Deconstructing general references via game semantics

Andrzej Murawski
University of Warwick

Nikos Tzevelekos
Queen Mary, U. of London

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

ML-like references

Creation

$$\texttt{if } M : \theta \texttt{ then ref}(M) : \texttt{ref } \theta$$

Assignment

$$\texttt{if } M : \texttt{ref } \theta, N : \theta \texttt{ then } M := N : \texttt{unit}$$

Dereferencing

$$\texttt{if } M : \texttt{ref } \theta \texttt{ then } !M : \theta$$

Equality check

$$\texttt{if } M, N : \texttt{ref } \theta \texttt{ then } M == N : \texttt{bool}$$

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':int ref) → x == x');;
val f : (int ref → bool) ref =
                          { contents = <fun> }

# (!f)(x);;
- : bool = true

# (!f)(ref(2013));;
- : bool = false
```

Expressivity

Divergence

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

Recursion, Objects, Aspects, ...

More on expressivity: beyond lexical env.

```
# 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);;
...
```

A language with higher-order references

- Types:

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

- Reference constructor:

$$\text{if } \texttt{M} : \theta \text{ then } \texttt{ref}_\theta(\texttt{M}) : \texttt{ref}\ \theta$$

- Lambda-calculus, conditionals, HO-references*.

Expressivity questions

1. *Are all constructors* $\texttt{ref}_\theta$ *necessary?*
   (or are some of them definable, hence redundant?)

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

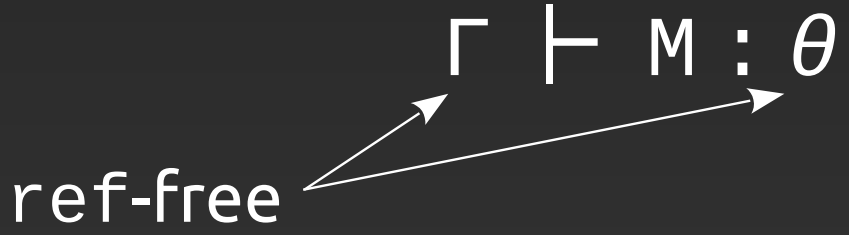
3. *When are all* $\texttt{ref}_\theta$ *redundant?*

Casting (the absence of)

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

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

- But we can still examine the case:



Expressivity questions, with answers

1. *Are all constructors* $\texttt{ref}_\theta$ *necessary?*

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

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

   - We have a full type-theoretic characterisation.

3. *When are all* $\texttt{ref}_\theta$ *redundant?*

   - We have a full type-theoretic characterisation.

First question

Two lines of attack:

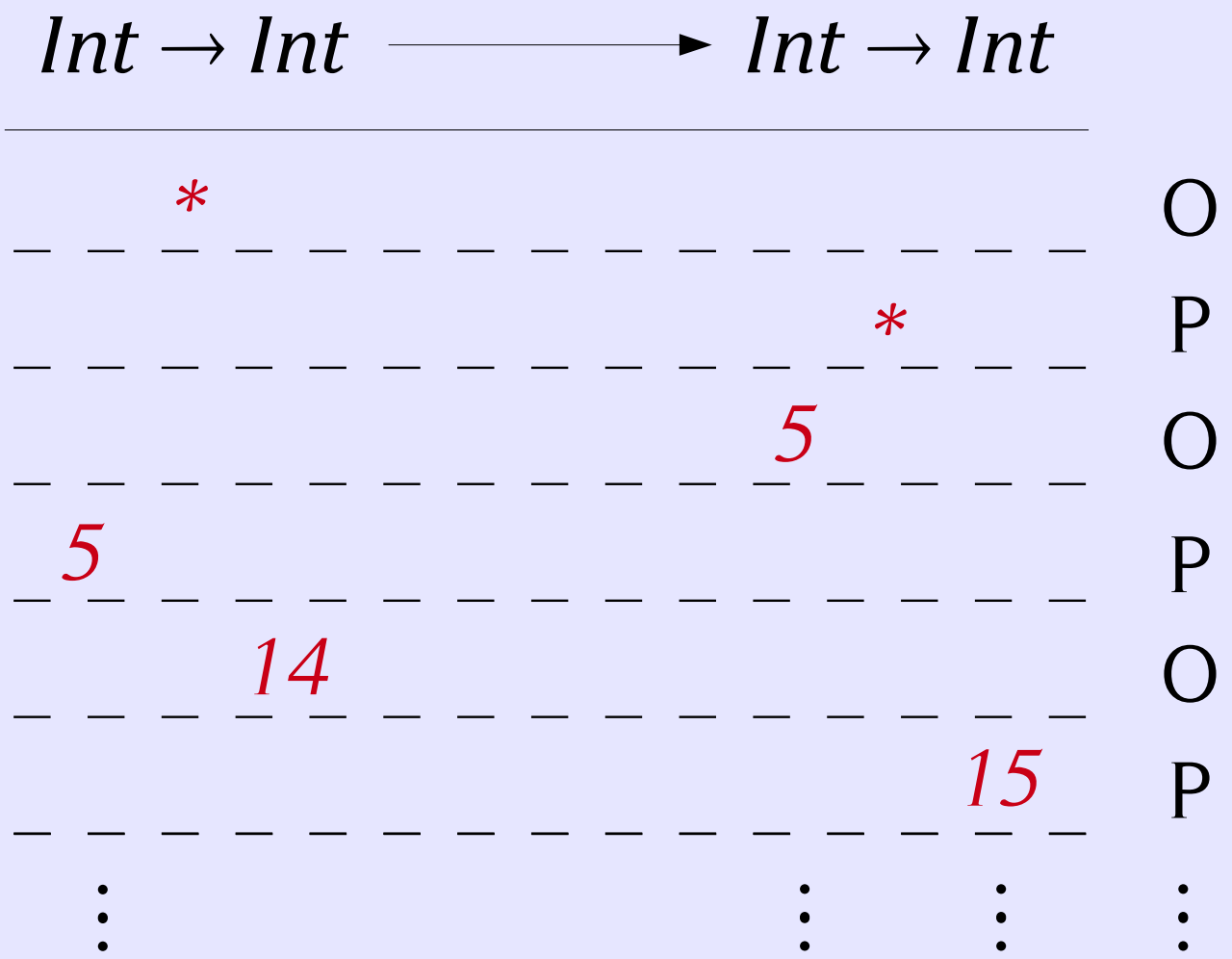
- A semantic one: using game semantics
- A syntactic one: using explicit transformations

Game Semantics

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

Example strategy

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



Example strategy

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

Games for HO-references

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

Abstract

A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The model is shown to be fully abstract by means of factorization arguments which reduce the question of definability for the language with higher-order store to that for its purely functional fragment.

1 Introduction

Over the past few years, game semantics has been used to give the first syntax-independent constructions of fully abstract models for a number of programming languages, including the prototypical functional language PCF [1, 8, 20],

created object to another object.

The key idea behind our model is to represent a reference by a certain form of *information flow*: a reference is modelled not as a static entity, but as a dynamic behaviour which mediates the flow of information between readers and writers, connecting them in an appropriate fashion. In the presence of higher-order references, these connections have to be made dynamically, and the computations of the multiple readers and writers may be interleaved in arbitrarily complex ways. The technical apparatus of game semantics provides exactly the right setting in which to formalize this idea. The "dynamic connections" used to interpret references are modelled concretely by *copy-cat behaviour*, a ubiquitous notion in game semantics; and the categorical structure provided by games allows this modelling of references to be integrated smoothly and compositionally with the interpretation of the other program phrases. This dynamic representation of references is quite different in character to the traditional location-based models

Games for HO-references

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

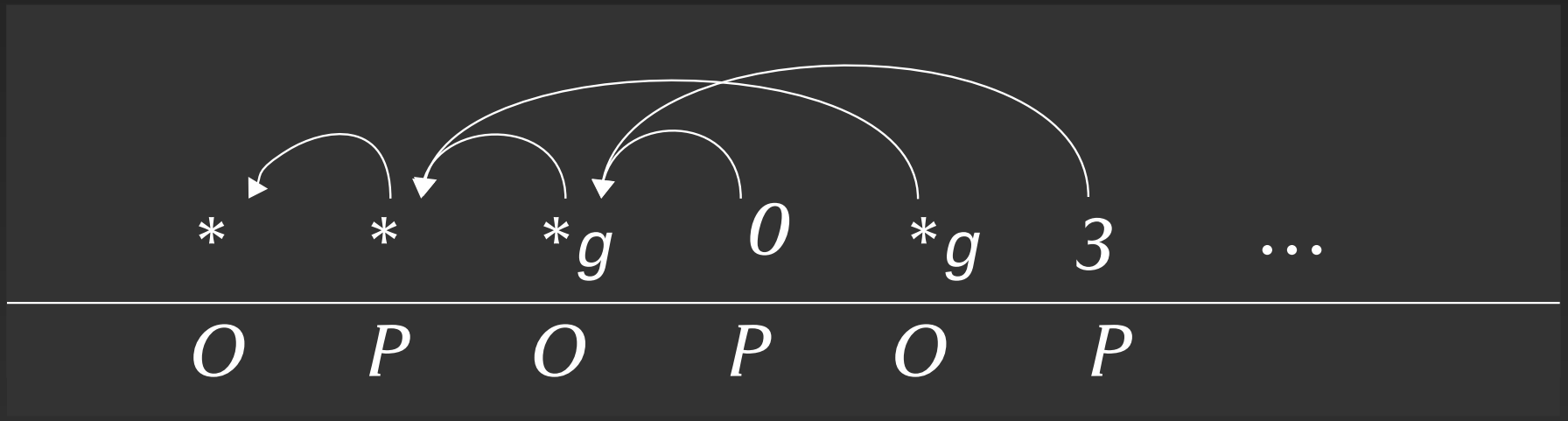
The *call-by-value* is a mode of calling procedures widely used in imperative and functional

No visibility

- Games break lexical environments:

$\texttt{let f = ref}_{\texttt{int}\to\texttt{int}}\texttt{(}\lambda\texttt{y.0) in}$

$\texttt{ }\lambda\texttt{g. let g}_0\texttt{ = f in f:=g; g}_0\texttt{(3) : (int}\to\texttt{int)}\to\texttt{int}$



Visible factorisation

- *For any strategy σ, there exists a strategy σ' such that:*
  - *σ' satisfies visibility*
  - $\sigma = [\texttt{ref}_{\texttt{unit}\to\texttt{unit}}(\ldots)];\sigma'$

Semantic decomposition

- *For any recursive visible strategy σ, there exists a term* `M` *such that:*
  - `M` *does not contain any* $\texttt{ref}_{\theta\to\theta'}$
  - $\sigma = [\![\texttt{M}]\!]$
- *For any term* `M` *there exists a term* `M'` *such that:*
  - `M'` *does not contain any* $\texttt{ref}_{\theta\to\theta'}$
  - $\texttt{M} \cong \texttt{let x = ref}_{\texttt{unit}\to\texttt{unit}}\texttt{(…) in M'}$

Syntactic decomposition

- We use a *bad* construct:
  - if `M : unit → θ, N : θ → unit`

    `then mkvar(M,N) : ref θ`

- which can be eliminated by reduction to normal form (for the examined types):
  - `! mkvar (λx.M, λy.N) ≅ M`
  - `mkvar (λx.M, λy.N) := Q  ≅  let y=Q in N`

Hacking references

$\mathtt{ref}_{\theta\to\theta'} \cong \mathtt{let\ x,x',f = ref}_{\theta}\mathtt{,ref}_{\theta'}\mathtt{,ref}_{\mathtt{unit}\to\mathtt{unit}}$
$\mathtt{in\ mkvar}(M_r, M_w)$

- $M_r \equiv \lambda y^{\mathtt{unit}}.\mathtt{let\ h =!f\ in}$
  $\lambda z^{\theta}.\ \mathtt{x := z; h(); !x'}$
- $M_w \equiv \lambda g^{\theta\to\theta'}.\ \mathtt{f := (}\lambda z^{\mathtt{unit}}.\ \mathtt{x' := g(!x))}$

Syntactic decomposition

- $\texttt{ref}_{\theta\to\theta'} \cong \texttt{let x,x',f = ref}_{\theta},\texttt{ref}_{\theta'},\texttt{ref}_{\texttt{unit}\to\texttt{unit}}$
  $\texttt{in mkvar}(M_r, M_w)$
- A similar decomposition holds for $\texttt{ref}_{\texttt{ref}\theta}$
- We can group all $\texttt{ref}_{\texttt{unit}\to\texttt{unit}}$ in one.

- *For any term* `M` *we construct a term* `M'` *such that:*
  - `M'` *does not contain any* $\texttt{ref}_{\theta\to\theta'}$
  - $\texttt{M} \cong \texttt{let x = ref}_{\texttt{unit}\to\texttt{unit}}\texttt{(…) in M'}$

Types not needing HO references

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

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

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

$\texttt{ref}_{\theta \to \theta'}$ *is redundant in exactly the above types.*

Types not needing any references

Similarly, in some types we can enforce *innocence*.

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

$\texttt{ref}_\theta$ *is redundant in exactly the above types.*

Conclusions

All HO-store effects boil down to using one $\texttt{ref}_{\texttt{unit}\to\texttt{unit}}$ *and one* $\texttt{ref}_{\texttt{int}}$

Conclusions

All HO-store effects boil down to using one $\texttt{ref}_{\texttt{unit}\to\texttt{unit}}$ *and one* $\texttt{ref}_{\texttt{int}}$

Further on:

- General (nominal) versions of factorisations:
  - It's complicated! (cf. *Towards Nominal Abramsky*)
- Is $\texttt{ref}_{\texttt{int}}$ really needed? (somewhat)

Conclusions

thank you!

All HO-store effects boil down to using one $\texttt{ref}_{\texttt{unit}\to\texttt{unit}}$ *and one* $\texttt{ref}_{\texttt{int}}$

Further on:

- General (nominal) versions of factorisations:
  - It's complicated! (cf. *Towards Nominal Abramsky*)
- Is $\texttt{ref}_{\texttt{int}}$ really needed? (somewhat)