Deconstructing general references via game semantics

Andrzej Murawski

University of Warwick

Nikos Tzevelekos

Queen Mary
University of London

References

```
# let x= ref(0) in x:=2013; !x;;
- : int = 2013

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

# (!y) 2013;;
- : int = 2014

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

# (!y) 2013;;
- : int = 2015
```

Circularity in the store

```
# y := ( fun _ -> (!y) 2013 );;
- : unit = ()

# (!y) 0;;

^CInterrupted.
```

- Divergence
- Recursion

More on expressivity

- Object-oriented programming
- Aspect-oriented programming

Beyond lexical environment

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

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

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 can give a full type-theoretic characterisation.

3. When can ref_θ be replaced altogether?

We can give a full type-theoretic characterisation.

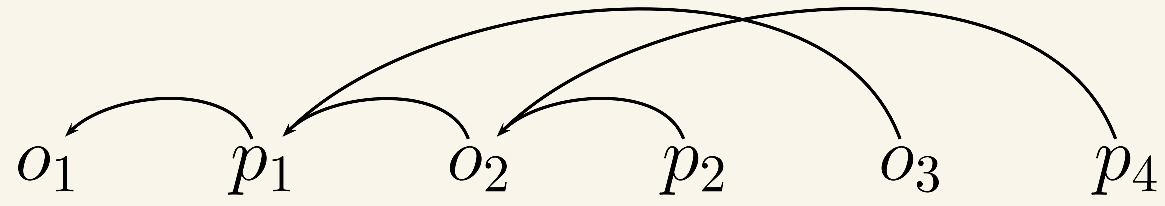
Two lines of attack

semantic: game semantics

syntactic: program transformation

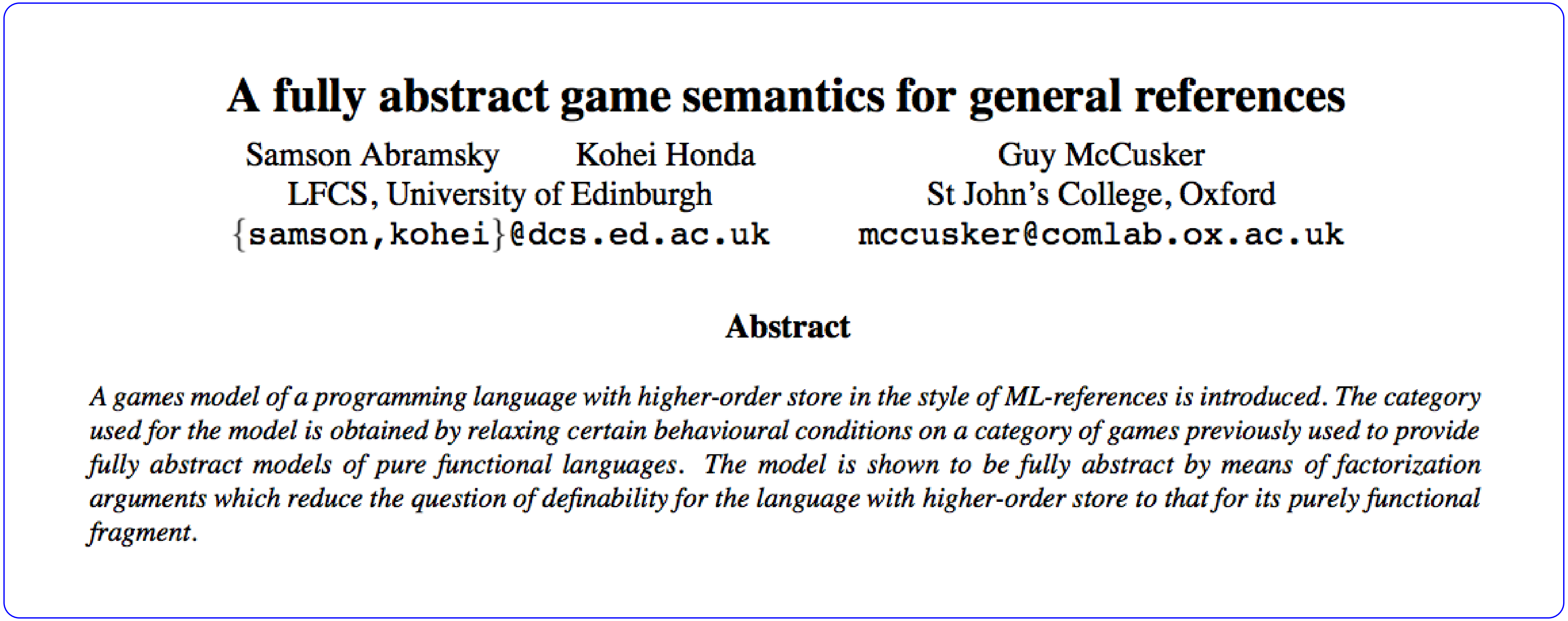
Game semantics

- **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 (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
{samson,kohei}@dcs.ed.ac.uk   mccusker@comlab.ox.ac.uk

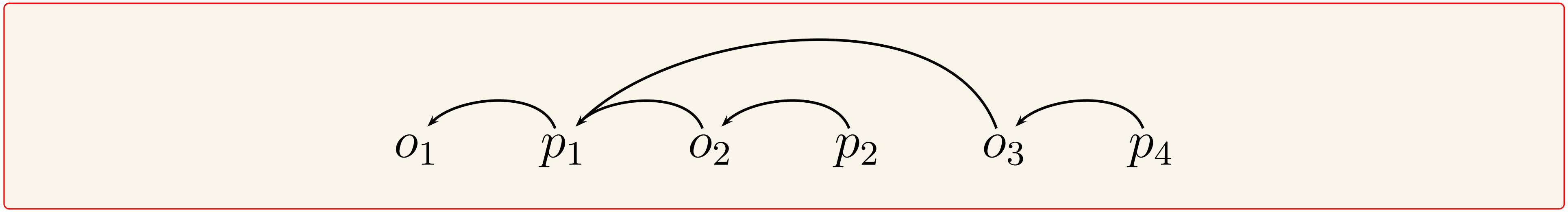
Abstract

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.

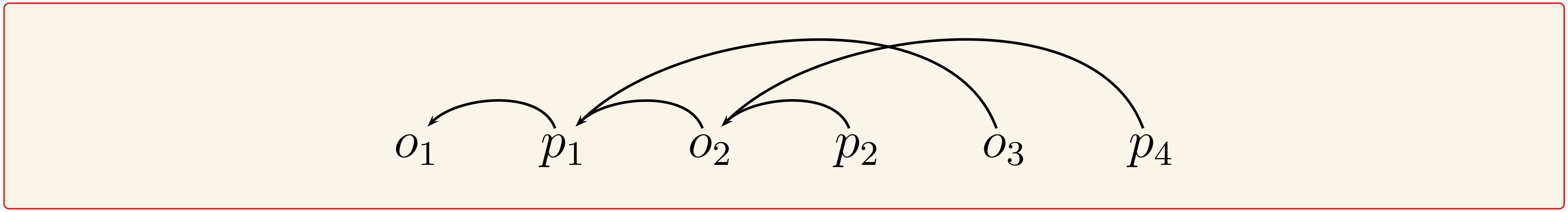
Higher-order state in game semantics

First-order state corresponds to a technical condition called *visibility*.

Visibility satisfied

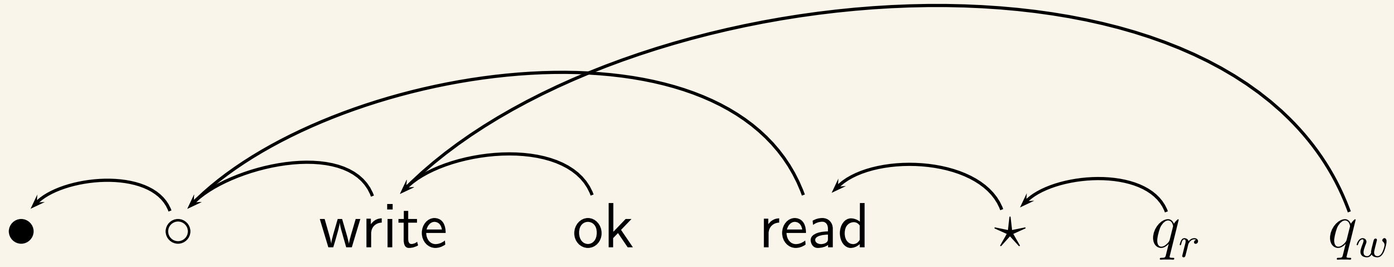


Visibility violated



cell$_{u\to u}$

Here is a play from a strategy corresponding to ref$_{\mathsf{unit}\to\mathsf{unit}}$.



● ○ write ok read $\star$ q_r q_w

Semantic solution

Strategies can be composed

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

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

Theorem

For any strategy σ, there exists a strategy $\sigma_{\mathrm{visible}}$ satisfying the visibility condition and such that

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

Syntactic solution

Idea: use bad variables as intermediate objects

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

Theorem

For all θ_1, θ_2,

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

where

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

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.

$$\begin{array}{rcl} !\,\mathsf{mkvar}\,(\lambda u.M, \lambda v.N) & \cong & \mathsf{let}\, u = ()\,\mathsf{in}\, M \\ \mathsf{mkvar}\,(\lambda u.M, \lambda v.N) {:=} Q & \cong & \mathsf{let}\, v = Q\,\mathsf{in}\, N \end{array}$$

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

A single use of $\mathsf{ref}_{\mathsf{unit} \to \mathsf{unit}}$ **suffices for 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?

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

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.
- Can anything be done in presence of reference types?