Nominal game semantics

Nikos Tzevelekos, Queen Mary U. of London

jointly with:

Andrzej Murawski, Oxford

Guilhem Jaber, Lyon

Dan Ghica, Birmingham

Steven Ramsay, Bristol

Thomas Cuvillier, QMUL

First Congress of Greek Mathematicians – June 2018

what this talk is about

What is the meaning of higher-order (computer) programs?

We look into semantic models

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

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

Present an approach based on game semantics:

- programs = games between two players
- syntactic composition = composition of games
- models presented as categories of games/strategies

λ-calculus + arithmetic

$$M ::= x \mid \lambda x^{\vartheta}.M \mid M\ M$$
$$\mid i \mid \text{if } M \text{ then } M \text{ else } M \mid M + M$$

$x \in \mathsf{Var}$

$i \in \mathsf{Int}$

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

examples:

$42 : \mathsf{int}$

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

$\lambda f^{\mathsf{int}\to\mathsf{int}}.x^{\mathsf{int}}.fx+1 :$
$(\mathsf{int}\to\mathsf{int})\to\mathsf{int}\to\mathsf{int}$

Typing rules

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

$$\frac{}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{\Gamma \vdash M: \mathsf{int} \quad \Gamma \vdash N: \mathsf{int}}{\Gamma \vdash M + N: \mathsf{int}} \qquad \frac{\Gamma \vdash M: \mathsf{int} \qquad \Gamma \vdash N, N': \vartheta}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N' : \vartheta}$$

Operational semantics

$$(\lambda x.M)v \;\rightarrow\; M\{v/x\} \qquad \text{(call-by-value)}$$

e.g.

$(\lambda x^{\text{int}}.x{+}1)5 \rightarrow 5{+}1 \rightarrow 6$

$(\lambda f^{\text{int}\rightarrow\text{int}}.f(f5))(\lambda x^{\text{int}}.x{+}1) \rightarrow (\lambda x^{\text{int}}.x{+}1)((\lambda x^{\text{int}}.x{+}1)5)\ldots$

+ rules for arithmetic and ifzero-then-else

Modelling problem

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

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

such that:

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

Why?

- reasoning about programs → formal reasoning in the model

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

Denotational models

$$\Gamma \vdash M : \vartheta$$

$$[\![M]\!] : [\![\Gamma]\!] \longrightarrow [\![\vartheta]\!]$$

types → objects, terms/programs → morphisms

- denotations of atomic constructs (e.g. $[\![5]\!]$, $[\![\mathsf{if}]\!]$, …)
- composition:

$[\![N]\!] \circ [\![M]\!]$ (or: $[\![M]\!];[\![N]\!]$)

$$[\![MN]\!] : [\![\vartheta_1]\!] \xrightarrow{[\![M]\!]} [\![\vartheta_2]\!] \xrightarrow{[\![N]\!]} [\![\vartheta_3]\!]$$

The quest for full abstraction

1977 [Milner, Plotkin]:

- Formulation of the problem
- Functions cannot capture sequentiality (PCF)

TCS'77

FULLY ABSTRACT MODELS OF TYPED λ-CALCULI

Robin MILNER
Computer Science Department, Edinburgh University, Edinburgh, Scotland

Communicated by Maurice Nivat
Received October 1975
Revised June 1976

Abstract. A semantic interpretation $\mathcal{A}$ for a programming language L is fully abstract if, whenever $\mathcal{A}[\![\mathscr{C}[M]]\!] \sqsubseteq \mathcal{A}[\![\mathscr{C}[N]]\!]$ for two program phrases M, N and for all program contexts $\mathscr{C}[\ ]$, it follows that $\mathcal{A}[\![M]\!] \sqsubseteq \mathcal{A}[\![N]\!]$. A model $\mathcal{M}$ for the language is fully abstract if the natural interpretation $\mathcal{A}$ of L in $\mathcal{M}$ is fully abstract.

We show that under certain conditions there exists, for an extended typed λ-calculus, a unique fully abstract model.

1. Introduction

We are concerned with the problem of finding, for a programming language, a denotational semantic definition which is not over-generous in a certain sense. We can describe quite informally what we mean by 'over-generosity'. Suppose that L is the set of well-formed phrases of the language. Often it is the case that not every such phrase is a whole program; for example, a procedure declaration may not be one, though of course may be part of one.

TCS'77

LCF CONSIDERED AS A PROGRAMMING LANGUAGE

G.D. PLOTKIN
Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW, Scotland

Communicated by Robin Milner
Received July 1975

Abstract. The paper studies connections between denotational and operational semantics for a simple programming language based on LCF. It begins with the connection between the behaviour of a program and its denotation. It turns out that a program denotes ⊥ in any of several possible semantics iff it does not terminate. From this it follows that if two terms have the same denotation in one of these semantics, they have the same behaviour in all contexts. The converse fails for all the semantics. If, however, the language is extended to allow certain parallel facilities, behavioural equivalence does coincide with denotational equivalence in one of the semantics considered, which may therefore be called "fully abstract". Next a connection is given which actually determines the semantics up to isomorphism from the behaviour alone. Conversely, by allowing further parallel facilities, every r.e. element of the fully abstract semantics becomes definable, thus characterising the programming language, up to interdefinability, from the set of r.e. elements of the domains of the semantics.

1. Introduction

We present here a study of some connections between the operational and

The quest for full abstraction

1977 [Milner, Plotkin]:

- Formulation of the problem
- Functions cannot capture sequentiality (PCF)

1980-90's:

- Function stability [Berry, Bucciarelli, Erhard]
- Sequential algorithms [Berry, Currien]

1993 [AJM, HO/N *]: Game semantics (PCF)

- 'Functions' with operational content (*games*)

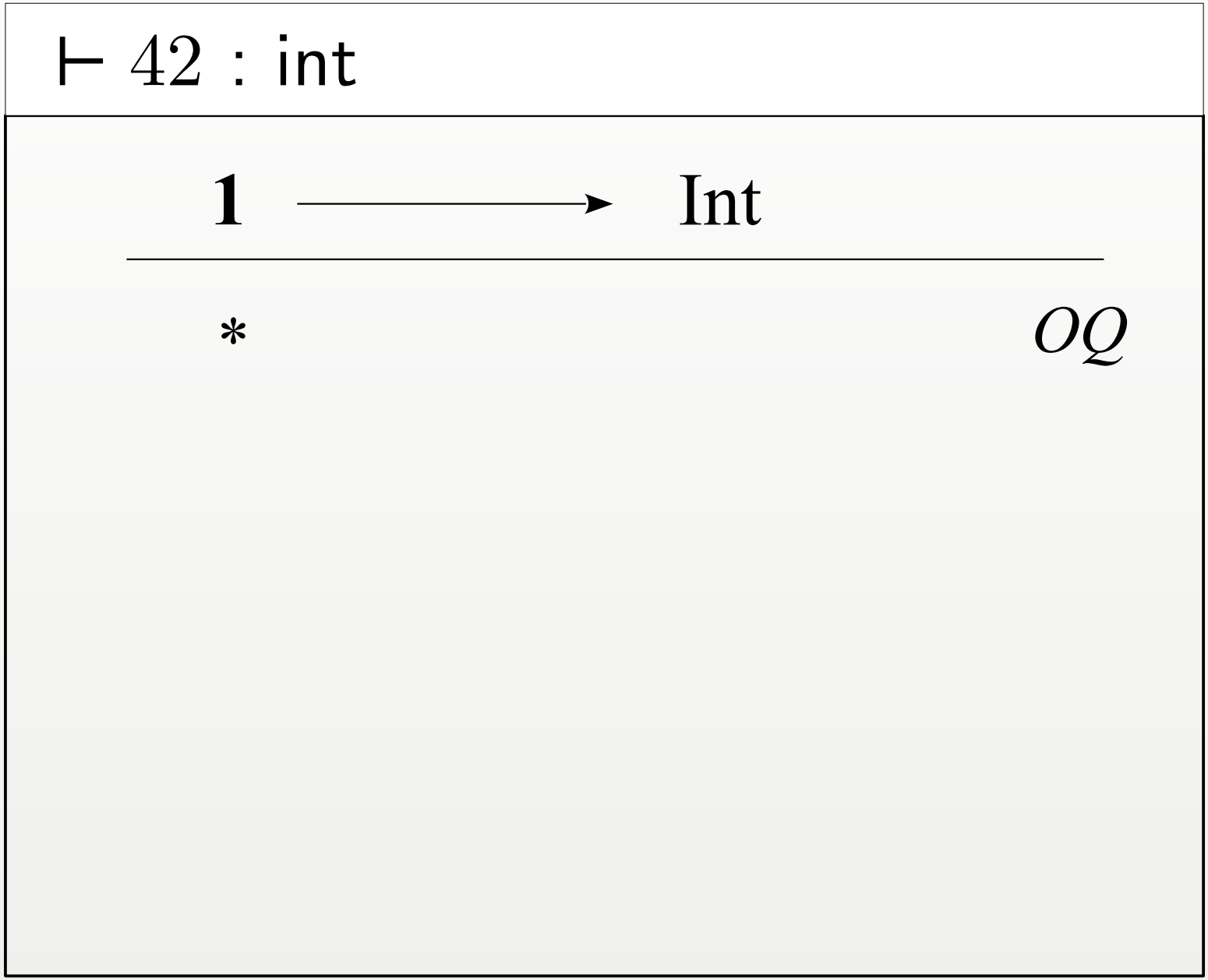
* Abramsky, Jagadeesan, Malacaria; Hyland, Ong; Nickau

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*

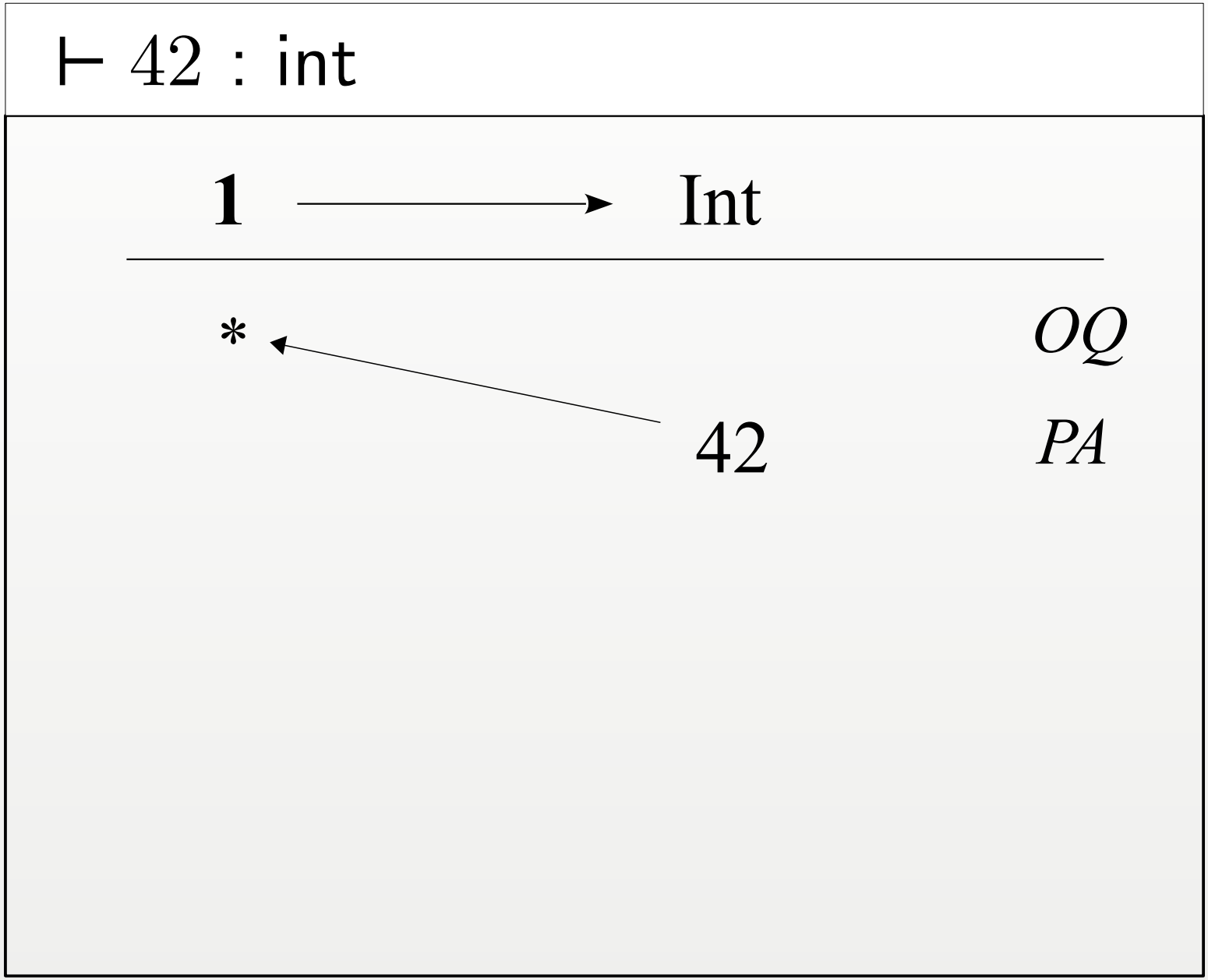
Example games

$\vdash 42 : \mathsf{int}$



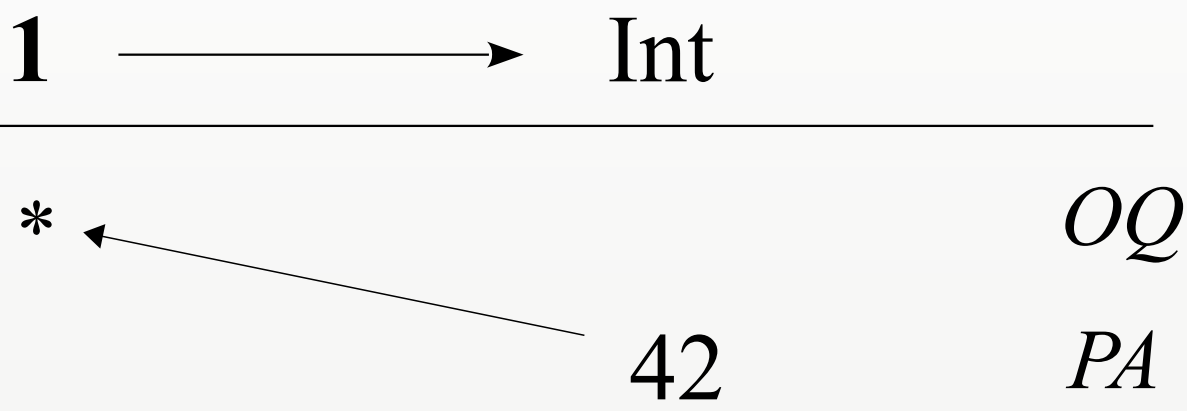
Example games

$\vdash 42 : \mathsf{int}$



Example games

$\vdash 42 : \mathsf{int}$



$[\![42]\!] = \{ \; * \;\; 42 \; \}$

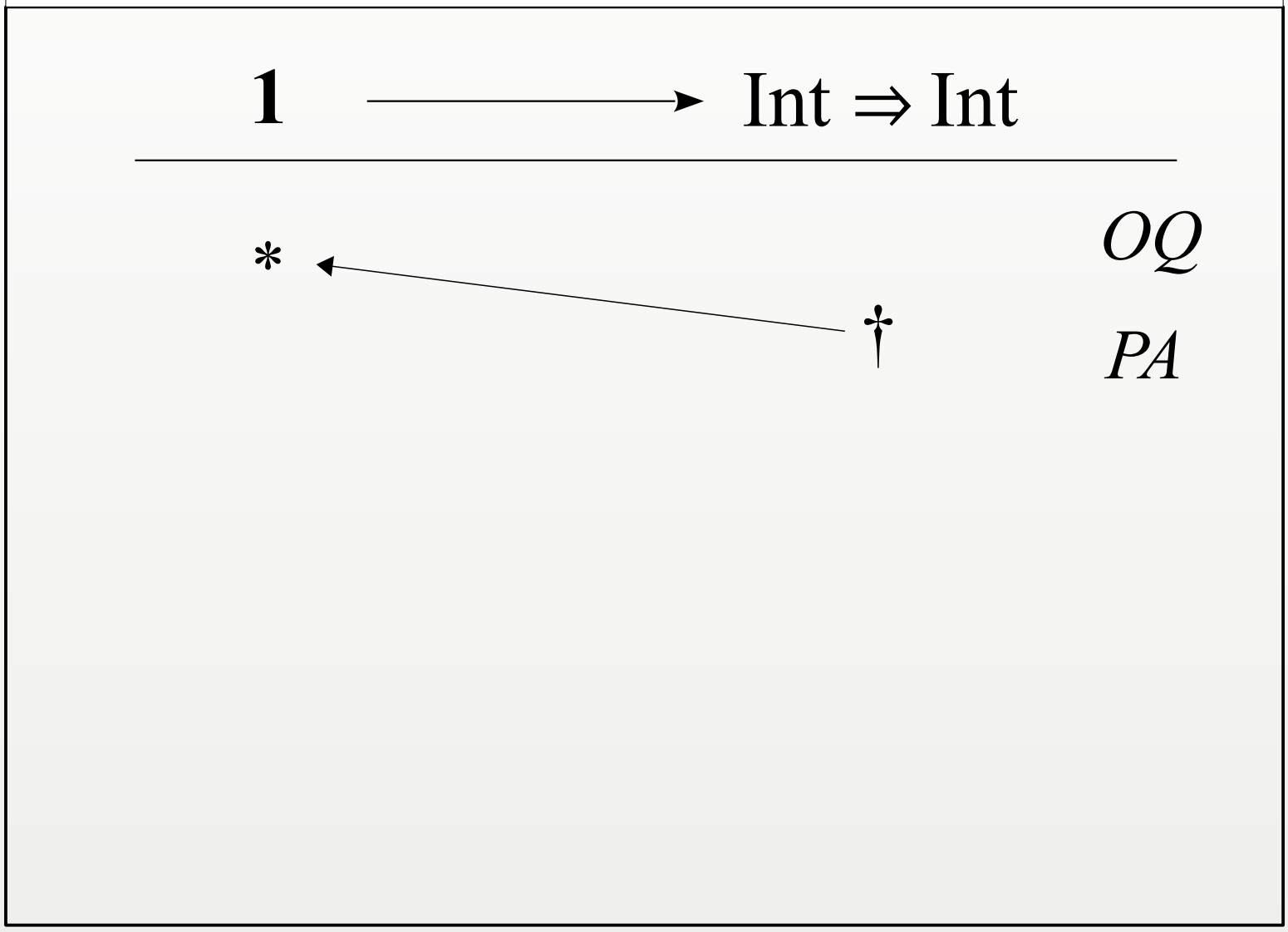
OQ *PA*

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*

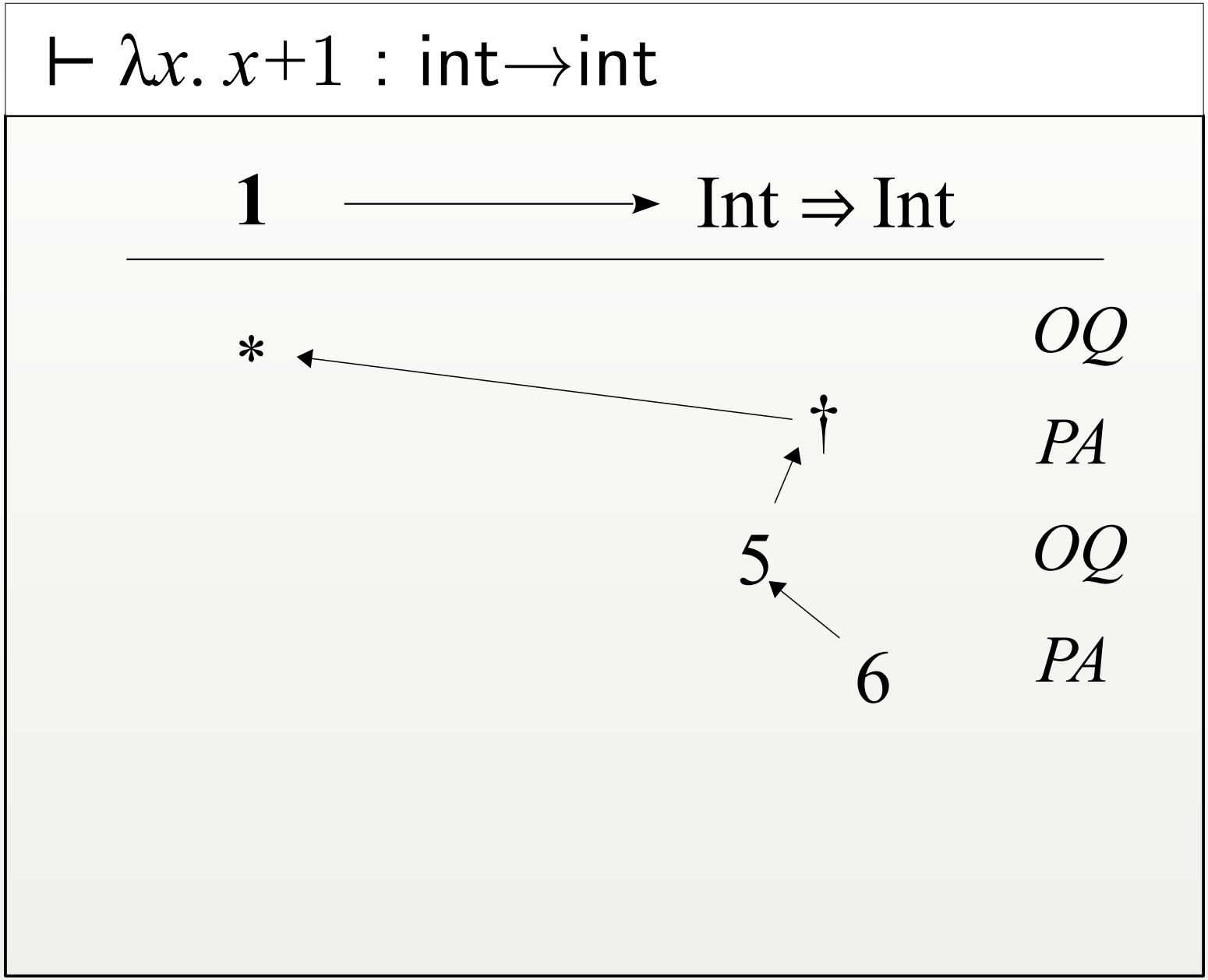
Example games

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



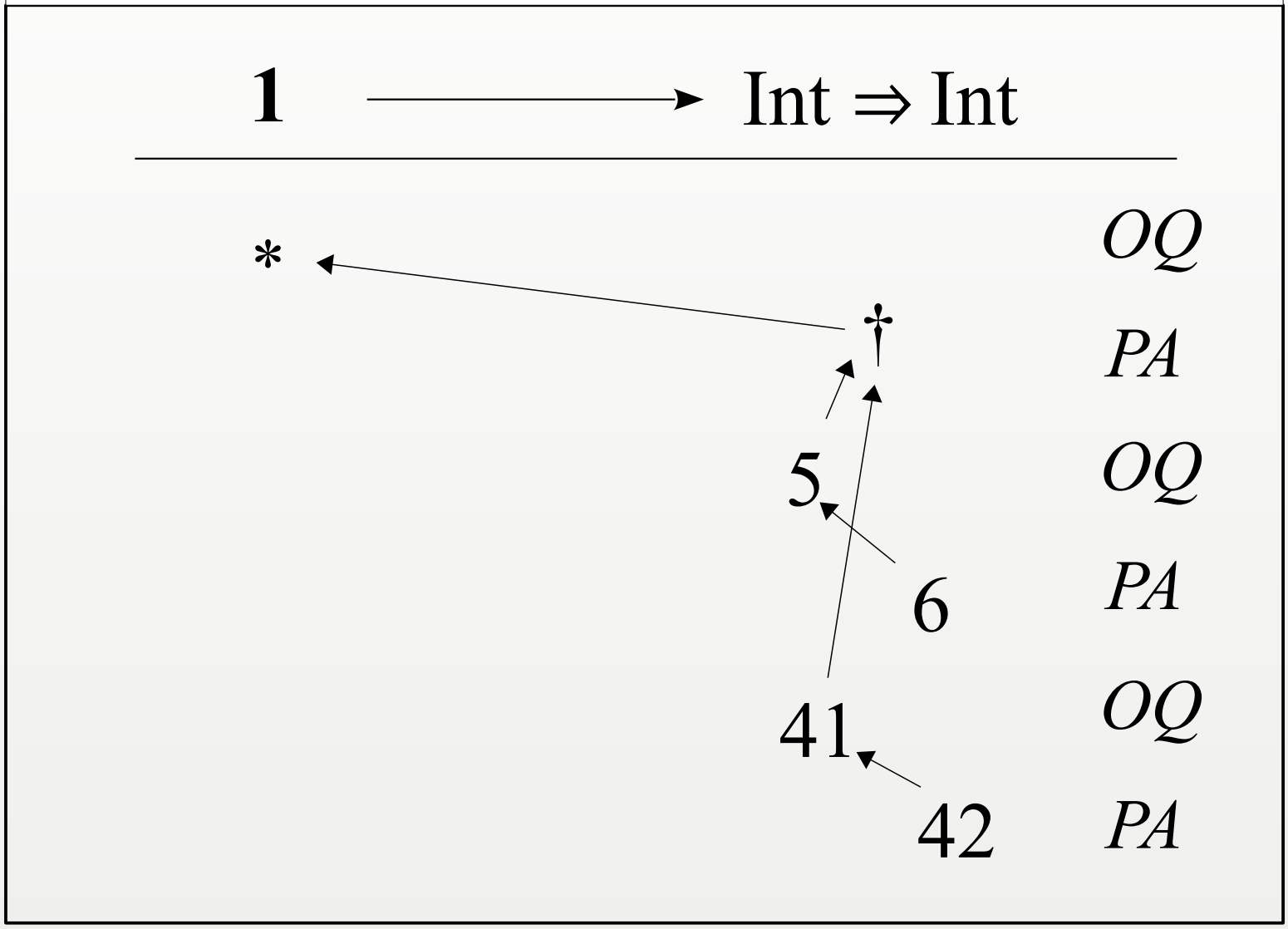
Example games

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



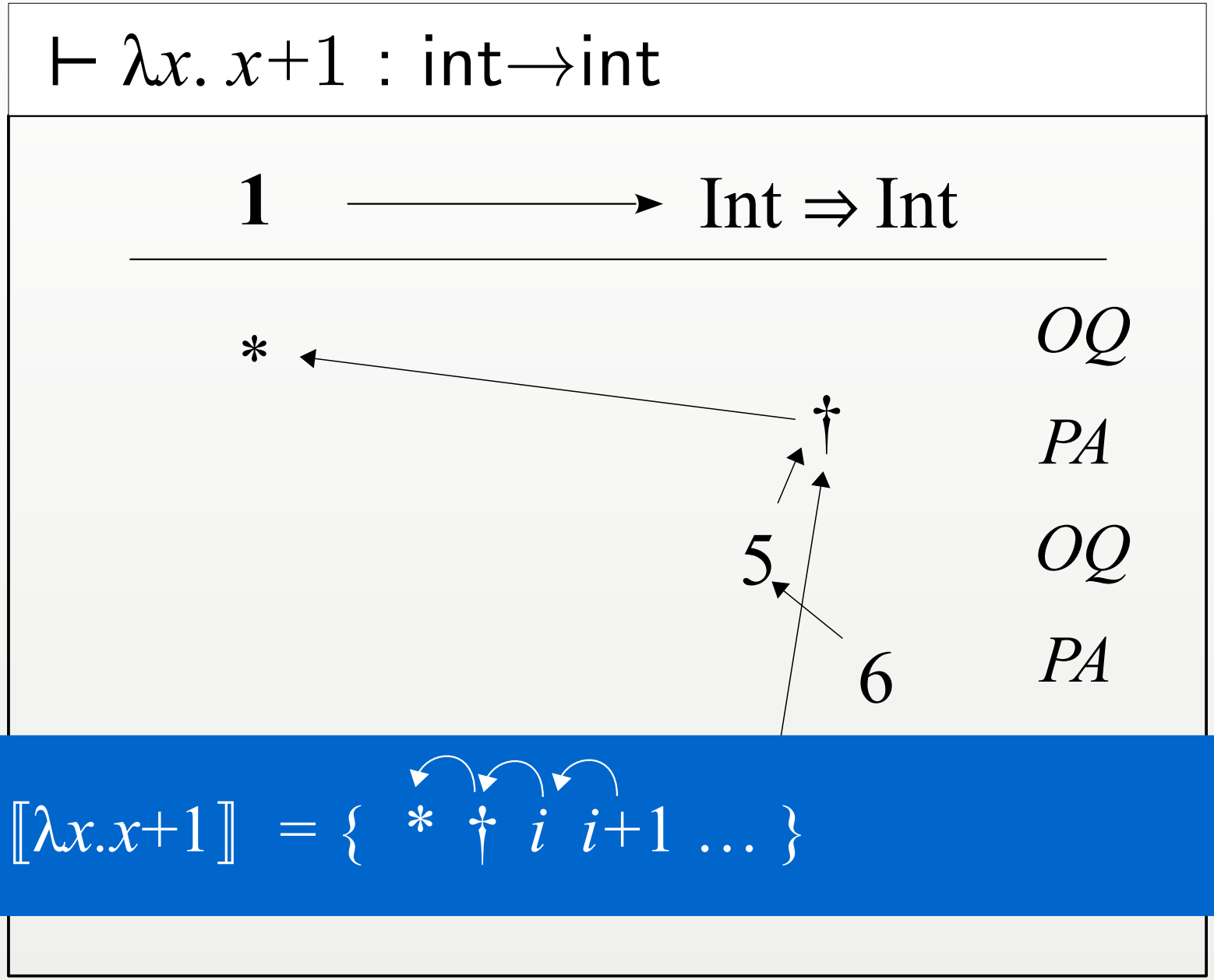
Example games

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



Example games

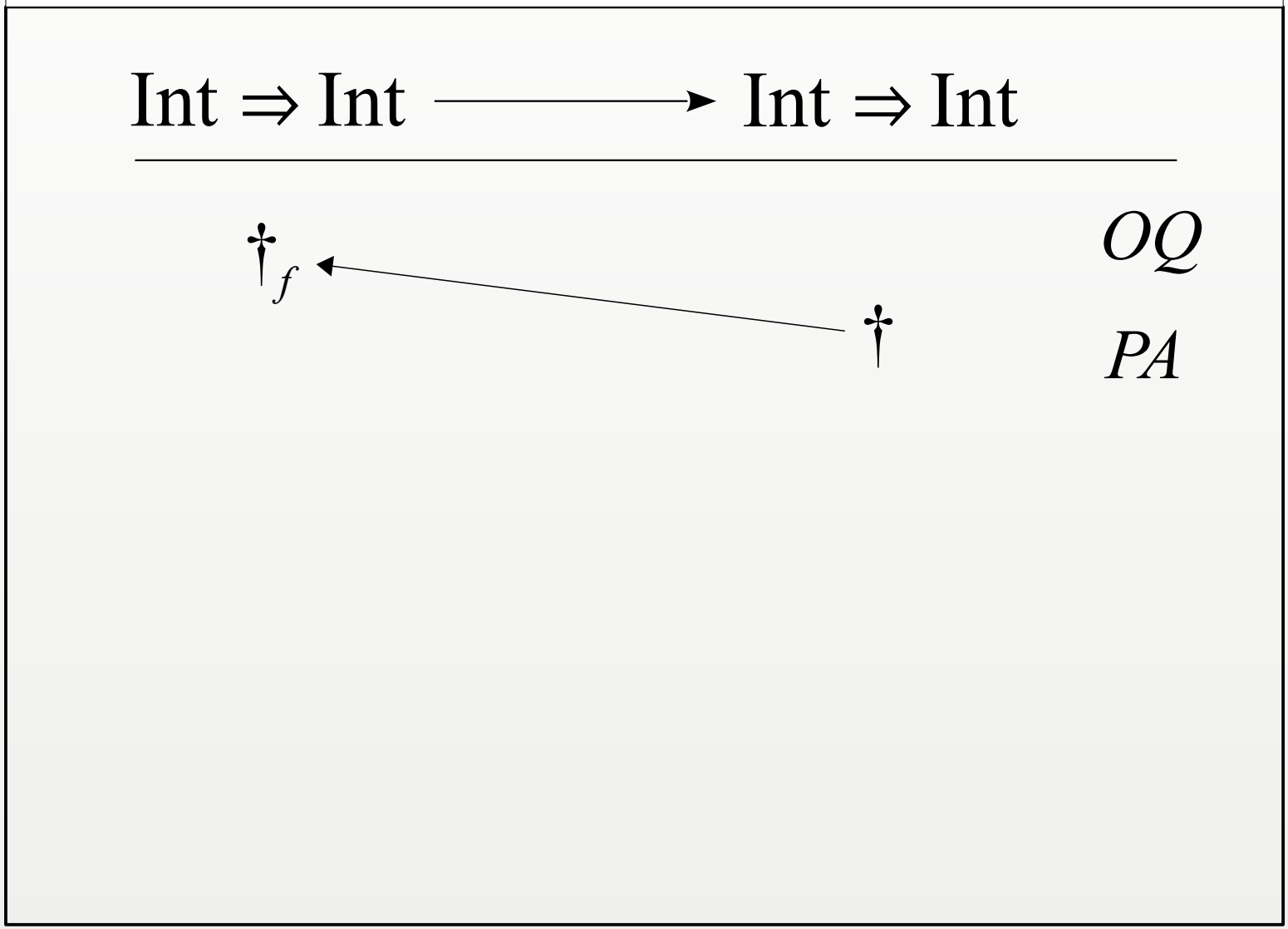
$\vdash \lambda x.\ x+1 : \mathsf{int} \to \mathsf{int}$



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

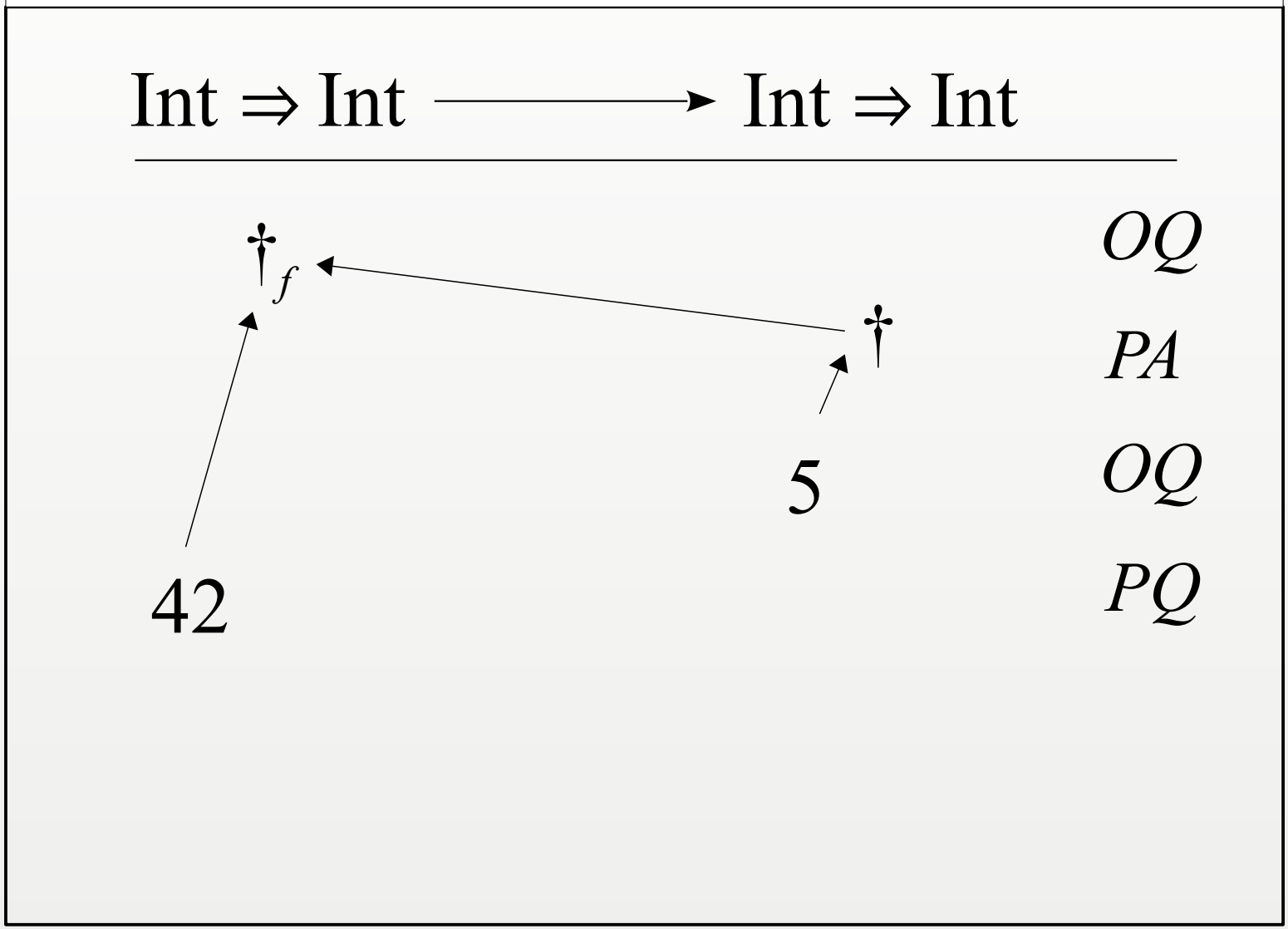
Example games

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



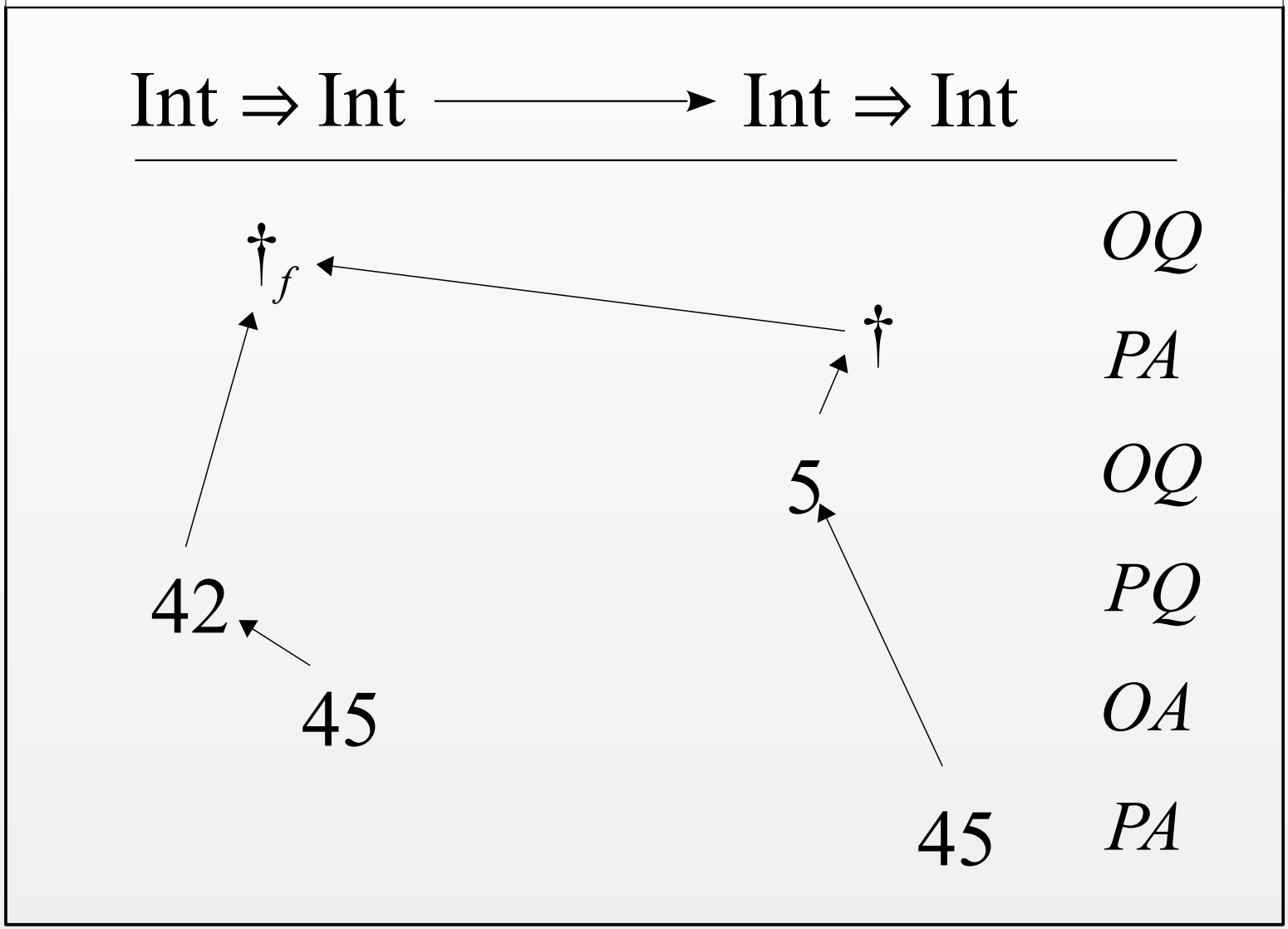
Example games

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



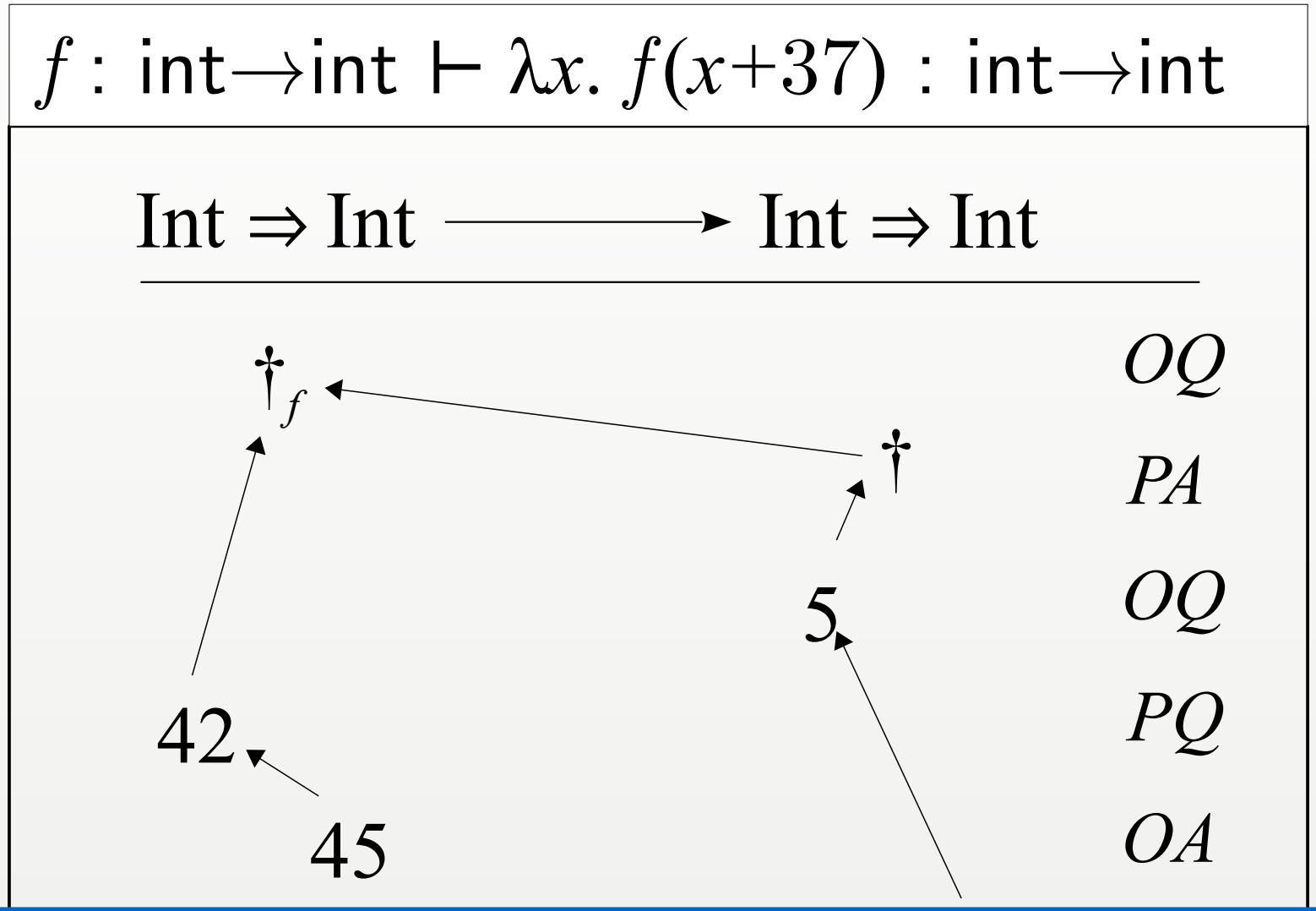
Example games

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



Example games

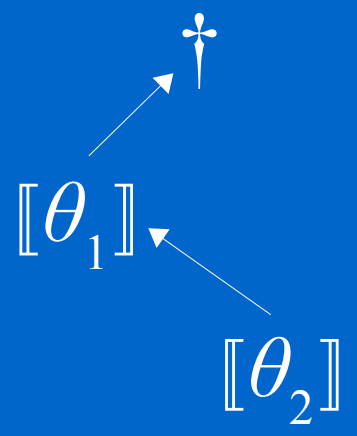
$f : \mathsf{int} \to \mathsf{int} \vdash \lambda x.\, f(x{+}37) : \mathsf{int} \to \mathsf{int}$



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

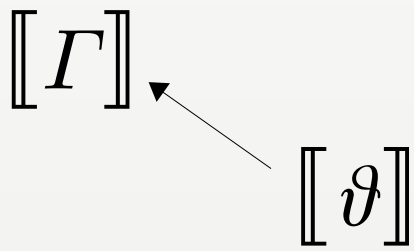
Denotations of types and programs

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

$[\![\theta_1 \to \theta_2]\!] =$ 

$\Gamma \vdash M : \vartheta$

$[\![M]\!]$ = a set of *plays* in the game:



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

$\ldots$)

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 combined via composition

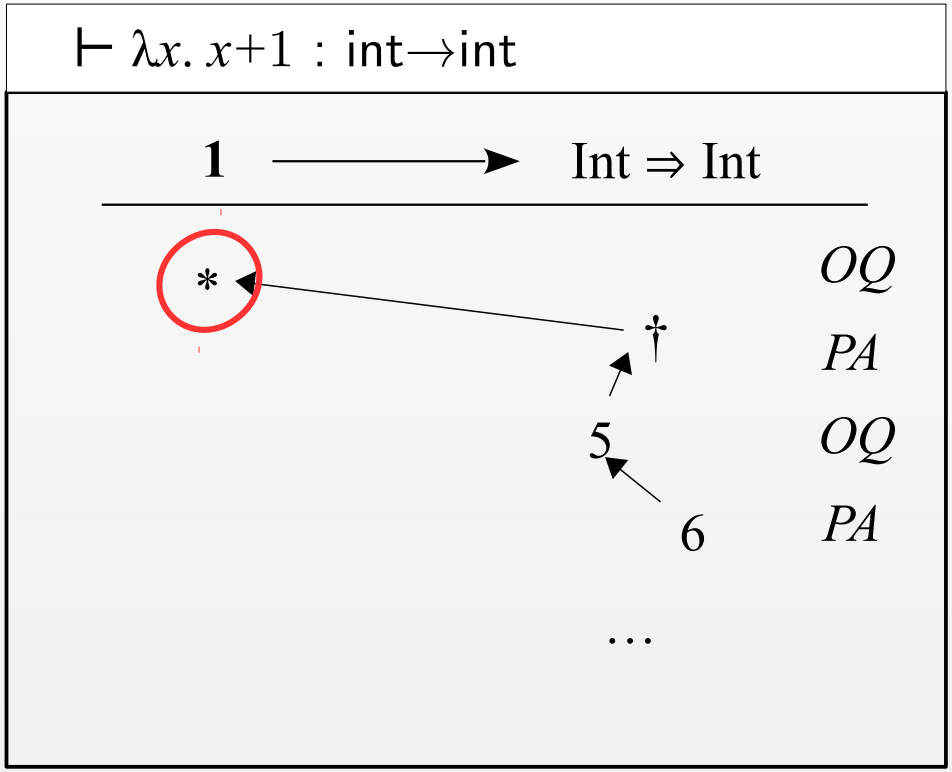
Composition

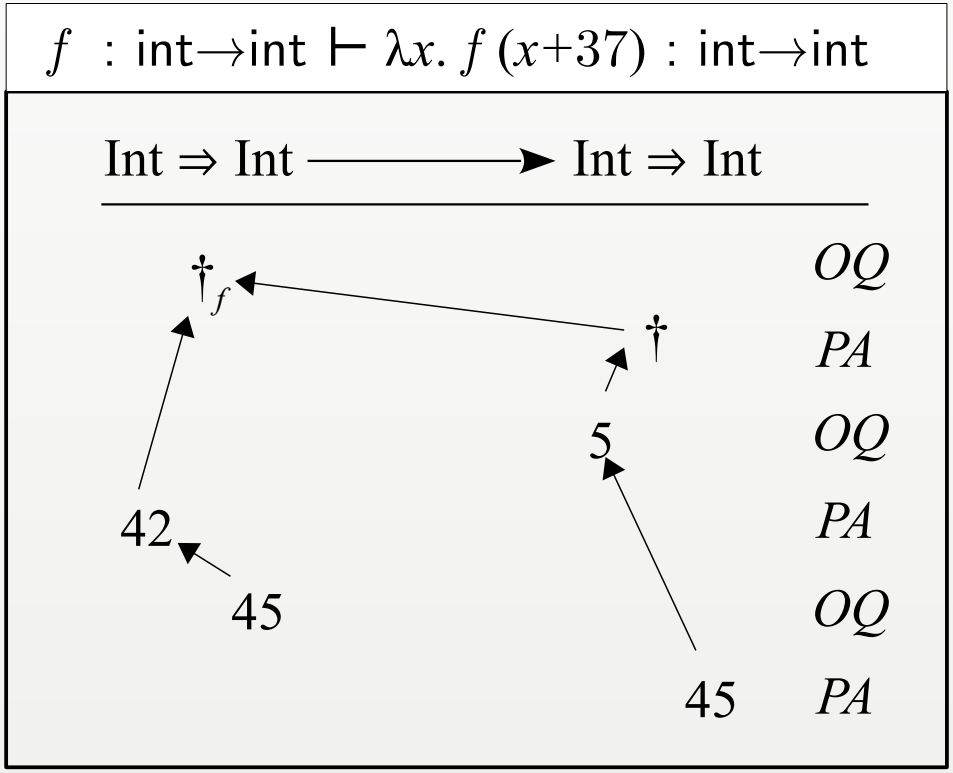
Strategies can be composed by matching O/P behaviours

Composition

Strategies can be composed by matching O/P behaviours

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

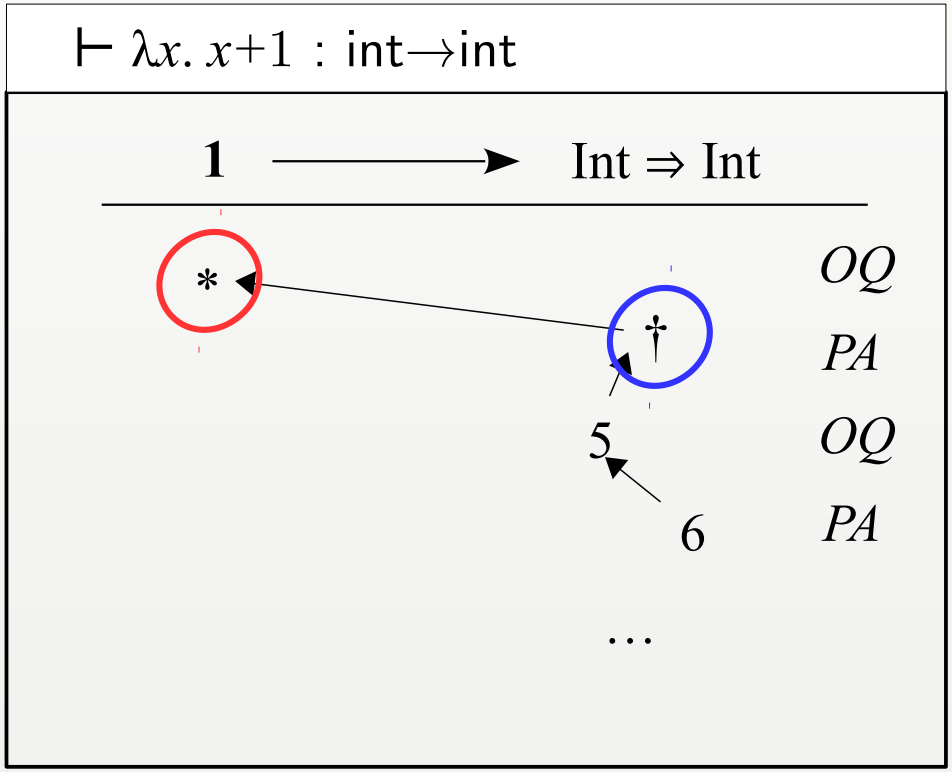


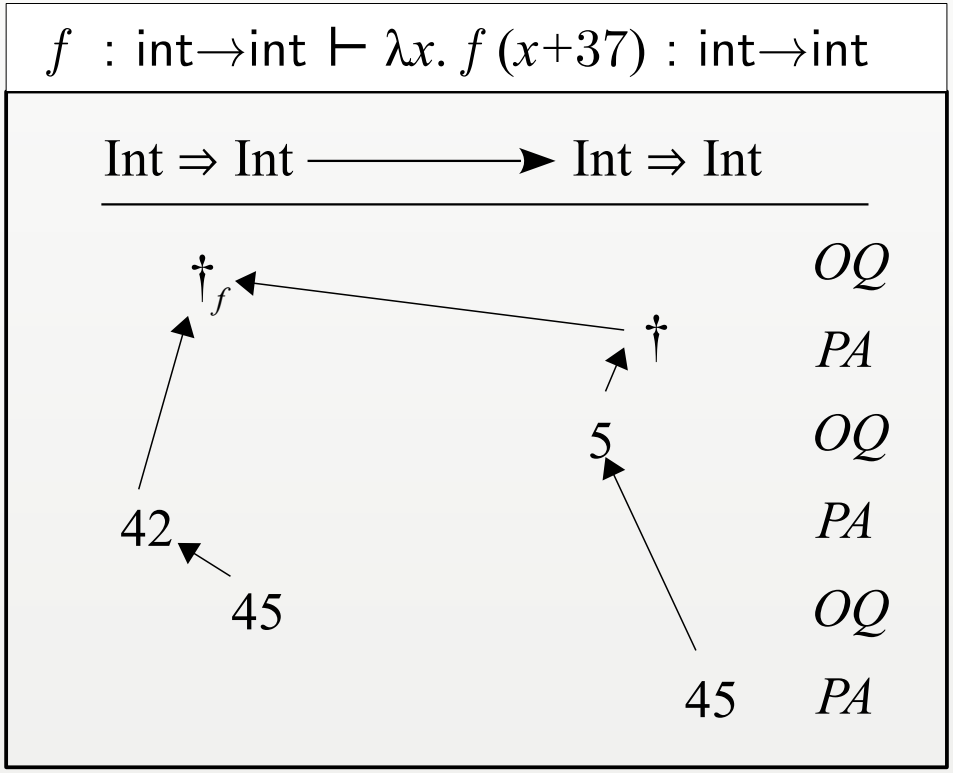


Composition

Strategies can be composed by matching O/P behaviours

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

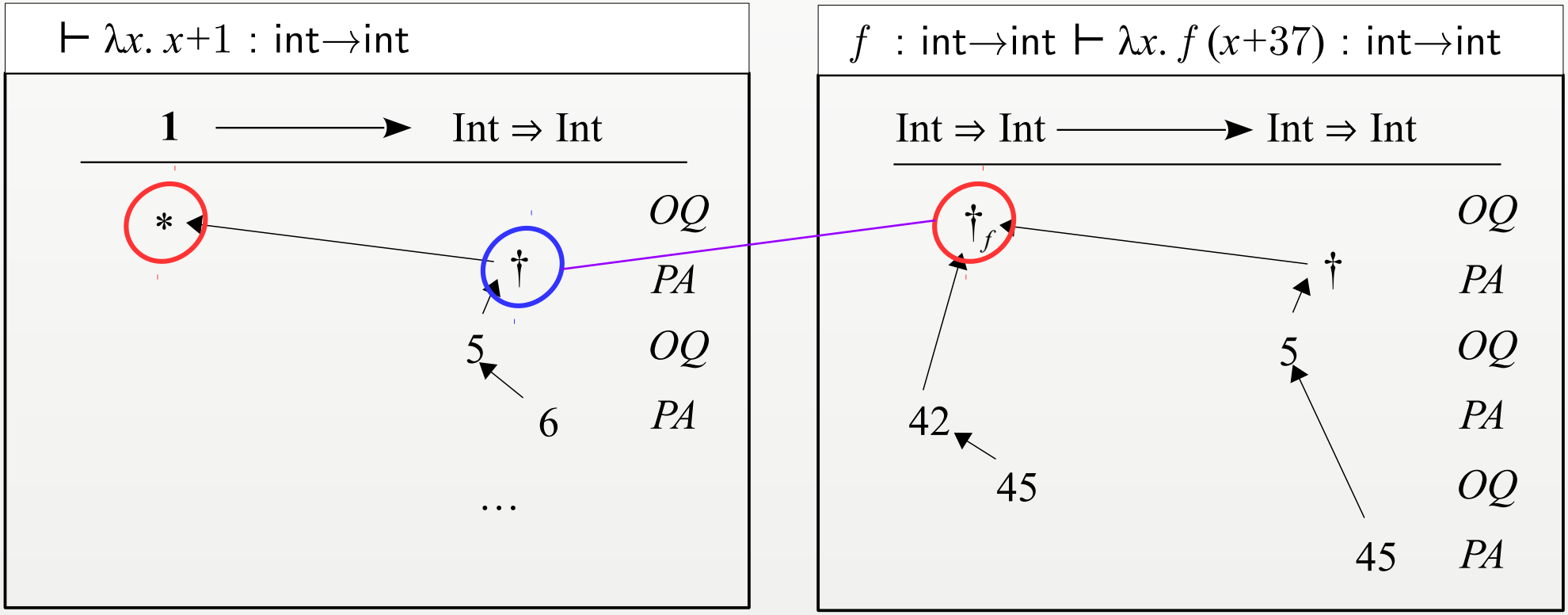




Composition

Strategies can be composed by matching O/P behaviours

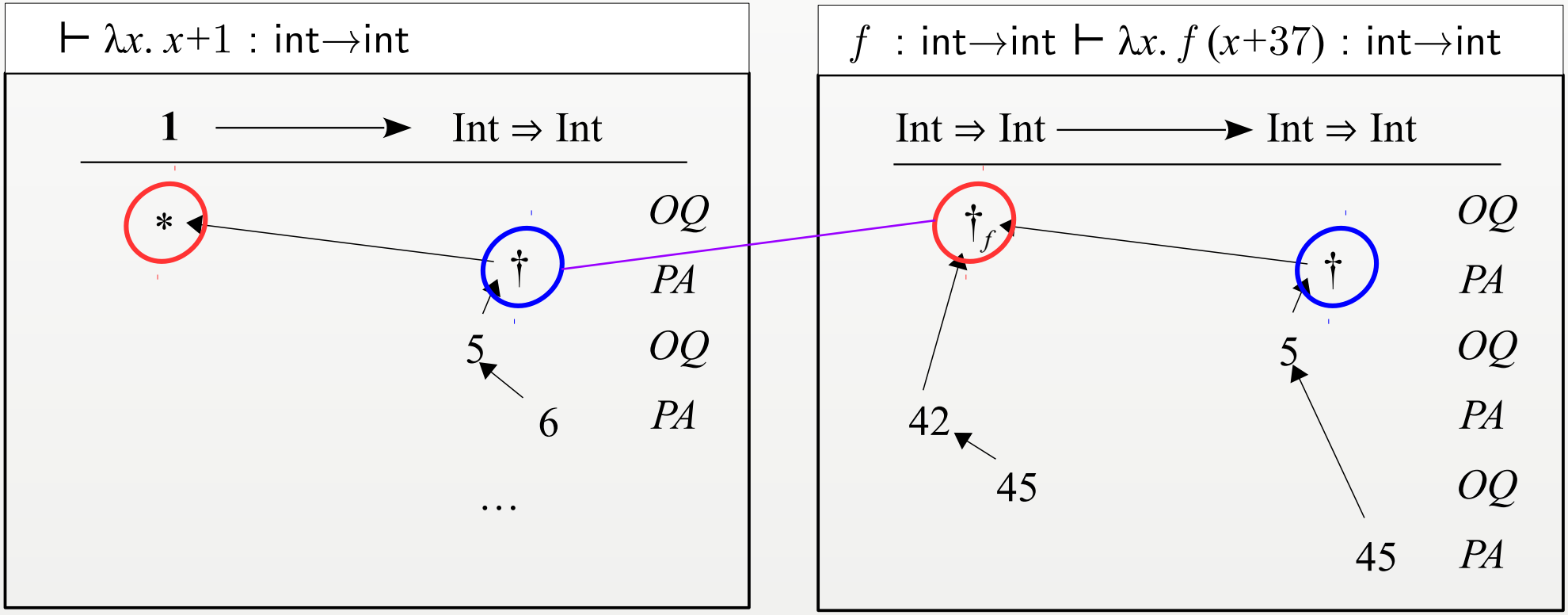
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

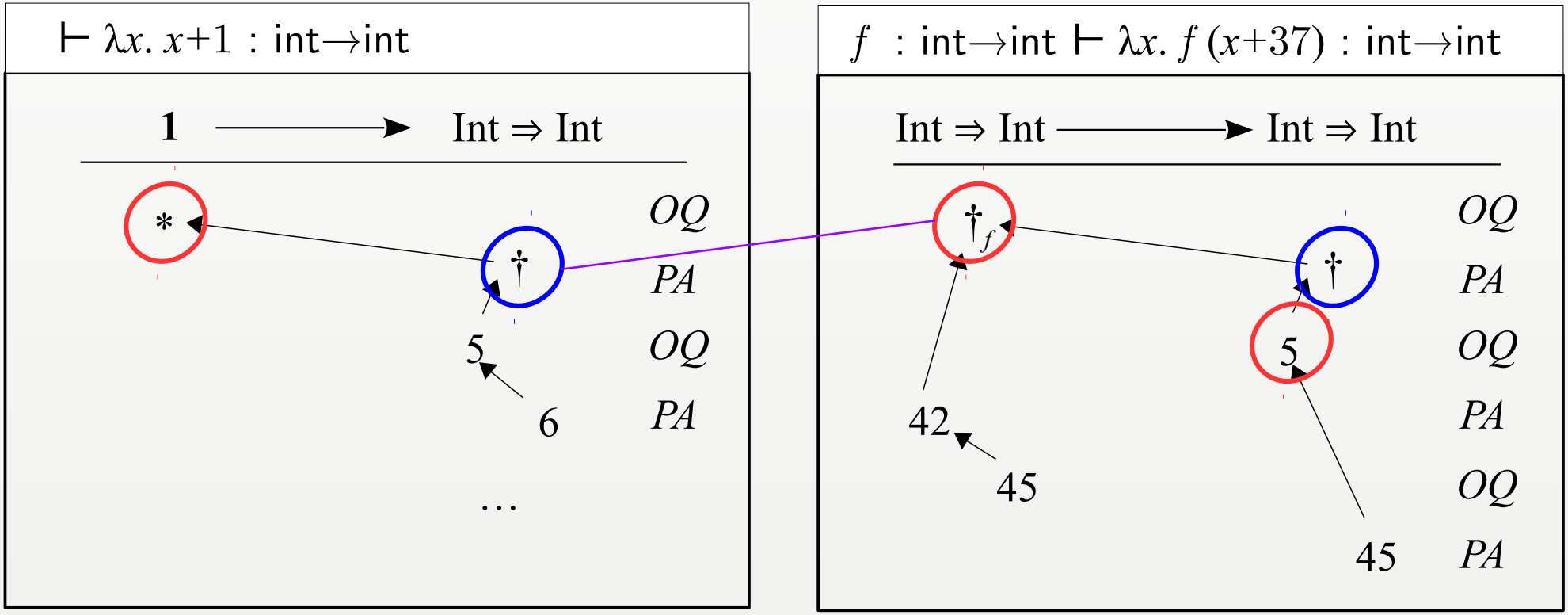
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

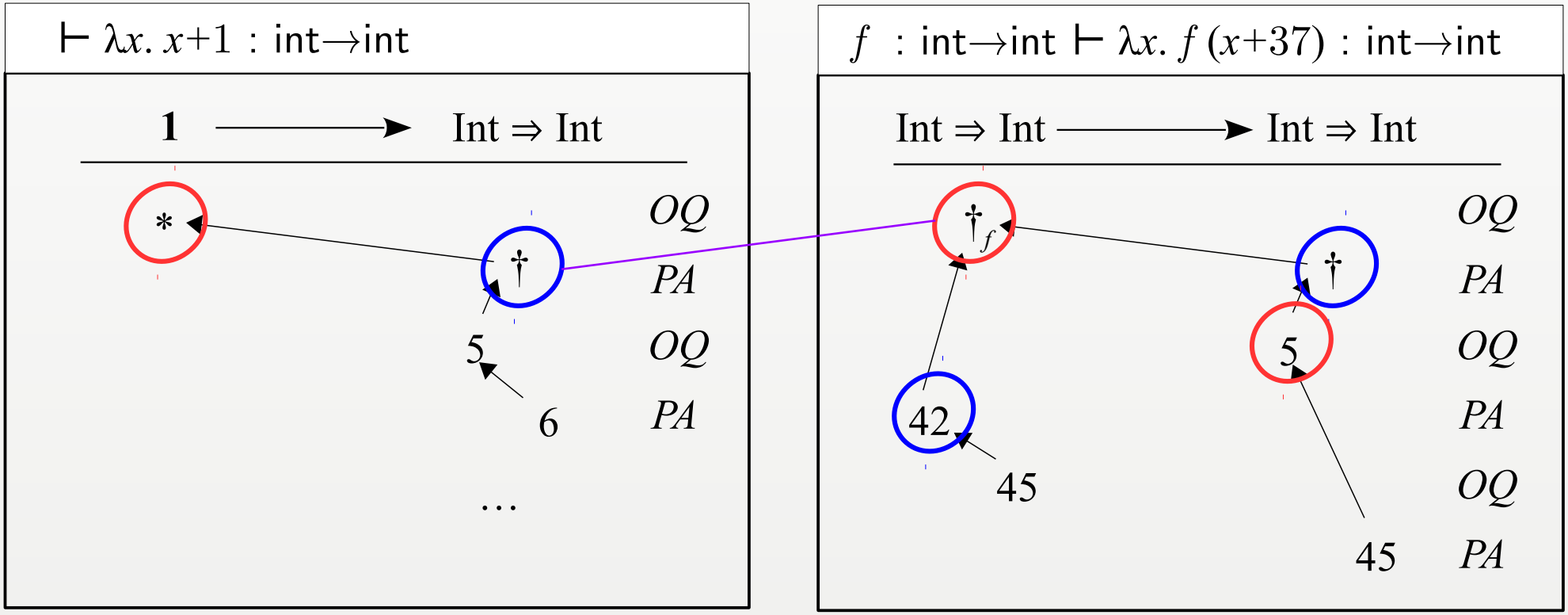
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

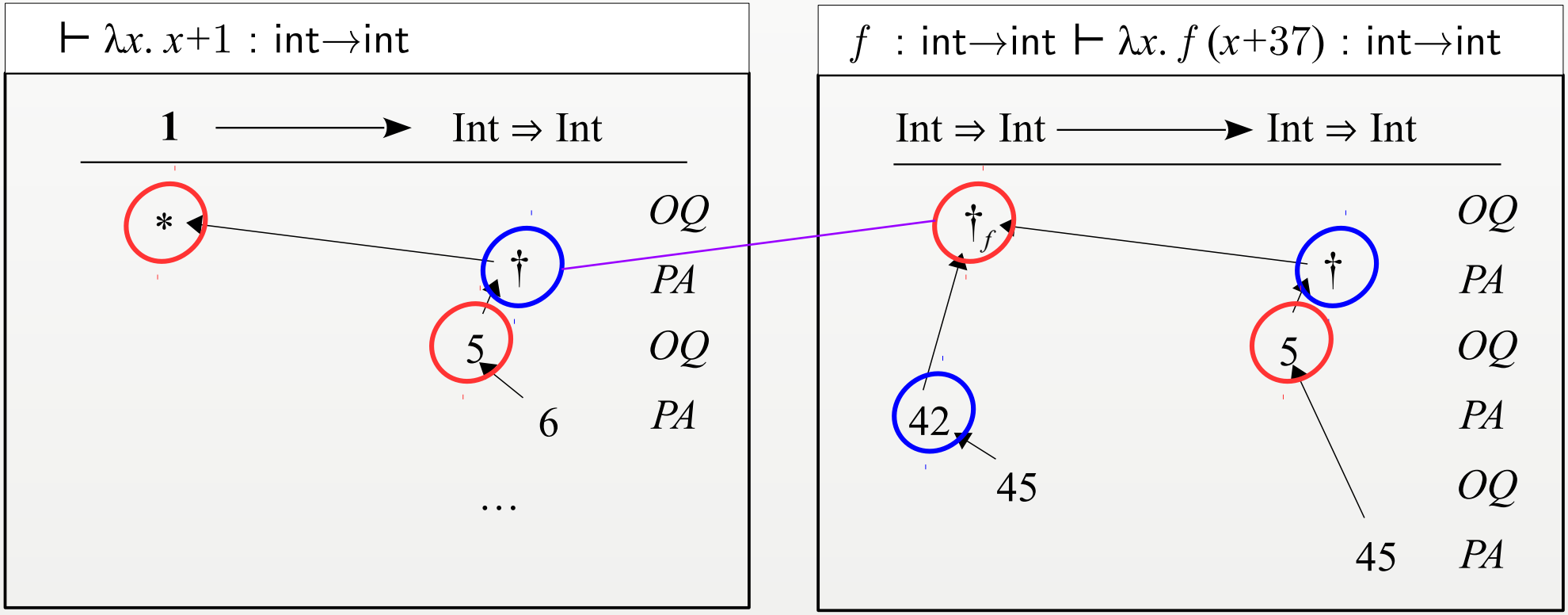
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

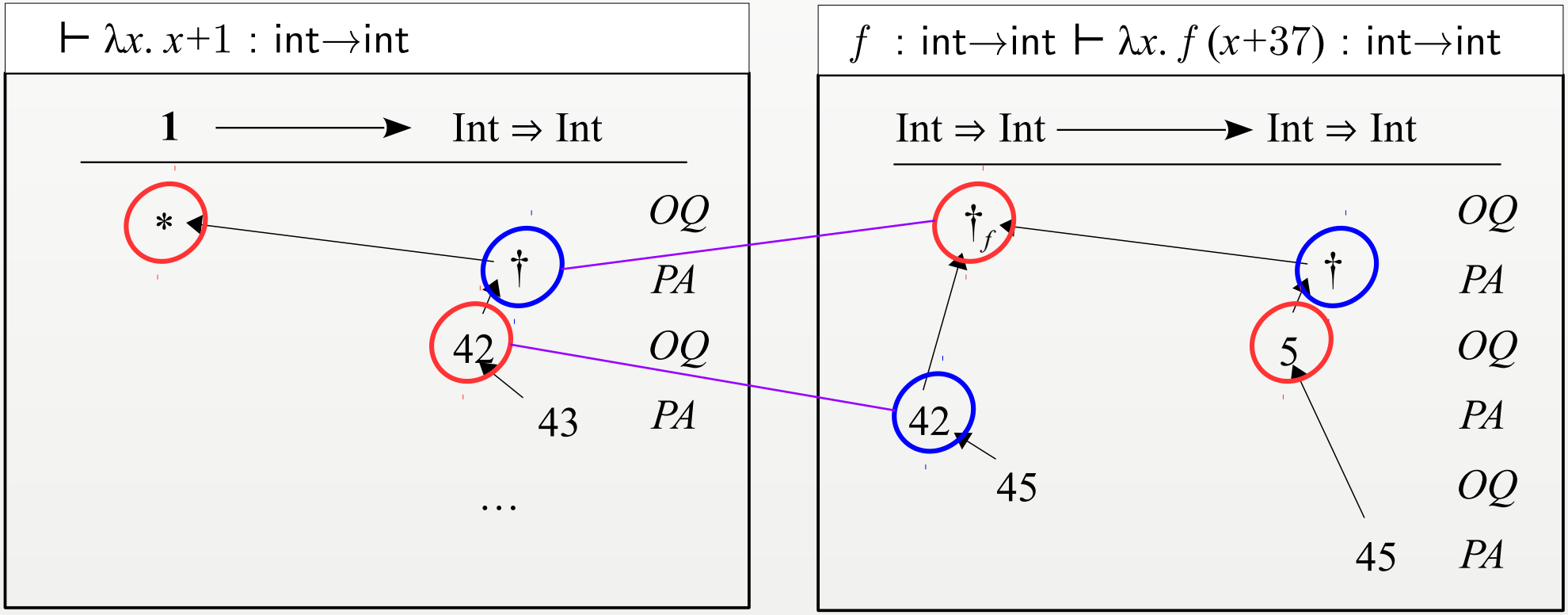
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

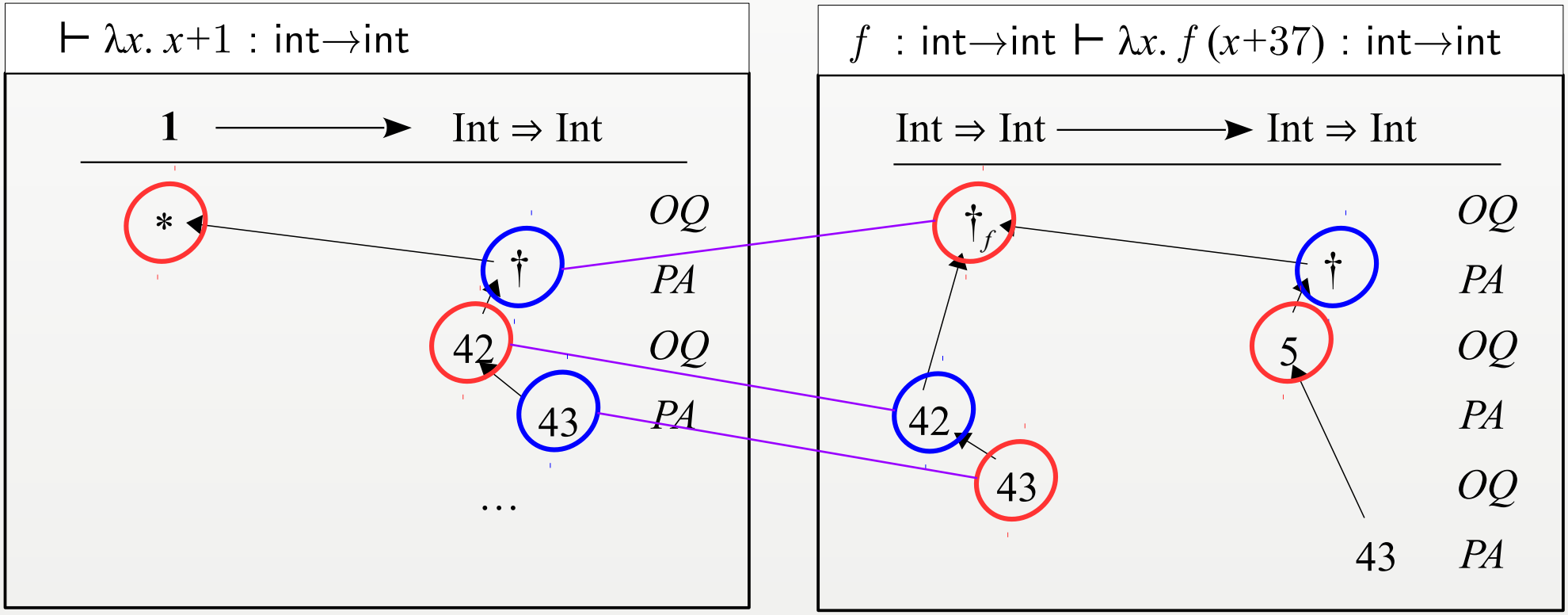
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

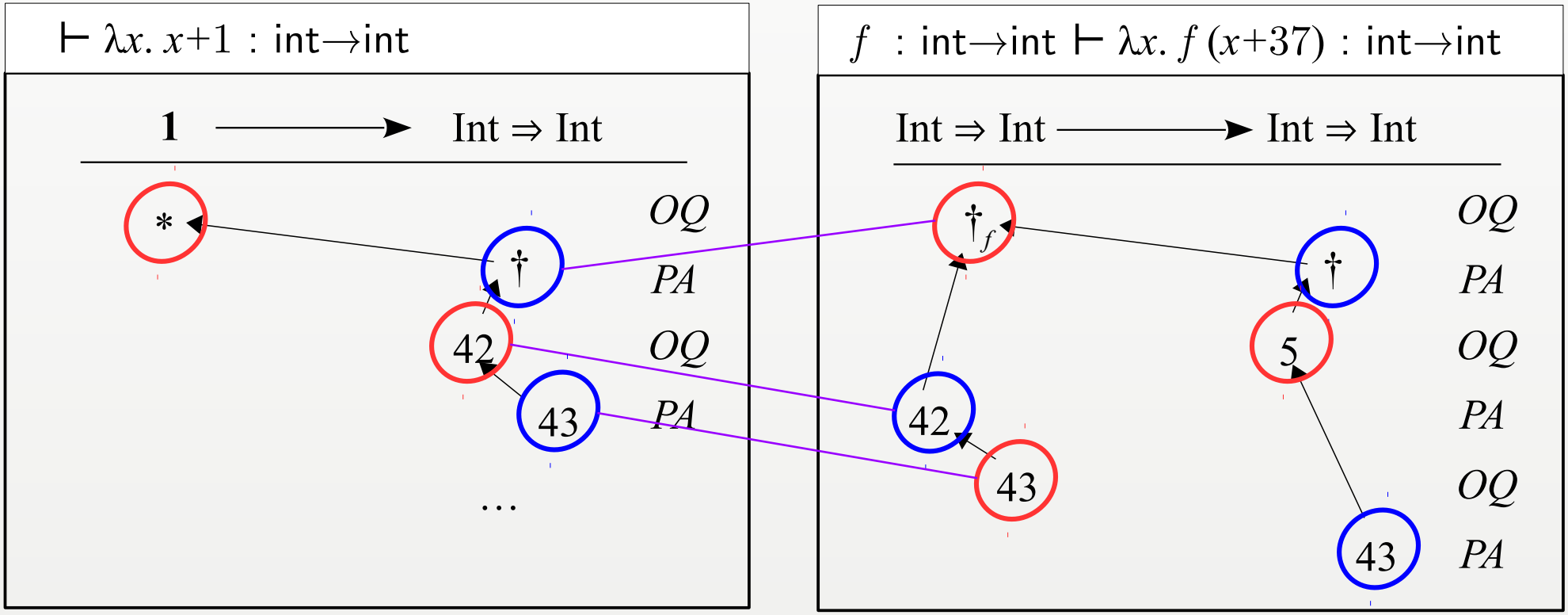
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

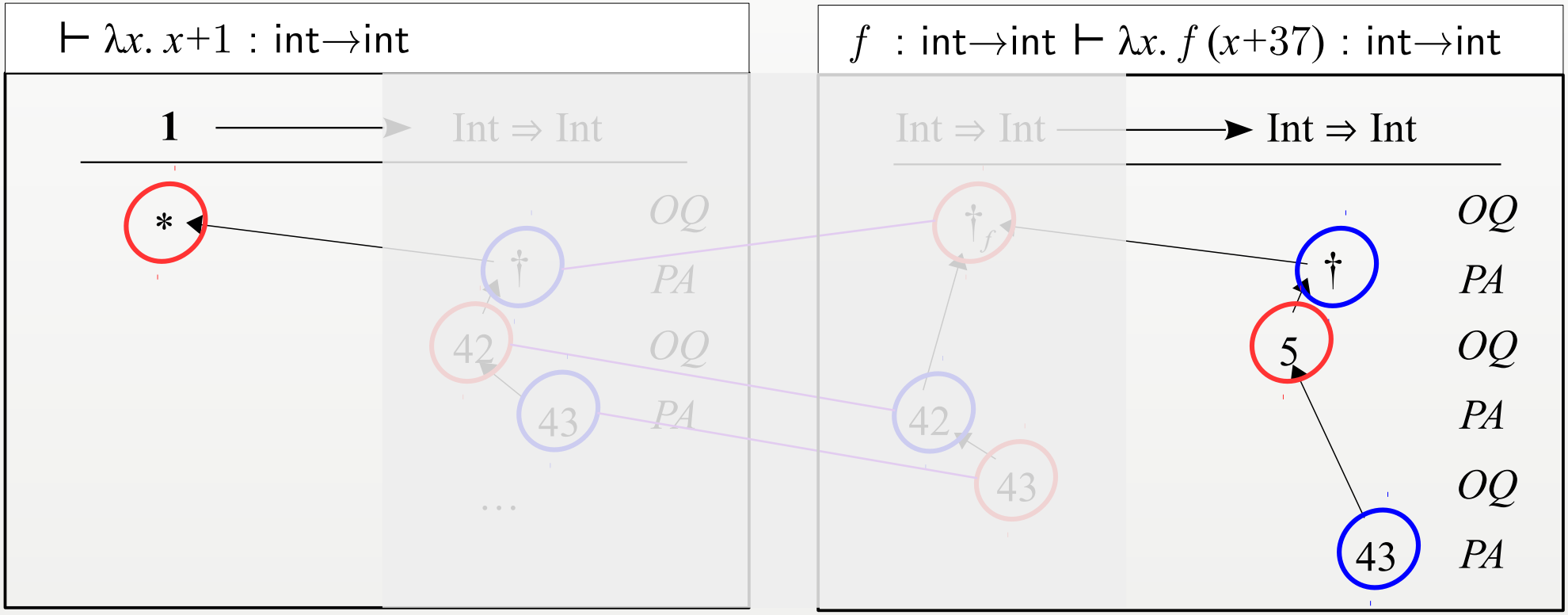
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37)$: int→int



Composition

Strategies can be composed by matching O/P behaviours

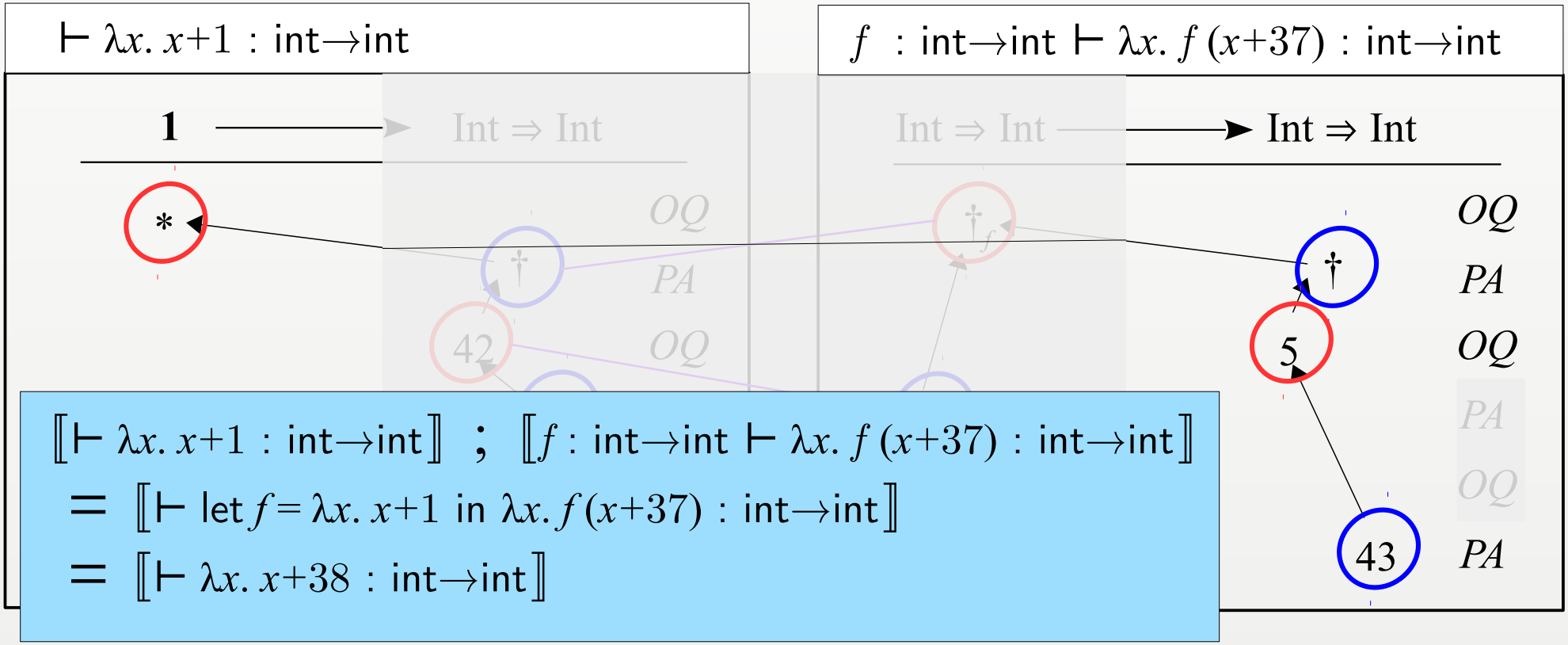
→ e.g. compute let $f = \lambda x.\, x{+}1$ in $\lambda x.\, f\,(x{+}37) : \mathsf{int}{\to}\mathsf{int}$



Composition

Strategies can be composed by matching O/P behaviours

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



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 combined via composition

→ categories of games and strategies (CCC's, Freyd, etc.)

→ well-behaved to model e.g. λ-calculus (PCF, IA, …)

1990-2004: Abramsky, Jagadeesan, Malacaria, Hyland, Ong, Nickau, McCusker, Ghica, Murawksi, Laird, Danos, Harmer, Honda, Yoshida, …

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 combined via composition
- Realistic programs → nominal games (2004-):
  - moves contain abstract atomic data (*names*)
  - for: references (ML), objects & threads (Java), etc.

λ-calculus + effects

$$M ::= x \mid \lambda x^{\vartheta}.M \mid M\ M$$
$$\mid i \mid \text{if } M \text{ then } M \text{ else } M \mid M + M$$
$$\mid a \mid \text{ref } M \mid M = M \mid M := M \mid !M$$

$x \in \text{Var}$

$i \in \text{Int}$

$a \in \text{Loc}_{\theta}$

$$\vartheta ::= \text{int} \mid \vartheta \to \vartheta \mid \text{ref } \vartheta$$

examples:

$42 : \text{int}$

$\text{ref } 42 : \text{ref int}$

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

can encode HO recursion ("Landin's knot"), and so all computable functions (PCF)

Typing rules

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

$$\frac{}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{\Gamma \vdash M:\mathsf{int} \quad \Gamma \vdash N:\mathsf{int}}{\Gamma \vdash M + N:\mathsf{int}} \qquad \frac{\Gamma \vdash M:\mathsf{int} \qquad \Gamma \vdash N, N':\vartheta}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ N':\vartheta}$$

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

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

Operational semantics (sample)

$$M,\ S \quad \rightarrow \quad M',\ S'$$

S stores pairs of locations + values

$$a := 23,\ S \quad \rightarrow \quad 23,\ S[a \mapsto 23] \qquad a \in \mathsf{Loc}_{\mathsf{int}}$$

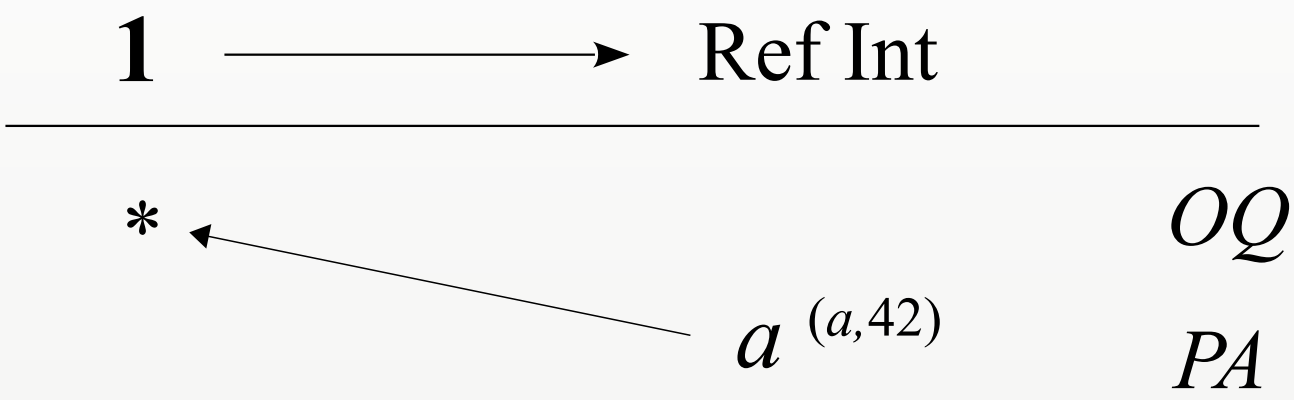
$$\mathsf{ref}\ 42,\ S \quad \rightarrow \quad a,\ S \uplus [a \mapsto 42] \qquad a \in \mathsf{Loc}_{\mathsf{int}}$$

$$\mathsf{ref}\ (\lambda x.x{+}1),\ S \quad \rightarrow \quad a,\ S \uplus [a \mapsto \lambda x.x{+}1] \qquad a \in \mathsf{Loc}_{\mathsf{int}\rightarrow\mathsf{int}}$$

Nominal games

$\vdash$ ref 42 : ref int

$[\![\text{ref } \theta]\!] = \{ a, b, \ldots \}$
$a, b, \ldots \in \text{Nam}_\theta$

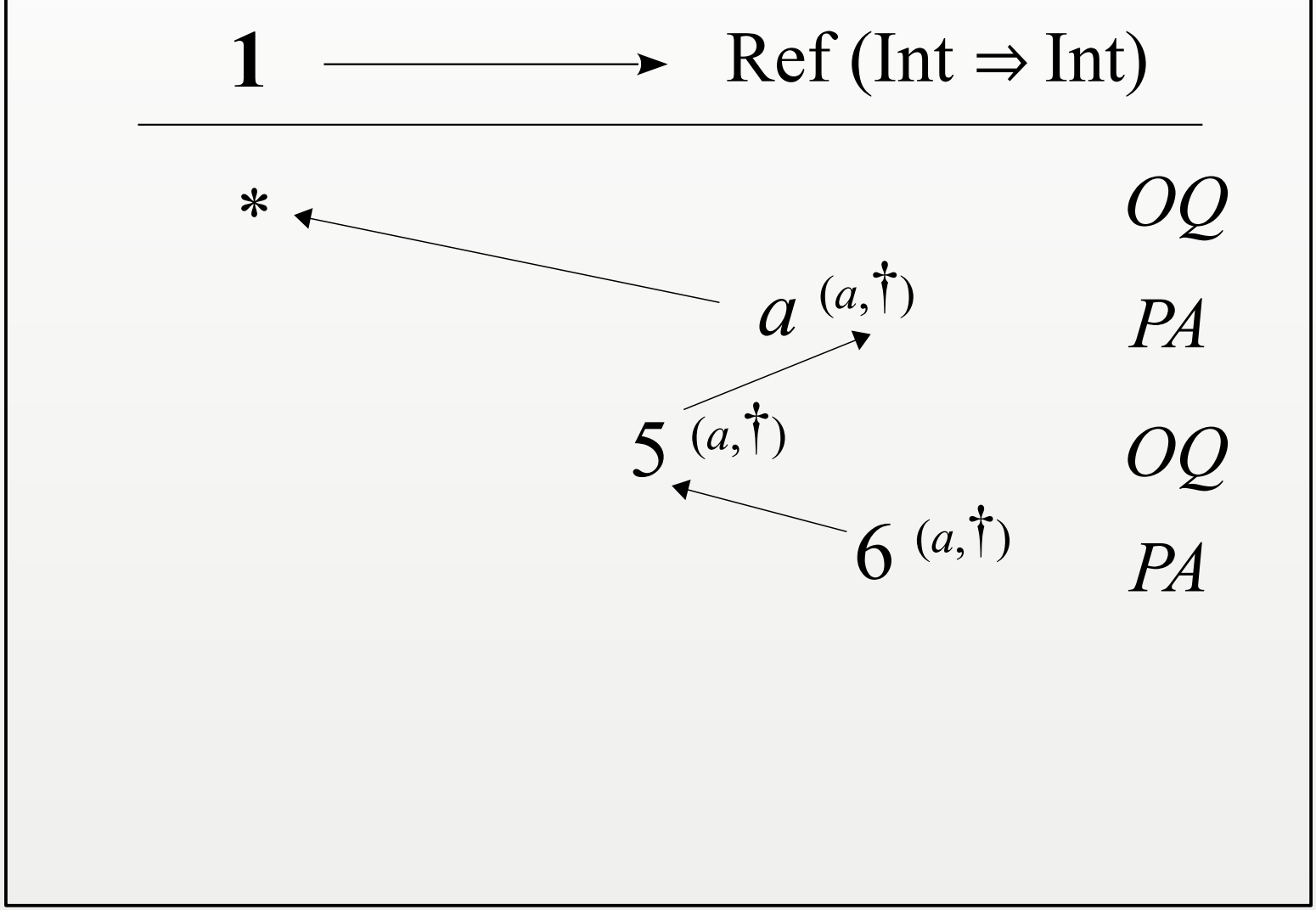


where: a is a name of type int ($a \in \text{Nam}_{\text{int}}$)

$[\![\text{ref } 42]\!] = \{ * \; a^{(a,42)} \}$
$OQ \quad PA$

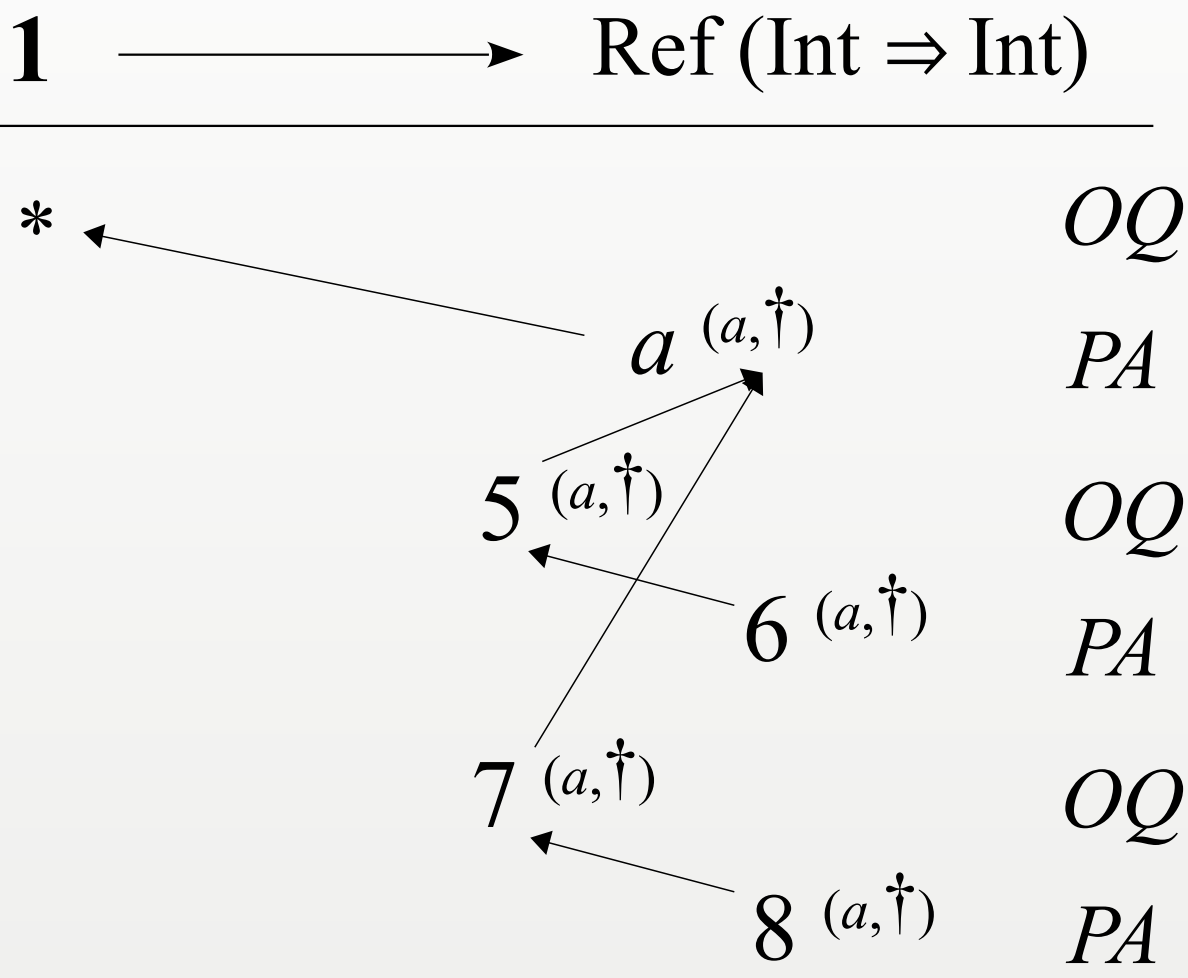
Nominal games

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



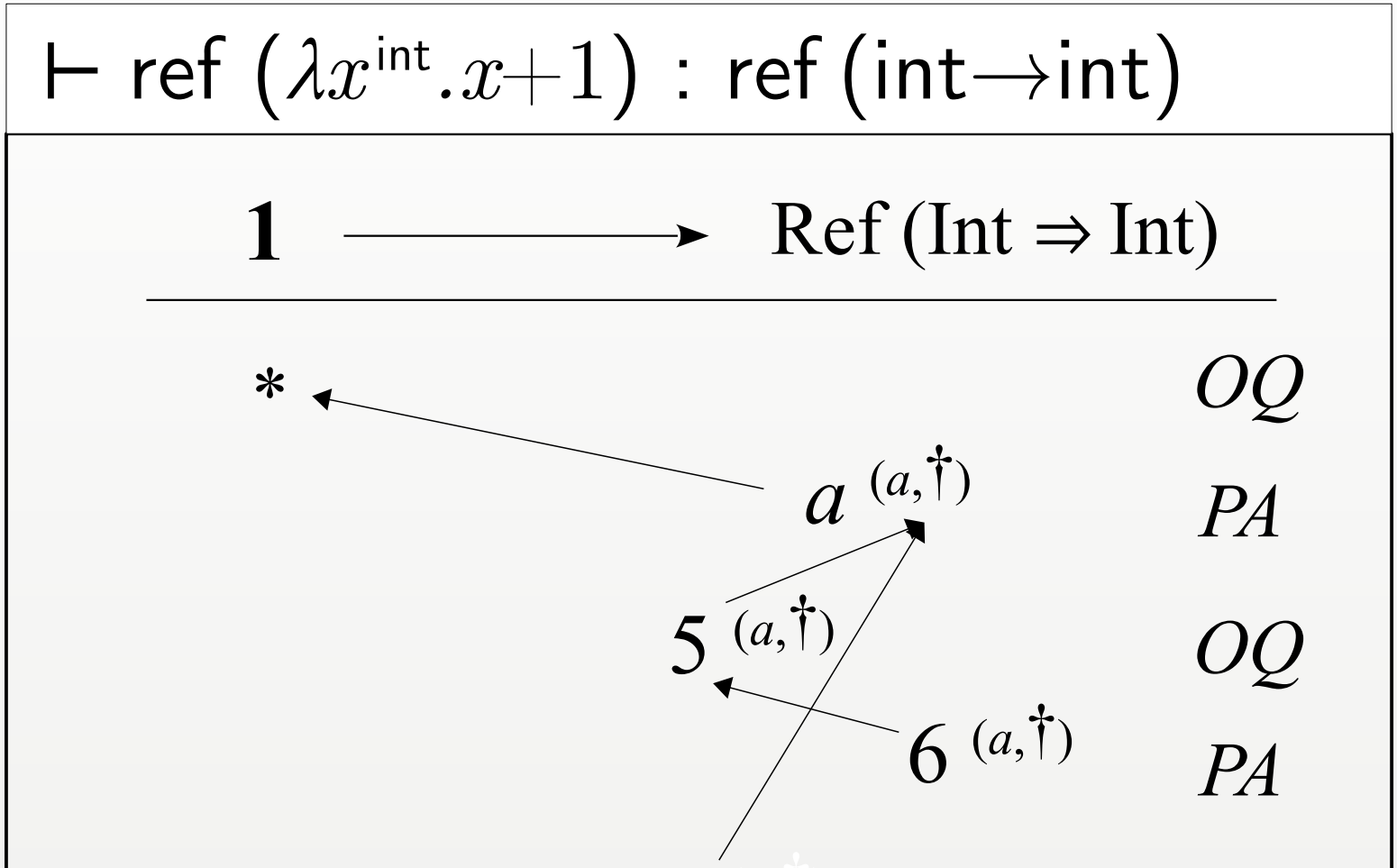
Nominal games

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



Nominal games

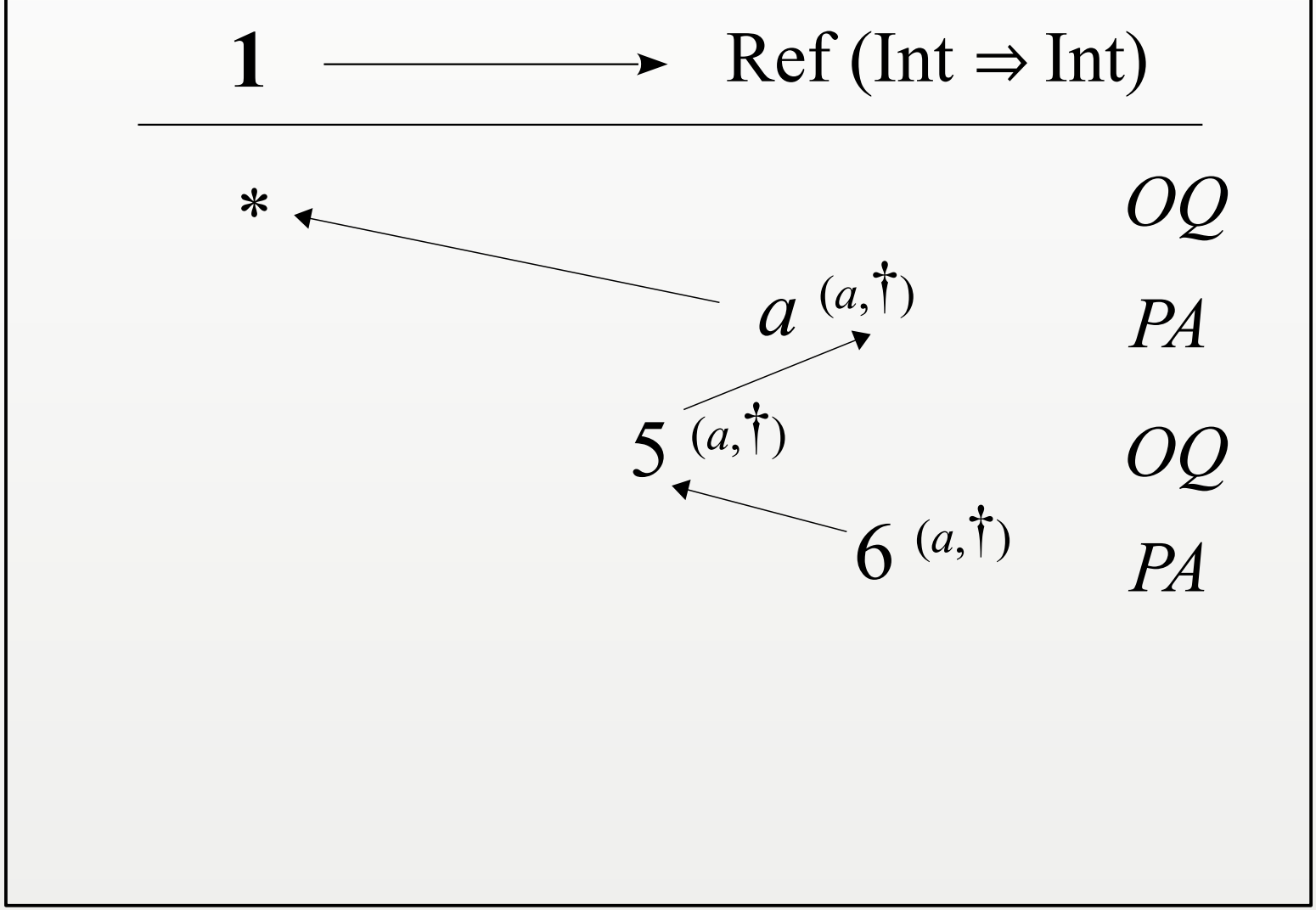
$\vdash \mathsf{ref}\ (\lambda x^{\mathsf{int}}.x+1) : \mathsf{ref}\ (\mathsf{int}\to\mathsf{int})$



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

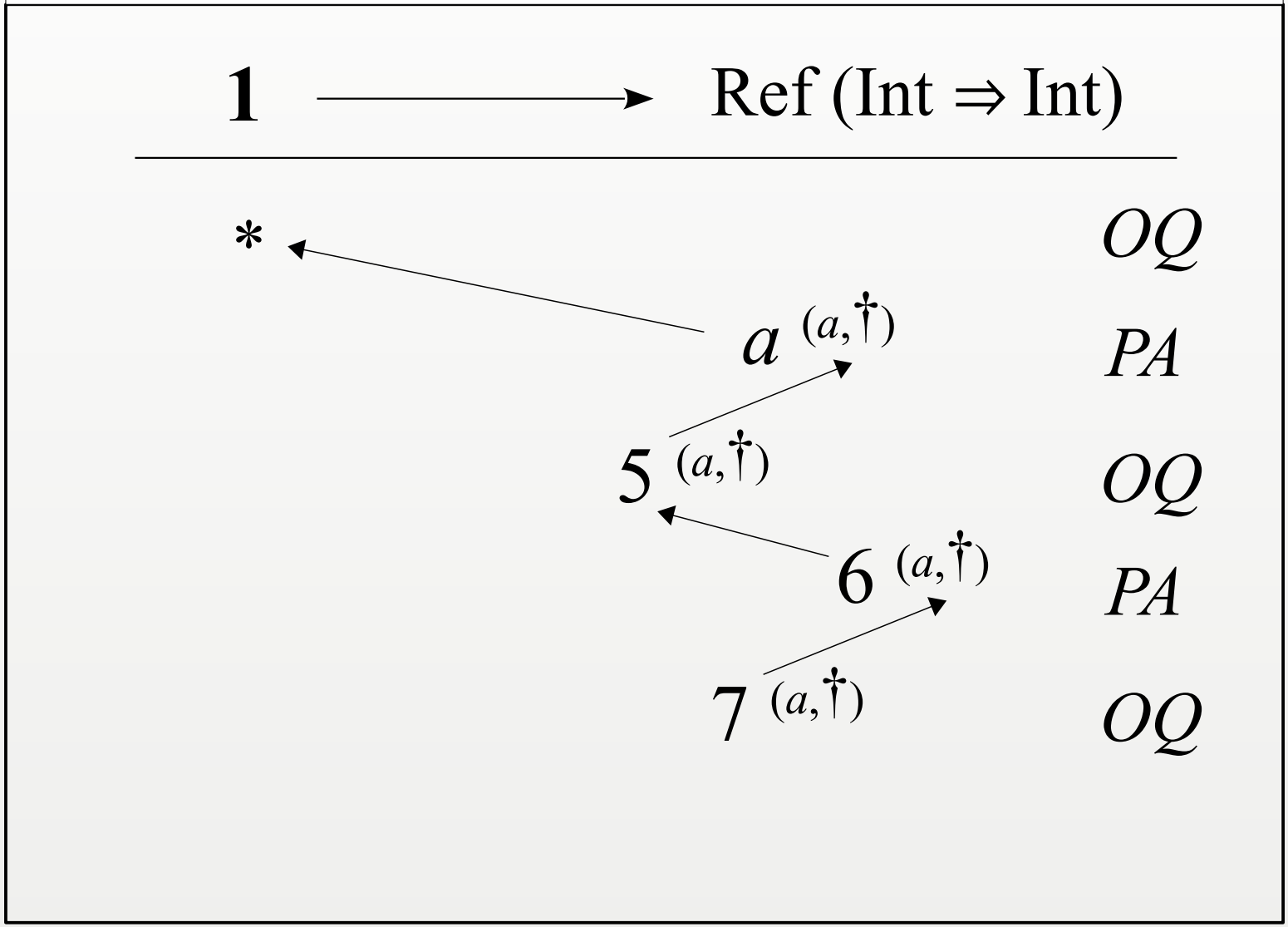
Nominal games

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



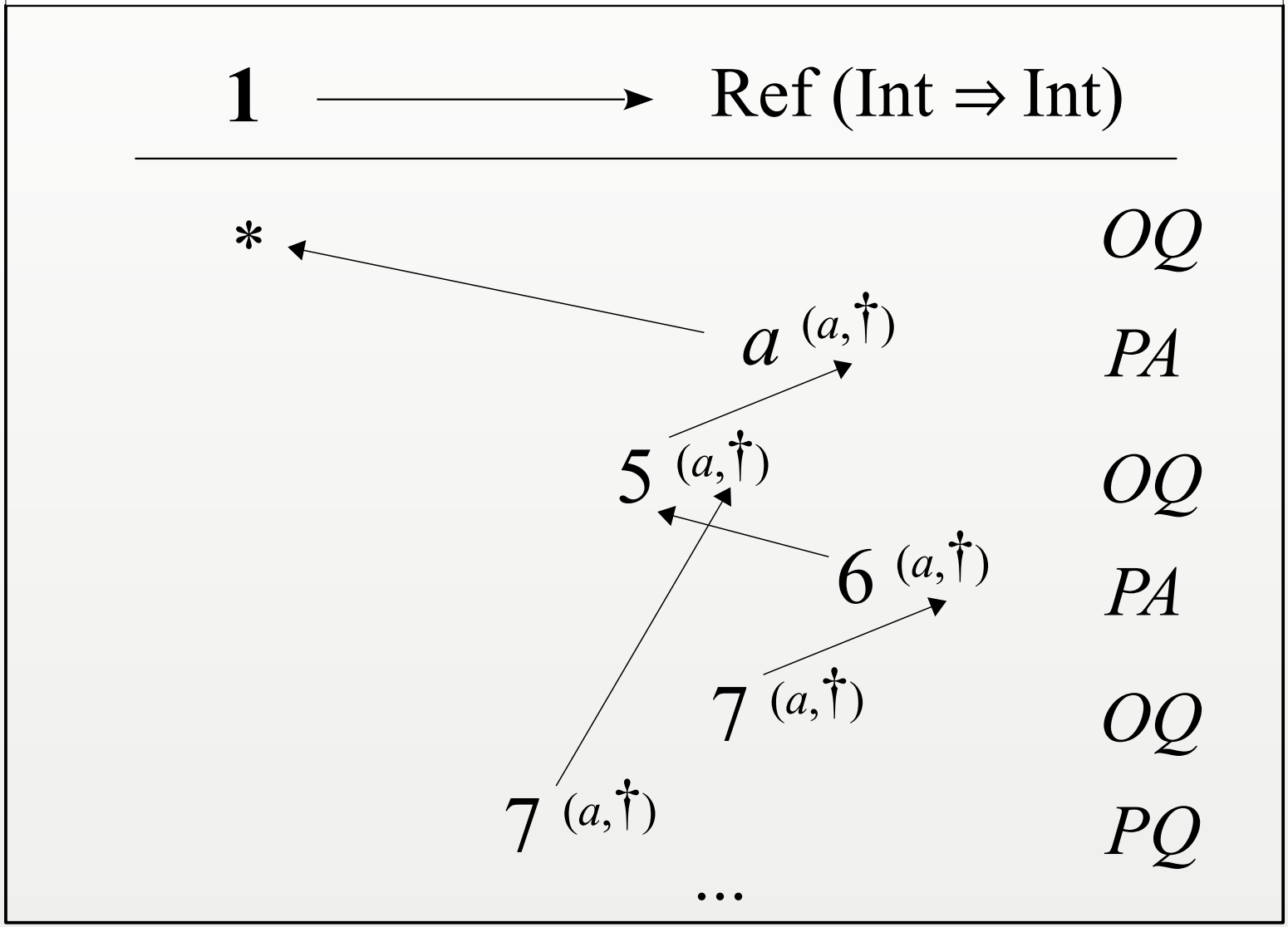
(there are more plays)

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



(there are more plays)

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



Game model construction

Game ingredients:

- work in nominal sets [Gabbay & Pitts '99]
- move-play infrastructure (e.g. stores)
- sets of conditions for plays and strategies (e.g. privacy)

Preparation (*correctness*):

- construct a category of games and strategies
- prove std conditions for lambda-calculus (CBV)
- model each effect constructor (e.g. define $[\![\text{ref}]\!]$)

Main results: $[\![M]\!] = [\![M']\!] \Leftrightarrow M \cong M'$

Nominal game models

- unit names

[Abramsky, Ghica, Murawski, Ong & Stark '04, Laird '04]

- ML-style references

[Laird '08, Murawski & T. '09, '11, '12]

- exceptions

[Murawski & T. '14]

- objects

[Murawski & T. '14]

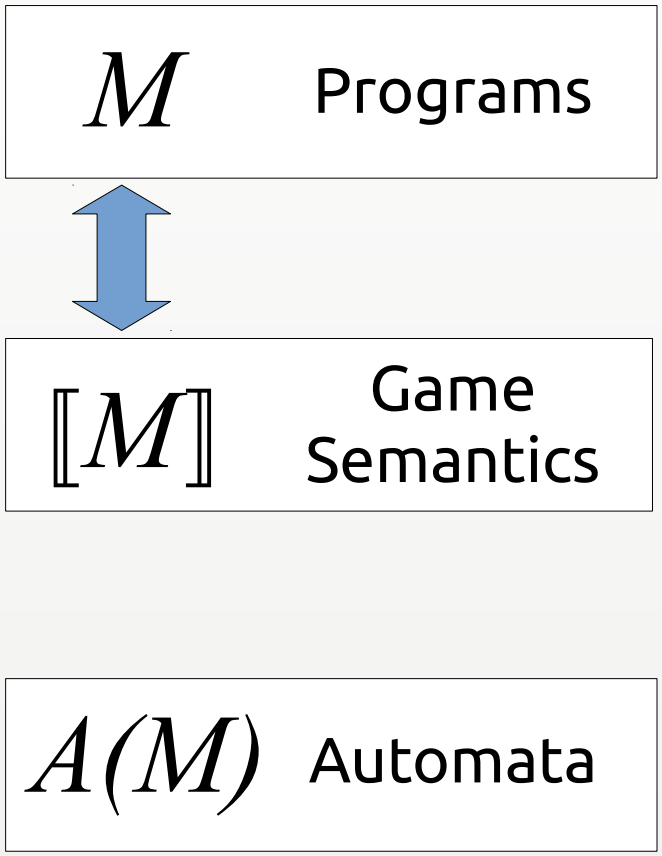
- low-level code (~C)

[Ghica & T. '12]

- polymorphism

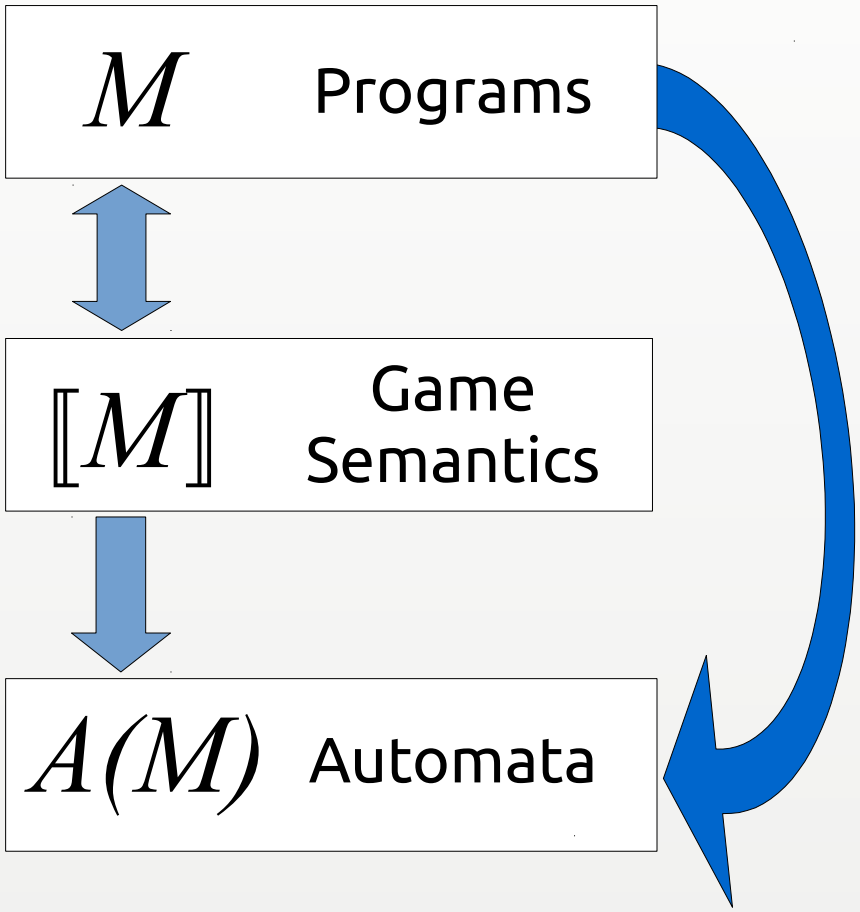
[Jaber & T. '16, '18]

Games and equivalence algorithmically



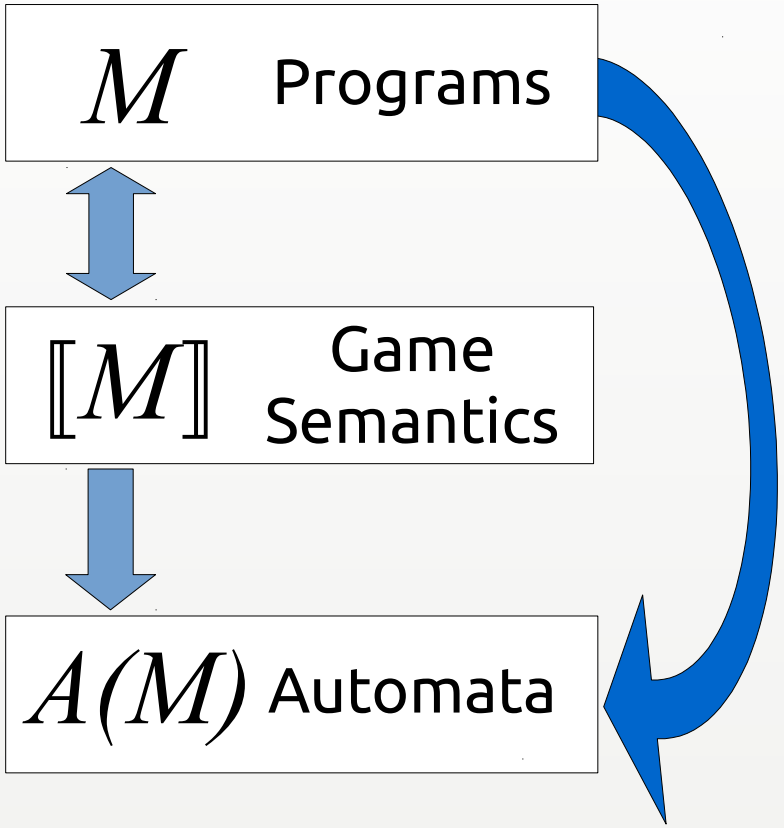
$$M \cong M' \Leftrightarrow [\![M]\!] = [\![M']\!]$$

Games and equivalence algorithmically



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

Games and equivalence algorithmically



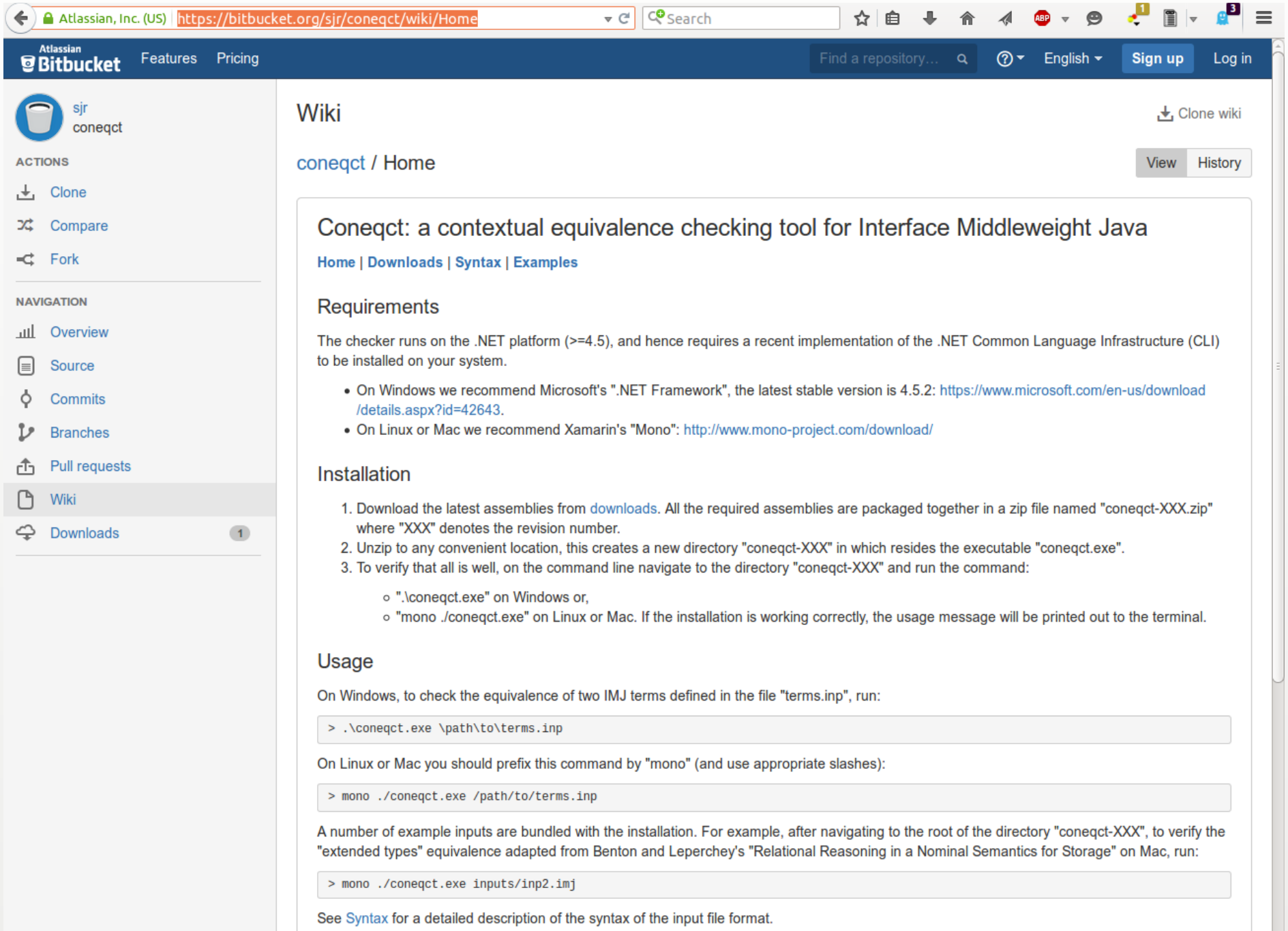
Transformation done automatically → decision procedure for program equivalence

- 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. '12; Murawski, Ramsay & T.' 15]

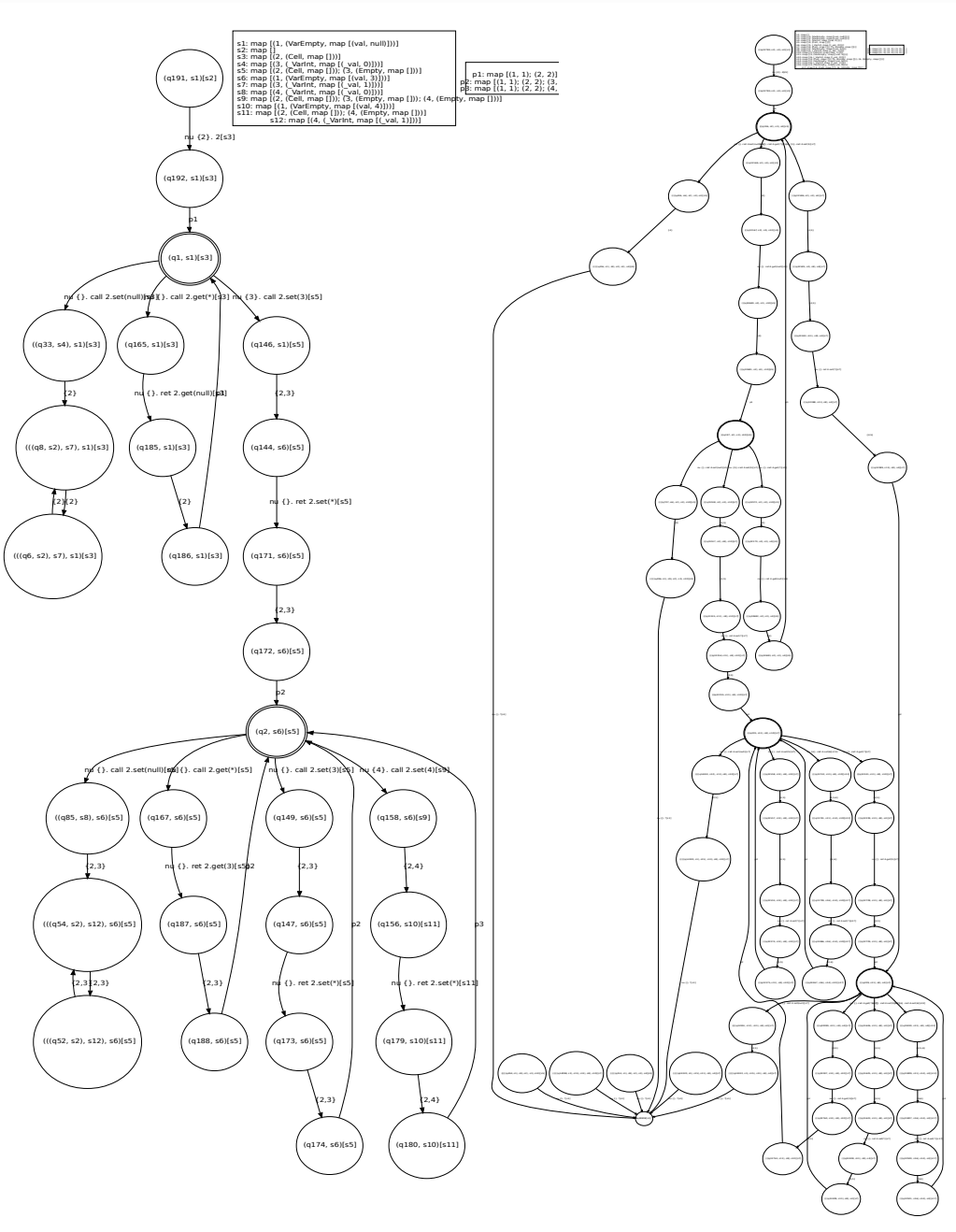
Coneqct


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

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

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



Recap and current work

Game semantics for modelling HO programs:

- compositional models, ‘operational functions’
- fully abstract models
- add names for generative effects: fragments of ML and Java, even C

Open problems:

- from language fragments to full languages
- applications to verification and compilation

Notes: Murawski & T., Nominal Game Semantics, *Foundations and Trends in Programming Languages*, 2016