Game Semantics for Nominal Exceptions

Andrzej Murawski

University of Warwick

Nikos Tzevelekos

Queen Mary University of London

FOSSACS 2014

What this talk is about

Fully abstract denotational models for higher-order languages with exceptions

The models are constructed in game semantics

They are the first such models for nominal exceptions

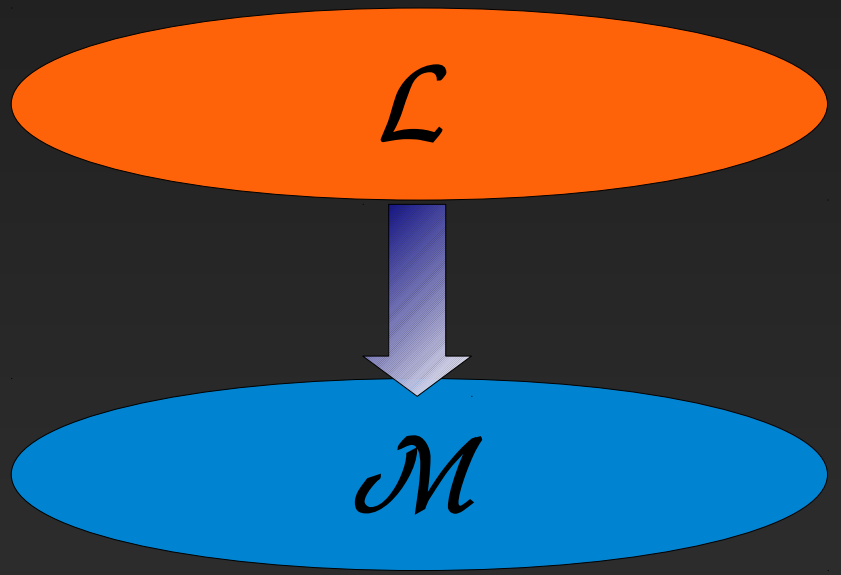
Denotational Semantics

Translate programs into a formal model (of 'functions'):

- abstract mathematical description
- free-of-syntax meaning

program: P

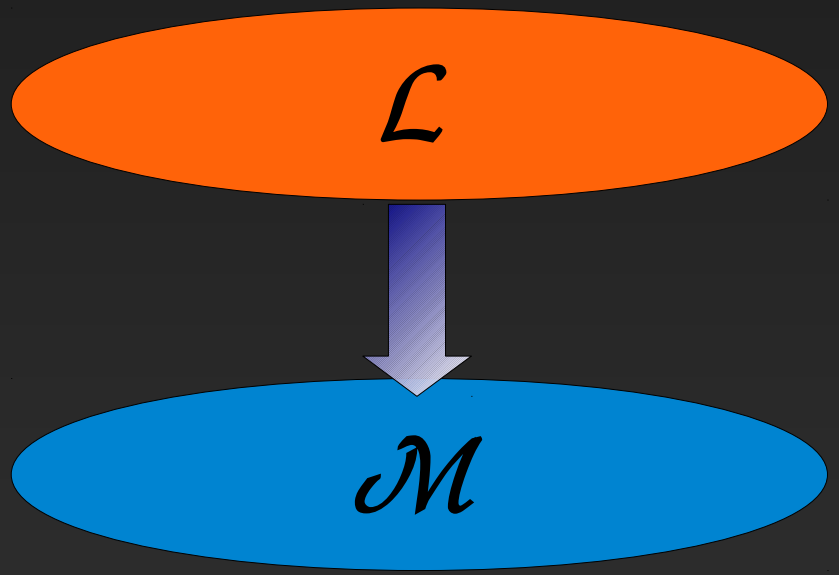
denotation: $[\![P]\!]$



Power of denotations

- Abstract away from implementation details
- Compositional translation:
  - modelling of components in isolation
  - modularity

program: P



denotation: $[\![P]\!]$

Useful for understanding & analysing programs

Full Abstraction

Ideally, a denotational model exactly describes the meaning, i.e. the observable behaviour, of programs

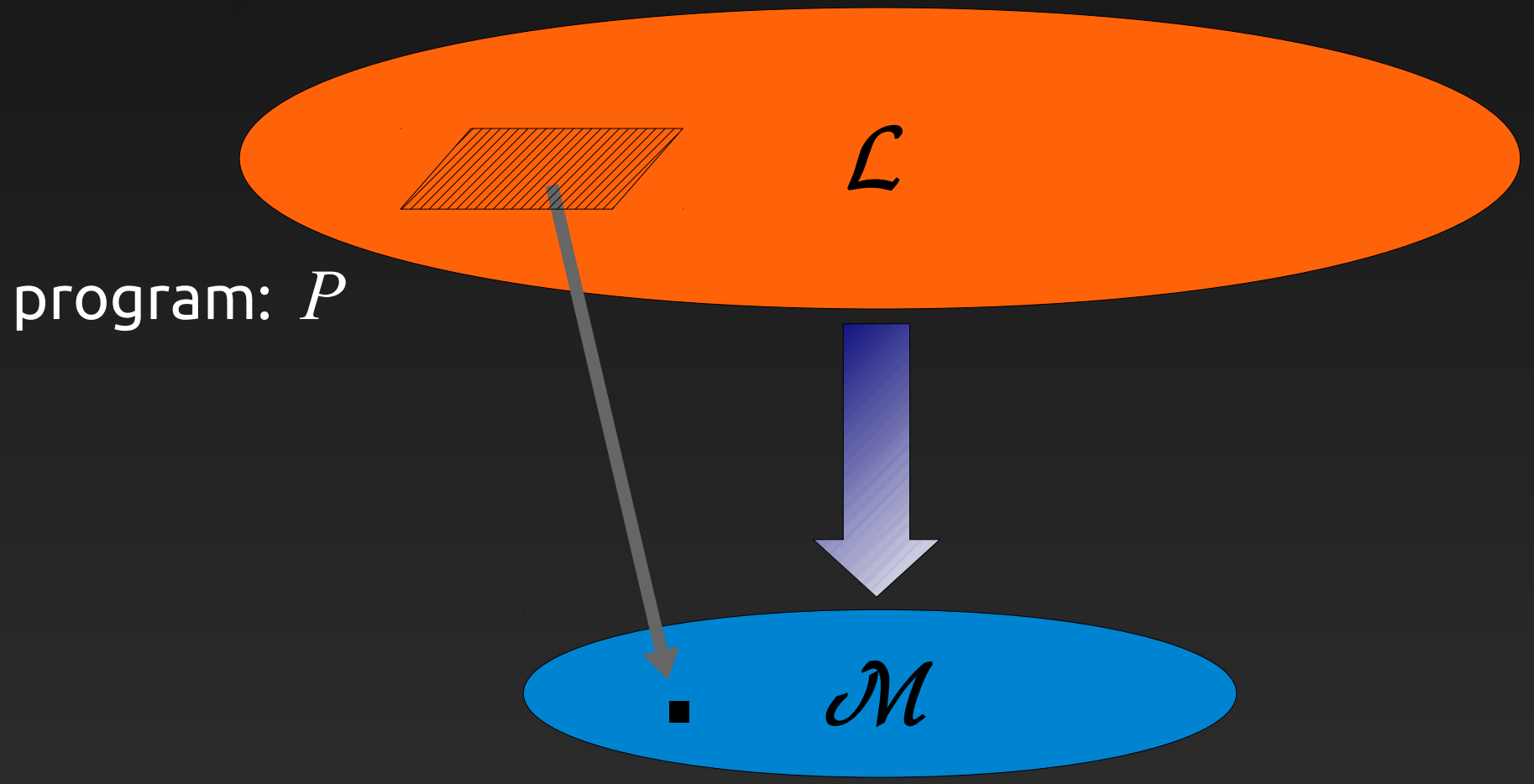
- Full abstraction (equational):

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

- Full abstraction (inequational):

$$P \sqsubseteq P' \iff [\![P]\!] \subseteq [\![P']\!]$$

Full abstraction pictorially



The quest for full abstraction

1977 [Milner, Plotkin]:

- Formulation of the problem
- Functions cannot model sequentiality

1980-90's:

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

1993 [AJM, HO/N *]: Full abstraction for PCF

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

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

Game semantics so far

From PCF towards general HO-programs:

- Variants of Idealized Algol:
  - non-determinism, probabilities, concurrency, polymorphism, exceptions, …
- Nominal game semantics (2004-):
  - *use names for dynamic resource generation*

    → fragments of ML, CML, C, Java

The need for names

The need for names

Exceptions as *pairs:*

$$\texttt{exn} = (\texttt{unit} \to \texttt{emp})\ \texttt{x}\ ((\texttt{unit} \to \texttt{emp}) \to \texttt{unit})$$

$$\Longrightarrow\ (1 \to 0)\texttt{x}((1 \to 0) \to 1)$$

- Theoretically attractive
- but: $\texttt{mkexn}(R, H)$, all R, H (*bad exceptions*)

The need for names

Exceptions as *pairs:*

$\texttt{exn} = (\texttt{unit} \to \texttt{emp}) \times ((\texttt{unit} \to \texttt{emp}) \to \texttt{unit})$

$\Longrightarrow \; (1 \to 0) \times ((1 \to 0) \to 1)$

- Theoretically attractive
- but: $\texttt{mkexn}(R, H)$, all R, H (*bad exceptions*)

Exceptions as *names:*

$\texttt{exn} = \texttt{base type}$

$\Longrightarrow \; \mathcal{E}$ (exception names)

- Notion of *resource (name):*
  - atomic values
  - infinitely many
  - comparable for equality

The use of names in games allows us to pass from 'idealised' resources to 'realistic' ones

ExnML: HO-programs with exceptions

Types

$$\beta ::= \text{unit} \mid \text{int} \mid \text{exn} \mid \text{ref}\ \beta$$

$$\theta ::= \beta \mid \theta \rightarrow \theta$$

ExnML: HO-programs with exceptions

Types $\beta ::= \text{unit} \mid \text{int} \mid \text{exn} \mid \text{ref}\ \beta$

$\theta ::= \beta \mid \theta \to \theta$

Terms $M ::= () \mid 0 \mid \ldots \mid \Omega \mid x \mid \lambda x^{\theta}.M \mid MM$

$\mid \text{if0}\ MMM \mid M \oplus M$

ExnML: HO-programs with exceptions

Types $\beta ::= \text{unit} \mid \text{int} \mid \text{exn} \mid \text{ref}\ \beta$

$\theta ::= \beta \mid \theta \to \theta$

Terms $M ::= () \mid 0 \mid \ldots \mid \Omega \mid x \mid \lambda x^{\theta}.M \mid MM$

$\mid \text{if0}\ MMM \mid M \oplus M$

$\mid l \mid e \mid M = M$

$\mid \text{exn}() \mid \text{raise}\ M \mid M\ \text{handle}\ x \Rightarrow M$

$\mid \text{ref}_{\beta}(M) \mid !M \mid M := M$

location names: $l \in \mathcal{L}_{\theta}$

exception names: $e \in \mathcal{E}$

ExnML: operational semantics

$$\Sigma, M \longrightarrow \Sigma', M'$$

Σ stores exception names and location names&values

$$\Sigma, \mathsf{ref}_\beta(v) \longrightarrow \Sigma \uplus \{(l, v)\}, l \qquad \Sigma, l = l' \longrightarrow \Sigma, 0/1$$

$$\Sigma, l := v \longrightarrow \Sigma[l \mapsto v], () \qquad \Sigma, !l \longrightarrow \Sigma, \Sigma(l)$$

ExnML: operational semantics

$$\Sigma, M \longrightarrow \Sigma', M'$$

Σ stores exception names and location names&values

$$\Sigma, \text{exn}() \longrightarrow \Sigma \uplus \{e\}, e \qquad \Sigma, e = e' \longrightarrow \Sigma, 0/1$$

$$\Sigma, v \text{ handle } x \Rightarrow M \longrightarrow \Sigma, v$$

$$\Sigma, (\text{raise } e) \text{ handle } x \Rightarrow M \longrightarrow \Sigma, M[e/x]$$

$$\Sigma, E_{\neg H}[\text{raise } e] \longrightarrow \Sigma, \text{raise } e$$

$$\Sigma, \text{ref}_\beta(v) \longrightarrow \Sigma \uplus \{(l, v)\}, l \qquad \Sigma, l = l' \longrightarrow \Sigma, 0/1$$

$$\Sigma, l := v \longrightarrow \Sigma[l \mapsto v], () \qquad \Sigma, !l \longrightarrow \Sigma, \Sigma(l)$$

Nominal exceptions in OCaML

```
# exception MyExn
  let e1 = MyExn
  let e2 = MyExn
  let foo x = raise x;;

# try foo(e1) with x -> (x==e1, x==e2);;
```

Similar examples for Java and SML

ExnML example

M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

ExnML example

M_1 : let $y = \text{exn}()$ in $\lambda x^{\text{unit}}.$ raise y : unit $\to$ unit

M_2 : $\lambda x^{\text{unit}}.$ raise $(\text{exn}())$: unit $\to$ unit

$$M_1 \ncong M_2$$

let $g = [\_]$ in
  let $f = \lambda z^{\text{unit}}.\ g()$ handle x => x in
    $f() = f()$

Game semantics

Computation = a 2-player game between:

- *Opponent* (the environment, O)
- *Proponent* (the program, P)

Qualitative games (≠ Game Theory)

Game semantics

Computation = a 2-player game between:

- *Opponent* (the environment, O)
- *Proponent* (the program, P)

Qualitative games (≠ Game Theory)

Computations = *plays* of a specified game

Programs = *strategies* for P

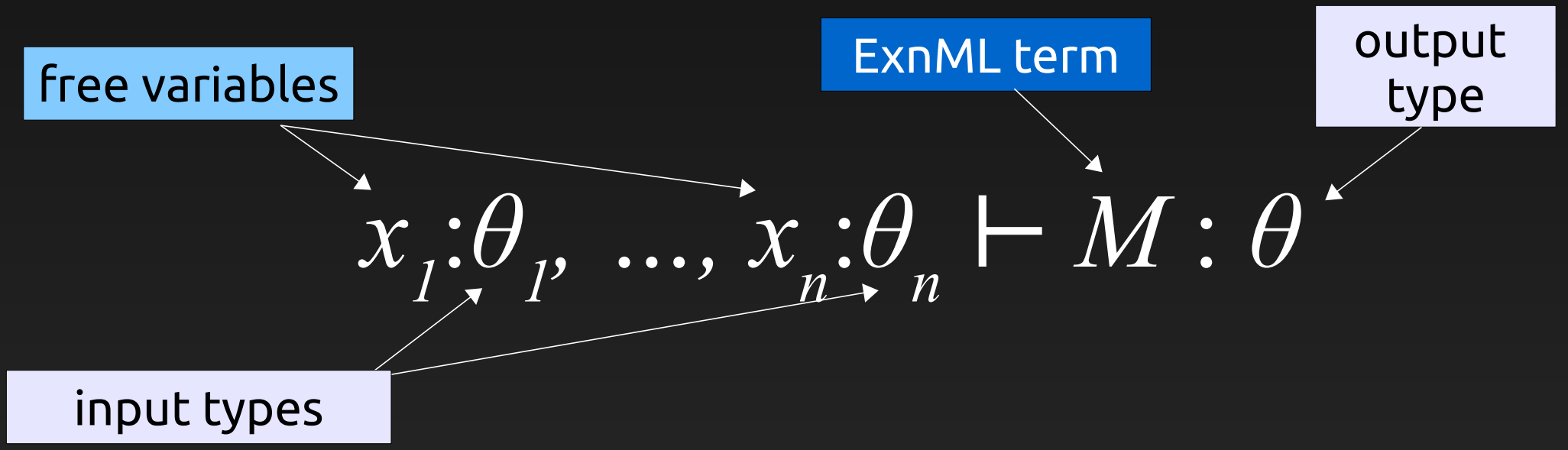
Strategies compose → *categories* of games

Games in detail

$$x_1{:}\theta_1,\ \ldots,\ x_n{:}\theta_n \vdash M : \theta$$

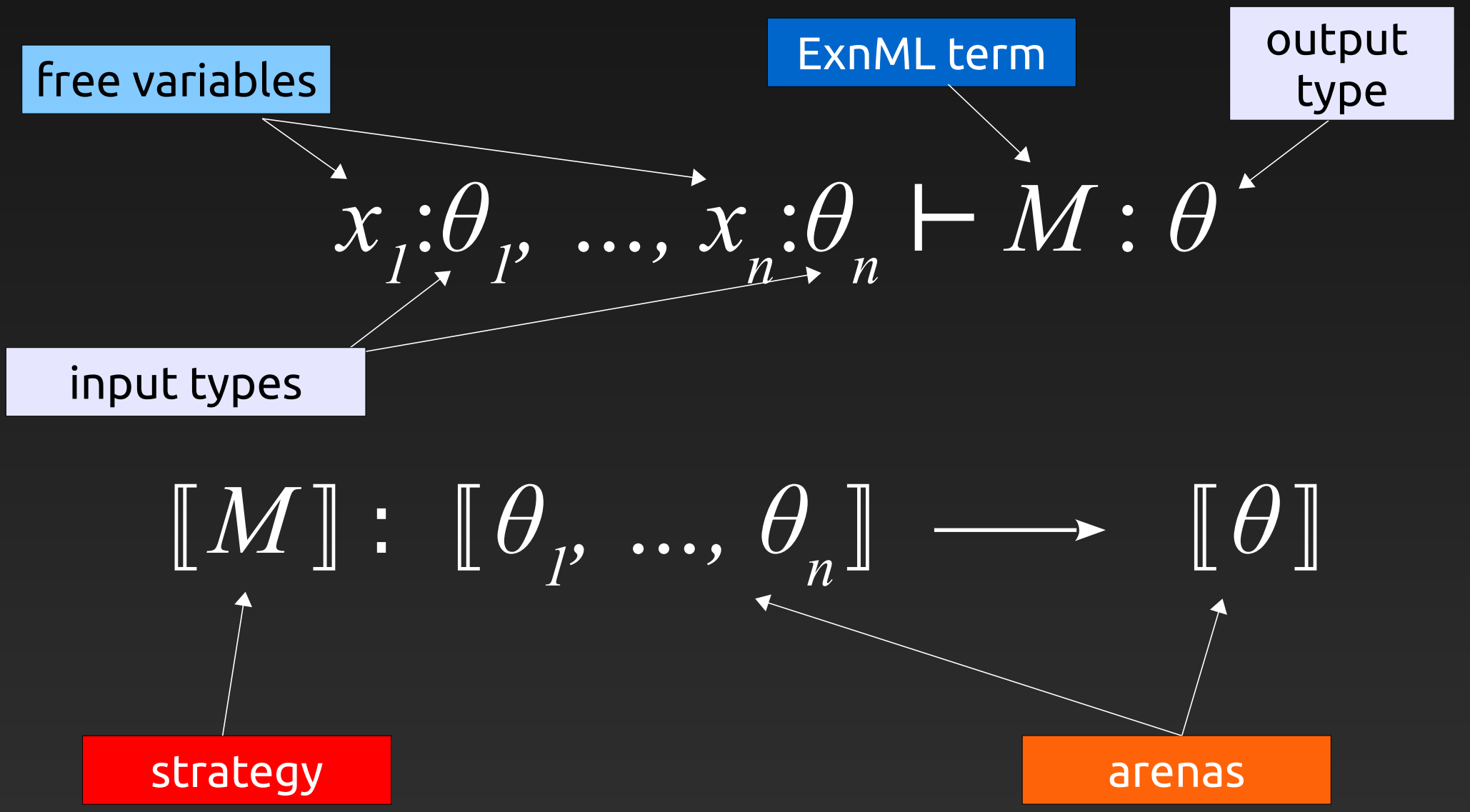
$$[\![M]\!] :\ [\![\theta_1,\ \ldots,\ \theta_n]\!] \longrightarrow\ [\![\theta]\!]$$

Games in detail

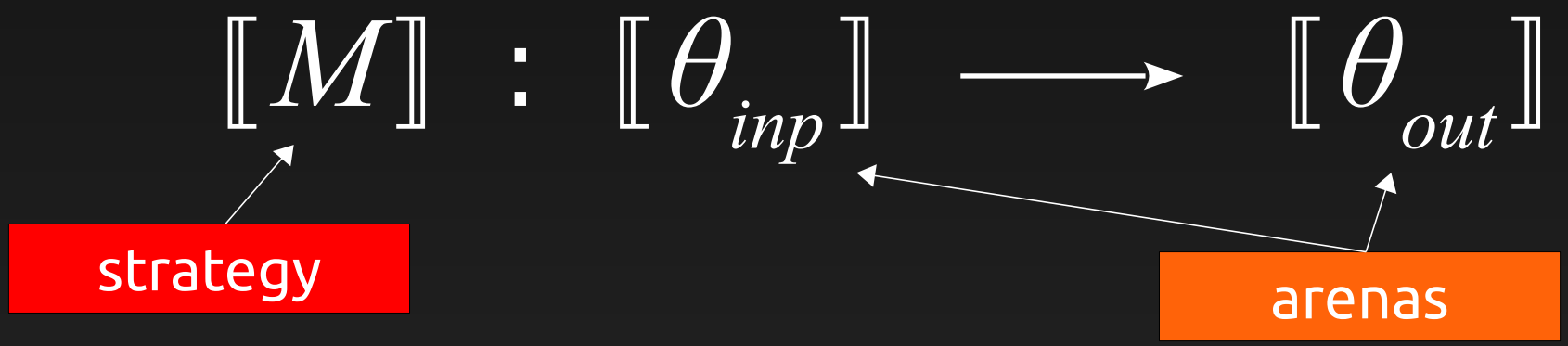


$$[\![M]\!] : [\![\theta_1, \ldots, \theta_n]\!] \longrightarrow [\![\theta]\!]$$

Games in detail



Arenas, moves



Arenas: sets of *moves* with:

- assigned polarities: *O*/*P* and *Q*/*A*
- dependence (justification) pointers
- names (locations and unraised exceptions)
- exceptional answers $e!$ for *O* and *P*

Arenas, moves

$$[\![M]\!] : [\![\theta_{inp}]\!] \longrightarrow [\![\theta_{out}]\!]$$

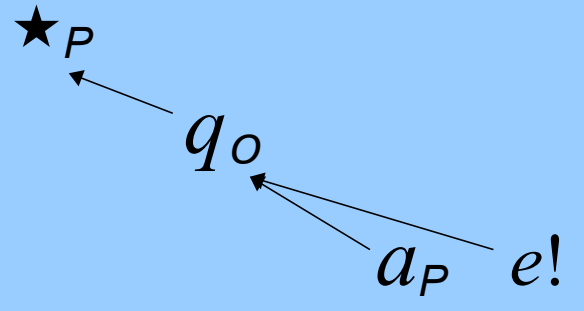
strategy

arenas

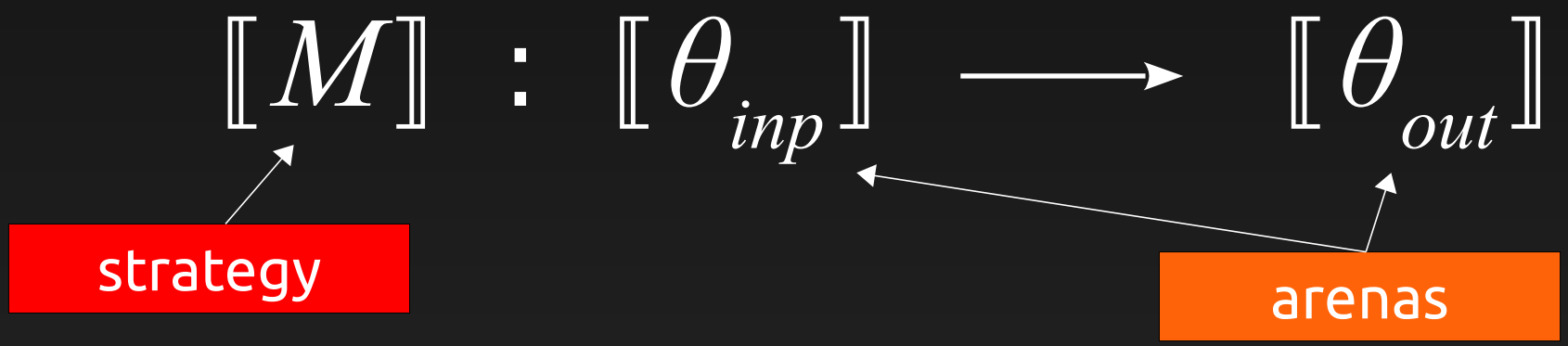
Arenas: sets of *moves* with:

- assigned polarities: *O*/*P* and *Q*/*A*
- dependence (justification) pointers
- names (locations and unraised exceptions)
- exceptional answers $e!$ for *O* and *P*

$[\![\text{unit} \to \text{unit}]\!]$:



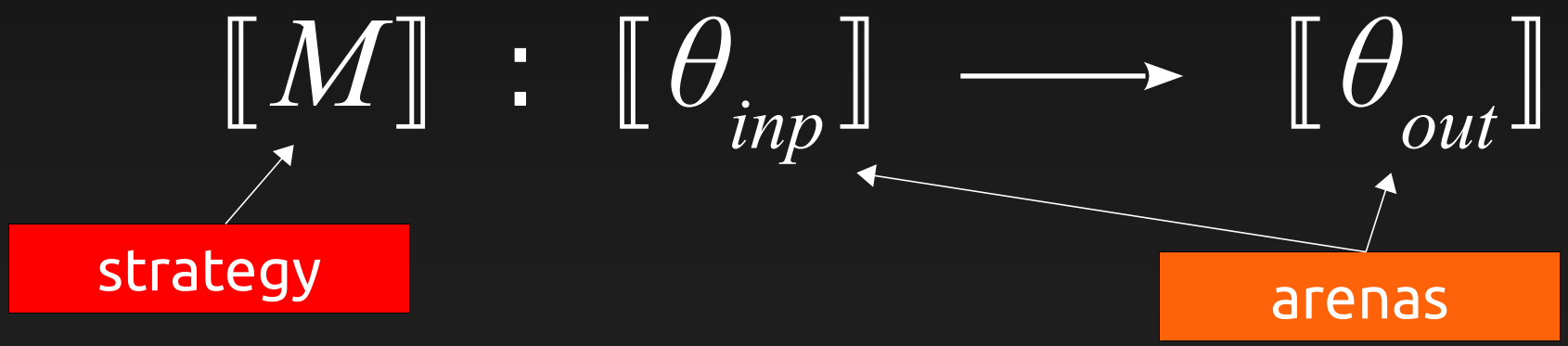
Strategies, plays



Strategies : sets of *plays*, i.e. sequences of moves-with-store, satisfying:

- *O*/*P* alternation
- well bracketing of *Q*'s and *A*'s
- determinacy, modulo fresh names
- …

Strategies, plays



Strategies : sets of *plays*, i.e. sequences of moves-with-store, satisfying:

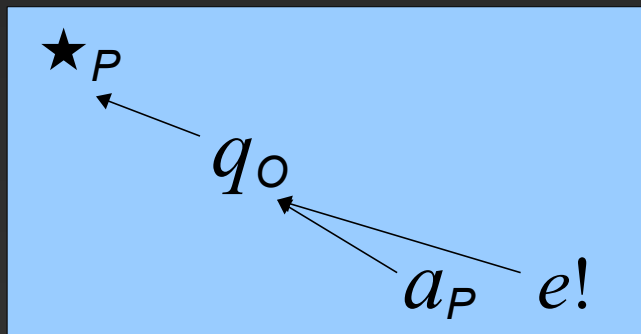
- *O*/*P* alternation
- well bracketing of *Q*'s and *A*'s
- determinacy, modulo fresh names
- …

Composition : via interaction, respecting name privacy

ExnML example: game semantics

M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\,(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$



ExnML example: game semantics

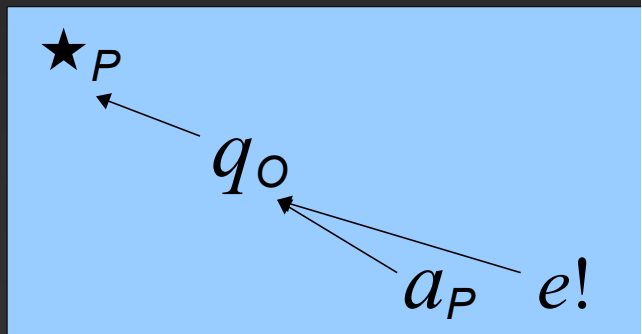
M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

$$[\![M_1]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1} \ldots$$
$$P\ \ O\quad P\quad O\quad P\quad O\quad P$$

$$[\![M_2]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_2!^{\Sigma_2}\ q^{\Sigma_2}\ e_3!^{\Sigma_3} \ldots$$

$$\Sigma_i = \{ e_1, \ldots, e_i \}$$



ExnML example: game semantics

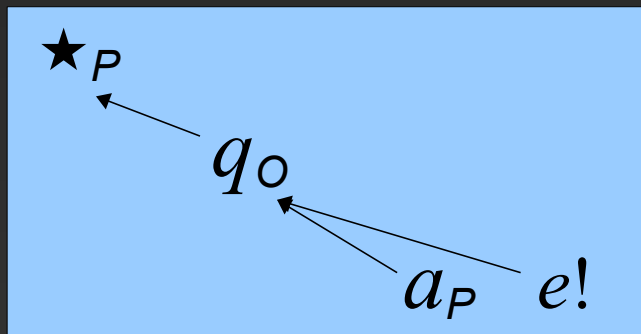
M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

$$[\![M_1]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1} \ldots$$
$$P\ O\quad P\quad O\quad P\quad O\quad P$$

$$[\![M_2]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_2!^{\Sigma_2}\ q^{\Sigma_2}\ e_3!^{\Sigma_3} \ldots$$
$$P\ O\quad P\quad O\quad P\quad O\quad P$$

$\Sigma_i = \{ e_1, \ldots, e_i \}$



Full abstraction for ExnML

Lemma. The game model is sound

Lemma. Every finitary strategy is ExnML-definable

Proof goes by factorising steps which produce strategies with increasingly less power on exceptions

Full abstraction for ExnML

Lemma. The game model is sound

Lemma. Every finitary strategy is ExnML-definable

Proof goes by factorising steps which produce strategies with increasingly less power on exceptions

Theorem. The game model is fully abstract

$$P \sqsubseteq P' \iff [\![P]\!] \subseteq [\![P']\!]$$

ExnML$: idealized exceptions

Shouldn't private exceptions always escape their context?

ExnML$: idealized exceptions

$$\Sigma, M \longrightarrow \Sigma', M'$$

Σ stores exception names and location names&values

$$\Sigma, \text{exn}() \longrightarrow \Sigma \uplus \{e\}, e \qquad \Sigma, e = e' \longrightarrow \Sigma, 0/1$$

$$\Sigma, v \text{ handle } x \Rightarrow M \longrightarrow \Sigma, v$$

$$\Sigma, (\text{raise } e) \text{ handle } x \Rightarrow M \longrightarrow \Sigma, M[e/x]$$

$$\Sigma, E_{\neg H}[\text{raise } e] \longrightarrow \Sigma, \text{raise } e$$

$$\Sigma, \text{ref}_\beta(v) \longrightarrow \Sigma \uplus \{(l, v)\}, l \qquad \Sigma, l = l' \longrightarrow \Sigma, 0/1$$

$$\Sigma, l := v \longrightarrow \Sigma[l \mapsto v], () \qquad \Sigma, !l \longrightarrow \Sigma, \Sigma(l)$$

ExnML$: idealized exceptions

$$\Sigma, M \longrightarrow \Sigma', M'$$

Σ stores exception names and location names&values

$$\Sigma, \text{exn}() \longrightarrow \Sigma \uplus \{e\}, e \qquad \Sigma, e = e' \longrightarrow \Sigma, 0/1$$

$$\Sigma, v \text{ handle } e \text{ -> } M \longrightarrow \Sigma, v$$

M handle M -> M

$$\Sigma, (\text{raise } e) \text{ handle } e \text{ -> } M \longrightarrow \Sigma, M$$

$$\Sigma, E_{\neg e}[\text{raise } e] \longrightarrow \Sigma, \text{raise } e$$

$$\Sigma, \text{ref}_\beta(v) \longrightarrow \Sigma \uplus \{(l, v)\}, l \qquad \Sigma, l = l' \longrightarrow \Sigma, 0/1$$

$$\Sigma, l := v \longrightarrow \Sigma[l \mapsto v], () \qquad \Sigma, !l \longrightarrow \Sigma, \Sigma(l)$$

Games for idealised exceptions

Strategies are restricted by *propagation conditions* for private exceptions:

- if O plays $e!$ for some fresh e then
  - P must also play $e!$
  - P cannot record this e in any way
- dually for P

Games for idealised exceptions

Strategies are restricted by *propagation conditions* for private exceptions:

- if O plays $e!$ for some fresh e then
  - P must also play $e!$
  - P cannot record this e in any way
- dually for P

These restrictions give a fully abstract model for $\mathrm{ExnML}_{\$}$

ExnML example: game semantics

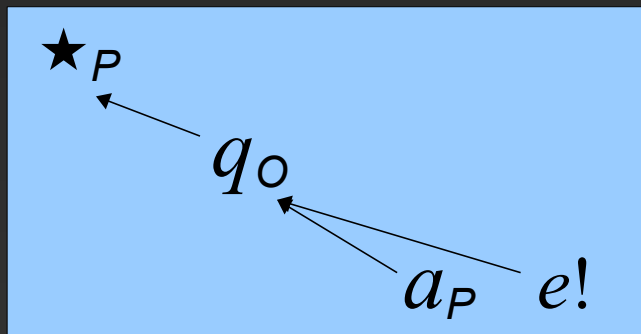
M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

$$[\![M_1]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1} \ldots$$
$$P\ O\quad P\quad O\quad P\quad O\quad P$$

$$[\![M_2]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_2!^{\Sigma_2}\ q^{\Sigma_2}\ e_3!^{\Sigma_3} \ldots$$
$$P\ O\quad P\quad O\quad P\quad O\quad P$$

$\Sigma_i = \{ e_1, \ldots, e_i \}$



ExnML example: game semantics

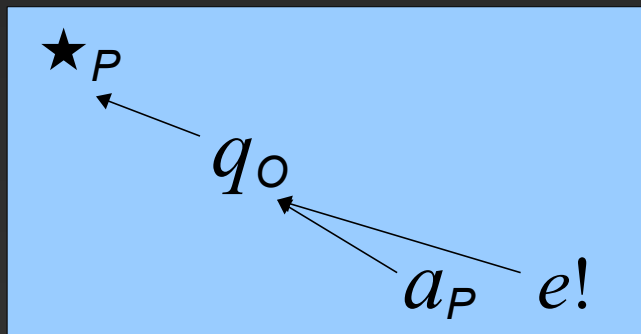
M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

$$[\![M_1]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_1!^{\Sigma_1}\ \ldots$$
$$P\ \ O\ \ P\ \ O\ \ P\ \ O\ \ P$$

$$[\![M_2]\!] = \star\ q^{\Sigma_0}\ e_1!^{\Sigma_1}\ q^{\Sigma_1}\ e_2!^{\Sigma_2}\ q^{\Sigma_2}\ e_3!^{\Sigma_3}\ \ldots$$
$$P\ \ O\ \ P\ \ O\ \ P\ \ O\ \ P$$

$\Sigma_i = \{ e_1, \ldots, e_i \}$



ExnML$ example: game semantics

M_1 : let $y = \mathsf{exn}()$ in $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}\ y$: $\mathsf{unit} \to \mathsf{unit}$

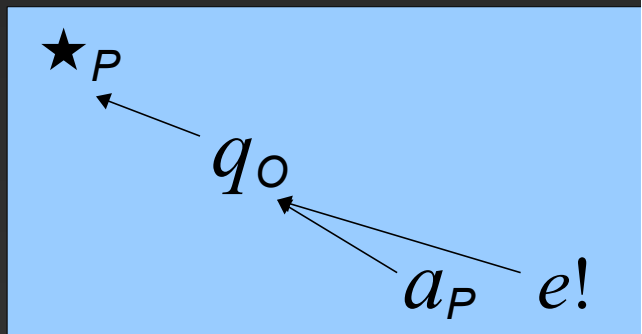
M_2 : $\lambda x^{\mathsf{unit}}.\ \mathsf{raise}(\mathsf{exn}())$: $\mathsf{unit} \to \mathsf{unit}$

$[\![M_1]\!]_{\$} = [\![M_2]\!]_{\$} = \star$

P

$\Sigma_0 = \{\}$

$M_1 \cong M_2$



Related and further work

Previous game models

- Laird's models of “bad exceptions” ('98, '01, '13)
- Quotiented nominal model in Tz'08

Future work

- Model checking ML fragments
- Algorithmic nominal games, automata & logics
- Games for Java with exceptions

Related and further work

thanks!

Previous game models

- Laird's models of “bad exceptions” ('98, '01, '13)
- Quotiented nominal model in Tz'08

Future work

- Model checking ML fragments
- Algorithmic nominal games, automata & logics
- Games for Java with exceptions