Nominal games: a semantics paradigm for effectful languages

Nikos Tzevelekos, Queen Mary U. of London

jointly with:

Andrzej Murawski, Oxford

Guilhem Jaber, Lyon

Dan Ghica, Birmingham

Steven Ramsay, Bristol

Thomas Cuvillier, QMUL

Dagstuhl Seminar on Program Equivalence, April 2018

what this talk is about

Examine higher-order programming languages with effects:
state, exceptions, polymorphism

Look into semantic models capturing these effects

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

so that: program equivalence = equality of $[\![\_]\!]$'s

Present nominal game semantics, a technique whereby:

- programs are modelled compositionally as 2-player games
- effects are be captured by use of name-abstractions

Setting: HO programs + effects

$$M ::= () \mid i \mid x \mid \lambda x^{\vartheta}.M \mid M\ M \mid \mathsf{if} \mid \oplus \mid a \mid \mathsf{ref}\ M \mid M = M$$

$$\vartheta ::= \mathsf{unit} \mid \mathsf{int} \mid \vartheta \to \vartheta \mid \mathsf{ref}\ \vartheta$$

$\Gamma = \{\ f : \mathsf{ref}(\mathsf{unit} \to \mathsf{int}) \to \mathsf{int},\ x : \mathsf{unit},\ \ldots\ \}$

$$\Gamma \vdash () : \mathsf{unit} \qquad \Gamma \vdash i : \mathsf{int} \qquad \Gamma, x : \vartheta \vdash x : \vartheta \qquad \frac{\Gamma \vdash M : \vartheta \to \vartheta' \qquad \Gamma \vdash N : \vartheta}{\Gamma \vdash M\,N : \vartheta'}$$

$$\Gamma \vdash \oplus : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$$

$$\frac{a \in \mathsf{Loc}_{\vartheta}}{\Gamma \vdash a : \mathsf{ref}\ \vartheta} \qquad \frac{\Gamma \vdash M, N : \mathsf{ref}\ \vartheta}{\Gamma \vdash M = N : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \vartheta}{\Gamma \vdash \mathsf{ref}\ M : \mathsf{ref}\ \vartheta}$$

$$\frac{\Gamma \vdash M : \mathsf{ref}\ \vartheta}{\Gamma \vdash\ !M : \vartheta} \qquad \frac{\Gamma \vdash M : \mathsf{ref}\ \vartheta \qquad \Gamma \vdash N : \vartheta}{\Gamma \vdash M := N : \mathsf{unit}}$$

Operational semantics (examples)

$$M,\ h \quad \rightarrow \quad M',\ h'$$

h stores locations + their values

$$(\lambda x.M)v,\ h \quad \rightarrow \quad M[v/x],\ h$$

$$a = a',\ h \quad \rightarrow \quad 0/1,\ h$$

$a,a' \in \mathsf{Loc}_\theta$

$$\mathsf{ref}\ 0,\ h \quad \rightarrow \quad a,\ h \uplus [a \mapsto 0]$$

$a \in \mathsf{Loc}_{\mathsf{int}}$

$$\mathsf{ref}\ (\lambda x.x{+}1),\ h \quad \rightarrow \quad a,\ h \uplus [a \mapsto \lambda x.x{+}1]$$

$a \in \mathsf{Loc}_{\mathsf{int}\rightarrow\mathsf{int}}$

Full abstraction

Build model $\mathcal{M}$ and denotation map

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

such that:

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

Why?

- reasoning about programs → model checking [e.g. model checking equivalence]

$M \cong M'$: *same observable behaviour in every context*

Full abstraction

Build model $\mathcal{M}$ and denotation map

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

such that:

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

Why?

- reasoning about programs → model checking [e.g. model checking equivalence]

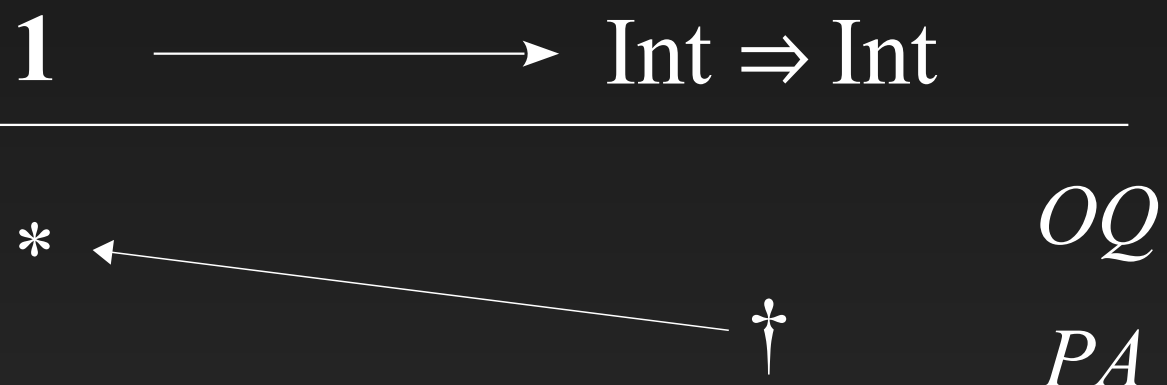
$M \cong M'$: *mutual termination* ***in every context***

Nominal game semantics

- Computation is a 2-player game between:
  - *Opponent* (the environment), aka *O*
  - *Proponent* (the program), aka *P*
- Moves of the game are:
  - function calls, aka *Questions*
  - function returns, aka *Answers*
- Programs = *strategies* for *P*

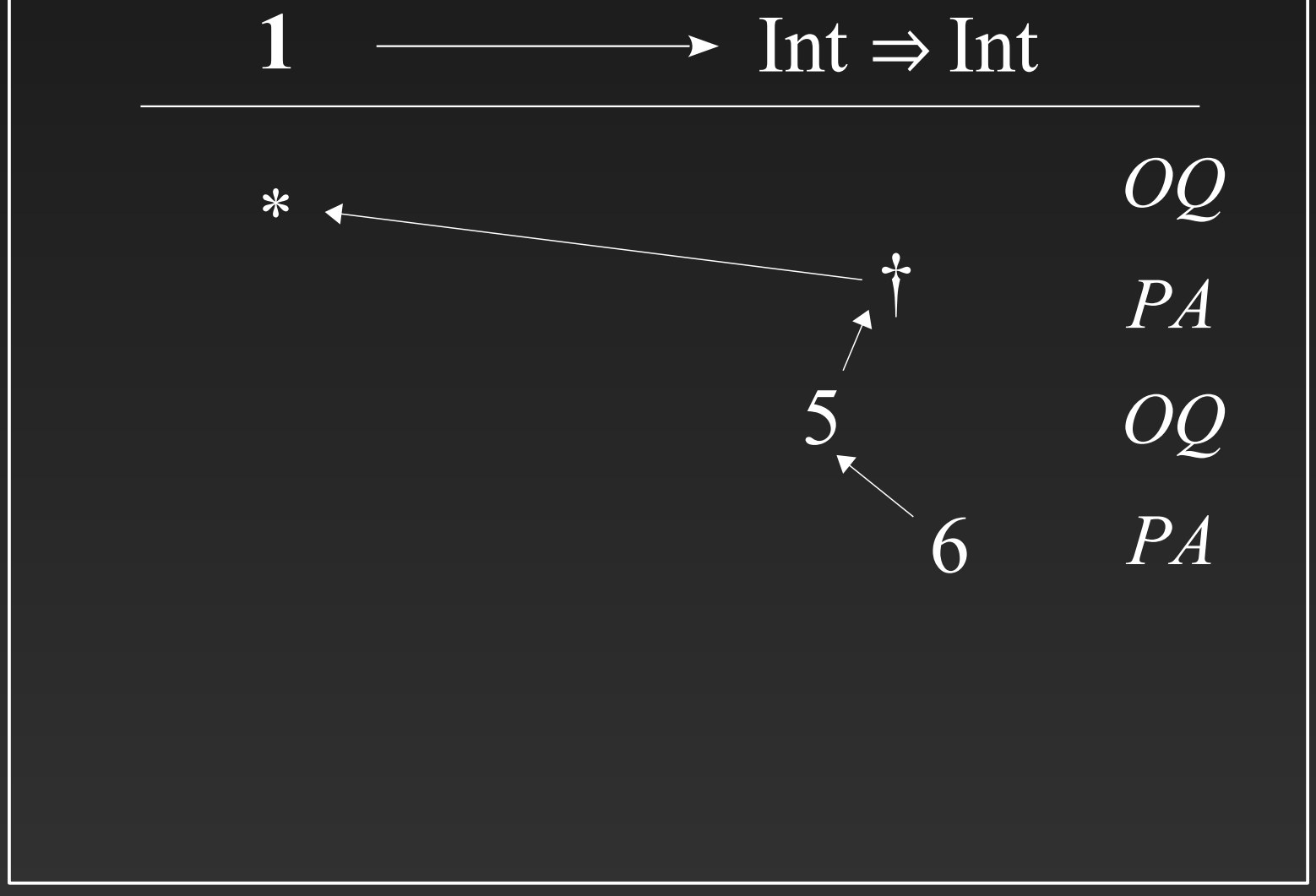
Example games

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



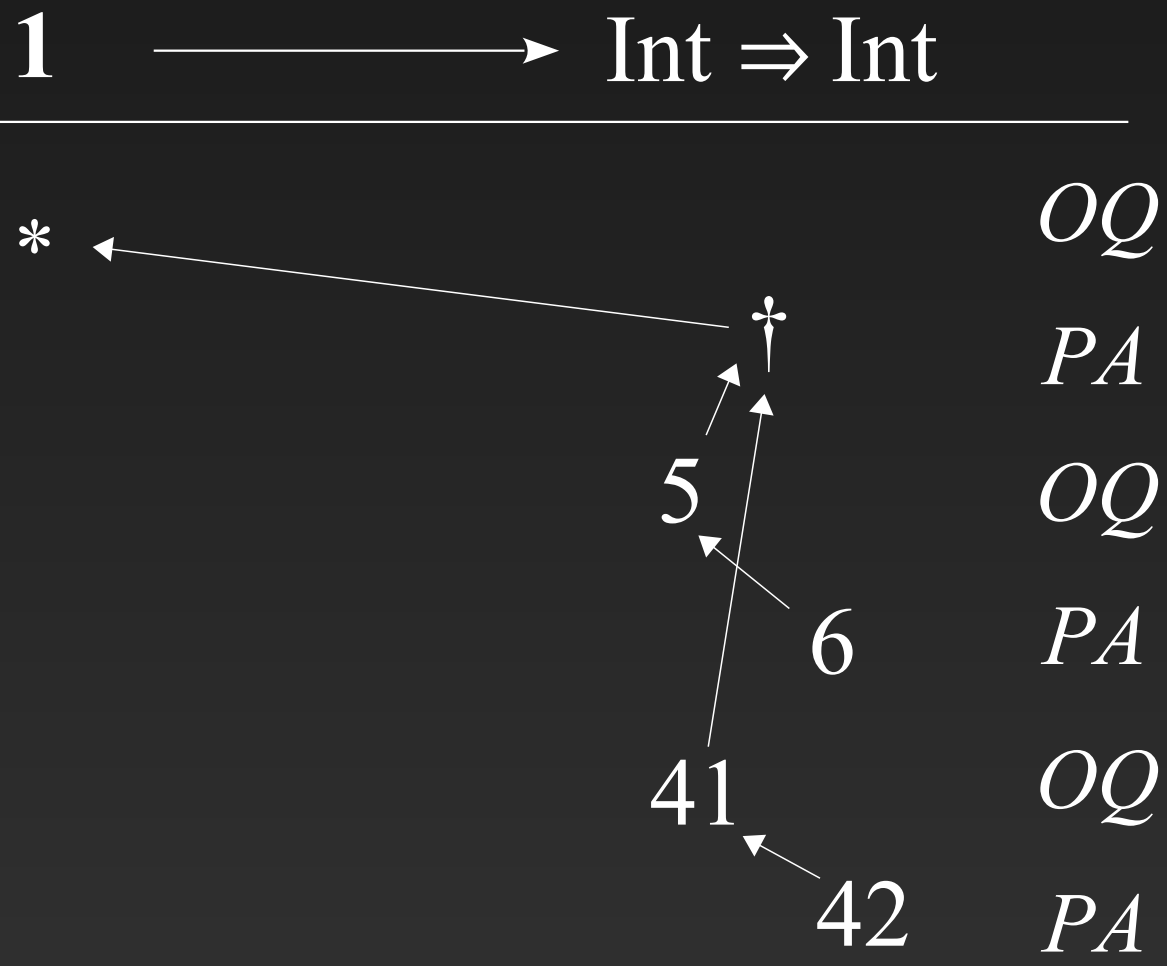
Example games

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



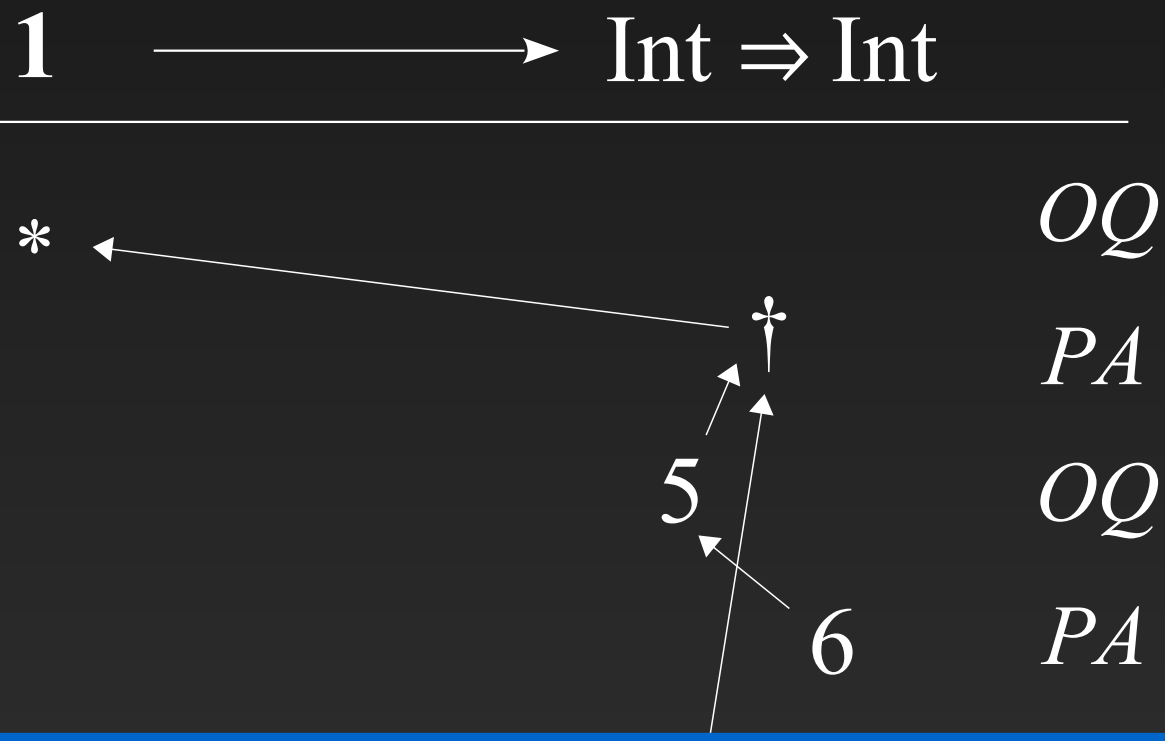
Example games

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



Example games

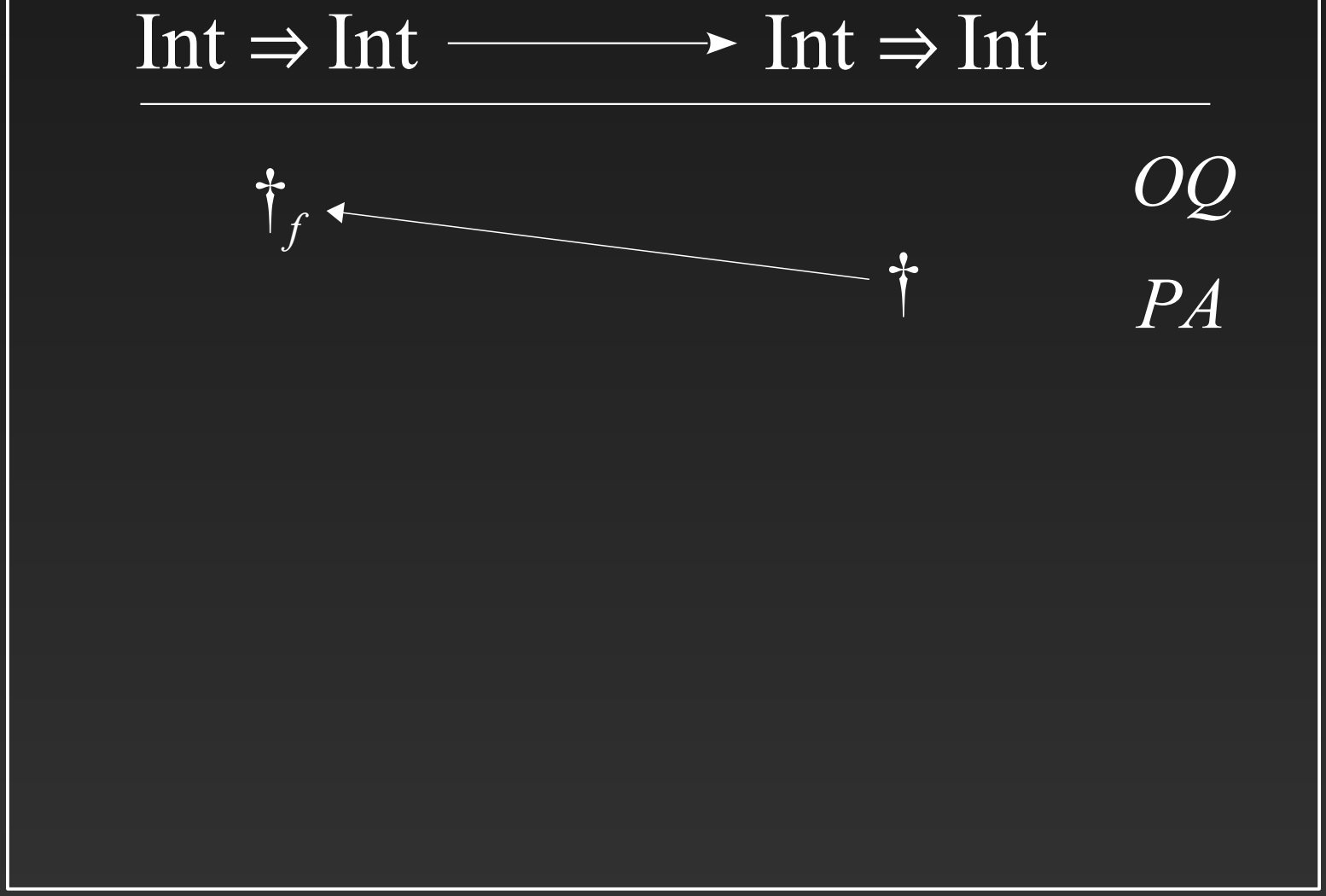
$\vdash \lambda x.\, x{+}1 : \mathsf{int} \to \mathsf{int}$



$[\![\lambda x.x{+}1]\!] = \{ \; * \; \dagger \; i \; i{+}1 \; \ldots \; \}$

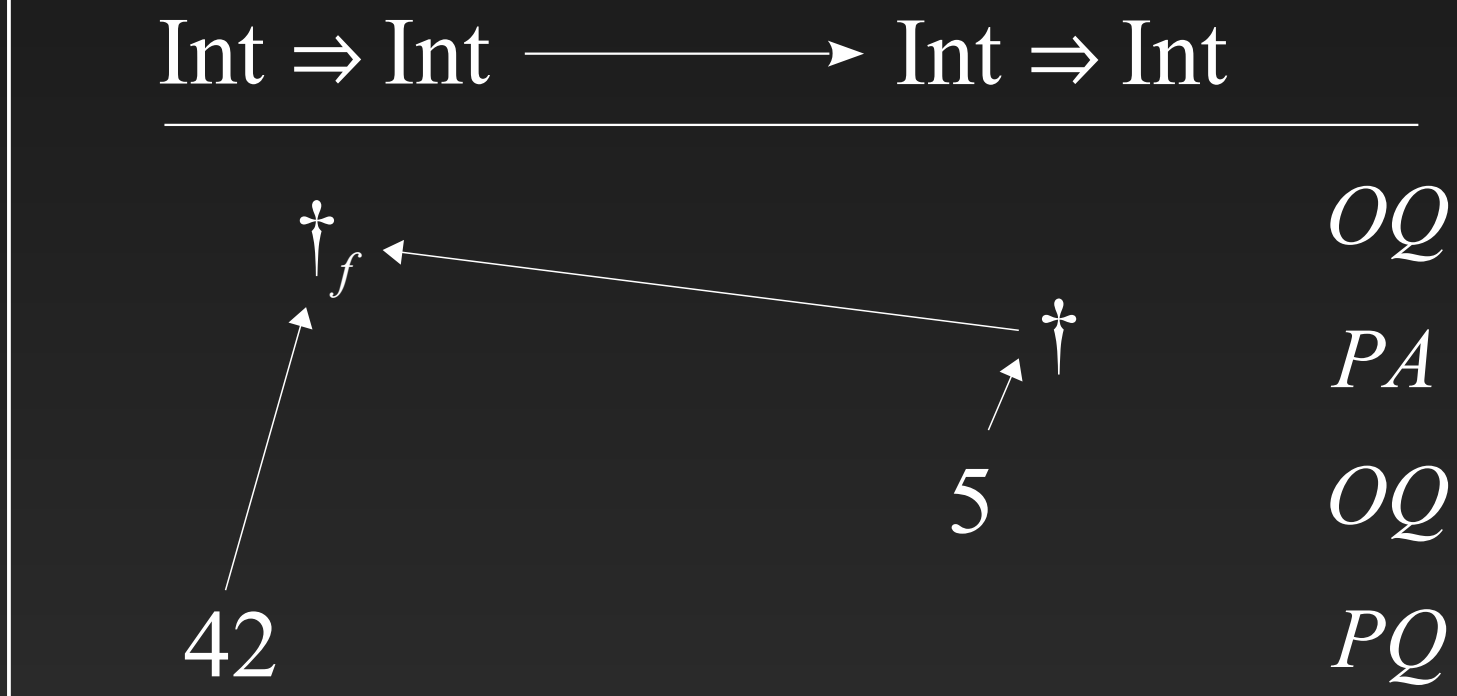
Example games

$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$



Example games

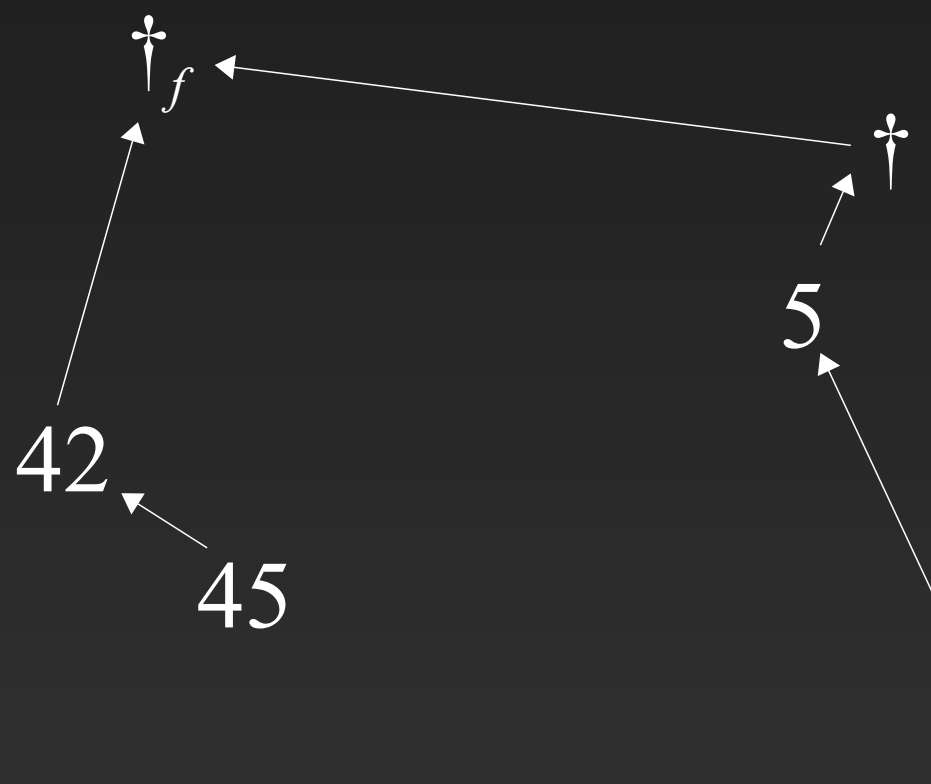
$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$



Example games

$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$

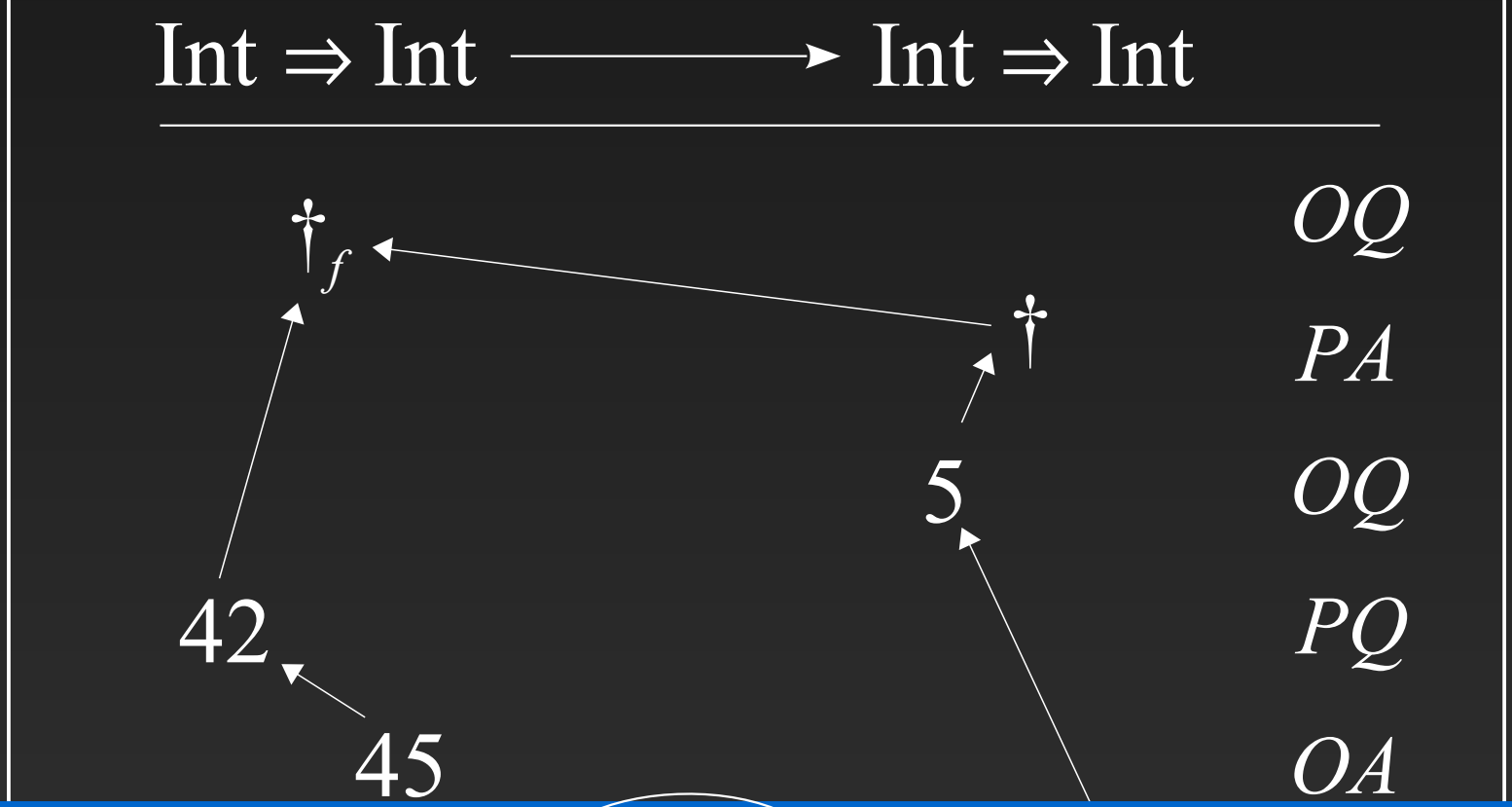
$\mathrm{Int} \Rightarrow \mathrm{Int} \longrightarrow \mathrm{Int} \Rightarrow \mathrm{Int}$



OQ
PA
OQ
PQ
OA
PA

Example games

$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$

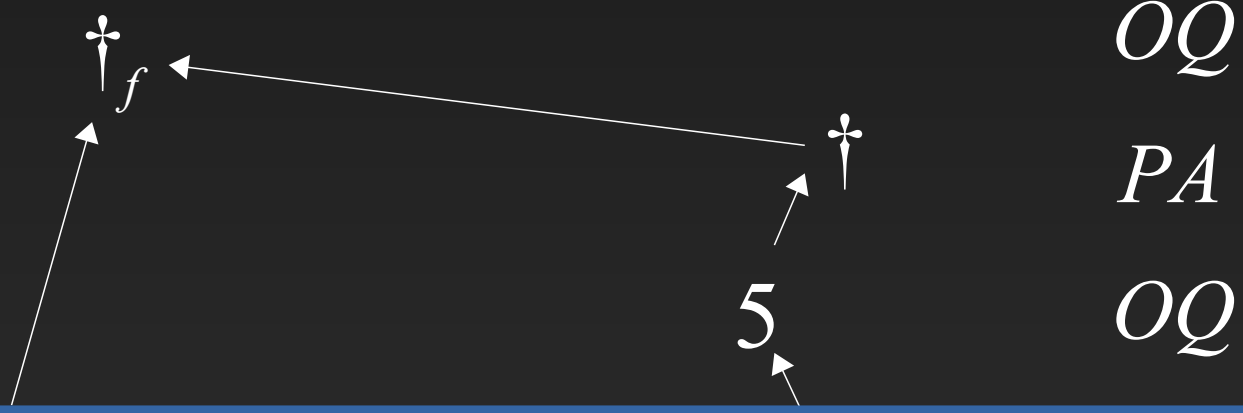


$[\![\lambda x.f(x{+}37)]\!] = \{\ \dagger_f\ \dagger\ i\ (i{+}37)\ j\ j\ \ldots\ \}$

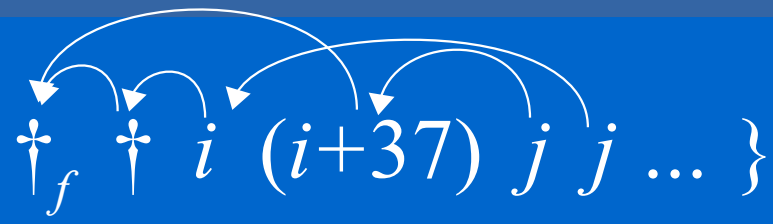
Example games

$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$

$$\mathrm{Int} \Rightarrow \mathrm{Int} \longrightarrow \mathrm{Int} \Rightarrow \mathrm{Int}$$



$f \quad g \quad \mathrm{call}\, g(i) \quad \mathrm{call}\, f(i{+}37) \quad \mathrm{ret}\, f(j) \quad \mathrm{ret}\, g(j)$ ...

$[\![\lambda x.f(x{+}37)]\!] = \{$  ... $\}$

Nominal game semantics

- Computation is a 2-player game between:
  - *Opponent* (the environment), aka *O*
  - *Proponent* (the program), aka *P*
- Moves of the game are:
  - function calls, aka *Questions*
  - function returns, aka *Answers*
- Programs = *strategies* for *P*
- Strategies constructed via composition

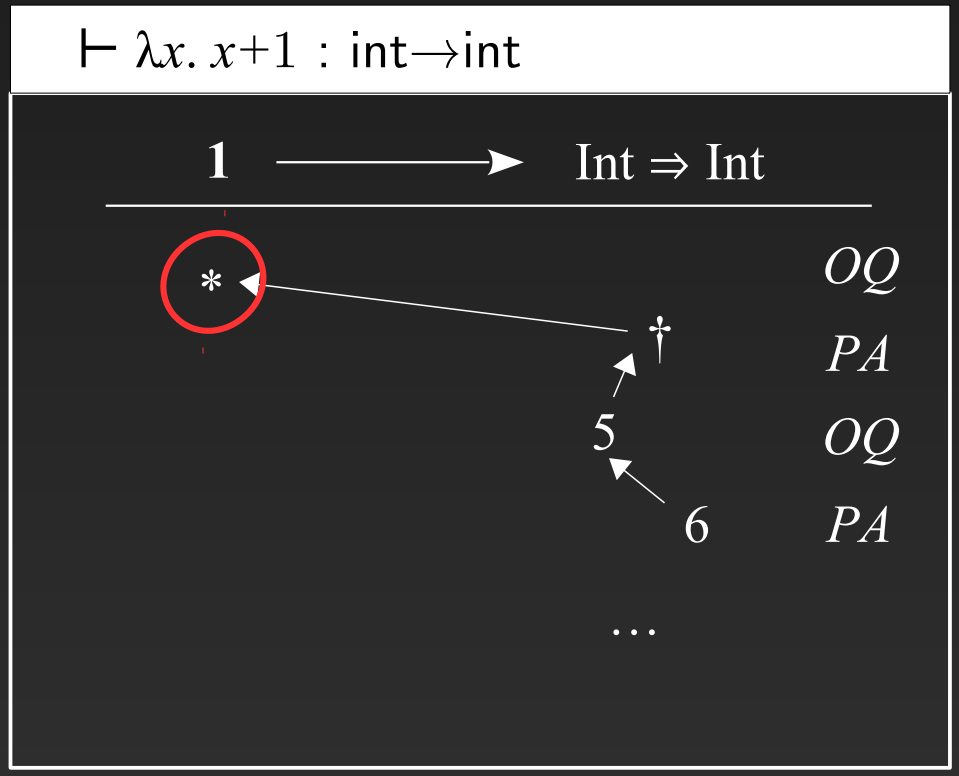
Composition

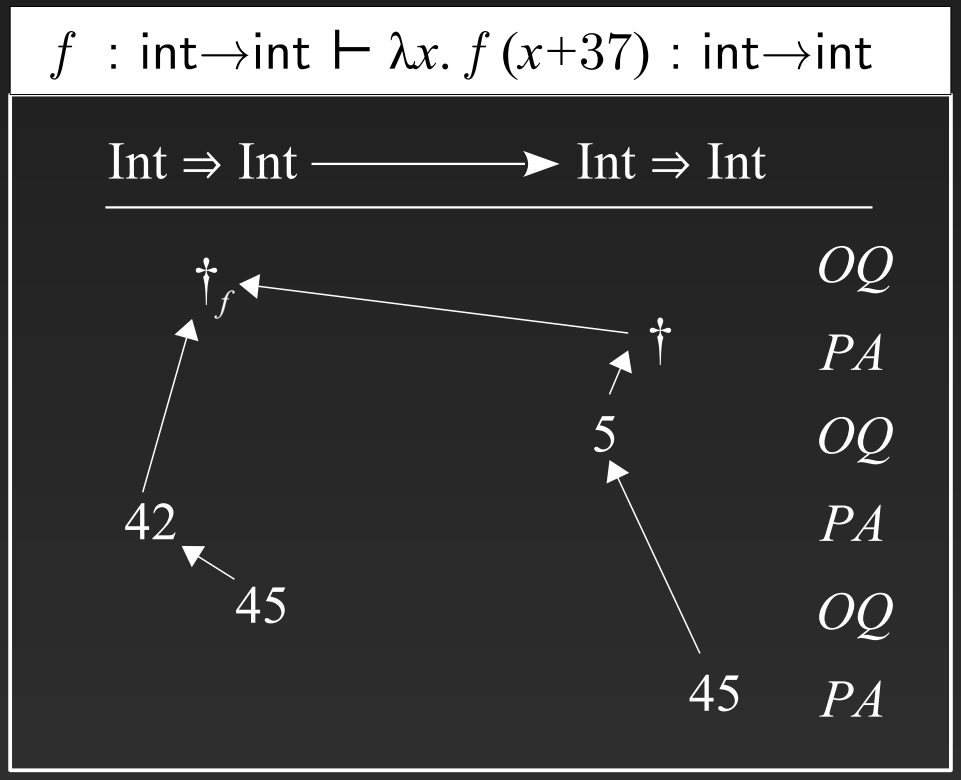
Strategies are composed by matching O/P behaviours

Composition

Strategies are composed by matching O/P behaviours

→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int

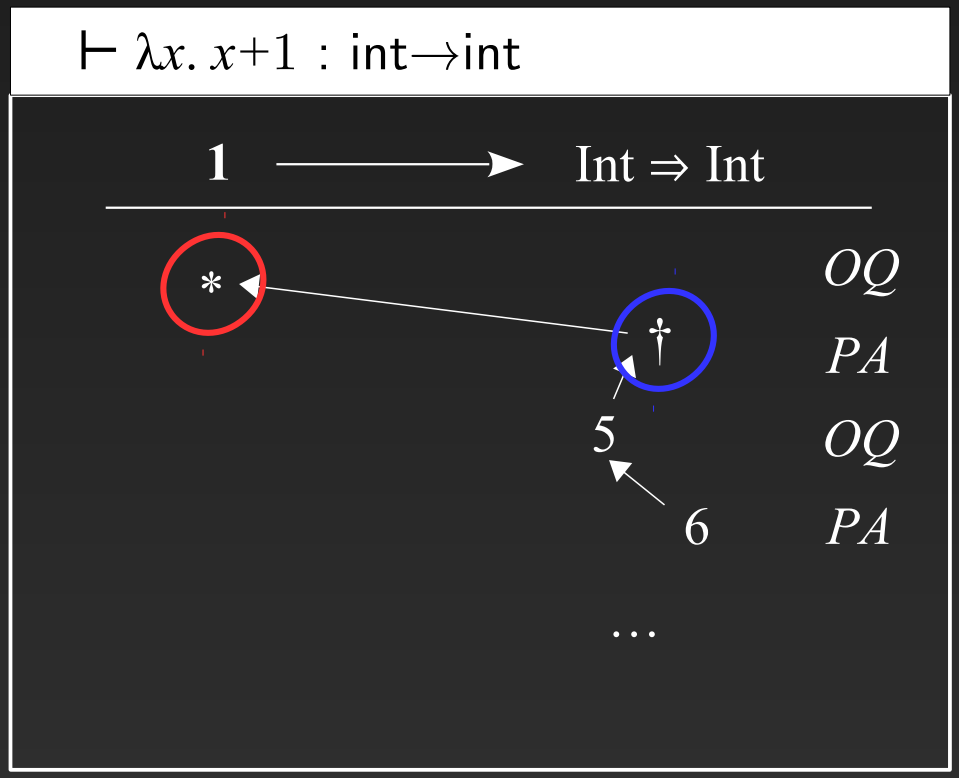


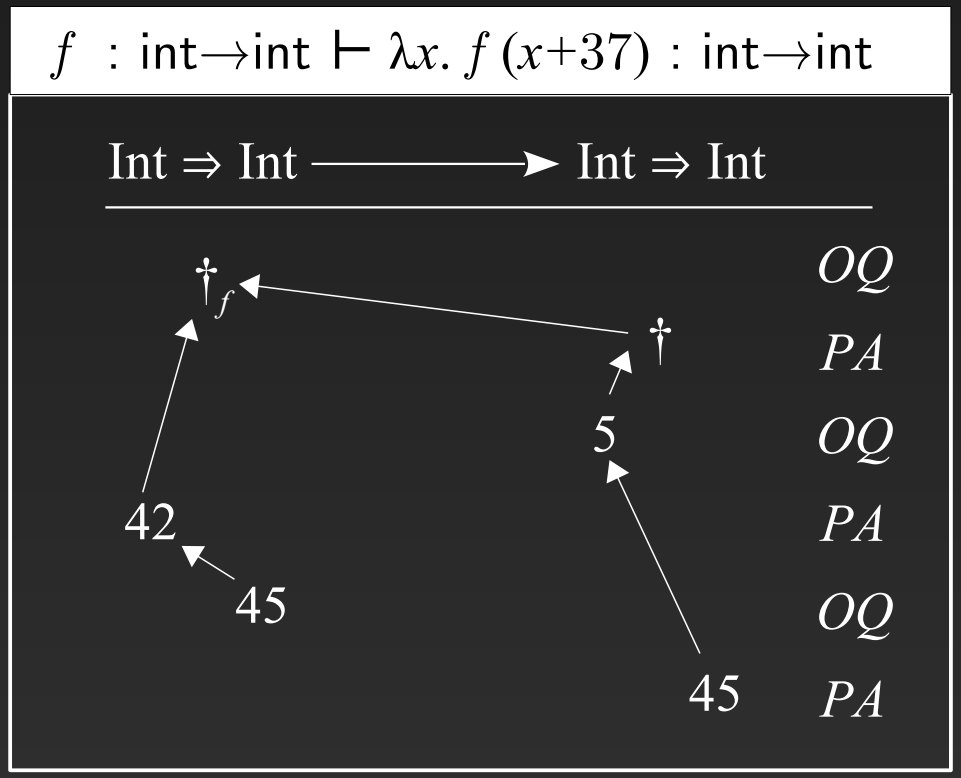


Composition

Strategies are composed by matching O/P behaviours

→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int

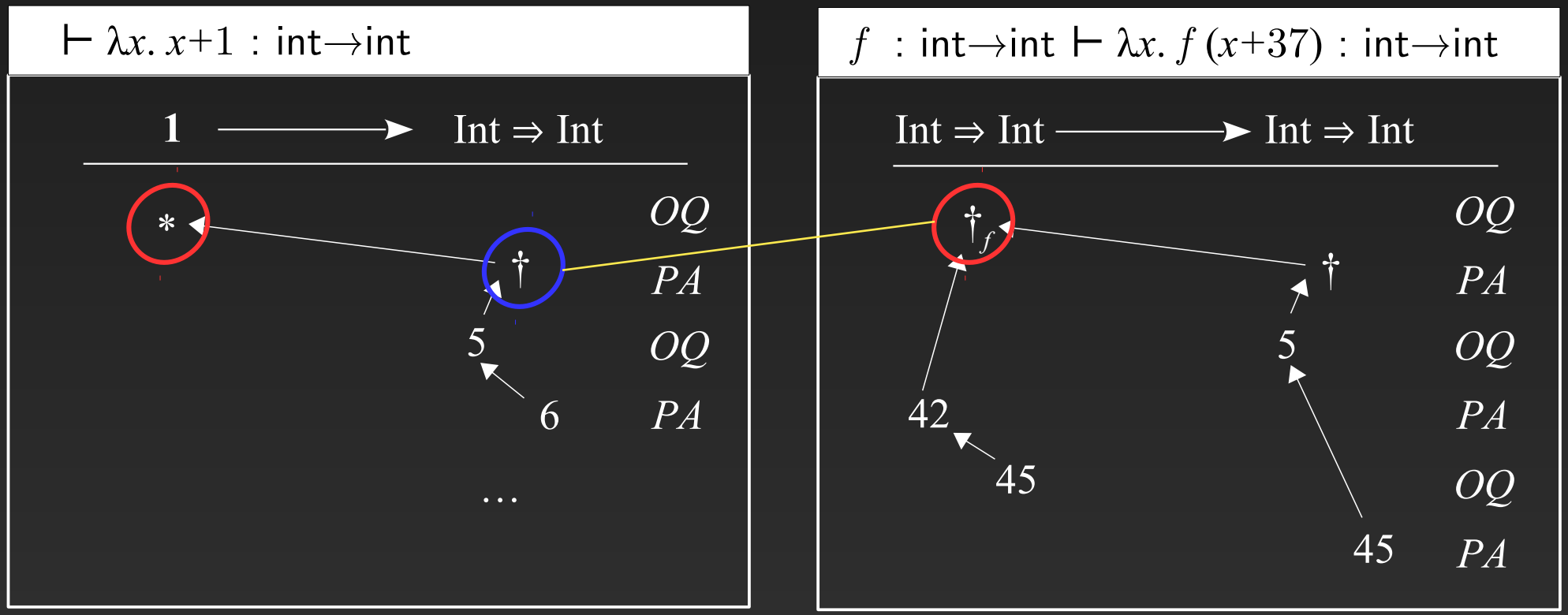




Composition

Strategies are composed by matching O/P behaviours

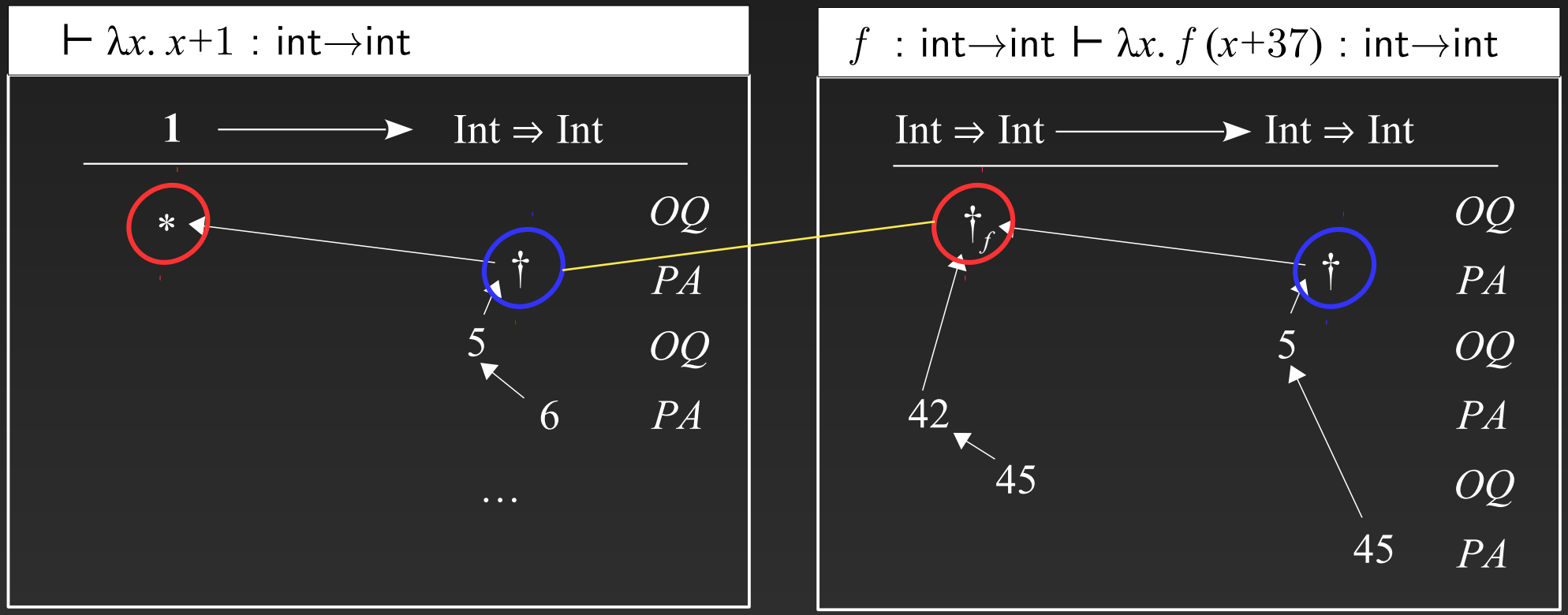
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

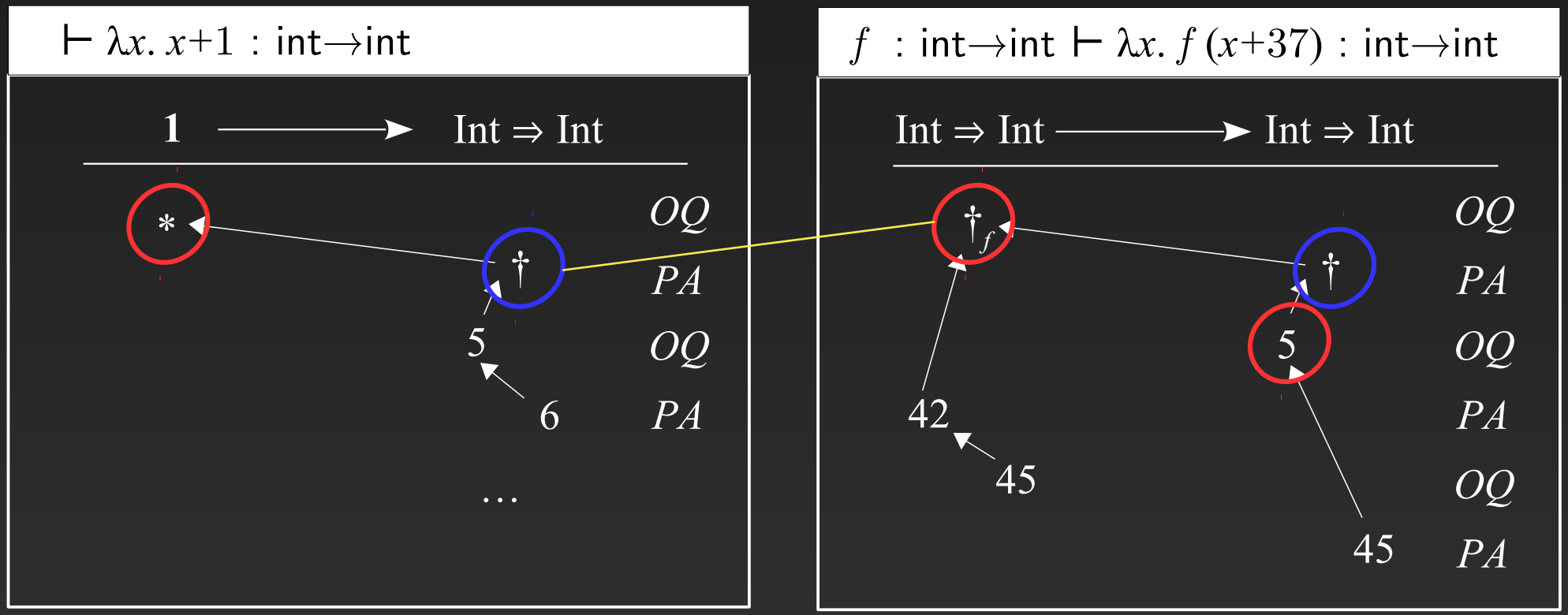
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

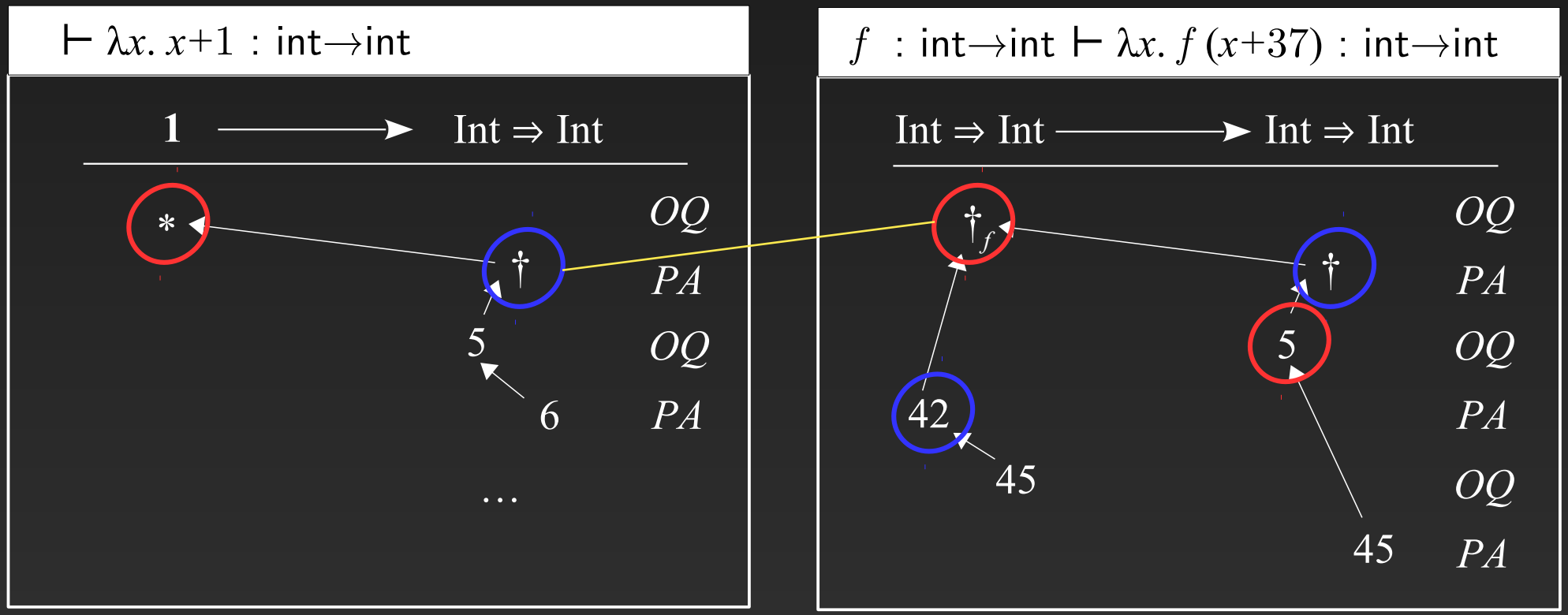
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

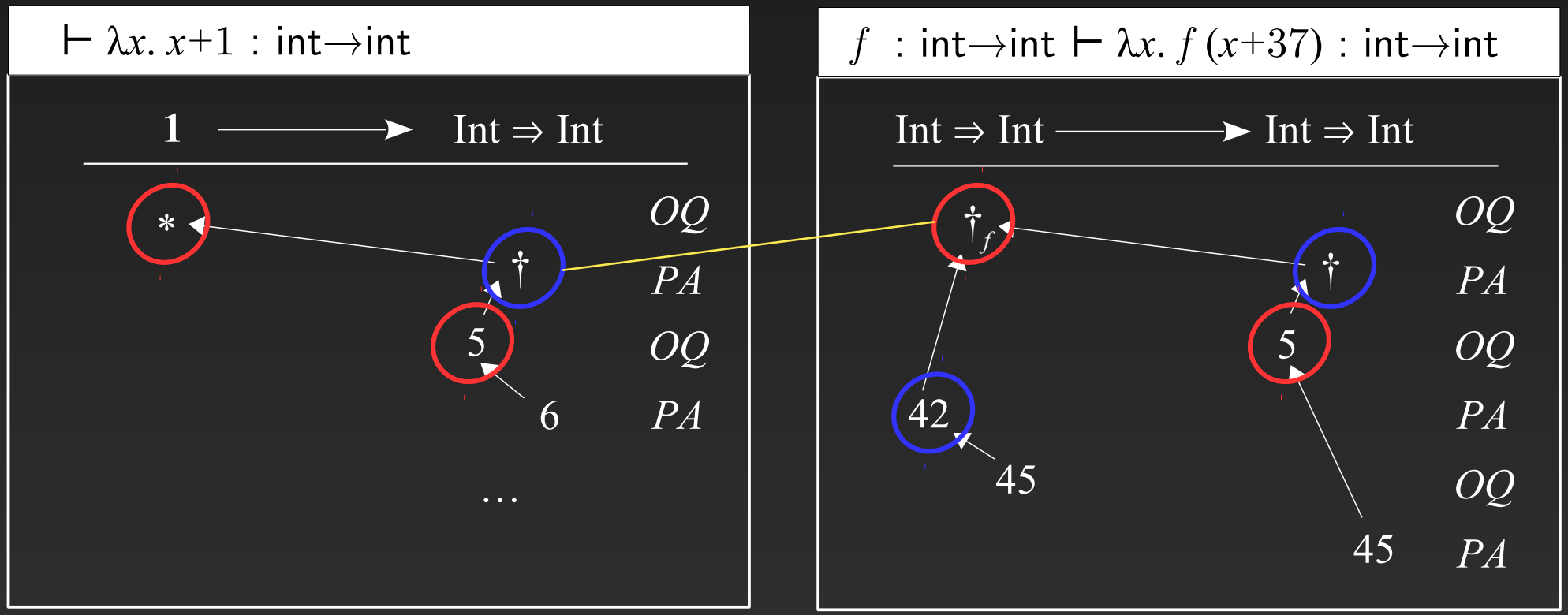
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

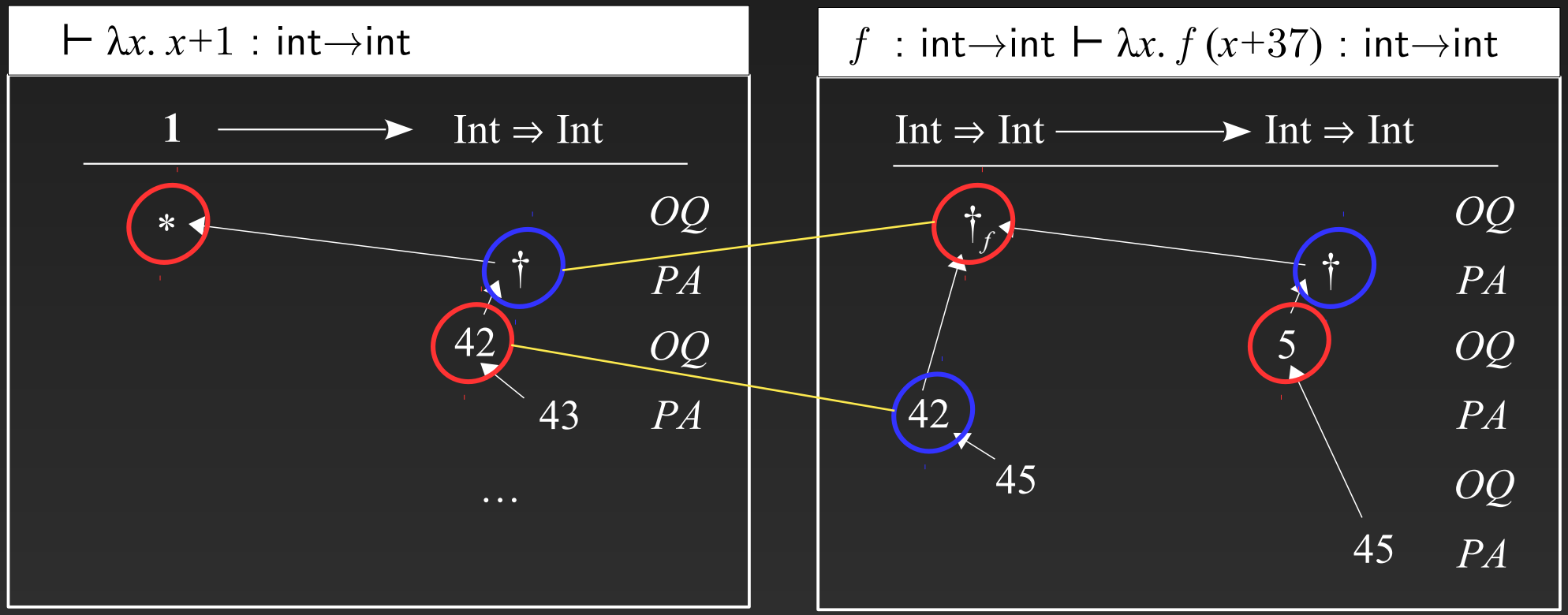
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

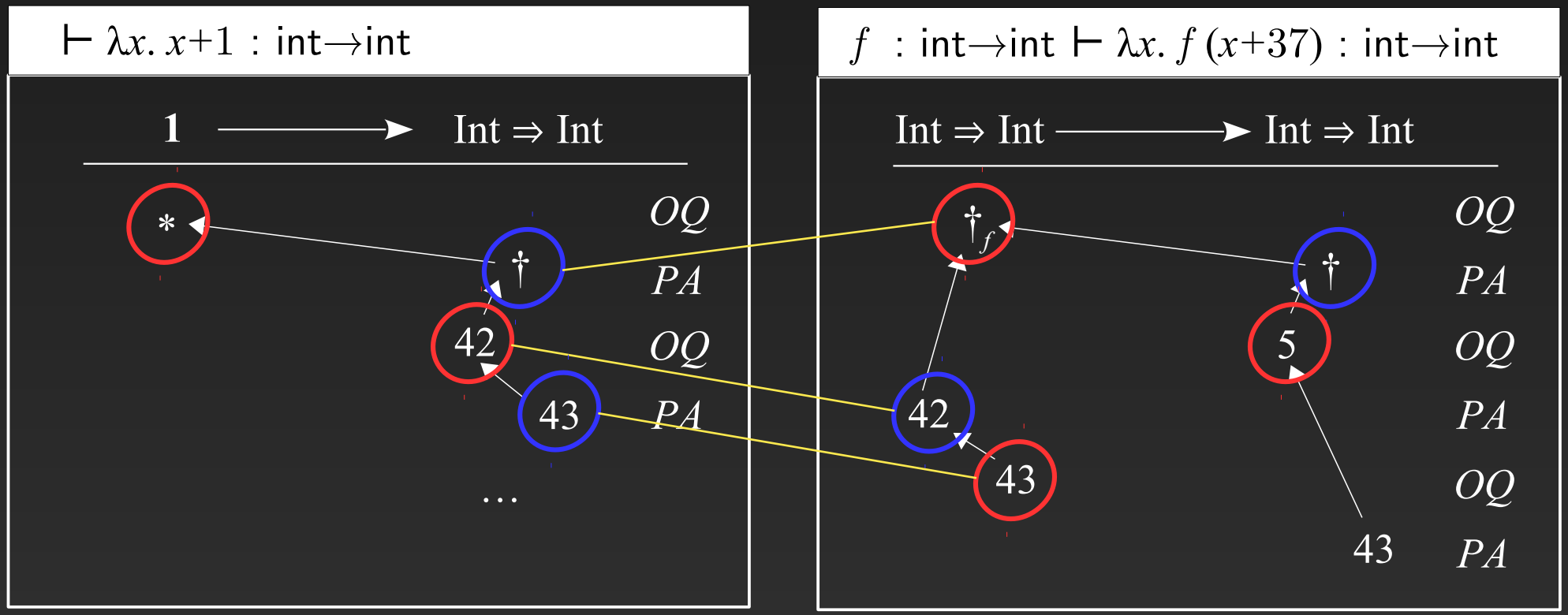
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

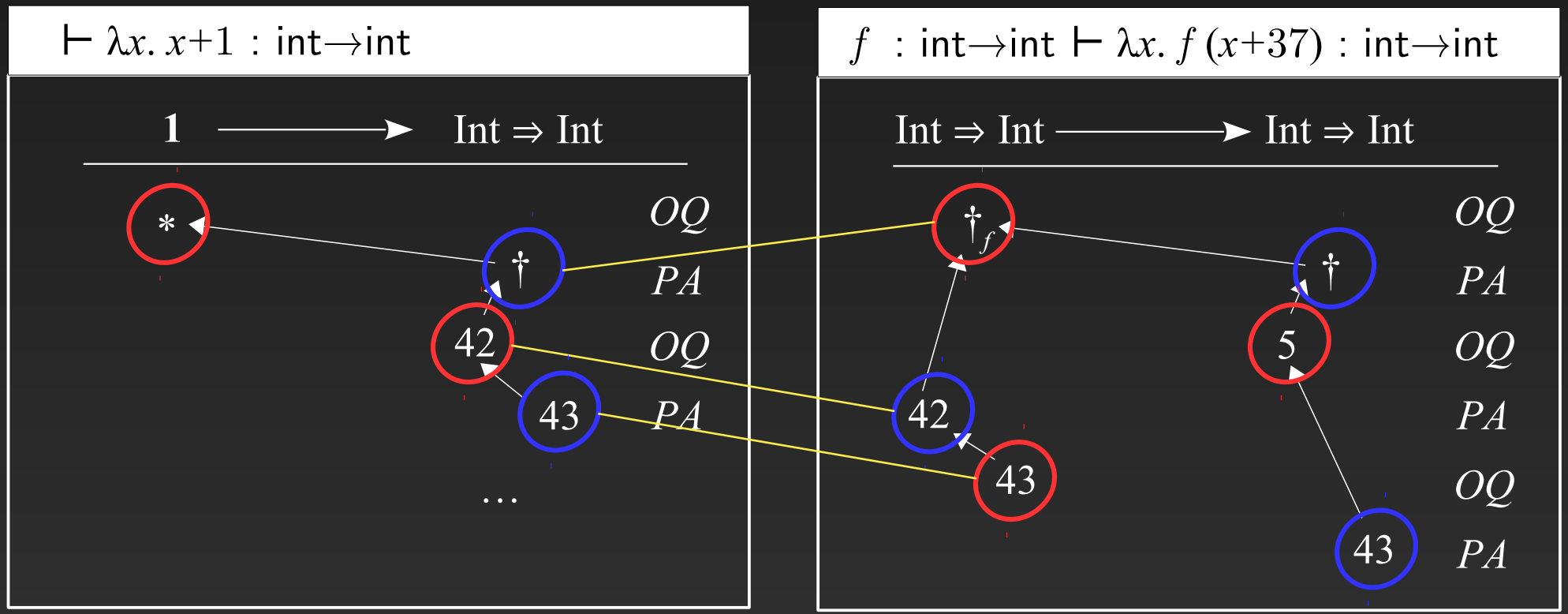
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

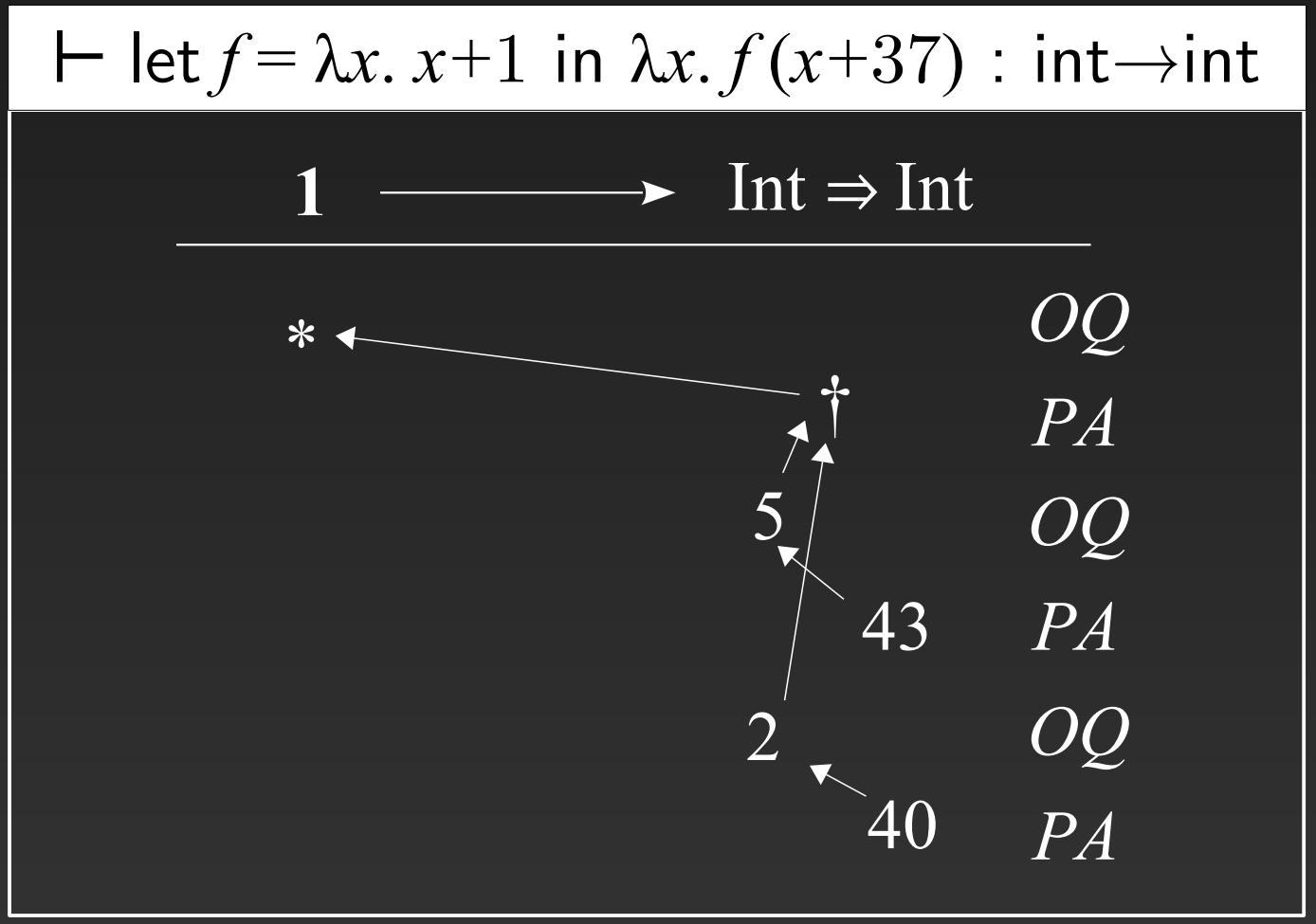
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies are composed by matching O/P behaviours

→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int

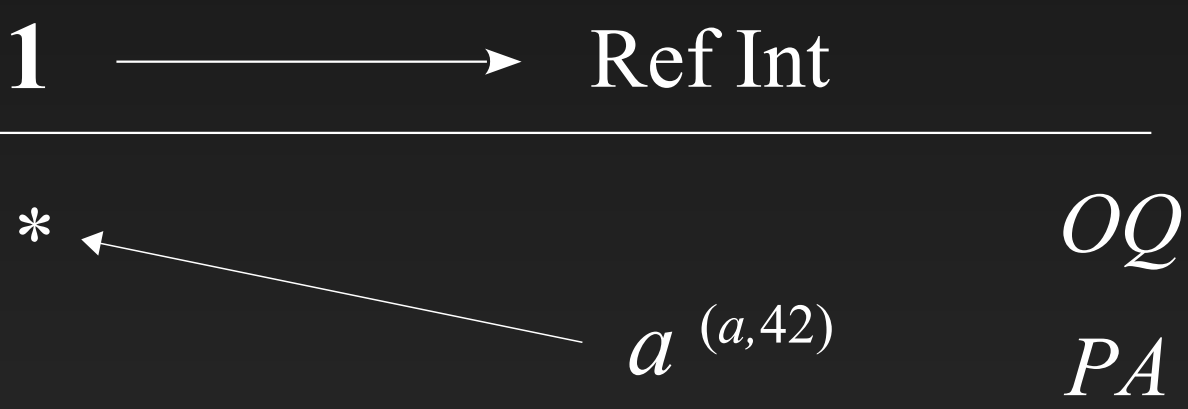


Nominal game semantics

- Computation is a 2-player game between:
  - *Opponent* (the environment), aka *O*
  - *Proponent* (the program), aka *P*
- Moves of the game are:
  - function calls, aka *Questions*
  - function returns, aka *Answers*
- Programs = *strategies* for *P*
- Strategies constructed via composition
- Games are nominal:
  - they contain abstract atomic data (*names*)
  - names: references (ML), objects & threads (Java), etc.

Nominal games

$\vdash$ ref 42 : ref int



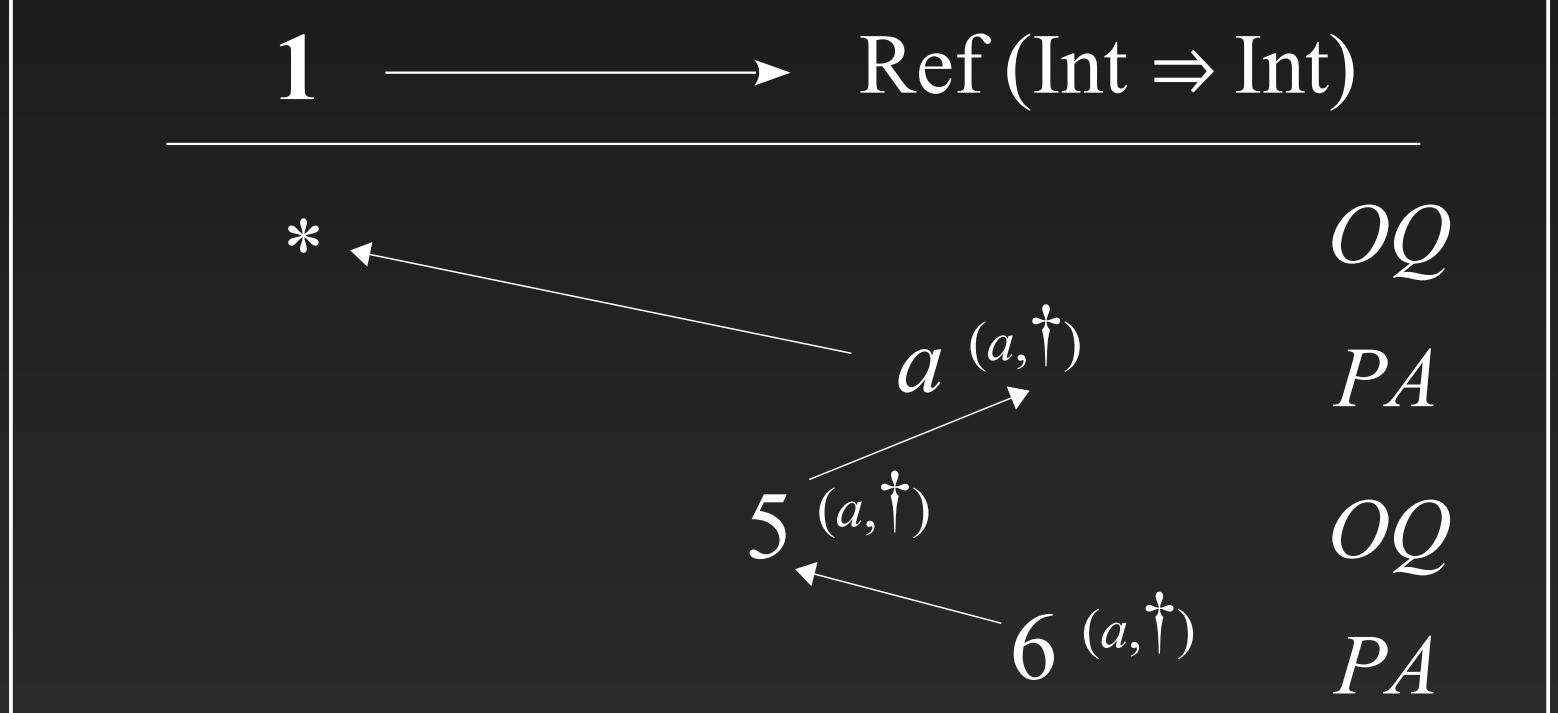
where: a is a name of type int $(a \in \mathcal{N}_{\text{int}})$

$$[\![\text{ref } 42]\!] = \{\ * \xleftarrow{a^{(a,42)}} *\ \}$$

$OQ \quad PA$

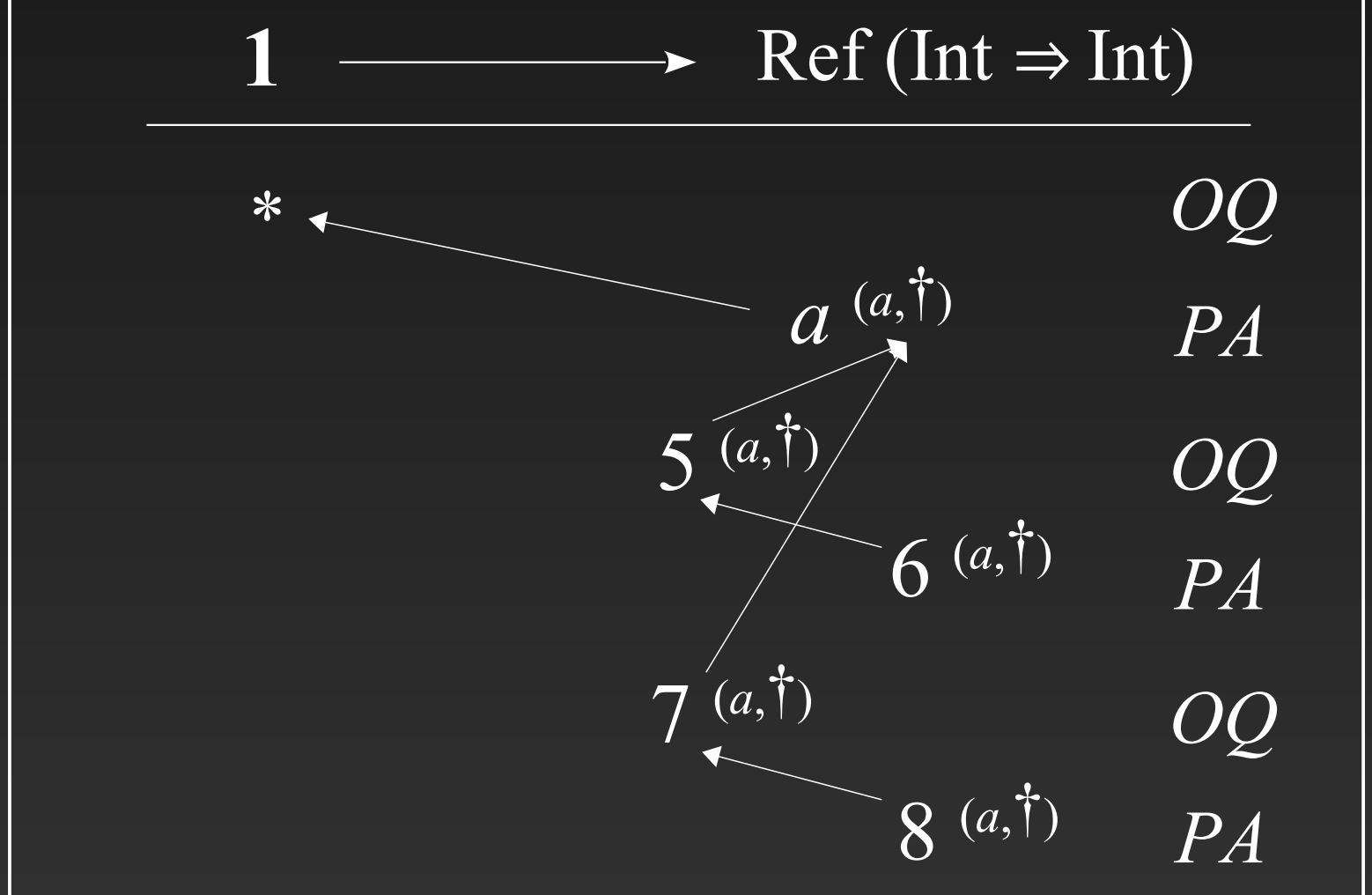
Nominal games

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x{+}1) : \mathsf{ref}\,(\mathsf{int}{\to}\mathsf{int})$



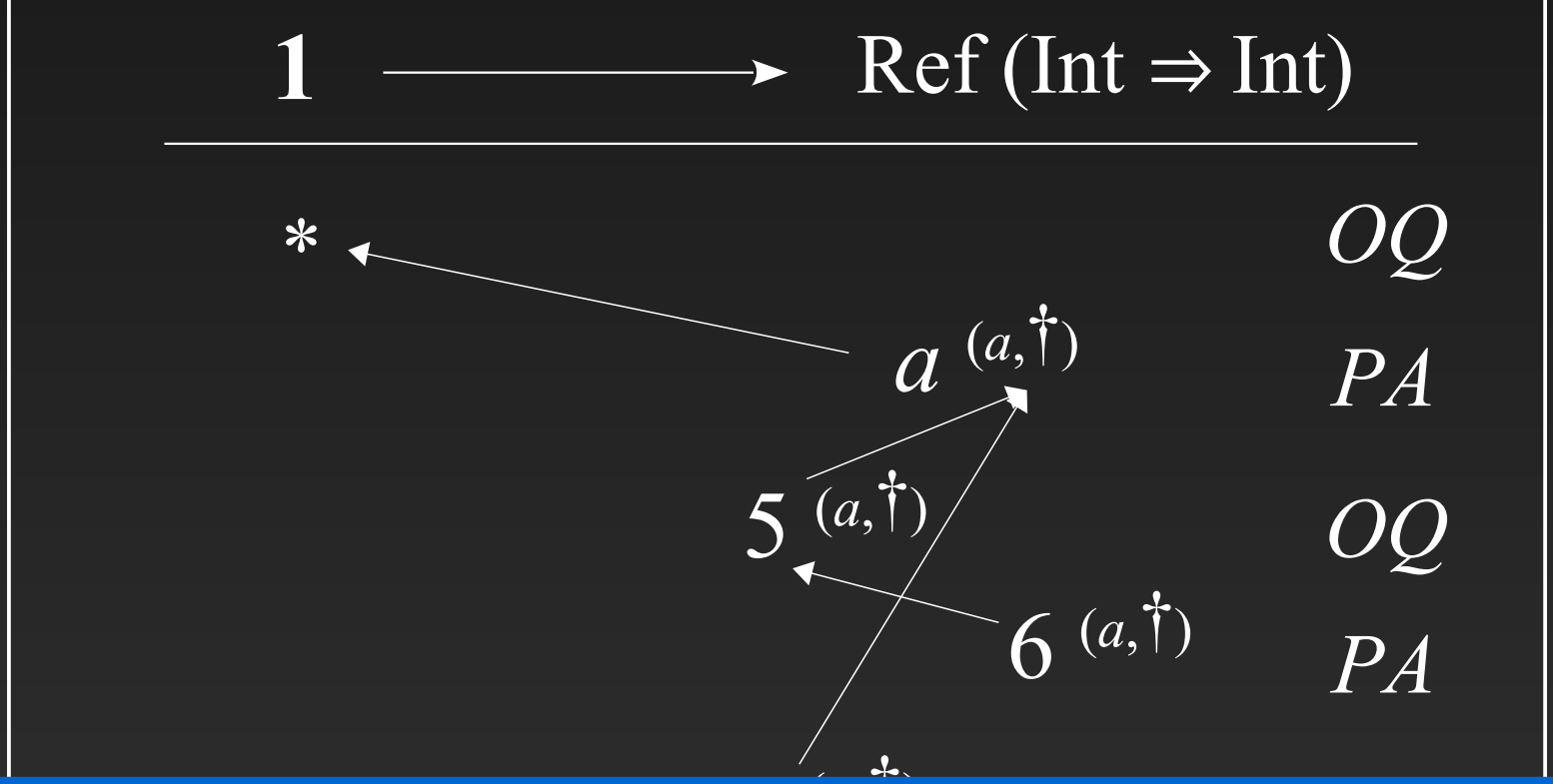
Nominal games

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x{+}1) : \mathsf{ref}\,(\mathsf{int}{\to}\mathsf{int})$



Nominal games

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x{+}1) : \mathsf{ref}\,(\mathsf{int}{\to}\mathsf{int})$

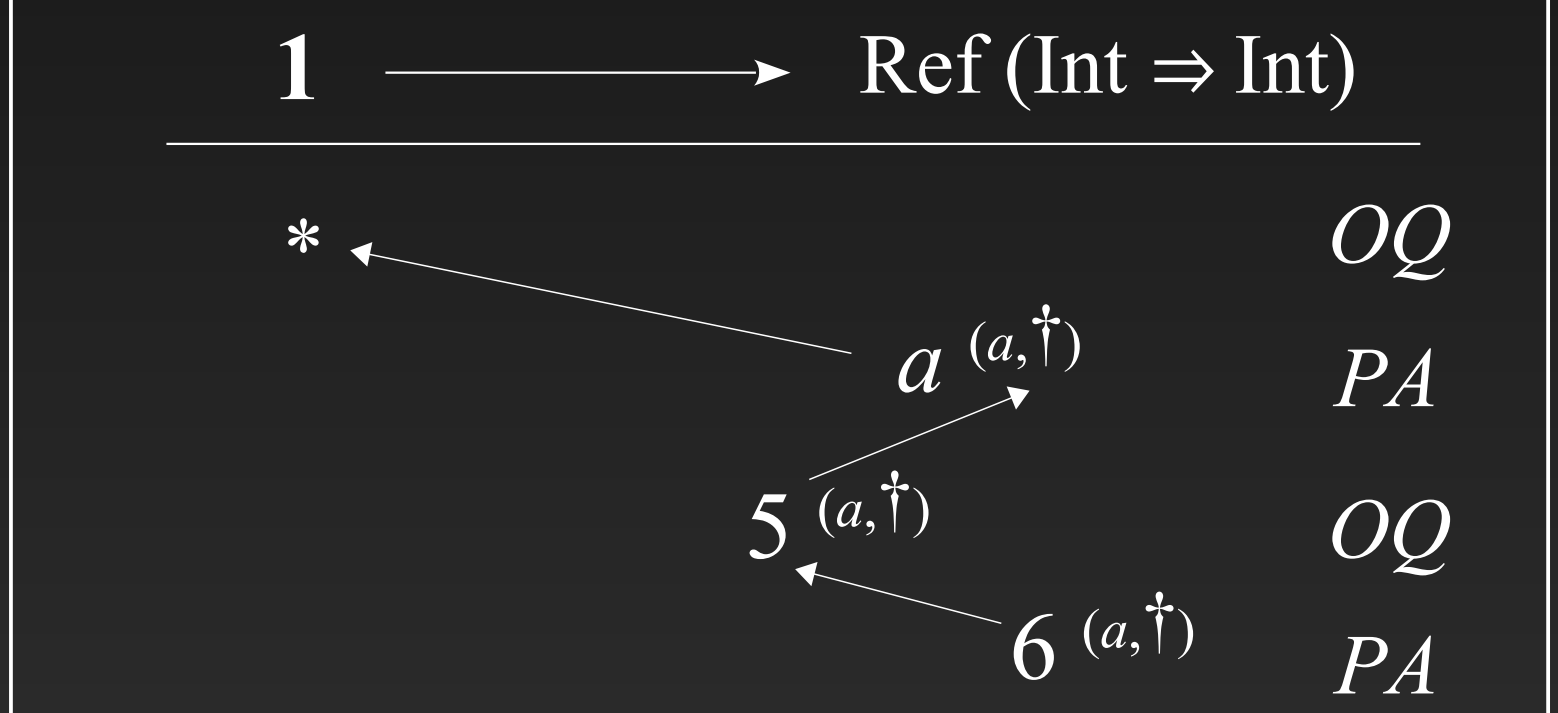


$[\![\mathsf{ref}\ \lambda x.x{+}1]\!] = \{\ *\ a^{(a,\dagger)}\ 5^{(a,\dagger)}\ 6^{(a,\dagger)}\ 7^{(a,\dagger)}\ 8^{(a,\dagger)}\ \ldots\ \}$

$OQ\ PA\quad OQ\quad PA\quad OQ\quad PA$

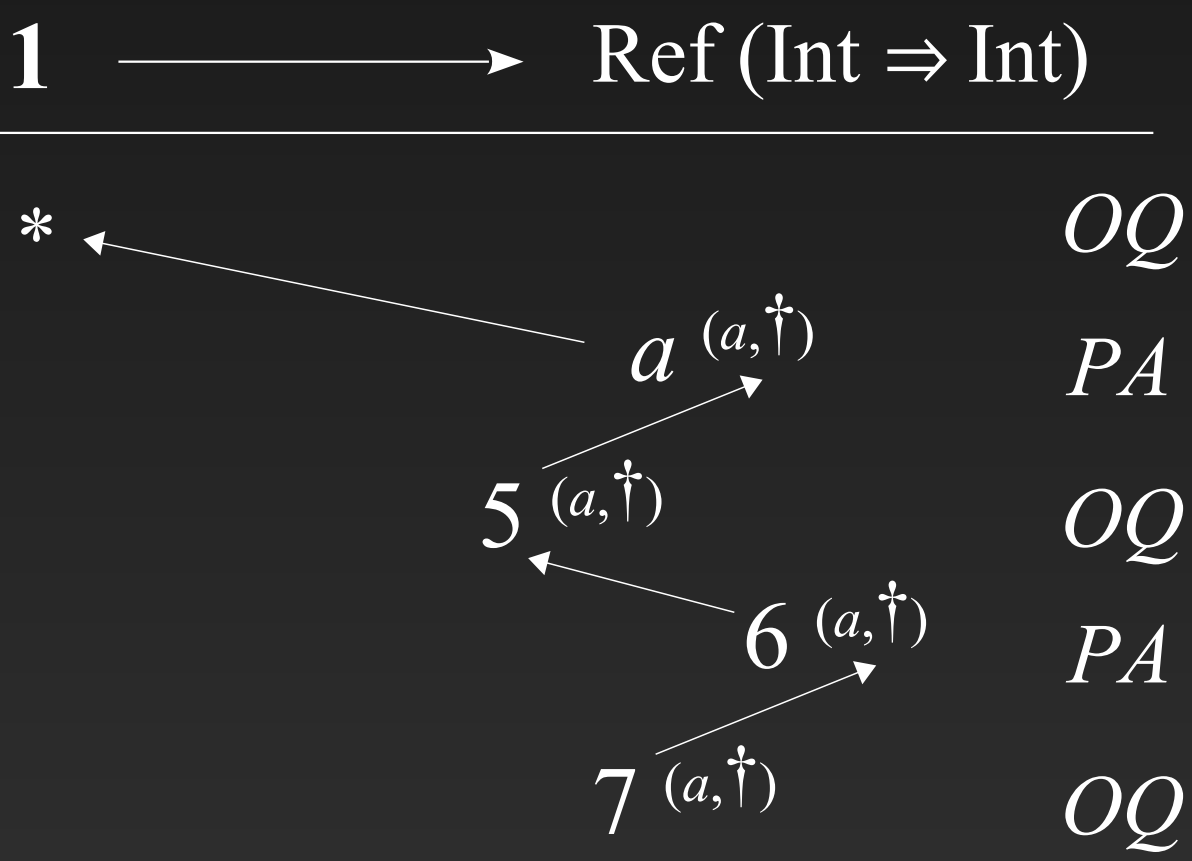
Nominal games

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x{+}1) : \mathsf{ref}\,(\mathsf{int}{\to}\mathsf{int})$



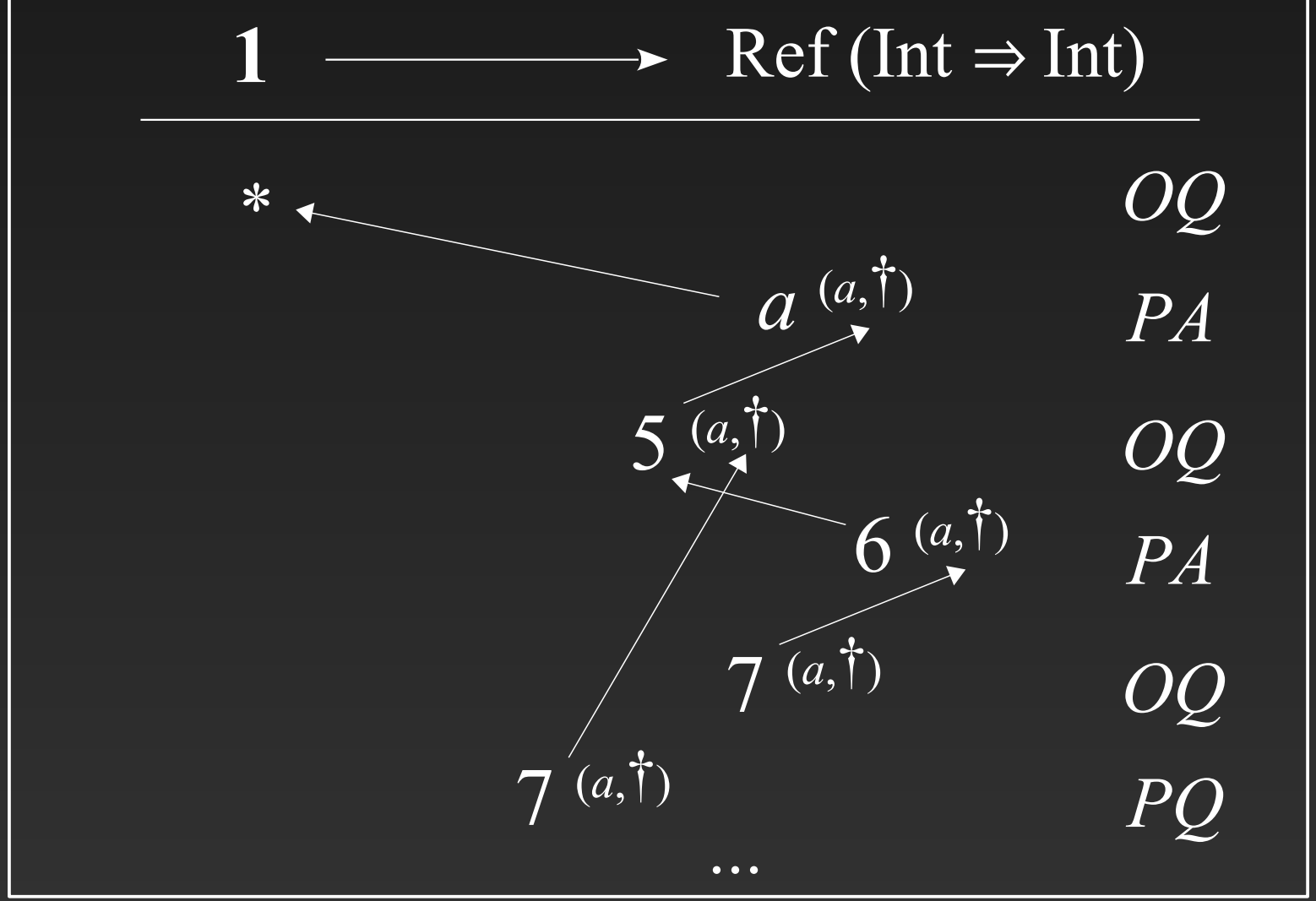
(there are more plays)

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x+1) : \mathsf{ref}\,(\mathsf{int}\to\mathsf{int})$



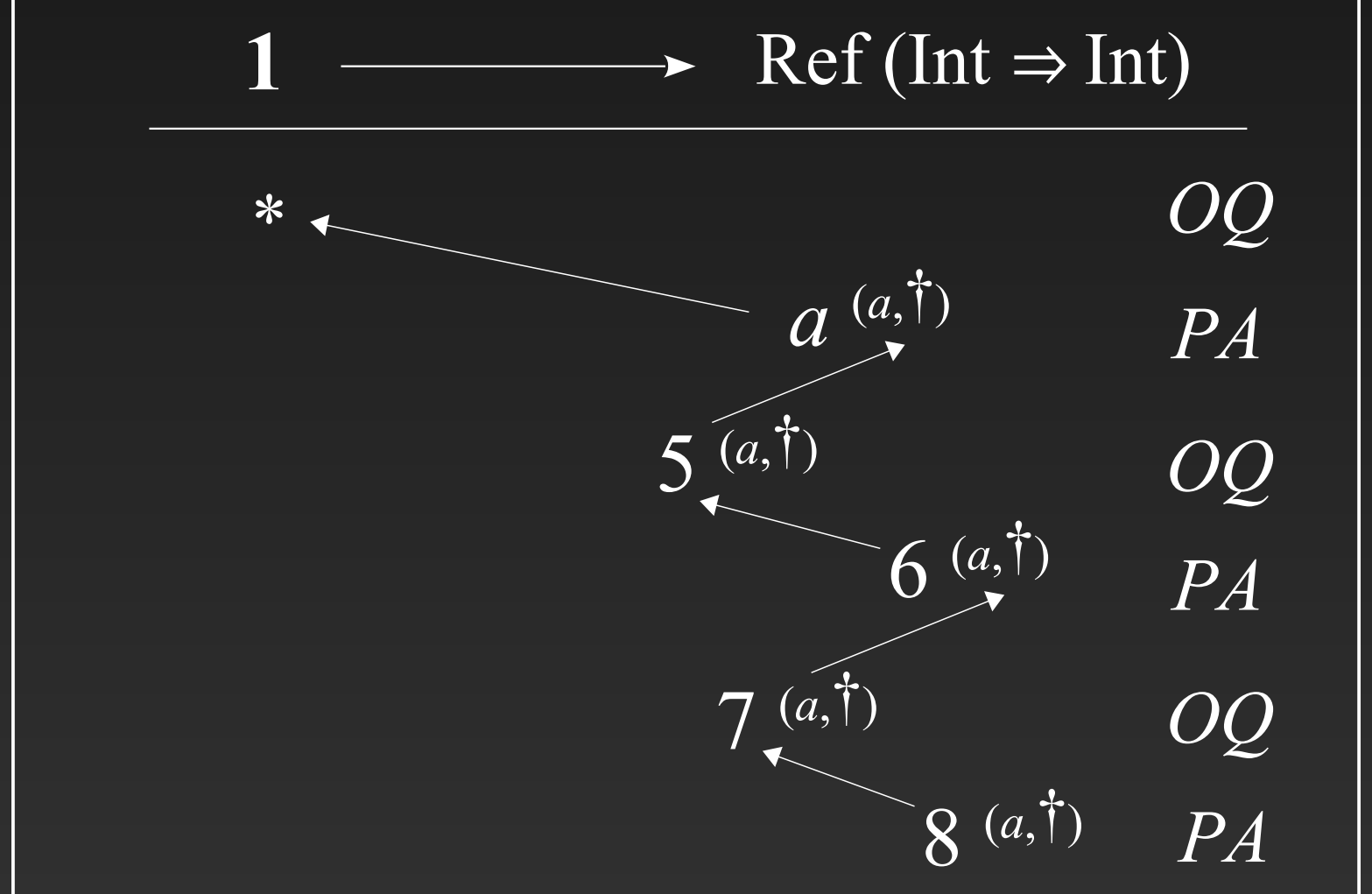
(there are more plays)

$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x+1) : \mathsf{ref}\,(\mathsf{int}\to\mathsf{int})$

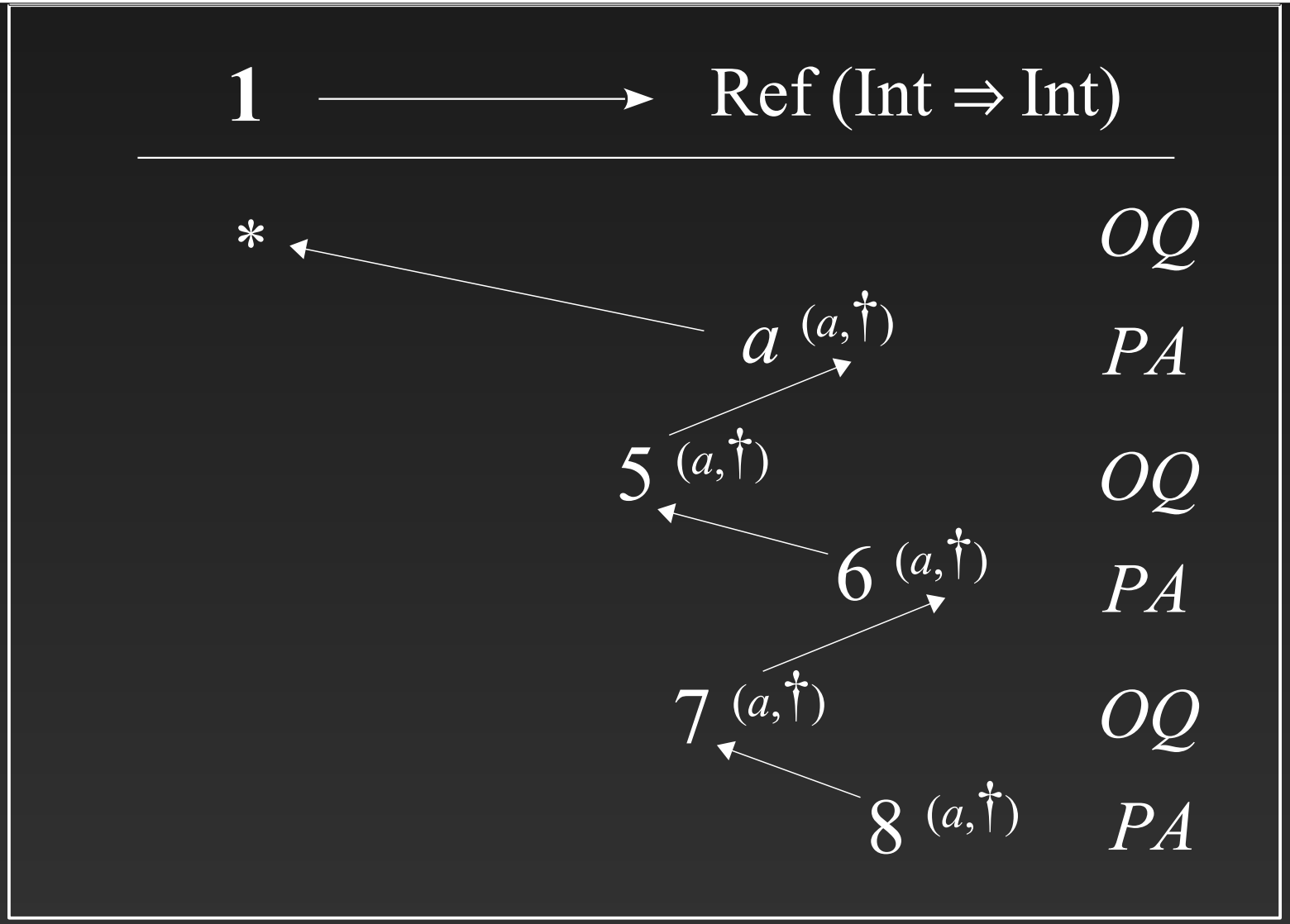


Quiz

$\vdash$??? : ref (int$\rightarrow$int)



$\vdash$ let l = ref $\lambda x^{\mathsf{int}}.x$ in
$(l := \lambda x^{\mathsf{int}}.(l := \lambda y^{\mathsf{int}}.y{+}1);\ x{+}1);\ l$: ref(int$\to$int)



Full abstraction

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

e.g:

Inc = let c = ref 0 in $\lambda z^{\mathsf{unit}}.\ c{+}{+};\ !c$

$x :=$ Inc; $y := !x$; $(!x)() + (!y)()$

$x :=$ Inc; $y := !x$; 3

$x :=$ Inc; $y :=$ Inc; $(!x)() + (!y)()$

[Laird APAL'08, Murawski & T. LICS'11, ICALP'12, POPL'14]

Games for Polymorphism

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{}{\Gamma \vdash \mathsf{if} : \mathsf{int} \to \vartheta \to \vartheta \to \vartheta} \qquad \frac{}{\Gamma \vdash \oplus : \mathsf{int} \to \mathsf{int}}$$

$$\frac{}{\Gamma, x : \vartheta \vdash x : \vartheta} \qquad \frac{\Gamma, x : \vartheta \vdash M : \vartheta'}{\Gamma \vdash \lambda x^{\vartheta}.M : \vartheta \to \vartheta'} \qquad \frac{\Gamma \vdash M : \vartheta \to \vartheta' \quad \Gamma \vdash N : \vartheta}{\Gamma \vdash M\,N : \vartheta'}$$

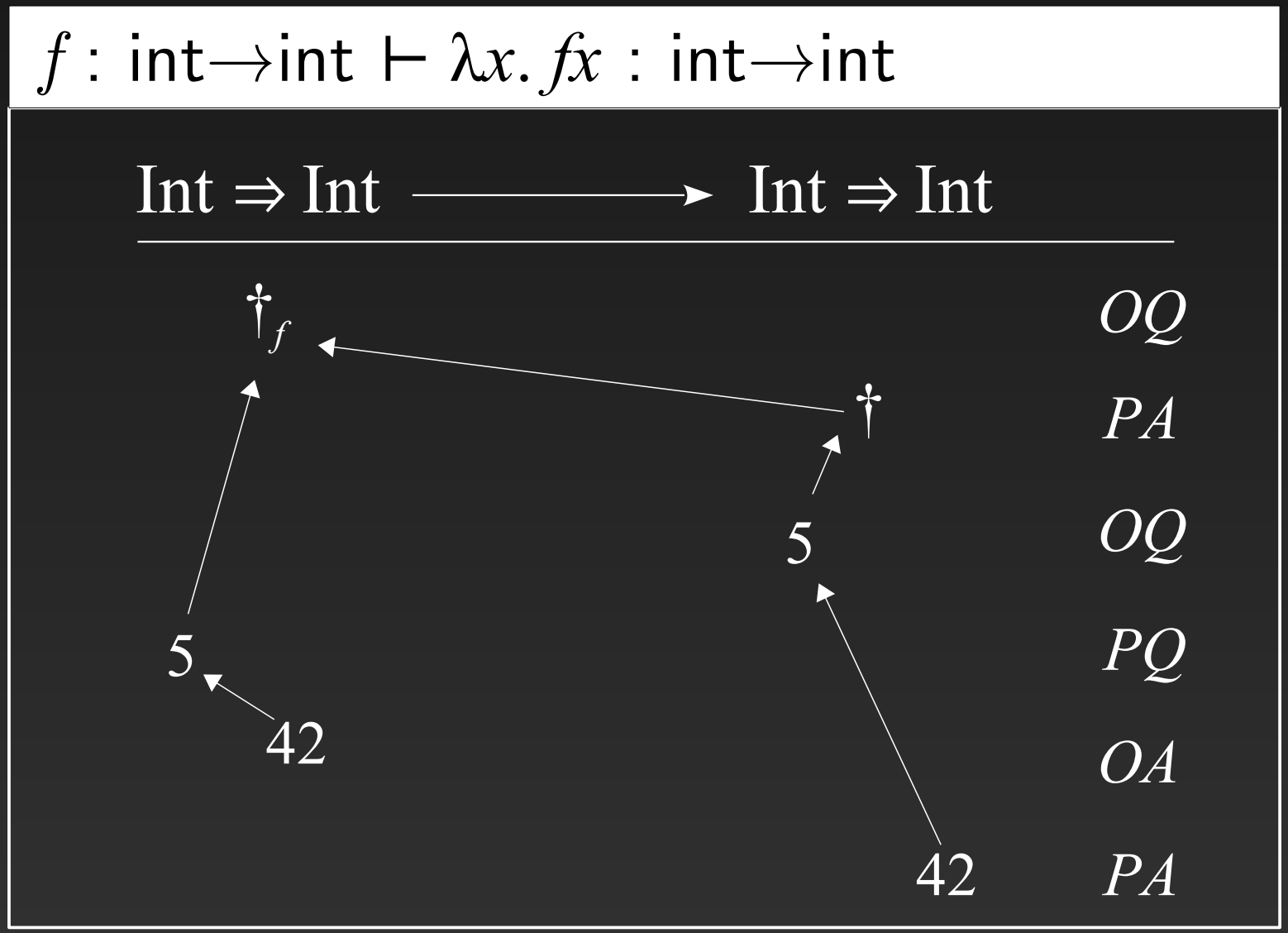
$$\vartheta, \vartheta' ::= \mathsf{unit} \mid \mathsf{int} \mid \alpha \mid \vartheta \to \vartheta' \mid \forall \alpha.\, \vartheta$$

$$\frac{\Gamma \vdash M : \vartheta}{\Gamma \vdash \Lambda \alpha.\, M : \forall \alpha.\, \vartheta} \qquad \frac{\Gamma \vdash M : \forall \alpha.\, \vartheta}{\Gamma \vdash M\vartheta' : \vartheta[\vartheta'/\alpha]}$$

$$v ::= () \mid i \mid \mathsf{if} \mid \Lambda \alpha.\, M \mid \lambda x.M$$

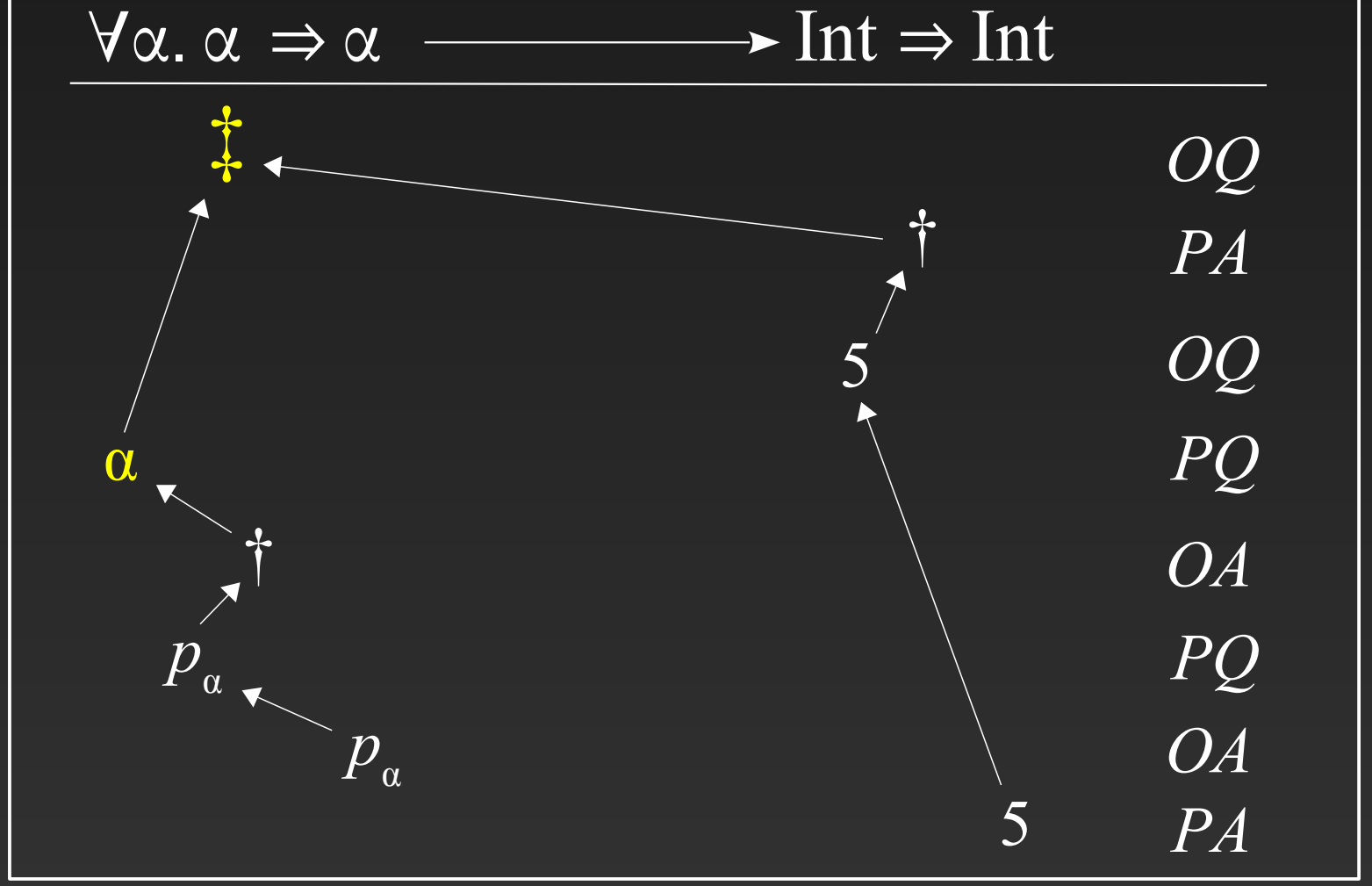
Parametricity using names

$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, fx : \mathsf{int} \to \mathsf{int}$



Parametricity using names

$f : \forall\alpha.\alpha \to \alpha \vdash \lambda x.\, f(\mathsf{int})(x) : \mathsf{int} \to \mathsf{int}$

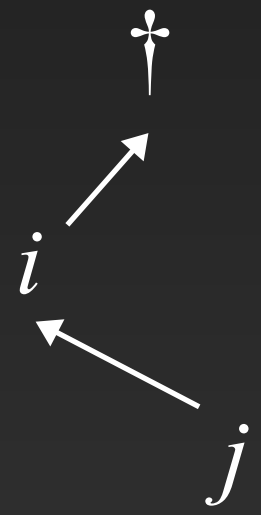


Model for polymorphism

We use names to abstract away types and values

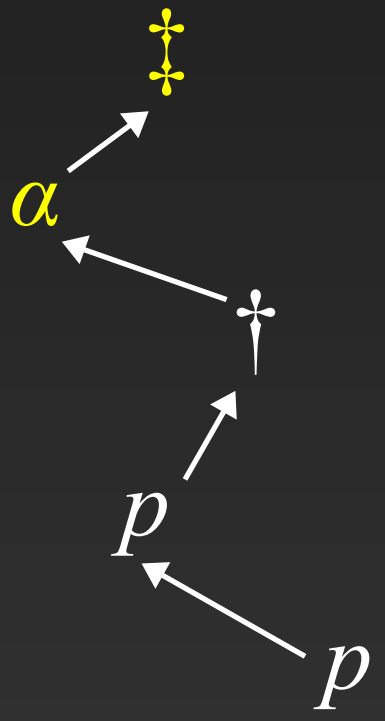
- impose uniform polymorphic behaviour
- not as clean if we also have references (*type disclosure*)

$\mathrm{Int} \Rightarrow \mathrm{Int} \;=$



(all $i,j \in \mathbb{Z}$)

$\forall \alpha.\ \alpha \Rightarrow \alpha \;=$

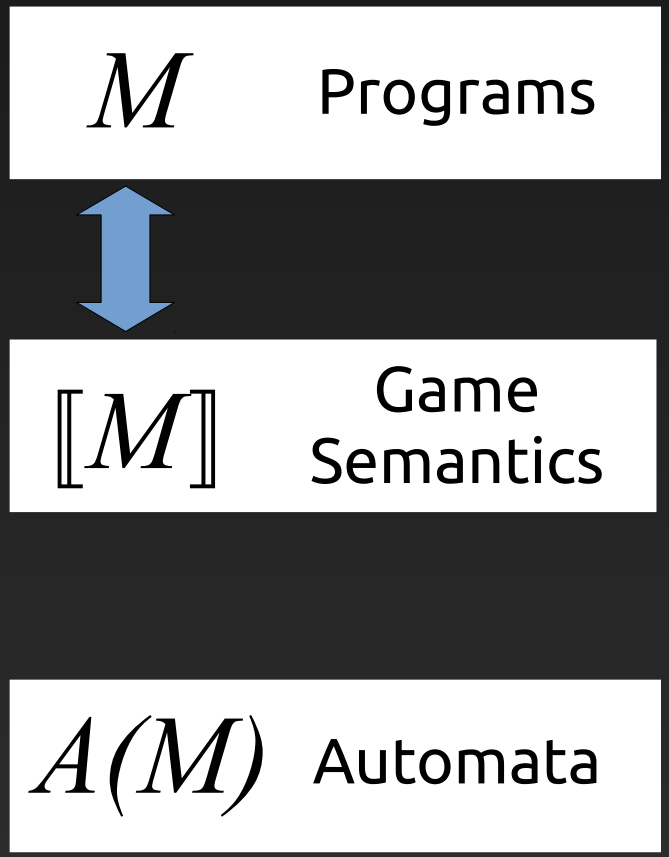


Names for:

- type variables $\alpha \in \mathcal{N}_{\mathrm{TVar}}$
- polymorphic values $p \in \mathcal{N}_{\alpha}$

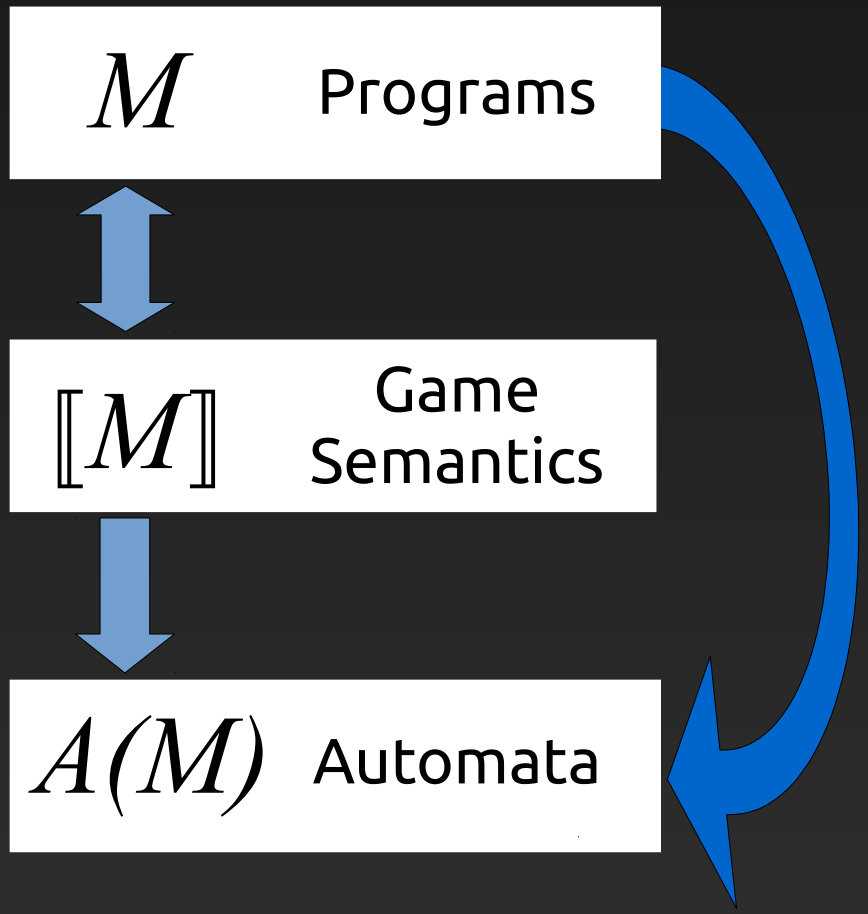
[Laird ICALP'10; Jaber & T. LICS'16]

Games and equivalence algorithmically



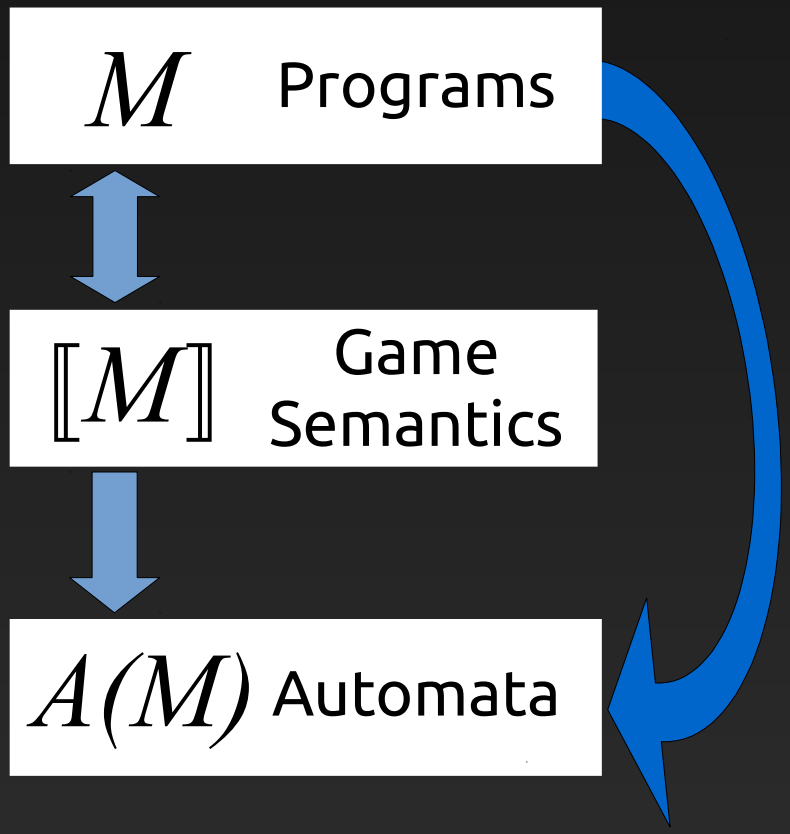
$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$

Games and equivalence algorithmically



$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!] \Leftrightarrow A \sim A'$$

Games and equivalence algorithmically



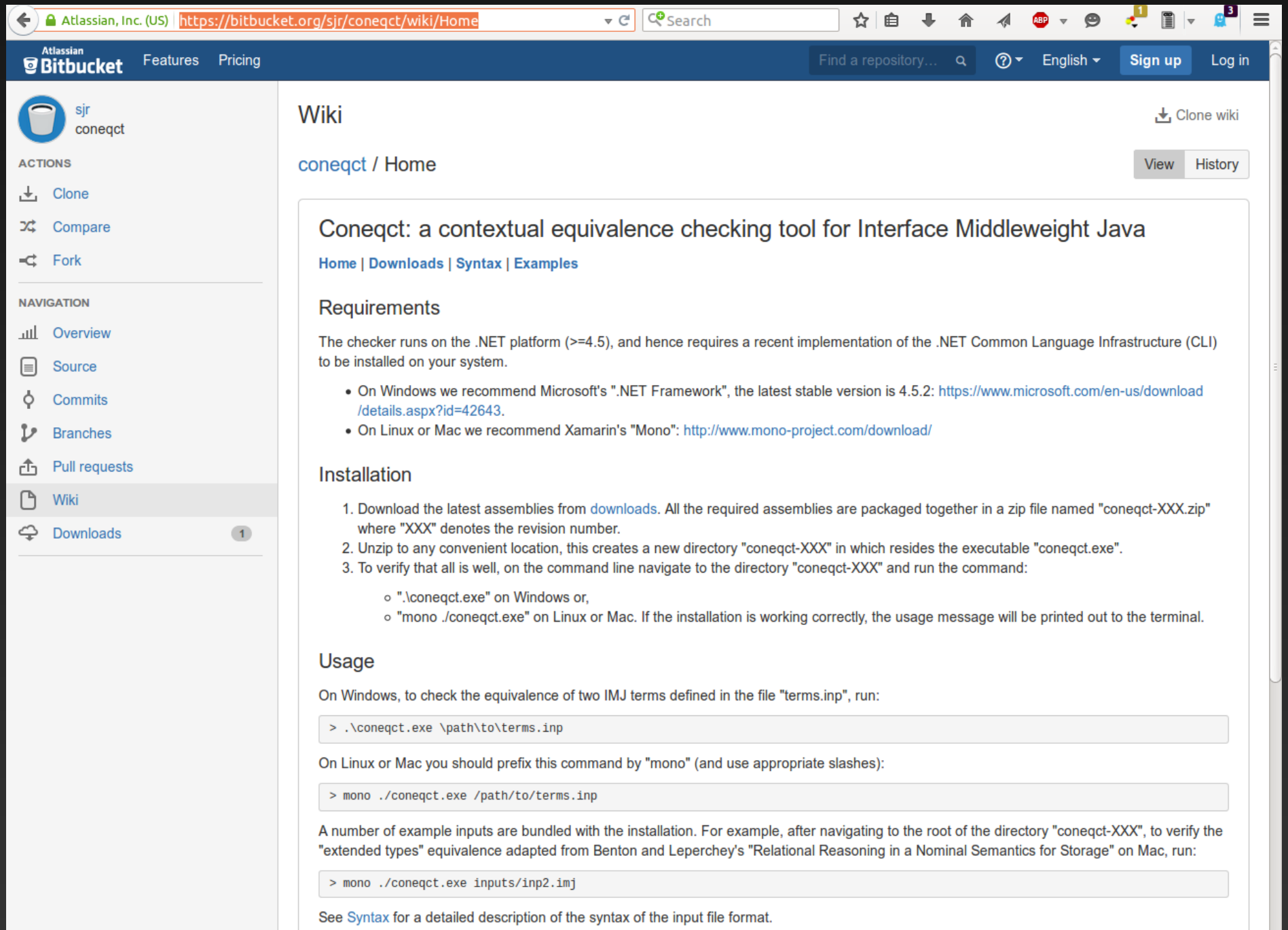
Transformation done automatically
→ decision procedure for program equivalence

- Restrict to the finitary fragment (bounded datatypes)
- Classification based on types (at each type, either the problem is undecidable or we get a procedure)
- employ automata over infinite alphabets

$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!] \Leftrightarrow A \sim A' \Leftrightarrow A \otimes A' = 0$$

[Murawski & T. ICALP'12; Murawski, Ramsay & T. ATVA'15]

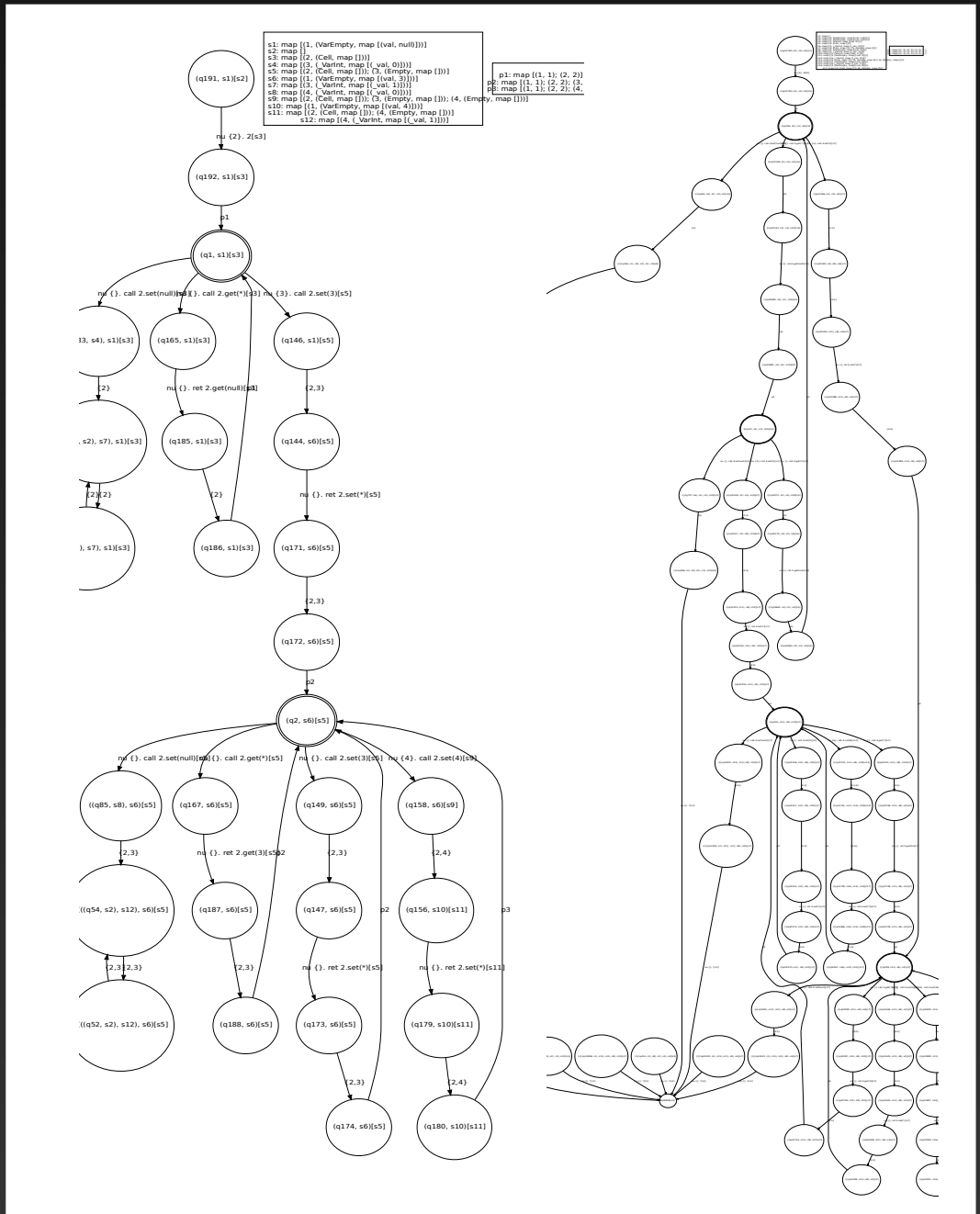
Coneqct


Atlassian, Inc. (US) https://bitbucket.org/sjr/coneqct/wiki/Home
Search
Atlassian Bitbucket
Features
Pricing
Find a repository...
English
Sign up
Log in
sjr
coneqct
ACTIONS
Clone
Compare
Fork
NAVIGATION
Overview
Source
Commits
Branches
Pull requests
Wiki
Downloads 1
Wiki
Clone wiki
coneqct / Home
View
History
Coneqct: a contextual equivalence checking tool for Interface Middleweight Java
Home | Downloads | Syntax | Examples
Requirements
The checker runs on the .NET platform (>=4.5), and hence requires a recent implementation of the .NET Common Language Infrastructure (CLI) to be installed on your system.
On Windows we recommend Microsoft's ".NET Framework", the latest stable version is 4.5.2: https://www.microsoft.com/en-us/download/details.aspx?id=42643.
On Linux or Mac we recommend Xamarin's "Mono": http://www.mono-project.com/download/
Installation
1. Download the latest assemblies from downloads. All the required assemblies are packaged together in a zip file named "coneqct-XXX.zip" where "XXX" denotes the revision number.
2. Unzip to any convenient location, this creates a new directory "coneqct-XXX" in which resides the executable "coneqct.exe".
3. To verify that all is well, on the command line navigate to the directory "coneqct-XXX" and run the command:
   - ".\coneqct.exe" on Windows or,
   - "mono ./coneqct.exe" on Linux or Mac. If the installation is working correctly, the usage message will be printed out to the terminal.
Usage
On Windows, to check the equivalence of two IMJ terms defined in the file "terms.inp", run:
> .\coneqct.exe \path\to\terms.inp
On Linux or Mac you should prefix this command by "mono" (and use appropriate slashes):
> mono ./coneqct.exe /path/to/terms.inp
A number of example inputs are bundled with the installation. For example, after navigating to the root of the directory "coneqct-XXX", to verify the "extended types" equivalence adapted from Benton and Leperchey's "Relational Reasoning in a Nominal Semantics for Storage" on Mac, run:
> mono ./coneqct.exe inputs/inp2.imj
See Syntax for a detailed description of the syntax of the input file format.

Coneqct

$$
\begin{aligned}
M_1 \equiv\ & \text{let } v = \text{new } \{\ \_\colon \mathsf{Var}_{\mathsf{Empty}}\} \text{ in} \\
& \text{new } \{\ \_\colon \text{Cell}; \\
& \quad \mathit{get}\ :\ \lambda\_.\ v.\mathit{val}, \\
& \quad \mathit{set}\ :\ \lambda y.\ \text{if } y{=}\text{null then div else } v.\mathit{val}{:=}y\ \}
\end{aligned}
$$

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

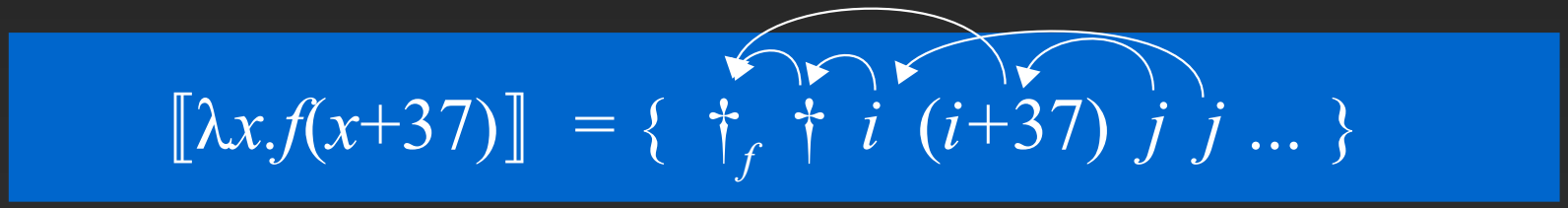


Games and traces

Games can be given a simpler, operational presentation:

- pointer structure → named functions
- definition: denotational/compositional
  → operational/executable

$$f : \mathsf{int}\to\mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int}\to\mathsf{int}$$



$$f\ g\ \mathrm{call}\, g(i)\ \ \mathrm{call}\, f(i{+}37)\ \ \mathrm{ret}\, f(j)\ \ \mathrm{ret}\, g(j)\ \ldots$$

Operational games ~ trace semantics

Full Abstraction Factory

Alan Jeffrey | Julian Rathke

Introduction

Papers

Papers from the Full Abstraction Factory

1. **Full Abstraction for Polymorphic Pi-Calculus.** A. S. A. Jeffrey and J. Rathke. In *Theoretical Computer Science*. 2007. To appear. Available on-line. Extended abstract in *Proc. Foundations of Software Science and Computation Structures*. Springer-Verlag. 2005. pp. 266-281. Available on-line.
2. **A Fully Abstract May Testing Semantics for Concurrent Objects.** A. S. A. Jeffrey and J. Rathke. In *Theoretical Computer Science*. vol. 338. 2005. pp. 17-63. Available on-line.
3. **Contextual Equivalence for Higher-Order Pi-Calculus Revisited.** A. S. A. Jeffrey and J. Rathke. In *Logical Methods in Computer Science*. 1 (1:4). 2005. pp. 1-22. Available on-line. Extended abstract in *Proc. Mathematical Foundations of Programming Semantics*. Elsevier. 2003. Available on-line.
4. **Java Jr.: Fully Abstract Trace Semantics for a Core Java Language.** A. S. A. Jeffrey and J. Rathke. In *Proc. European Symposium on Programming*. Springer-Verlag. 2005. pp. 423-438. Available on-line.
5. **A Theory of Bisimulation for a Fragment of Concurrent ML with Local Names.** A. S. A. Jeffrey and J. Rathke. In *Theoretical Computer Science*. vol. 323. 2004. pp. 1-48. Available on-line. Extended abstract in *Proc. IEEE Logic in Computer Science*. IEEE Press. 2000. pp. 311-321. Available on-line.
6. **Towards a Theory of Bisimulation for Local Names.** A. S. A. Jeffrey and J. Rathke. In *Proc. IEEE Logic in Computer Science*. IEEE Press. 1999. pp. 56--66. Available on-line.
7. **A Fully Abstract Semantics for a Nondeterministic Functional Language with Monadic Types.** A. S. A. Jeffrey. In *Theoretical Computer Science*. vol. 228. 1999. pp. 105-150. Available on-line. Extended abstract in *Proc. Mathematical Foundations of Programming Semantics*. Elsevier. 1995. Available on-line.
8. **Semantics for Core Concurrent ML Using Computation Types.** A. S. A. Jeffrey. In *Proc. Higher Order Operational Techniques in Semantics*. Cambridge University Press. 1997. pp. 55-89. Available on-line.
9. **A Fully Abstract Semantics for a Concurrent Functional Language with Monadic Types.** A. S. A. Jeffrey. In *Proc. IEEE Logic in Computer Science*. IEEE Press. 1995. pp. 255--264. Available on-line.
10. **A Fully Abstract Semantics for Concurrent Graph Reduction.** A. S. A. Jeffrey. In *Proc. IEEE Logic in Computer Science*. IEEE Press. 1994. pp. 82-91. Available on-line.

This material is partly based upon work supported by the National Science Foundation under Grant No. 0430175, by the Engineering and Physical Sciences Research Council grants *Foundations for the Integration of Concurrent, Distributed and Functional Computation* and *Linear Type Systems for Concurrent Systems*, and by the Nuffield Foundation grant *A Semantic Study of Behavioural Properties for Systems of Distributed Objects*.

[Laird ICALP'07; Ghica & T. MFPS'12; Levy & Staton LICS'14; Jaber FOSSACS'15]

Operational games

Defined operational, via an open transition system

- Configurations

$\langle\ M, S, C, \mathcal{U}\ \rangle$ (for P) $\qquad$ $\langle\ S, C, \mathcal{U}\ \rangle$ (for O)

- Transition rules

[Int] if $M, S \to M', S'$ then $\langle M, S, C, \mathcal{U}\rangle \longrightarrow \langle M', S', C, \mathcal{U}\rangle$

[OQ] $\langle S, C, \mathcal{U}\rangle \xrightarrow{\text{call } f(v), S'} \langle \mathcal{U}(f),\ S[S'], f::C, \mathcal{U}'\rangle$

[PA] $\langle v, S, f::C, \mathcal{U}\rangle \xrightarrow{\text{ret } f(v'), S'} \langle\ S, C, \mathcal{U}'\rangle$

[PQ] $\langle E[f\,v], S, C, \mathcal{U}\rangle \xrightarrow{\text{call } f(v'), S'} \langle\ S, (f,E)::C, \mathcal{U}'\rangle$

[OA] $\langle S, (f,E)::C, \mathcal{U}\rangle \xrightarrow{\text{ret } f(v), S'} \langle E[v], S[S'], C, \mathcal{U}'\rangle$

Recap and further work

Nominal game semantics for modelling HO programs

- fragments of ML and Java
- also low-level languages (C-like)

Denotational and operational presentations

- compositional presentation
- trace-based, executable semantics

Further on:

- from language fragments to full languages
- abstractions for practical verification