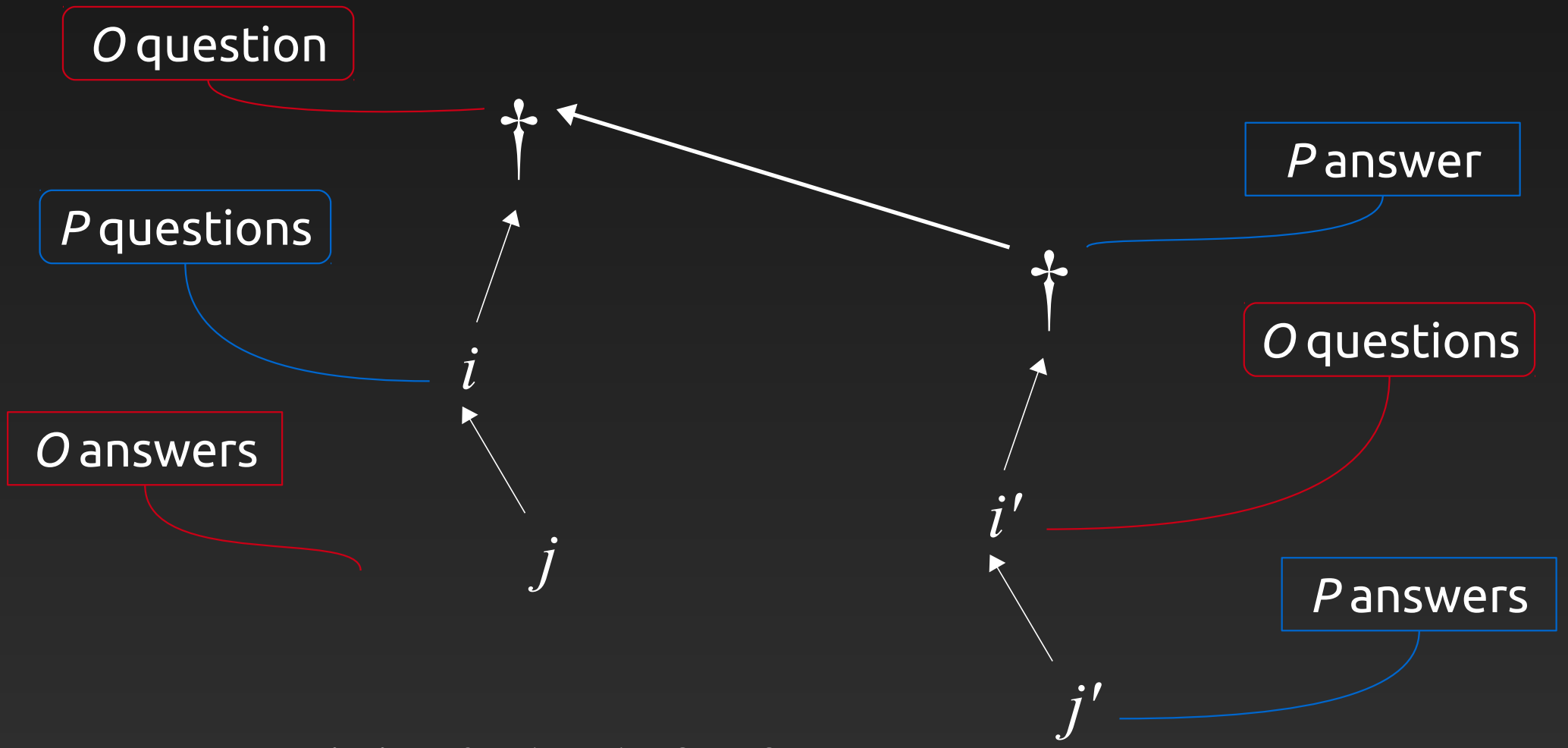
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

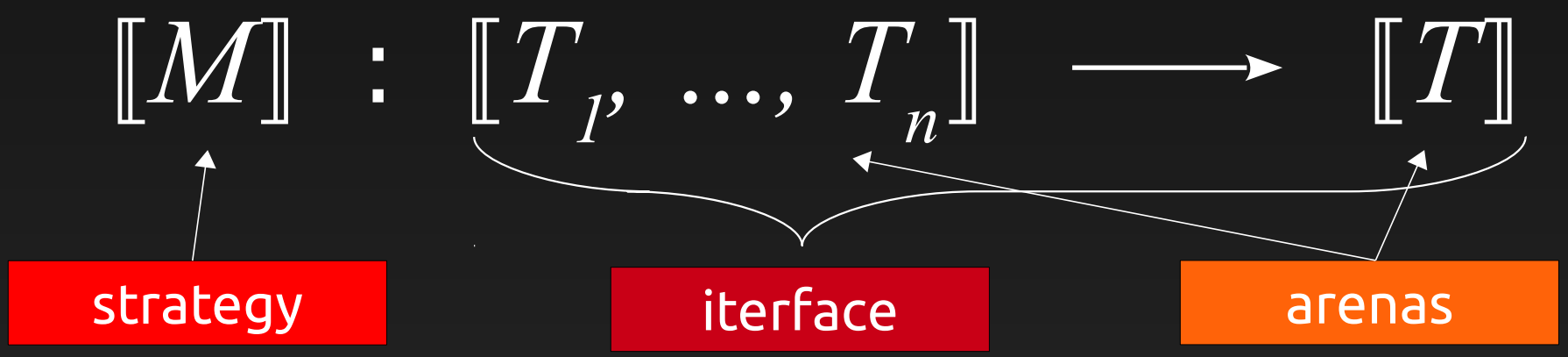


$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



$i, j, i', j' = 0, 1, -1, 2, -2, \ldots$

Strategies



Strategies are “instructions” for *P* on how to play on a given interface

- Formally, sets of even-length plays satisfying combinatorial conditions linked to language expressivity

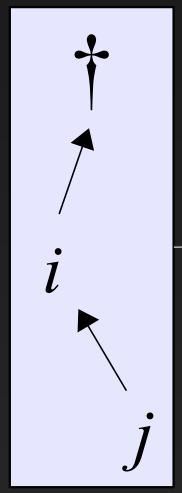
Example

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

$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

Example

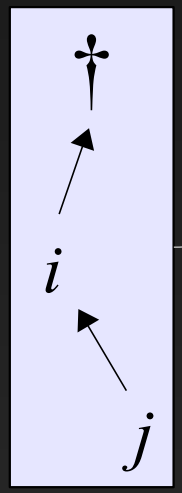
$$f : \text{int} \to \text{int} \;\vdash\; \lambda x.\, f\, x + 1 : \text{int} \to \text{int}$$



$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

Example

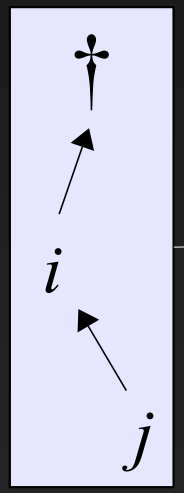
$$f : \text{int} \to \text{int} \;\vdash\; \lambda x.\, f\, x + 1 : \text{int} \to \text{int}$$

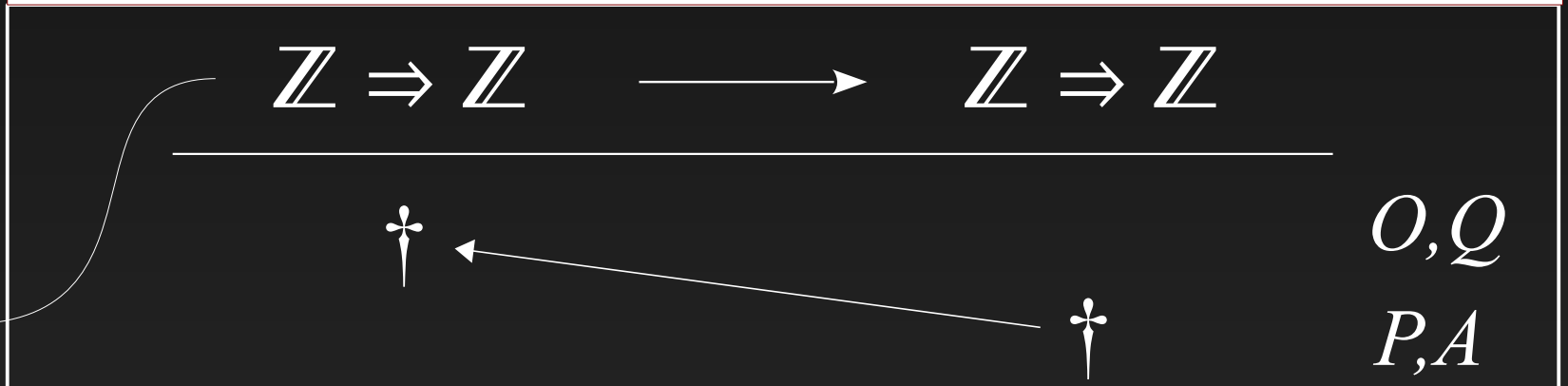


$$\frac{\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}}{\dagger} \; O,Q$$

Example

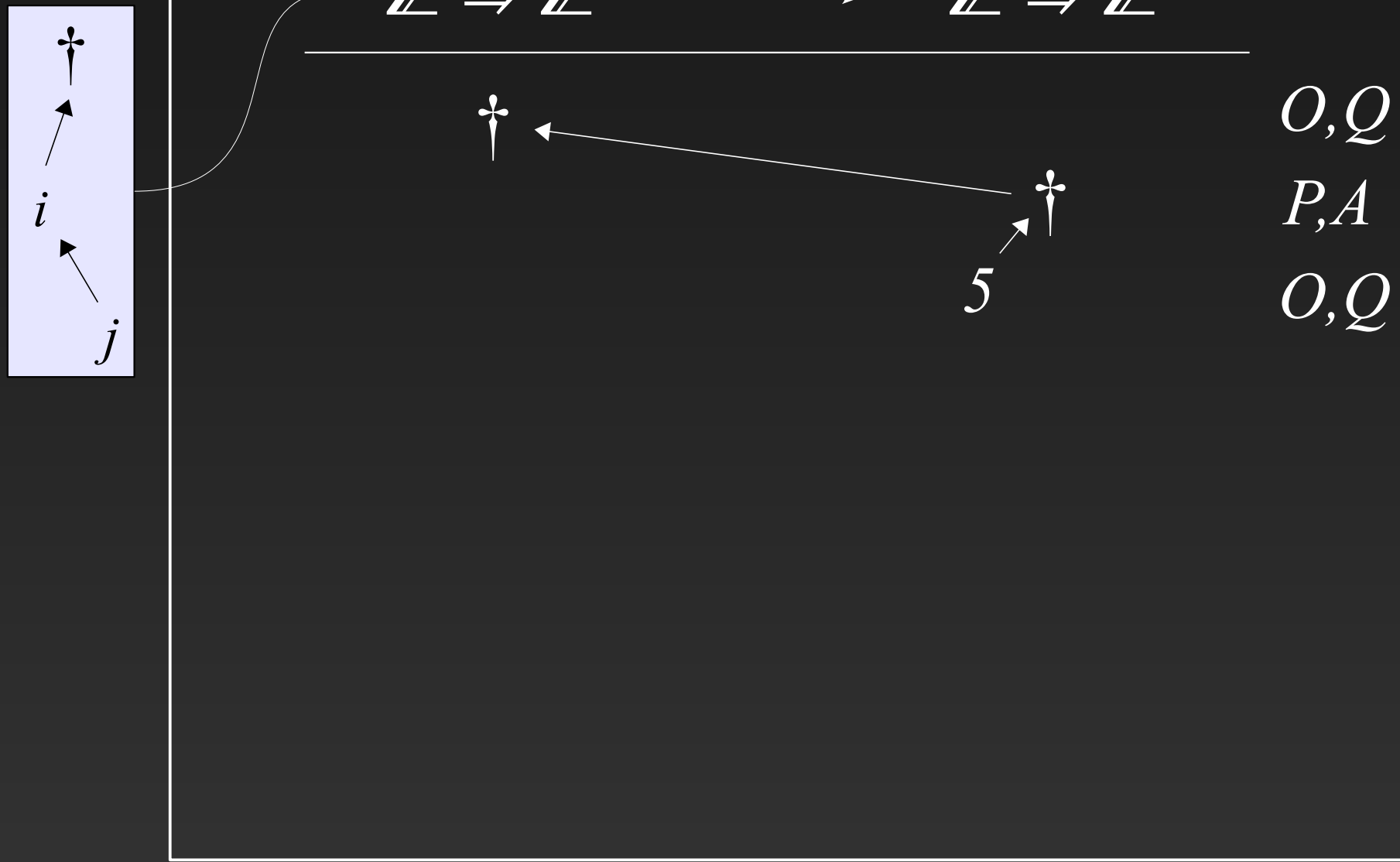
$$f : \mathrm{int} \to \mathrm{int} \;\vdash\; \lambda x.\, f\, x + 1 : \mathrm{int} \to \mathrm{int}$$





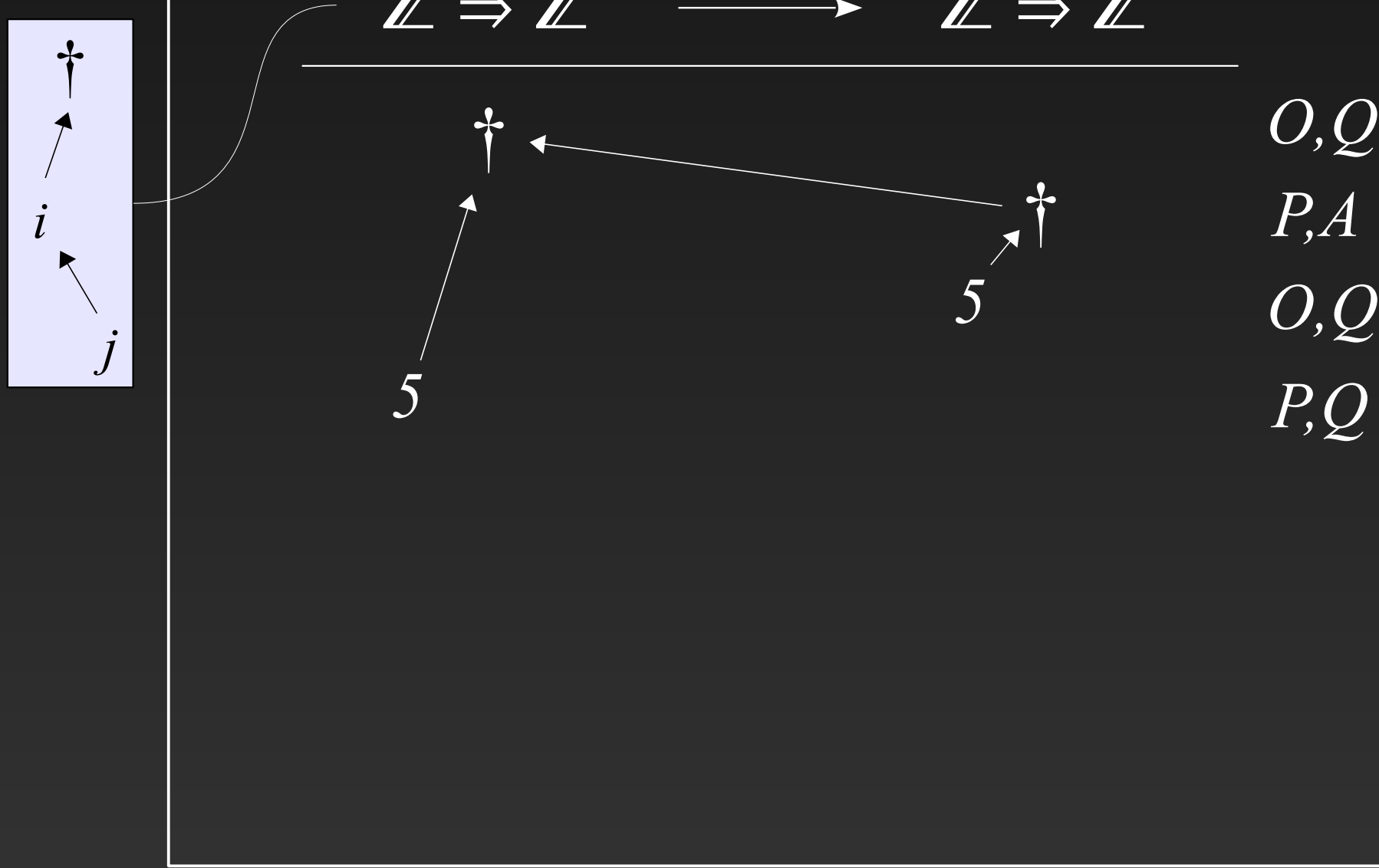
Example

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



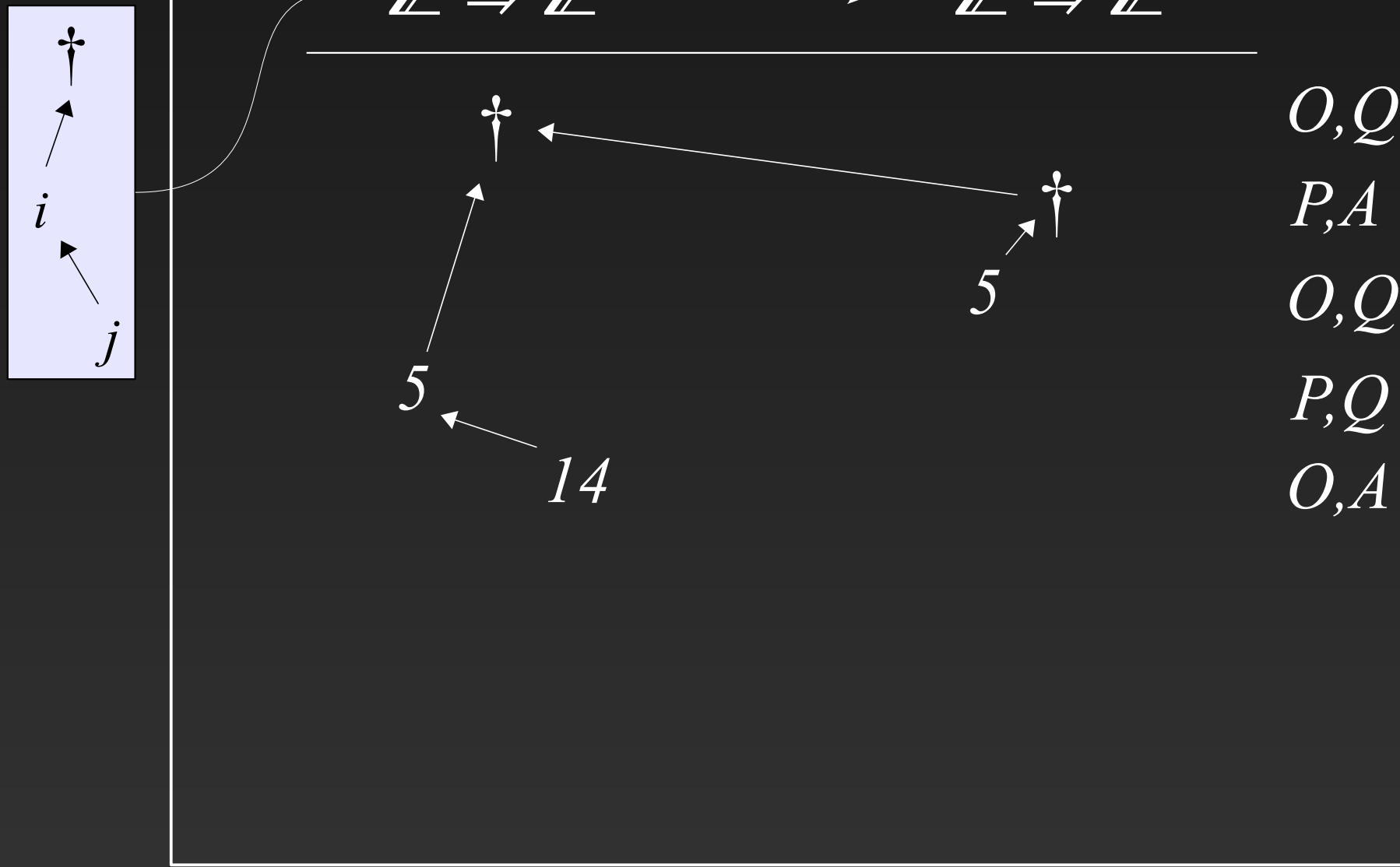
Example

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



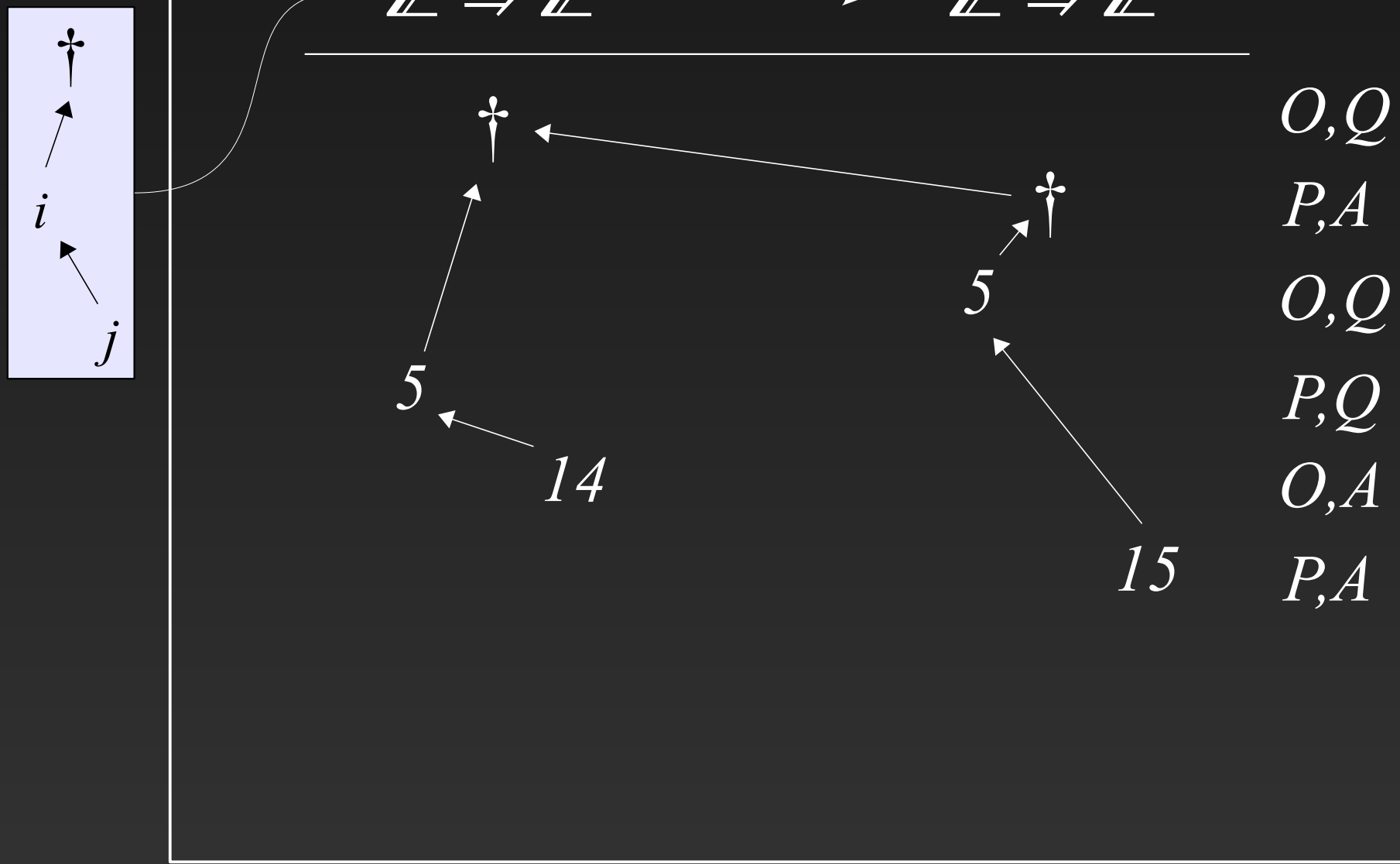
Example

$$f : \mathrm{int} \to \mathrm{int} \vdash \lambda x.\, f\, x + 1 : \mathrm{int} \to \mathrm{int}$$



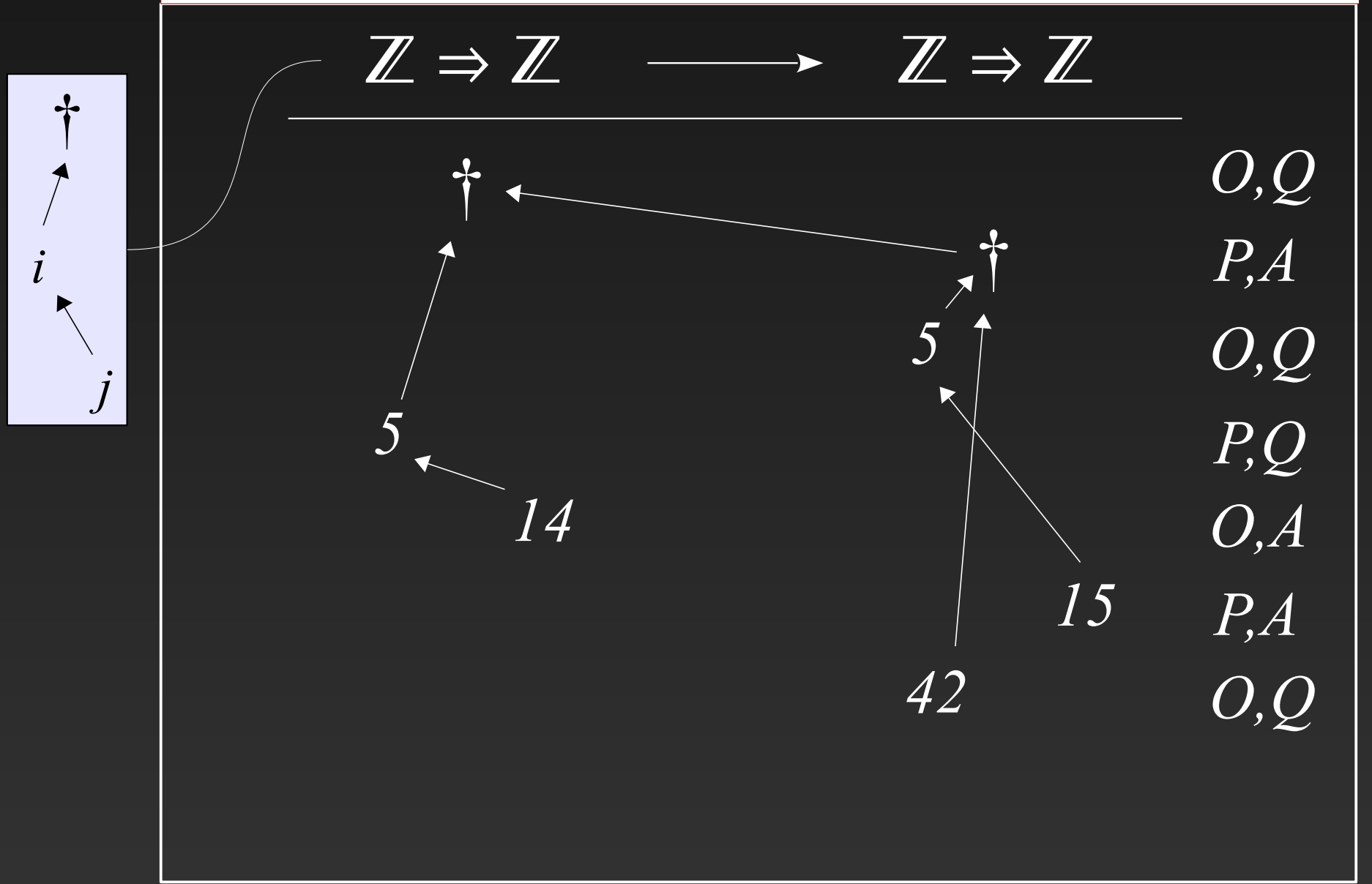
Example

$$f : \mathrm{int} \to \mathrm{int} \vdash \lambda x.\, f\, x + 1 : \mathrm{int} \to \mathrm{int}$$



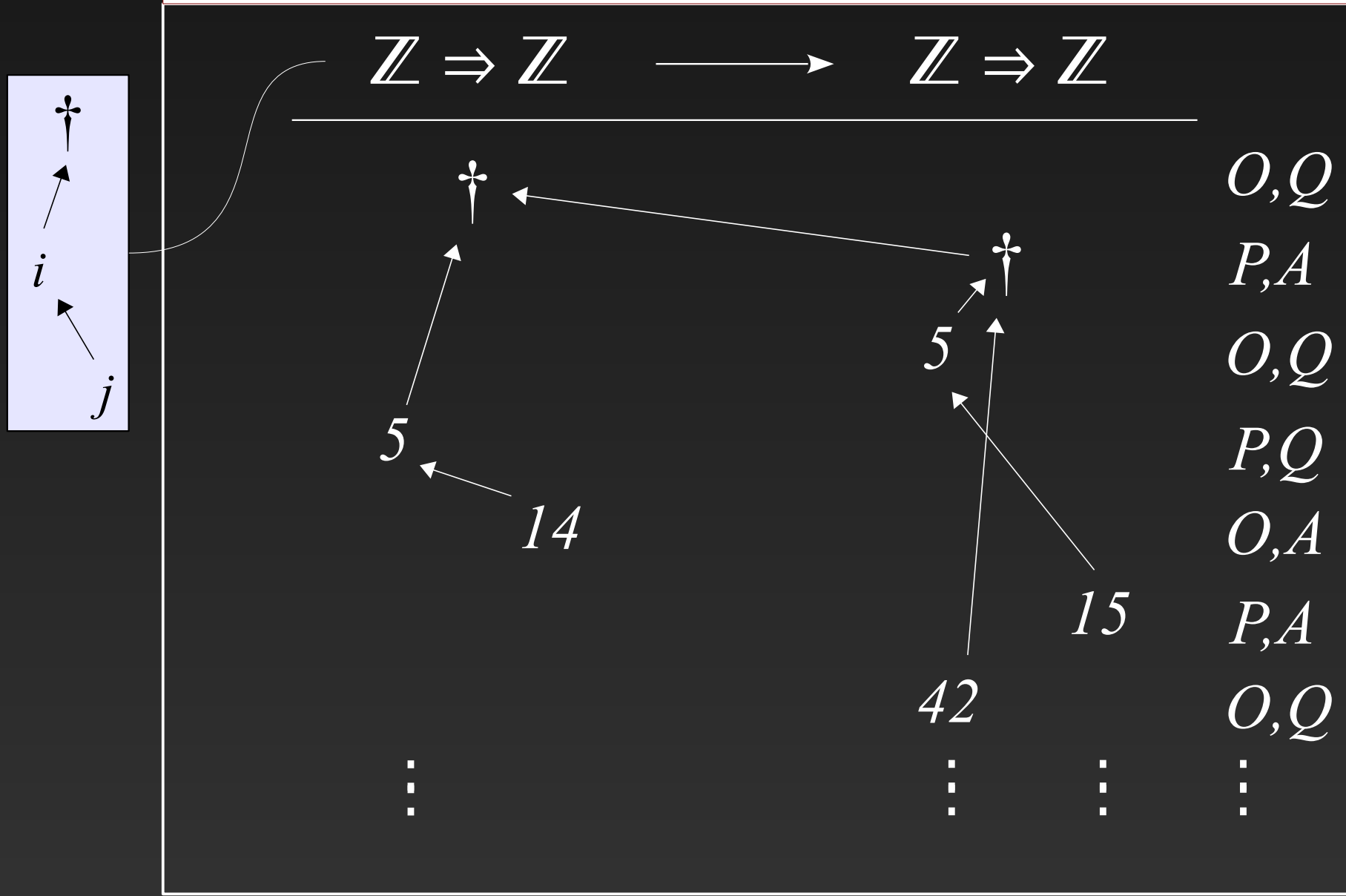
Example

$$f : \mathrm{int} \to \mathrm{int} \;\vdash\; \lambda x.\, f\, x + 1 : \mathrm{int} \to \mathrm{int}$$



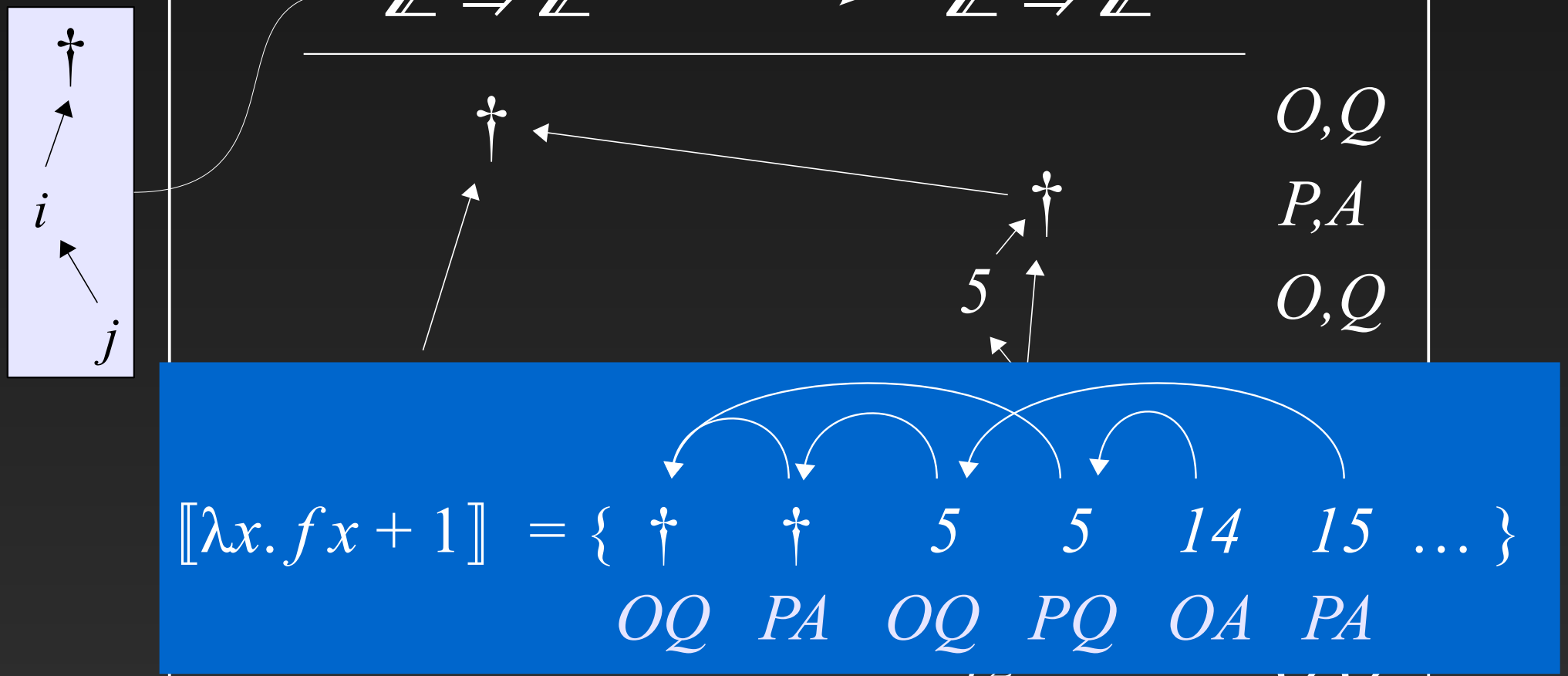
Example

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



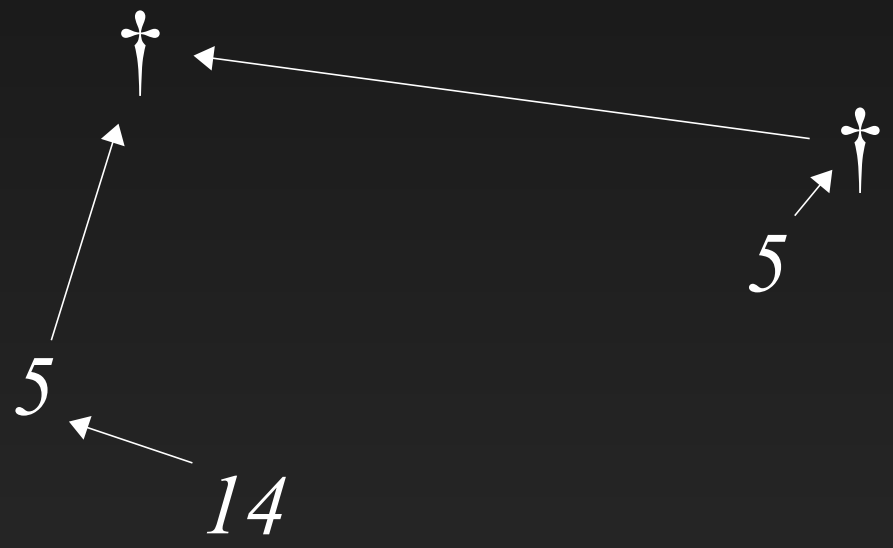
Example

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



Another example

$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



O,Q

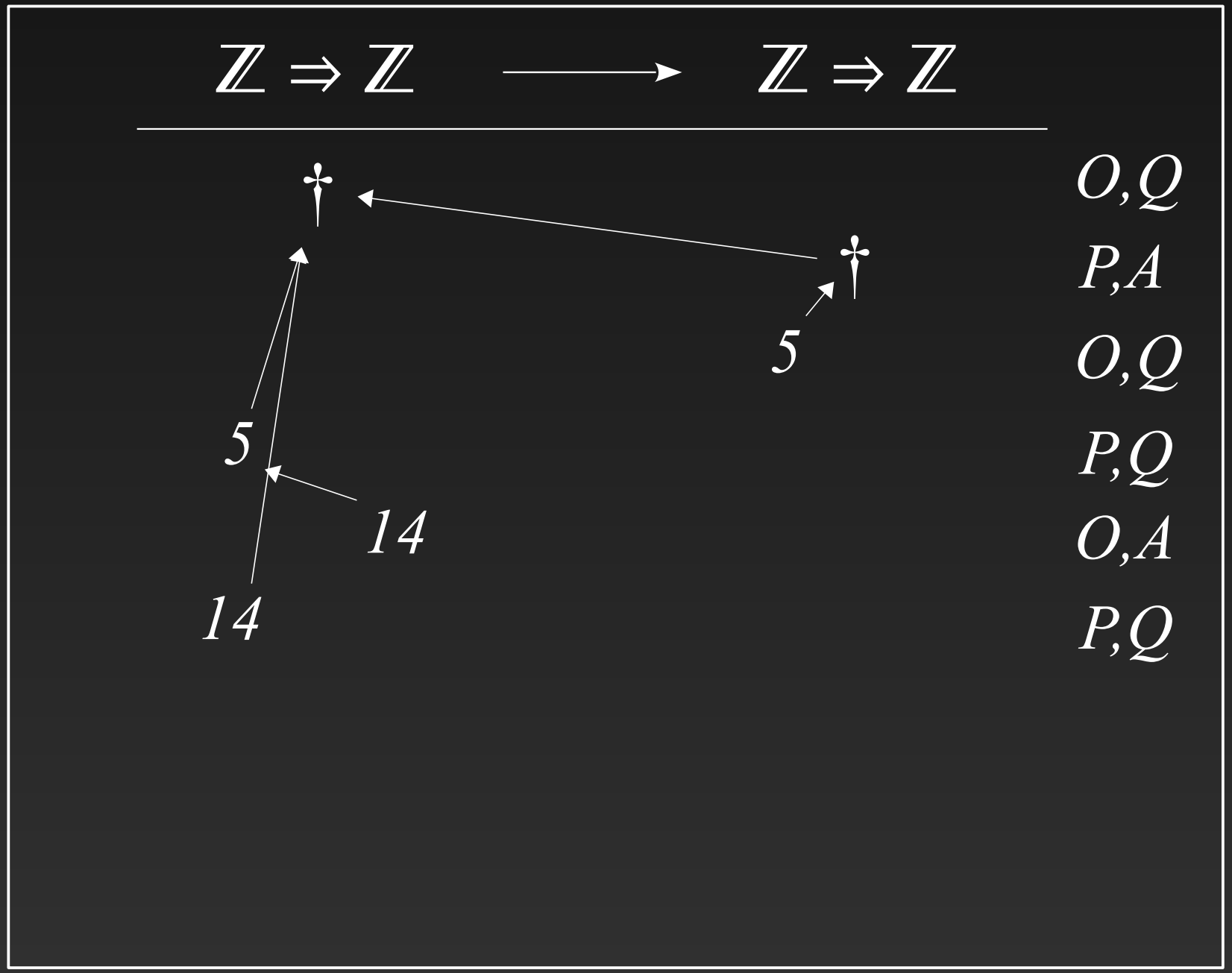
P,A

O,Q

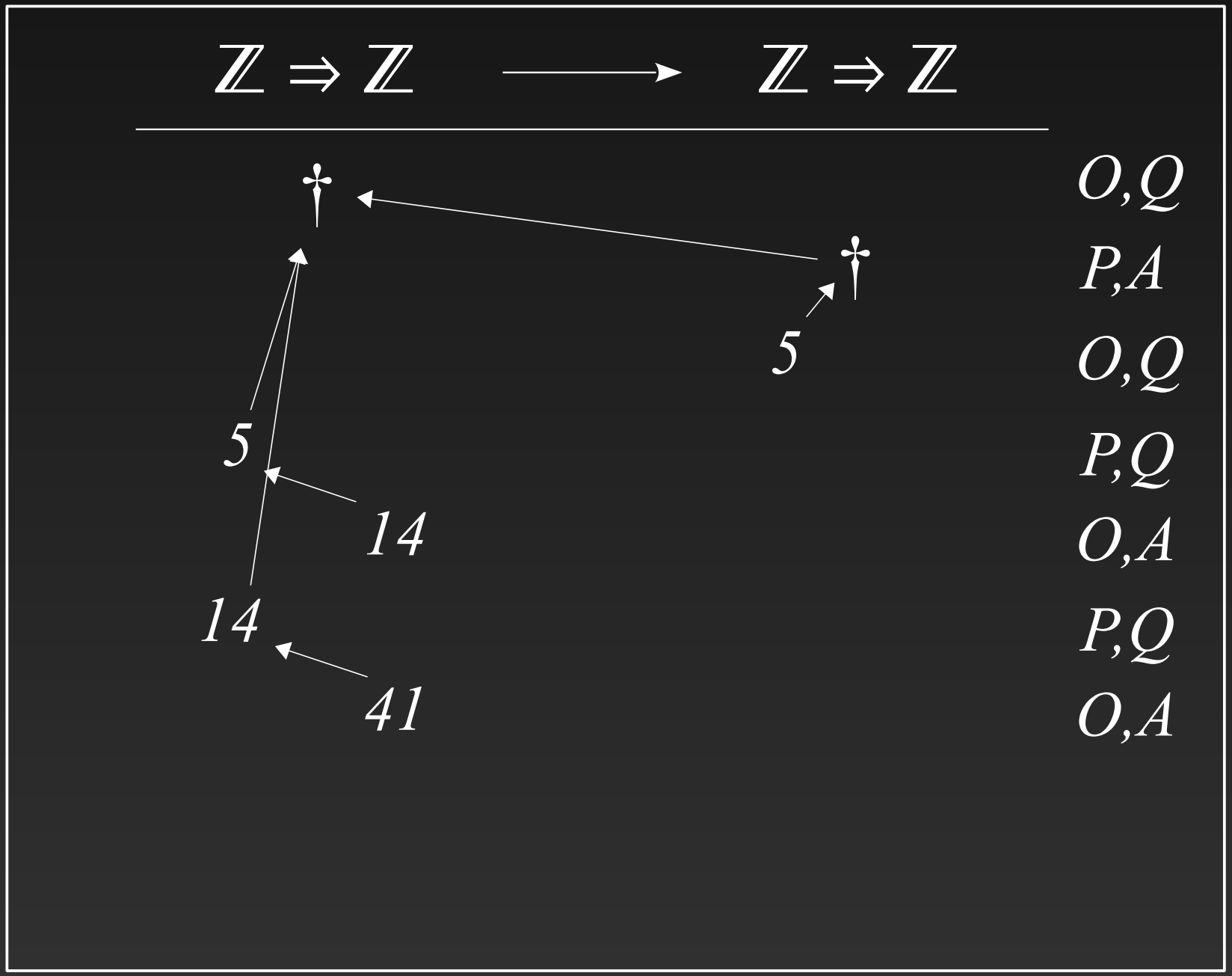
P,Q

O,A

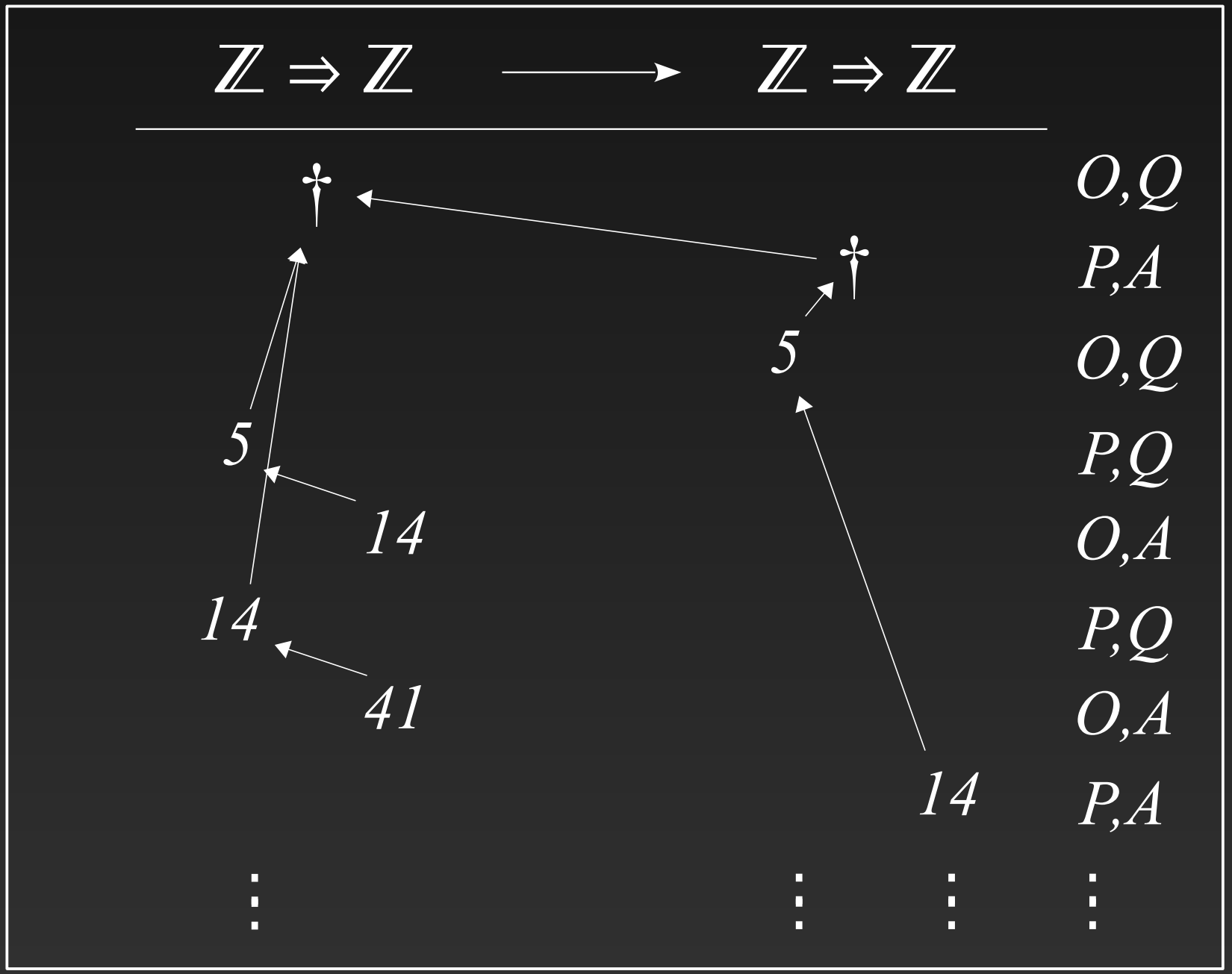
Another example



Another example

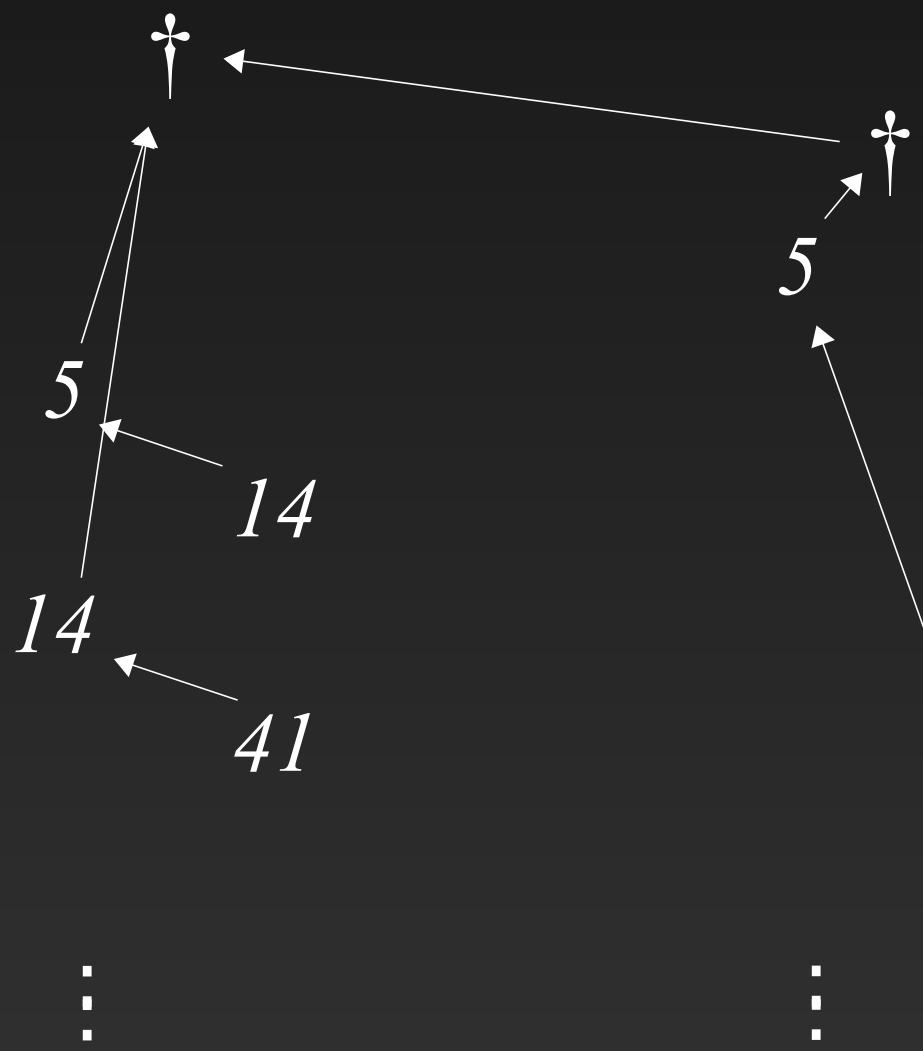


Another example



$$f : \mathrm{int} \to \mathrm{int} \;\vdash\; \lambda x.\, f\,(f\,(x)) - 27 : \mathrm{int} \to \mathrm{int}$$

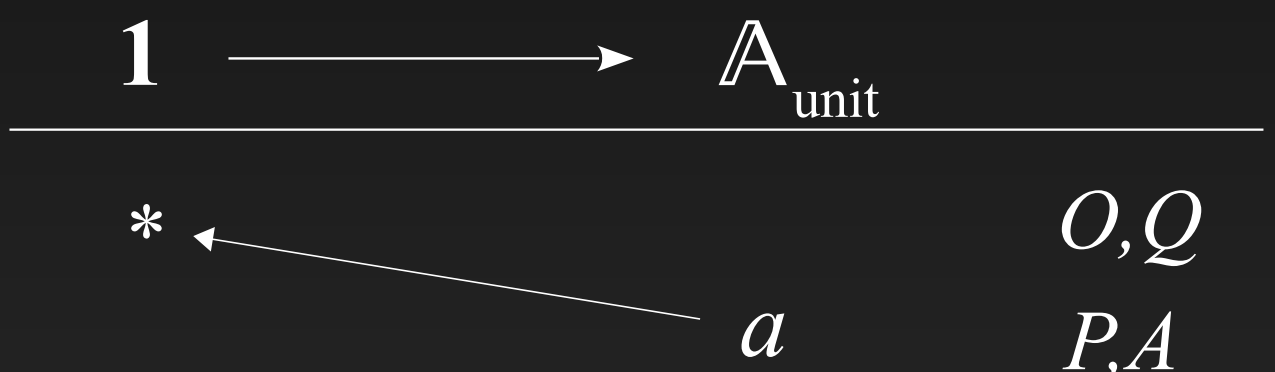
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



O,Q
P,A
O,Q
P,Q
O,A
P,Q
O,A
P,A
⋮ ⋮ ⋮ ⋮

Nominal games

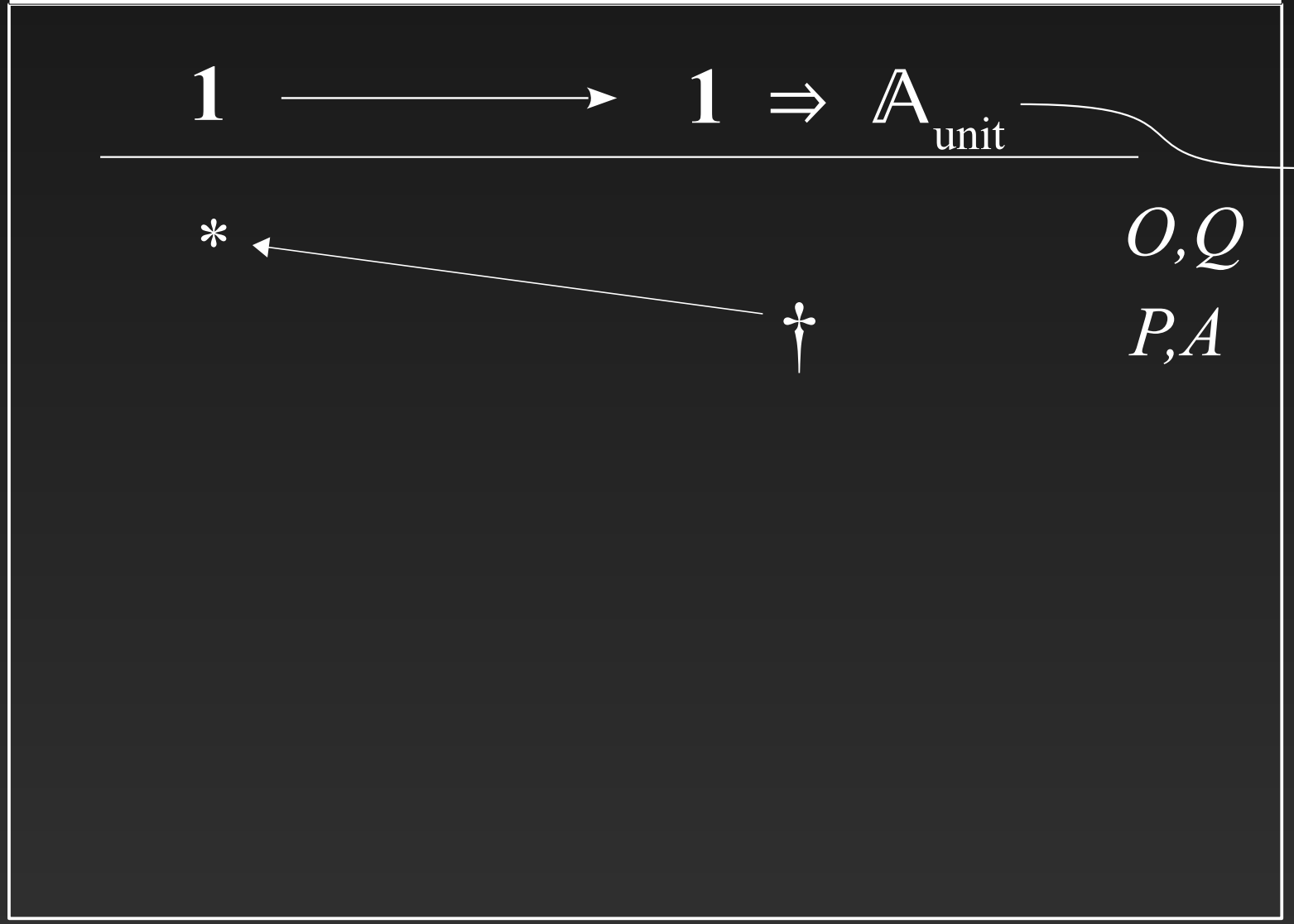
$\vdash \mathrm{ref}() : \mathrm{ref\ unit}$



$a \in \mathcal{N}_{\mathrm{unit}}$

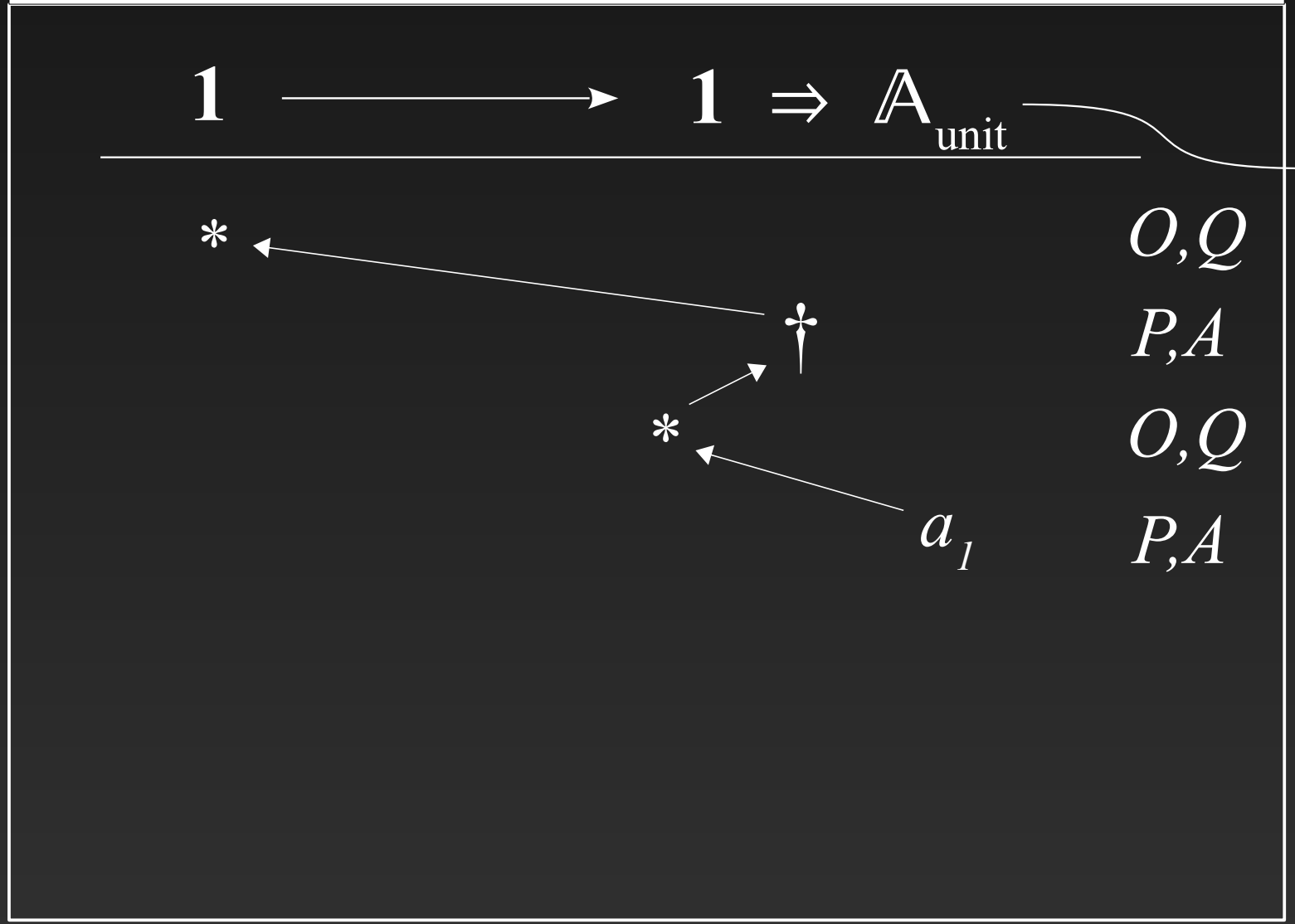
Nominal games (2)

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



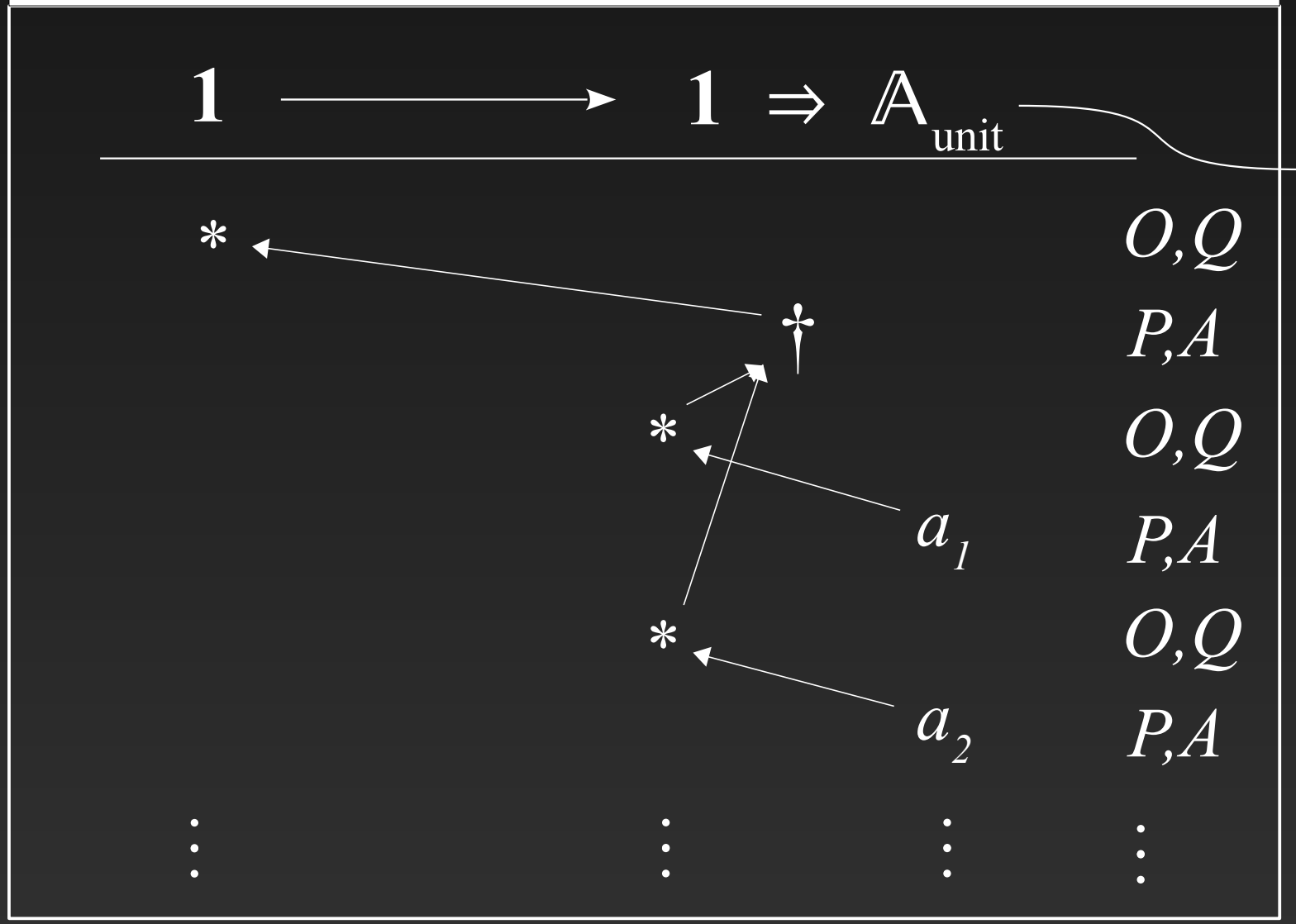
Nominal games (2)

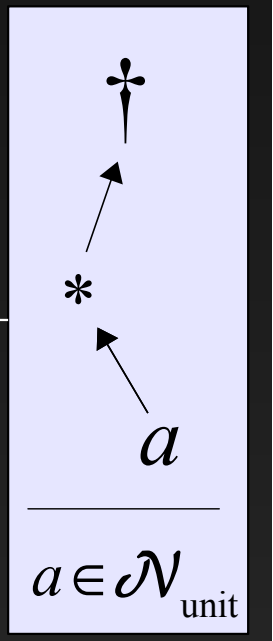
$\vdash \lambda x.\ \mathrm{ref}() : \mathrm{unit} \to \mathrm{ref\ unit}$



Nominal games (2)

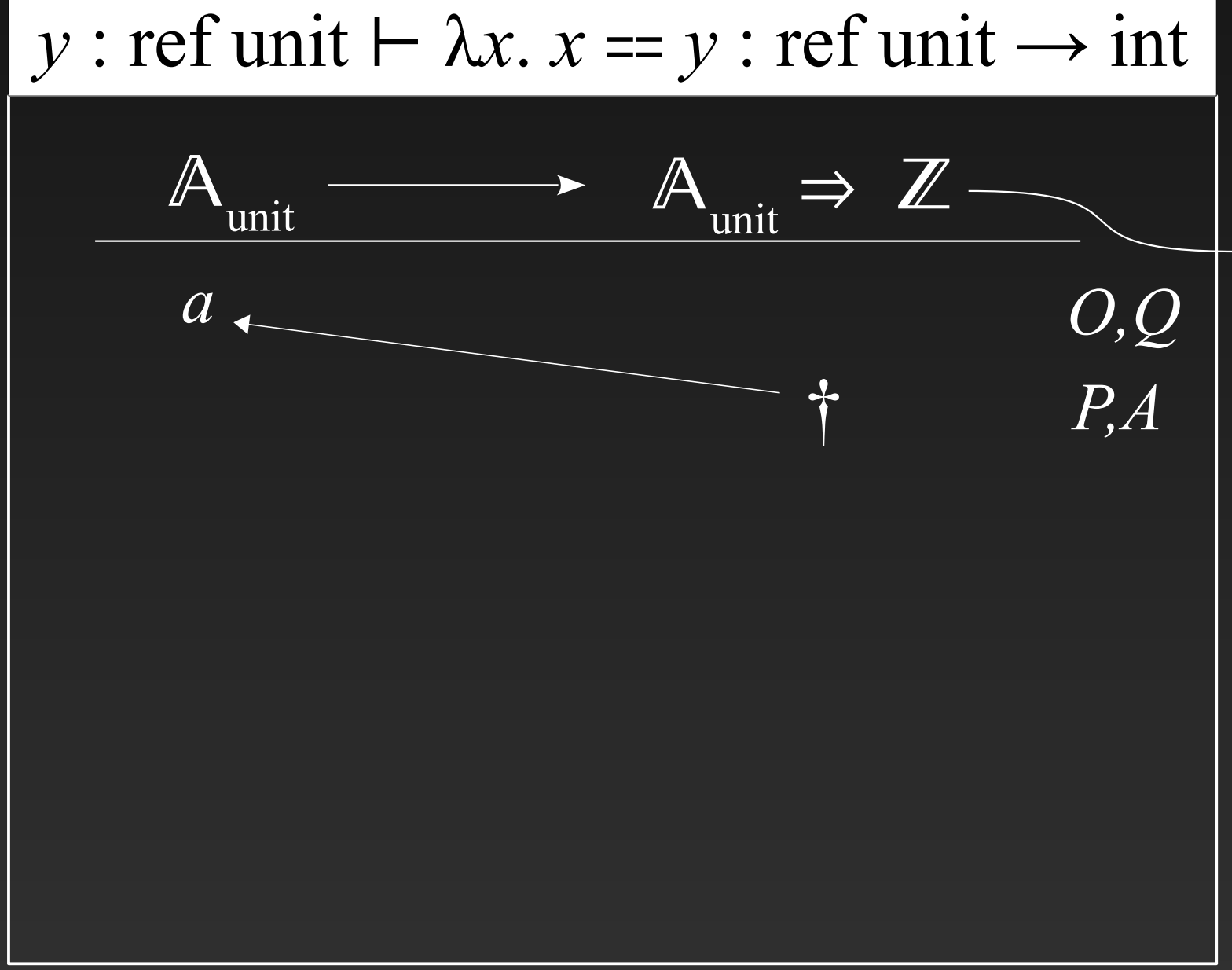
$\vdash \lambda x.\ \mathrm{ref}() : \mathrm{unit} \to \mathrm{ref\ unit}$





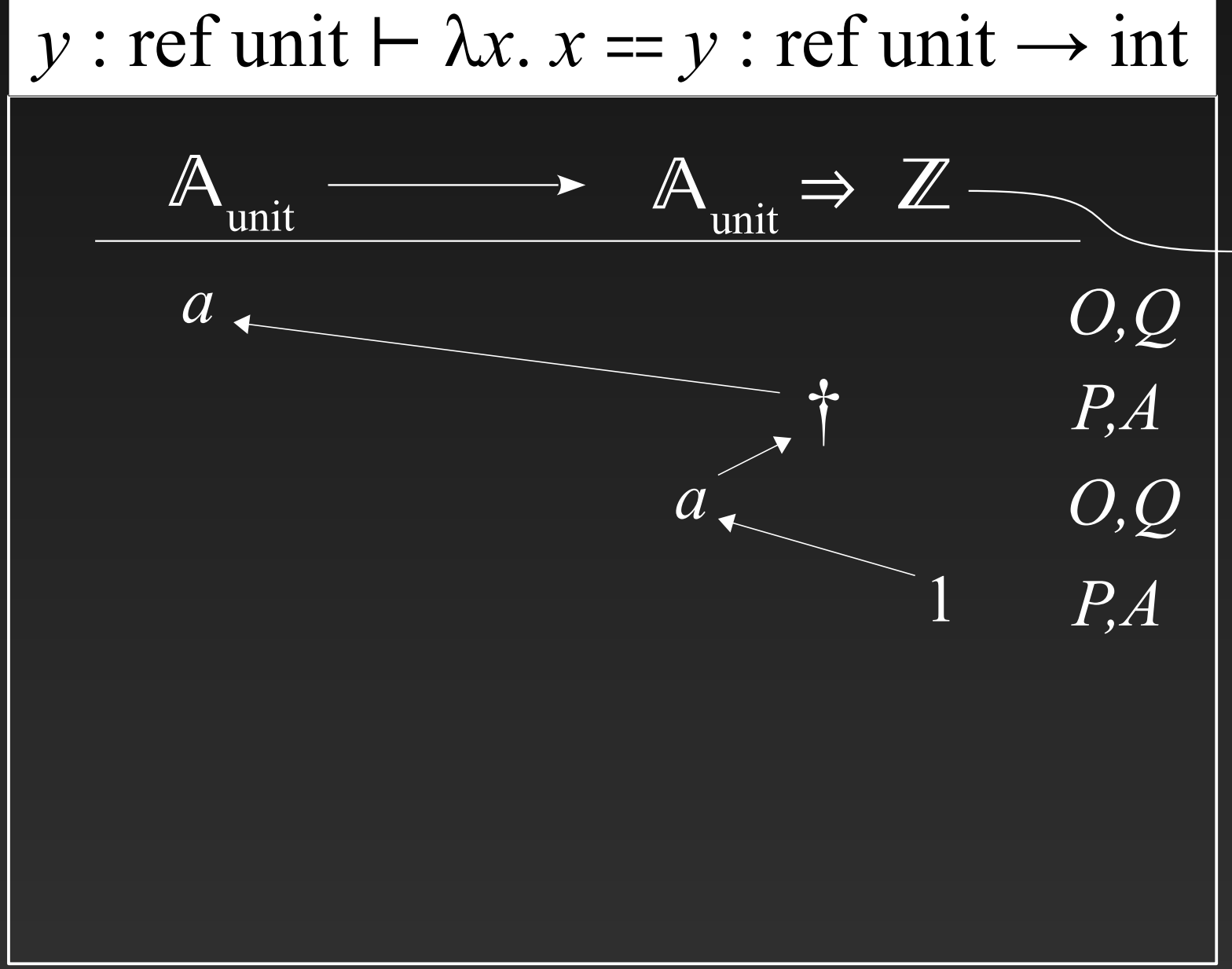
Nominal games (3)

$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \longrightarrow \mathrm{int}$



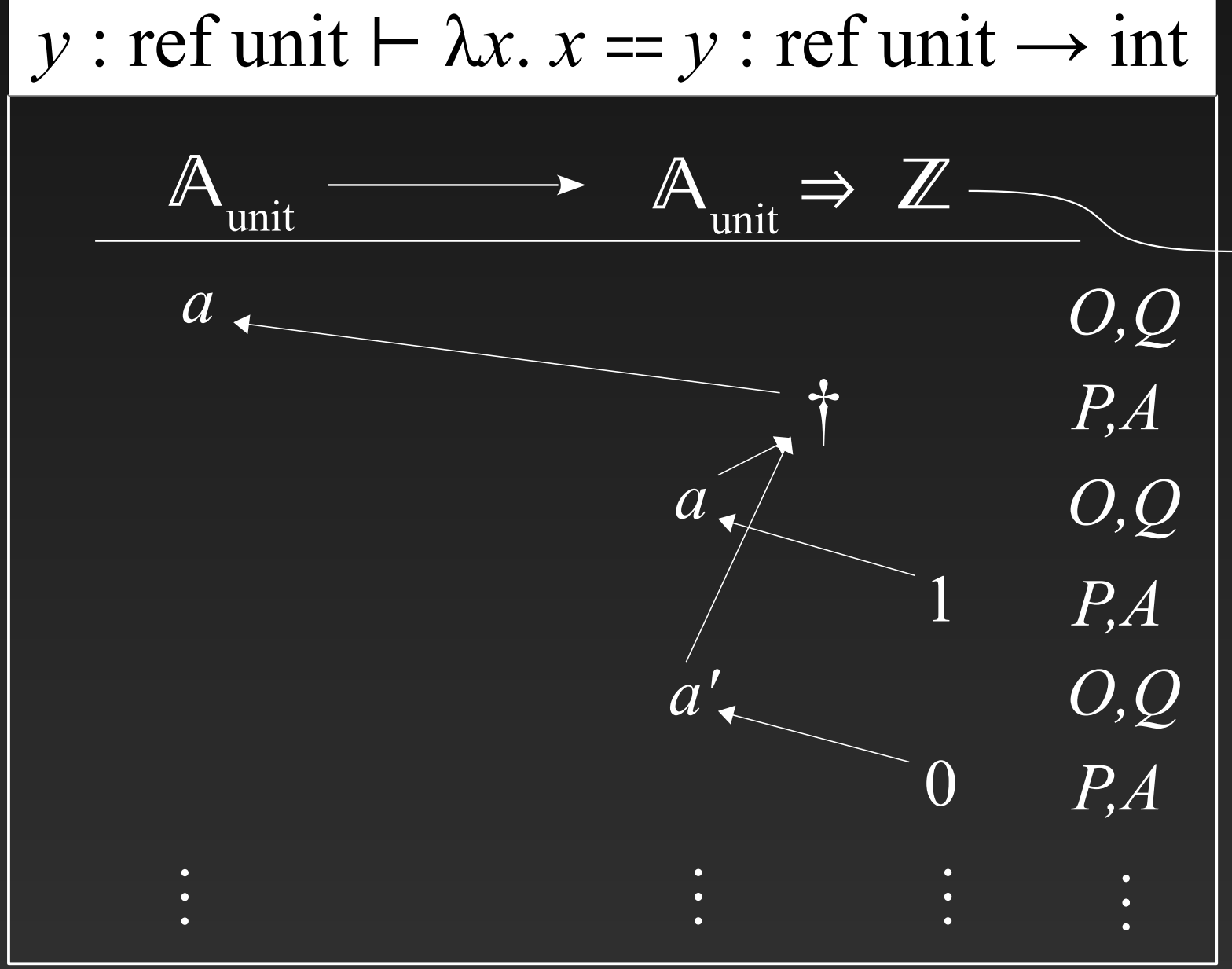
Nominal games (3)

$$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \longrightarrow \mathrm{int}$$

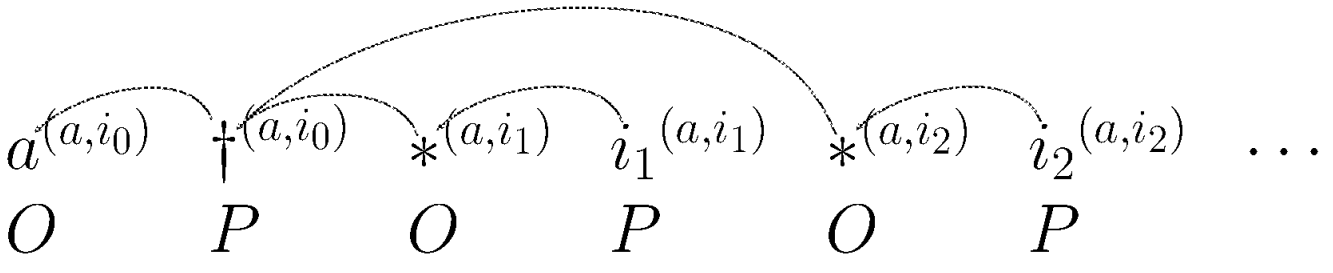


Nominal games (3)

$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \longrightarrow \mathrm{int}$

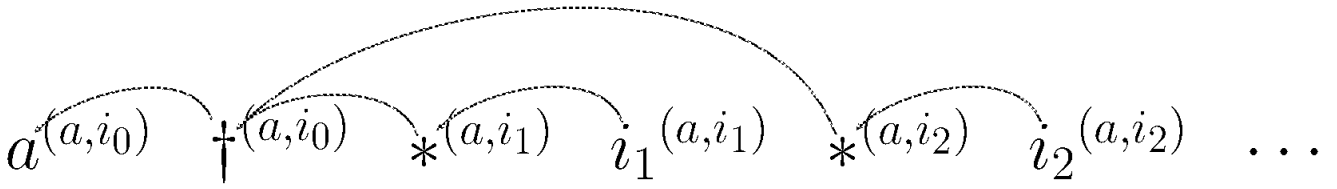


Nominal games: explicit store

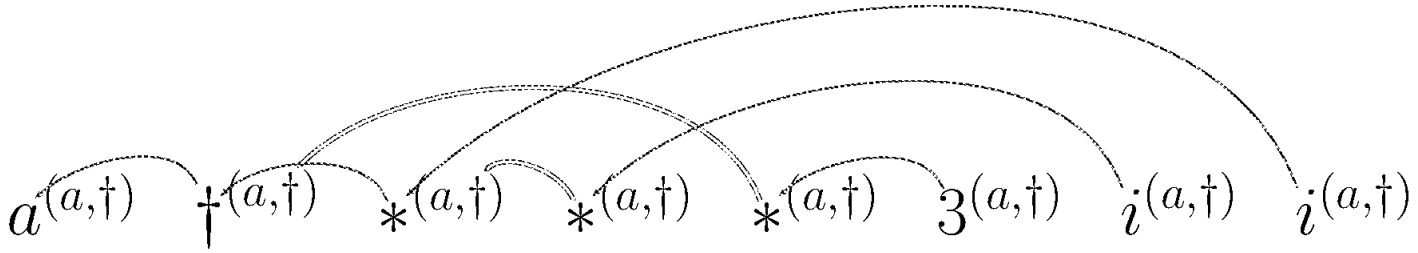


$$= \llbracket x : \mathsf{int\ ref} \vdash \lambda y.\, !x \ : \mathsf{unit} \to \mathsf{int} \rrbracket : \mathbb{A}_{\mathsf{int}} \to (1 \Rightarrow \mathbb{Z})$$

Nominal games: explicit store



$= \ [\![x : \mathsf{int\ ref} \vdash \lambda y.\, !x \ : \mathsf{unit} \to \mathsf{int}]\!] : \mathbb{A}_{\mathsf{int}} \to (1 \Rightarrow \mathbb{Z})$



$= \ [\![x : (\mathsf{unit} \to \mathsf{int})\ \mathsf{ref} \vdash x := \lambda y.3;\ \lambda y.(!x)()]\!] : \mathbb{A}_{\mathsf{unit}\to\mathsf{int}} \to (1 \Rightarrow \mathbb{Z})$

In LaTeX

Games live in a category of *nominal sets* [Gabbay&Pitts]

Definition An ***arena*** $A = (M_A, I_A, \vdash_A, \lambda_A)$ is given by:

- a strong nominal set M_A of *moves*,
- a nominal subset $I_A \subseteq M_A$ of *initial moves*,
- a nominal *labelling* function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$,
- a nominal *justification* relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$;

satisfying the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$
- $m \vdash_A n \implies \lambda_A^{O,P}(m) \neq \lambda_A^{O,P}(n) \wedge (\lambda_A^{Q,A}(m) = A \implies \lambda_A^{Q,A}(n) = Q)$

In LaTeX

Games live in a category of *nominal sets* [Gabbay&Pitts]

Definition An ***arena*** $A = (M_A, I_A, \vdash_A, \lambda_A)$ is given by:

- a strong nominal set M_A of *moves*,
- a nominal subset $I_A \subseteq M_A$ of *initial moves*,
- a nominal *labelling* function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$,
- a nominal *justification* relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$;

satisfying the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)$
- $m \vdash_A n \implies \lambda_A^{O,P}(m) \neq \lambda_A^{O,P}(n) \wedge (\lambda_A^{Q,A}(m) = A \implies \lambda_A^{Q,A}(n) = Q)$

Given arenas A, B, the ***interface*** $A \to B = (M_{A\to B}, I_{A\to B}, \vdash_{A\to B}, \lambda_{A\to B})$ is given by:

- $M_{A\to B} = M_A \uplus M_B$ and $I_{A\to B} = I_A$
- $\lambda_{A\to B} = [\lambda_A, \lambda_B]$
- $\vdash_{A\to B} = (I_A \times I_B) \cup \vdash_A \cup \vdash_B$

In LaTeX

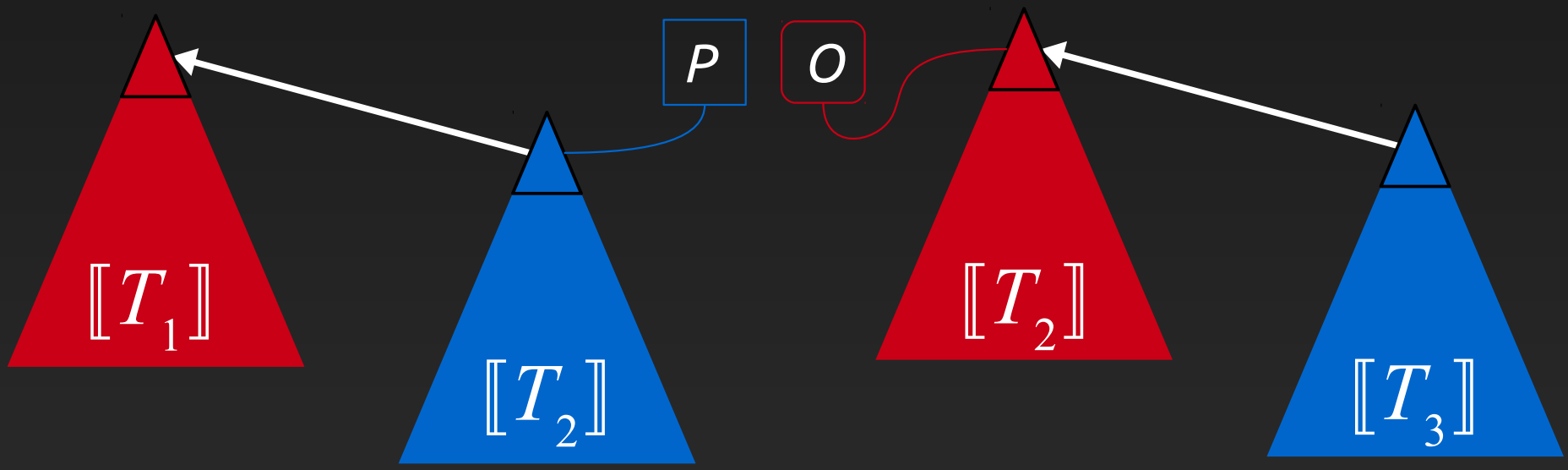
Definition A ***play*** in $A \to B$ is a sequence of *moves-with-store*, written m^{Σ}, where $m \in M_{A \to B}$ and Σ a finite *store*, satisfying the conditions:

- O/P alternation,
- justification,
- well-bracketing,
- frugality (stores contain only visible/reachable names),
- ...

Definition Given an interface $A \to B$, a ***strategy*** $\sigma : A \to B$ is a nominal set of even-length plays satisfying the conditions: *[. . .]*

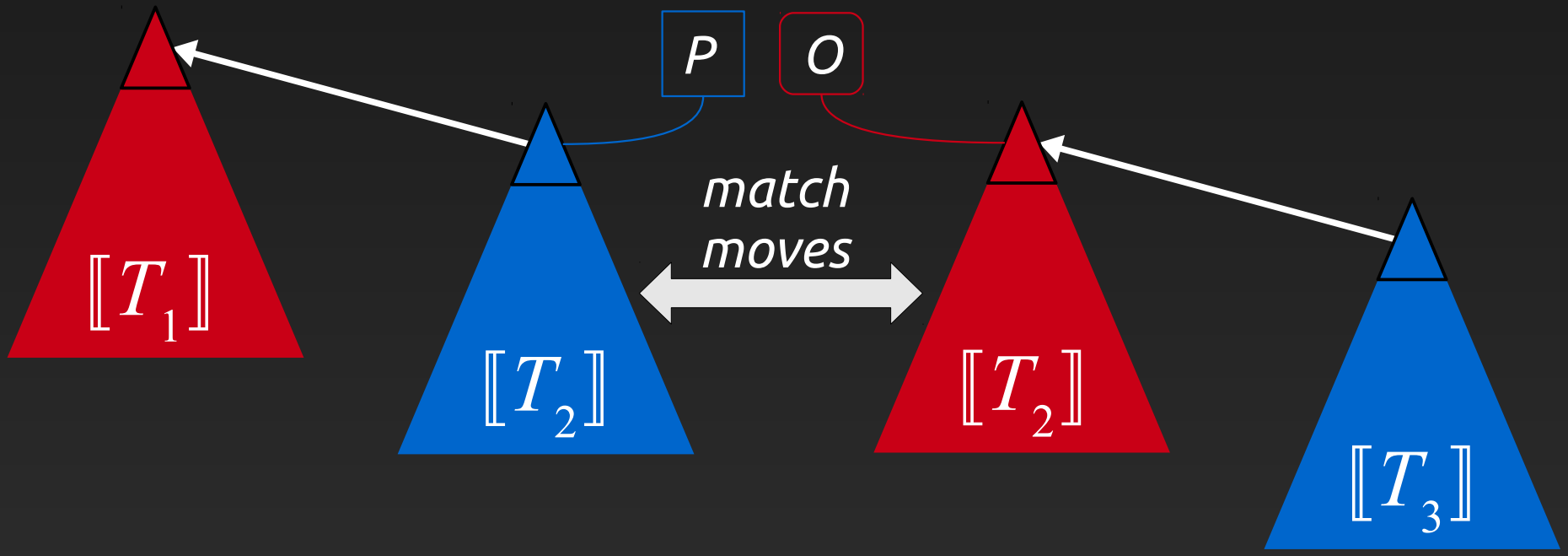
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



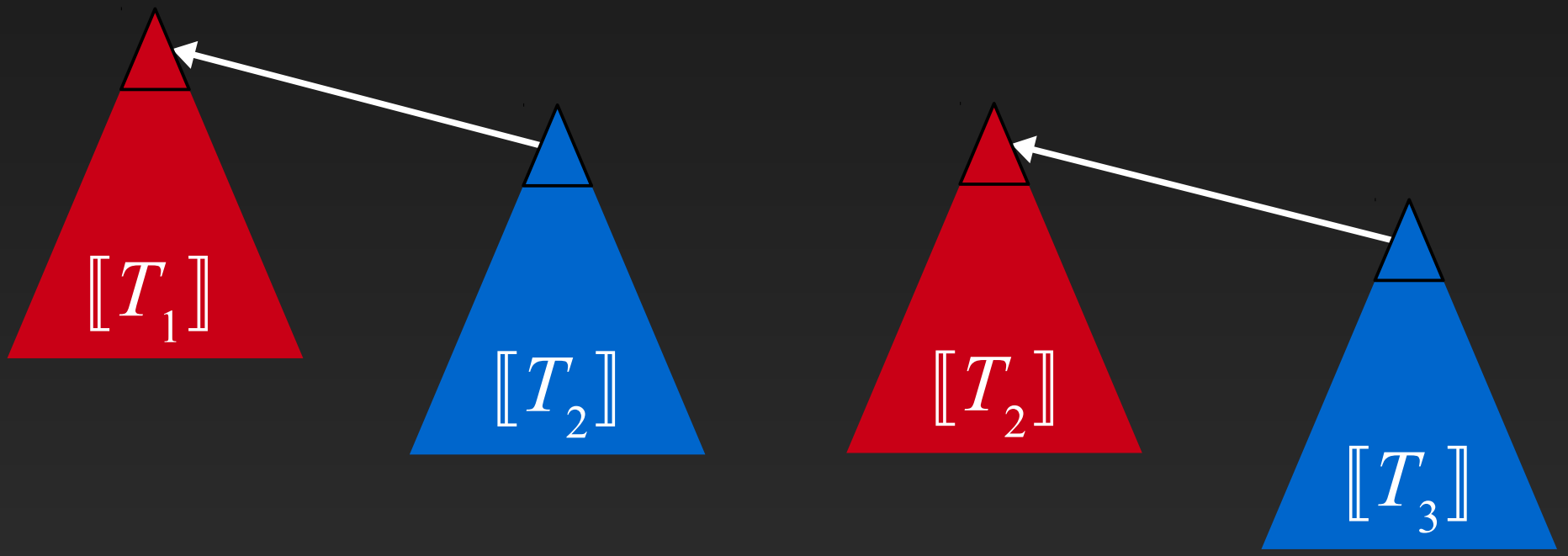
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



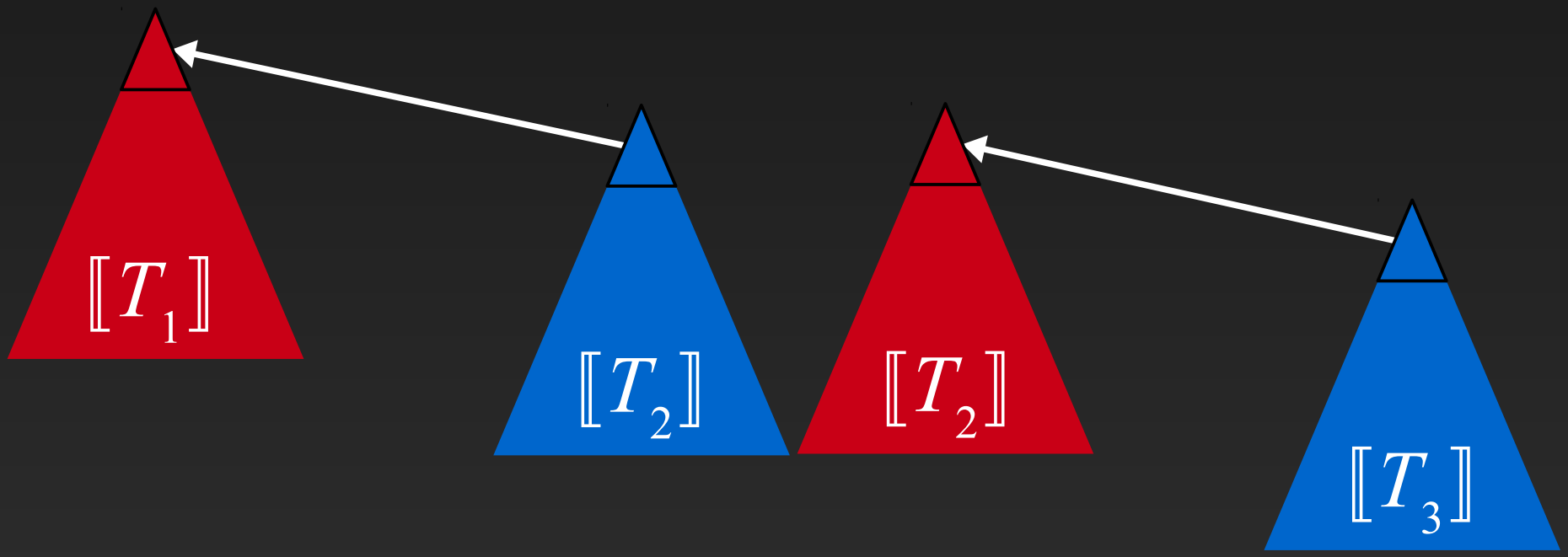
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



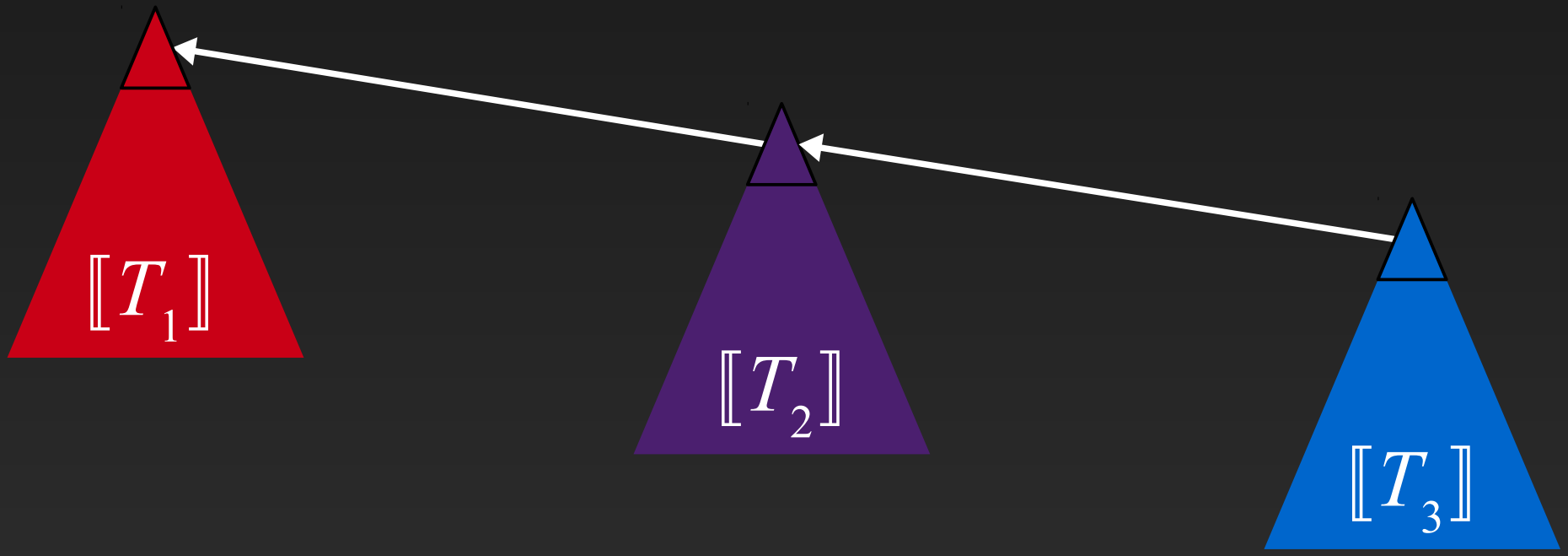
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



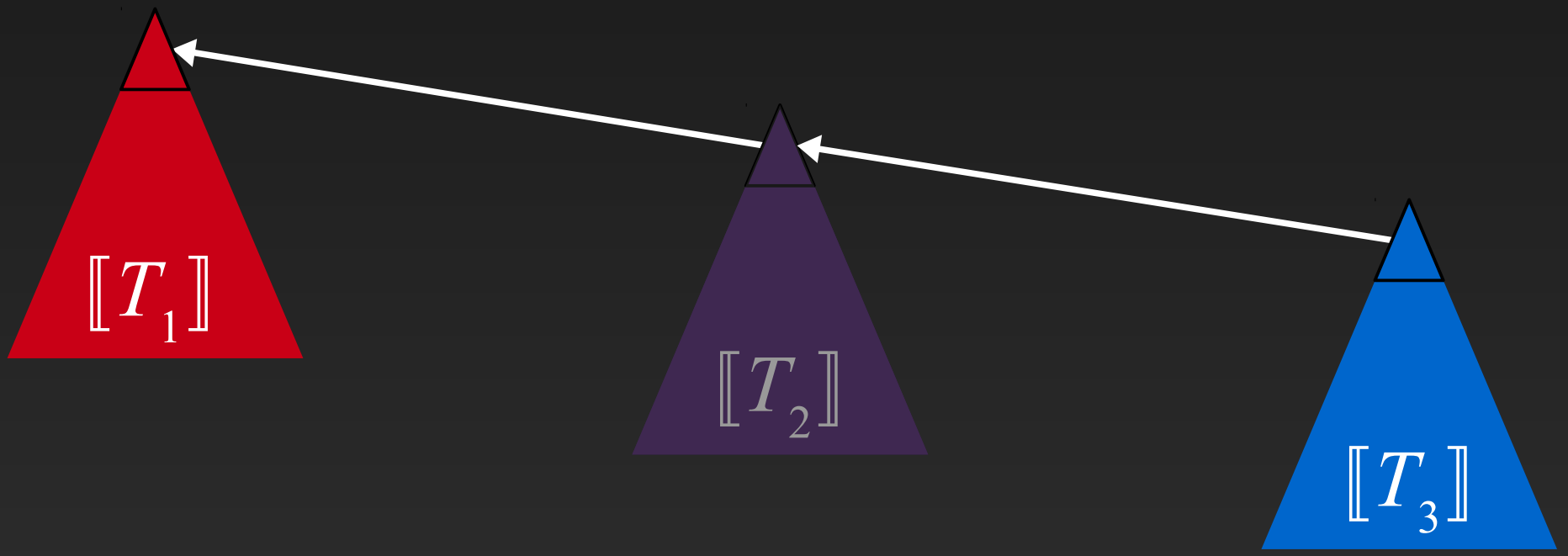
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



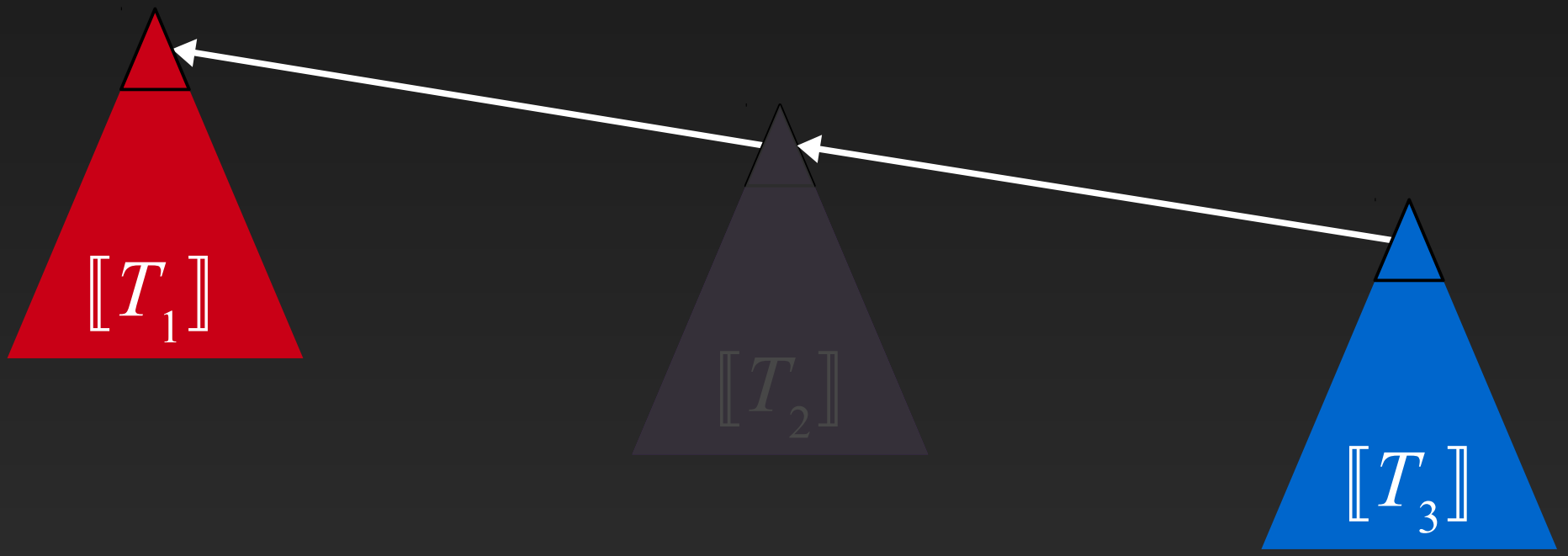
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



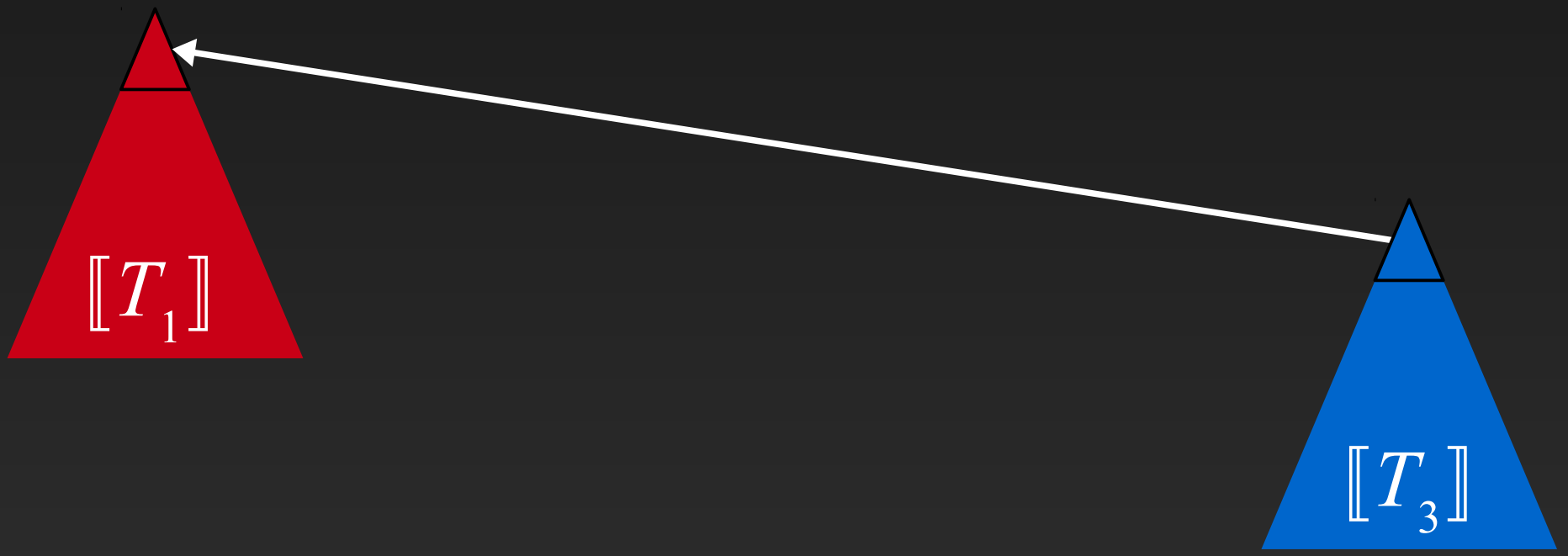
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



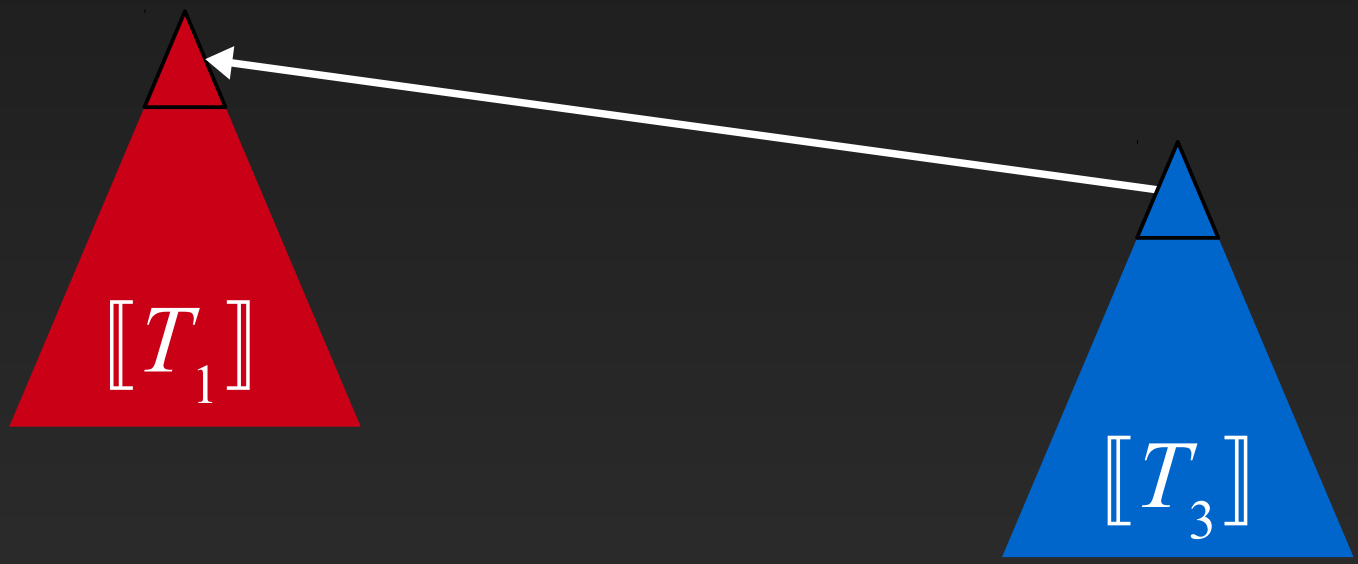
Composition

$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \qquad [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$



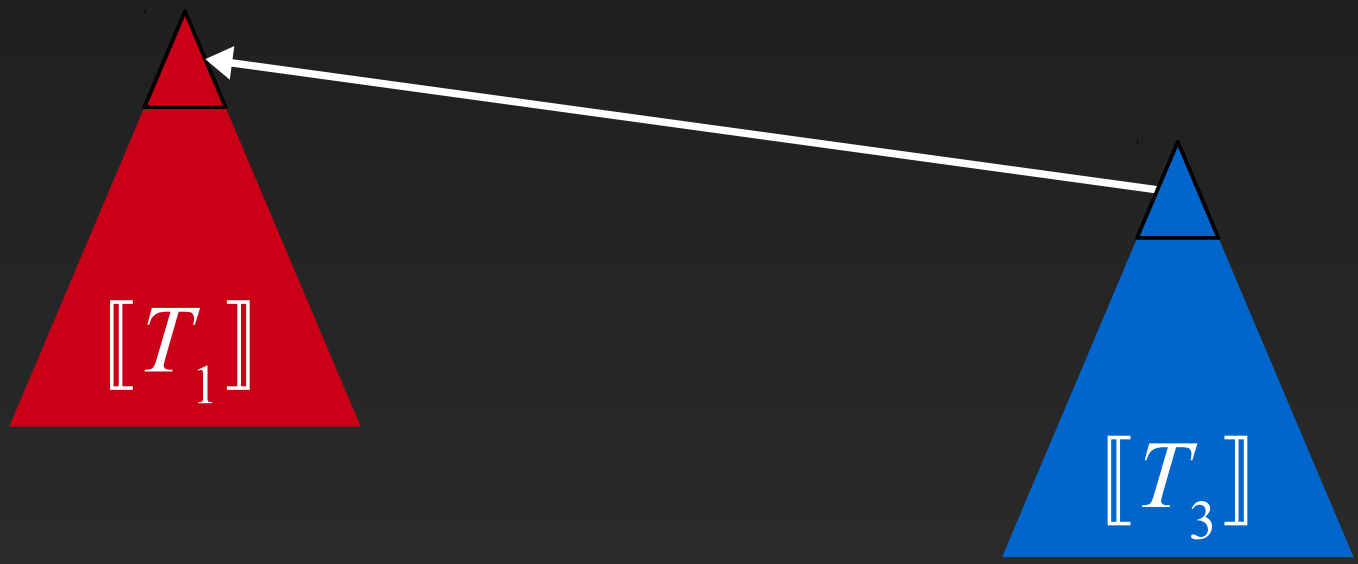
Composition

$$[\![f_1{:}T_1 \vdash M_1{:}T_2]\!] \;;\; [\![f_2{:}T_2 \vdash M_2{:}T_3]\!]$$
$$= [\![f_1{:}T_1 \vdash \mathrm{let}\, f_2 = M_1 \,\mathrm{in}\, M_2{:}T_3]\!]$$



Composition

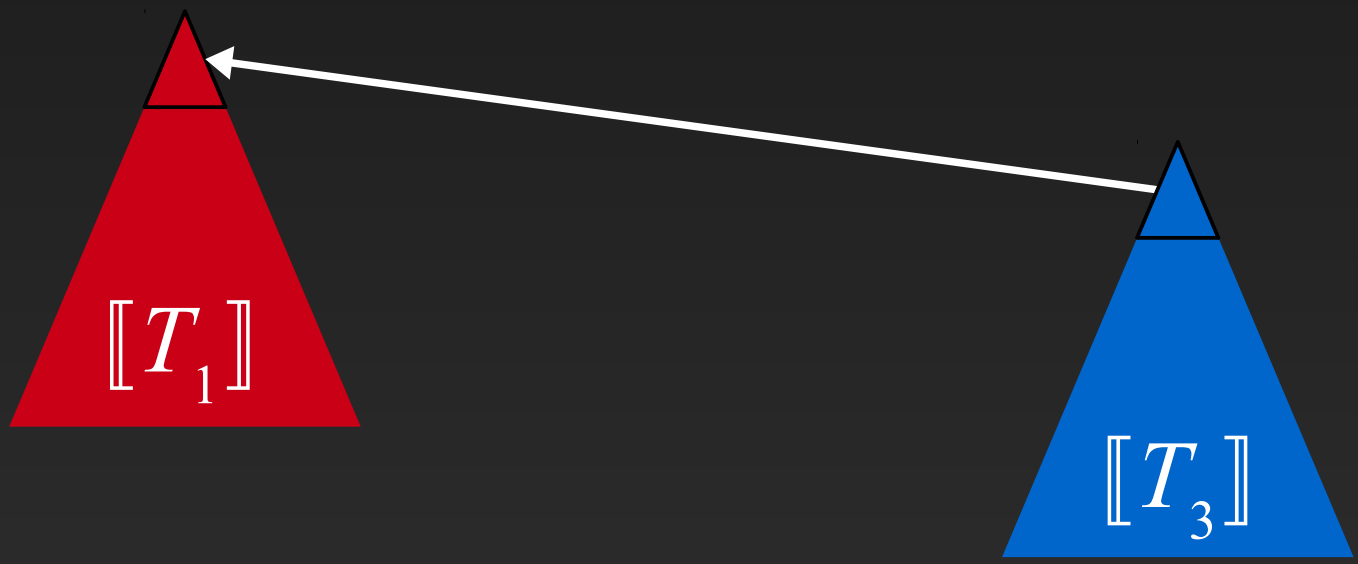
$$[\![f_1 : T_1 \vdash M_1 : T_2]\!] \;;\; [\![f_2 : T_2 \vdash M_2 : T_3]\!]$$
$$= [\![f_1 : T_1 \vdash \text{let } f_2 = M_1 \text{ in } M_2 : T_3]\!]$$



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

Composition

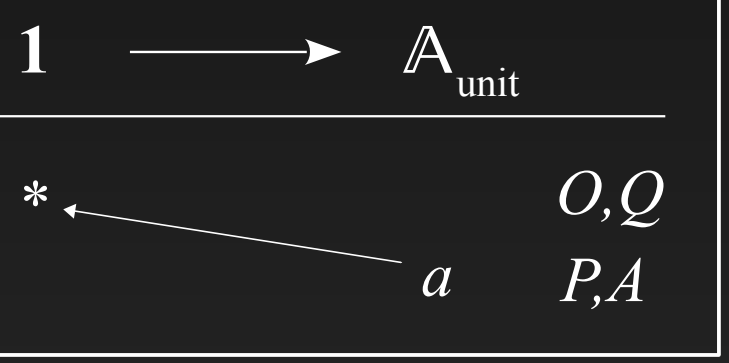
$$[\![f_1{:}T_1 \vdash M_1{:}T_2]\!] \;;\; [\![f_2{:}T_2 \vdash M_2{:}T_3]\!]$$
$$= [\![f_1{:}T_1 \vdash \mathrm{let}\, f_2 = M_1 \,\mathrm{in}\, M_2{:}T_3]\!]$$



Nominal composition requires *Laird conditions* :
these ensure name and store privacy during composition

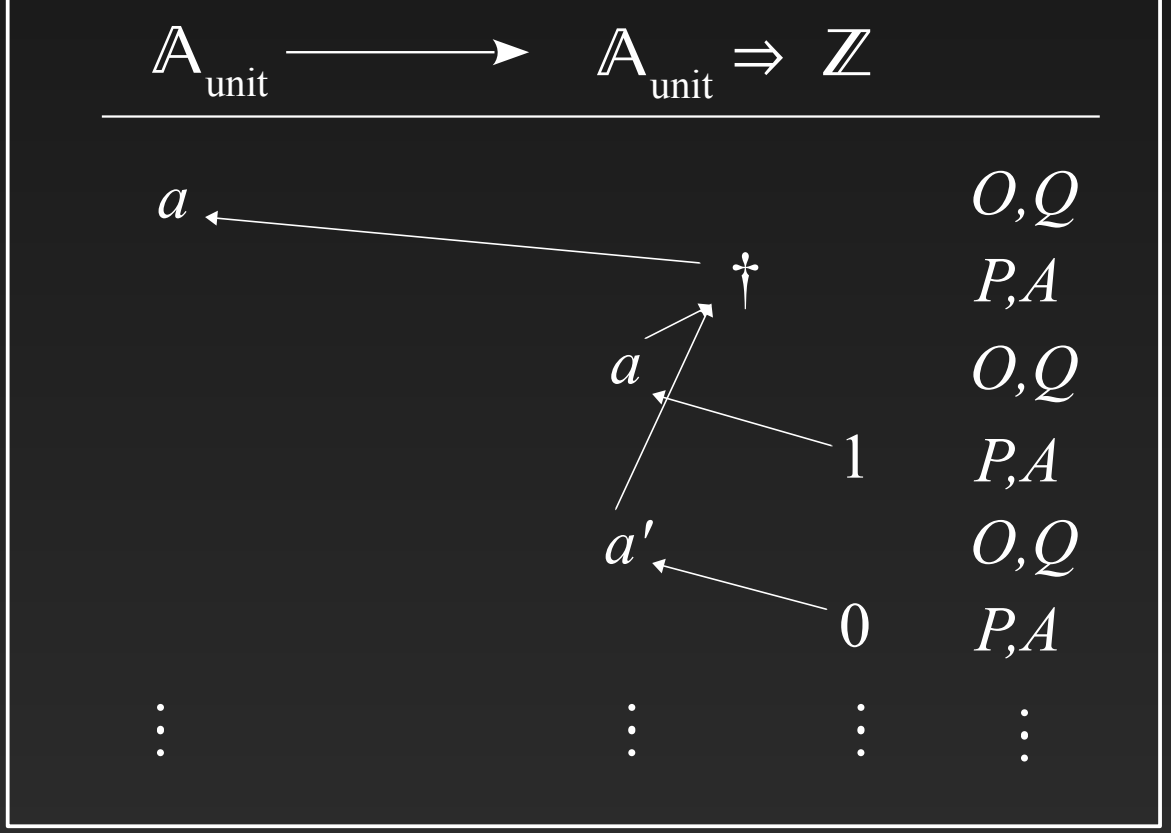
Nominal composition example

$\vdash \mathrm{ref}() : \mathrm{ref\ unit}$



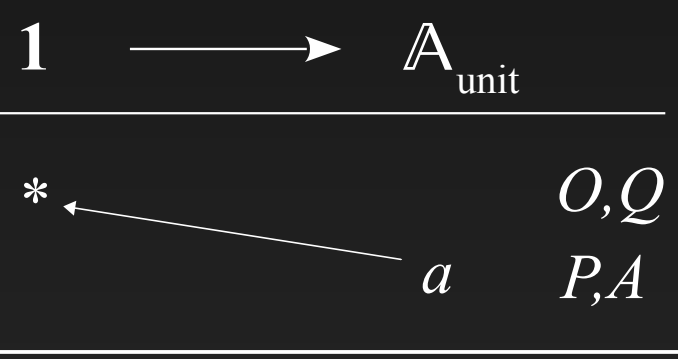
;

$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \to \mathrm{int}$



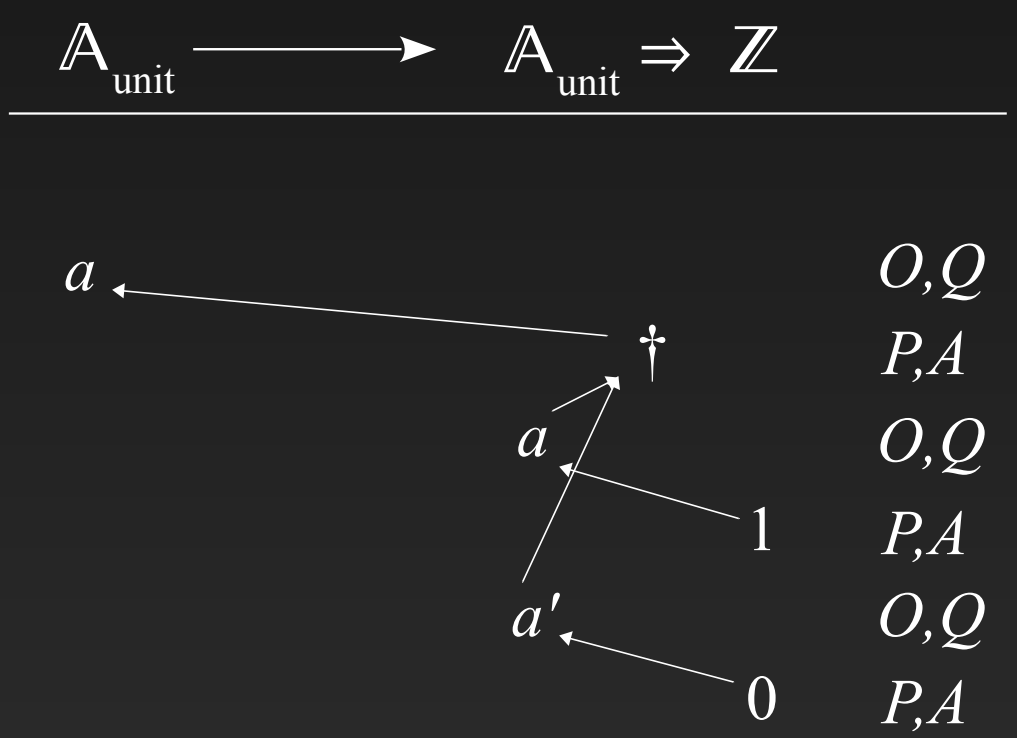
Nominal composition example

$\vdash \mathrm{ref}() : \mathrm{ref\ unit}$

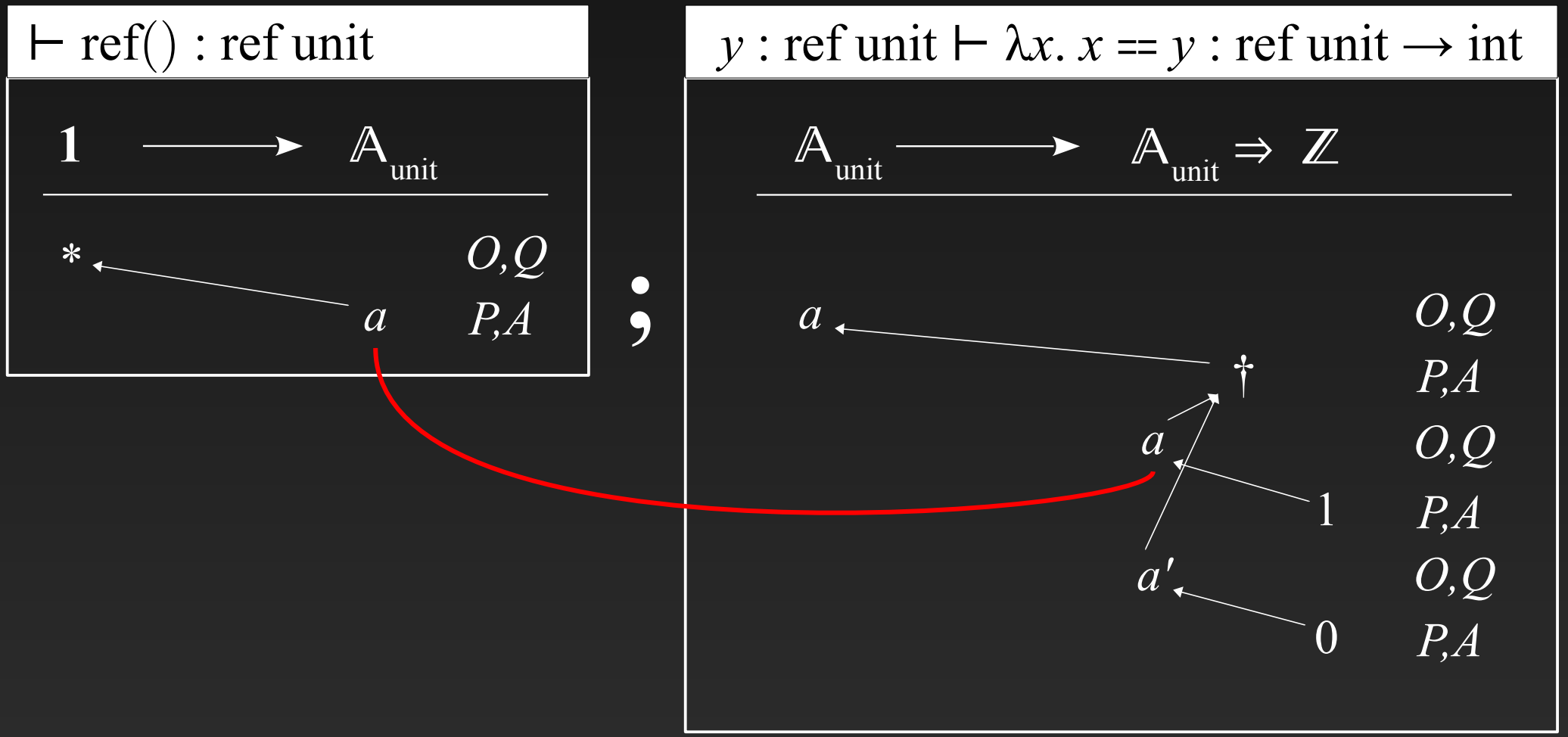


;

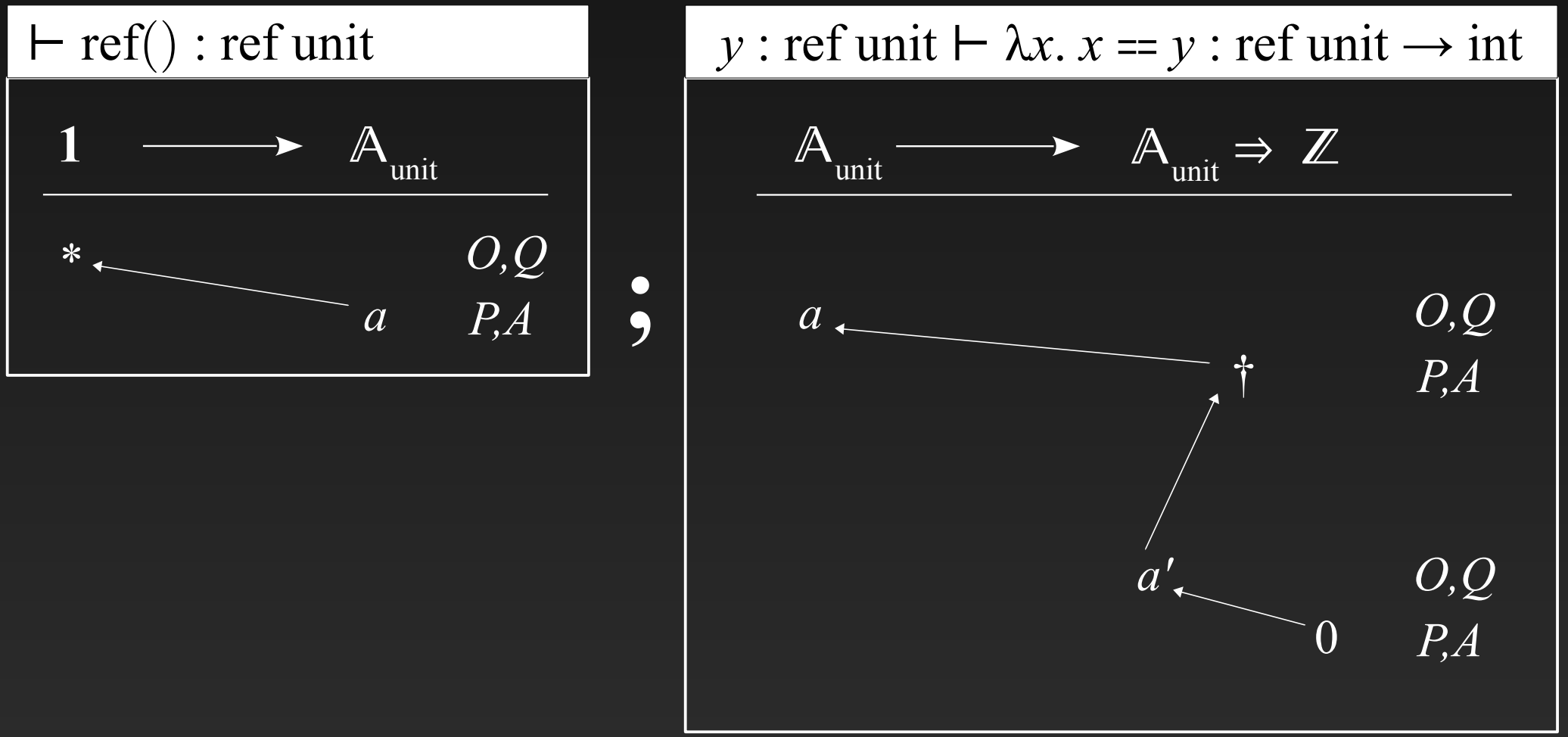
$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \to \mathrm{int}$



Nominal composition example



Nominal composition example



Nominal composition example

$\vdash \mathrm{ref}() : \mathrm{ref\ unit}$

$\mathbb{1} \longrightarrow \mathbb{A}_{\mathrm{unit}}$

$*$ ← a $\quad O,Q\ P,A$

;

$y : \mathrm{ref\ unit} \vdash \lambda x.\ x == y : \mathrm{ref\ unit} \to \mathrm{int}$

$\mathbb{A}_{\mathrm{unit}} \longrightarrow \mathbb{A}_{\mathrm{unit}} \Rightarrow \mathbb{Z}$

a ← $\dagger$ $\quad O,Q\ P,A$

a' ← 0 $\quad O,Q\ P,A$

$= [\![\ \lambda x.0 : \mathrm{ref\ unit} \to \mathrm{int}\ ]\!]$

Taking stock

- Freyd categories for several different effects
- Soundness – Definability – Full Abstraction

Effects = – conditions + name generators

- Ground store:

  – innocence + ground reference names

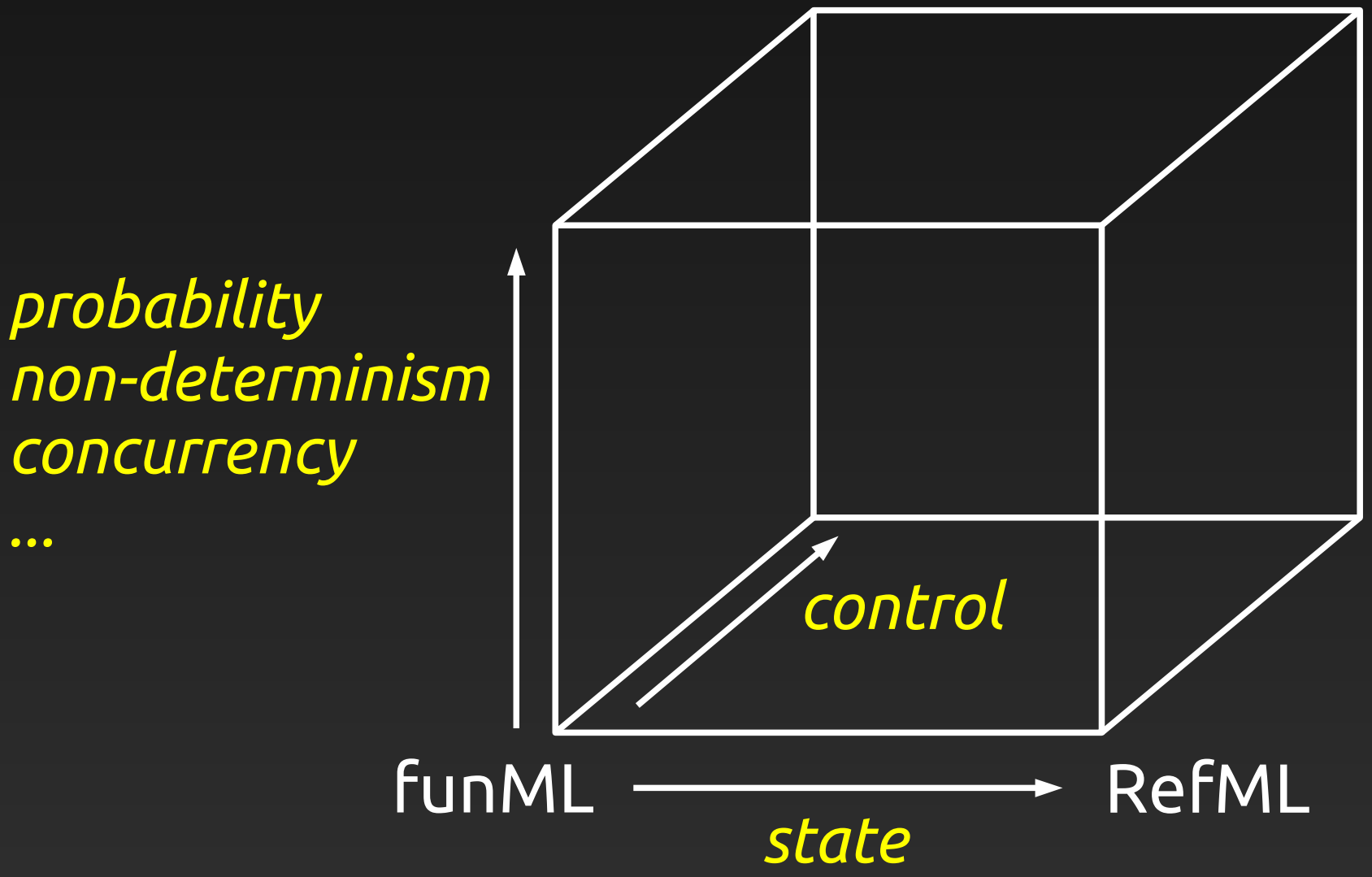
- Higher-order store:

  – visibility + HO reference names

- Exceptions:

  – well-bracketing + exception names

Nominal Abramsky cube



Nominal games so far

- Full abstraction for languages with:
  - References (ground & higher-order)
  - Concurrency (higher-order channels)
  - Exceptions (local, private)
  - Objects (Java)

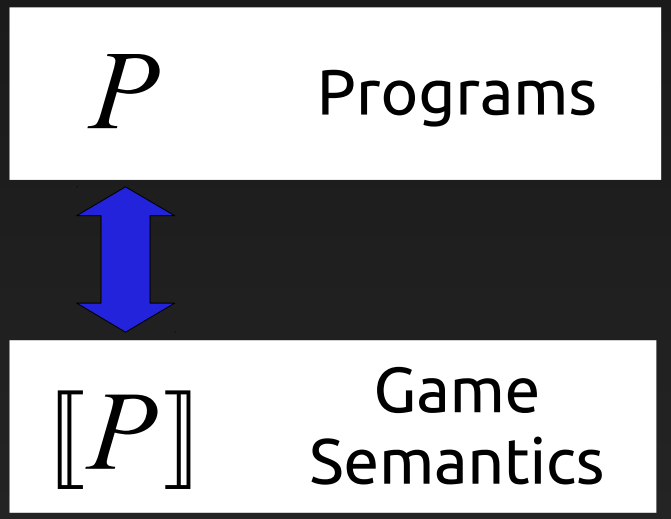
`dblp:` AGMOS, Laird, Murawski, T.

- Trace models

`dblp:` Laird, Ghica, Jaber, T.

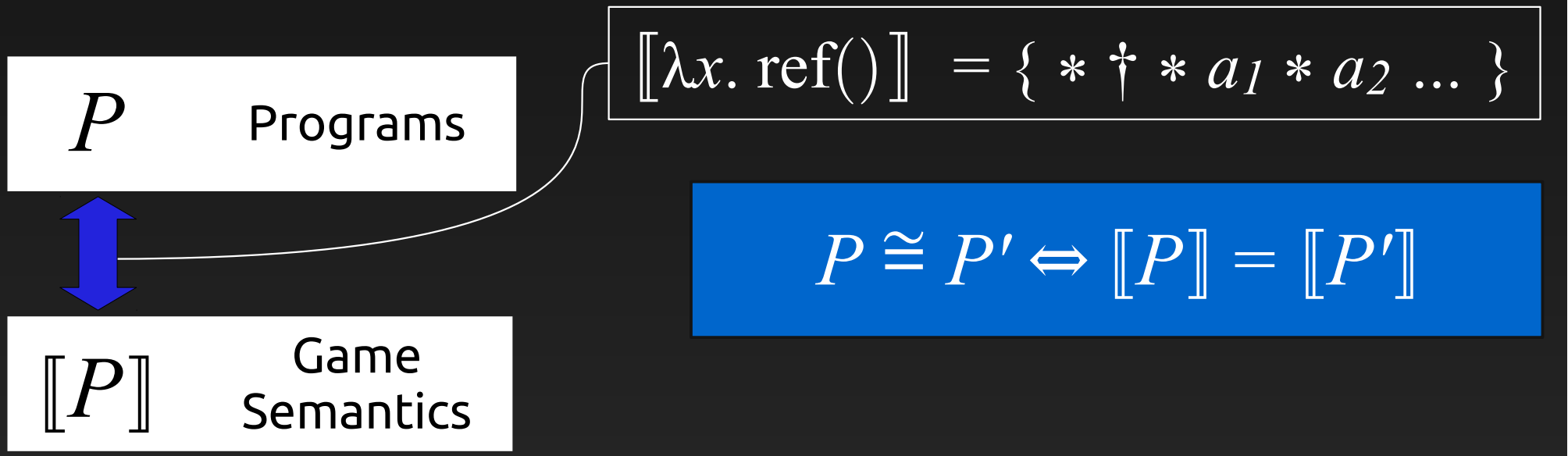
- Algorithmic games

Algorithmic game semantics

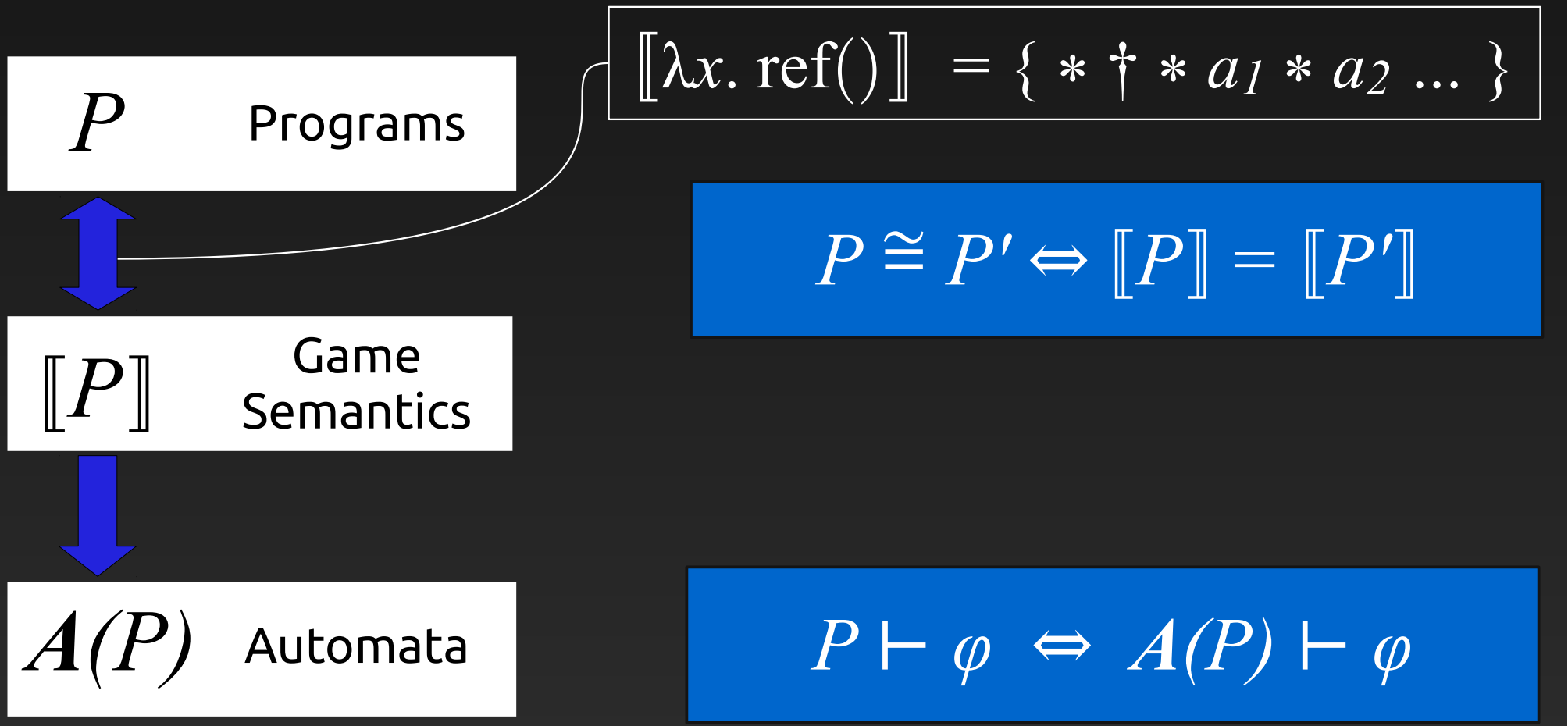


$$P \cong P' \Leftrightarrow [\![P]\!] = [\![P']\!]$$

Algorithmic game semantics



Algorithmic game semantics




Nominal Automata

The need for nominal automata

- Nice (fully abstract) game models
- Represented as sets of strings (with structure)

- *How to make them algorithmic/automated?*
- Use automata

- *But, the alphabet is infinite...*
- Use nominal automata

Fresh-register automata

$$[\![\lambda x.\ \mathrm{ref}()]\!] = \{ * \dagger * a_1 * a_2 \ldots \mid a_i\text{'s distinct} \}$$

Automata with names

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

Fresh-register automata

$$[\![\lambda x.\ \mathrm{ref}()]\!] = \{ * \dagger * a_1 * a_2 \ldots \mid a_i\text{'s distinct} \}$$

Automata with names

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

Part of a huge body of work on automata over infinite alphabets!

Finite-state machines with registers

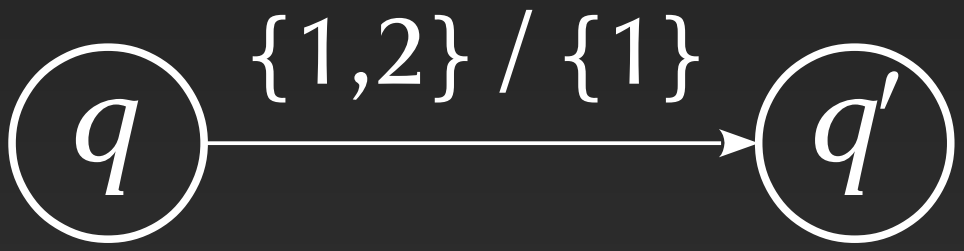
- *Register Automata (Kaminski & Francez, 1994)*

Register automata

X,Y: *sets of register indices*

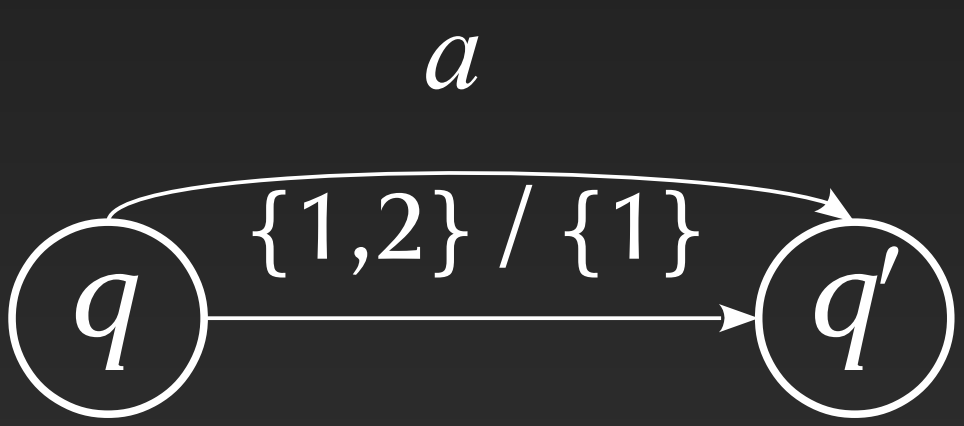
Register automata

X,Y: *sets of register indices*



Register automata

X,Y: *sets of register indices*



Fresh-register automata: add “fresh”

Y: *set of register indices*

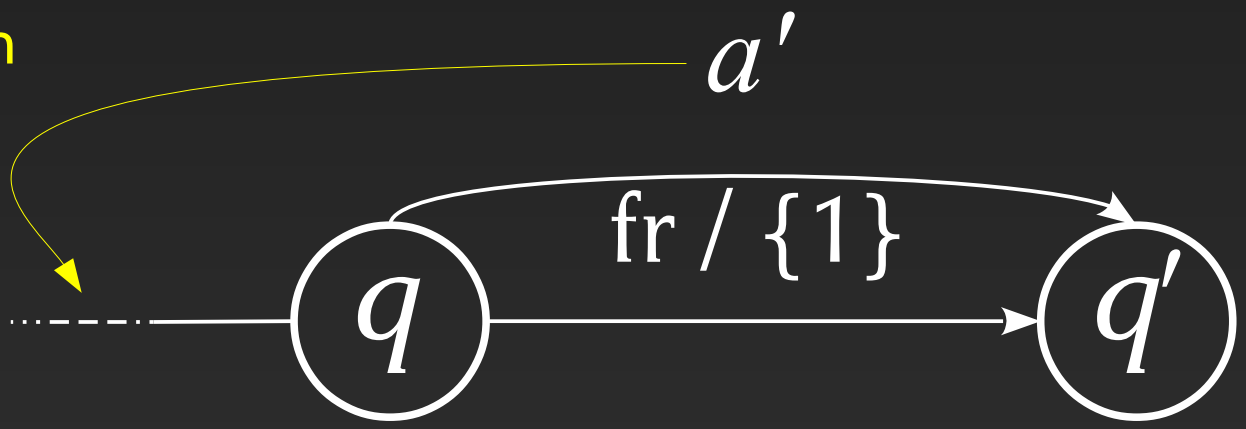
Fresh-register automata: add "fresh"

Y: *set of register indices*

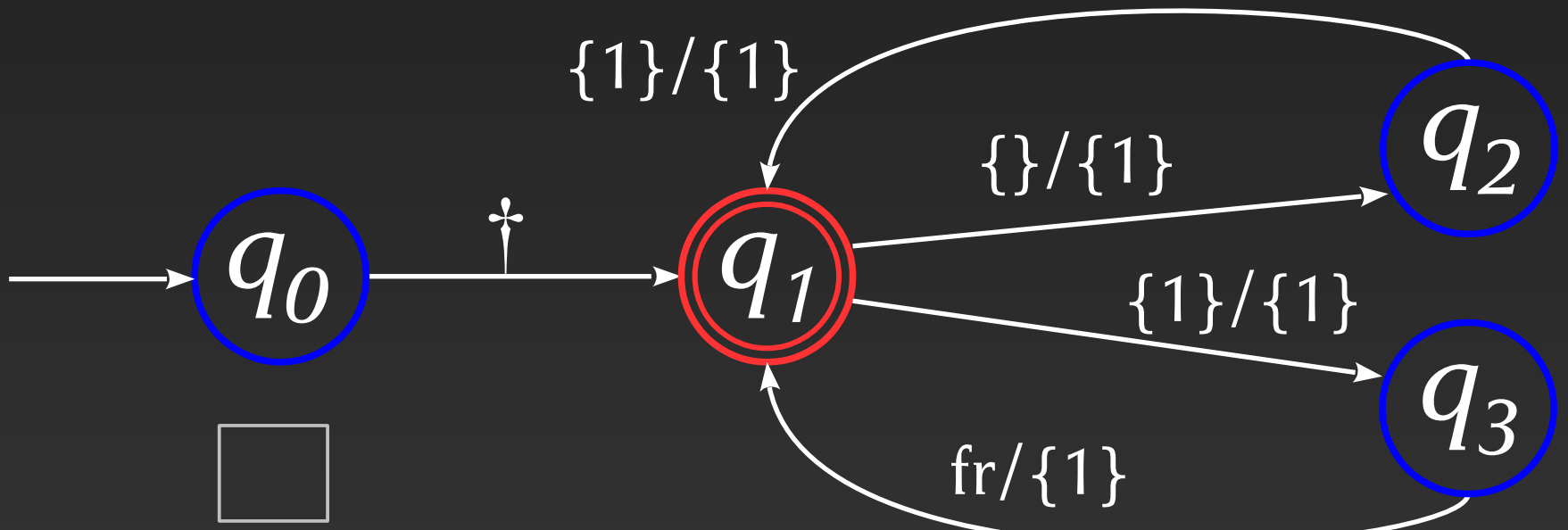
Fresh-register automata: add "fresh"

Y: *set of register indices*

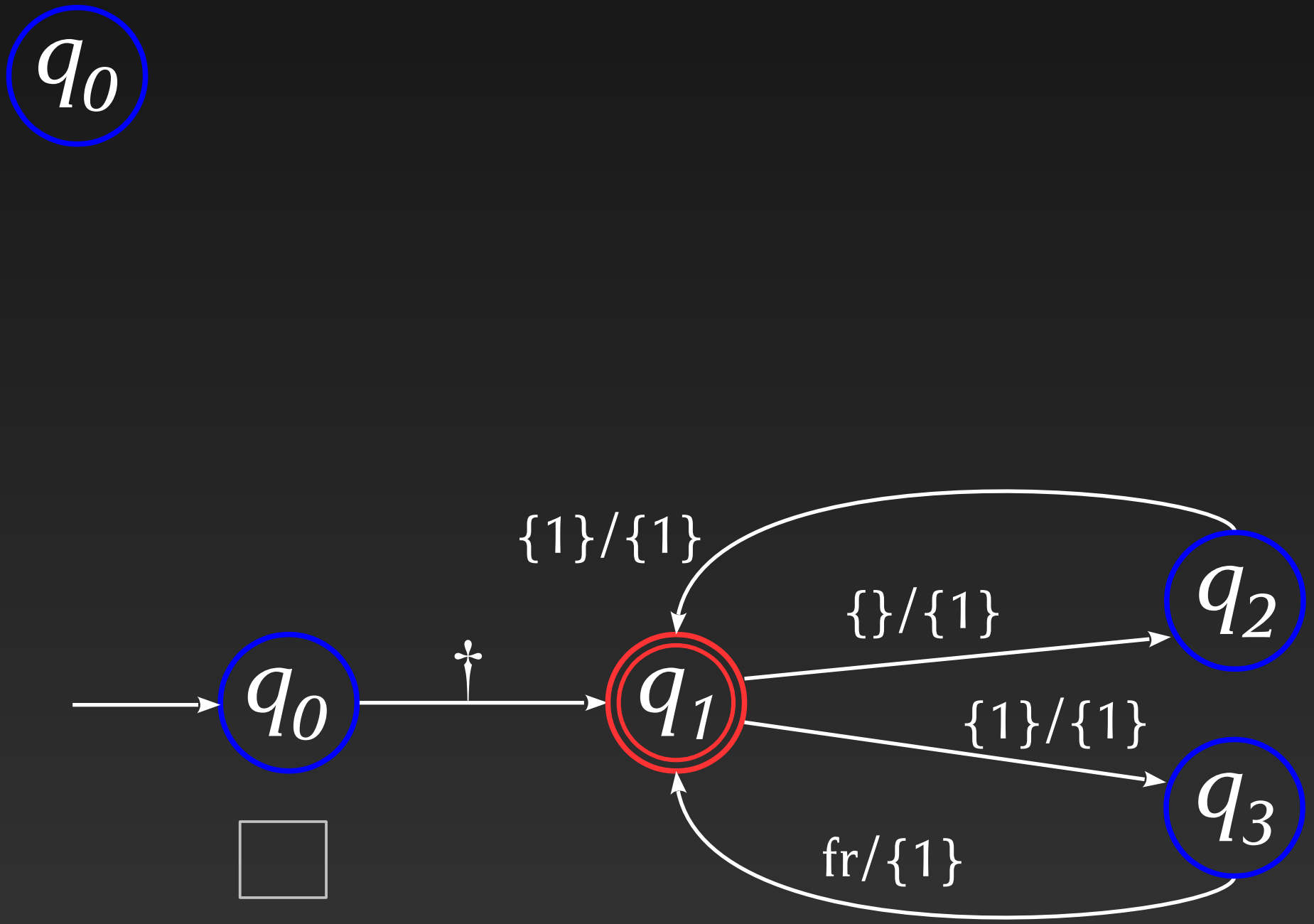
fresh here



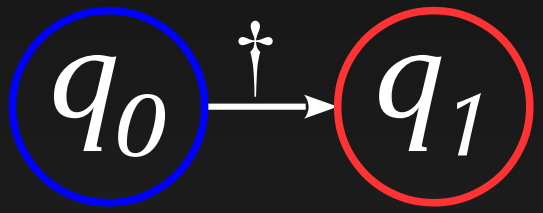
Fresh-register automata

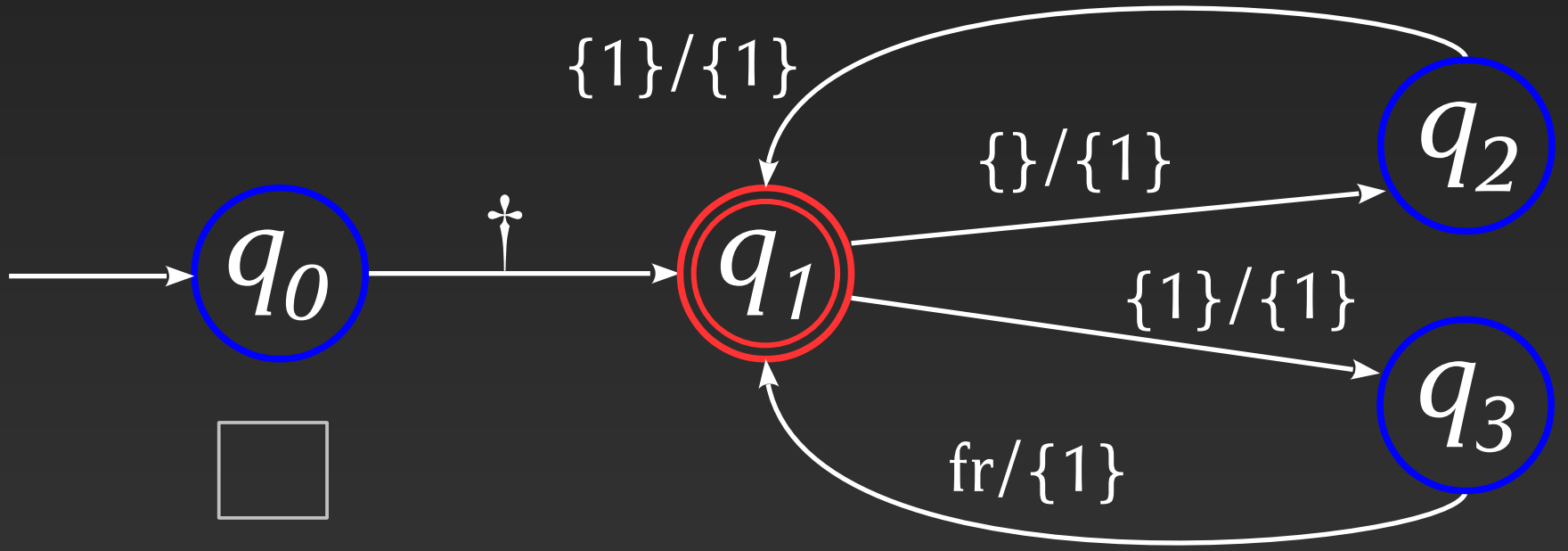


Fresh-register automata

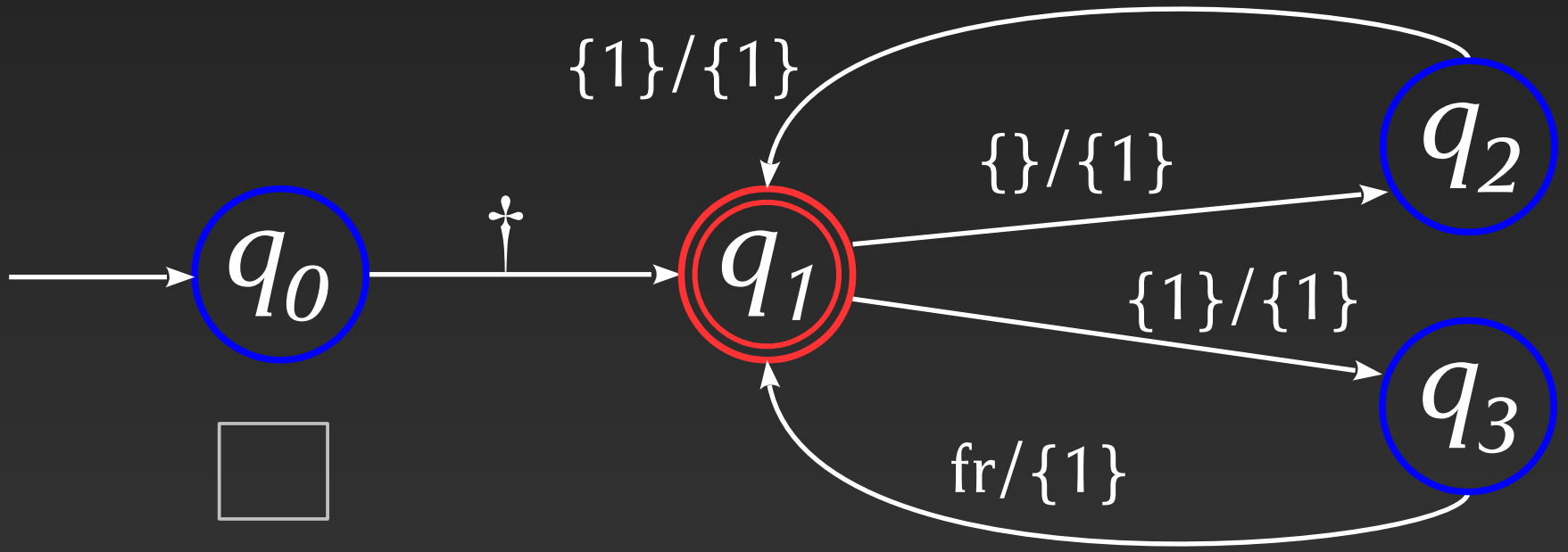


Fresh-register automata

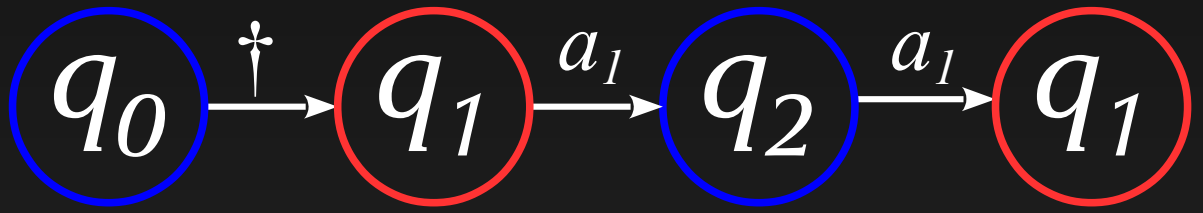


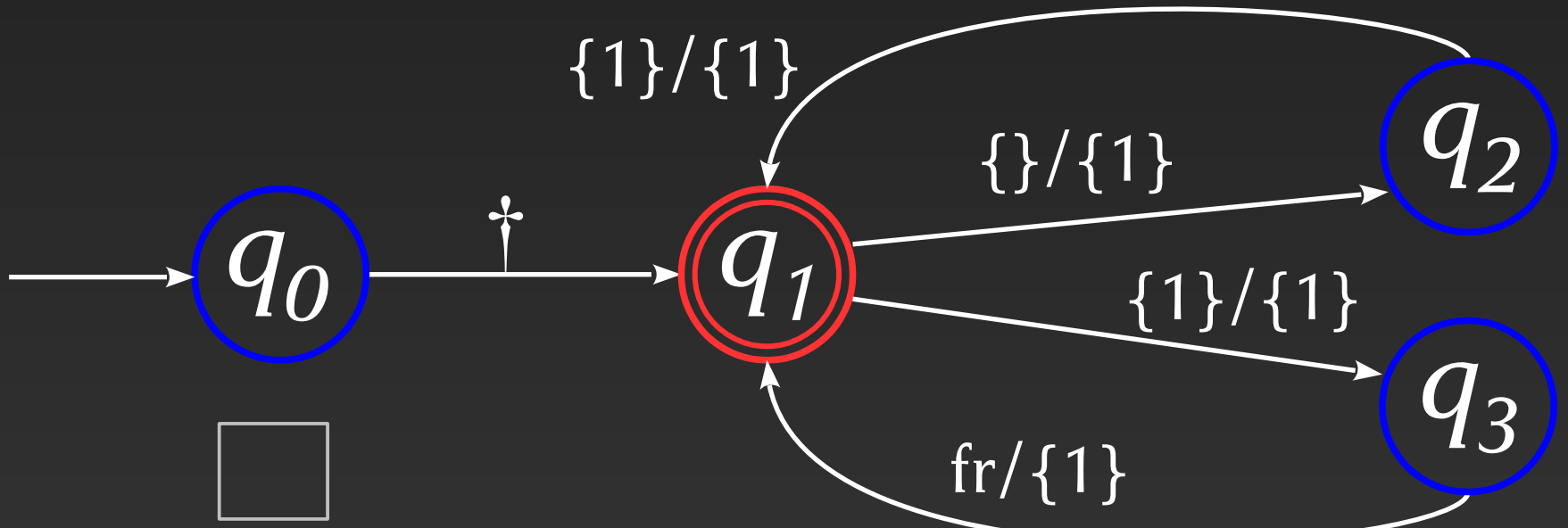


Fresh-register automata

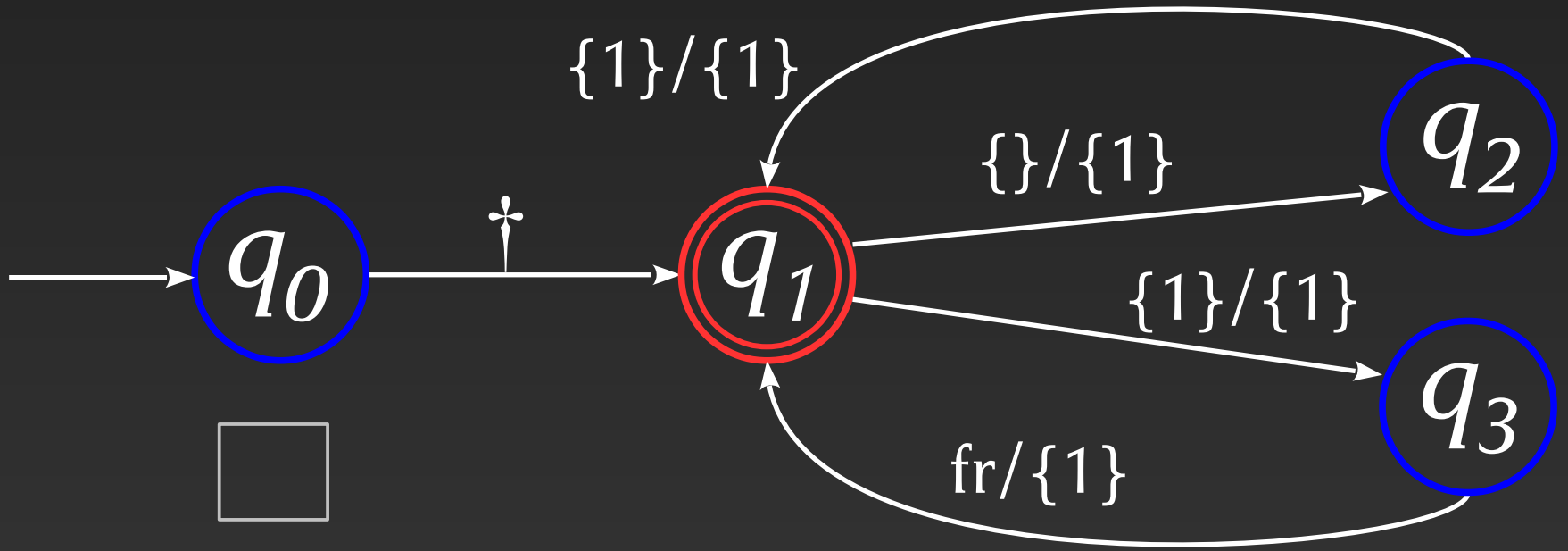


Fresh-register automata

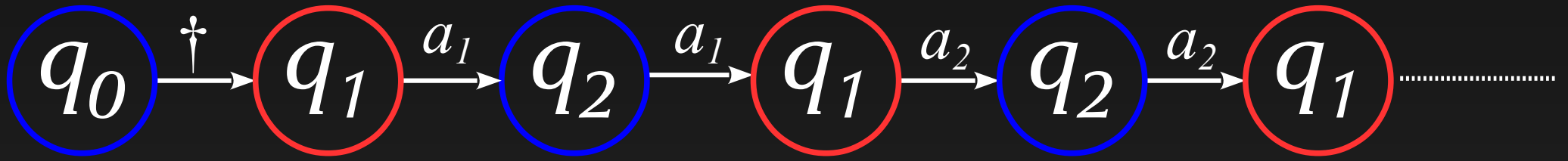


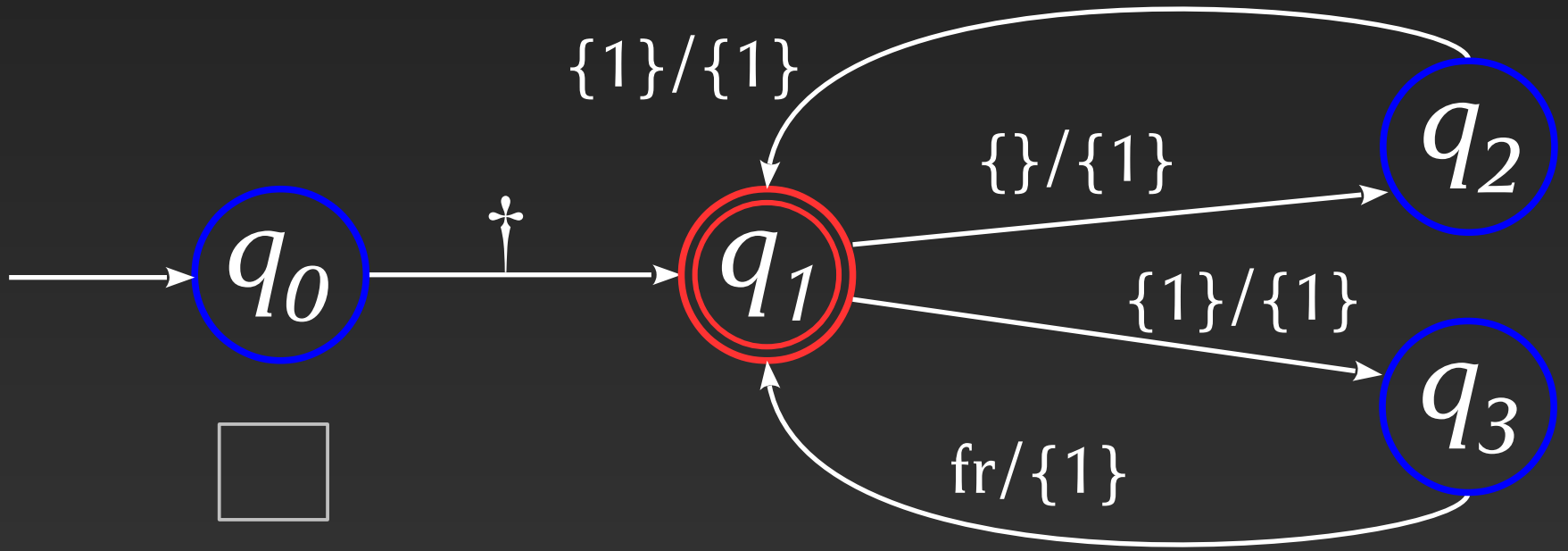


Fresh-register automata

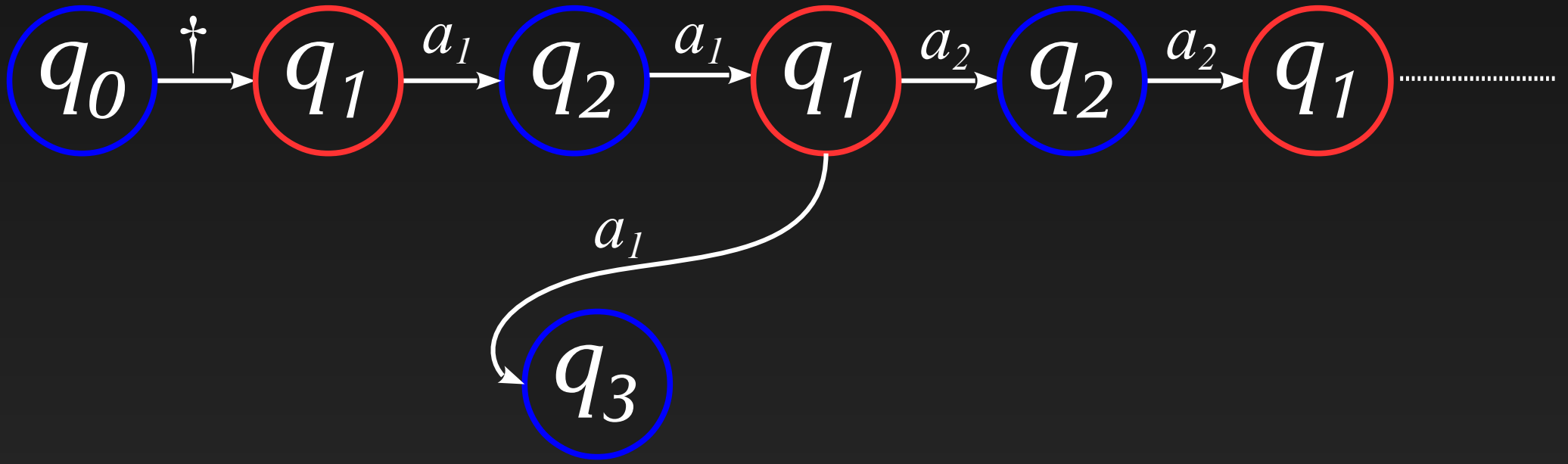


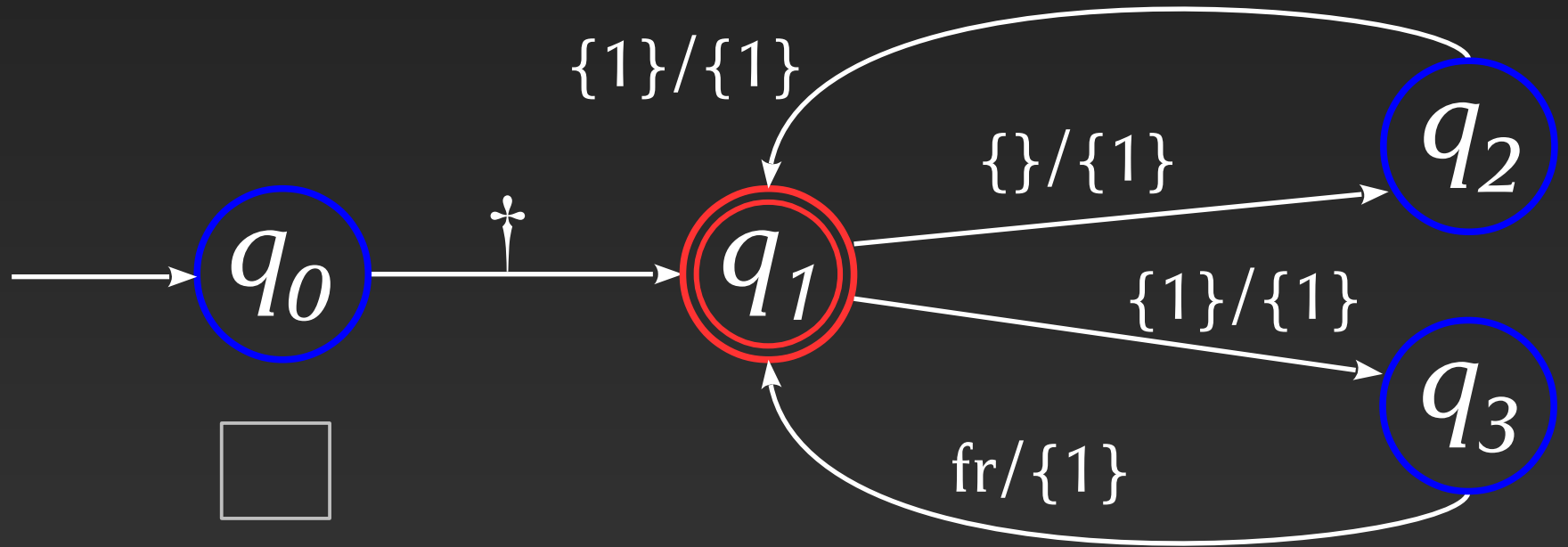
Fresh-register automata



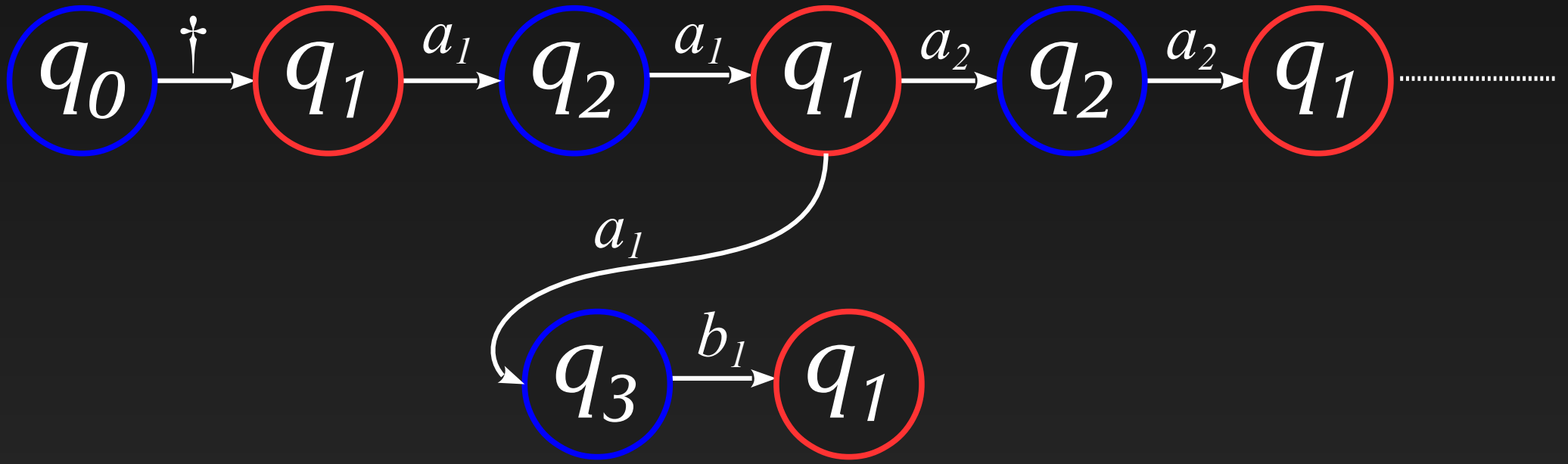


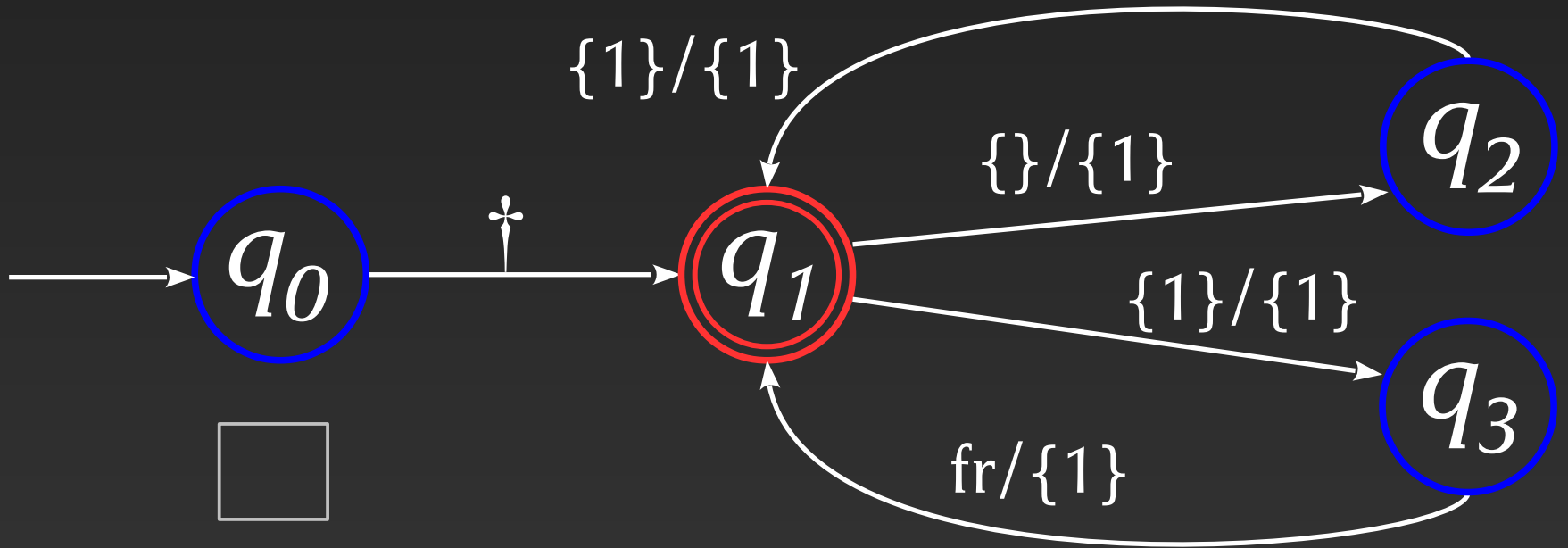
Fresh-register automata



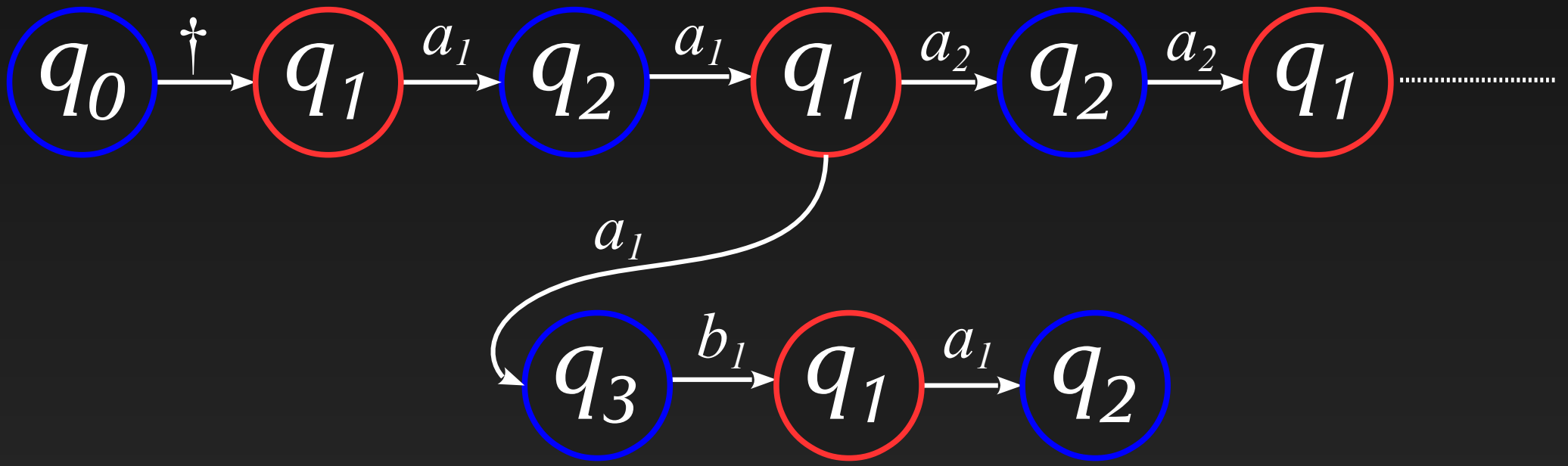


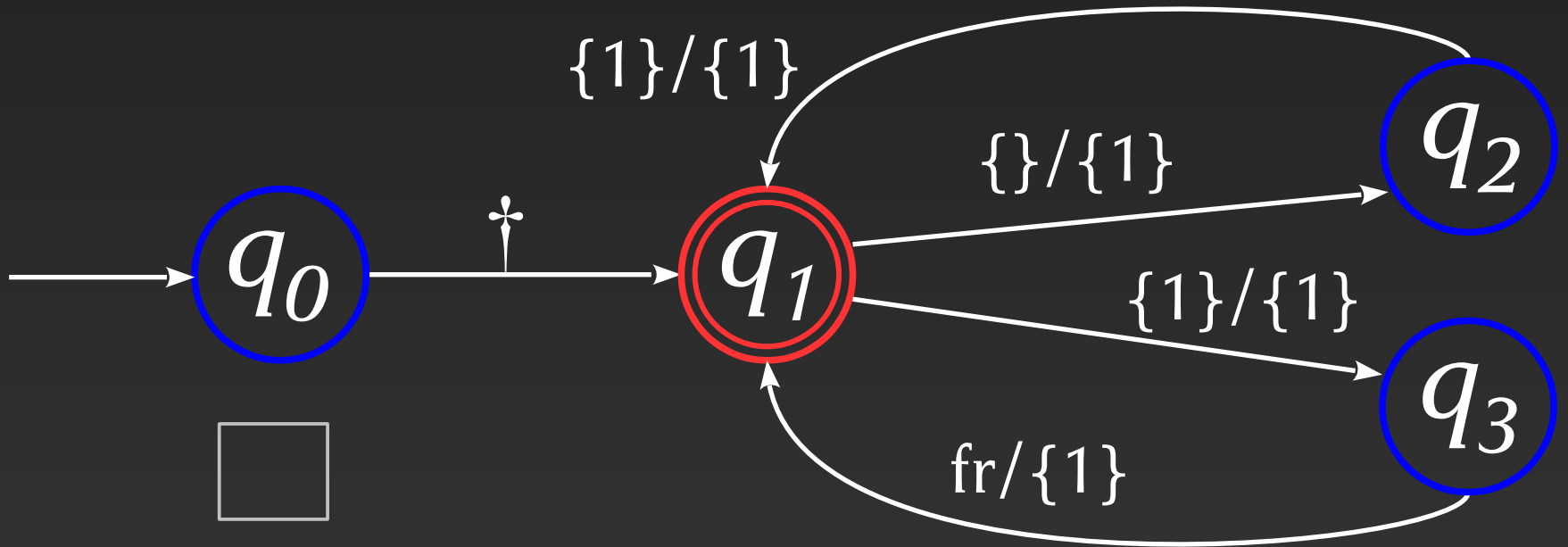
Fresh-register automata



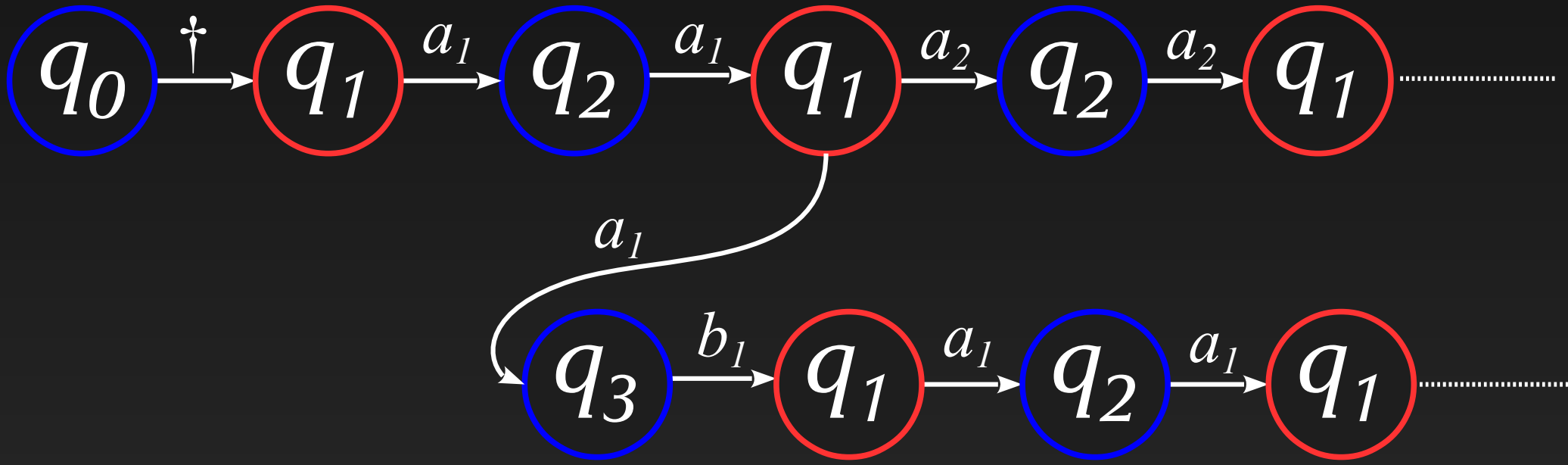


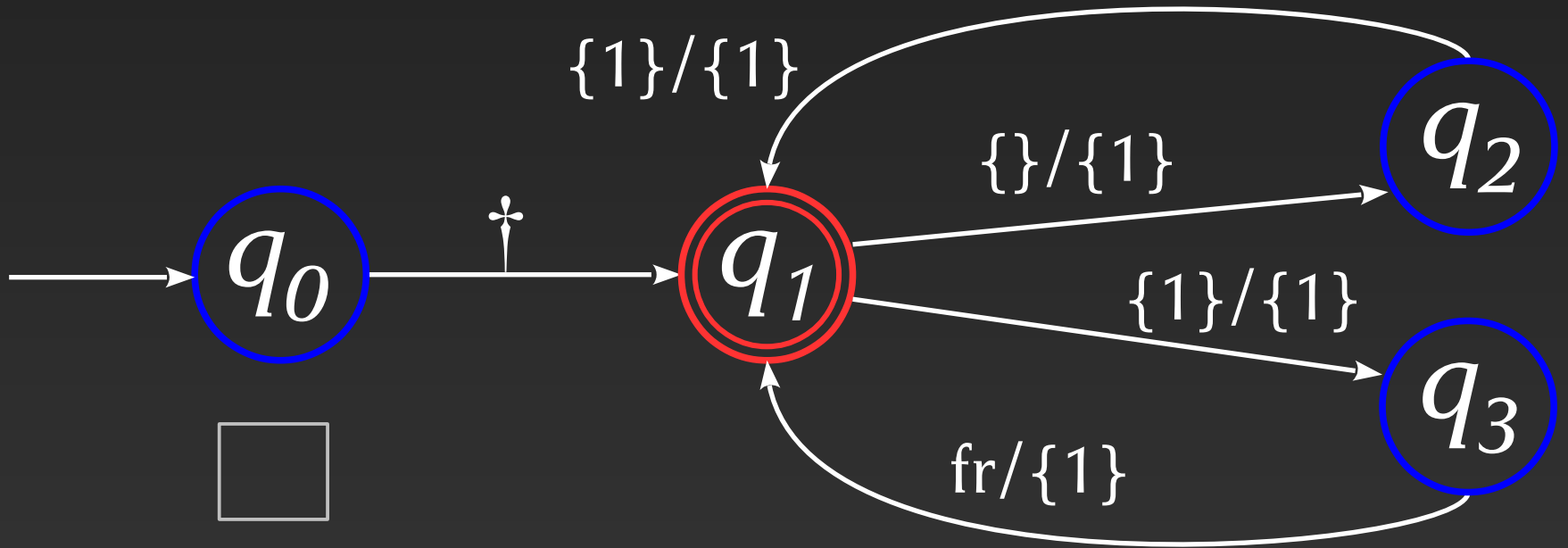
Fresh-register automata



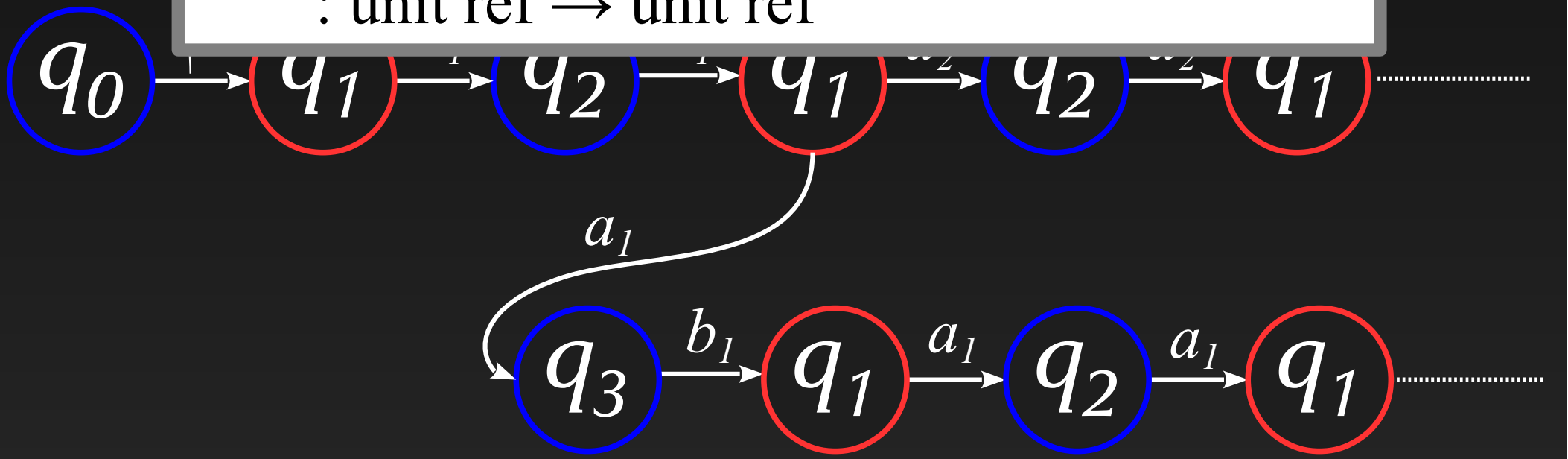


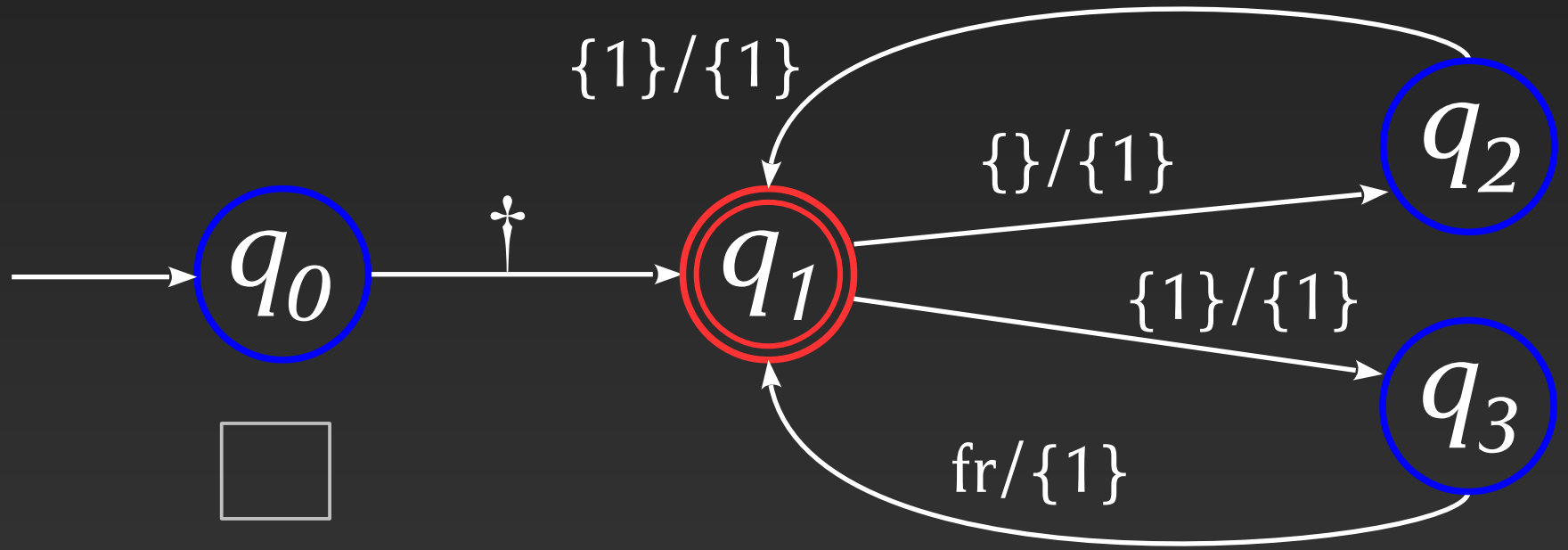
Fresh-register automata





```
let y = ref(ref()) in
    λx. y := ( if x==!y then ref() else x ); !y
    : unit ref → unit ref
```





Properties of FRAs

- Cleanly extend Register Automata (RA's)
- Closed under union and intersection;
  not under complementation, concatenation, _*
- Universality undecidable (from RA's)
- Emptiness decidable (from RA's)
- Bisimilarity decidable

Algorithmic nominal game semantics

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

Representation of strategies as FRA's

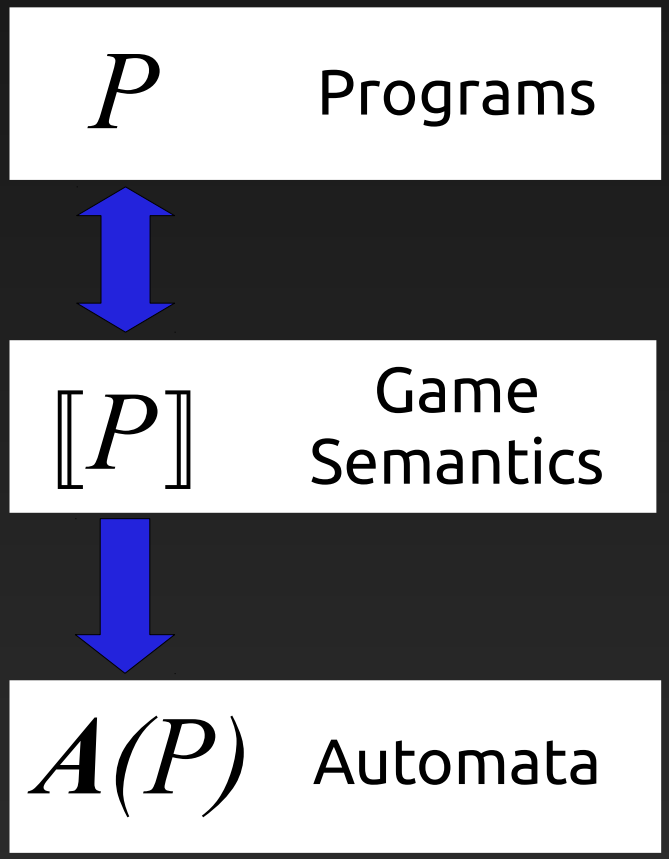
- with “data words”
- pushdown variants

pairs (f, a):
- f from finite alph.
- a a name

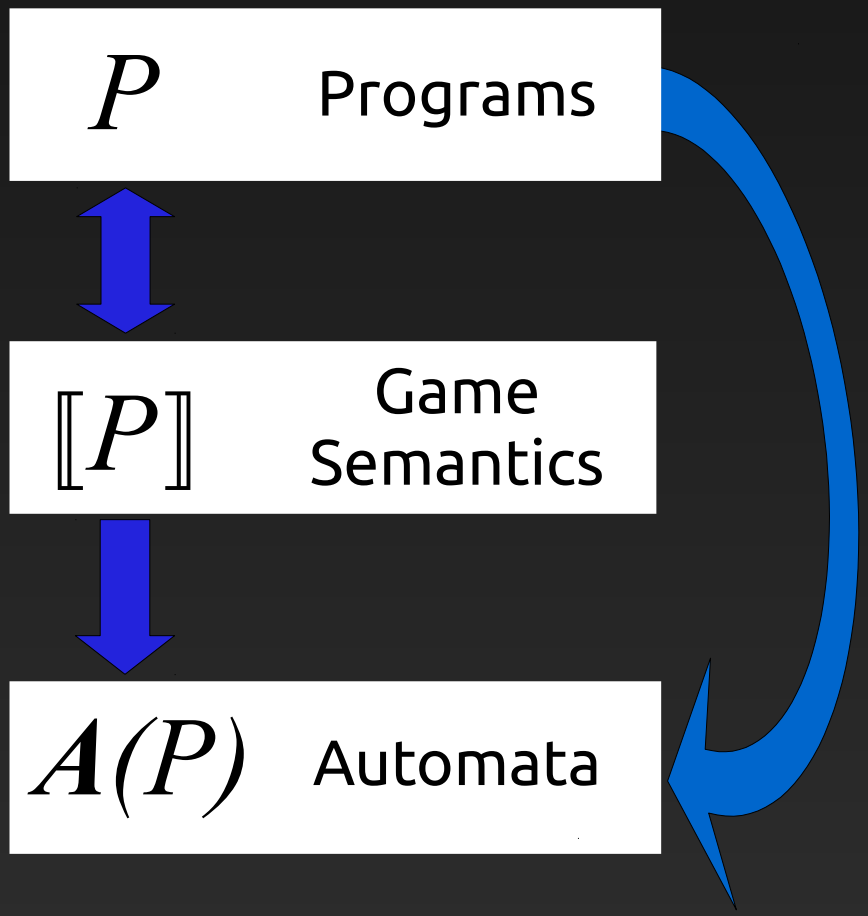
stack stores constants and register “snapshots”

`dblp:` Murawski, Ramsay, T.

From programs to automata



From programs to automata



Algorithmic results

For Ground ML [Murawski & T, ICALP'12]

- Decide equivalence for terms in contexts $\Theta_L \vdash \Theta_R$

$$\Theta_L ::= \beta \mid \Theta_R \to \Theta_L \qquad \Theta_R ::= \beta \mid \Theta_1 \to \beta$$

- This is a full characterisation:

  in higher-order contexts, strategies can be used to encode queue machines

"Weaker" results for Int ML [Murawski & T, ESOP'11]

Other nominal automata in semantics

- History-Dependent Automata
  - finitary π-calculus, symmetries, minimisation

`dblp:` Montanari, Pistore, Ciancia, Ferrari, Tuosto, …

- Automata for languages with scoping

Distefano *etal*

  - Dynamic (de)allocation of registers

`dblp:` Kurz, Suzuki, Tuosto

- History-Register Automata
  - Registers → histories

`dblp:` Grigore, T.

- Automata over infinite alphabets (~ XML)

`dblp:` Bojanczyk, Demri, Figueira, Lasota, Lazic, Segoufin, …

Nominal automata in program verification

- Algorithmic game semantics
- Pushdown analyses (fresh RA's!)
  - Pointers
  - Concurrent processes
  - Reachability, MSO-decidability

`dblp:` Atig, Bouajjani, Fratani, Qadeer, Bollig *etal*

- Runtime Java verification
  - TOPL (register automata)

`dblp:` Grigore *etal*

What's next

- More ML effects out there!
  - *one model to rule them all*
- More languages out there! (e.g. Java)
- Verification
  - Expand algorithmic approach to gen. model checking
  - Program logics from games

What's next

thanks

- More ML effects out there!
  - *one model to rule them all*
- More languages out there! (e.g. Java)
- Verification
  - Expand algorithmic approach to gen. model checking
  - Program logics from games