Deconstructing general references via game semantics

Andrzej Murawski

University of Oxford

Nikos Tzevelekos

Queen Mary University of London

Applied game semantics

- Use game models to prove results about programming languages!
- Expressivity questions: can simpler syntax be used to achieve the same goal?
- General references: if a program stores a third-order function, is it possible to replace the reference with a simpler one, e.g. a first-order one, an integer one or none at all?
- Deconstructing general references using game semantics
- Complementary syntactic account

The relevant game model (AHM, LICS'98)

A fully abstract game semantics for general references

Samson Abramsky Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk

Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk

Abstract

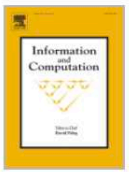
A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The 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.

The AJM paper

ELSEVIER

Information and Computation

Volume 163, Issue 2, 15 December 2000, Pages 409-470



Regular Article

Full Abstraction for PCF ☆

Samson Abramsky [a], Radha Jagadeesan [b], Pasquale Malacaria [c]

References

```
# let x=ref(0);;
val x : int ref = {contents = 0}

# x:=2023; !x;;
- : int = 2023

# let y=ref(x);;
val y : int ref ref = {contents={contents=2023}}

# !y;;
- : int ref = {contents = 2023}

# !(!y);;
- : int = 2023
```

References

```
# let y= ref( fun n -> n+1 );;
val y : (int -> int) ref = {contents = <fun>}

# (!y) 2023;;
- : int = 2024

# y:= ( fun n -> n+2 );;
- : unit = ()

# (!y) 2023;;
- : int = 2025
```

Circularity in the store

```
# y := ( fun _ -> (!y) 2023 );;
- : unit = ()

# (!y) 0;;

^CInterrupted.
```

- Divergence
- Recursion

Fixed-point combinator

```
# let fix = fun  f ->
                let t = ref(fun a -> f(!t) a) in !t;;
val fix :(('a -> 'b) -> 'a -> 'b) -> 'a -> 'b =<fun>


# let fact = fix (
       fun (f:int->int) -> fun x ->
                  if (x=0) then 1 else x*f(x-1)
       );;
val fact : int -> int = <fun>

# fact 3;;
- : int = 6
```

Support for higher-order recursion.

Beyond lexical environment

```
    ...
    y := <intermediate value/closure>
    ...

SOMEWHERE ELSE
    ...
    (!y) ...
    ...
```

- Any intermediate stage of computation can be recorded (along with local values).
- Syntactic scope does not restrict access.

Higher-order state is a very expressive primitive.
Good as intermediate language for analyzing many existing paradigms.

ML-like language with higher-order state

Types

$$\theta ::= \quad \mathsf{unit} \quad | \quad \mathsf{int} \quad | \quad \theta \to \theta \quad | \quad \mathsf{ref}(\theta)$$

Reference constructor

$$\mathsf{ref}_\theta(M)$$

Creates a fresh memory cell for storage of type θ and initialises it to M.

Expressivity problems

1. When can one replace

$$\mathsf{ref}_\theta$$

with $\mathsf{ref}_{\theta'}$ for simpler θ'?

2. When can one eliminate higher-order state, i.e.,

$$\mathsf{ref}_{\theta_1 \to \theta_2}?$$

3. When can ref_θ be replaced altogether?

In all cases we would like program behaviour to be preserved.

Solvability

In general the problem cannot be solved: reference names are typed!

$$\mathsf{ref}_{\mathsf{int}} \qquad \mathsf{ref}_{\mathsf{unit}\to\mathsf{unit}} \qquad \mathsf{ref}_{(\mathsf{int}\to\mathsf{unit})\to\mathsf{int}}$$

But it can be attacked in cases when references are used internally, i.e. they are never communicated to the environment.

Hence, we pose the problem for terms

$$x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$$

where $\theta_1, \cdots, \theta_n, \theta$ is ref-free.

Answers

1. Can we replace uses of ref_θ with $\mathsf{ref}_{\theta'}$ for some simpler θ'?

Single uses of $\mathsf{ref}_{\mathsf{int}}$ and $\mathsf{ref}_{\mathsf{unit}\to\mathsf{unit}}$ suffice!

2. When can we eliminate uses of $\mathsf{ref}_{\theta_1\to\theta_2}$?

We give a full type-theoretic characterisation.

3. When can ref_θ be replaced altogether?

We give a full type-theoretic characterisation.

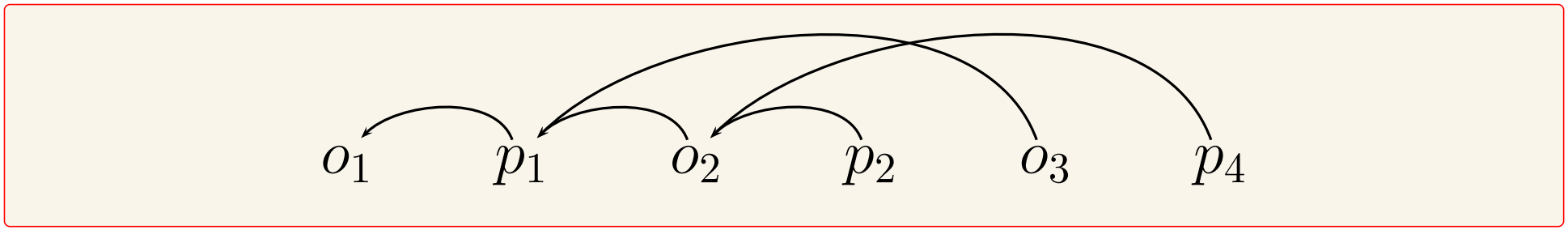
Two lines of attack

semantic: game semantics

syntactic: program transformation

Game semantics

- **Two players**: environment (O) and program (P)
- **Moves** determined by types
- Programs interpreted as **strategies**



Strategies capture interactions of a program with environments in which it can be placed.

The relevant game model (AHM, LICS'98)

A fully abstract game semantics for general references

Samson Abramsky Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk

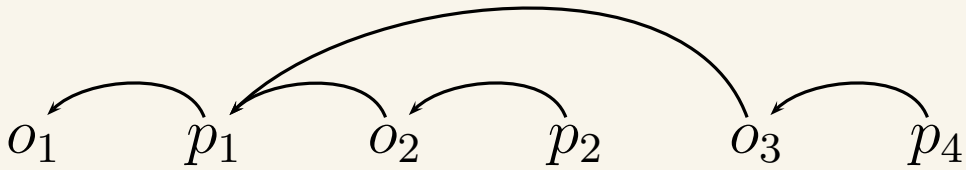
Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk

Abstract

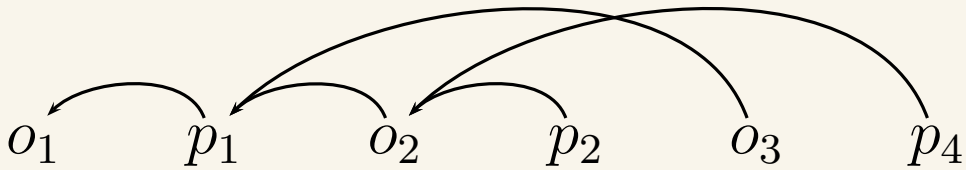
A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The 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.

First-order vs higher-order state

Visibility satisfied

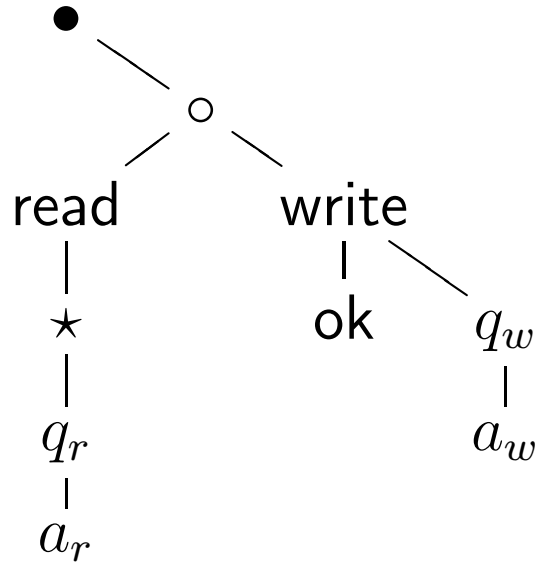


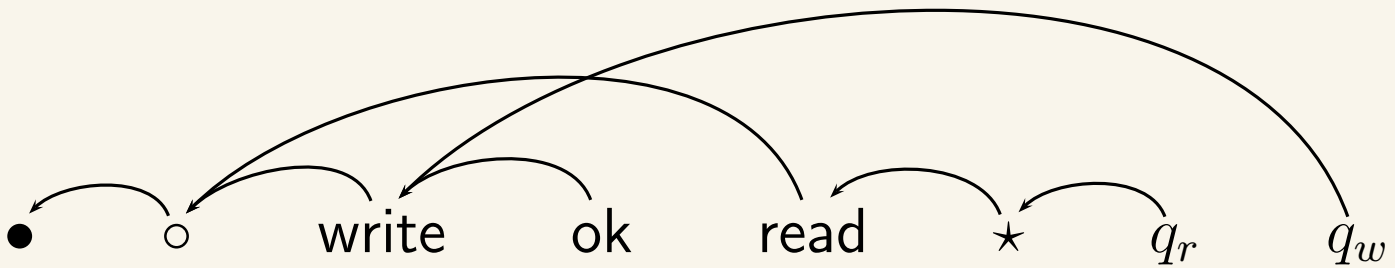
Visibility violated



Higher-order state corresponds to violations of *visibility*.

$\mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}$





Semantic solution

Strategy composition

$$\sigma_1 : A \Rightarrow B \qquad \sigma_2 : B \Rightarrow C$$

$$\sigma_1 ; \sigma_2 : A \Rightarrow C$$

$$\sigma_1 ; \sigma_2 \quad = \quad (\sigma_1 \;||_B\; \sigma_2) \;\setminus\; B$$

We will be interested in decomposition results.

Factorisation result

Theorem [Visible factorisation]

For any strategy σ, there exists a strategy σ_{visible} satisfying the visibility condition such that

$$\sigma = \mathsf{cell}_{u \to u}; \sigma_{\text{visible}}.$$

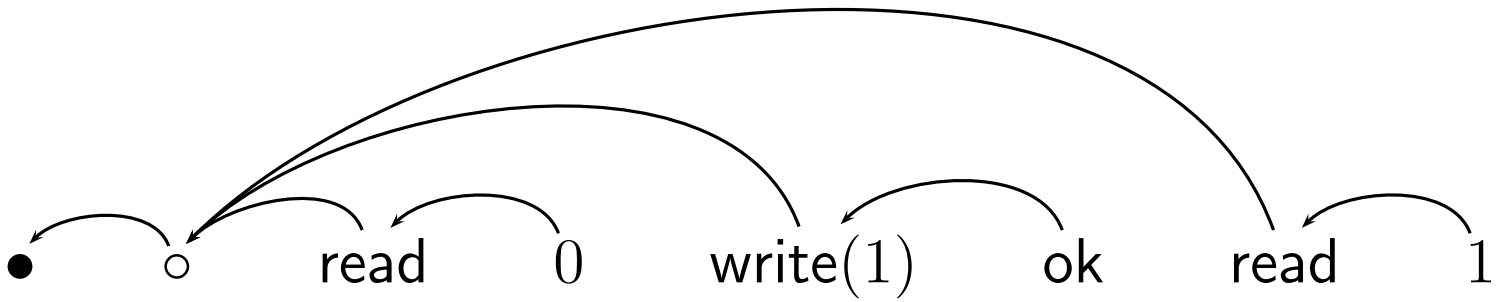
AHM proved a weaker version of the result: σ had to be finite and the argument used unboundedly many copies of $\mathsf{cell}_{u \to u}$.

Innocence [Hyland, Ong, Nickau]

Strengthening of visibility

> *In an innocent strategy P must not only point at his view, but his responses must solely depend on it.*

Typical failure: $\mathsf{cell}_{\mathsf{int}}$



An older factorisation result

Theorem [Abramsky, McCusker (CSL'97)]

For any visible strategy σ, there exists an innocent strategy $\sigma_{\mathrm{innocent}}$ such that

$$\sigma = \mathsf{cell}_{\mathsf{int}}; \sigma_{\mathrm{innocent}}.$$

Universality

Theorem [Innocent Universality]

Let $\sigma : [\![\Gamma \vdash \theta]\!]$ be a recursively presentable innocent strategy. There exists a ref-free term $\Gamma \vdash M : \theta$ such that $[\![\Gamma \vdash M : \theta]\!] = \sigma$.

Theorem [Universality]

Let $\sigma : [\![\Gamma \vdash \theta]\!]$ be a recursively presentable strategy. Then there exists $\Gamma \vdash M : \theta$ such that $[\![\Gamma \vdash M : \theta]\!] = \sigma$.

Transformation result

Theorem

Let $\Gamma \vdash M : \theta$. There exists a term

$$\Gamma, \quad f : \mathsf{ref}(\mathsf{unit} \to \mathsf{unit}), \quad x : \mathsf{ref}(\mathsf{int}) \quad \vdash \quad M' : \theta$$

satisfying the following conditions.

- M' is ref-free.
- $\Gamma \vdash M \cong \mathsf{let}\, f, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M'$.

The transformation is effective, but hardly practical!

Syntactic solution

Bad variables (Reynolds)

$$\mathsf{ref}(\theta) \equiv (\mathsf{unit} \to \theta) \times (\theta \to \mathsf{unit})$$

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

Idea: use bad variables as intermediate objects

Syntactic decomposition I

Lemma

For all θ_1, θ_2,

$$\mathsf{new}_{\theta_1 \to \theta_2} \cong \mathsf{let}\, x_1, x_2, f = \mathsf{new}_{\theta_1}, \mathsf{new}_{\theta_2}, \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}\, \mathsf{in}\, \mathsf{mkvar}\,(M_r, M_w)$$

where

$$\begin{array}{rcl} M_r & \equiv & \lambda y^{\mathsf{unit}}.\mathsf{let}\, h = !f\, \mathsf{in}\, \lambda z^{\theta_1}.\, (x_1 := z;\ h();\ !x_2), \\ M_w & \equiv & \lambda g^{\theta_1 \to \theta_2}.\ f := (\lambda z^{\mathsf{unit}}.\, x_2 := g(!x_1)). \end{array}$$

Syntactic decomposition II

Lemma

For any θ,

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

for all θ, where

$$\begin{array}{rcl} M_r & \equiv & \lambda z^{\mathsf{unit}}.\, \mathsf{mkvar}(!r, !w), \\ M_w & \equiv & \lambda g^{\mathsf{ref}(\theta)}.\, (r{:=}(\lambda z^{\mathsf{unit}}.\, !g);\ w{:=}(\lambda z^{\theta}.\, g{:=}z)). \end{array}$$

Bad variables can be eliminated

When $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$ and $\theta_1, \cdots, \theta_n, \theta$ are ref-free, bad variables can be *eliminated* from the language.

$$!\,\mathsf{mkvar}\,(\lambda u.M, \lambda v.N) \quad \cong \quad \mathsf{let}\, u = () \,\mathsf{in}\, M$$

$$\mathsf{mkvar}\,(\lambda u.M, \lambda v.N) := Q \quad \cong \quad \mathsf{let}\, v = Q \,\mathsf{in}\, N$$

Altogether occurrences of $\mathsf{ref}_{\theta_1 \to \theta_2}$ can be successively removed so that only those of $\mathsf{ref}_{\mathsf{int}}$ and $\mathsf{ref}_{\mathsf{unit} \to \mathsf{unit}}$ remain. These can subsequently be merged into single occurrences of $\mathsf{ref}_{\mathsf{int}}$ and $\mathsf{ref}_{\mathsf{unit} \to \mathsf{unit}}$.

Transformation

Theorem

Let $\Gamma \vdash M : \theta$. There exists a term

$$\Gamma, \quad f : \mathsf{ref}(\mathsf{unit} \to \mathsf{unit}), \quad x : \mathsf{ref}(\mathsf{int}) \quad \vdash \quad M' : \theta$$

satisfying the following conditions.

- M' is ref-free.
- $\Gamma \vdash M \cong \mathsf{let}\, f, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M'$.

In particular, a single $(\mathsf{unit} \to \mathsf{unit})$-valued reference cell suffices for implementing higher-order state.

When higher-order references are replaceable

Visibility distinguishes between first-order and higher-order state.

What types determine plays in which the visibility condition holds for free?

$$\begin{array}{rrcl} \cdots, & f : \mathsf{int} \to \cdots \to \mathsf{int} & , \cdots \vdash & M : \mathsf{int} \\ \cdots, & f : (\mathsf{int} \to \cdots \to \mathsf{int}) \to \mathsf{int} & , \cdots \vdash & M : \mathsf{int} \to \cdots \to \mathsf{int} \end{array}$$

If a piece of code has a type of the above shape then the same effect can be achieved without higher-order state!

When all references are replaceable

There is another technical condition called *innocence* (Hyland, Ong, Nickau) that corresponds to the absence of state.

$$\cdots, \quad f : \mathsf{int} \to \cdots \to \mathsf{int} \quad , \cdots \vdash M : \mathsf{int}$$

Programs of the above type can be written without using state (purely functional).

Conclusions

- $\mathsf{ref}_{\mathsf{unit}\to\mathsf{unit}}$ is very expressive.
- Focus on simple higher-order types will not lead to decidability.

Ideas for future work

- Consider weakened references, e.g. without cycles in the store.
- In presence of ref(bool), ref(unit $\to$ unit) can be used to simulate ref(int). What is the relationship between ref(bool) and ref(unit $\to$ unit)?