

Nominal game semantics and automata

Nikos Tzevelekos

Queen Mary University of London

Aarhus, Feb 2015

Supported by a Royal Academy of Engineering Research Fellowship

Computation with names (ML)

type with only value: ()

“pure” names

$\lambda x. \text{ref}() : \text{unit} \rightarrow (\text{unit ref})$

Computation with names (ML)

type with only value: ()

“pure” names

$\lambda x. \text{ref}() : \text{unit} \rightarrow (\text{unit ref})$

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

Computation with names (ML)

type with only value: ()

"pure" names

$\lambda x. \text{ref}() : \text{unit} \rightarrow (\text{unit ref})$

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

Computation with names (ML)

type with only value: ()

"pure" names

$\lambda x. \text{ref}() : \text{unit} \rightarrow (\text{unit ref})$

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

$(\lambda x. \text{ref}()) () == (\lambda x. \text{ref}()) ()$

Computation with names (ML)

type with only value: ()

"pure" names

$\lambda x. \text{ref}() : \text{unit} \rightarrow (\text{unit ref})$

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

$(\lambda x. \text{ref}()) () == (\lambda x. \text{ref}()) ()$

false

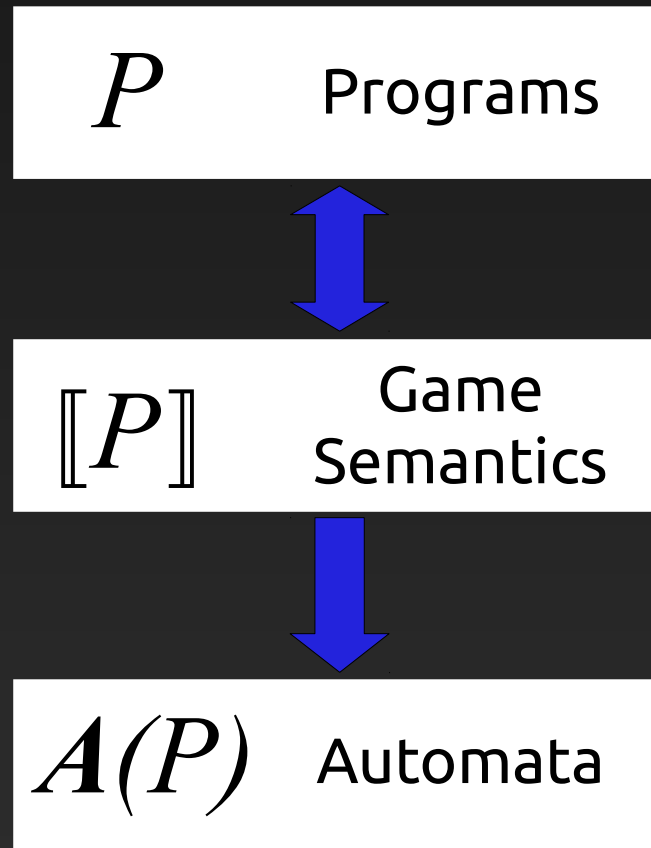
What this talk is about

We give an overview of techniques for modelling and analysing **computation with names**

Names are an abstraction for handling **dynamic resources**: *references, objects, channels, exceptions...*

Our approach starts denotationally, using **game semantics**, and reaches concrete algorithms using **automata over infinite alphabets**

Overview



Example

Call-by-value language with unit references

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

$$P \cong P'$$

same observable behaviour in every context

Example

Call-by-value language with unit references

$$\lambda x. \text{ref}() \not\cong \text{let } y = \text{ref}() \text{ in } \lambda x. y : \text{unit} \rightarrow \text{unit ref}$$
$$\text{let } y = \text{ref}() \text{ in } \lambda x. (x == y) \cong \lambda x. \text{false} : \text{unit} \rightarrow \text{bool}$$

$$P \cong P'$$

same observable behaviour in every context

Example

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”

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

$\text{let } y = \text{ref}() \text{ in } \lambda x. (x == y) \cong \lambda x. \text{false} : \text{unit} \rightarrow \text{bool}$

$P \cong P'$

same observable behaviour in every context

Quiz

Call-by-value language with **int** references

$g : \text{int ref} \rightarrow \text{unit} \quad \vdash \quad \text{let } x, y = \text{ref}(0) \text{ in } (g x); y := !x; y : \text{int ref}$

vs

$g : \text{int ref} \rightarrow \text{unit} \quad \vdash \quad \text{let } x = \text{ref}(0) \text{ in } (g x); x : \text{int ref}$

Models

How to assign denotations to programs,

$$\llbracket - \rrbracket : \text{Syntax} \longrightarrow \mathcal{M}$$

such that:

$$P \cong P' \iff \llbracket P \rrbracket = \llbracket P' \rrbracket$$

Models

How to assign denotations to programs,

$$\llbracket - \rrbracket : \text{Syntax} \longrightarrow \mathcal{M}$$

such that:

$$P \cong P' \iff \llbracket P \rrbracket = \llbracket P' \rrbracket$$

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

$$T ::= \text{unit} \mid \text{int} \mid T \text{ ref} \mid T \rightarrow T \mid \dots \quad \boxed{\quad} \quad M = M \mid \dots$$

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

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 (\neq Game Theory)
- Programs = *strategies* for P
- *Categories* of games

Games played in arenas

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

Games played in arenas

free variables

program

output type

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

input types

Games played in arenas

free variables

program

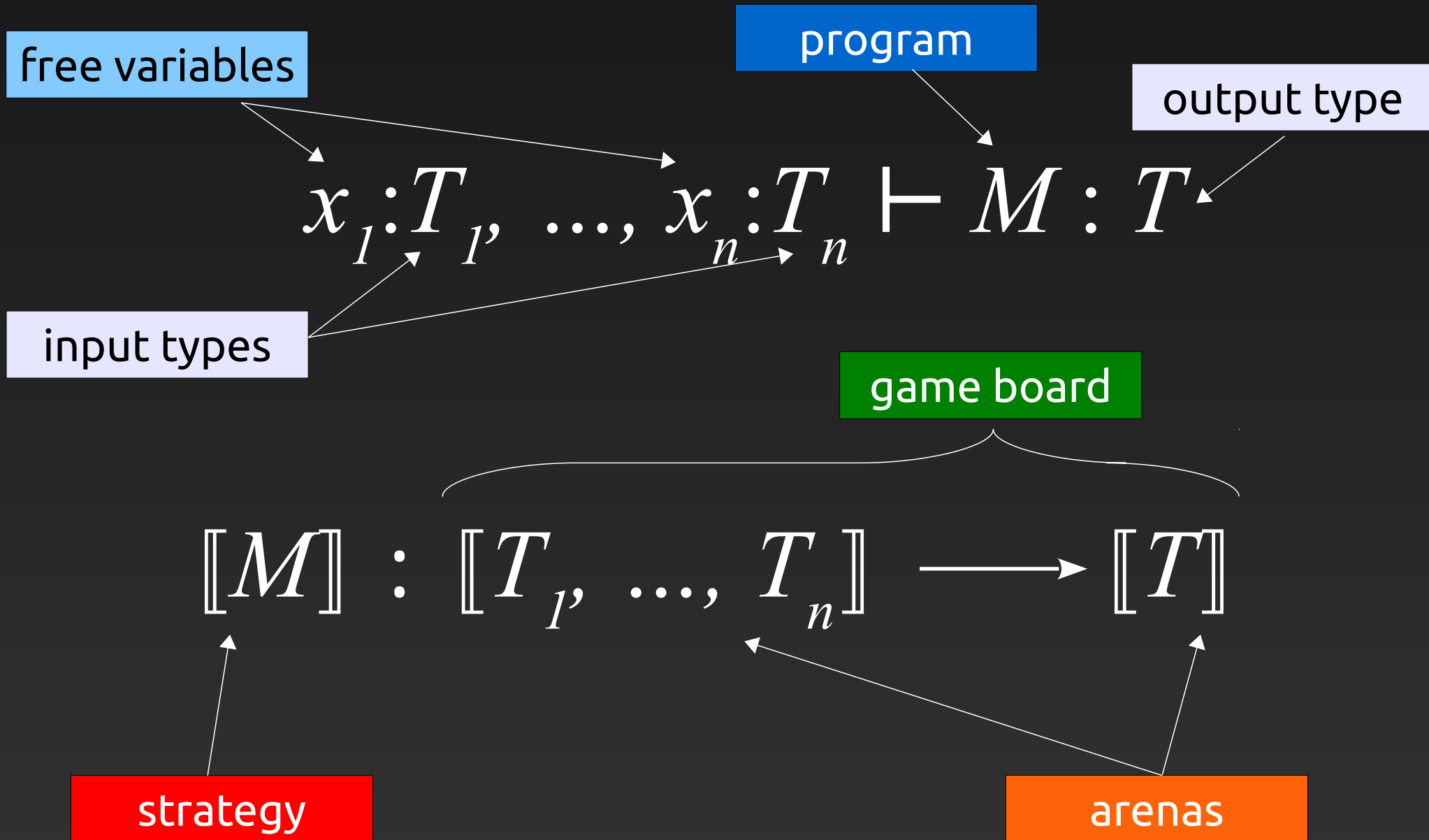
output type

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

input types

$$\llbracket M \rrbracket : \llbracket T_1, \dots, T_n \rrbracket \longrightarrow \llbracket T \rrbracket$$

Games played in arenas



Arenas of moves

$$[[M]] : [[T_1, \dots, T_n]] \longrightarrow [[T]]$$

strategy

arenas

Arenas of moves

$$[[M]] : [[T_1, \dots, T_n]] \longrightarrow [[T]]$$

arenas

moves

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

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

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

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

...

Arenas of moves

$$[[M]] : [[T_1, \dots, T_n]] \longrightarrow [[T]]$$



arenas

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

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

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

\mathcal{N}_T a set of *names*:

- infinitely many
- comparable for equality only

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

Arenas of moves

$$[[M]] : [[T_1, \dots, T_n]] \longrightarrow [[T]]$$



arenas

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

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

$$[[T \text{ ref}]] = \{ a, b, \dots \} = \mathbb{A}_T \quad a, b, \dots \in \mathcal{N}_T$$

\mathcal{N}_T a set of *names*:

- infinitely many
- comparable for equality only

A simple game board

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

(i.e. calls)

questions

$\{ 0, 1, -1, \dots \}$

(i.e. returns)

answers

$\{ 0, 1, -1, \dots \}$

O : Opponent

P : Proponent

A simple game board

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

(i.e. calls)

questions

$\{ 0, 1, -1, \dots \}$

causality

(i.e. returns)

answers

$\{ 0, 1, -1, \dots \}$

O : Opponent

P : Proponent

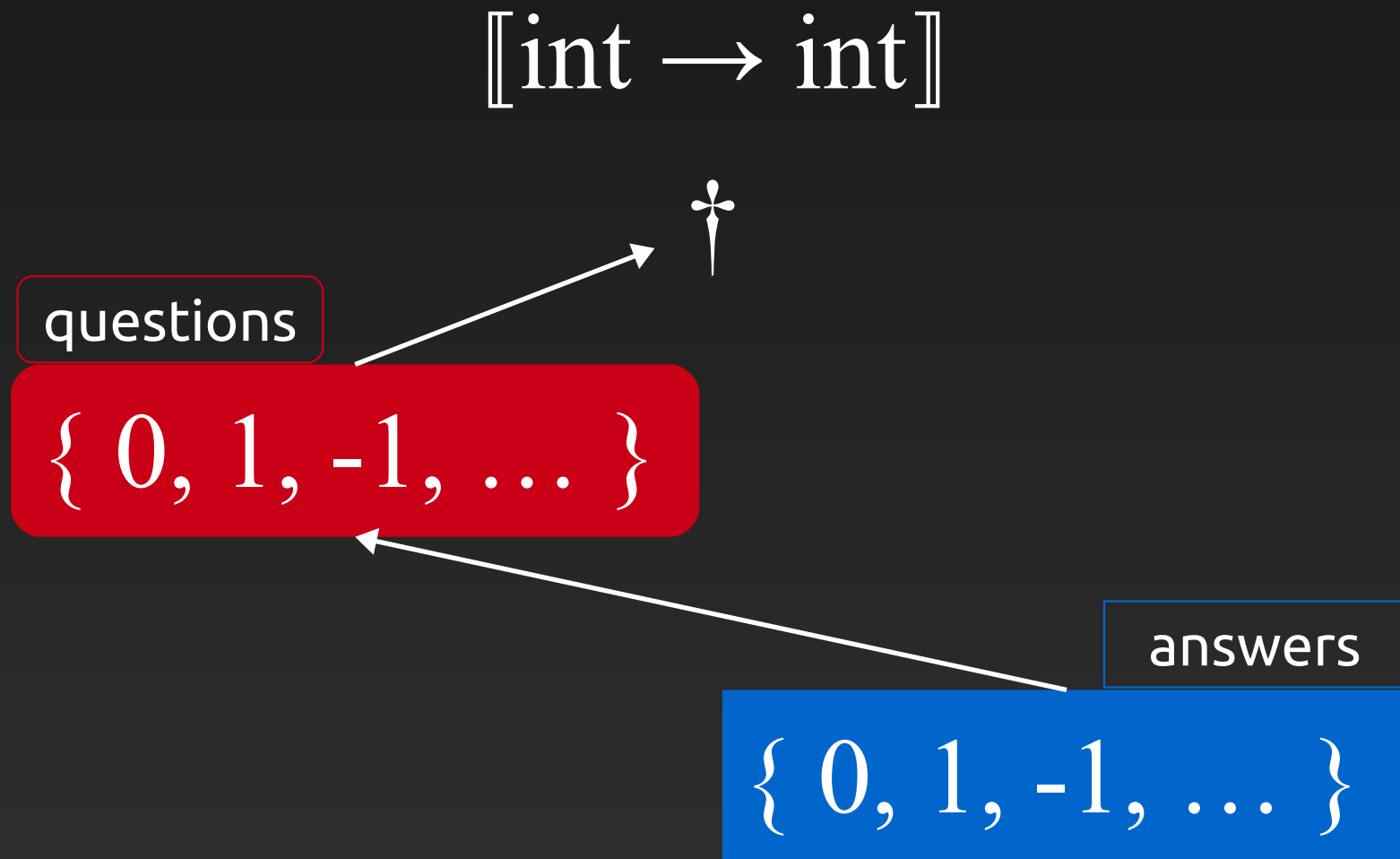
A higher-order arena

$[[\text{int} \rightarrow \text{int}]]$



"here is a function"

A higher-order arena

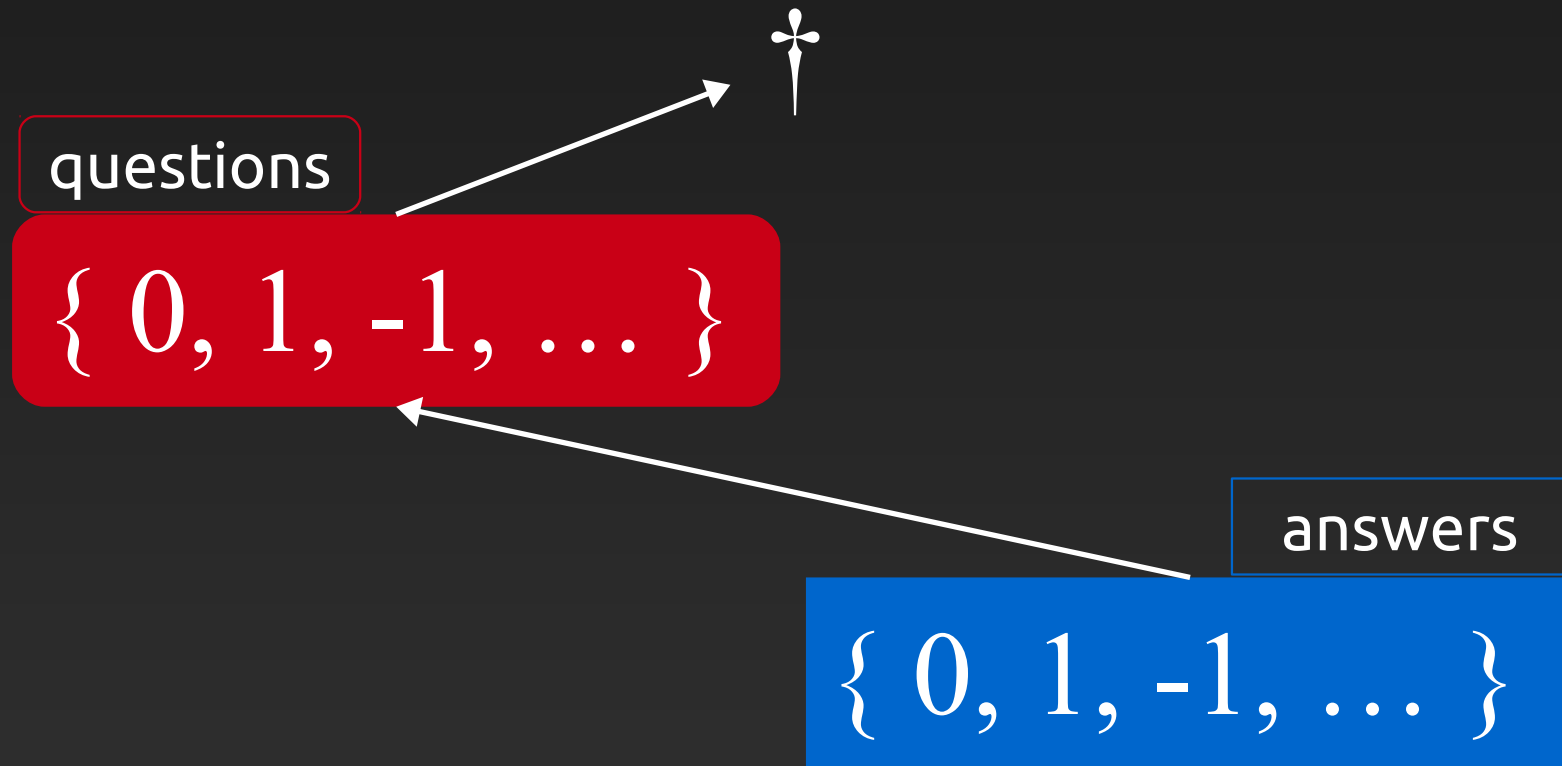


O : Opponent

P : Proponent

A higher-order arena

$$\llbracket \text{int} \rightarrow \text{int} \rrbracket = \mathbb{Z} \Rightarrow \mathbb{Z}$$

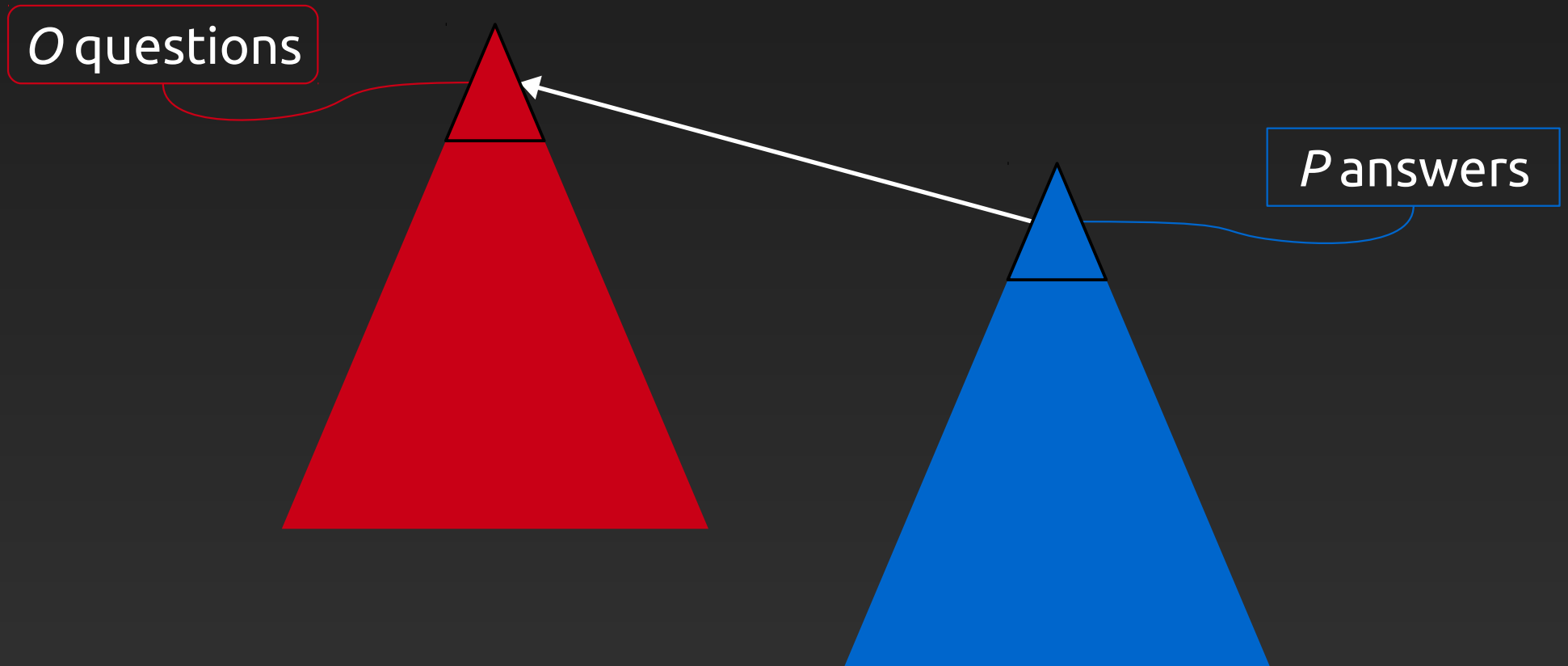


O : Opponent

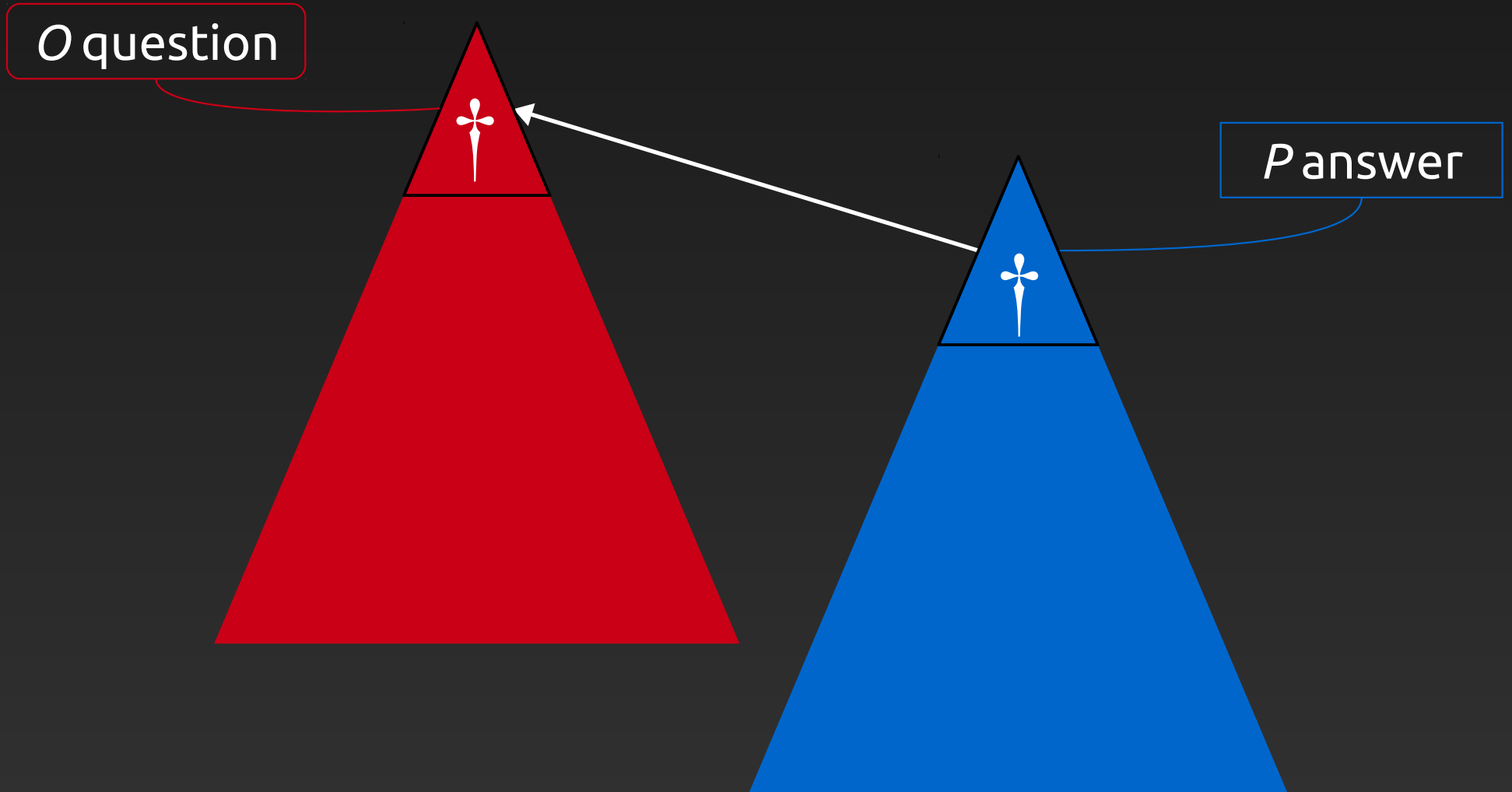
P : Proponent

A higher-order board

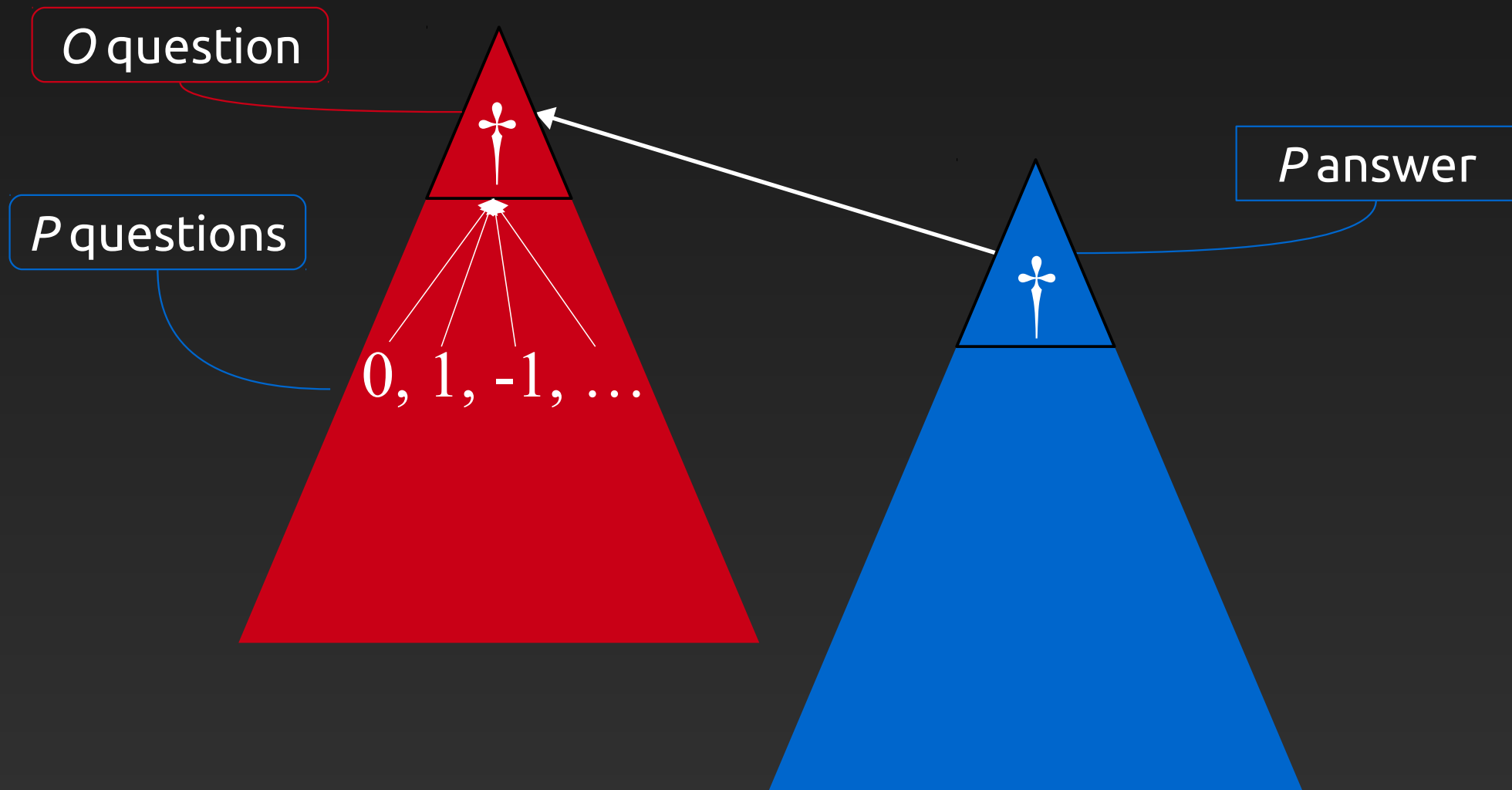
$$[T] \longrightarrow [T']$$



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



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



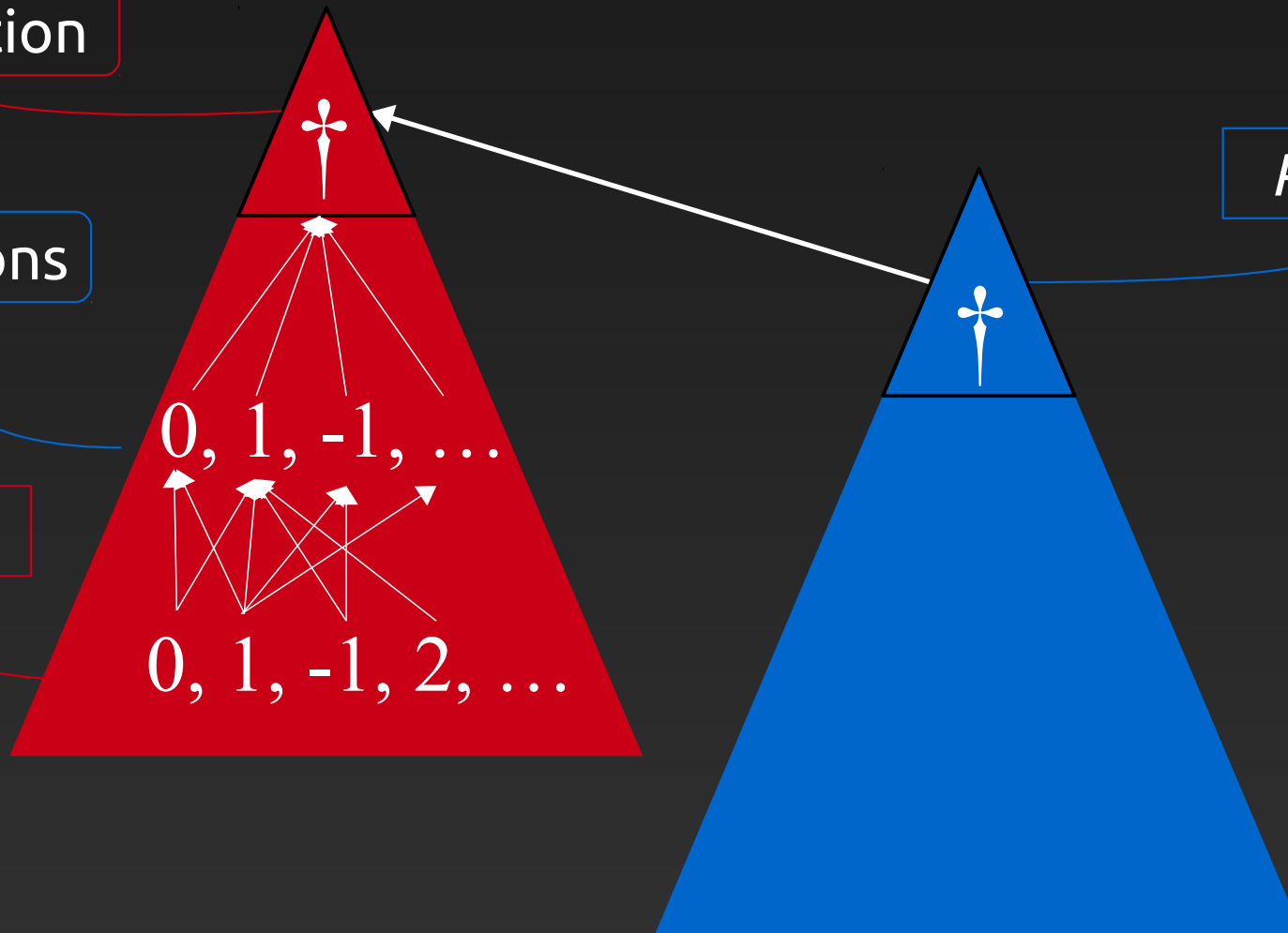
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

O question

P questions

O answers

P answer

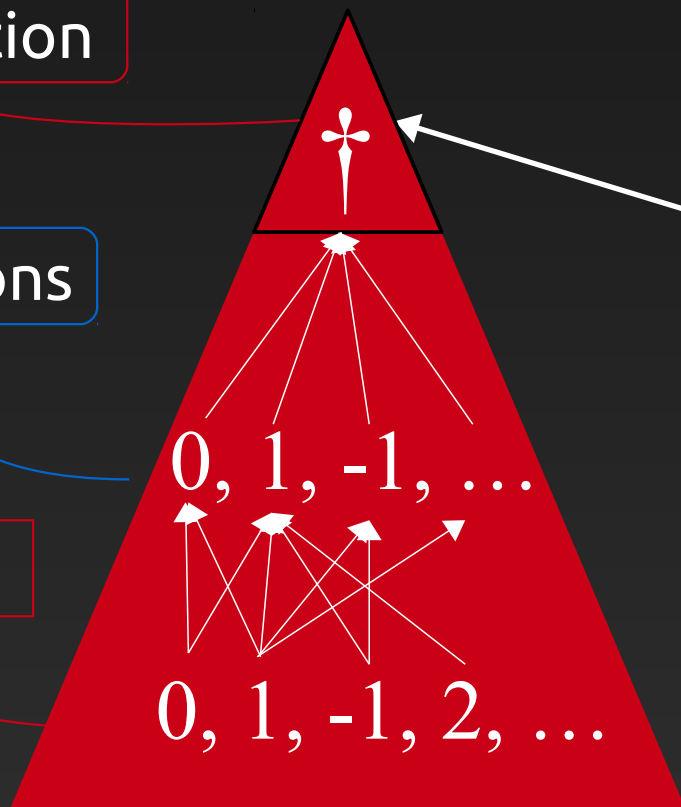


$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

O question

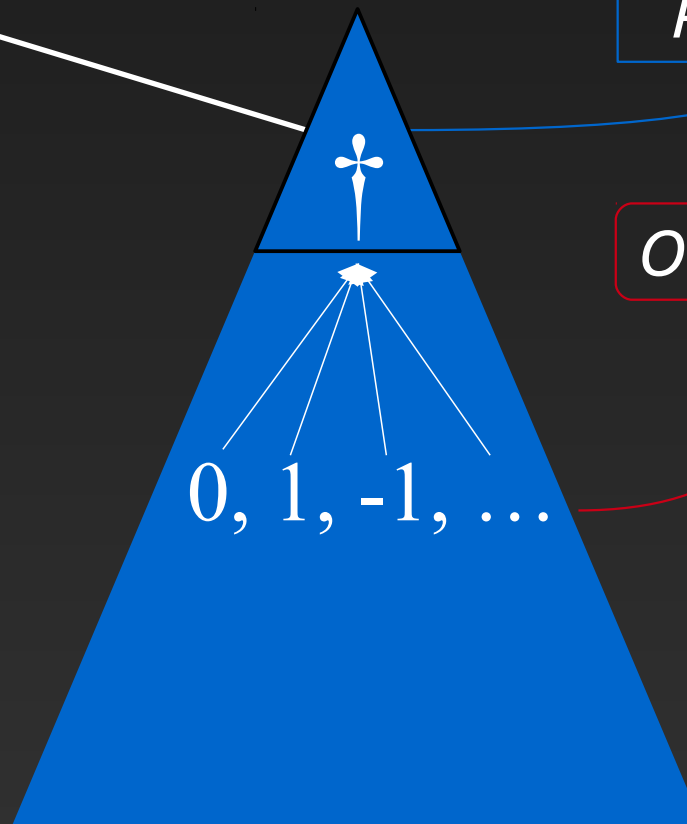
P questions

O answers



P answer

O questions

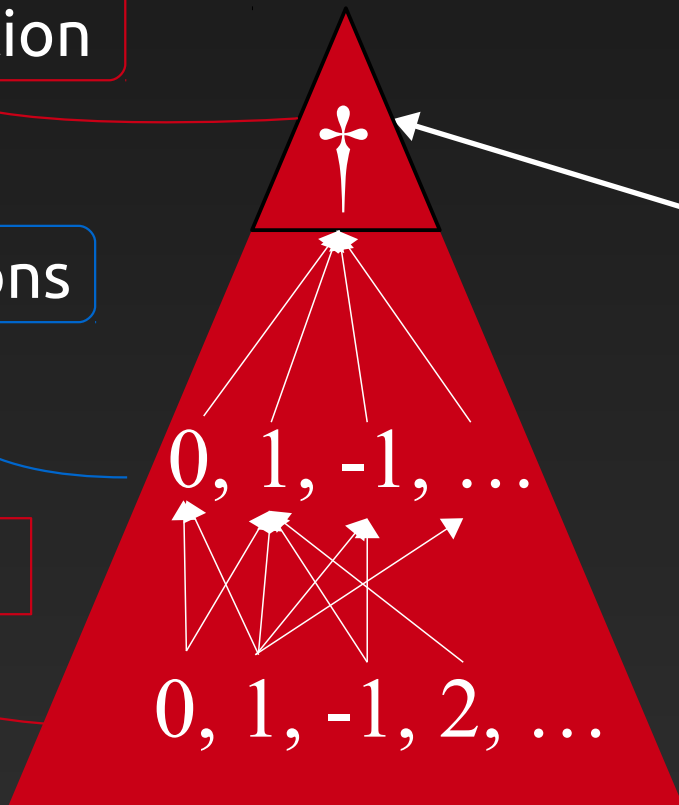


$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

O question

P questions

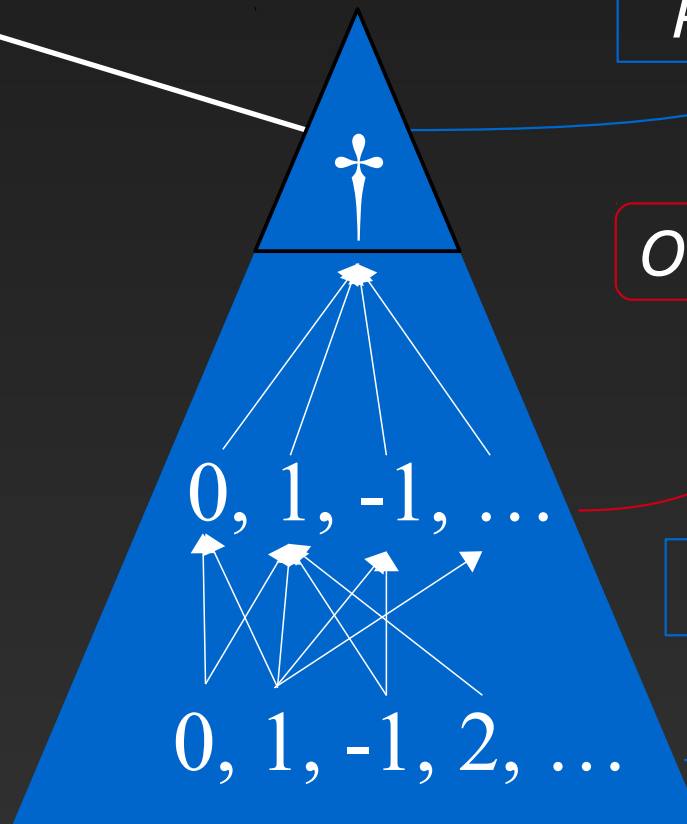
O answers



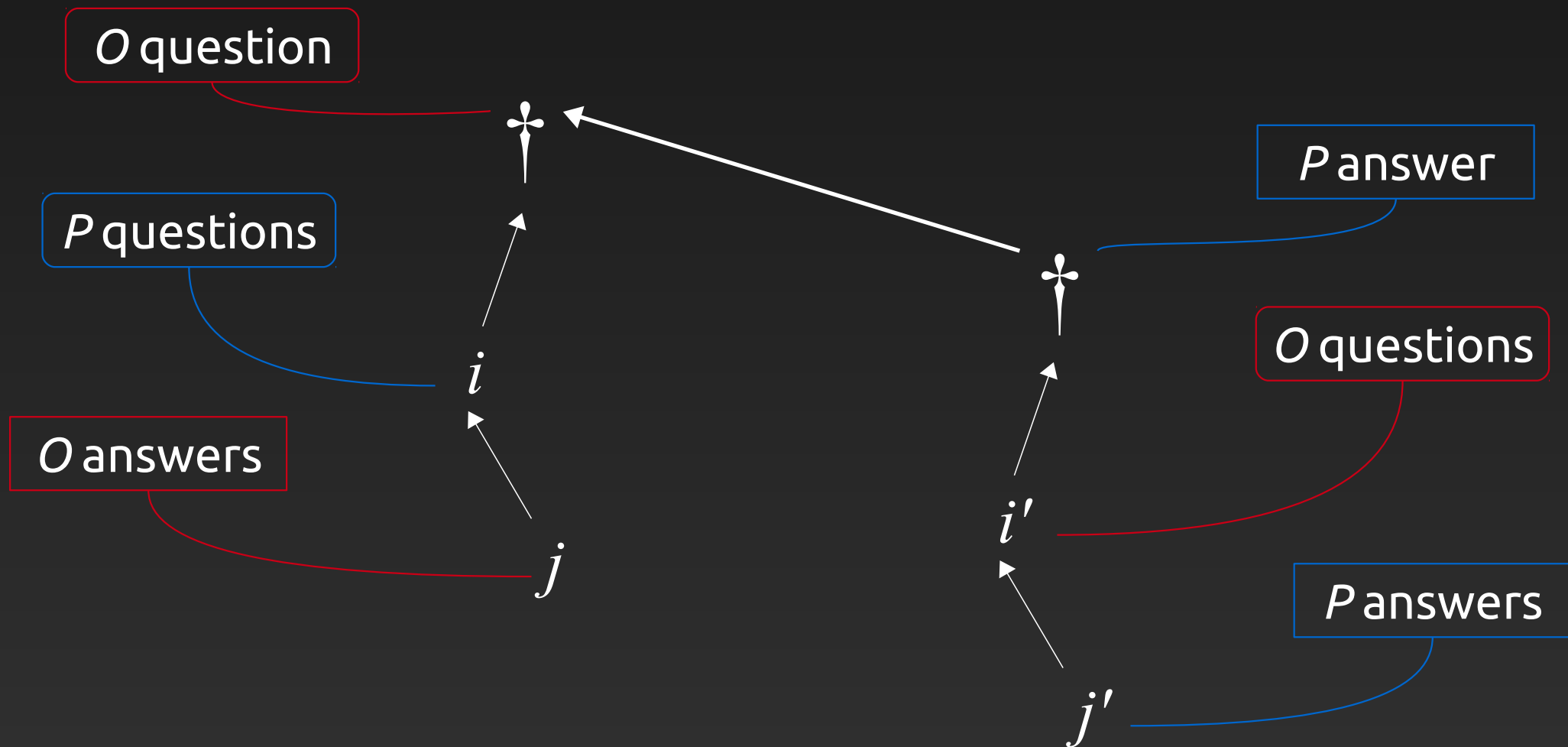
P answer

O questions

P answers

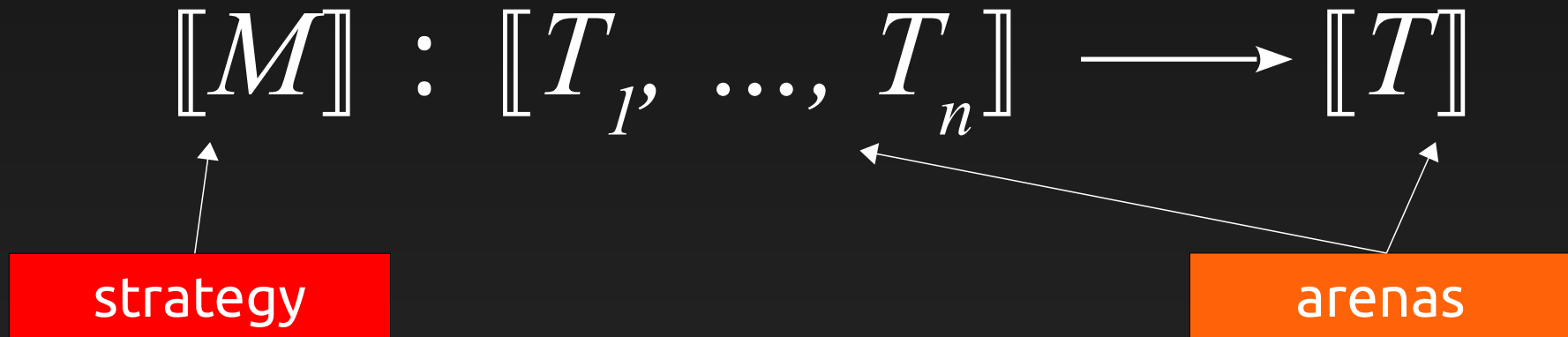


$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$



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

Strategies



Strategies are “instructions” for P on how to play on a given game board

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

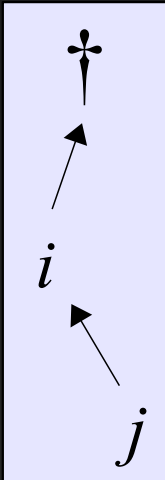
Example

$$f : \text{int} \rightarrow \text{int} \vdash \lambda x. f x + 1 : \text{int} \rightarrow \text{int}$$
$$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$$

Example

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

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



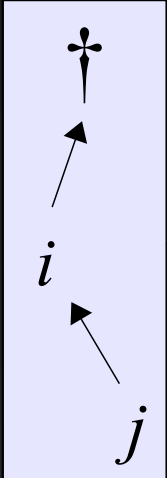
Example

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

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

\dagger

O, Q



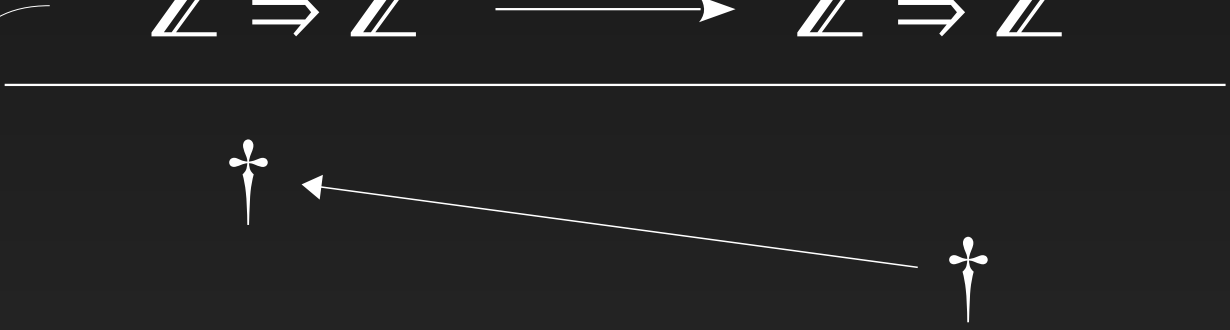
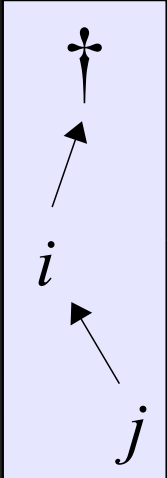
Example

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

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

O, Q

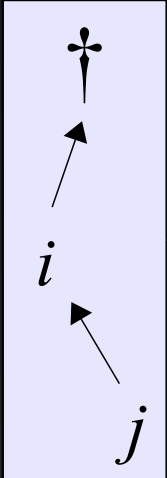
P, A



Example

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

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



⊗

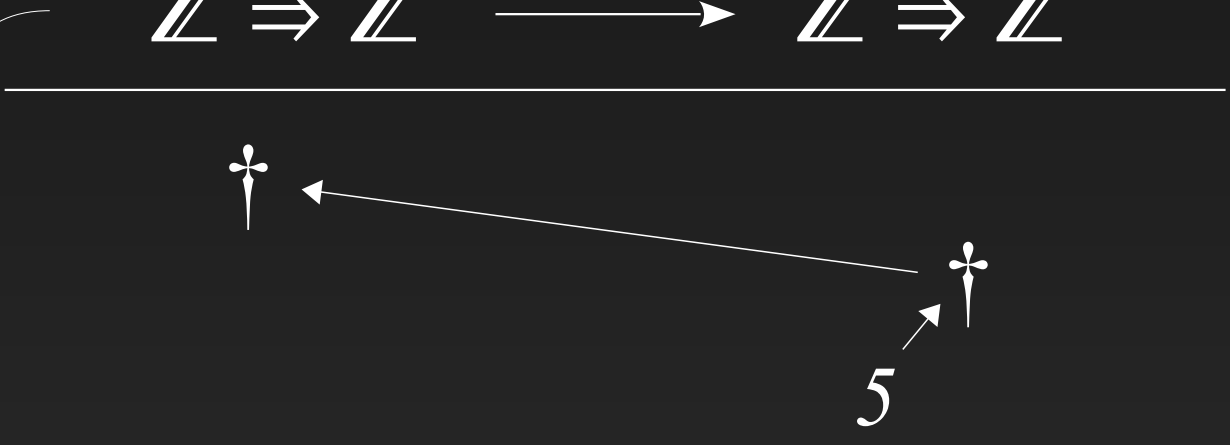
O, Q

⊗

P, A

5

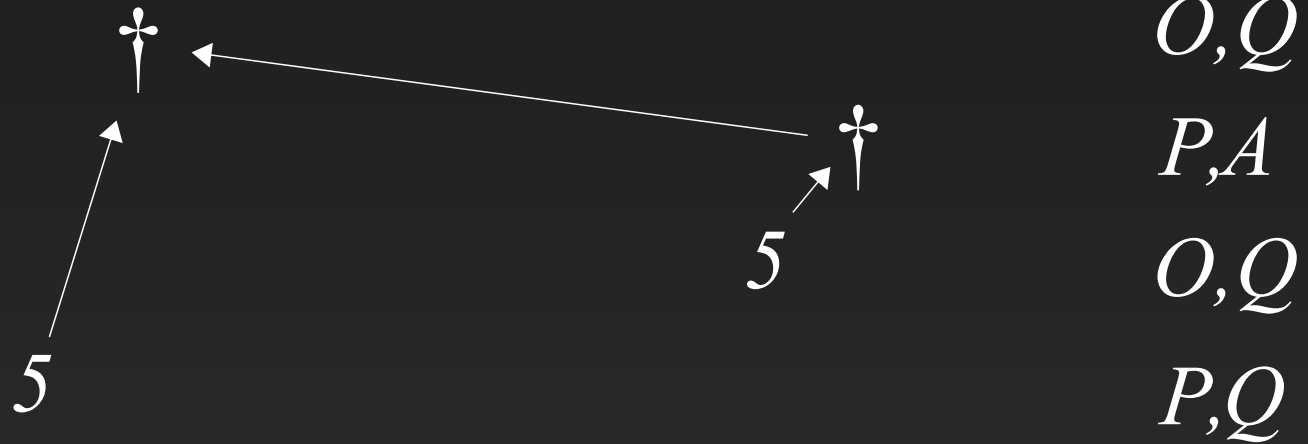
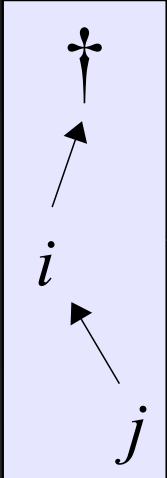
O, Q



Example

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

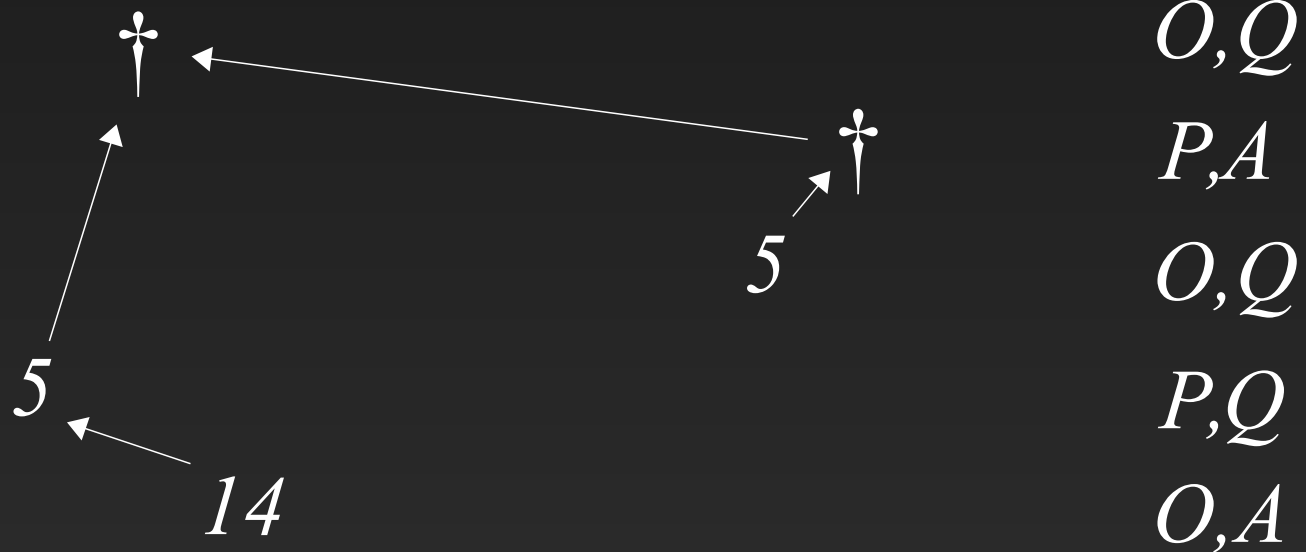
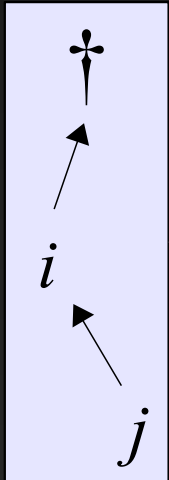
$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



Example

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

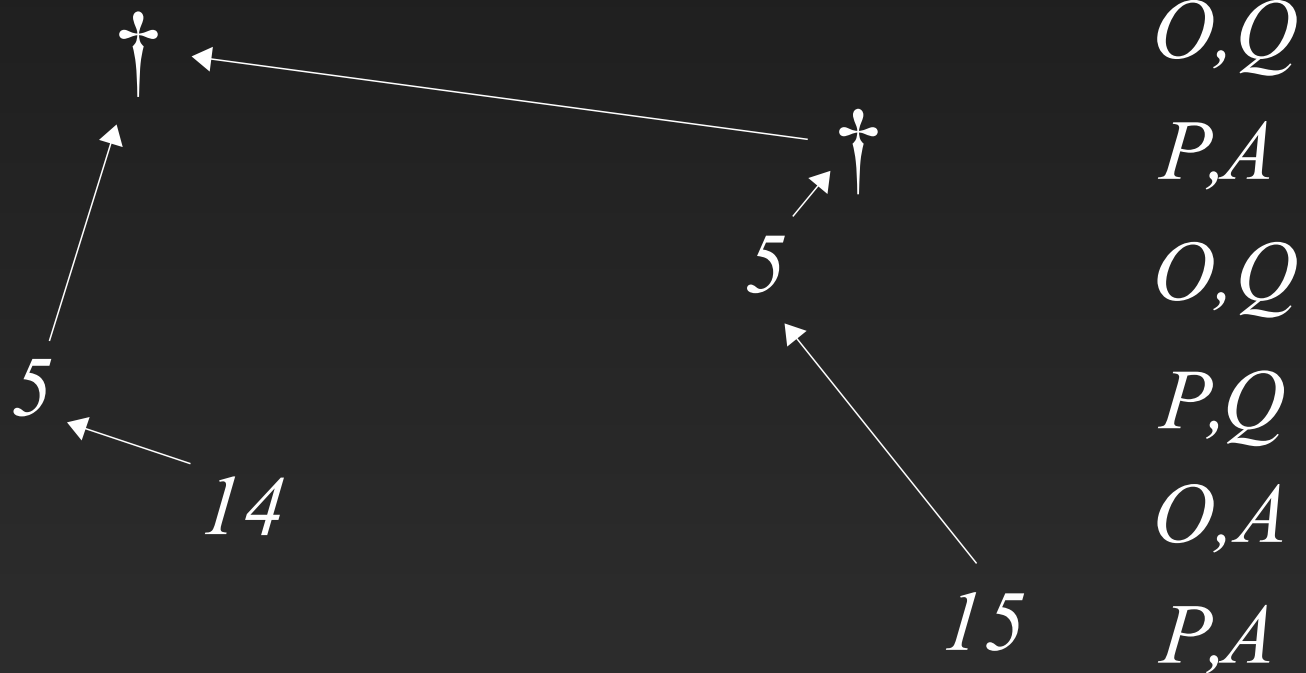
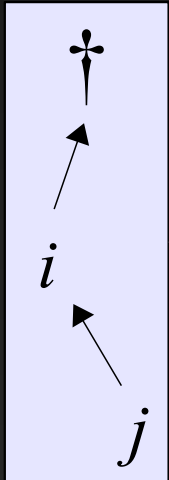
$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



Example

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

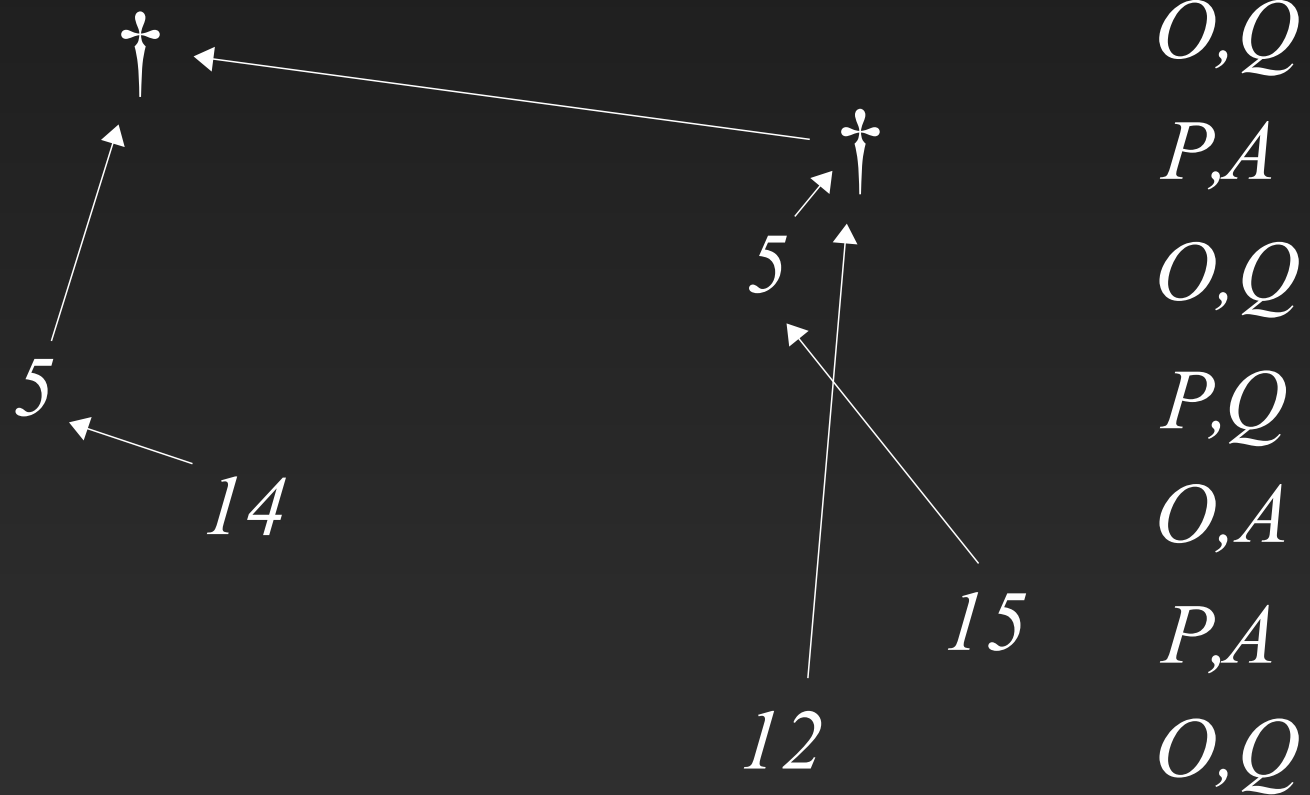
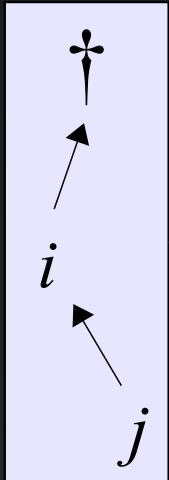
$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



Example

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

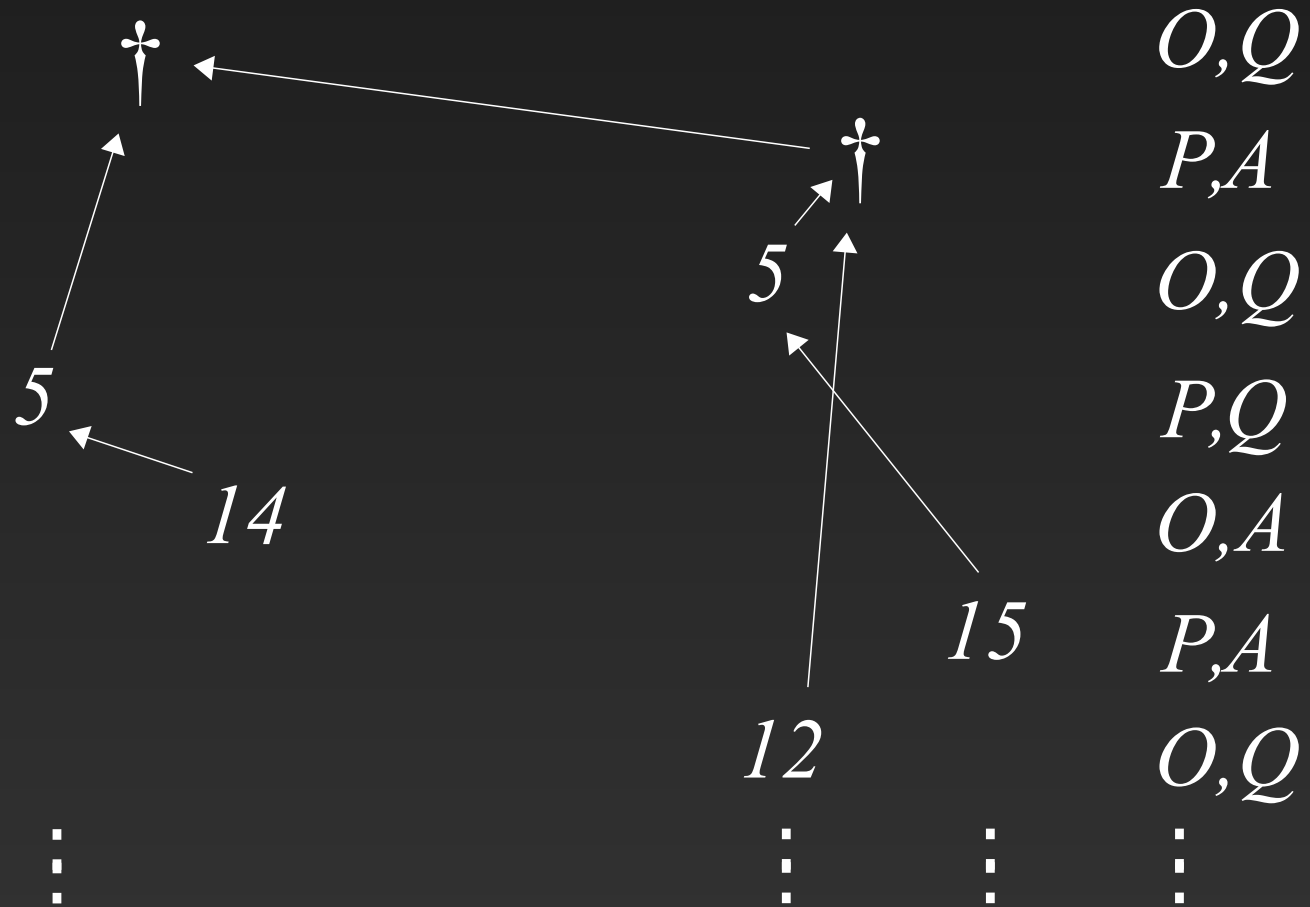
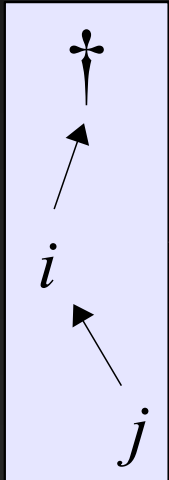
$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



Example

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

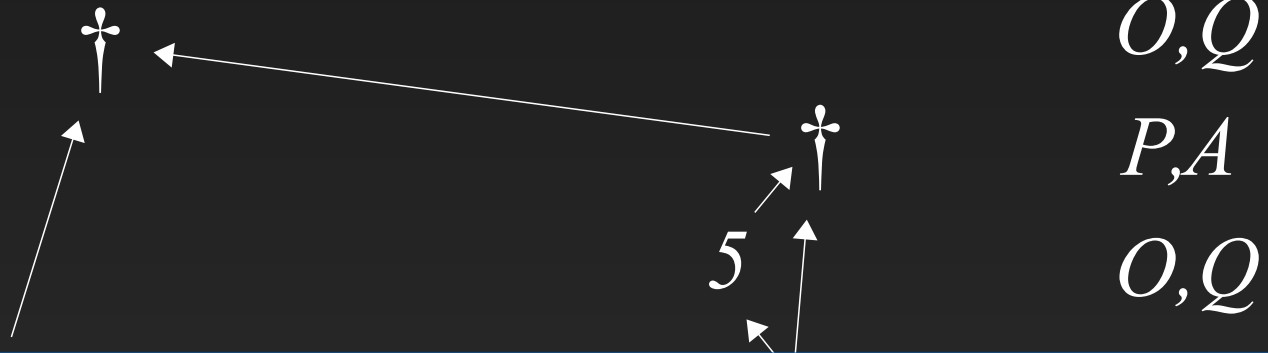
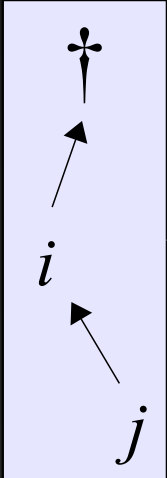
$\mathbb{Z} \Rightarrow \mathbb{Z} \longrightarrow \mathbb{Z} \Rightarrow \mathbb{Z}$



Example

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

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



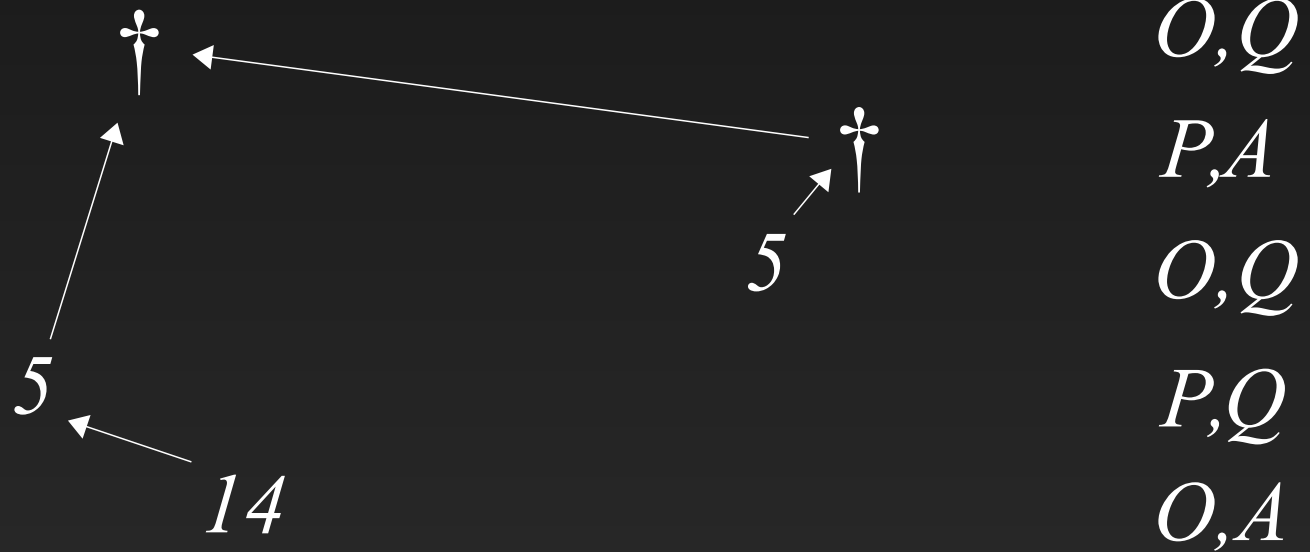
$[[\lambda x. f x + 1]] = \{ \dagger \quad \dagger \quad 5 \quad 5 \quad 14 \quad 15 \dots \}$

$OQ \quad PA \quad OQ \quad PQ \quad OA \quad PA$

$\vdots \quad \vdots \quad \vdots \quad \vdots$

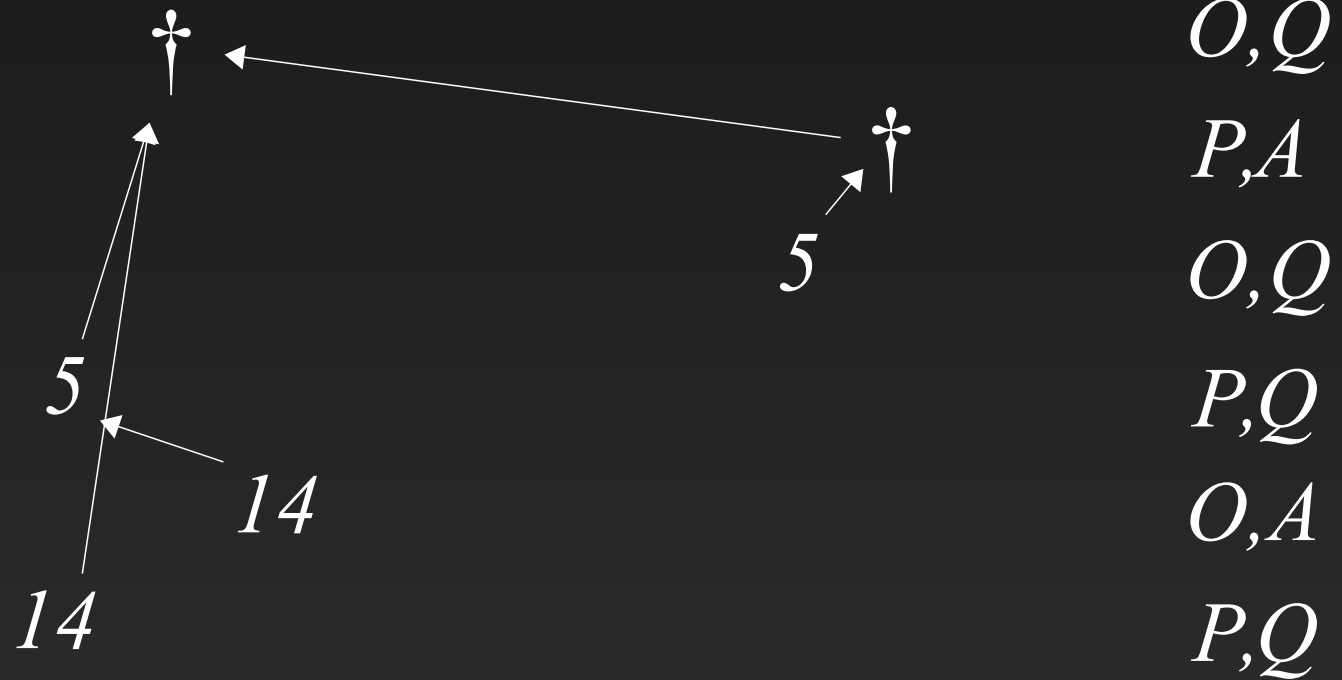
Another example

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



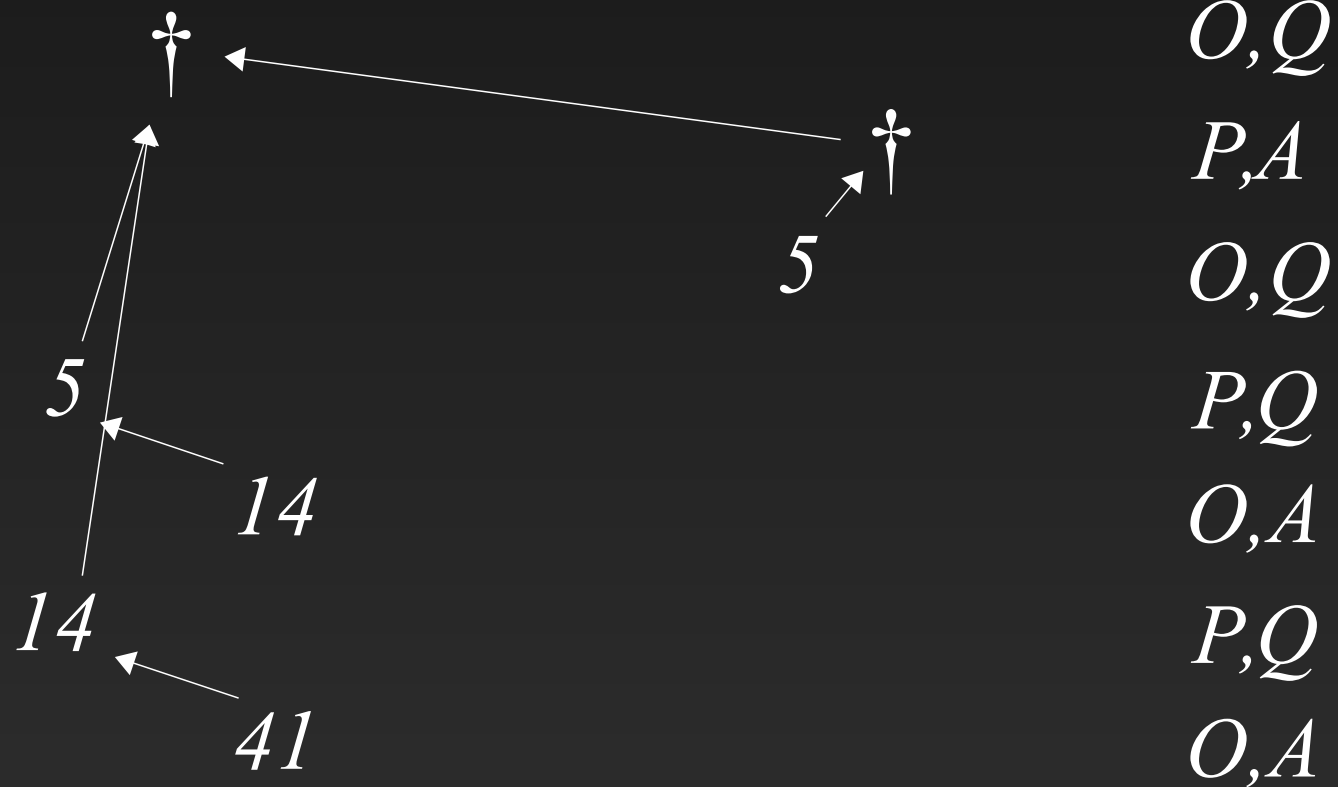
Another example

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



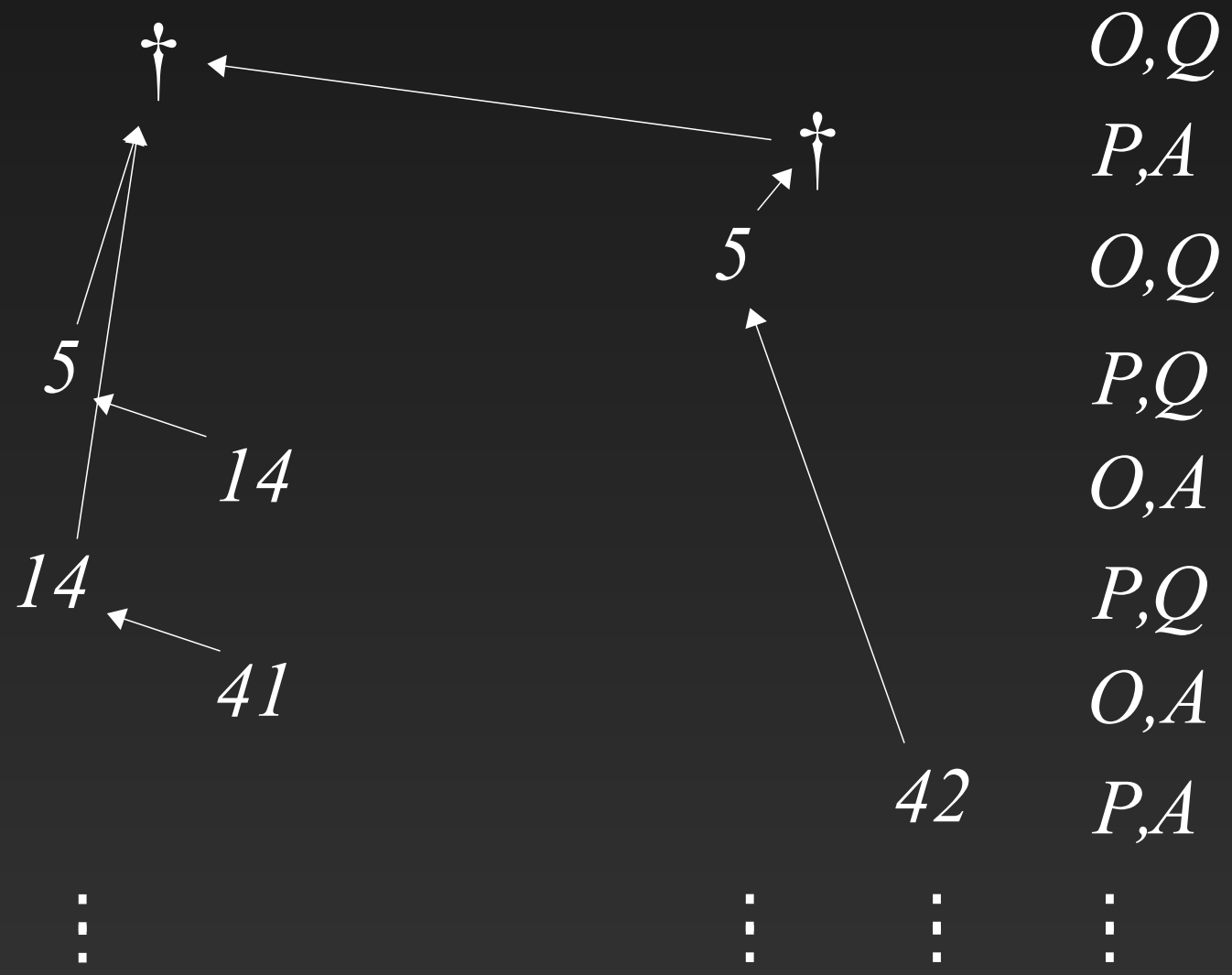
Another example

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



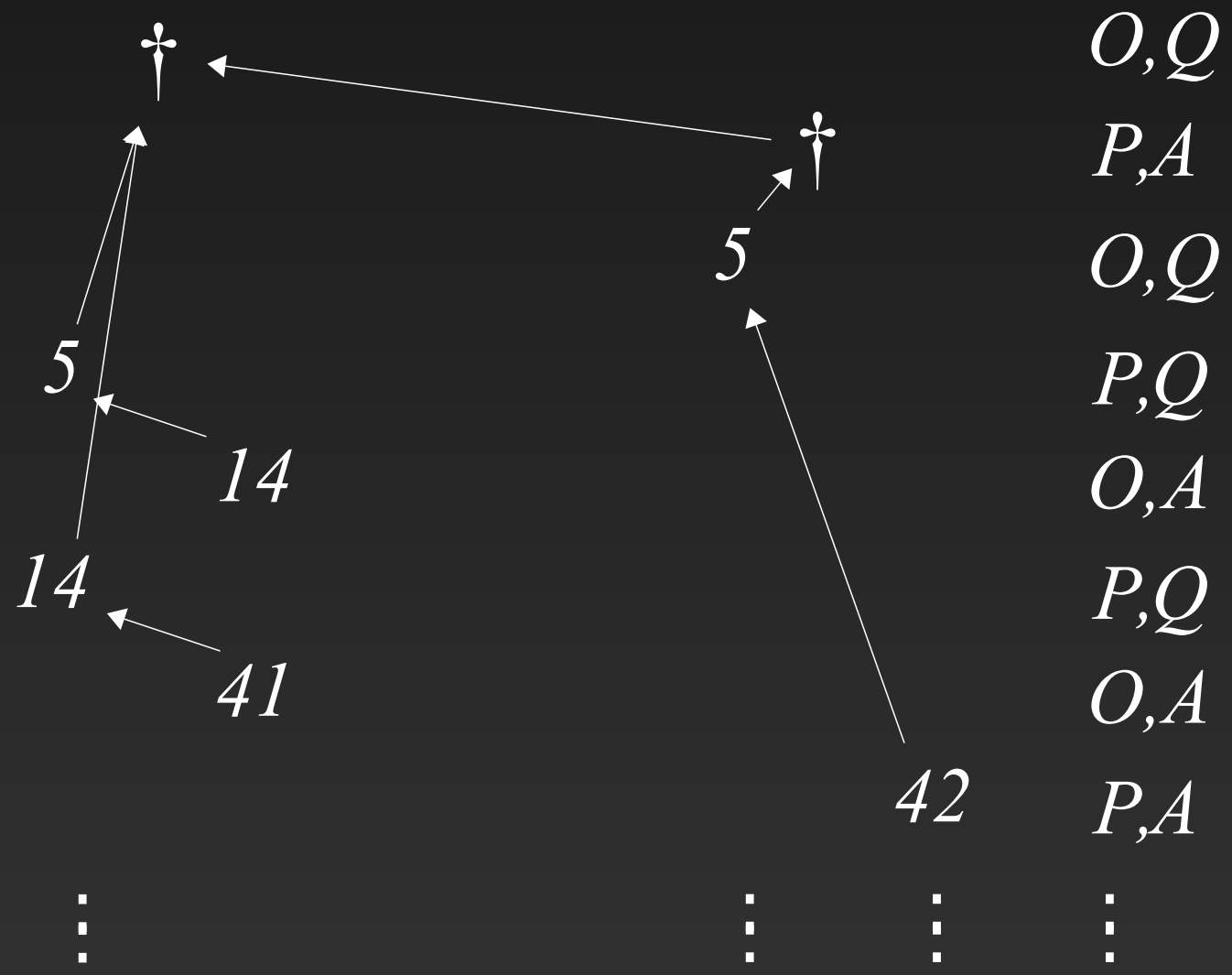
Another example

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



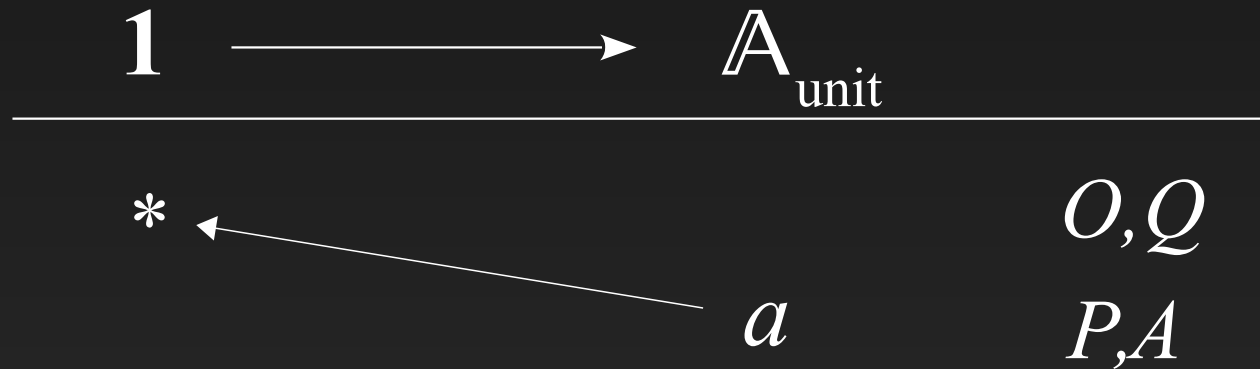
$f : \text{int} \rightarrow \text{int} \vdash \lambda x. f(f(x)) + 1 : \text{int} \rightarrow \text{int}$

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



Nominal games

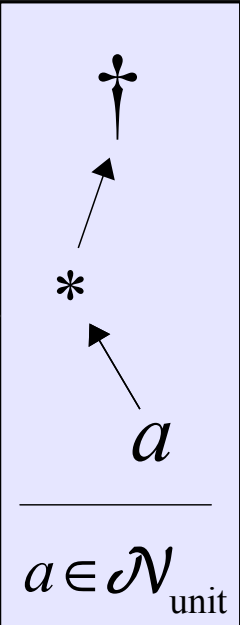
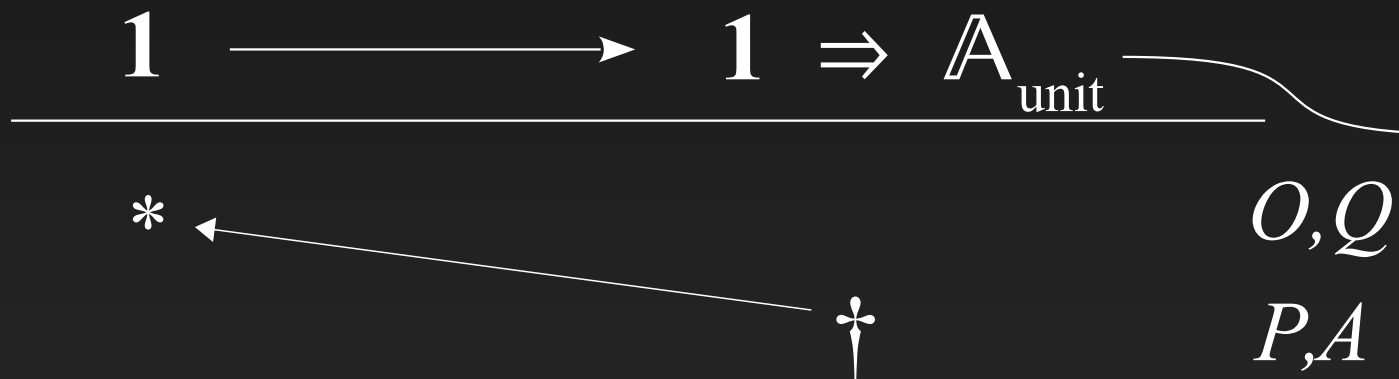
$\vdash \text{ref}() : \text{unit ref}$



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

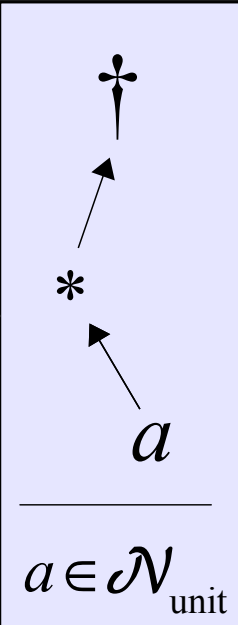
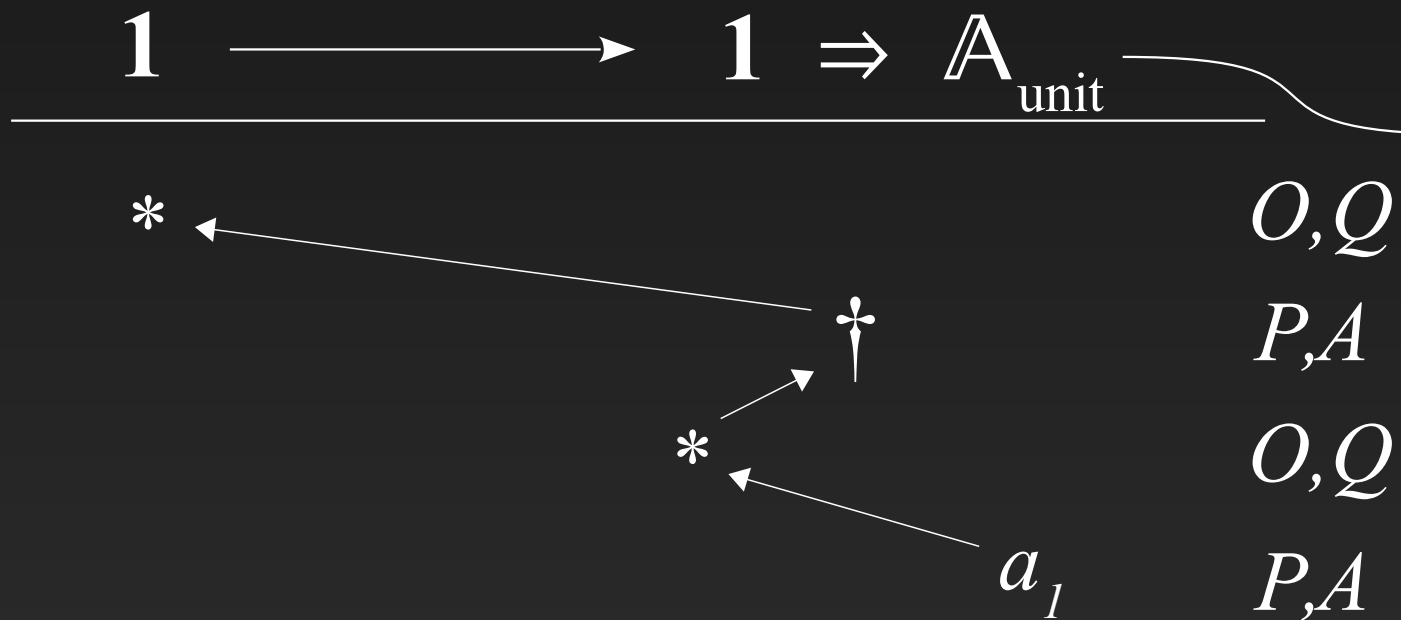
Nominal games (2)

$\vdash \lambda x. \text{ref}() : \text{unit} \rightarrow \text{unit ref}$



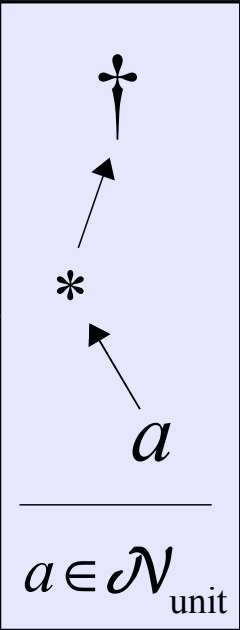
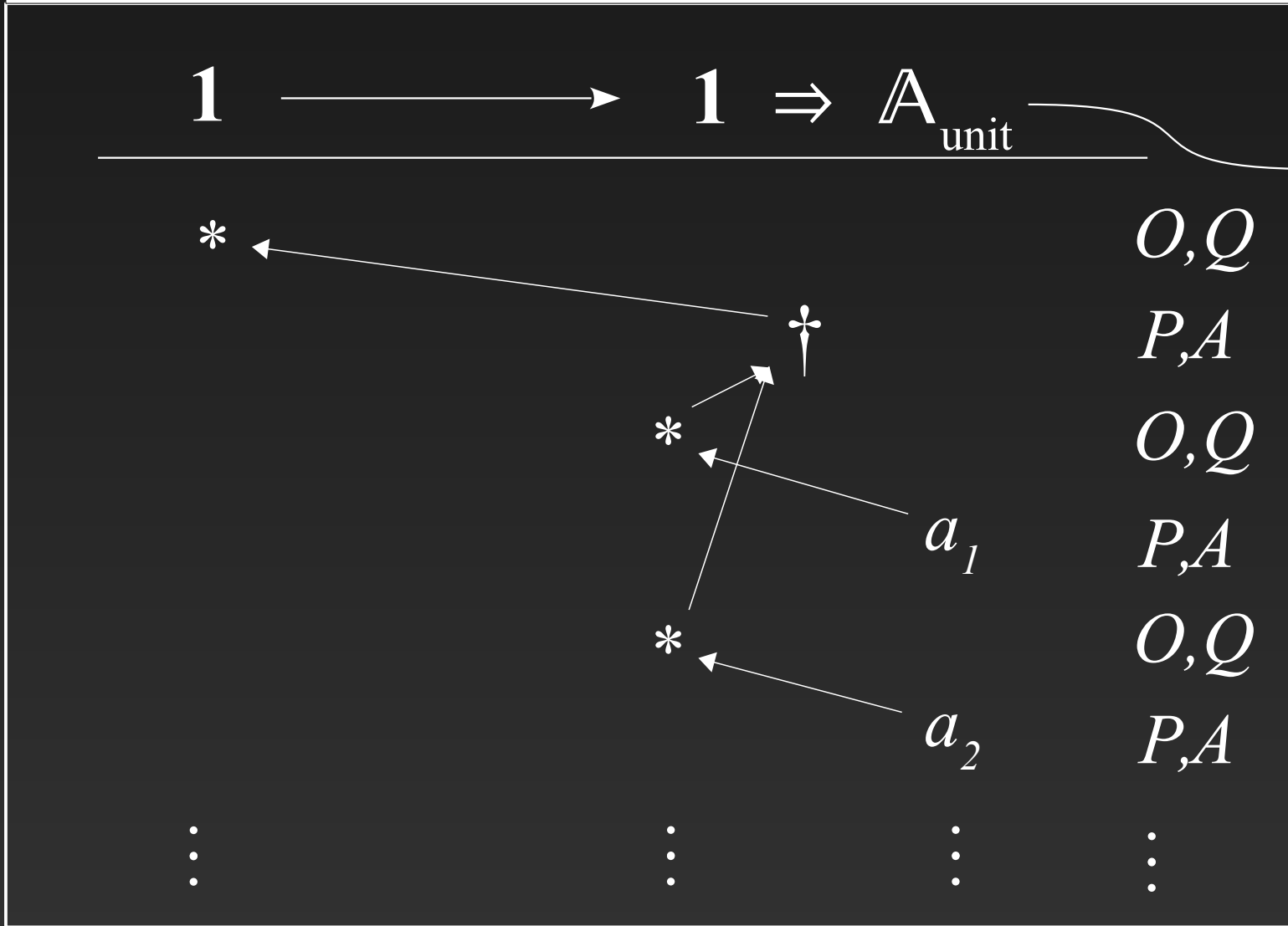
Nominal games (2)

$\vdash \lambda x. \text{ref}() : \text{unit} \rightarrow \text{unit ref}$



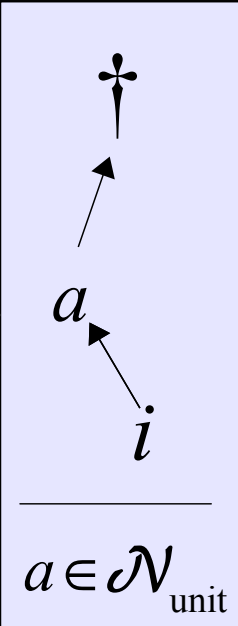
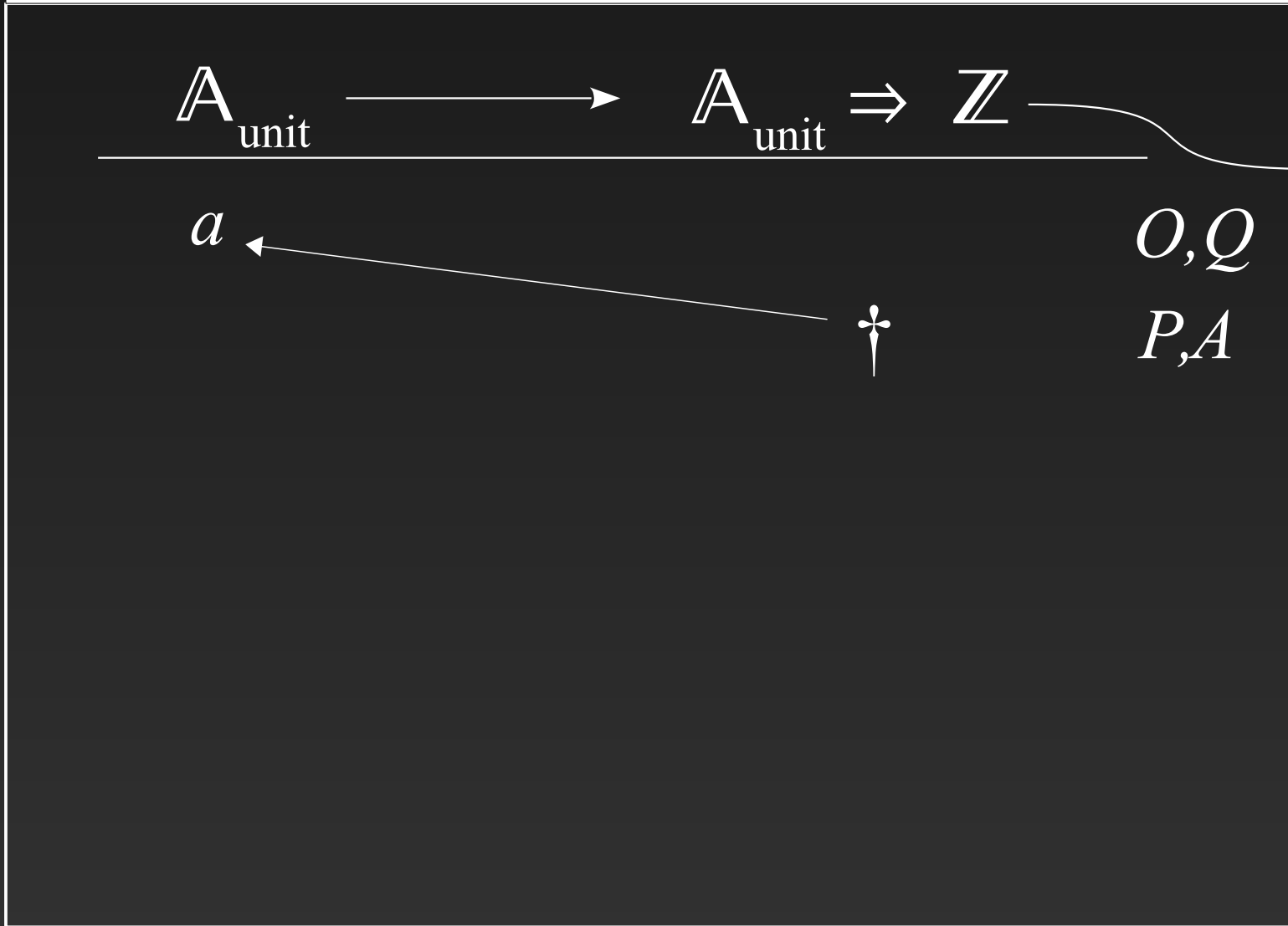
Nominal games (2)

$\vdash \lambda x. \text{ref}() : \text{unit} \rightarrow \text{unit ref}$



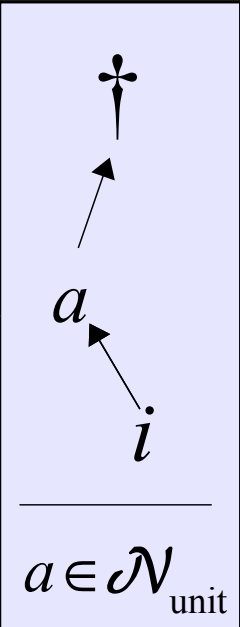
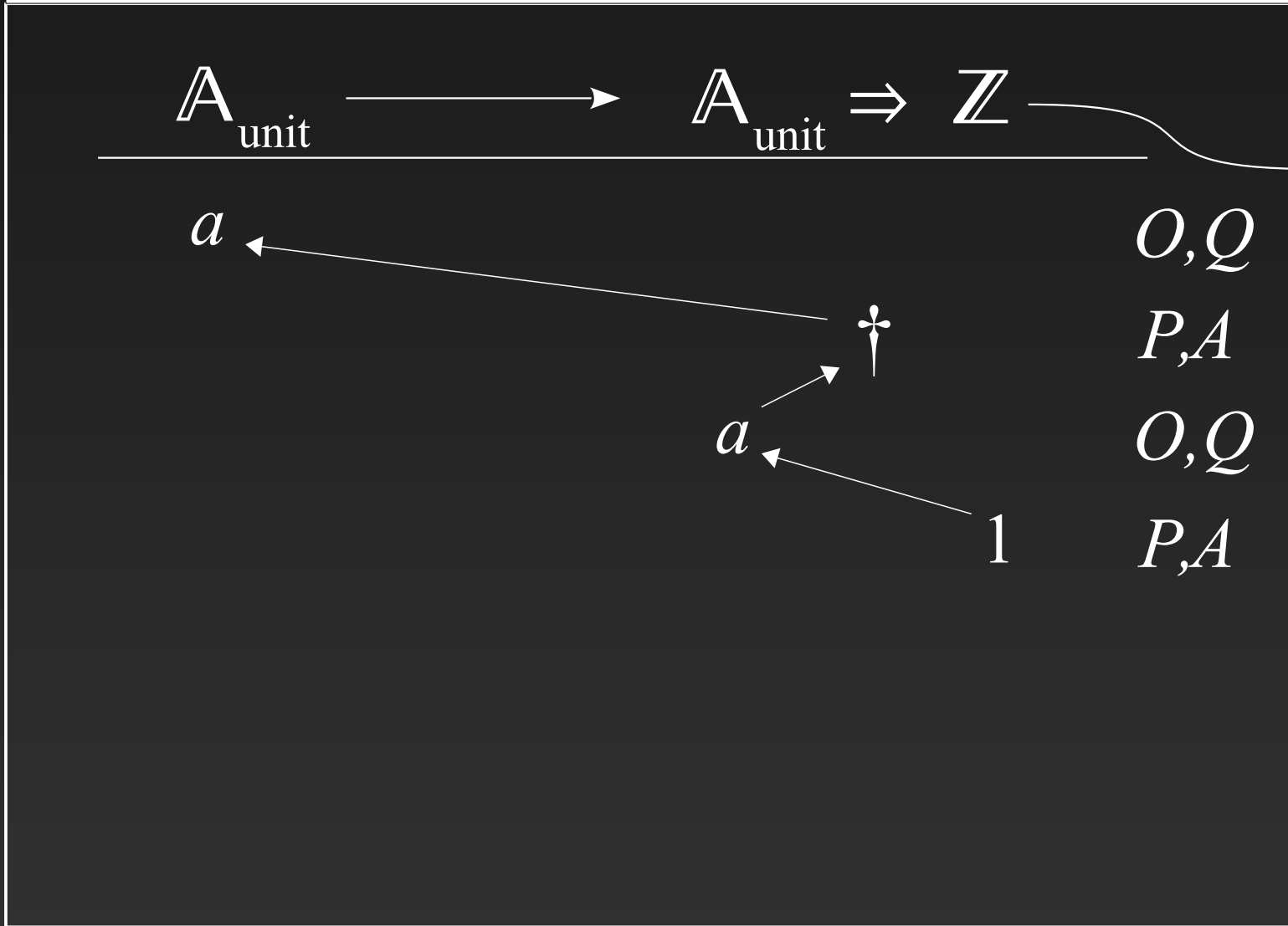
Nominal games (3)

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$



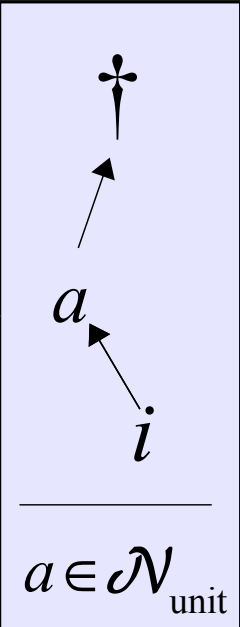
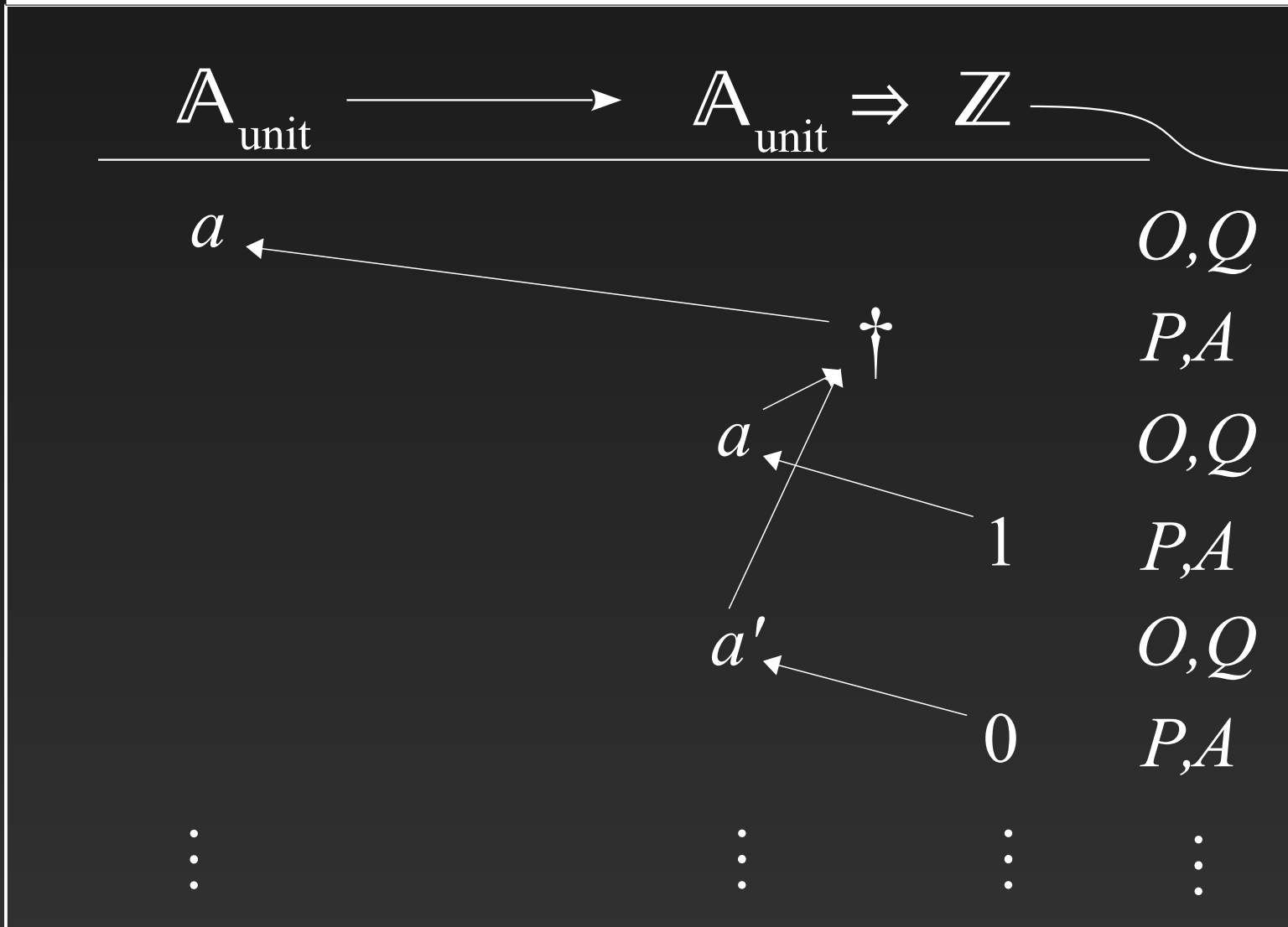
Nominal games (3)

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

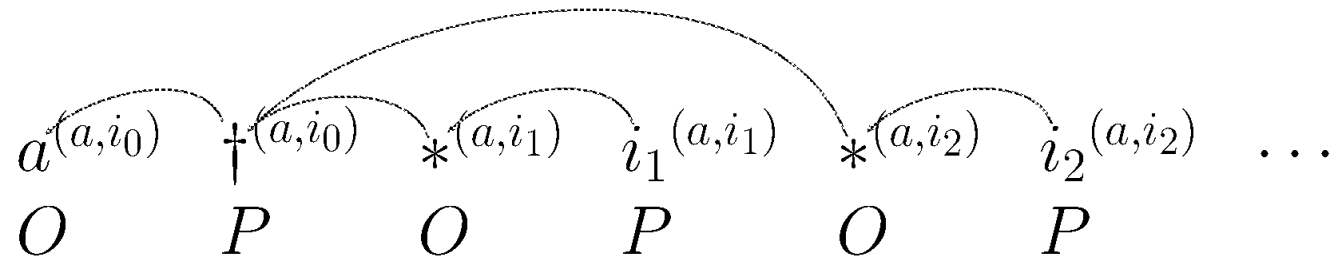


Nominal games (3)

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

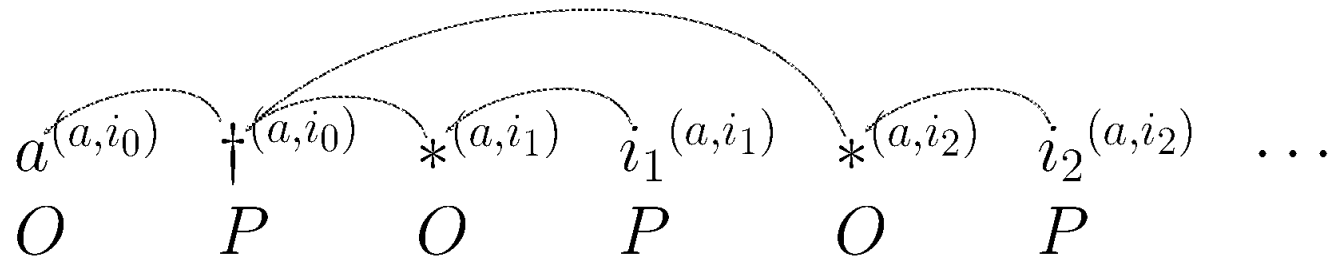


Nominal games: explicit store

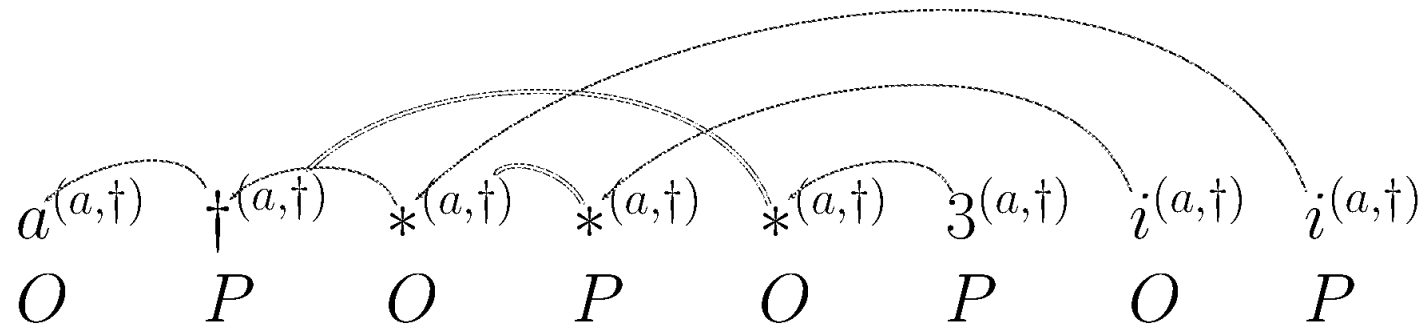


$$= \llbracket x : \text{int ref} \vdash \lambda y. !x : \text{unit} \rightarrow \text{int} \rrbracket : \mathbb{A}_{\text{int}} \rightarrow (1 \Rightarrow \mathbb{Z})$$

Nominal games: explicit store



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



$$= \llbracket x : (\text{unit} \rightarrow \text{int}) \text{ ref} \vdash x := \lambda y. 3; \lambda y. (!x)() \rrbracket : \mathbb{A}_{\text{unit} \rightarrow \text{int}} \rightarrow (1 \Rightarrow \mathbb{Z})$$

In LaTeX

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 $\lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\}$,
- a nominal *justification* relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$;

satisfying the conditions:

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

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

In LaTeX

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

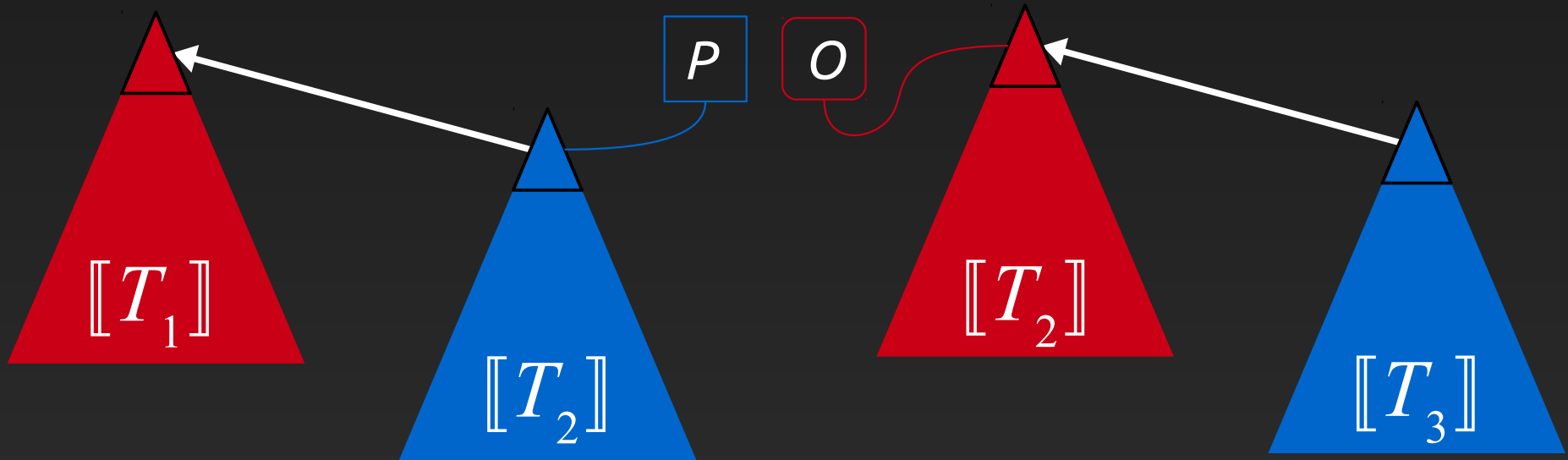
- O/P alternation,
- justification,
- well-bracketing,
- frugality (stores contain only visible/reachable names),
- ...

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

Composition

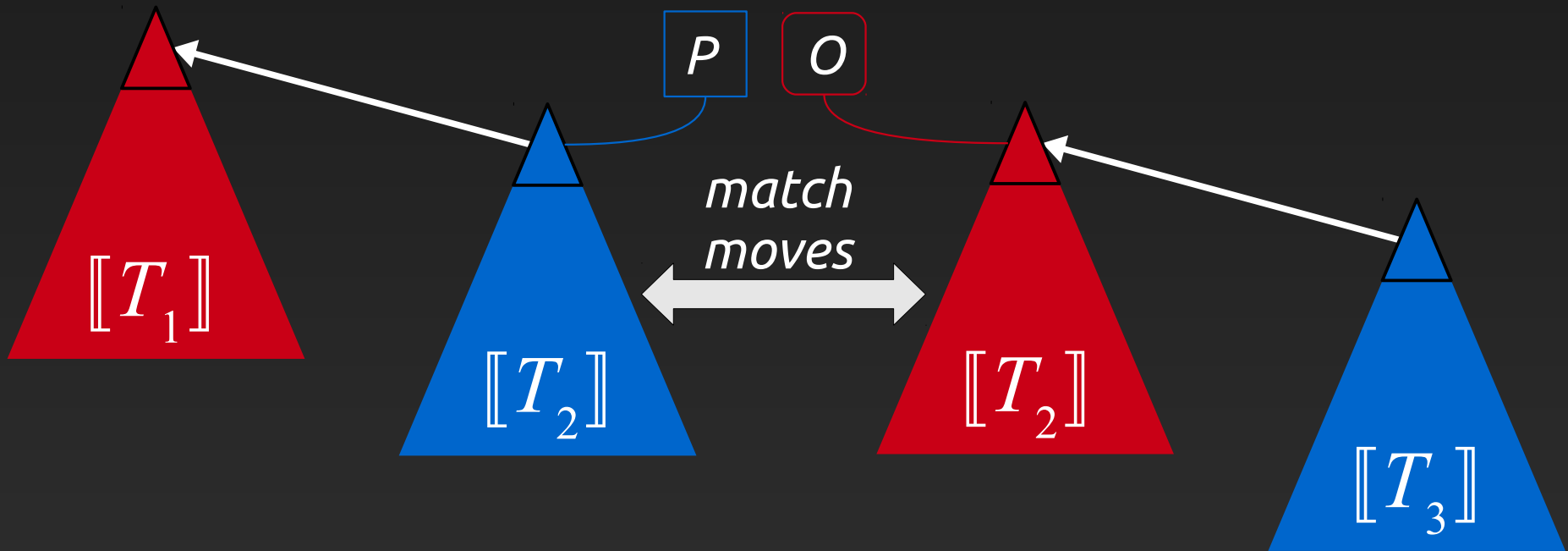
$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket$

$\llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$



Composition

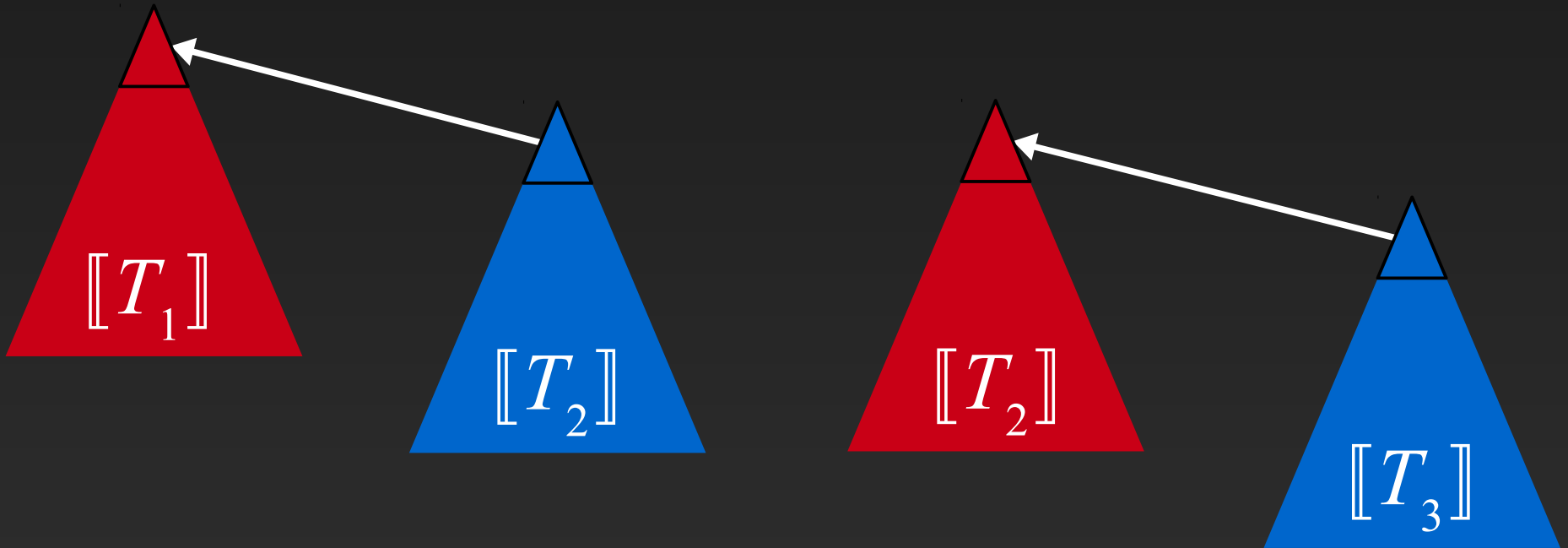
$$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket \quad \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$$



Composition

$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket$

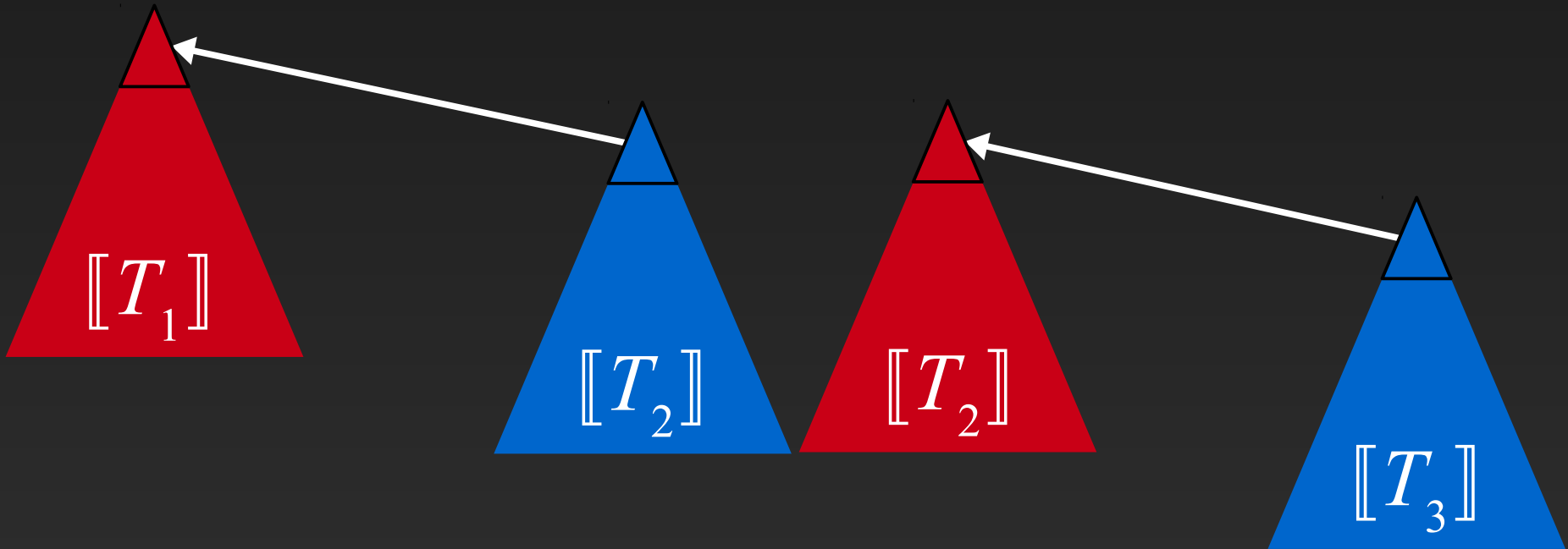
$\llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$



Composition

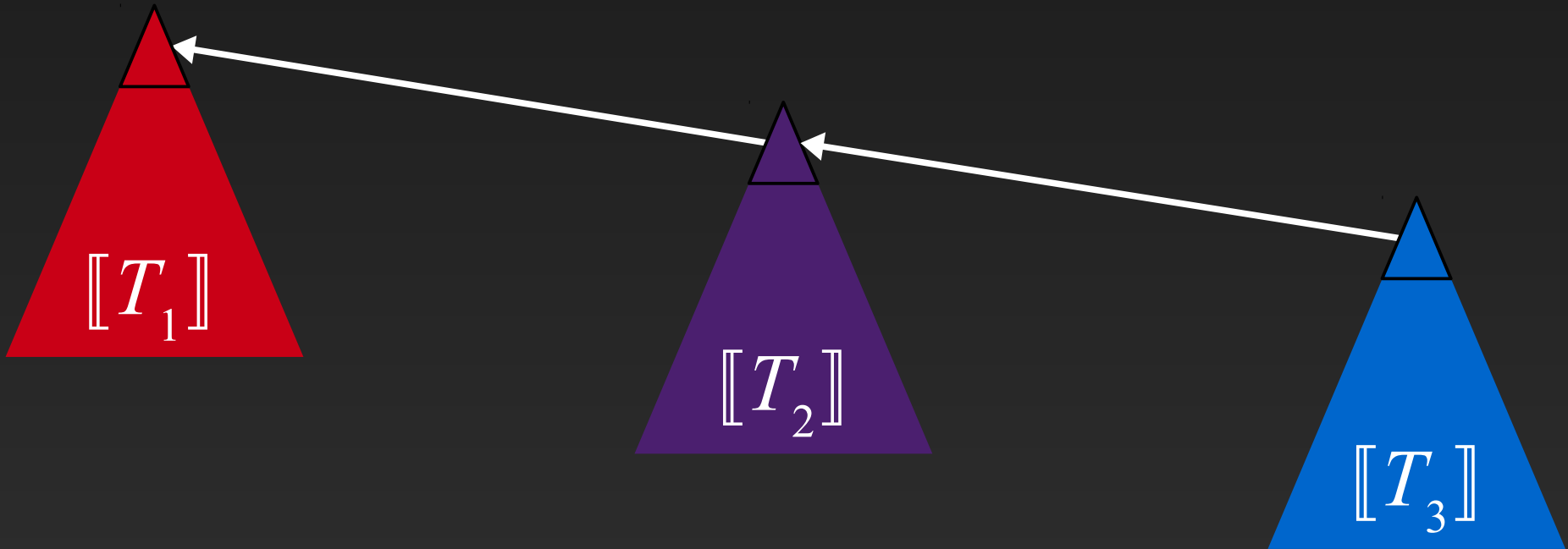
$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket$

$\llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$



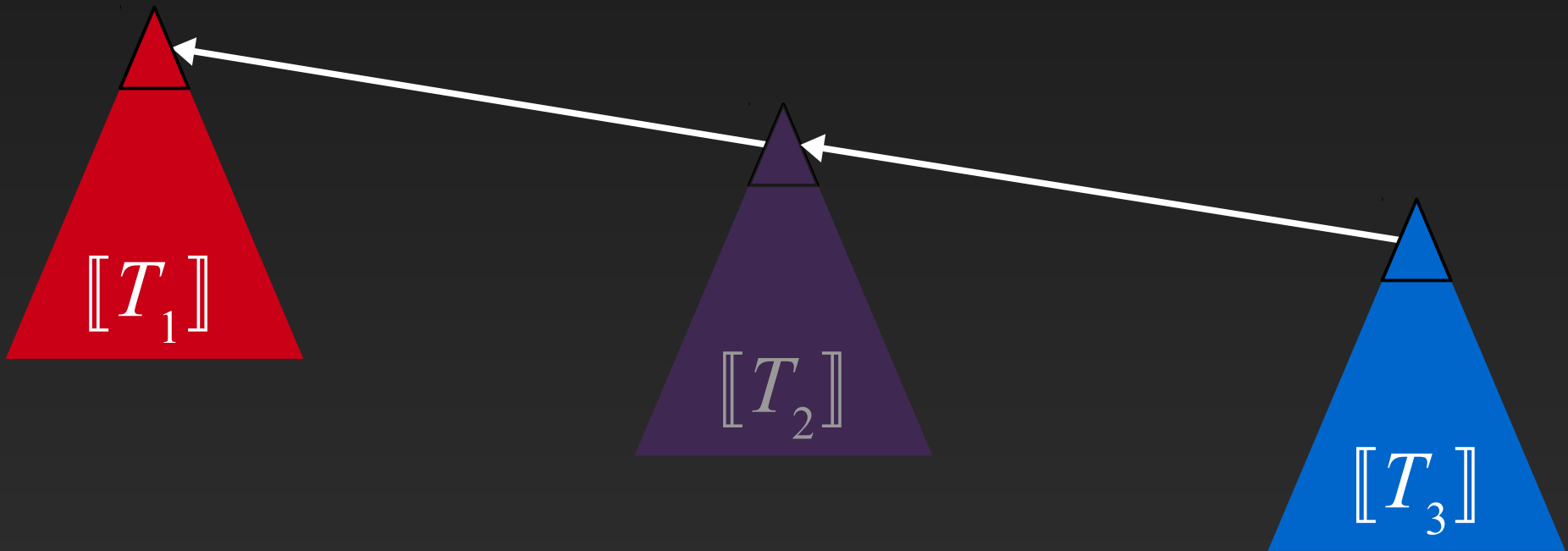
Composition

$$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket \quad \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$$



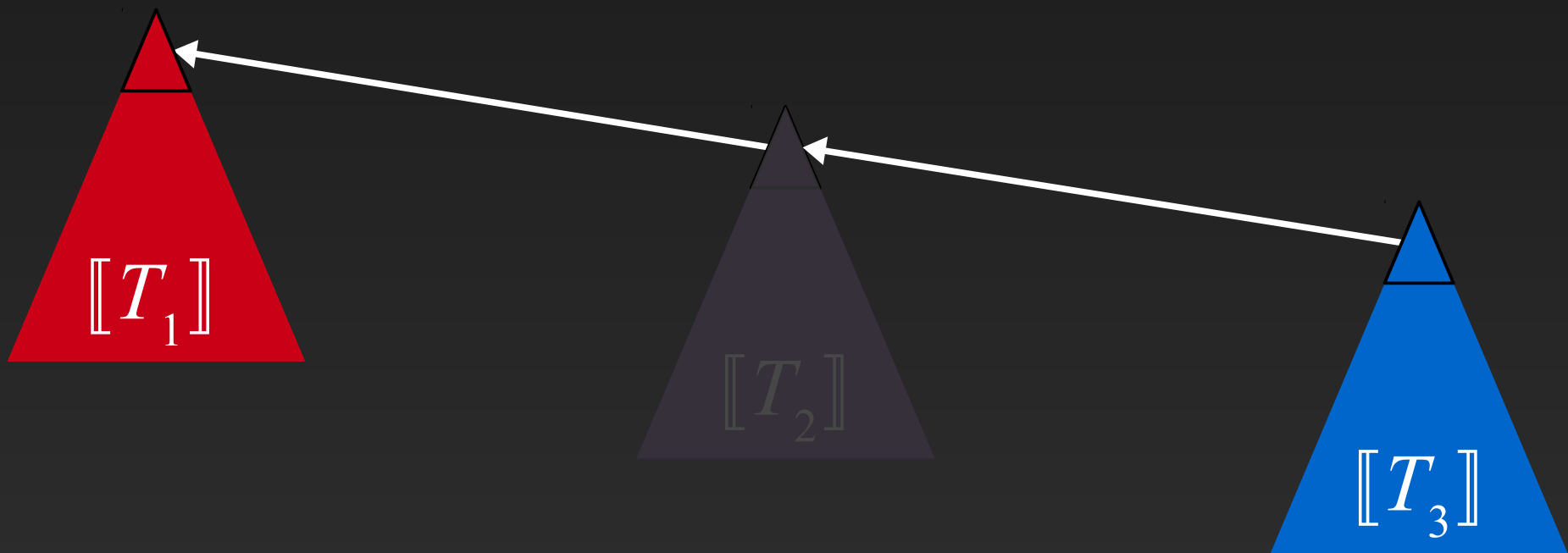
Composition

$$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket \quad \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$$



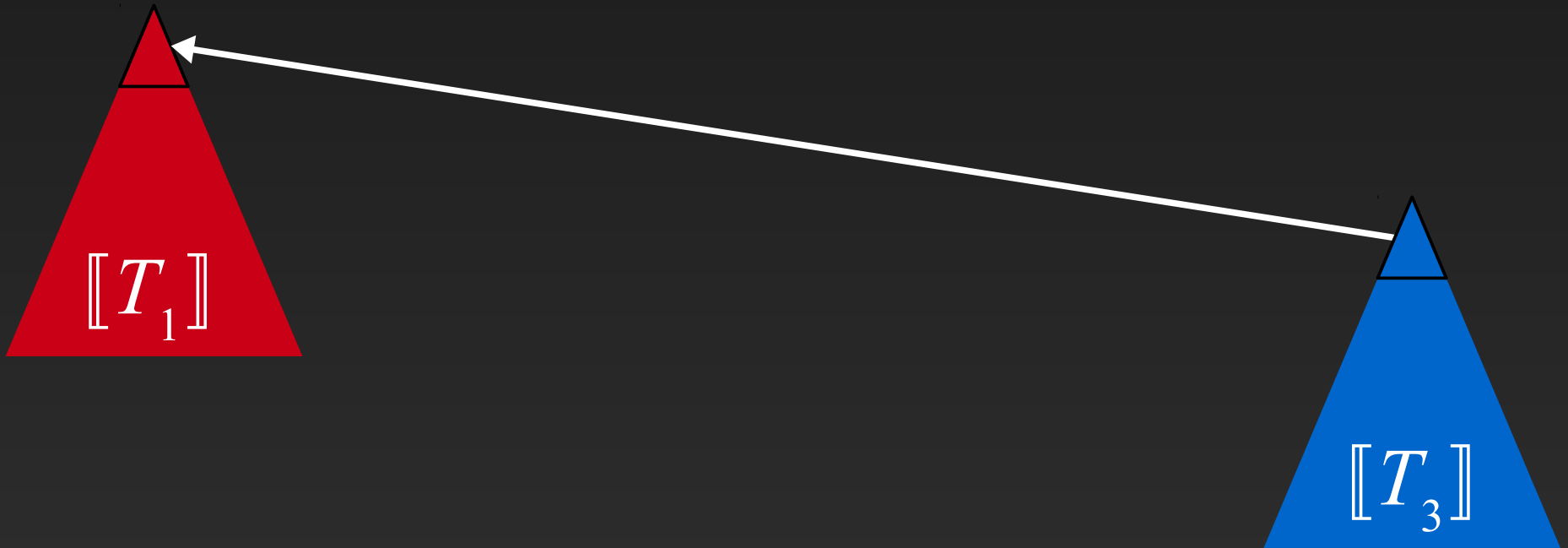
Composition

$$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket \quad \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$$



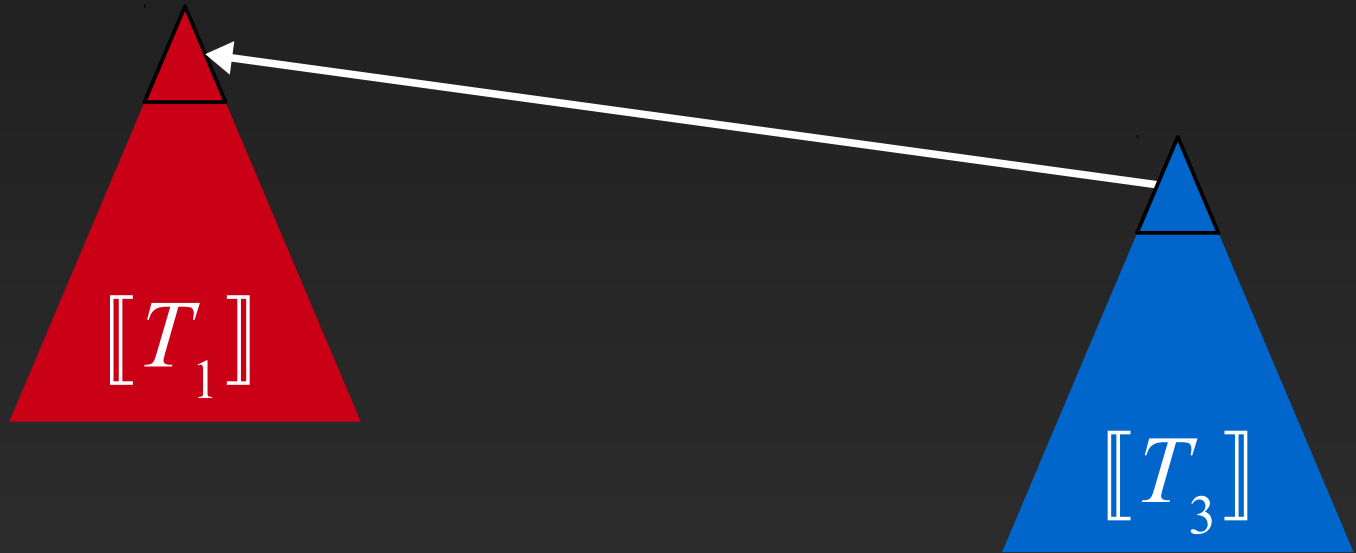
Composition

$$\llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket \quad \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket$$



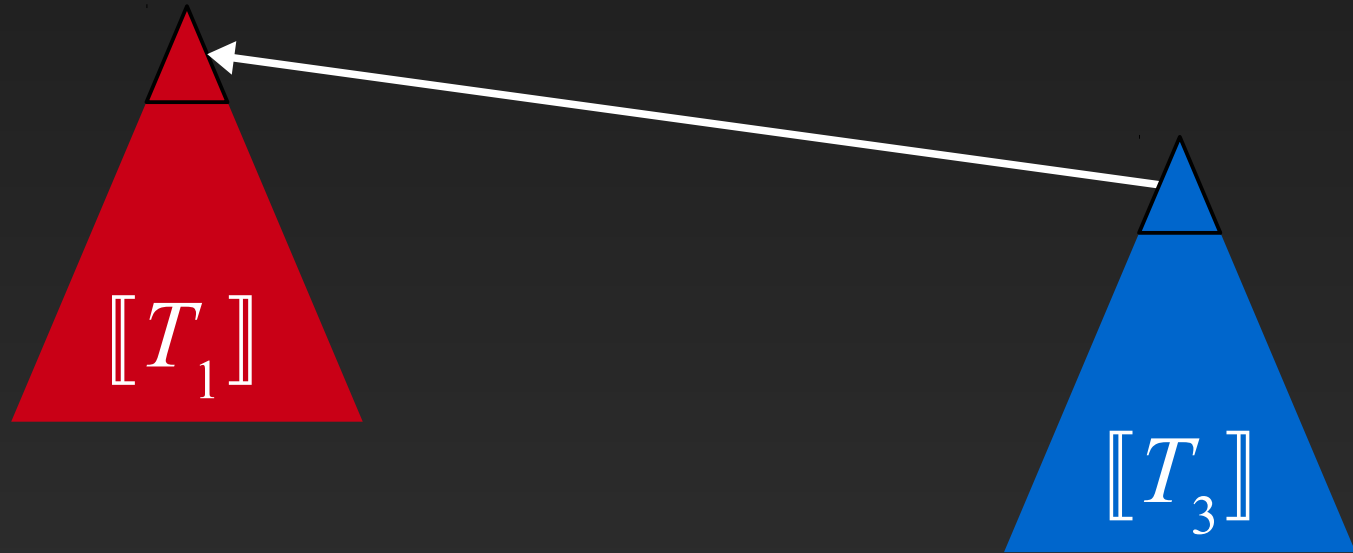
Composition

$$\begin{aligned} & \llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket ; \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket \\ & = \llbracket f_1:T_1 \vdash \text{let } f_2 = M_1 \text{ in } M_2:T_3 \rrbracket \end{aligned}$$



Composition

$$\begin{aligned} & \llbracket f_1:T_1 \vdash M_1:T_2 \rrbracket ; \llbracket f_2:T_2 \vdash M_2:T_3 \rrbracket \\ & = \llbracket f_1:T_1 \vdash \text{let } f_2 = M_1 \text{ in } M_2:T_3 \rrbracket \end{aligned}$$



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

Nominal composition example

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

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

$\begin{array}{ccc} * & \longleftarrow & a \\ & & O, Q \\ & & P, A \end{array}$

$y : \text{unit ref} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

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

$\begin{array}{ccc} a & \longleftarrow & \dagger \\ & & O, Q \\ & & P, A \\ & & O, Q \\ & & P, A \\ & & O, Q \\ & & P, A \\ \vdots & & \vdots \end{array}$

Nominal composition example

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

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

$\frac{}{* \longleftarrow a \quad \begin{matrix} O, Q \\ P, A \end{matrix}}$

•
;

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

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

$a \longleftarrow \dagger \quad \begin{matrix} O, Q \\ P, A \end{matrix}$

$\begin{matrix} a \\ \longleftarrow \\ a' \end{matrix} \quad \begin{matrix} O, Q \\ P, A \end{matrix}$

$\begin{matrix} \longleftarrow 1 \\ \longleftarrow 0 \end{matrix} \quad \begin{matrix} O, Q \\ P, A \end{matrix}$

Special conditions to ensure name and store privacy (*Laird conditions*)

Nominal composition example

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

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

$\frac{}{* \longleftarrow a \quad \begin{array}{l} O, Q \\ P, A \end{array}}$

•
;

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

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

$a \longleftarrow \dagger \quad \begin{array}{l} O, Q \\ P, A \end{array}$

$a' \longleftarrow 0 \quad \begin{array}{l} O, Q \\ P, A \end{array}$

Nominal composition example

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

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

$\frac{}{* \longleftarrow a \quad \begin{array}{l} O, Q \\ P, A \end{array}}$

$y : \text{ref unit} \vdash \lambda x. x == y : \text{unit ref} \rightarrow \text{int}$

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

$\frac{}{a \longleftarrow \dagger \quad \begin{array}{l} O, Q \\ P, A \end{array}}$

$\frac{}{a' \longleftarrow 0 \quad \begin{array}{l} O, Q \\ P, A \end{array}}$

$\dagger \longleftarrow a'$

$= \llbracket \lambda x. 0 : \text{unit ref} \rightarrow \text{int} \rrbracket$

Now an advanced example

$$\sigma : Ref_{\text{unit} \rightarrow \text{int}} \rightarrow (1 \Rightarrow \text{Int})$$

$$= a^a \begin{array}{c} \vdots \\ *^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ *^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ *^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ *^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ *^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ 3^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ i^a \\ \vdots \end{array} \begin{array}{c} \vdots \\ i^a \\ \vdots \end{array}$$

O P O P O P O P

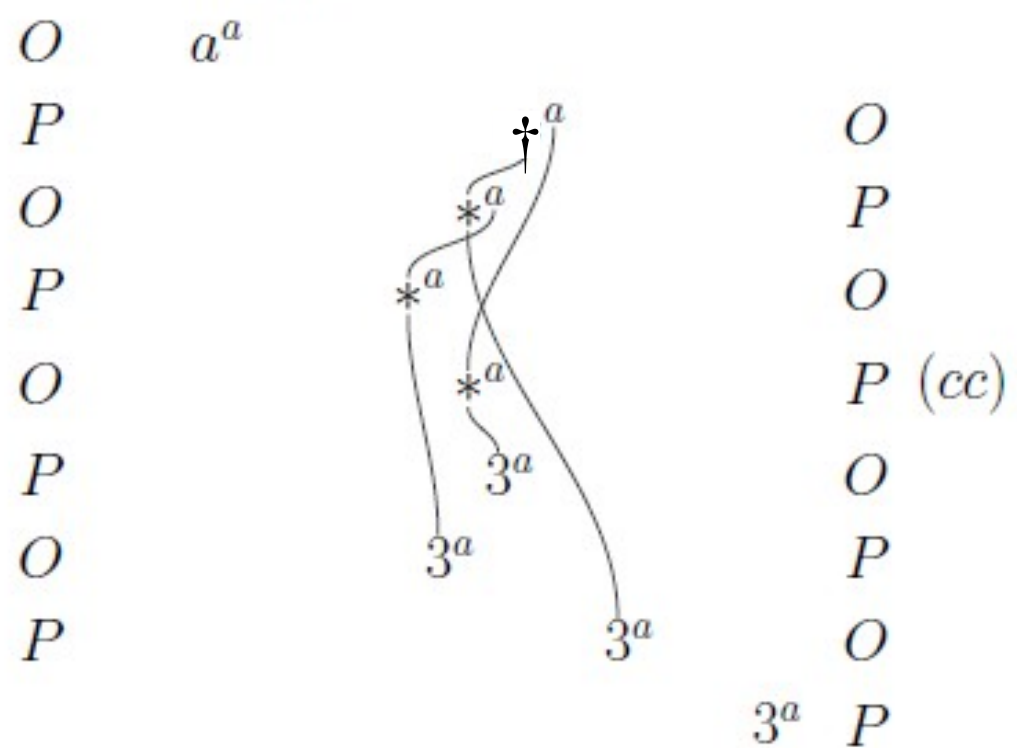
$$= \llbracket x := \lambda y.3; \lambda y.!x \rrbracket$$

$$\tau : (1 \Rightarrow \text{Int}) \rightarrow \text{Int}$$

$$= * \begin{array}{c} \vdots \\ \vdots \end{array} j \quad j = \llbracket f() \rrbracket$$

O P O P

$$Ref_{\text{unit} \rightarrow \text{int}} \xrightarrow{\sigma} 1 \Rightarrow \text{Int} \xrightarrow{\tau} \text{Int}$$



$$\text{let } f = (x := \lambda y.3; \lambda y.!x) \text{ in } f() \cong x := \lambda y.3; 3$$

Taking stock

Games yield denotational models, which:

- are compositional / operational
- precisely capture what is going on:
 - sound & complete: $P \cong P' \Leftrightarrow \llbracket P \rrbracket = \llbracket P' \rrbracket$
 - every “computable” strategy is in the language

Taking stock

Games yield denotational models, which:

- are compositional / operational
- precisely capture what is going on:
 - sound & complete: $P \cong P' \Leftrightarrow \llbracket P \rrbracket = \llbracket P' \rrbracket$
 - every “computable” strategy is in the language

Effects = – conditions + name generators

- Ground store: – innocence + ground reference names
- Higher-order store: – visibility + HO reference names
- Exceptions: – well-bracketing + exception names

Nominal games portfolio

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

dblp: Laird, Murawski, T.

- Trace models

dblp: Laird, Ghica, Jaber, T.

- Algorithmic games

Algorithmic game semantics

P Programs



$\llbracket P \rrbracket$ Game Semantics

$$P \cong P' \Leftrightarrow \llbracket P \rrbracket = \llbracket P' \rrbracket$$

Algorithmic game semantics

P Programs



$\llbracket P \rrbracket$ Game Semantics

$$\llbracket \lambda x. \text{ref}() \rrbracket = \{ * \dagger * a_1 * a_2 \dots \}$$

$$P \cong P' \Leftrightarrow \llbracket P \rrbracket = \llbracket P' \rrbracket$$

Algorithmic game semantics

P Programs



$\llbracket P \rrbracket$ Game Semantics



$A(P)$ Automata

$$\llbracket \lambda x. \text{ref}() \rrbracket = \{ * \dagger * a_1 * a_2 \dots \}$$

$$P \cong P' \Leftrightarrow \llbracket P \rrbracket = \llbracket P' \rrbracket$$

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

The image features the text "Nominal Automata" in a bold, 3D-style font with a color gradient from red to yellow. The text is centered and overlaid on a blue circular background. Surrounding the circle are several blue triangles pointing outwards, and a large yellow sunburst shape is positioned below the circle. The entire graphic is set against a dark gray background.

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 (i.e. automata over infinite alphabets)

Fresh-register automata

$$\llbracket \lambda x. \text{ref}() \rrbracket = \{ * \dagger * a_1 * a_2 \dots \mid a_i\text{'s distinct} \}$$

Automata with names

- Infinite alphabet
- Freshness recognition

Fresh-register automata

$$\llbracket \lambda x. \text{ref}() \rrbracket = \{ * \dagger * a_1 * a_2 \dots \mid a_i\text{'s distinct} \}$$

Automata with names

- Infinite alphabet
- Freshness recognition

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

Finite-state machines with registers

Register Automata (RA)

Let $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$ be an **infinite** alphabet of **names**



finitely many
(say R) **registers**

registers store names

Label λ of the form:

$\text{read}(i), i \in \{1, \dots, R\}$

$\text{fresh}(i), i \in \{1, \dots, R\}$

- $\kappa, \kappa \in F$

a finite set
of **constants**

Register Automata (RA)

Let $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$ be an **infinite** alphabet of **names**



finitely many
(say R) **registers**

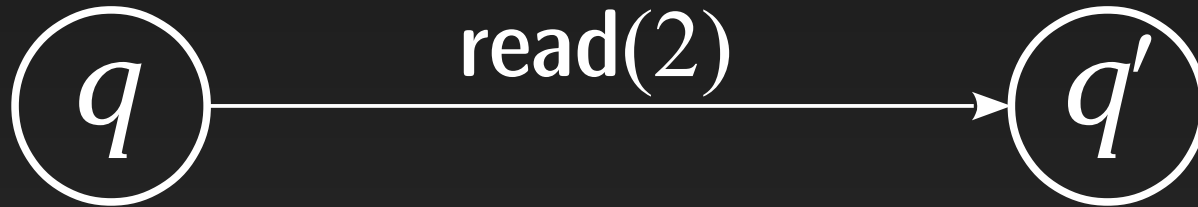
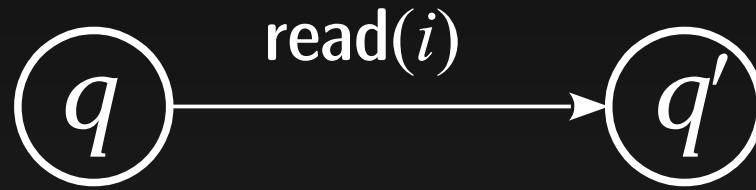
registers store names

Label λ of the form:

- **read**(i), $i \in \{1, \dots, R\}$
- **fresh**(i), $i \in \{1, \dots, R\}$
- κ , $\kappa \in F$

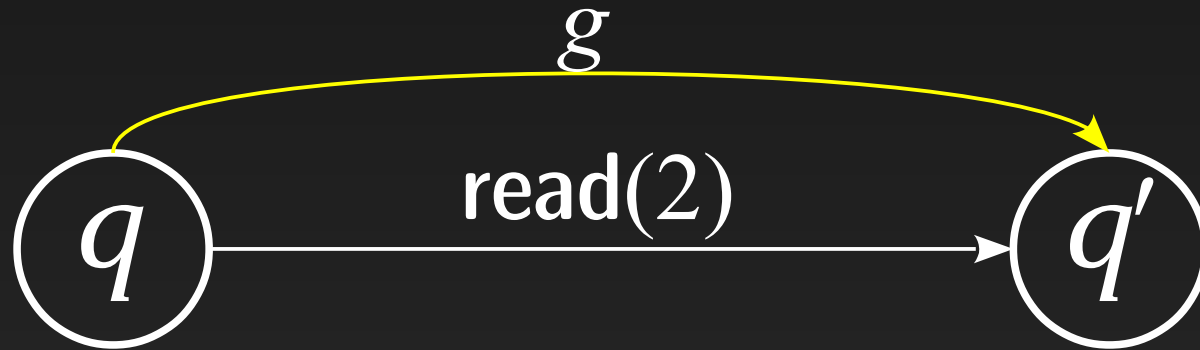
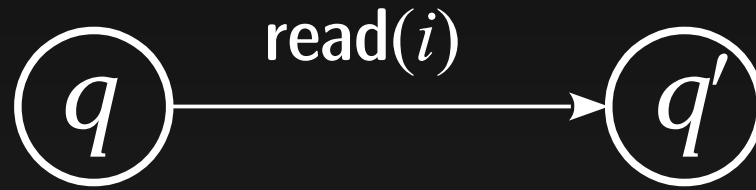
a finite set
of **constants**

Transitions:



a	g	b
-----	-----	-----

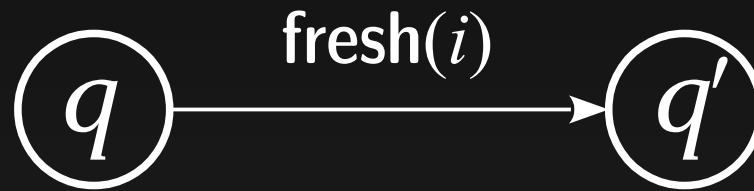
Transitions:



a	g	b
-----	-----	-----

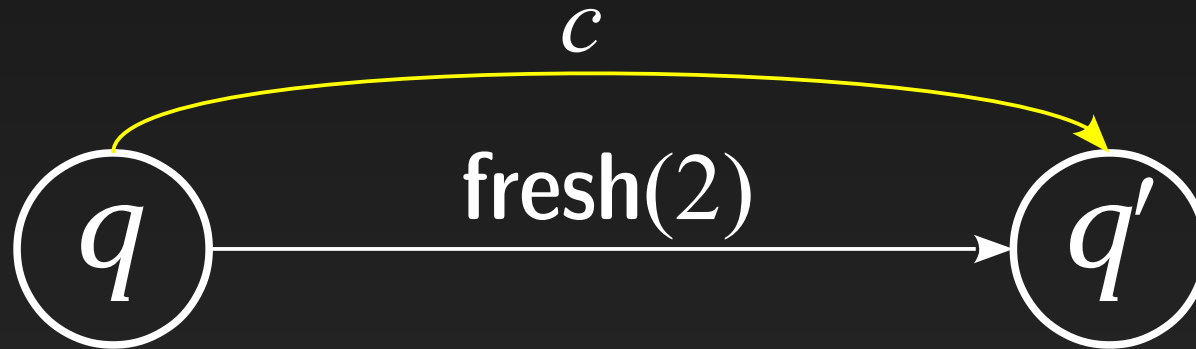
a	g	b
-----	-----	-----

Transitions:



a	g	b
-----	-----	-----

Transitions:

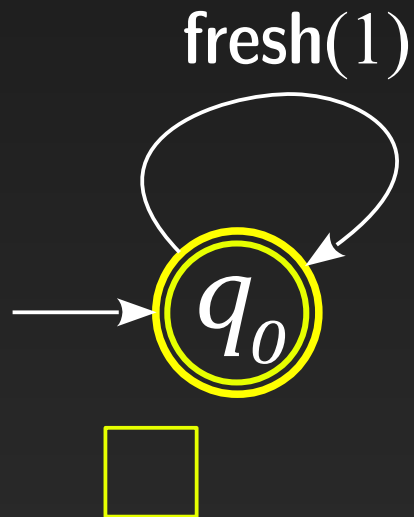


fresh

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



a

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



ab

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abc

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abca

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abcd

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abcade

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abcadeb

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abcadebagcabab

Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

(all strings where each name is distinct from its predecessor)



abcadebagcabab and we love cake

Application: parametric properties

```
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();  
}
```

we can express properties like:

*if an iterator modifies its collection x
then other iterators of x become invalid*

(the code on the left is bad)

[Grigore, Distefano, Petersen & T. '13]

Global freshness?

```
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();  
}
```

we can express properties like:

*if an iterator modifies its collection x
then other iterators of x become invalid*

(the code on the left is bad)

[Grigore, Distefano, Petersen & T. '13]

but we cannot model **new**

Fresh-Register Automata (FRA)

Let $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$ be an **infinite** alphabet of **names**



finitely many
(say R) **registers**

registers store names

Label λ of the form:

- **read**(i), $i \in \{1, \dots, R\}$
- **fresh**(i), $i \in \{1, \dots, R\}$
- κ , $\kappa \in F$
- **gl-fresh**(i), $i \in \{1, \dots, R\}$

global freshness oracle

Transitions:



Transitions:



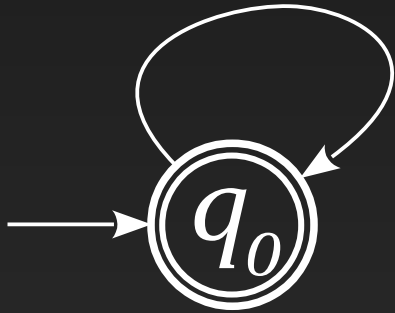
globally fresh

Examples

$$L_2 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

(all strings of pairwise distinct names)

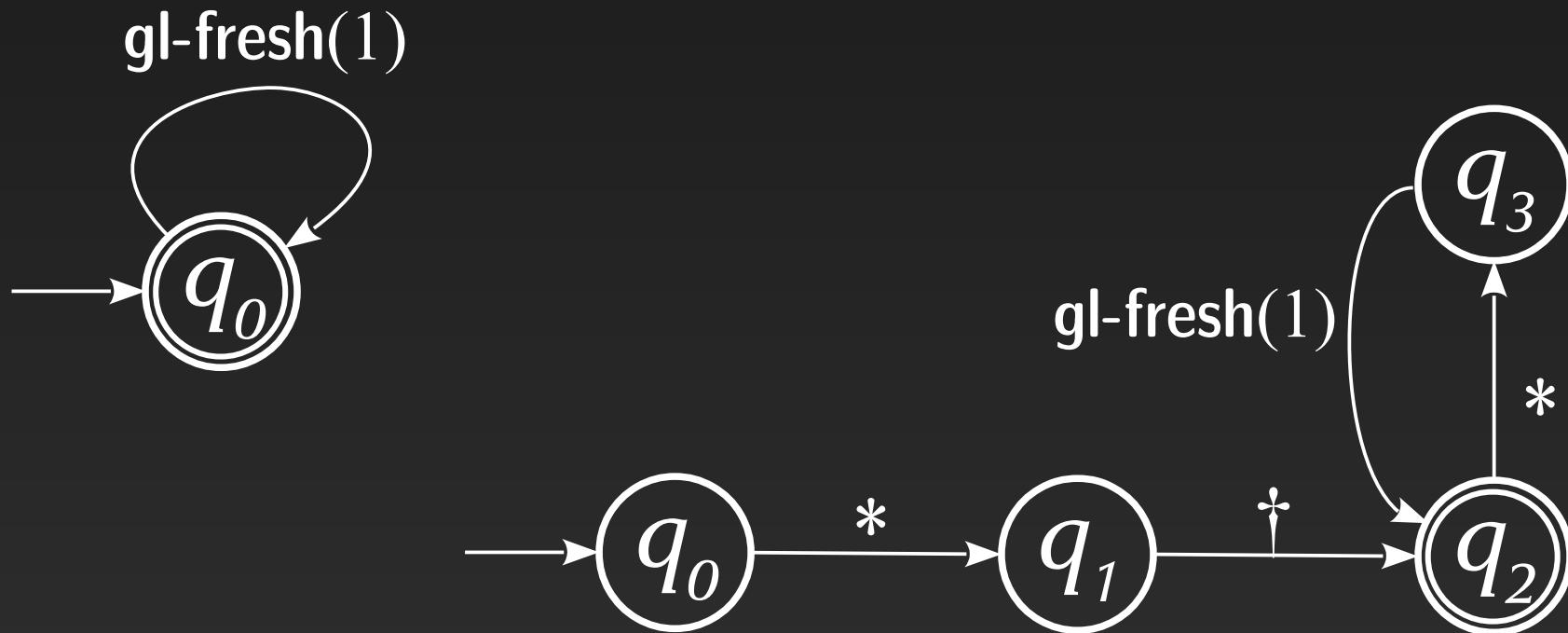
gl-fresh(1)



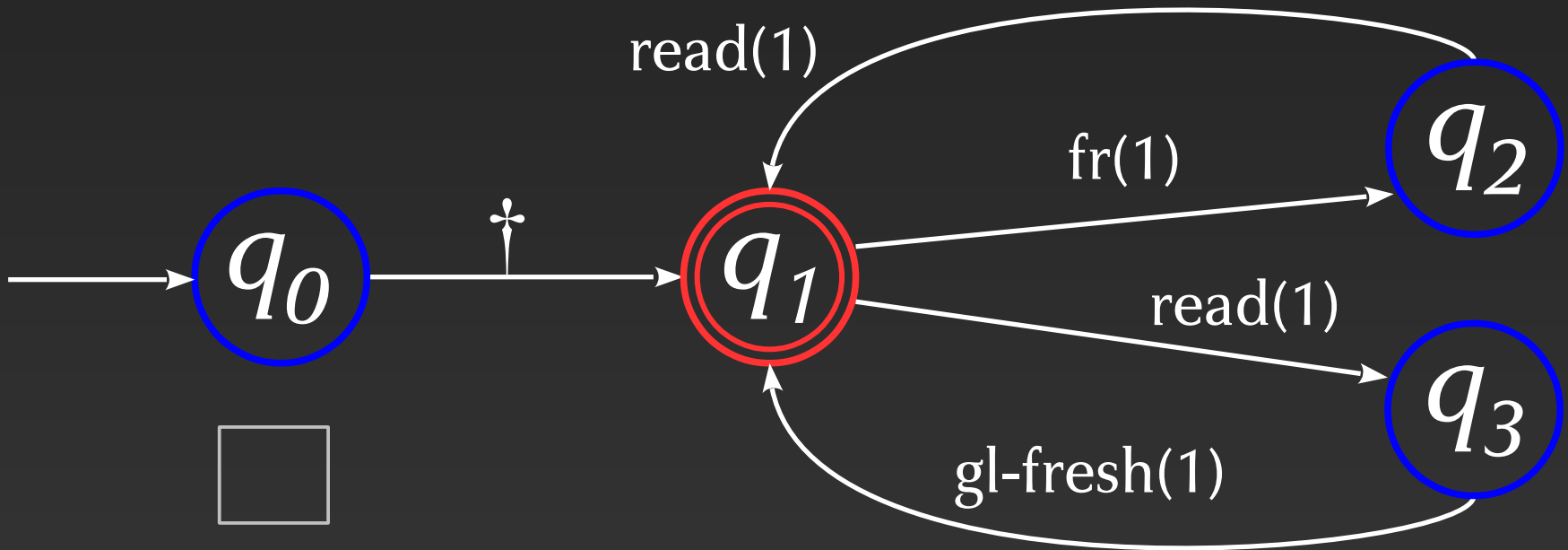
Examples

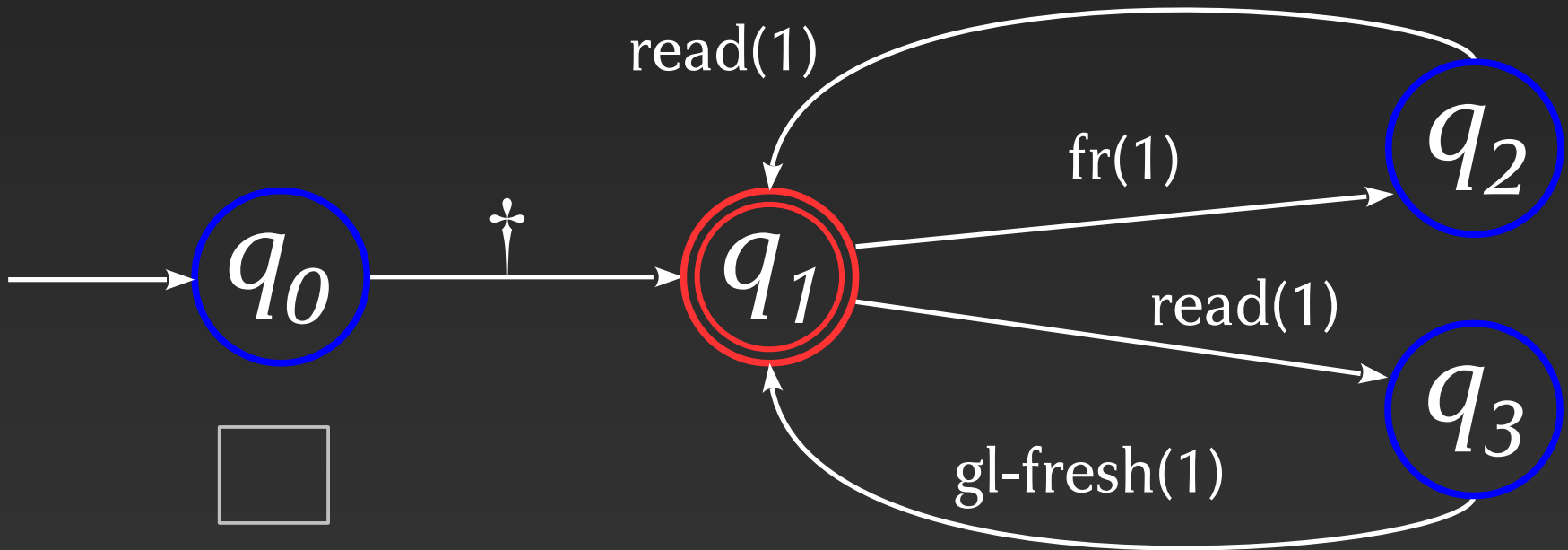
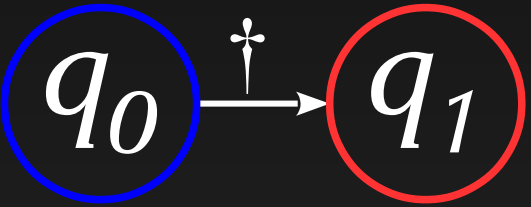
$$L_2 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

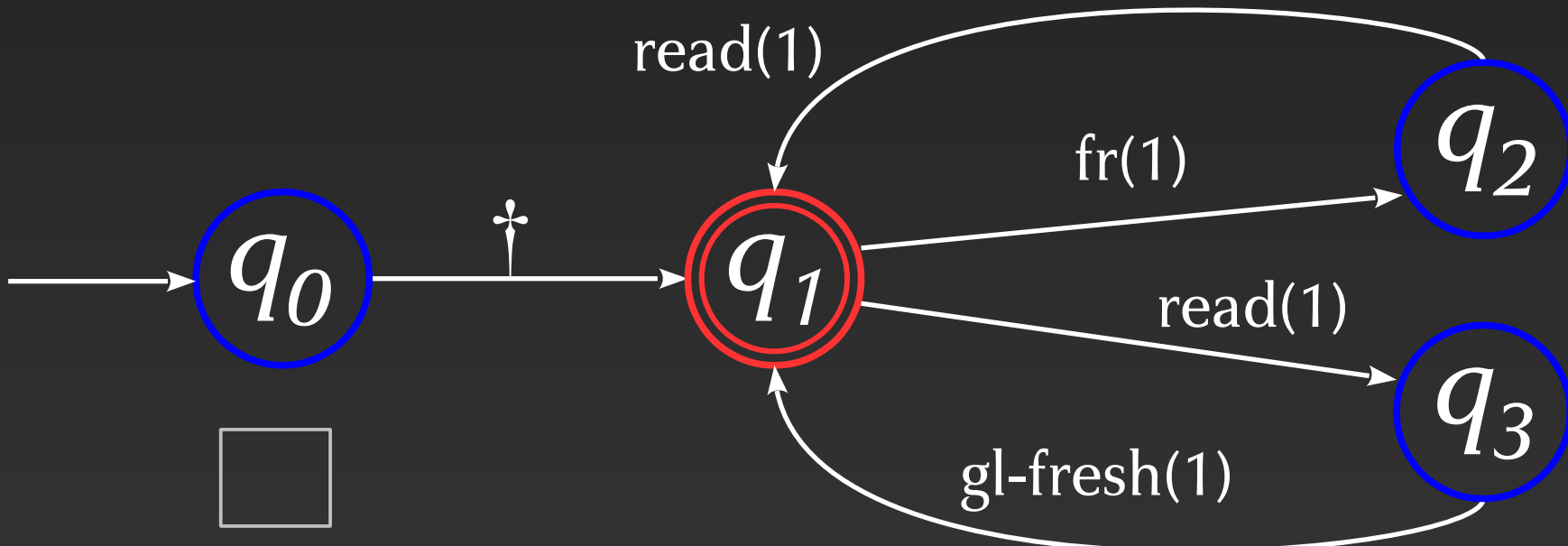
(all strings of pairwise distinct names)

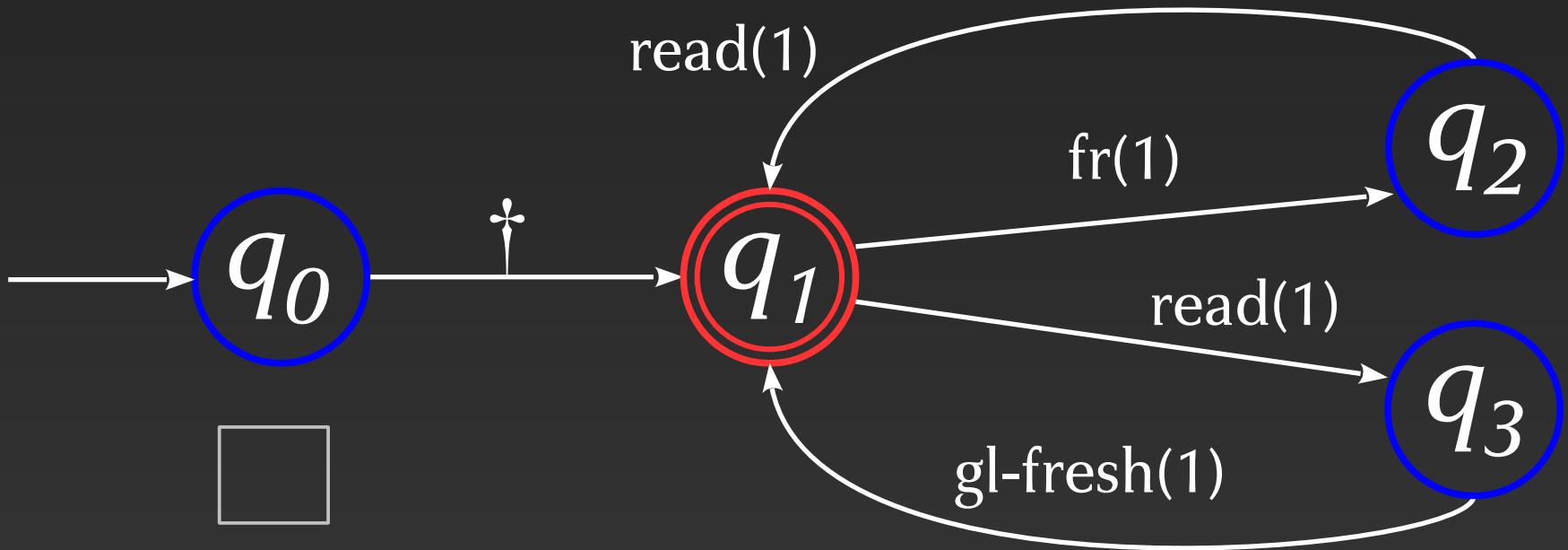
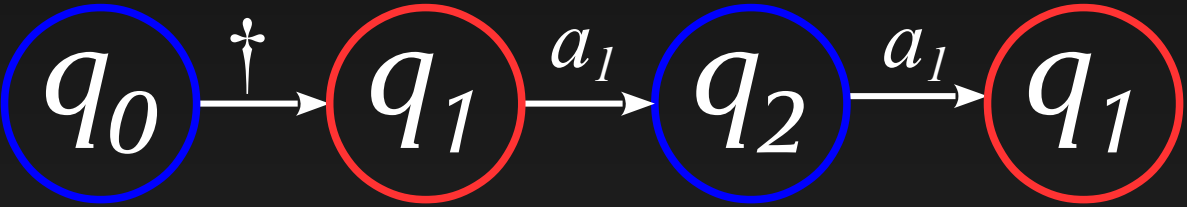


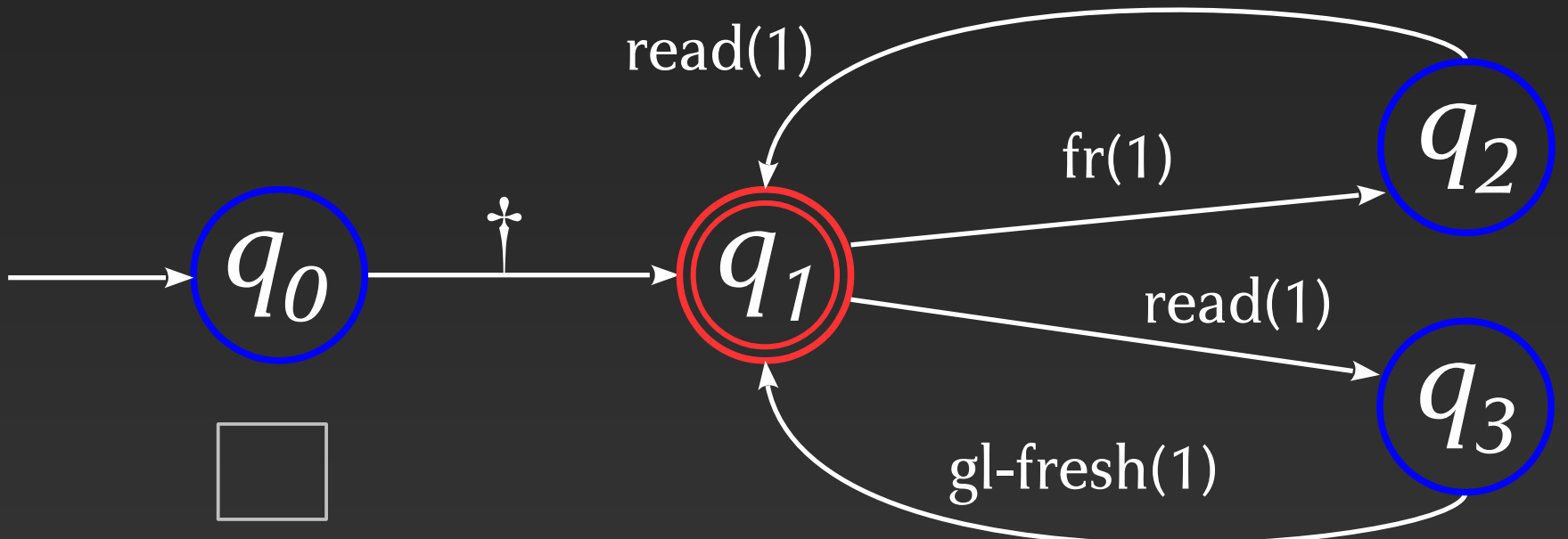
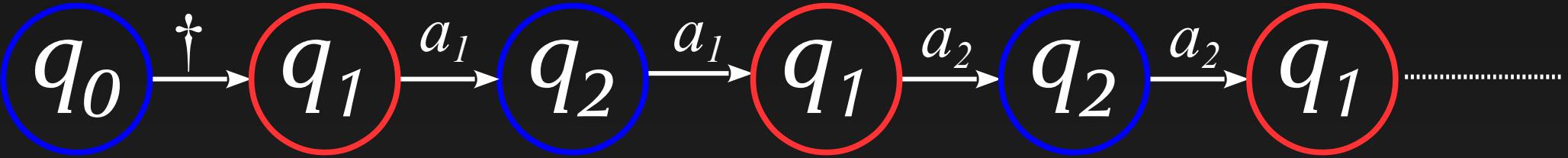
$$\llbracket \lambda x. \text{ref}() \rrbracket = \{ * \dagger * a_1 * a_2 \dots \mid a_i \text{'s distinct} \}$$

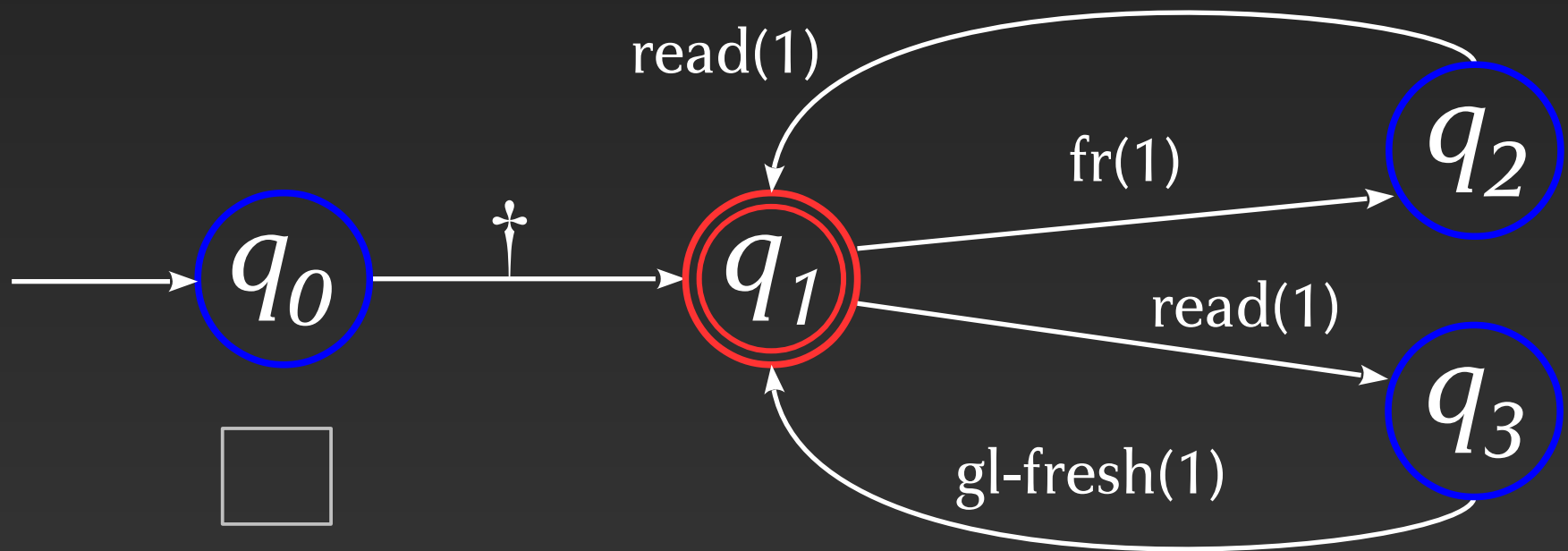
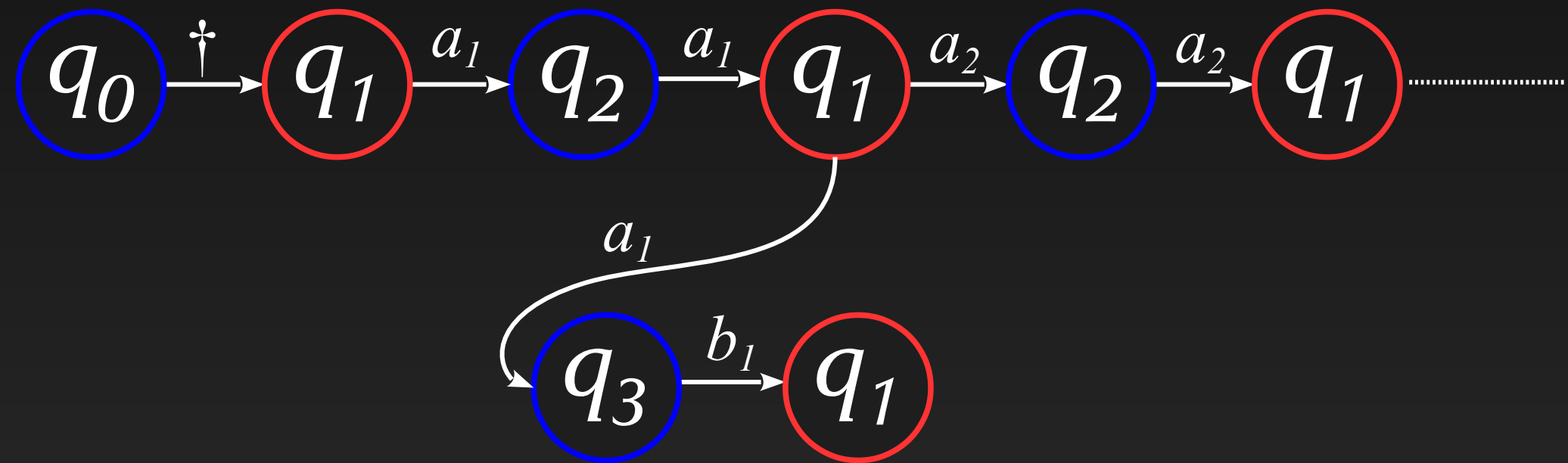


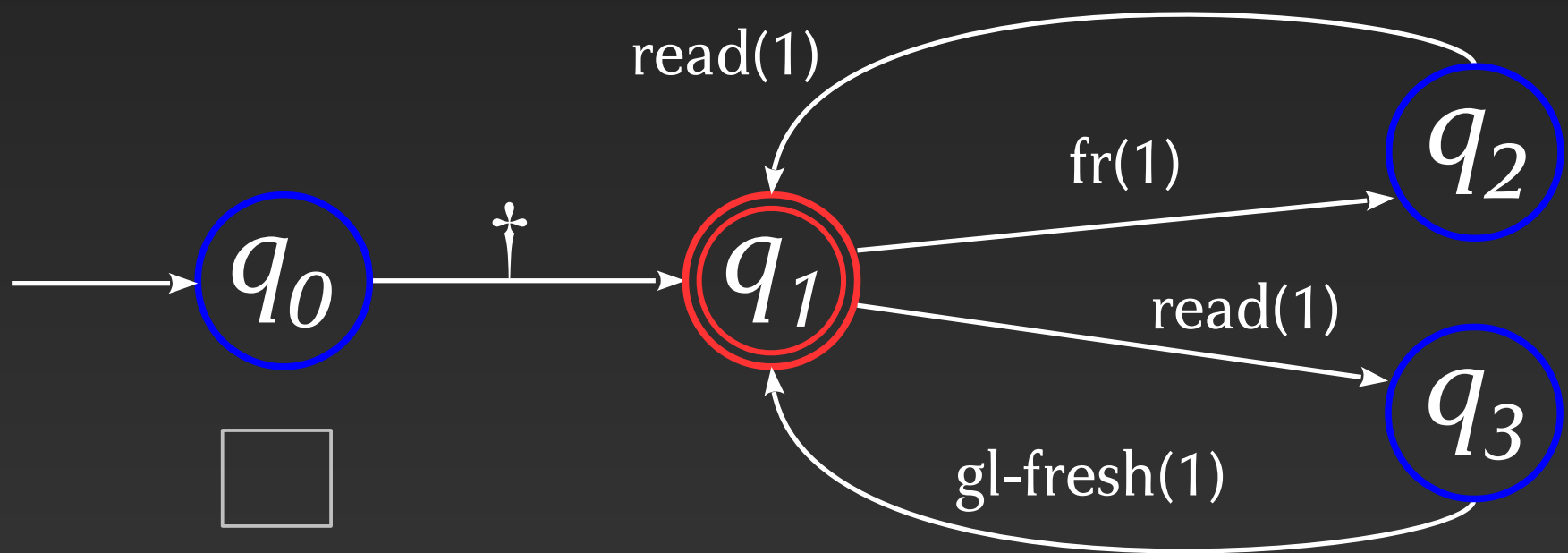
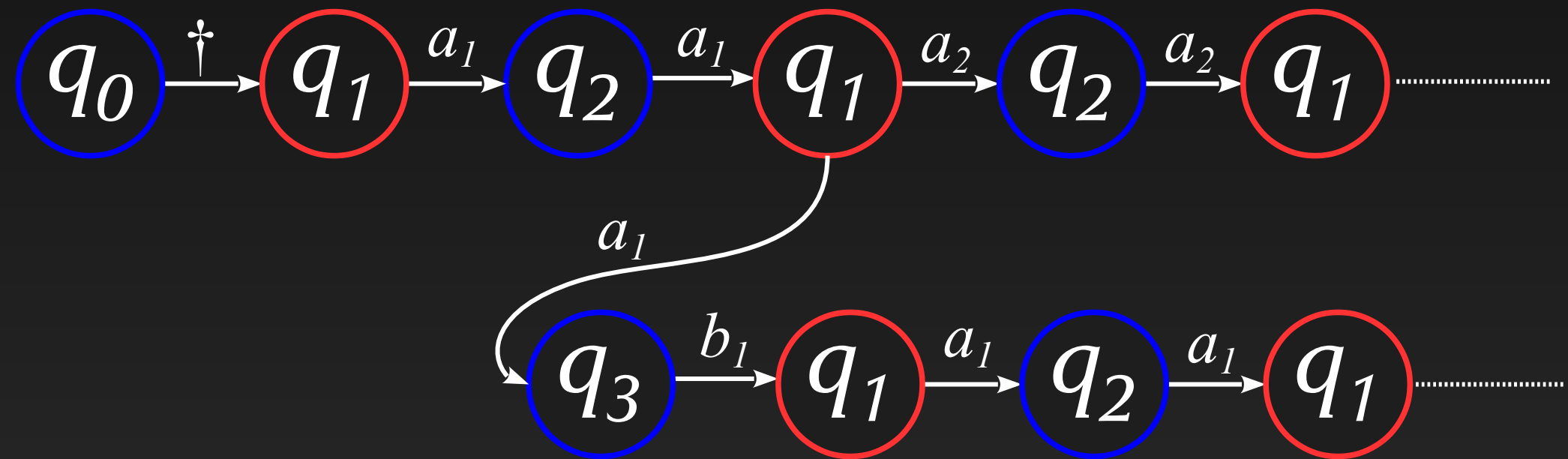










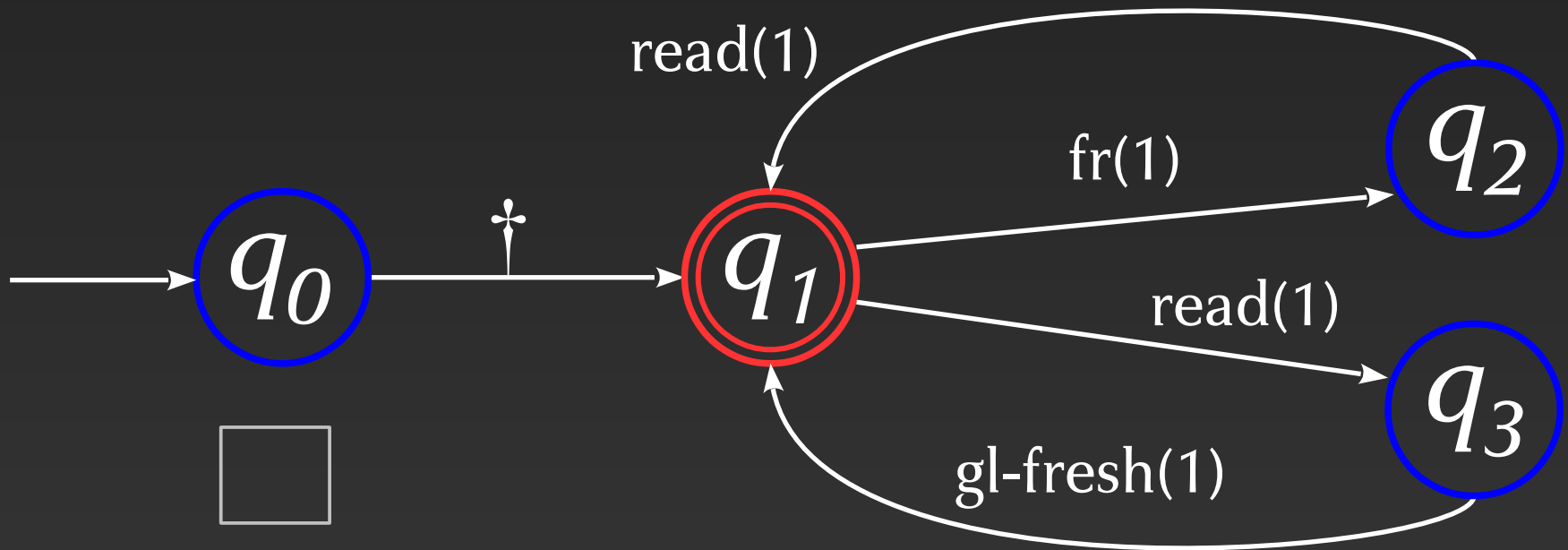
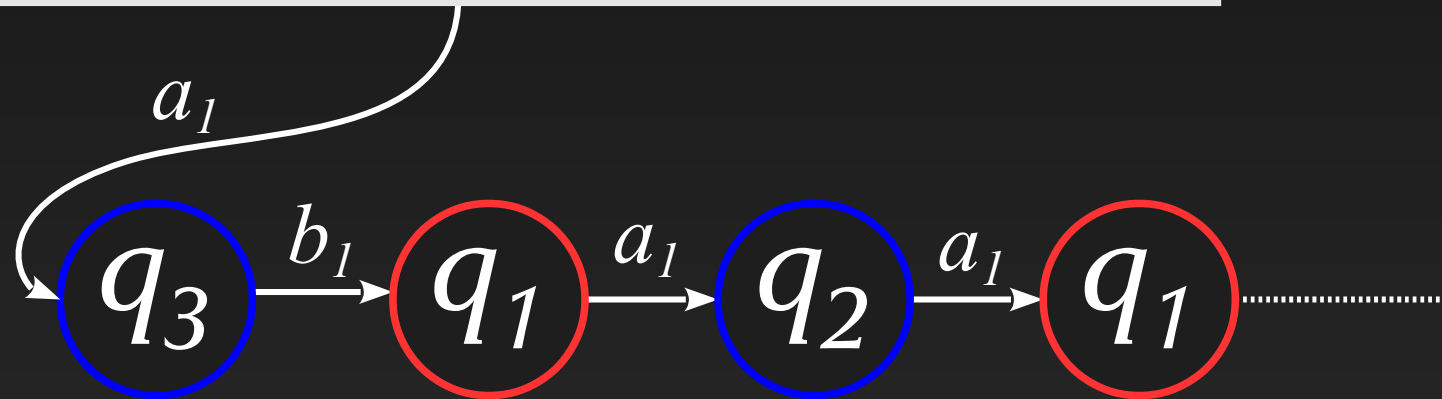


let $y = \text{ref}(\text{ref}())$ in

$\lambda x. y := (\text{if } x == !y \text{ then } \text{ref}() \text{ else } x); !y$

: unit ref \rightarrow unit ref

q_0



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

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

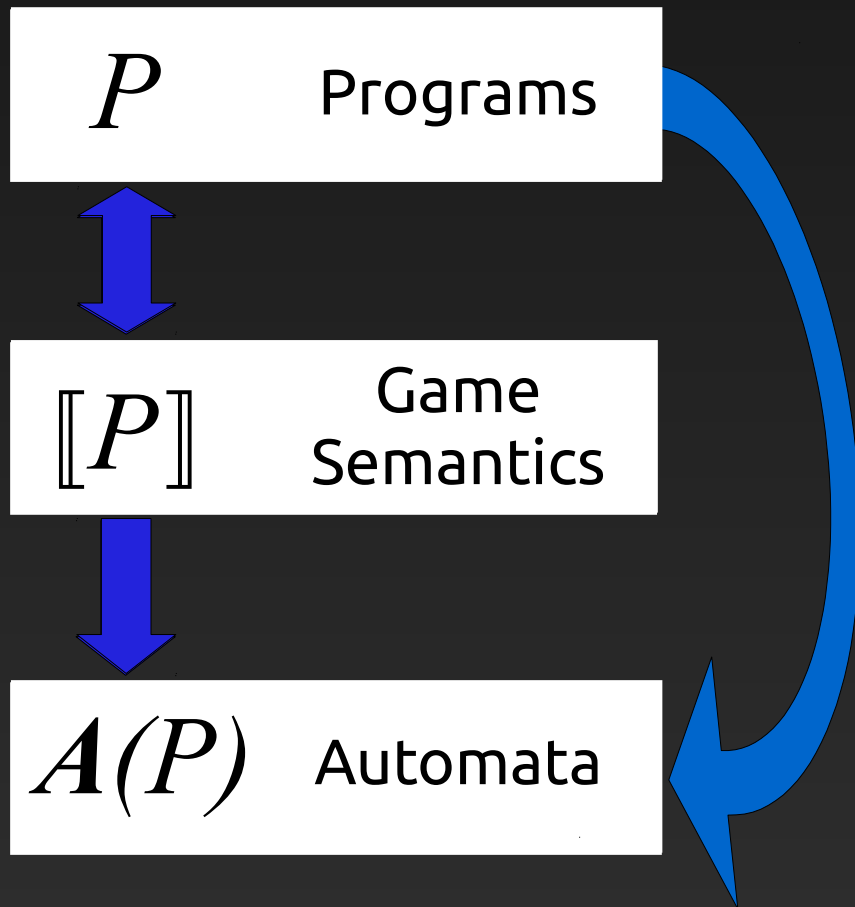
Representation of strategies as FRA's

- with “data words”
- pushdown variants

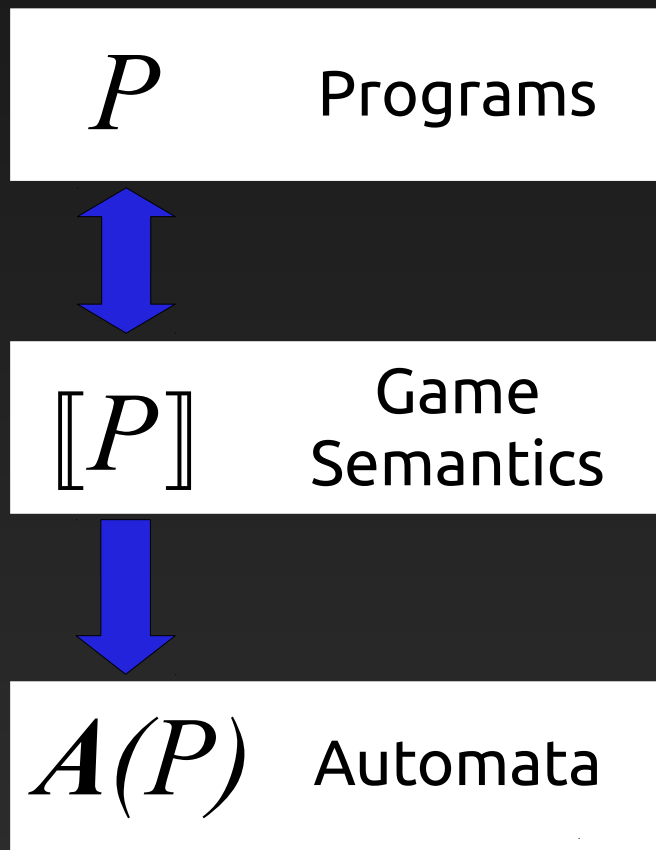
pairs (κ, a) :
- κ from finite alph.
- a a name

stack stores constants and register “snapshots”

From programs to automata



From programs to automata



The passage can be done automatically
→ decision procedure for program
equivalence

- Possible at specific types
- Full type-based classification
(at each type, either the problem is
undecidable or we get a procedure)

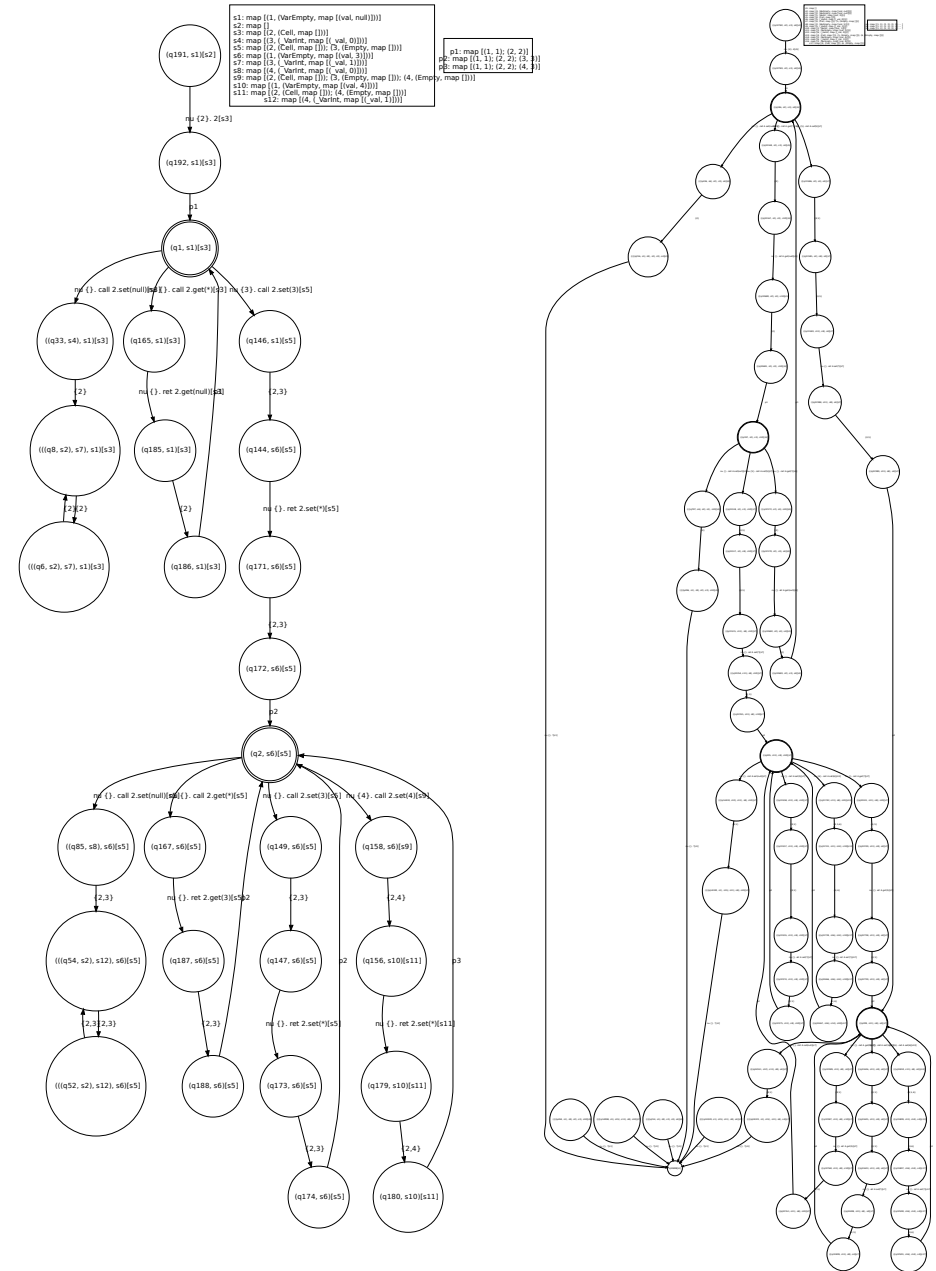
Applied to ML with ground references
and Interface Middleweight Java

[Murawski & T. '11,'12; Murawski, Ramsay & T.]

Implementation

$M_1 \equiv \text{let } v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{new } \{ _ : \text{Cell};$
 $\text{get} : \lambda _ . v.\text{val},$
 $\text{set} : \lambda y . \text{if } y = \text{null} \text{ then div else } v.\text{val} := y \}$

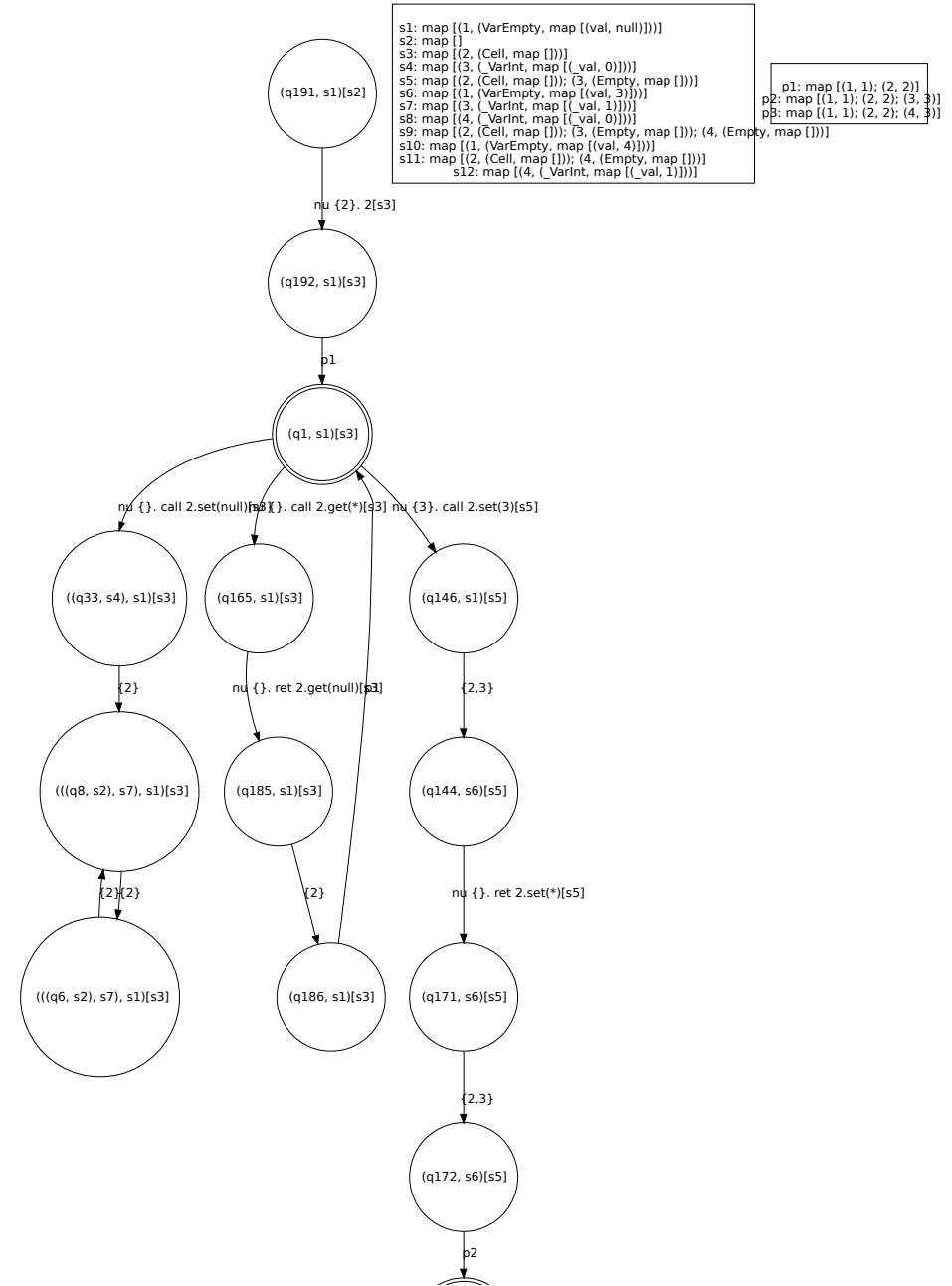
$M_2 \equiv \text{let } b = \text{new } \{ _ : \text{Var}_{\text{int}} \} \text{ in}$
 $\text{let } v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{let } w = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{new } \{ _ : \text{Cell};$
 $\text{get} : \lambda _ . \text{if } b.\text{val} = 1 \text{ then } (b.\text{val} := 0; v.\text{val})$
 $\qquad \qquad \qquad \text{else } (b.\text{val} := 1; w.\text{val}),$
 $\text{set} : \lambda y . \text{if } y = \text{null} \text{ then div}$
 $\qquad \qquad \qquad \text{else } v.\text{val} := y; w.\text{val} := y \}$



Implementation

$M_1 \equiv \text{let } v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{new } \{ _ : \text{Cell};$
 $\text{get} : \lambda _ . v.\text{val},$
 $\text{set} : \lambda y . \text{if } y = \text{null} \text{ then div else } v.\text{val} := y \}$

$M_2 \equiv \text{let } b = \text{new } \{ _ : \text{Var}_{\text{int}} \} \text{ in}$
 $\text{let } v = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{let } w = \text{new } \{ _ : \text{Var}_{\text{Empty}} \} \text{ in}$
 $\text{new } \{ _ : \text{Cell};$
 $\text{get} : \lambda _ . \text{if } b.\text{val} = 1 \text{ then } (b.\text{val} := 0; v.\text{val})$
 $\text{else } (b.\text{val} := 1; w.\text{val}),$
 $\text{set} : \lambda y . \text{if } y = \text{null} \text{ then div}$
 $\text{else } v.\text{val} := y; w.\text{val} := y \}$



Summary and what's next

Our journey: programs → games → automata

- games capture essential program behaviour
- denotational / operational / algorithmic

Further on:

- More effects (can we have one model for all?)
- Trace models and verification
- Games as a paradigm for open/HO programs

Summary and what's next

thanks

Our journey: programs \rightarrow games \rightarrow automata

- games capture essential program behaviour
- denotational / operational / algorithmic

Further on:

- More effects (can we have one model for all?)
- Trace models and verification
- Games as a paradigm for open/HO programs