Deconstructing general references via game semantics

Andrzej Murawski
University of Warwick

Nikos Tzevelekos
Queen Mary, U. of London

HOPE 2013, Boston, Massachusetts

What this talk is about

We examine the expressivity of higher-order references in an ML-like language

We provide semantic and syntactic decomposition results which reduce the complexity of reference types

OCaml session – ground references

```
# let x = ref(3);;
val x : int ref = { contents = 3 }

# x := !x + 2010; !x;;
- : int ref = { contents = 2013 }

# let y = ref(x);;
val y : int ref ref =
          { contents = { contents = 2013 } }
```

More Ocaml – higher-order references

```
# let f = ref(fun x → 0) in
  let recall_g (g:int → int): int =
    let g_0 = !f in
      f := g;
      g_0(3);;

# recall_g(fun x → x);;
# recall_g(fun x → x+1);;
...
```

Expressivity

Divergence

```
# let f = ref(fun () → ()) in
          f := fun () → (!f)();
          (!f)();;
^CInterrupted.
```

Recursion, Objects, Aspects, ...

A language with higher-order references

- Types:

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

- Reference constructor:

$$\frac{\Gamma \vdash M : \theta}{\Gamma \vdash \text{ref}_\theta(M) : \text{ref}\,\theta}$$

- Lambda-calculus, conditionals, HO-references*

* no $==_\theta$ i.e. no $==_{\text{unit}}$

Expressivity questions

1. *Are all constructors* ref_θ *necessary?*
   (or are some of them definable, hence redundant?)

2. *When are all* $\mathrm{ref}_{\theta \to \theta'}$ *redundant?*

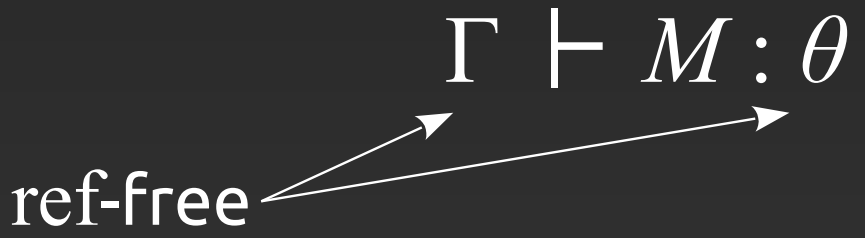
3. *When are all* ref_θ *redundant?*

Casting (the absence of)

- In general, the answers are: yes, never, never.
- References not *castable* to other types!

$$\Gamma\ ,\ x : \mathrm{ref}\,\theta' \to \mathrm{int} \ \vdash\ M : \mathrm{ref}\,\theta$$

- But we can still examine the case:



Expressivity questions, with answers

1. Are all constructors ref_θ *necessary?*

- No, just one $\mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}$ and $\mathrm{ref}_{\mathrm{int}}$ suffice

2. When are all $\mathrm{ref}_{\theta\to\theta'}$ *redundant?*

- We have a full type-theoretic characterisation

3. When are all ref_θ *redundant?*

- We have a full type-theoretic characterisation

First question

Two lines of attack:

- A semantic one: using game semantics
- A syntactic one: using explicit transformations

The syntactic attack: use bad variables

View references as read/write pairs:

$$x : \mathrm{ref}\ \theta \quad \longmapsto \quad \begin{array}{l} read \equiv \lambda y^{\mathrm{unit}}.\ !x : \mathrm{unit} \to \theta \\ write \equiv \lambda y^{\theta}.\ x := y : \theta \to \mathrm{unit} \end{array}$$

generalise to:

$$\frac{\Gamma \vdash M : \mathrm{unit} \to \theta \qquad \Gamma \vdash N : \theta \to \mathrm{unit}}{\Gamma \vdash \mathrm{mkvar}\ (M, N) : \mathrm{ref}\ \theta}$$

- $!\ \mathrm{mkvar}\ (V, V') \ \to\ V\ ()$
- $\mathrm{mkvar}\ (V, V') := U \ \to\ V'\ U$

Breaking HO references

$$\mathrm{ref}\,(\theta \to \theta') \quad \mapsto \quad \mathrm{ref}\,(\mathrm{unit} \to \mathrm{unit}) + \mathrm{ref}\,\theta + \mathrm{ref}\,\theta'$$

Breaking HO references

$$\text{ref}\,(\theta \to \theta') \;\;\mapsto\;\; \text{ref}\,(\text{unit} \to \text{unit}) + \text{ref}\,\theta + \text{ref}\,\theta'$$

$$\begin{aligned}\text{new}_{\theta \to \theta'} \;\cong\; & \text{let} \;\; in, re, ap = \text{new}_{\theta}, \text{new}_{\theta'}, \text{new}_{\text{unit}\to\text{unit}} \\ & \text{in} \;\; \text{mkvar}\,(M_r, M_w)\end{aligned}$$

$$\text{new}_{\theta} \equiv \;\text{ref}_{\theta}\,(\text{<default val>})$$

Breaking HO references

$$\text{ref}\,(\theta \to \theta') \;\;\mapsto\;\; \text{ref}\,(\text{unit} \to \text{unit}) + \text{ref}\,\theta + \text{ref}\,\theta'$$

$$\text{new}_{\theta \to \theta'} \cong \text{let}\;\; in, re, ap = \text{new}_{\theta}, \text{new}_{\theta'}, \text{new}_{\text{unit}\to\text{unit}} \;\; \text{in}\;\; \text{mkvar}\,(M_r, M_w)$$

- $M_r \equiv \lambda y^{\text{unit}}.\ \text{let}\;\; h = !ap \;\; \text{in} \;\; \lambda z^{\theta}.\ in := z;\ h();\ !re$
- $M_w \equiv \lambda y^{\theta \to \theta'}.\ ap := (\ \lambda z^{\text{unit}}.\ re := y(!in)\ )$

$\text{new}_{\theta} \equiv \text{ref}_{\theta}\,(\langle\text{default val}\rangle)$

Breaking ground references

$$\text{ref}\,(\text{ref}\,\theta) \quad \mapsto \quad \text{ref}\,(\text{unit} \to \theta) + \text{ref}\,(\theta \to \text{unit})$$

Breaking ground references

$$\text{ref}\,(\text{ref}\,\theta) \;\;\mapsto\;\; \text{ref}\,(\text{unit} \to \theta) + \text{ref}\,(\theta \to \text{unit})$$

$$\begin{aligned}\text{new}_{\text{ref}\,\theta} \;\cong\; & \text{let}\;\; r, w = \text{new}_{\text{unit}\to\theta},\; \text{new}_{\theta\to\text{unit}} \\ & \text{in}\;\; \text{mkvar}\,(M_r, M_w)\end{aligned}$$

- $M_r \equiv\; \lambda y^{\text{unit}}.\; \text{mkvar}\,(!r,\, !w)$
- $M_w \equiv\; \lambda y^{\text{ref}\,\theta}.\; r := (\lambda z^{\text{unit}}.\; !y);\; w := (\lambda z^{\theta}.\; y := z)$

Breaking ground references

$$\text{ref}\,(\text{ref}\,\theta) \;\;\mapsto\;\; \text{ref}\,(\text{unit}\to\theta) + \text{ref}\,(\theta\to\text{unit})$$

$$\text{new}_{\text{ref}\,\theta} \;\cong\; \text{let}\;\; r, w = \text{new}_{\text{unit}\to\theta},\; \text{new}_{\theta\to\text{unit}} \;\; \text{in}\;\; \text{mkvar}\,(M_r, M_w)$$

- $M_r \equiv \lambda y^{\text{unit}}.\ \text{mkvar}\,(!r,\ !w)$
- $M_w \equiv \lambda y^{\text{ref}\,\theta}.\ r := (\lambda z^{\text{unit}}.\ !y);\ w := (\lambda z^{\theta}.\ y := z)$

$$\text{new}_{\text{unit}} \;\cong\; \text{mkvar}\,(\lambda z^{\text{unit}}.(),\ \lambda z^{\text{unit}}.())$$

Syntactic decomposition

- All ref constructors can be replaced by (several) constructors of the form $\mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}$ and $\mathrm{ref}_{\mathrm{int}}$
- We can group all of them into one of each

For any term M we construct a term M' such that:

- *M' does not contain any ref_θ*
- *M' may contain mkvar*
- $M \cong \mathrm{let}\ g, x = \mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}, \mathrm{ref}_{\mathrm{int}}\ \mathrm{in}\ M'$

Removing bad variables

- We can perform a reduction to canonical form which eliminates mkvar by the rules:

$$!\,\mathrm{mkvar}\,(\lambda x.M, \lambda y.N) \quad \cong \quad M$$
$$\mathrm{mkvar}\,(\lambda x.M, \lambda y.N) := K \quad \cong \quad \mathrm{let}\; y = K \;\mathrm{in}\; N$$

For any $\Gamma \vdash M : \theta$ *with* Γ, θ ref-*free we can construct a term* M' *such that:*

- M' *does not contain any* ref_θ
- $M \quad \cong \quad \mathrm{let}\; g, x = \mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}, \mathrm{ref}_{\mathrm{int}} \;\mathrm{in}\; M'$

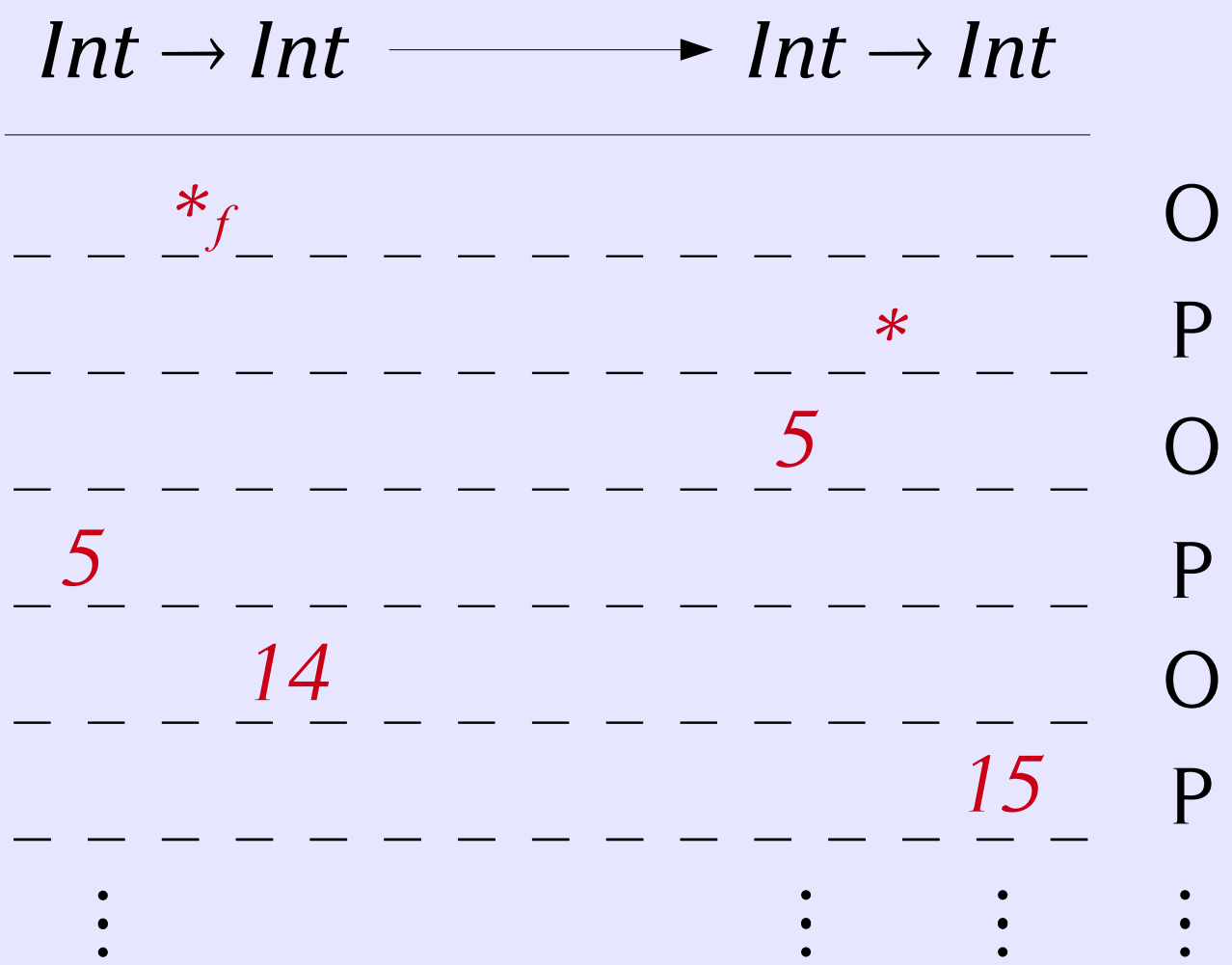
The semantic attack

Game Semantics:

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

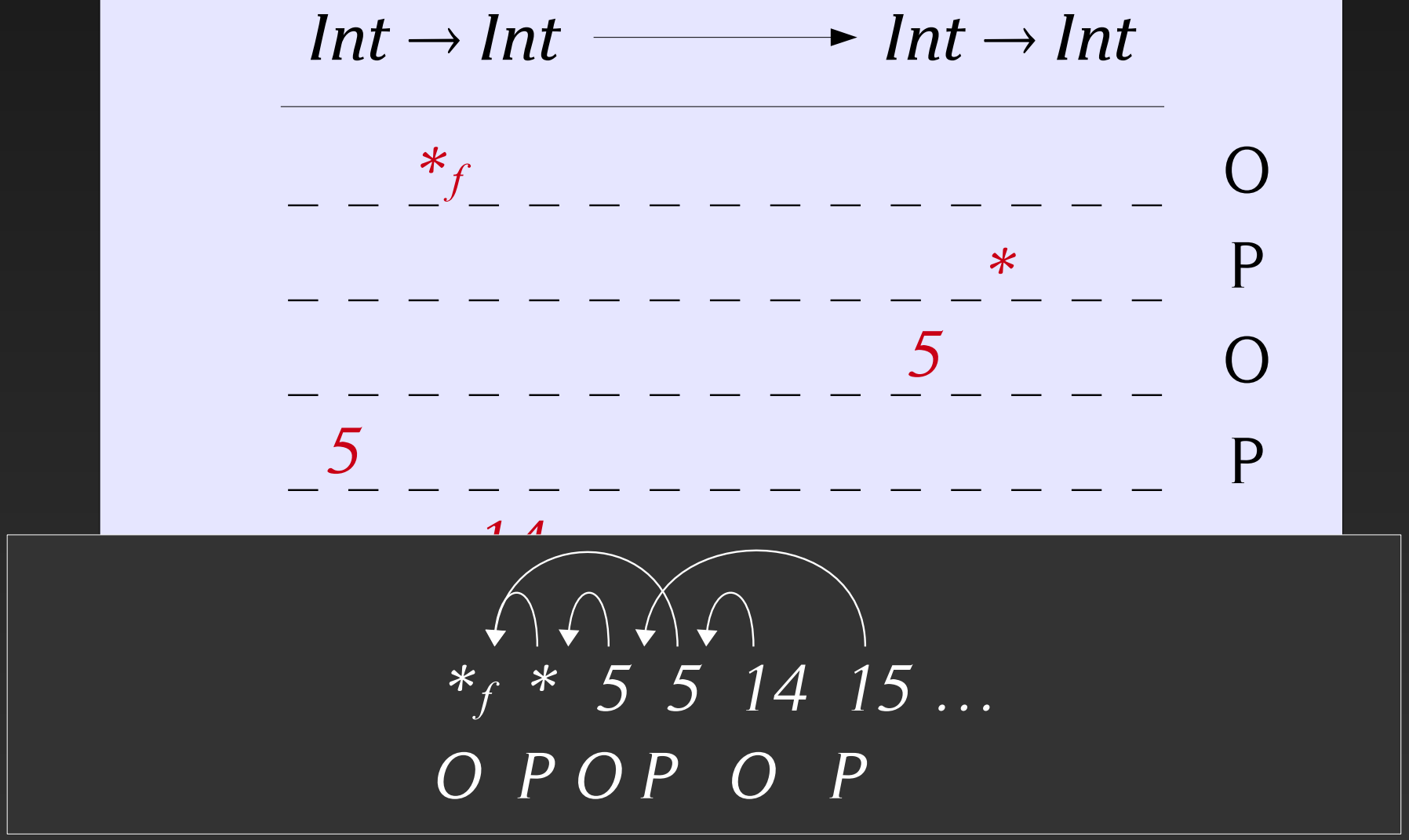
Example strategy

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



Example strategy

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



References in game semantics

LICS'98

A fully abstract game semantics for general references

Samson Abramsky Kohei Honda Guy McCusker*
LFCS, University of Edinburgh St John's College, Oxford

A games model of a p
order store in the style
category used for the m
behavioural conditions
used to provide fully ab
guages. The model is sh
factorization argument
ability for the languag
its purely functional fra

1 Introductio

Over the past few year
give the first syntax-in
stract models for a num
cluding the prototypica

ICALP'97

Game Theoretic Analysis of Call-by-Value Computation

KOHEI HONDA NOBUKO YOSHIDA

ABSTRACT. We present a general semantic universe of call-by-value computation based on elements of game semantics, and validate its appropriateness as a semantic universe by the full abstraction result for call-by-value PCF, a generic typed programming language with call-by-value evaluation. The key idea is to consider the distinction between call-by-name and call-by-value as that of the structure of information flow, which determines the basic form of games. In this way call-by-name computation and call-by-value computation arise as two independent instances of sequential functional computation with distinct algebraic structures. We elucidate the type structures of the universe following the standard categorical framework developed in the context of domain theory. Mutual relationship between the presented category of games and the corresponding call-by-name universe is also clarified.

1. INTRODUCTION

The *call-by-value* is a mode of calling procedures widely used in imperative and functional

References in game semantics

Syntactic constructs	*Semantic conditions*
HO references ($\mathrm{ref}_{\theta \to \theta'}$)	Visibility ~ no play out of lexical environment

Play out of lexical environment

```
# let f = ref(fun x → 0) in
  let recall_g (g:int → int): int =
    let g_0 = !f in
      f := g;
      g_0(3);;

# recall_g(fun x → x);;
# recall_g(fun x → x+1);;
...
```

References in game semantics

Syntactic constructs	*Semantic conditions*
HO references ($\mathrm{ref}_{\theta \to \theta'}$)	Visibility ~ no play out of lexical environment
Integer references ($\mathrm{ref}_{\mathrm{int}}$)	Innocence ~ no view out of lexical environment

Visible factorisation

For any strategy σ, there exists a strategy σ' such that:

- *σ' satisfies visibility*
- $\sigma = |[\ \mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}\ ]| ; \sigma'$

Semantic decomposition

- Similarly: visible strat. = $\mathrm{ref}_{\mathrm{int}}$ + innocent strat.
- Each (recursive) innocent strategy can be defined by a pure term

For any $\Gamma \vdash M : \theta$ *with* Γ, θ ref-*free there is a term* M' *such that:*

- M' *does not contain any* ref_{θ}
- $M \cong \mathrm{let}\ g, x = \mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}, \mathrm{ref}_{\mathrm{int}}\ \mathrm{in}\ M'$

Types not needing HO references

By restricting types, we can ban the possibility of breaking visibility!

$$\text{int} \to \ldots \to \text{int}, \ldots \vdash \text{int}$$

$$(\text{int} \to \ldots \to \text{int}) \to \text{int}, \ldots \vdash \text{int} \to \ldots \to \text{int}$$

- $\text{ref}_{\theta \to \theta'}$ *is redundant exactly in types of this form*

Types not needing any references

Similarly, in some types we can enforce innocence:

$$\mathrm{int} \to \ldots \to \mathrm{int}, \ldots \vdash \mathrm{int}$$

- ref_θ *is redundant exactly in types of this form*

Conclusions

Types permitting, higher-order store boils down to using one $\mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}$ and one $\mathrm{ref}_{\mathrm{int}}$

(HO) references add expressivity at specific types

- Can we decompose further? (e.g. discard $\mathrm{ref}_{\mathrm{int}}$)
- What about $==_{\mathrm{unit}}$?
- General (nominal) versions of decompositions:
  - It's complicated! (cf. *Towards Nominal Abramsky*)

Conclusions

thank you!

Types permitting, higher-order store boils down to using one $\mathrm{ref}_{\mathrm{unit}\to\mathrm{unit}}$ and one $\mathrm{ref}_{\mathrm{int}}$

(HO) references add expressivity at specific types

- Can we decompose further? (e.g. discard $\mathrm{ref}_{\mathrm{int}}$)
- What about $=\!=_{\mathrm{unit}}$?
- General (nominal) versions of decompositions:
  - It's complicated! (cf. *Towards Nominal Abramsky*)

No visibility

- Games break lexical environments:

$$\text{let } f = \text{ref}_{\text{int}\to\text{int}}(\lambda y.0) \text{ in}$$
$$\lambda g.\ \text{let } g_0 = f \text{ in } f := g;\ g_0(3) : (\text{int} \to \text{int}) \to \text{int}$$