Deconstructing General References via Game Semantics

Andrzej S. Murawski[1] and Nikos Tzevelekos[2,⋆]

[1] University of Warwick
[2] Queen Mary, University of London

Abstract. We investigate the game semantics of general references through the fully abstract game model of Abramsky, Honda and McCusker (AHM), which demonstrated that the visibility condition in games corresponds to the extra expressivity afforded by higher-order references with respect to integer references.

First, we prove a stronger version of the visible factorisation result from AHM, by decomposing any strategy into a visible one and a single strategy corresponding to a reference cell of type $\mathsf{unit} \to \mathsf{unit}$ (AHM accounted only for finite strategies and its result involved unboundedly many cells).

We show that the strengthened version of the theorem implies universality of the model and, consequently, we can rely upon it to provide semantic proofs of program transformation results. In particular, one can prove that any program with general references is equivalent to a purely functional program augmented with a single $\mathsf{unit} \to \mathsf{unit}$ reference cell and a single integer cell. We also propose a syntactic method of achieving such a transformation.

Finally, we provide a type-theoretic characterisation of terms in which the use of general references can be simulated with an integer reference cell or through purely functional computation, without any changes to the underlying types.

1 Introduction

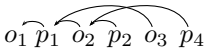
In computer science, references are a programming idiom that allows the programmer to manipulate objects in computer memory. The referenced content can be accessed (dereferenced) or overwritten (updated). The most common sort of reference is that to a ground-type value, such as an integer. However, most modern programming languages allow more complicated values to be referenced. For example, the language ML features general references, where memory locations can contain values of any type, in particular of function type. Higher-order references are a very expressive construct. Among others, they can be used to simulate recursion, objects [5] and aspects [10]. In this paper we investigate higher-order references taking inspiration from their game-semantic model [1]. In particular, we shall provide both semantic and syntactic accounts of how they can be decomposed, which highlight the fact that their full expressive power is already present in their simplest instance, references of type $\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$. Although in general the inclusion of higher-order references strictly increases expressivity, we also consider the question whether there are circumstances, delineated by types, where there is no such increase. Finally, we shall also answer the question when references

⋆ Supported by a Royal Academy of Engineering research fellowship.

F. Pfenning (Ed.): FOSSACS 2013, LNCS 7794, pp. 241–256, 2013.

in general, integer-valued and higher-order ones, can be replaced altogether by purely functional computation over the same types.

Game semantics [2,7] is a semantic theory that interprets computation as an exchange of moves between two players: O (environment) and P (program). In the Hyland-Ong style of playing [7], moves are equipped with pointers to moves made earlier in the game, giving rise to plays like the one shown below.



Existing literature on modelling integer [3] and higher-order references [1] showed that the expressive gap between the two paradigms can be captured by a property called *visibility*, which restricts the range of targets (earlier moves) for pointers: moves by P can only point at moves from a restricted fragment of the history, called the *view*. For example, the last move in the play above violates visibility, because before it is played, the view is $o_1\, p_1\, o_3$. Intuitively, the visibility constraint captures the intuition that, without higher-order references, the set of values available to a program is limited to the lexical environment (as captured by the notion of view). In particular, function values that are available at one point, cannot be taken for granted later during the course of computation. In contrast, in presence of higher-order references, such values can be recorded and reused at will. Hence, the visibility condition needs to be relaxed for modelling higher-order references.

A fully abstract game model of an ML-like language with references was presented in [1] and founded on plays that need not obey the visibility condition. As part of the full abstraction argument, the authors showed how to decompose every *finite* strategy into a finite strategy satisfying visibility and *several* strategies corresponding to reference cells of type $\mathsf{unit} \to \mathsf{unit}$ (Proposition 5 in [1]).

As our first contribution, we sharpen the result to *arbitrary* strategies as well as showing that *one* strategy corresponding to a $(\mathsf{unit} \to \mathsf{unit})$-valued memory cell is sufficient (Theorem 13). This brings two benefits. On the theoretical side, one can show universality: any recursively presentable strategy corresponds to a program with higher-order references. On a more practical note, the refined factorisation result can be applied to denotations of arbitrary programs to yield a powerful expressivity result: any program with higher-order references is equivalent to one of the shape $\mathsf{let}\ u = \mathsf{ref}(\lambda x^{\mathsf{unit}}.\Omega_{\mathsf{unit}})\ \mathsf{in}\ M$, where the ref-constructor in M is restricted to integers (Theorem 17).

Our first proof of the result is purely semantic and relies on recursion theory. Consequently, it does not offer much insight into how to transform the use of higher-order references into uses of a $(\mathsf{unit} \to \mathsf{unit})$-reference cell. Motivated by this, we try to identify semantics-preserving program transformations that will allow us to reprove the same result through syntactic means. The key element of our approach is the bad-variable constructor mkvar, which enables one to create terms of reference types with non-standard behaviour. Although the translation introduces extra occurrences of mkvar, we show how to eliminate them under certain conditions, namely, when the types associated with its free variables and the type of the term do not contain reference types. Note that this allows for arbitrarily complex private uses of general references inside terms as long as the references are not communicated through the program's type interface. From this perspective, mkvar emerges as a useful intermediate construct for

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{Z}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{(x : \theta) \in \Gamma}{\Gamma \vdash x : \theta} \qquad \frac{\Gamma \vdash M_1 : \mathsf{int} \quad \Gamma \vdash M_2 : \mathsf{int}}{\Gamma \vdash M_1 \oplus M_2 : \mathsf{int}}$$

$$\frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N_0 : \theta \quad \Gamma \vdash N_1 : \theta}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_0 : \theta} \qquad \frac{\Gamma \vdash M : \theta}{\Gamma \vdash \mathsf{ref}(M) : \mathsf{ref}(\theta)} \qquad \frac{\Gamma \vdash M : \mathsf{ref}(\theta)}{\Gamma \vdash !M : \theta}$$

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

$$\frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : \theta \to \theta' \quad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma \vdash M : (\theta \to \theta') \to (\theta \to \theta')}{\Gamma \vdash \mathsf{Y}(M) : \theta \to \theta'}$$

Fig. 1. Typing judgments of $\mathcal{L}$

program transformation. Altogether our transformations yield an alternative syntactic proof of Theorem 17.

In the remainder of the paper, we give a type-theoretic characterization of terms in which the use of arbitrary references can be faithfully simulated using integer storage alone. Here is a representative selection of types in typing judgments that turn out to guarantee this property.

$$\cdots, \mathsf{int} \to \cdots \to \mathsf{int}, \cdots \vdash M : \mathsf{int},$$
$$\cdots, (\mathsf{int} \to \cdots \to \mathsf{int}) \to \mathsf{int}, \cdots \vdash M : \mathsf{int} \to \cdots \to \mathsf{int}$$

By highlighting the shape of types in the context, we mean to say that all free identifiers should have types of that form or simpler ones. These are the typing judgments over which there is no distinction in expressive power between integer and higher-order references.

Finally, we show that, as long as terms of the form

$$\cdots, x : \Theta_1, \cdots \vdash M : \beta$$

are considered, where $\beta ::= \mathsf{unit} \mid \mathsf{int}$ and $\Theta_1 ::= \beta \mid \mathsf{ref}(\beta) \mid \beta \to \Theta_1$, the use of higher-order references can be replaced with purely functional computation. That is to say, references do not contribute any expressive power. The last two results are obtained in a semantic way, by referring to game models and associated compositionality and universality results.

2 Syntax of the Language

We shall rely on the programming language $\mathcal{L}$ with general references introduced in [1]. Its types θ are generated from unit and int using the $\to$ and ref type constructors, as shown below.

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

The typing rules are reproduced in Figure 1, where $\oplus$ is meant to cover standard arithmetic operations. The operational semantics of the language relies on a countable set L of typed locations. The values of the language are then the locations themselves, $()$, i, λ-abstractions and $\mathsf{mkvar}(V_1, V_2)$, where V_1, V_2 must be values. The big-step reduction judgments have the form $s, M \Downarrow s', V$, where s, s' are stores (partial functions from L to the set of values) and V is a value. Most reduction rules take the form

$$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2 \quad \cdots \quad M_n \Downarrow V_n}{M \Downarrow V}$$

which is meant to abbreviate

$$\frac{s_1, M_1 \Downarrow s_2, V_1 \quad s_2, M_2 \Downarrow s_3, V_2 \quad \cdots \quad s_n, M_n \Downarrow s_{n+1}, V_n}{s_1, M_1 \Downarrow s_{n+1}, V}.$$

In particular, this means that the ordering of the hypotheses is significant.

$$\frac{V \text{ is a value}}{s, V \Downarrow s, V} \qquad \frac{M \Downarrow 0 \quad N_0 \Downarrow V}{\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_0 \Downarrow V} \qquad \frac{i \neq 0 \quad M \Downarrow i \quad N_1 \Downarrow V}{\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_0 \Downarrow V}$$

$$\frac{M_1 \Downarrow i_1 \quad M_2 \Downarrow i_2}{M_1 \oplus M_2 \Downarrow i_1 \oplus i_2} \qquad \frac{M \Downarrow \lambda x.M' \quad N \Downarrow V' \quad M'[V'/x] \Downarrow V}{MN \Downarrow V}$$

$$\frac{s, M \Downarrow s', \ell \quad s'(\ell) = V}{s, !M \Downarrow s', V} \qquad \frac{s, M \Downarrow s', \ell \quad s', N \Downarrow s'', V}{s, M := N \Downarrow s''(\ell \mapsto V), ()}$$

$$\frac{M \Downarrow \mathsf{mkvar}(V_1, V_2) \quad V_1() \Downarrow V}{!M \Downarrow V} \qquad \frac{M \Downarrow \mathsf{mkvar}(V_1, V_2) \quad N \Downarrow V \quad V_2\, V \Downarrow ()}{M := N \Downarrow ()}$$

$$\frac{M \Downarrow V_1 \quad N \Downarrow V_2}{\mathsf{mkvar}(M, N) \Downarrow \mathsf{mkvar}(V_1, V_2)} \qquad \frac{s, M \Downarrow s', V \quad \ell \notin \mathsf{dom}(s')}{s, \mathsf{ref}(M) \Downarrow s' \cup (\ell \mapsto V), \ell}$$

$$\frac{M \Downarrow \lambda x.M' \quad N \Downarrow V' \quad M'[V'/x] \Downarrow V}{MN \Downarrow V} \qquad \frac{M \Downarrow V}{\mathsf{fix}(M) \Downarrow \lambda x^{\theta}.(V(\mathsf{fix}(V)))x}$$

Given a closed term $\vdash M : \theta$ we write $M \Downarrow$ if there exist s', V such that $\emptyset, M \Downarrow s', V$.

Definition 1. *We shall say that two terms $\Gamma \vdash M_1 : \theta$ and $\Gamma \vdash M_2 : \theta$ are contextually equivalent (written $\Gamma \vdash M_1 \cong M_2$) if, for any context $C[-]$ such that $C[M_1], C[M_2]$ are closed, we have $C[M_1] \Downarrow$ if and only if $C[M_2] \Downarrow$.*

Remark 2. $\mathcal{L}$ features the "bad-reference" constructor mkvar in the style of Reynolds [9]. This makes it possible to construct objects of reference types from arbitrary read and write methods. In general this strengthens the discriminating power of contexts, as terms of ref-type can exhibit non-standard behaviour. However, it can be shown that when there are no ref-types in Γ or θ, this extension is inconsequential. At the technical level, this is due to the fact that the corresponding definability argument [1] need not rely on ref then.

Remark 3. $\mathcal{L}$ does not feature reference-equality testing as a primitive, as in general it would not make sense in a setting with bad references. Still, it is possible to construct

a term that can tell two different locations apart by writing different values to them and testing their content. This is of course conditional on the existence of such values and our ability to distinguish them. In our setting, this method will be applicable to all types $\mathsf{ref}(\theta)$ except when $\theta \equiv \mathsf{unit}$.

Remark 4. In earlier work, we considered a language called RefML [8] with general references and equality testing for locations, in which bad references could not be created. The above comments imply that the respective notions of contextual equivalence induced by $\mathcal{L}$ and RefML coincide on mkvar-free $\mathcal{L}$-terms $\Gamma \vdash M : \theta$ such that there are no ref-types in Γ or θ. Similarly, one can also say that they converge for RefML-terms $\Gamma \vdash M : \theta$ such that Γ, θ do not contain ref-types and M does not use equality testing for references of type $\mathsf{ref}(\mathsf{unit})$.

We will now define a number of auxiliary terms that will turn out useful in subsequent arguments. As usual, $\mathsf{let}\, x = M \,\mathsf{in}\, N$ stands for $(\lambda x.N)M$. If x does not occur in N, we may also write $M; N$. We also rely on abbreviated notation for nested let's, e.g. $\mathsf{let}\, x, y = M_x, M_y \,\mathsf{in}\, N$ stands for $\mathsf{let}\, x = M_x \,\mathsf{in}\, \mathsf{let}\, y = M_y \,\mathsf{in}\, N$. We shall write Ω_θ for the divergent term $\mathsf{Y}(\lambda f^{\mathsf{unit}\to\theta}.f)\,()$. Also, for any type θ, we define a term $\vdash \mathsf{new}_\theta : \mathsf{ref}(\theta)$, which creates a suitably initialised reference cell.

$$\mathsf{new}_{\mathsf{unit}} \equiv \mathsf{ref}(()) \qquad \mathsf{new}_{\mathsf{ref}(\theta)} \equiv \mathsf{ref}(\mathsf{mkvar}(\lambda x^{\mathsf{unit}}.\, \Omega_\theta, \lambda x^{\theta}.\, \Omega_{\mathsf{unit}}))$$
$$\mathsf{new}_{\mathsf{int}} \equiv \mathsf{ref}(0) \qquad \mathsf{new}_{\theta\to\theta'} \equiv \mathsf{ref}(\lambda x^{\theta}.\Omega_{\theta'})$$

3 Game Model

The following arguments are couched in the game model of general references due to Abramsky, Honda and McCusker [1]. We use a more direct, yet equivalent, presentation due to Honda and Yoshida [6].

Definition 5. *An* **arena** $A = (M_A, I_A, \vdash_A, \lambda_A)$ *is given by*

- *a set M_A of moves, and a subset $I_A \subseteq M_A$ of* initial *moves,*
- *a justification relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$, and*
- *a labelling function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$*

such that $\lambda_A(I_A) = \{PA\}$. Additionally, whenever $m' \vdash_A m$, we have $(\pi_1\lambda_A)(m) \neq (\pi_1\lambda_A)(m')$, and $(\pi_2\lambda_A)(m') = A$ implies $(\pi_2\lambda_A)(m) = Q$.

The role of λ_A is to label moves as *Opponent* or *Proponent* moves and as *Questions* or *Answers*. We typically write them as $m, n, \ldots$, or $o, p, q, a, q_P, q_O, \ldots$ when we want to be specific about their kind. The simplest arena is $0 = (\emptyset, \emptyset, \emptyset, \emptyset)$. Other "flat" arenas are 1 and $\mathbb{Z}$, defined by $M_1 = I_1 = \{\star\}$, $M_{\mathbb{Z}} = I_{\mathbb{Z}} = \mathbb{Z}$. The two standard constructions on arenas are presented below, where $\bar{I}_A$ stands for $M_A \setminus I_A$, the domain restriction of a function is denoted by $\restriction$, the OP-complement of λ_A is written as $\bar{\lambda}_A$, and i_A, i_B range over initial moves in the respective arenas.

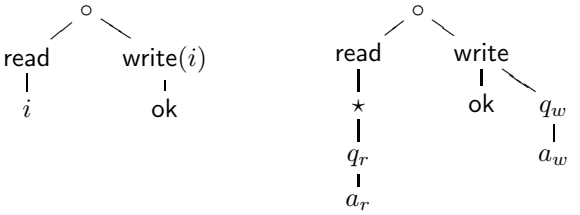
- $M_{A\Rightarrow B} = \{\star\} \uplus I_A \uplus \overline{I}_A \uplus M_B$, $I_{A\Rightarrow B} = \{\star\}$, $\lambda_{A\Rightarrow B} = [(\star, PA), (i_A, OQ), \bar{\lambda}_A \restriction \overline{I}_A, \lambda_B]$, $\vdash_{A\Rightarrow B} = \{(\star, i_A), (i_A, i_B)\} \cup \vdash_A \cup \vdash_B$.

– $M_{A\otimes B} = (I_A \times I_B) \uplus \overline{I}_A \uplus \overline{I}_B$, $I_{A\otimes B} = I_A \times I_B$, $\lambda_{A\otimes B} = [((i_A, i_B), PA), \lambda_A \restriction \overline{I}_A, \lambda_B \restriction \overline{I}_B]$, $\vdash_{A\otimes B} = \{((i_A, i_B), m) \mid i_A \vdash_A m \vee i_B \vdash_B m\} \cup (\vdash_A \restriction \overline{I}_A{}^2) \cup (\vdash_B \restriction \overline{I}_B{}^2)$.

Types of $\mathcal{L}$ can now be interpreted with arenas in the following way.

$$\begin{array}{rr} [\![\mathsf{unit}]\!] = 1 & [\![\mathsf{ref}(\theta)]\!] = (1 \Rightarrow [\![\theta]\!]) \otimes ([\![\theta]\!] \Rightarrow 1) \\ [\![\mathsf{int}]\!] = \mathbb{Z} & [\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!] \end{array}$$

Example 6. $[\![\mathsf{ref}(\mathsf{int})]\!]$ and $[\![\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})]\!]$ have the following respective shapes.



Although arenas model types, the actual games will be played in ***prearenas***, which are defined in the same way as arenas with the exception that initial moves must be O-questions. Given arenas A and B, we can construct the prearena $A \to B$ by setting: $M_{A\to B} = M_A \uplus M_B$, $I_{A\to B} = I_A$, $\lambda_{A\to B} = [(i_A, OQ) \cup (\bar{\lambda}_A \restriction \overline{I}_A)\ ,\ \lambda_B]$ and $\vdash_{A\to B} = \{(i_A, i_B)\} \cup \vdash_A \cup \vdash_B$. A ***justified sequence*** in a prearena A is a finite sequence s of moves of A satisfying the following condition: the first move must be initial, but all other moves m must be equipped with a pointer to an earlier occurrence of a move m' such that $m' \vdash_A m$. We then say that m' *justifies* m. If m is an answer, we also say that m *answers* m'. Given a justified sequence, the last unanswered question will be called *pending*.

Definition 7. *A* **play** *in A is a justified sequence satisfying alternation (players take turns) and well-bracketing (whenever a player plays an answer, it must answer the current pending question). A* **strategy** *in a prearena A is a subset σ of even-length plays in A that is closed under the operation of taking even-length prefixes and satisfies determinacy: if $sp_1, sp_2 \in \sigma$ then $sp_1 = sp_2$.*

Example 8. $\mathsf{cell}_{\mathsf{int}} : 1 \to [\![\mathsf{ref}(\mathsf{int})]\!]$ answers the initial question with $\circ$. Whenever O plays $\mathsf{write}(i)$, it responds with ok. After O plays read, it responds with an integer value present in the latest $\mathsf{write}(i)$ move by O or, if none has been played, with 0. This strategy will model $\vdash \mathsf{ref}(0) : \mathsf{ref}(\mathsf{int})$.

$\mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}} : 1 \to [\![\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})]\!]$ answers the initial question with $\circ$, responds to write and read with ok and $*$ respectively. If O plays q_r justified by an occurrence of $\star$, P plays q_w justified by the last occurrence of ok that precedes the relevant occurrence of $\star$. If none such exists, P has no response. Similarly, if O plays a_w, P will respond with a_r. This strategy will interpret $\vdash \mathsf{ref}(\lambda x^{\mathsf{unit}}.\Omega_{\mathsf{unit}}) : \mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$.

Strategies compose [6], yielding a category of games where objects are arenas and morphisms between objects A and B are strategies in $A \to B$. Let $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$. We shall write $[\![\Gamma \vdash \theta]\!]$ for the prearena $[\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$ (if $n = 0$ we take the left-hand side to be 1). The game model proposed in [1] interprets a term $\Gamma \vdash M : \theta$ by a strategy in $[\![\Gamma \vdash \theta]\!]$.

We now introduce another condition on plays, known to characterize denotations of terms with ground-type storage only.

Definition 9 (Visibility). *The* view *of a play is inductively defined by:*

$$view(\epsilon) = \epsilon \qquad view(m) = m \qquad view(s_1\, m\, s_2\, n) = view(s_1)\, m\, n\,.$$

A play s satisfies the **visibility** *condition if, for all even-length prefixes $s'm$ of s, the justifier of m occurs in $view(s')$. A strategy is called* **visible** *if it contains only visible plays.*

It can be shown that in plays the above condition is never violated by answers, because the pending question is always present in the view.

Proposition 10 ([3]). *Let $\Gamma \vdash M : \theta$ be a term in which applications of the $\mathsf{ref}(-)$-constructor are restricted to terms of type* unit *and* int. *Then $[\![\Gamma \vdash M : \theta]\!]$ satisfies the visibility condition.*

4 Factorisation

We shall next write $!_A$ for the strategy in $A \to 1$ that responds to the initial move on the left with the unique move on the right. Given strategies $\sigma_i : 1 \to A_i$ that all respond to the initial question, we write $\langle\, \sigma_1, \cdots, \sigma_n \,\rangle$ for the strategy in $1 \to \bigotimes_{i=1}^{n} A_i$ that responds to the initial move with the tuple containing the individual responses of the n strategies and otherwise behaves like σ_i, depending on the component A_i in which O chooses to play.

Let us recall the factorisation result from [1].

Theorem 11 ([1]). *Let $\sigma : A_1 \to A_2$ be a finite strategy and $A = [\![\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})]\!]$. There exists a natural number n and a visible strategy $\overline{\sigma} : (\bigotimes_{i=1}^{n} A) \otimes A_1 \to A_2$ such that $\langle\, \tau, \cdots, \tau, \mathsf{id}_{A_1} \,\rangle ; \overline{\sigma} = \sigma$, where $\tau = !_{A_1} ; \mathsf{cell}_{\mathsf{unit} \to \mathsf{unit}}$.*

Note that in the result above n may depend on σ. In fact, the proof shows that n can be taken to be (roughly) the length of the longest play in σ.

Remark 12. Violations of visibility describe computational scenarios in which a program attempts to refer to a value that was previously encountered during computation, yet which is not in current scope. The argument from [1] proposes to repair such violations by using free (higher-order) reference variables. Intuitively, they provide an opportunity to record the values currently available to the program. A later attempt to access the reference makes it possible to use the required value. In contrast, our argument will take advantage of a single reference cell. We shall also record the scope at each step, but before doing so we will embed the previous value into the current scope, thus allowing backtracking. In this way, the sought value can be found by backtracking to the desired computational step.

Theorem 13 (Visible Factorisation). *Let $\sigma : A_1 \to A_2$ be a strategy and $A = [\![\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})]\!]$. There exists a visible strategy $\overline{\sigma} : A \otimes A_1 \to A_2$ such that $\langle \tau, \mathsf{id}_{A_1} \rangle ; \overline{\sigma} = \sigma$, where $\tau = !_{A_1} ; \mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}$. If σ is recursively presentable, so is $\overline{\sigma}$.*

Proof. We shall define $\overline{\sigma}$ to be the least strategy containing the plays from $\{\overline{s} \mid s \in \sigma\}$, where $\overline{s}$ will be defined below by induction on the length of a play. Roughly, $\overline{s}$ will consist of s augmented with moves from A.

– In particular, immediately after each O-move of s we shall insert the sequence $\mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}$ and, if the move is an answer, it will be followed by a sequence consisting of answers a_w, a_r. Intuitively, each sequence $\mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}$ corresponds to reading the current value of the reference (the one modelled in A) and updating it with a new value.

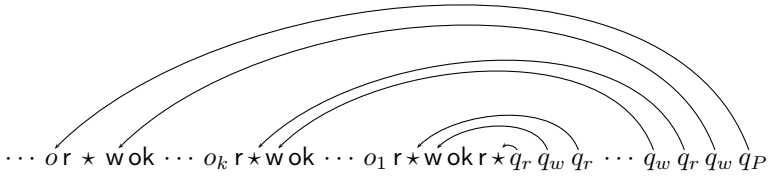
– For P-questions, we shall insert $\mathsf{read}\ \star$ followed by a sequence consisting of questions q_r, q_w *in front of* the P-question. The last q_w will point at the value stored in the reference immediately after the justifier of q was played. P-answers will simply be copied without any extra moves.

We give a precise definition below. The targets of pointers from read, $\star$, write, ok are obvious so, when discussing pointers, we shall focus on those from q_r, q_w, a_r, a_w.

– $\overline{s\,q_O} = \overline{s}\,q_O\ \mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}$ (if $|s| > 0$); and $\overline{q_O} = q_O\ \mathsf{write}\,\mathsf{ok}$
– $\overline{s\,q_P} = \overline{s}\,\mathsf{read}\ \star\ q_r\,(q_w\,q_r)^k\,q_w\,q_P$

  We take k to be the number of O-moves occurring after the justifier o of q_P in s. Let us list them (in order of occurrence) as $o_k, \cdots, o_1$. Then the ith q_w and q_r in $(q_w\,q_r)^k$ are meant to be justified by respectively write and $\star$ from the $\mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}$ segment introduced immediately after o_i. The last q_w is justified by write from the $\mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}$ segment added after o.

  Note that the resultant sequence will satisfy P-visibility, even if q_P may not have. Additionally, the extra O-moves $\star, \mathsf{ok}, q_w$ are consistent with the behaviour of the $\mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}$ strategy.



– $\overline{s\,a_O} = \overline{s}\,a_O\ \mathsf{read}\ \star\ \mathsf{write}\,\mathsf{ok}\,(a_w\,a_r)^{k+1}$

  Suppose a_O answers q_P in s. Then we take k to be the same as in the clause for q_P, i.e. k is the number of O-moves separating q_P's justifier and q_P. The sequence $(a_w\,a_r)^{k+1}$ simply answers all the questions q_w, q_r that were introduced for q_P. Because the pending question of $\overline{s}$ stays the same as that in s, this will yield a valid play. Note also that the O-moves a_r (in response to a_w) are consistent with the $\mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}$ strategy.

$$\cdots\ \mathsf{r}\ \star\ q_r\,(q_w\,q_r)^k\,q_w\,q_P\ \cdots\ a_O\ \ (a_w\,a_r)^{k+1}$$

– $\overline{s\, a_P} = \overline{s}\, a_P$
  As we have already mentioned, the construction of $\overline{s}$ from s preserves the pending question. Hence, the above clause leads to a play.

Consequently, $\overline{\sigma}$ is visible and, because the inserted moves are consistent with $\mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}$, we have $\langle\, !_{A_1}; \mathsf{cell}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{id}_{A_1}\,\rangle; \overline{\sigma} = \sigma$. That $\overline{\sigma}$ is recursively presentable follows from our description above. □

In order to apply the Theorem we need two more results. The first of them is classic and concerns decomposing visible strategies into innocent ones. Innocence [7] is a condition even stricter than visibility: responses of innocent strategies are uniquely determined by views.

Theorem 14 (Innocent Factorisation [4]). *Let $\sigma : A_1 \to A_2$ be a visible strategy and $A = [\![\mathsf{ref}(\mathsf{int})]\!]$. There exists an innocent strategy $\hat{\sigma} : A \otimes A_1 \to A_2$ such that $\langle\, !_{A_1}; \mathsf{cell}_{\mathsf{int}}, \mathsf{id}_{A_1}\,\rangle; \hat{\sigma} = \sigma$.*

Note that this result already applies to arbitrary strategies rather than just finite ones. Also, the construction of $\hat{\sigma}$ is effective and shows that $\hat{\sigma}$ is recursively presentable if σ is.

Finally, we prove a universality result for recursively presentable innocent strategies. Universality results were not necessary in research on full abstraction, because their weaker variants phrased for finite (or finitely generated) strategies sufficed to capture possible separating contexts. Hence, after the initial ones for PCF [2,7], they all but disappeared from subsequent papers. For program transformations, though, we need to be able to express arbitrary recursive strategies, hence the need for universality. Note that there is a huge difference between finite and recursive strategies. For instance, the strategy corresponding to $\lambda x^{\mathsf{int}}.x$ is not finite.

Theorem 15 (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 $[\![\Gamma \vdash M : \theta]\!] = \sigma$. If Γ and θ do not contain occurrences of* ref*-types, then M can be taken to be* mkvar*-free.*

By appealing to Theorems 13, 14 and 15 one can deduce Universality.

Theorem 16 (Universality). *Let $\sigma : [\![\Gamma \vdash \theta]\!]$ be a recursively presentable strategy. Then there exists $\Gamma \vdash M : \theta$ such that $[\![\Gamma \vdash M : \theta]\!] = \sigma$.*

In fact, in the above statement M can be taken to be of the form

$$\mathsf{let}\, f, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M',$$

where M' is ref-free. Because the game semantics of a term is recursively presentable, we can conclude the following result.

Theorem 17 (Transformation). *Let $\Gamma \vdash M : \theta$. There exists a term $\Gamma, f : \mathsf{ref}(\mathsf{unit} \to \mathsf{unit}), x : \mathsf{ref}(\mathsf{int}) \vdash M' : \theta$ satisfying the following conditions.*

– $\Gamma \vdash M \cong \mathsf{let}\, f, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M'$.
– *M' is* ref*-free.*
– *If there are no occurrences of* ref *in Γ, θ, then M' is* mkvar*-free.*

Thus, general references in $\mathcal{L}$ can be simulated by two memory cells that store values of type $\mathsf{unit} \to \mathsf{unit}$ and int respectively. Our proof was semantic, but the passage from M to M' can be made effective. However, due to reliance on the universality result, we would need to pass through enumerations of partial recursive functions. This is hardly a reasonable way of transforming programs! Next we shall identify several syntactic decomposition principles for general references, which will yield an alternative proof of the Theorem.

5 Syntactic Transformation

Note that $\mathsf{ref}(M)$ is equivalent to $\mathsf{let}\, x = \mathsf{new}_\theta\, \mathsf{in}\, (x := M; x)$ for a suitable θ. Consequently, w.l.o.g. we can assume that the only occurrences of $\mathsf{ref}(\cdots)$ inside terms are those associated with new_θ. Similarly, we assume that terms do not contain fixed-point subterms, as these can be simulated using higher-order reference cells [1].

Next we show new_θ can be decomposed using instances of new at simpler types. Ultimately, this will allow us to replace any occurrences of $\mathsf{ref}(M)$ with $\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}$ and $\mathsf{new}_{\mathsf{int}}$. The mkvar constructor is central to the transformations.

Lemma 18 (Decomposition of $\mathsf{ref}(\theta_1 \to \theta_2)$). *For all θ_1, θ_2, $\vdash \mathsf{new}_{\theta_1\to\theta_2} \cong \mathsf{let}\, f, x_1, x_2 = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\theta_1}, \mathsf{new}_{\theta_2}\, \mathsf{in}\, \mathsf{mkvar}(M_r, M_w)$, where*

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

We can show the equivalence formally by comparing strategies corresponding to each term. Intuitively, the equivalence is valid because on assignment M_w indirectly records the assigned value g in f. On dereferencing, M_r ensures that the latest value of f is accessed and the corresponding value g applied to the right argument through the internal references x_1 and x_2.

Lemma 19 (Decomposition of $\mathsf{ref}(\mathsf{ref}(\theta))$). *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*

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

Here, instead of storing a reference of type θ, we store the associated read and write methods, of types $\mathsf{unit} \to \theta$ and $\theta \to \mathsf{unit}$ respectively, which is what references r and w are used for.

Lemma 20. $\vdash \mathsf{new}_{\mathsf{unit}} \cong \mathsf{mkvar}(\lambda x^{\mathsf{unit}}.\, (), \lambda x^{\mathsf{unit}}.\, ())$.

The Lemma is easy to verify by reference to the game model. It illustrates the rather strange status of type $\mathsf{ref}(\mathsf{unit})$ in $\mathcal{L}$, in particular the fact that it is not possible to compare reference names (of type $\mathsf{ref}(\mathsf{unit})$) in the language.

The last three Lemmas imply the following corollary.

Corollary 21. *For any $\Gamma \vdash M : \theta$ there exists $\Gamma \vdash M'' : \theta$ such that $\Gamma \vdash M \cong M'' : \theta$ and occurrences of the* ref *constructor in M'' are restricted to terms of the form* $\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}$ *or* $\mathsf{new}_{\mathsf{int}}$.

In the result above, $\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}$ and $\mathsf{new}_{\mathsf{int}}$ are allowed to occur multiple times. In what follows we shall show that one occurrence of each suffices.

Lemma 22. *There exist* ref*-free terms M, N such that*

$$\vdash \lambda x^{\mathsf{unit}}.\, \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}} \cong \mathsf{let}\, f, x = \mathsf{new}_{\mathsf{int}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M : \mathsf{unit} \to \mathsf{ref}(\mathsf{unit} \to \mathsf{unit}),$$
$$\vdash \lambda x^{\mathsf{unit}}.\, \mathsf{new}_{\mathsf{int}} \cong \mathsf{let}\, x = \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, N : \mathsf{unit} \to \mathsf{ref}(\mathsf{int}).$$

Proof. We can encode an unbounded number of references of type $\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$ with a reference f of type $\mathsf{ref}(\mathsf{int} \to \mathsf{unit})$ by giving to each $(\mathsf{unit} \to \mathsf{unit})$-valued reference a unique integer identifier i, and encoding the value of the ith such reference as $\lambda v^{\mathsf{unit}}.\, (!f)i$. We use the internal variable x to count the number of generated references, so as to assign them unique identifiers. Thus, M can be taken to be $\lambda z^{\mathsf{unit}}.\mathsf{let}\, i = !x\, \mathsf{in}\, (x := !x + 1); \mathsf{mkvar}(M_r, M_w)$, where $M_r \equiv \lambda u^{\mathsf{unit}}.\mathsf{let}\, h = !f\, \mathsf{in}\, \lambda v^{\mathsf{unit}}.\, h\, i$ and $M_w \equiv \lambda g^{\mathsf{unit}\to\mathsf{unit}}.\, \mathsf{let}\, g' = !f\, \mathsf{in}\, f := (\lambda y^{\mathsf{int}}.\, \mathsf{if}\, y = i\, \mathsf{then}\, g()\, \mathsf{else}\, g' y)$.

For the second part, assume a standard encoding $\mathsf{G}(-) : \mathbb{Z}^* \to \mathbb{Z}$ of lists of integers into integers such that $\mathsf{G}(\epsilon) = 0$. Clearly, one can construct closed PCF terms $\mathsf{len} : \mathsf{int} \to \mathsf{int}$, $\mathsf{add} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$, $\mathsf{proj} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$ and $\mathsf{upd} : \mathsf{int} \to \mathsf{int} \to \mathsf{int} \to \mathsf{int}$ such that, for all $s \in \mathbb{Z}^*$ and $i, j \in \mathbb{Z}$:

$$\mathsf{len}\, \mathsf{G}(s) \Downarrow |s|\,, \quad \mathsf{add}\, \mathsf{G}(s)\, i \Downarrow \mathsf{G}(si)\,, \quad \mathsf{proj}\, \mathsf{G}(s)\, j \Downarrow s_j\,, \quad \mathsf{upd}\, \mathsf{G}(s)\, j\, i \Downarrow \mathsf{G}(s[j \mapsto i])\,,$$

where $|s|$ is the length of s, s_j is the jth element of s, and $s[j \mapsto i]$ is the list s with its jth element changed to i. We can then keep track of an unbounded number of integer-valued references by taking N to be

$$\lambda z^{\mathsf{unit}}.\; x := \mathsf{add}\, (!x)\, 0; \mathsf{let}\, j = \mathsf{len}(!x)\, \mathsf{in}\, \mathsf{mkvar}(\lambda z^{\mathsf{unit}}.\, \mathsf{proj}\, (!x)\, j, \lambda i^{\mathsf{int}}.\, x := \mathsf{upd}\, (!x)\, j\, i).$$

□

Now we are ready to give a new proof of Theorem 17. For a start, we tackle the first two claims therein.

Proof. Given $\Gamma \vdash M : \theta$, from Corollary 21 we can obtain an equivalent term M'', in which occurrences of ref are restricted to $\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}$ and $\mathsf{new}_{\mathsf{int}}$. Observe that M'' is thus equivalent to $\mathsf{let}\, h = \lambda x^{\mathsf{unit}}.\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}\, \mathsf{in}\, M_1$, where $M_1 \equiv M''[h()/\mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}]$ and the only occurrences of ref in M_1 are those of $\mathsf{new}_{\mathsf{int}}$. Applying Lemmata 22, 20 and 18, M'' is further equivalent to $\mathsf{let}\, f = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}\, \mathsf{in}\, M_2$, where the only occurrences of ref in M_2 are those of $\mathsf{new}_{\mathsf{int}}$. Finally, noting that M_2 is equivalent to $\mathsf{let}\, h' = \lambda x^{\mathsf{unit}}.\mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M_3$, where M_3 is ref-free, and invoking Lemma 18 we can conclude that M_2 is equivalent to $\mathsf{let}\, x = \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M_4$, where M_4 is ref-free. Now we can take M' (from the statement of the Theorem) to be $\mathsf{let}\, f, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, M_4$. □

Note that the decompositions presented in this Section relied on the availability of mkvar and the term M' from the above proof will in general contain many occurrences

of mkvar. We devote the remainder of this Section to showing that when Γ and θ are ref-free, all the occurrences of mkvar can actually be eliminated. To that end, we shall rely on a notion of canonical form, defined below.

$$\begin{array}{rl}\mathbb{C} ::= & () \mid x^{\mathsf{int}} \mid \mathsf{mkvar}(\lambda u^{\mathsf{unit}}.\mathbb{C}, \lambda v^{\theta}.\mathbb{C}) \mid \lambda x^{\theta}.\mathbb{C} \mid \mathsf{if}\,\mathbb{C}\,\mathsf{then}\,\mathbb{C}\,\mathsf{else}\,\mathbb{C} \mid \\ & \mathsf{let}\,y = i\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\,y = \mathbb{C} \oplus \mathbb{C}\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\,y = !x\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\,y = (x := \mathbb{C})\,\mathsf{in}\,\mathbb{C} \mid \\ & \mathsf{let}\,y = x\,\mathbb{C}\,\mathsf{in}\,\mathbb{C} \mid \mathsf{let}\,y = \mathsf{ref}(\mathbb{C})\,\mathsf{in}\,\mathbb{C}\end{array}$$

The canonical forms enjoy the following property.

Lemma 23. *For any $\Gamma \vdash M : \theta$ without fixed points, there exists a term $\mathbb{C}_M$ in canonical form such that $\Gamma \vdash M \cong \mathbb{C}_M : \theta$. Moreover, $\mathbb{C}_M$ can be effectively found and the conversion does not add any occurrences of* ref.

It turns out that canonical subterms of canonical terms have types drawn from a rather restricted set. We make this statement precise below.

Definition 24. *Given a type θ, the sets $\mathsf{PST}(\theta)$ (of positive subtypes of θ) and $\mathsf{NST}(\theta)$ (of negative subtypes of θ) are defined respectively as follows. Let us write $\mathsf{ST}(\theta)$ for $\mathsf{PST}(\theta) \cup \mathsf{NST}(\theta)$.*

$$\begin{array}{rlrl}\mathsf{PST}(\mathsf{unit}) &= \{\mathsf{unit}\} & \mathsf{PST}(\mathsf{ref}(\theta)) &= \mathsf{ST}(\theta) \cup \{\mathsf{ref}(\theta)\} \\ \mathsf{PST}(\mathsf{int}) &= \{\mathsf{int}\} & \mathsf{PST}(\theta_1 \to \theta_2) &= \mathsf{NST}(\theta_1) \cup \mathsf{PST}(\theta_2) \cup \{\theta_1 \to \theta_2\} \\ \mathsf{NST}(\mathsf{unit}) &= \emptyset & \mathsf{NST}(\mathsf{ref}(\theta)) &= \mathsf{ST}(\theta) \\ \mathsf{NST}(\mathsf{int}) &= \emptyset & \mathsf{NST}(\theta_1 \to \theta_2) &= \mathsf{PST}(\theta_1) \cup \mathsf{NST}(\theta_2)\end{array}$$

Given a canonical form $\mathbb{C}$ such that $\Gamma \vdash \mathbb{C} : \theta$, let $\mathsf{RT}(\mathbb{C})$ stand for the set of types θ' such that $\mathbb{C}$ contains an occurrence of $\mathsf{ref}(\mathbb{C}')$, where $\mathbb{C}'$ of type θ'. It turns out that the types in $\mathsf{RT}(\mathbb{C})$ together with types present in the original typing judgment determine types of canonical subterms, as made precise below.

Lemma 25. *Suppose $\Gamma \vdash \mathbb{C} : \theta$. Let*

$$\begin{array}{l}\mathbf{L} = (\bigcup_{(x:\theta_x)\in\Gamma} \mathsf{PST}(\theta_x)) \cup \mathsf{NST}(\theta) \cup (\bigcup_{\theta_r\in\mathsf{RT}(\mathbb{C})} \mathsf{ST}(\mathsf{ref}(\theta_r))) \cup \{\mathsf{unit}, \mathsf{int}\}, \\ \mathbf{R} = (\bigcup_{(x:\theta_x)\in\Gamma} \mathsf{NST}(\theta_x)) \cup \mathsf{PST}(\theta) \cup (\bigcup_{\theta_r\in\mathsf{RT}(\mathbb{C})} \mathsf{ST}(\theta_r)) \cup \{\mathsf{unit}, \mathsf{int}\}.\end{array}$$

Then, for any subterm $\mathbb{C}'$ of $\mathbb{C}$ which is also in canonical form, we have $\Gamma' \vdash \mathbb{C}' : \theta'$, where $\mathsf{cod}(\Gamma') \subseteq \mathbf{L}$ and $\theta' \in \mathbf{R}$.

Corollary 26. *Suppose $\Gamma \vdash \mathbb{C} : \theta$, $\mathsf{RT}(\mathbb{C}) = \{\mathsf{int}, \mathsf{unit} \to \mathsf{unit}\}$ and Γ, θ are* ref*-free. Then $\mathbb{C}$ does not contain any occurrences of* mkvar.

Proof. Because $\mathsf{RT}(\mathbb{C}) = \{\mathsf{int}, \mathsf{unit} \to \mathsf{unit}\}$, by Lemma 25, $\mathbb{C}$ can only contain mkvar if $(\bigcup_{(x:\theta_x)\in\Gamma} \mathsf{NST}(\theta_x)) \cup \mathsf{PST}(\theta)$ contains a ref-type. Since Γ and θ are ref-free this cannot be the case. □

This completes a syntactic proof of Theorem 17.

Remark 27. Note that Lemma 22 may reintroduce fixed points into the language, because it relies on numerical operations defined in PCF. We can still reduce terms containing such definitions to canonical form by assuming that the required operations are primitive (represented by $\oplus$). If this is not desirable then, after the elimination of mkvar under the above assumption, we can put back the PCF definitions without jeopardizing the result (mkvar is not available in PCF).

6 When Integer References Suffice

Next we shall examine the conditions under which references of type $\mathsf{unit} \to \mathsf{unit}$ can also be eliminated, i.e. all uses of general references can be replaced with a single integer-valued memory cell. In technical terms, this requires us to characterize the arenas where plays are guaranteed to satisfy visibility.

Definition 28. *Let A be an arena and $m_1, m_2 \in M_A$. We shall say that m_1 and m_2 are* equireachable *if there are paths ms_1m_1 and ms_2m_2 in the graph $(M_A, \vdash_A)$ such that m is initial and, if s_1 and s_2 both start with an answer, say a_1 and a_2 respectively, then $a_1 = a_2$.*

Remark 29. For arenas which are denotations of types, as is the case in Lemma 32, the notion of equireachability trivialises somewhat. In particular, any non-initial O-moves m_1 and m_2 are equireachable. We introduced a more general definition above so as to be able to state Lemma 31.

Definition 30. *An arena A is called* **visible** *if there are no equireachable non-initial moves $m, m' \in M_A$ such that m is an O-question and m' enables a P-question.*

Lemma 31. *Let A be an arena such that each question enables an answer. All plays of A satisfy the visibility condition if and only if A is visible.*

Proof. Let s be a play of A that violates the visibility condition. Suppose further that s ends in the P-move p_2, which breaks visibility for the first time and let o_1 be its justifier. Then, since s breaks visibility at p_2, it must look like:

$$m \cdots p_1 \cdots o_1 \cdots o_2 \cdots p_2$$

for some initial move m, where o_2 appears in the view right before p_2 and where p_2 is a question. Observe also that, since p_2 violates visibility, its justifier o_1 cannot be initial. If o_2 is a question we are done: A is not visible because of $(m, m') = (o_2, o_1)$. So, suppose that o_2 is an answer. Then, p_1 is a question and the move o_2' immediately following it in s is also a question (otherwise it would answer p_1). Moreover o_2' is not initial. Consequently, A is not visible due to $(m, m') = (o_2', o_1)$.

Conversely, suppose that A is not visible and let the latter be witnessed by paths $ms_1p_1o_2$ and $ms_2o_1p_2$ in $(M_A, \vdash_A)$. We form a play s as follows.

– If s_2 does not start with an answer, we set

$$s \;=\; m\, s_1\, p_1\, o_2\, s_2\, o_1\, p_2\, o_2\, p_2\,.$$

– If s_1p_1, s_2 both start with an answer, say $s_1p_1 = as_1'$ and $s_2 = as_2'$, we set

$$s \;=\; m\, a\, s_1'\, s_2'\, o_1\, p_2\, o_2\, p_2$$

where the leftmost pointer points to the last move of s_1'.

– If s_2 starts with an answer but s_1 does not, we set

$$s \;=\; m\, s_1\, p_1\, s_1'\, s_2\, o_1\, p_2\, o_2\, p_2$$

where s_1' is a sequence of moves answering all open questions of $s_1 p_1$.

Now observe that, in each case, the play s breaks visibility at move p_2. □

As a next step we would like to understand what typing judgments give rise to visible arenas. Our answer will be phrased in terms of syntactic shape. For simplicity, we shall now restrict our discussion to types generated from unit (Remark 33 explores the consequences of the results for the full type system). The following two lemmas capture scenarios relevant to verifying visibility for arenas. We write Θ_1 for the collection of first-order types, generated by the grammar $\Theta_1 ::= \mathsf{unit} \,|\, \mathsf{unit} \to \Theta_1$. Similarly, $\Theta_1 \to \mathsf{unit}$ stands for $\{\theta_1 \to \mathsf{unit} \,|\, \theta_1 \in \Theta_1\}$.

Lemma 32. *Let $A = [\![\theta_1, \cdots, \theta_k \vdash \theta]\!]$, where $\theta_1, \cdots, \theta_k, \theta$ are generated from* unit.

– *All O-questions in A are initial iff $\theta_i \in \Theta_1$ for all $1 \le i \le k$ and $\theta = \mathsf{unit}$.*
– *A does not contain a P-question enabled by a non-initial O-move iff $\theta_i \in \{\mathsf{unit}\} \cup (\Theta_1 \to \mathsf{unit})$ for $1 \le i \le k$ and $\theta \in \Theta_1$.*

Consequently, A is visible if and only if one of the conditions above is satisfied.

Remark 33. To see whether any occurrences of ref-types generate visible arenas, recall that $[\![\mathsf{ref}(\theta)]\!] = [\![\theta \to \mathsf{unit}]\!] \times [\![\mathsf{unit} \to \theta]\!]$. Consequently, for the purpose of determining visiblity $\mathsf{ref}(\mathsf{unit})$ can be viewed as $\mathsf{unit} \to \mathsf{unit}$. Thus, $\mathsf{ref}(\mathsf{unit})$ can be used whenever $\mathsf{unit} \to \mathsf{unit}$ is allowed. Note also that it is immaterial whether we consider unit or int. The observations yield the following typing constraints for visible arenas: ($\theta_i ::= \beta \,|\, \mathsf{ref}(\beta) \,|\, \Theta_1 \to \beta$ and $\theta ::= \Theta_1$) or ($\theta_i ::= \Theta_1$ and $\theta ::= \beta$), where $\beta ::= \mathsf{unit} \,|\, \mathsf{int}$ and $\Theta_1 ::= \beta \,|\, \mathsf{ref}(\beta) \,|\, \beta \to \Theta_1$. Analogously, $\mathsf{ref}(\beta \to \beta)$ should be viewed as a combination of $(\beta \to \beta) \to \beta$ and $\beta \to \beta \to \beta$. The results above do not give us much room for using this type: it cannot occur on the right but, if $\theta \equiv \beta$ we can have $\theta_i \equiv \mathsf{ref}(\beta \to \beta)$.

Thanks to Theorems 14 and 15 we can derive:

Theorem 34. *Let $\Gamma \vdash \theta$ be such that $[\![\Gamma \vdash \theta]\!]$ is visible. For any $\Gamma \vdash M : \theta$, there exists $\Gamma, y : \mathsf{ref}(\mathsf{int}) \vdash M' : \theta$ such that the following conditions are satisfied.*

– *$\Gamma \vdash M \;\cong\; \mathsf{let}\, y = \mathsf{ref}(0)\, \mathsf{in}\, M'$.*
– *M' is* ref*-free.*
– *If $\Gamma \vdash \theta$ does not contain occurrences of* ref*, then M' is* mkvar*-free.*

Next we give several examples of terms in which uses of $\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$ are definitely not eliminable. This is because the terms generate plays that violate the visibility condition, to be contrasted with Proposition 10.

Example 35. The first example is simply $\vdash \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}} : \mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$. Its semantics contains the play

$$\star \circ \;\mathsf{write}\; \mathsf{ok}\; \mathsf{read}\; \star\; q_r\; q_w\,.$$

Other examples are obtained by extending the shape of types from Lemma 32 in various ways.

$$\vdash \mathsf{let}\, x, y = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\\ \lambda f^{\mathsf{unit}\to\mathsf{unit}}.\ (\mathsf{if}\ (!y{=}0)\ \mathsf{then}\ (y := 1; x := f)\ \mathsf{else}\ ()); (!x)() : (\mathsf{unit}\to\mathsf{unit})\to\mathsf{unit}$$

$$g : ((\mathsf{unit}\to\mathsf{unit})\to\mathsf{unit})\to\mathsf{unit} \vdash \mathsf{let}\, x, y = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\\ g(\lambda f^{\mathsf{unit}\to\mathsf{unit}}.\ (\mathsf{if}\ (!y = 0)\ \mathsf{then}\ (y := 1; x := f)\ \mathsf{else}\ ()); (!x)()) : \mathsf{unit}$$

$$g : \mathsf{unit}\to\mathsf{unit}\to\mathsf{unit} \vdash \mathsf{let}\, x, y = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}, \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\\ \lambda u^{\mathsf{unit}}.(\mathsf{if}\ (!y = 0)\ \mathsf{then}\ (y := 1; x := g())\ \mathsf{else}\ ()); (!x)() : \mathsf{unit}\to\mathsf{unit}$$

$$g : (\mathsf{unit}\to\mathsf{unit})\to\mathsf{unit}\to\mathsf{unit} \vdash\\ \mathsf{let}\, x = \mathsf{new}_{\mathsf{unit}\to\mathsf{unit}}\, \mathsf{in}\ (\ x := g(\lambda z^{\mathsf{unit}}.(!x)())\ );\ (!x)() : \mathsf{unit}$$

7 When All References Are Dispensible

Finally, at some types memory allocation turns out dispensible, i.e. there exist purely functional terms with equivalent observable behaviour. In game-semantic terms, these are types where all strategies are necessarily innocent [7].

Definition 36. *Let A be an arena such that any question enables an answer*[1]*. A is called* **innocent** *if all O-questions are initial.*

Remark 37. Let us observe that $[\![\theta_1, \cdots, \theta_k \vdash \theta]\!]$ is an innocent arena if and only if $\theta_i ::= \Theta_1$ and $\theta ::= \beta$.

Lemma 38. *Let A be an arena such that any question enables an answer. Every strategy $\sigma : A$ is innocent if and only if A is innocent.*

Proof. Suppose A is not innocent, i.e. there exists a non-initial O-question q_O. Let s be the chain of enablers leading from some initial move to q_O and let a_P be an answer to q_O. Then the strategy on A consisting of prefixes of sa_P is not innocent, because it will not contain $sa_Pq_Oa_P$. Thus, not all strategies in A are innocent.

Now assume that A is innocent. Consequently, all non-initial O-moves are answers. Thus, each odd-length play s in A must have the shape $q(qa)^*$. Consequently, $view(s) = s$ and each strategy on A is thus innocent. □

The following result then follows from Theorem 15.

Theorem 39. *Suppose $\Gamma \vdash M : \theta$ is such that $[\![\Gamma \vdash M : \theta]\!]$ is innocent. Then there exists $\Gamma \vdash M' : \theta$ satisfying all the conditions below.*

- $\Gamma \vdash M \cong M'$.
- *M' is* ref*-free.*
- *If there are no occurrences of* ref*-types in $\Gamma \vdash \theta$, then M' is* mkvar*-free.*

[1] All arenas corresponding to types are of this kind.

Example 40. Here are two examples of terms not covered by Theorem 39, i.e. terms that do not have purely functional counterparts, because the corresponding strategies are not innocent.

$$\vdash \mathsf{let}\, y = \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, \lambda z^{\mathsf{unit}}.\mathsf{if}\, (!y = 0)\, \mathsf{then}\, y := 1\, \mathsf{else}\, \Omega : \mathsf{unit} \to \mathsf{unit}$$

$$g : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathsf{let}\, y = \mathsf{new}_{\mathsf{int}}\, \mathsf{in}\, g(\lambda z^{\mathsf{unit}}.\mathsf{if}\, (!y = 0)\, \mathsf{then}\, y := 1\, \mathsf{else}\, \Omega) : \mathsf{unit}$$

8 Conclusion

We showed that general references in $\mathcal{L}$ can be simulated with two reference cells, of types $\mathsf{ref}(\mathsf{unit} \to \mathsf{unit})$ and $\mathsf{ref}(\mathsf{int})$ respectively. This was first demonstrated through a game-semantic argument and subsequently complemented by a syntactic recipe for program transformation. The latter was facilitated by the presence of the mkvar constructor. However, the results apply equally well to the mkvar-free framework, provided no reference types occur in the type of the term or those of its free identifiers (arbitrary internal uses are still allowed). Then the auxiliary occurrences of mkvar can actually be eliminated, so, in this context, mkvar can be viewed as a useful temporary addition to the language.

In the future, we would like to conduct a similar study using the nominal game model of [8]. In the nominal setting, decomposition results such as Lemmata 18 and 19 cannot be expected to hold. Another surprising challenge is that the obvious adaptation of the visibility condition fails to be preserved by composition.

References

1. Abramsky, S., Honda, K., McCusker, G.: Fully abstract game semantics for general references. In: Proceedings of LICS, pp. 334–344. Computer Society Press (1998)
2. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation 163, 409–470 (2000)
3. Abramsky, S., McCusker, G.: Call-by-Value Games. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 1–17. Springer, Heidelberg (1998)
4. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: O'Hearn, P.W., Tennent, R.D. (eds.) Algol-Like Languages, pp. 297–329. Birkhäuser (1997)
5. Bruce, K.B., Cardelli, L., Pierce, B.C.: Comparing object encodings. Inf. Comput. 155(1-2), 108–133 (1999)
6. Honda, K., Yoshida, N.: Game-theoretic analysis of call-by-value computation. Theoretical Computer Science 221(1–2), 393–456 (1999)
7. Hyland, J.M.E., Ong, C.-H.L.: On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. Information and Computation 163(2), 285–408 (2000)
8. Murawski, A.S., Tzevelekos, N.: Game semantics for good general references. In: Proceedings of LICS, pp. 75–84. IEEE Computer Society Press (2011)
9. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) Algorithmic Languages, pp. 345–372. North Holland (1981)
10. Sanjabi, S.B., Ong, C.-H.L.: Fully abstract semantics of additive aspects by translation. In: Proceedings of AOSD, pp. 135–148. ACM (2007)