Decidability and Syntactic Control of Interference [1]

J. Laird

Department of Computer Science, University of Bath, UK

Abstract

We investigate the decidability of observational equivalence and approximation in Reynolds' "Syntactic Control of Interference" (SCI), a prototypical functional-imperative language in which covert *interference* between functions and their arguments is prevented by the use of an affine typing discipline.

By associating denotations of terms in a fully abstract "relational" model of finitary basic SCI (due to Reddy) with multitape finite state automata, we show that observational approximation is not decidable (even at first order), but that observational equivalence is decidable for all terms.

We then consider the same problems for basic SCI extended with non-local control in the form of backwards jumps. We show that both observational approximation and observational equivalence are decidable in this "observably sequential" version of the language by describing a fully abstract games model in which strategies are regular languages.

1 Introduction

A significant difficulty for reasoning about programming languages that combine higher-order functions with imperative variables is the possibility of covert *interference* between functions and their arguments. A key example is the phenomenon of *aliasing*, in which two identifiers may be bound to the same cell. For instance, in a typical sequential imperative language without higher-order functions, the commands $x := 0; y := 1$ and $y := 1; x := 0$ are contextually equivalent, because the implicit assumption that x and y refer to different cells means that the order in which they are assigned is not observable.

Email address: jiml@cs.bath.ac.uk (J. Laird).

[1] This research was supported by UK EPSRC grant GR/S72181. An extended abstract of this paper appeared in the Proceedings of ICALP 2005 [13]

However, in functional-imperative languages such as Idealized Algol, aliasing means that this syntactic distinction between identifiers cannot be maintained under reduction, and the equivalence fails: e.g. $(\lambda y.x := 0; y := 1)\, x$ and $(\lambda y.y := 0; x := 1)\, x$ are clearly inequivalent.

This has led to interest in ways of controlling interference, and eliminating aliasing, since this should make it easier to reason about program behaviour, both informally and using inference systems such as forms of Hoare logic. Reynolds' *Syntactic Control of Interference* (SCI) [28] is a typing system for Idealized Algol which prevents covert interference by preventing sharing of variables between functions and their arguments. The object of this paper is to investigate decidability of the formal properties of observational equivalence and approximation in SCI, and related languages, over bounded datatypes.

The full SCI system (and variants such as SCIR [25]) contains a notion of passive type; the programs (and contexts) typable at purely passive types are essentially those of PCF, and so equivalence at such types is conservative over PCF equivalence and is therefore not decidable even when ground types are bounded [16]. Consequently, we restrict attention to the "actively typed" fragment or *basic* SCI, which may be regarded as the core of a family of related typing systems for interference control. We also restrict basic SCI to containing `while` loops rather than full recursion: we show that including the latter leads to undecidability of termination.

In the first part of this paper, we will show that observational equivalence for basic SCI (over bounded datatypes) is decidable, but that observational approximation is not, even for terms which contain no procedural abstractions. This result is perhaps suprising, since basic SCI is a subsystem of Idealized Algol, for which equivalence and approximation are both decidable for terms at (finitary) third-order types [24,22] and undecidable at fourth-order [21] . The undecidability of observational approximation (and associated results, such as the undecidability of equivalence in a non-deterministic variant of the language) raises the question of whether deciding such properties in the presence of interference control is always difficult. The problem in basic SCI is in some respects analogous to the situation in PCF: whilst it is a sequential language, it is not *observably sequential*; different sequentializations of a program may approximate each other (for example, $\lambda x.\lambda y.x; y$ is equivalent in SCI to $\lambda x.\lambda y.y; x$).

In the case of PCF, Cartwright and Felleisen [6] have shown that adding simple non-local control operators (`catch`) results in a language — "observably sequential PCF", or SPCF — with an effectively presentable fully abstract model, for which equivalence and approximation are therefore decidable at finitary types. Pursuing the analogy between PCF and SCI, we obtain a natural observably sequential version of the latter by adding backwards jumps to

labelled points (expressively equivalent to `catch`). We show that we may represent denotations in a fully abstract games model of observably sequential SCI as (single tape) finite state automata, and that observational approximation is therefore decidable.

1.1 Related Work

This study is inspired by "algorithmic game semantics" [7], in which the denotation of each term as a strategy in a fully abstract game semantics with a formal language. These techniques have been used to show that observational equivalence and approximation in (finitary) Idealized Algol are decidable at third-order types — by showing that the denoting strategies are recognized by deterministic pushdown automata [22] — but not at fourth order [21]. The latter undecidability result depends crucially on the nesting of function calls, which is not possible in SCI. Indeed, it is straightforward to give equationally *sound* interpretations of basic SCI terms as regular languages (cf. Abramsky's Serially Re-entrant Algol [2]).

These results suggest a methodological basis for investigating observational equivalence and approximation in basic SCI, but one obstacle to applying it is that game semantics appears to be "too sequential" to capture this directly. McCusker and Wall have described a game semantics of basic SCI (in fact, full SCIR) [20], based on a "non-linear" notion of view. Like the games models of PCF, the associated full abstraction result depends on a quotient operation, which renders it difficult to reason directly about equivalence.

However, basic SCI also has a simple semantics based on sets (or coherence spaces) and relations, described by Reddy [27], and investigated further by McCusker, who proved that it is fully abstract [18]. This model may also be thought of as a prototypical form of game semantics, and we may apply similar methodologies to reason about it. We show that the denotations of first-order terms correspond to a form of *multitape* deterministic finite state automata, as introduced by Rabin and Scott in 1959 [26]. It is straightforward to show that containment of 2-tape deterministic FSA is undecidable [11]. By contrast, the problem of decidability of equivalence for all multitape deterministic FSA remained open for thirty years, before being resolved (affirmatively) by Harju and Karhumäki [8]. These results have direct implications for basic SCI: denotational equivalence, which is observational equivalence, is decidable at first order but observational approximation (inclusion of denotations) is not (even though all first-order terms of Idealized Algol are typable in SCI — the point being that the possible *observations* of first-order terms are more restricted in SCI). Although denotations of terms at higher-order types do not correspond directly to multitape automata, we show that there is a "definable monomor-

phism" from every type to a first-order type, and that equivalence is therefore decidable at *all* types. We may compare this to languages such as SPCF, for which we may show that every type-object is a definable *retract* of a first-order type-object [14].

Our decidability results for observably sequential SCI are obtained using algorithmic game semantics: we define a category of games in which there are no explicit justification pointers, and so we may obtain finite state representations of strategies at all types, following [7]. The main development here is to define suitable constraints on strategies and to prove that the resulting semantics of observably sequential SCI is inequationally fully abstract (although it contains compact elements which are not definable). This model is more closely related to the game semantics of SCI of McCusker and Wall [20].

2 Basic SCI

The syntax of basic SCI [28] (to which we henceforth refer as simply "SCI") is, in essence, the same as that of Idealized Algol [29]: terms are generated from the λ-calculus with products, and constants representing numerals, conditionals and loops, and declaration, assignment and dereferencing of integer variables. Types are generated from a basis including the type of expressions returning integer values, and the type of variables storing integer values, using the constructors $\multimap$ (function-space) and & (additive product).

We work with a lean syntax which may be sugared up to more closely resemble that of Idealized Algol. We assume a base type $\underline{n}$ (of expressions) for each natural number n, containing the numerals $\{\mathtt{i} \mid i < n\}$. In particular, we have a type $\underline{0}$, containing no values, a unit type $\underline{1}$ and a binary type $\underline{2}$. We write T^n for the n-fold product of copies of T. We follow Reynolds in representing the type $\mathtt{var}[n]$ (for $n > 0$) of imperative variables with values in $\underline{n}$ as the type $\underline{n}\&\underline{1}^n$ (the product of the types of its "methods", dereferencing and assignment, which are therefore expressible as projections). So this version of the language implicitly contains "bad variables" — objects of `var`-type which do not behave as reference cells. This does not affect the results given here for basic SCI: approximation is shown to be undecidable for the `var`-free types, whilst McCusker has shown that observational equivalence in basic SCI with bad variables is conservative over equivalence in the language without bad variables [19].

Thus the types of finitary SCI are:

$$S, T ::= \underline{n} \mid S \multimap T \mid S\&T$$

where n ranges over the natural numbers, s $\underline{n}$. The depth or order of a

type (and the corresponding closed terms) is defined: $o(\underline{n}) = 0$, $o(S\&T) = \max\{o(S), o(T)\}$, $o(S \multimap T) = \max\{o(S)+1, o(T)\}$.

Typing judgements are based on the affine λ-calculus with pairing, with a typing coercion rule from $\underline{m}$ to $\underline{n}$ for $m < n$. The formation rules for terms in multiset contexts are:

$$\frac{}{x{:}T,\Gamma \vdash x{:}T} \qquad \frac{C{:}T \in \mathcal{C}}{\Gamma \vdash C{:}T} \qquad \frac{\Gamma \vdash M{:}\underline{m}}{\Gamma \vdash M{:}\underline{n}}\ m < n$$

$$\frac{\Gamma \vdash M{:}S \quad \Gamma \vdash N{:}T}{\Gamma \vdash \langle M,N\rangle{:}S\&T} \qquad \frac{\Gamma, x{:}S \vdash M{:}T}{\Gamma \vdash \lambda x.M{:}S \multimap T} \qquad \frac{\Gamma \vdash M{:}S \multimap T \quad \Delta \vdash N{:}S}{\Gamma,\Delta \vdash M\ N{:}T}$$

where the set of constants $\mathcal{C}$ consists of the following:

Projections $\pi_i : T_0\&T_1 \multimap T_i$, for $i \in \{0,1\}$ (yielding also $\pi_i : T^n \multimap T_i$ for $i < n$).
Numerals $\mathtt{i} : \underline{n}$ for $i < n$.
Case statements $\mathtt{case}_{n,m} : \underline{n}\&\underline{m}^n \multimap \underline{m}$ for $n > 0$,
Loops $\mathtt{while0} : \underline{m+1} \multimap \underline{1}$.
New variable declaration $\mathtt{new} : (\mathtt{var}[n] \multimap \underline{m}) \multimap \underline{m}$.

Note the additive typing of `case` which allows sharing of variables between sequentially executed procedures. We may derive terms $\mathtt{case}_{n,T} : \underline{n}\&T^n \multimap T$ for arbitrary T — if $M : \underline{1}$, we write $\mathtt{case}\,\langle M, N\rangle$ as $M; N$. We write Ω for $\mathtt{while0}\,\mathtt{0}$. Given $M : \mathtt{var}[n]$, we may write $!M$ for $\pi_0\, M$ and $M := N$ for $\mathtt{case}\,\langle N, \pi_1\, M\rangle$. We have omitted a constant for parallel composition, noting that in SCI it is (observationally) equivalent to sequential composition. We have included iteration (`while0`) rather than recursion (with appropriate typing restrictions) for reasons discussed in Section 2.1.

We say that a term-in-context $\Gamma \vdash M : T$ has order n if $o(T) \leq n$ and every variable in Γ has a type of order strictly less than n.

We give a "small-step" operational semantics based on *evaluation contexts*, which are defined by the following grammar:

$$E[\cdot] ::= [\_] \mid E[\_]\, M \mid \pi_i\, E[\_] \mid \mathtt{case}\, E[\_] \mid \mathtt{case}\,\langle E[\_], M\rangle \mid \mathtt{while0}\, E[\_]$$

A *store* is a finite partial function from location names to sets of pairs $\langle a, n\rangle$ of location names (variables of `var` type) and natural numbers. The reductions for pairs $M, \mathcal{E}$ of a term (of ground type) and a store are given in Table 1. These specify a unique reduction for every program (in environment) which is not a value, by "unique decomposition" of such programs into an evaluation context and a redex. In particular, note that a "head normal" term of product type may be either a pair, or else a location name – requiring several, non-conflicting rules for projection. We write $M \Downarrow$ if evaluation of $M, \varnothing$ terminates, and so

$$
\begin{array}{lcll}
E[(\lambda x.M)\,N], \mathcal{E} & \longrightarrow & E[M[N/x]], \mathcal{E} & \\
E[\pi_i\,\langle M_0, M_1\rangle] & \longrightarrow & E[M_i], \mathcal{E} & \\
E[\mathtt{case}\,\langle \mathtt{i}, M\rangle], \mathcal{E} & \longrightarrow & E[\pi_i\,M], \mathcal{E} & \\
E[\mathtt{while0}\,M], \mathcal{E} & \longrightarrow & E[\mathtt{case}\,\langle M, \langle \mathtt{while0}\,M, 0, \ldots, 0\rangle\rangle], \mathcal{E} & \\
E[\mathtt{new}\,M], \mathcal{E} & \longrightarrow & E[M\,a], \mathcal{E} \cup \{\langle a, 0\rangle\} & a \notin \pi_1(\mathcal{E}) \\
E[\pi_0\,a], \mathcal{E} & \longrightarrow & E[\mathtt{n}], \mathcal{E} & \langle a, n\rangle \in \mathcal{E} \\
E[\pi_i\,(\pi_1\,a)], \mathcal{E} & \longrightarrow & E[0], \mathcal{E}[a \mapsto i] &
\end{array}
$$

Table 1
Reduction rules for SCI programs

define standard notions of observational approximation and equivalence:
$M \lesssim N$ if $C[M] \Downarrow$ implies $C[N] \Downarrow$.
$M \simeq N$ if $M \lesssim N$ and $N \lesssim M$.

2.1 Undecidability of Termination in the Presence of Recursion

As in [18], we have presented basic SCI containing *iteration* (the `while0` constant). This is consistent with its role as the extension of a basic imperative language with higher-order procedures, although it differs from Reynolds' original SCI, which includes a fixpoint combinator, $\mathbf{Y} : (T \to T) \to T$ [28]. However, as Reynolds notes, $\mathbf{Y}$ cannot be used in an unrestricted way without causing interference (and breaking affine typing): reduction of $\mathbf{Y}M$ to $M\,(\mathbf{Y}M)$ is liable to result in interference between the two occurrences of M, unless they can be guaranteed to use any shared state passsively. In basic SCI, without any notion of passive typing, the obvious way to do this is to restrict the introduction rule for $\mathbf{Y}M$ so that M is a closed term — i.e. via the typing rule

$$\frac{\_ \vdash M : T \multimap T}{\mathbf{Y}M : T}$$

$\mathbf{Y}$ then preserves subject reduction with respect to SCI typing, and is clearly sufficient to express `while0`. What of the converse: can programs using the restricted form of the $\mathbf{Y}$ combinator be "refactored" to an equivalent form using iteration? For general SCI, over infinite data types, this is an interesting problem: it is well known that instances of tail recursion may be refactored in this way. In [14] it is established that in a (stateless) functional language with control we may reduce all instances of recursion to iteration.

In bounded basic SCI we may show that the $\mathbf{Y}$-combinator is *not* expressible using iteration, by showing that termination of programs which use $\mathbf{Y}$ is not decidable. (Thus all of the equivalence and approximation problems subsequently considered become undecidable in the presence of $\mathbf{Y}$.) Of course, equivalence in SCI with $\mathbf{Y}$ is conservative over equivalence in SCI with `while0`

by continuity — i.e. any equivalence which can be established using our methods will still hold when recursion is added.

It is not surprising that terms using recursion (and thus requiring an unbounded call stack for their execution) are not representable as finite state automata. Ong has shown, for example, that *queues* may be represented in Idealized Algol [24] over bounded datatypes, using fixpoints of second-order terms. This encoding is not SCI typable, as it requires state to be shared between recursive calls, but here we give a simple implementation for *Minsky machines* [23] in SCI with second order recursion.

We may present a (deterministic) 2-counter Minsky machine as a finite set of states S, a specified *final* state, and an instruction I_s for each non-final state in S, of one of the following forms:

- $\texttt{Increment}(i, s')$ $(i \in \{0, 1\})$ — increment counter i by one and move to state s'.
- $\texttt{Decrement}(i, s', s'')$ $(i \in \{0, 1\})$ — if counter i is non-zero then decrement it by one and move to state s', otherwise move to state s''.

This operates on configurations (m, n, s) representing the positive integer-values of the two counters and the current state s, respectively. The machine halts on a given initial configuration if it may reach the final state from that configuration.

The key to our representation of Minsky machines is the observation that the integer values in the counters may be represented as basic SCI terms of type $\underline{1} \multimap \underline{1}$ — specifically, the value n is represented (up to observational equivalence) as the term $\lambda x.x^n$, where $x^0 = \texttt{0}$, $x^{i+1} = x^i; x$. Given $M : \underline{1} \multimap \underline{1}$ we define:

- $\texttt{inc}(M) : \underline{1} \multimap \underline{1} = \lambda x.(M\, x); x$, which increments M.
- $\texttt{dec}(M) : \underline{1} \multimap \underline{1} = \lambda x.\texttt{new}\, \lambda y.M\, \texttt{case}\, \langle !y, \langle y := 1, x\rangle\rangle$, which decrements M.
- $\texttt{is0}(M) : \underline{2} = \texttt{new}\, \lambda y.(M\, (y := 1)); !y$ which returns 0 if M represents zero, and 1 otherwise.

Definition 2.1 *Given a Minsky machine P over the states $s_0, \ldots, s_{n-1}$, assume s_0 is the final state. For each $1 \leq i < n$ we define a term $F : (\underline{1} \multimap \underline{1}) \multimap (\underline{1} \multimap \underline{1}) \multimap \underline{n} \multimap \underline{1}, x : \underline{1} \multimap \underline{1}, y : \underline{1} \multimap \underline{1} \vdash P_i(F, x, y) : \underline{1}$ as follows:*

- *If I_i is $\texttt{Increment}(0, s_j)$ then $P_i = ((F\, \texttt{inc}(x))\, y)\, \texttt{j}$.*
  If I_i is $\texttt{Increment}(\texttt{1}, \texttt{s}_\texttt{j})$ then $P_i = ((F\, x)\, \texttt{inc}(y))\, \texttt{j}$.
- *If I_i is $\texttt{Decrement}(0, s_j, s_k)$ then $P_i = \texttt{case}\, \langle \texttt{is0}(x), ((F\, x)\, y)\, \texttt{k}, ((F\, \texttt{dec}(x))\, y)\, \texttt{j}\rangle$.*
  If I_i is $\texttt{Decrement}(\texttt{1}, \texttt{s}_\texttt{j}, \texttt{s}_\texttt{k})$ then $P_i = \texttt{case}\, \langle \texttt{is0}(y), ((F\, x)\, y)\, \texttt{k}, ((F\, x)\, \texttt{dec}(y))\, \texttt{j}\rangle$.

Thus we may represent P as the term $M_P : (\underline{1} \multimap \underline{1}) \multimap (\underline{1} \multimap \underline{1}) \multimap \underline{n} \multimap \underline{1}$

$$\mathbf{Y}\lambda F.\lambda x.\lambda y.\lambda z.\mathtt{case}\,\langle z, \langle \mathtt{0}, P_1(F,x,y), \ldots, P_{n-1}(F,x,y)\rangle\rangle$$

Proposition 2.2 *The Minsky machine P with initial configuration (m, n, s_i) halts if and only if $((M_p\,\lambda x.x^m)\,\lambda x.x^n)\,\mathtt{i} \Downarrow$.*

Proof: By induction on the number of steps to reach final state/number of unwindings of the fixpoint. (Using some evident observational equivalences, which we may verify using the fully abstract relational model.) □

Since the halting problem for Minsky machines is not decidable [23] we have:

Corollary 2.3 *Termination is not decidable in SCI with recursion.*

This leaves open the question of whether we may retain decidability of termination and equivalence whilst allowing fixpoints of terms lower than third order.

2.2 Relational Semantics of Basic SCI

We shall now describe the model of basic SCI, based on sets and relations, which we shall use to prove our main results. This is essentially a version of Reddy's semantics [27], simplified and proved to be fully abstract by McCusker [18], and we refer to these works for further details. (We shall give a self-contained proof of full abstraction, however.) In a previous version of this paper, we described both relational and games models of the affine λ-calculus and SCI using the notion of **BCK**-algebra rather than that of symmetric monoidal category with the necessary exponentials. This presentation is technically somewhat simpler, as it avoids the (unnecessary) definition of a tensor product, but we revert to the more familiar setting here as it allows more direct comparison with other models of related languages.

Reddy's semantics is presented using coherence spaces, but McCusker shows that, concretely, it is equivalent to an interpretation in the category **MonRel** of monoids and *monoidal relations*. A monoidal relation from $(|A|, e_A, \cdot)$ to $(|B|, e_B, \cdot)$ is a relation R on the underlying sets $|A|$ and $|B|$ with the following properties:

Identity $e_A \mathrm{R} e_B$.
Homomorphism If $a\mathrm{R}a'$ and $b\mathrm{R}b'$ then $a \cdot b \mathrm{R} a' \cdot b'$.
Decomposition If $a\mathrm{R}b_1 \cdot b_2$ then there exists $a_1, a_2 \in |A|$ such that $a_1\mathrm{R}b_1$, $a_2\mathrm{R}b_2$ and $a = a_1 \cdot a_2$.

We define symmetric monoidal structure on **MonRel**: the tensor product

$A \otimes B$ has underlying set $|A| \times |B|$ and pointwise monoidal operation, the unit I being the singleton monoid. Whilst $(\mathbf{MonRel}, I, \otimes)$ is not symmetric monoidal closed, it does have sufficient exponentials to define a model of the affine λ-calculus. For any set A, we have a free monoid $(A^*, \varepsilon, ++)$ of words over A with concatenation.

Lemma 2.4 *The (monoidal) functor* $|\_| : \mathbf{MonRel} \to \mathbf{Rel}$ *(forgetful) is left adjoint to* $(\_)^* : \mathbf{Rel} \to \mathbf{MonRel}$ *(free).*

Since right adjoints preserve limits, for any sets A and B, the free monoid $(A + B)^*$ is a cartesian product for A^* and B^* in **MonRel**.

Lemma 2.5 *For any monoid* A *and set* B, $(|A| \times B)^*$ *is the exponential of* A *by* B^* *in* **MonRel**.

Proof: We have a natural isomorphism $\mathbf{MonRel}(A \otimes B, C^*) \cong \mathbf{Rel}(|A \otimes B|, C) \cong \mathbf{Rel}(|A| \otimes |B|, C) \cong \mathbf{Rel}(|A|, (B \times C)) \cong \mathbf{MonRel}(A, (|B| \times C)^*)$. □

Thus we may interpret basic SCI types as sets, defining $[\![\underline{n}]\!] = \lceil n \rceil^*$, where $\lceil n \rceil = \{i \mid i < n\}$, $[\![S \& T]\!] = [\![S]\!] + [\![T]\!]$, and $[\![S \multimap T]\!] = [\![S]\!]^* \times [\![T]\!]$. Using the categorical structure of **MonRel**, we may interpret terms-in-context $x_1 : S_1, \ldots, x_n : S_n \vdash M : T$ of the affine λ-calculus with products as monoidal relations from $[\![S_1]\!]^* \otimes \ldots \otimes [\![S_n]\!]^*$ to $[\![T]\!]^*$, or equivalently, as relations from $|[\![S_1]\!]^*| \times \ldots \times |[\![S_n]\!]^*|$ to $[\![T]\!]$. Adopting the latter convention, we assign meanings to the remaining constants. [2]

- $[\![\mathtt{case}_{m,n}]\!] = \{\langle i^l j^{ir}, j\rangle \mid i < m \wedge j < n\}$
- $[\![\mathtt{while0}]\!] = \{\langle 0^* n, 0\rangle \mid n > 0\}$,
- $[\![\mathtt{new}]\!] = \{\langle\langle s, j\rangle, j\rangle \mid s \in \mathsf{cell}_0\}$, where $\mathsf{cell}_{n,k} \subseteq [\![\mathtt{var}[n]]\!]^* = (k^r)^*(\Sigma_{i<n} 0^{il}(i^r)^*)^*$.

(We may interpret the **Y**-combinator on closed terms as a fixpoint operator in **MonRel**, although **MonRel** does not have *parameterised* fixpoints.)

Proposition 2.6 $M \Downarrow$ *if and only if* $[\![M]\!] \neq \bot$.

Proof: See [18]. □

Corollary 2.7 $[\![M]\!] \subseteq [\![N]\!]$ *implies* $M \lesssim N$.

The converse is also true (i.e. the semantics is fully abstract) as shown in [18], although it does not have the "finite definability" property: there are compact elements which are not the denotation of any term. In particular, the

[2] We use superscripts to tag the elements of disjoint unions — e.g. if $a \in A$, then a^l represents $\mathsf{in}_\mathsf{l}(a) \in A + B$, and if $a \in A_i$, then a^i represents $\mathsf{in}_i(a) \in A_1 + \ldots + A_n$. We compound these operations by concatenation — e.g. if $a \in A_i$ then a^{il} represents $\mathsf{in}_\mathsf{l}(\mathsf{in}_i(a)) \in (A_1 + \ldots + A_n) + B$.

union of two definable elements is not definable in general (e.g. the union of left and right projection). Union may be used to interpret non-deterministic choice (with respect to partial correctness/may-testing): from $M, N : T$ form $M \mathtt{or} N$, interpreted as $[\![M]\!] \cup [\![N]\!]$.

We shall reduce full abstraction to the first-order case, for which we may give a simple proof.

Lemma 2.8 *Suppose $T \multimap \underline{l}$ is a first-order type type of order Then for any $e \in [\![T \multimap \underline{l}]\!]$ there exists a term $L_e : (T \multimap \underline{l}) \multimap \underline{1}$ such that $[\![L_e]\!] = \{\langle e, 0\rangle\}$.*

Proof: Suppose $T = \underline{n_0} \times \ldots \times \underline{n_k}$. For $j \leq k$, $i < n_j$, let $\Omega[M]_j : T = \langle \Omega_0, \ldots, \Omega_{j-1}, M, \Omega_{j+1}, \ldots, \Omega_k \rangle$. Let $\mathtt{eq} : \underline{n} \& \underline{n} \multimap \underline{1} =$
$\lambda x.\mathtt{case}\,\langle \pi_1\, x, \langle \mathtt{case}\,\langle \pi_2\, x, \Omega[\mathtt{0}]_0\rangle, \ldots, \mathtt{case}\,\langle \pi_2\, x, \Omega[\mathtt{0}]_{n-1}\rangle$ so that $[\![\mathtt{eq}\,\langle \mathtt{n}, \mathtt{n}\rangle]\!] = \{0\}$ and $[\![\mathtt{eq}\,\langle \mathtt{n}, \mathtt{m}\rangle]\!] = \{\}$ if $n \neq m$.

Suppose $e = \langle i_1^{j_1} \ldots i_m^{j_m}, a\rangle$. Then we define $L_e =$
$\lambda f.\mathtt{new}\,\lambda x.(\mathtt{eq}\,\langle f\ \mathtt{case}\,\langle !x, \langle \Omega[x := x{+}1; i_1]_{j_1}, \ldots, \Omega[x := x{+}1; i_m]_{j_m}, \Omega\rangle\rangle, \mathtt{a}\rangle); (\mathtt{eq}\,\langle !x, \mathtt{m}\rangle)$. □

Proposition 2.9 *If $x_1, \ldots, x_n \vdash M, N : \underline{m}$, are first-order terms then $M \lesssim N$ implies $[\![M]\!] \subseteq [\![N]\!]$.*

Proof: Suppose $[\![M]\!] \not\subseteq [\![N]\!]$. Then there exists $\langle\langle s_1, \ldots, s_n\rangle, i\rangle \in [\![M]\!]$ such that $\langle\langle s_1, \ldots, s_n\rangle, i\rangle \notin [\![N]\!]$. Let $C[\_] = L_{\langle s_1, 0\rangle}\, \lambda x_1. L_{\langle s_2, 0\rangle}\, \lambda x_2. \ldots L_{\langle s_n, i\rangle}\, \lambda x_n.[\_]$. Then $[\![C[M]]\!] = \{0\}$ and $C[N] = \{\}$ and hence $M \not\lesssim N$ as required. □

3 Multitape Automata and First-Order Terms

The particular form of (one-way) deterministic multitape finite state automaton we shall use is tailored for establishing a correspondence with the relational model. We shall call it a "disjoint final state automaton", because no transitions are possible from a final state. It is straightforward to show that it yields the same class of languages as the original notion [26], by using *end-markers*.

Definition 3.1 *A disjoint final state n-tape automaton α over a finite alphabet Σ is a tuple $(C_0, C_1, \ldots, C_n, F, s_0, \delta)$ consisting of disjoint sets of states $C_0, C_1, \ldots, C_n, F$, an initial state s_0 and a (partial) transition function $\delta : (\Sigma \cup \{\varepsilon\}) \times S \rightharpoonup S$, where $S = C_0 \cup \ldots \cup C_n \cup F$. C_0 consists of states in which (only) a ε-transition may be performed, C_i of states in which a symbol is read from tape i (for $0 < i \leq n$), and F of final states (for which δ is not defined).*

We say that α accepts a tuple of tapes $(t_1, \ldots, t_n)$ if it reads all symbols from each of them and then reaches a final state. Formally, for each final state f, we

may say that the set of pairs $A(\alpha) \subseteq (\Sigma^*)^n \times S$ of tuples and states accepted in by α in final state f is the union $\bigcup_{i\in\omega} A_f(\alpha, i)$, where:

- $A_f(\alpha, 0) = \{\langle\langle\varepsilon, \ldots, \varepsilon\rangle, f\rangle\}$,
- $A_f(\alpha, i+1) = \{\langle\langle t_1, \ldots, t_n\rangle, s'\rangle \mid s' \in C_0 \wedge \delta(\varepsilon, s') = s \wedge \langle\langle t_1, \ldots, t_n\rangle, s\rangle \in A_f(\alpha, i)\} \cup \{\langle\langle t_1, \ldots, mt_i, \ldots, t_n\rangle, s'\rangle \mid 0 < i \leq n \wedge s \in C_i \wedge \delta(m, s') = s \wedge \langle\langle t_1, \ldots, t_n\rangle, s\rangle \in A_f(\alpha, i)\}$.

We write $\mathcal{L}(\alpha)$ for the set $\{x \mid \exists f \in F.\langle x, s_0\rangle \in A_f(\alpha)\}$ of tuples accepted by α.

To simulate the more general notion of n-tape autmaton introduced in [26], we require *end-markers* for the tapes. Given a symbol $e \notin \Sigma$, we shall say that an automaton α over $\Sigma \cup \{e\}$ accepts $\mathcal{L} \subseteq \Sigma^*$ with end-marker e if $\mathcal{L}(\alpha) = \{we \mid w \in \mathcal{L}\}$. We write $\mathcal{L}^e(\alpha)$ for the language accepted by α, with end-marker e (if any).

The original definition of n-tape automaton [26] includes end-markers, but differs from Definition 3.1 in the following respects:

- for a tuple to be accepted only requires that all symbols from one of the tapes must be read.
- final states are not necessarily disjoint from the C_i, and the machine may make transitions from them.

It is straightforward to see that any language *with end-markers* accepted by one form of automaton is accepted by one in the alternate form. On the one hand, given an automaton of the form of [26], we may obtain a disjoint final state automaton by adding an operation which, as soon as it reads an end-marker in one tape in a final state, systematically consumes all of the remaining tapes and then halts in a new (disjoint) final state. On the other hand, given a disjoint final state automaton, we may add states to record which tapes have reached their end markers, and move into a final state only when all end-markers have been read.

3.1 2-tape FSA as SCI terms

There is a simple correspondence between multitape FSA and SCI terms at first-order types, based on the fact that the denotations of the latter are generated from sets of tuples of words over a finite alphabet. First, for any n-tape, deterministic, disjoint final state FSA α over the alphabet $\{0, \ldots, m-1\}$, we show that we may define a term $x_1 : \underline{m}, \ldots, x_n : \underline{m} \vdash \mathtt{new}\, \lambda s.M_\alpha : \underline{1}$ such that $\langle t_1, \ldots, t_n\rangle \in \mathcal{L}(\alpha) \Leftrightarrow \langle\langle t_1, \ldots, t_n\rangle, 0\rangle \in [\![M_\alpha]\!]$. We give the 2-tape case here, and since every deterministic n-tape FSA is equivalent to one with no ε

transitions, we assume C_0 is empty.

Definition 3.2 *We number the states of* α *as* $0, \ldots, k-1$*, with* 0 *being initial. We assume terms* $\mathtt{state} : \underline{k} \multimap \underline{3}$ *such that* $[\![\mathtt{state}\,\mathtt{n}]\!] = 0$ *if* $s_n \in F$ *and* $[\![\mathtt{state}\,\mathtt{n}]\!] = i$ *if* $s_n \in C_i$*, and* $\mathtt{tr} : \underline{k} \multimap \underline{k}^{\underline{m}}$ *such that* $[\![\pi_i\,(\mathtt{tr}\,j)]\!] = \delta(i,j)$*.* M_α *may then be defined as follows:*

$$\mathtt{while0}\,(\mathtt{case}\,\langle \mathtt{state}\,!s, \langle \mathtt{1}, s := \mathtt{case}\langle x_1, \mathtt{tr}\,!s\rangle, s := \mathtt{case}\langle x_2, \mathtt{tr}\,!s\rangle\rangle\rangle)$$

Proposition 3.3 $\langle t_1, t_2\rangle \in \mathcal{L}(\alpha)$ *if and only if* $\langle\langle t_1, t_2\rangle, 0\rangle \in [\![\mathtt{new}\,\lambda s.M_\alpha]\!]$.

Proof: We prove by induction on i that for any $t_1, t_2 \in \lceil m \rceil^*$ with combined length i, $(\langle t_1, t_2\rangle, s_j) \in A_f(\alpha, i)$ for some f if and only if there exists $u \in \mathsf{cell}_{k,j}$ such that $\langle\langle t_1, t_2, u\rangle, 0\rangle \in [\![M_\alpha]\!]$.

For the induction case, suppose $\langle\langle t_1, t_2\rangle, s_j\rangle \in A_f(\alpha, i+1)$. If $s_j \in C_1$ then $t_1 = mt_1'$ for some m, t_1' and $\delta(m, s_j) = s_a$ for some s_a such that $(s_l, \langle t_1', t_2\rangle) \in A_0(\alpha, i)$. So by hypothesis there exists $u \in \mathsf{cell}_{k,a}$ such that $\langle\langle t_1', t_2, u\rangle, 0\rangle \in [\![M_\alpha]\!]$. Let $u' = j^l 0^{ar} u$. Then $u' =\in \mathsf{cell}_{k,j}$ and $\langle\langle t_1, t_2, u'\rangle, 0\rangle = \langle\langle mt_1', t_2, u'\rangle, 0\rangle \in [\![\mathtt{case}\langle(\mathtt{case}\,\langle \mathtt{state}\,!s, \langle \mathtt{1}, s := \mathtt{case}\langle x_1, \mathtt{tr}\,!s\rangle, s := \mathtt{case}\langle x_2, \mathtt{tr}\,!s\rangle\rangle\rangle), \langle M_\alpha, 0\rangle\rangle]\!] = [\![M_\alpha]\!]$ as required.

The converse — if there exists $u \in \mathsf{cell}_{k,j}$ such that $\langle\langle t_1, t_2, u\rangle, 0\rangle \in [\![M_\alpha]\!]$ then $(\langle t_1, t_2\rangle, s_j) \in A_f(\alpha, i)$ for some f — is similar. $\square$

So we may establish undecidability of inclusion of denotations in the fully abstract model, and hence of observational approximation in SCI via the following result, which may be proved via an encoding of Post's correspondence problem.

Proposition 3.4 *[11] Inclusion of languages accepted by deterministic 2-tape FSA is undecidable.*

(This extends to disjoint-final state automata because $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\alpha')$ if and only if $\mathcal{L}^e(\alpha) \subseteq \mathcal{L}^e(\alpha')$.)

Corollary 3.5 *Observational approximation in SCI is undecidable at first-order.*

Since $M \lesssim N$ if and only if $M\,\mathtt{or}\,N \simeq N$, this entails that observational equivalence (w.r.t. may-testing) in SCI with erratic choice is undecidable at first order.

The situation with respect to *equivalence* of multitape automata (and hence, as we shall show, observational equivalence of SCI) is different.

Theorem 3.6 (Harju and Karhumäki [8]) *Equivalence of n-tape deterministic finite state automata is decidable for all n.*

This includes disjoint finite finite state automata: for any automaton over the alphabet Σ (with $e \notin \Sigma$) there exists an automaton α' over $\Sigma \cup \{e\}$, such that $\mathcal{L}(\alpha) = \mathcal{L}^e(\alpha')$. considered here, as we may deduce from the following lemma.

To use this result to show decidability of SCI equivalence, we prove a converse to Proposition 3.3: the denotation of every first-order term corresponds to the language accepted by a disjoint final state automaton.

Since basic SCI is a subsystem of the simply-typed λ-calculus with products (and suitable constants), closed under β and π reduction, it follows from strong normalization of the latter that every term is reducible to a $\beta\pi$-normal form (i.e. one which contains no subterms of the form $(\lambda x.P)\,Q$ or $\pi_i\,\langle M, N\rangle$) to which it is denotationally equivalent by soundness of these reductions.

For each first-order $\beta\pi$-normal term $x_1 : T_1, \ldots, x_n : T_n \vdash M : \underline{m}$ we define a n-tape FSA α_M over the alphabet $[\![T_1]\!] \cup \ldots \cup [\![T_n]\!]$ with final states $F = \{f_0, \ldots, f_{m-1}\}$ which accepts $\langle s_1, \ldots, s_n\rangle$ in final state f_i if and only if $\langle\langle s_1, \ldots, s_n\rangle, i\rangle \in [\![M]\!]$.

Definition 3.7 *We define $\alpha^M = (C_0^M, \ldots, C_n^M, F, s_0^M, \delta^M)$ as follows:*

- *If $M = \mathtt{k}$, then $C_i^M = \varnothing$ for each i, $s_0^M = f_k$ and $\delta^M = \varnothing$.*
- *If $M = x_k$ or $M = \pi_j\, x_k$ then $C_k^M = \{s\}$, $C_i^M = \varnothing$ for $i \neq k$, $s_0^M = s$, $\delta^M(l^j, s) = f_l$ and $\delta^M(m, s)$ is undefined otherwise.*
- *If $M = \mathtt{case}\,\langle L, N_0, \ldots, N_{k-1}\rangle$ then $C_i^M = C_i^L + (C_i^{N_0} + \ldots + C_i^{N_{k-1}})$ for each i, $s_0^M = (s_0^L)^l$ and*
  $\delta^M(m, s^l) = (s_0^{N_i})^{ir}$ if $\delta^L(m, s) = f_i$,
  $\delta^M(m, s^l) = (\delta^L(m, s))^l$ otherwise,
  $\delta^M(m, s^{ir}) = f_j$ if $\delta^{N_i}(m, s) = f_j$,
  $\delta^M(m, s^{ir}) = (\delta^{N_i}(m, s))^{ir}$ otherwise.
- *If $M = \mathtt{while0}\,N$ then $C_i^M = C_i^N$ for each i, $s_0^M = s_0^N$,*
  $\delta^M(m, s) = s_0^N$ if $\delta^N(m, s) = f_0$
  $\delta^M(m, s) = \delta^N(m, s)$, otherwise.
- *If $M = \mathtt{new}\,\lambda x_j : \mathtt{var}[n].N$, then $C_0^M = (C_0^N \cup C_j^N) \times \lceil n \rceil$ and $C_k^M = C_k^M \times \lceil n \rceil$ for $k \notin \{0, j\}$, $s_0^M = \langle s_0^N, 0\rangle$ and*
  $\delta^M(m, \langle s, i\rangle) = \langle s', i\rangle$ if $m = \varepsilon$, $s \in C_j^N$ and $\delta^N(i^l, s) = s'$,
  $\delta^M(m, \langle s, i\rangle) = \langle s', k\rangle$ if $m = \varepsilon$, $s \in C_j^N$ and $\delta^N(0^{kr}, s) = s'$,

$\delta^M(m, \langle s, i\rangle) = \langle \delta^N(m, s), i\rangle$, *otherwise.*

Lemma 3.8 *For any M, α^M is well-defined.*

PROOF: The only non-trivial aspect of well-definedness is determinacy — i.e. that δ^M is a well-defined partial function, according to the above definition. We show by structural induction on M both that δ^M is a well-defined partial function, and that if $T_i = \underline{m_1}\& \ldots \&\underline{m_k}$ then for any state $s \in C_i^M$, if $\delta^M(v^i, s)$ and $\delta^M(u^j, s)$ are both defined then $i = j$.

The key case is new variable declaration $M = \mathtt{new}\,\lambda x_j : \mathtt{var}[n].N$. Suppose $\delta^N(0^{k'}r, s) = s'$ and $\delta^N(0^{k''}r, s) = s''$ for some $s \in C_j^N$, so that $\delta^M(\varepsilon, \langle s, i\rangle) = \langle s', k'\rangle = \langle s'', k''\rangle$. But by hypothesis, we have $k' = k''$ and hence $s' = s''$. Similarly, it is not possible to have $\delta^N(0^k r, s)$ defined and $\delta^N(i^l, s) = s'$ both defined. $\square$

Proposition 3.9 *α_M accepts $\langle t_1, \ldots, t_n\rangle$ in state f_i iff $\langle\langle t_1, \ldots, t_n\rangle, i\rangle \in [\![M]\!]$.*

PROOF: By structural induction on M. We give the case for $M = \mathtt{new}\,\lambda x_{n+1} : \mathtt{var}[m].N$. We show by induction on j that $(\langle t_1, \ldots, t_n\rangle, \langle s, k\rangle) \in A_i(\alpha_M, j)$ if and only if there exists $t_{n+1} \in \mathsf{cell}_{m,k}$ such that $(\langle t_1, \ldots, t_n, t_{n+1}\rangle, s) \in A_i(\alpha_N, j)$.

Suppose $(\langle t_1, \ldots, t_n\rangle, \langle s, k\rangle) \in A_i(\alpha_M, j+1)$. If $\langle s, k\rangle \in C_0^M$ then there exists $\langle s', l\rangle$ such that $\delta^M(\varepsilon, \langle s, k\rangle) = \langle s', l\rangle$ and $(\langle t_1, \ldots, t_n\rangle, \langle s', l\rangle) \in A_i(\alpha_M, j)$, and so by induction hypothesis there exists $t_{n+1} \in \mathsf{cell}_{m,l}$ such that $(\langle t_1, \ldots, t_n, t_{n+1}\rangle, s') \in A_i(\alpha_N, j)$. By definition of α^M, either $s \in C_0^N$ — in which case $k = l$ and $\delta^N(\varepsilon, s) = s'$, and so $(\langle t_1, \ldots, t_n, t_{n+1}\rangle, s) \in A_i(\alpha_N, j+1)$ as required — or else $s \in C_{n+1}^N$, in which case either $\delta^N(k^l, s) = s'$ and $l = k$ — and so $(\langle t_1, \ldots, t_n, k^l t_{n+1}\rangle, s) \in A_i(\alpha_N, j+1)$ — otherwise $\delta((0^l)^r, s) = s'$ and therefore $(\langle t_1, \ldots, t_n, (0^l)^r t_{n+1}\rangle, s) \in A_i(\alpha_N, j+1)$ as required.

If $\langle s, k\rangle \in C_i^M$ for some $i > 0$, then $t_i = mt_i'$, where $\delta^M(\langle s, k\rangle) = \langle s', l\rangle$ for some s' such that $(\langle t_1, \ldots, t_i', \ldots, t_n\rangle, \langle s', l\rangle) \in A_i(\alpha_M, j)$. By definition of α_M, $k = l$ and $\delta^N(m, s) = s'$. By induction hypothesis, there exists $t_{n+1} \in \mathsf{cell}_{m,k}$ such that $(\langle t_1, \ldots, t_i', \ldots, t_n, t_{n+1}\rangle, s') \in A_i(\alpha_N, j)$ and so $(\langle t_1, \ldots, t_n, t_{n+1}\rangle, s) \in A_i(\alpha_N, j+1)$ as required.

The converse — if there exists $t_{n+1} \in \mathsf{cell}_{m,k}$ such that $(\langle t_1, \ldots, t_n, t_{n+1}\rangle, s) \in A_i(\alpha_N, j)$ then $(\langle t_1, \ldots, t_n\rangle, \langle s, k\rangle) \in A_i(\alpha_M, j)$ — is proved similarly.

Hence M accepts $\langle t_1, \ldots, t_n\rangle$ in final state f_i if and only if there exists $t_{n+1} \in \mathsf{cell}_{m,0}$ such that N accepts $\langle t_1, \ldots, t_n, t_{n+1}\rangle$ in state f_i. By induction hypothesis, N accepts $\langle t_1, \ldots, t_n, t_{n+1}\rangle$ in state f_i if and only if $\langle\langle t_1, \ldots, t_n, t_{n+1}\rangle, i\rangle \in [\![N]\!]$ and so M accepts $\langle t_1, \ldots, t_n\rangle$ in final state f_i if and only if $\langle\langle t_1, \ldots, t_n\rangle, i\rangle \in [\![M]\!]$ as required. $\square$

Corollary 3.10 *Observational equivalence in basic SCI is decidable at first-order.*

4 Full Abstraction and Decidability at Higher Types

At higher types, denotations no longer consist of tuples of words and so we cannot decide observational equivalence between terms at these types simply by constructing multitape automata which recognize their denotations. However, we will show that we can associate a first-order term (and hence a multitape automaton) to each higher-order term in a way which reflects observational equivalence, which is therefore decidable at all types.

Definition 4.1 *A definable monomorphism between types S and T is a term* $\mathtt{mono} : S \multimap T$ *such that for all* $e : I \to S$, $e; [\![\mathtt{mono}]\!] = e'; [\![\mathtt{mono}]\!]$ *implies* $e = e'$.

Lemma 4.2 *If (equational) full abstraction holds at type T, and there is a definable monomorphism from S to T then full abstraction holds at type S.*

Proof: For any $M, N : S$, if $M \simeq N$ then $\mathtt{mono}\, M \simeq \mathtt{mono}\, N$ and so $[\![M]\!]; [\![\mathtt{mono}]\!] = [\![\mathtt{mono}\, M]\!] = [\![\mathtt{mono}\, N]\!] = [\![N]\!]; [\![\mathtt{mono}]\!]$. Hence $[\![M]\!] = [\![N]\!]$ as required. □

(Since morphisms in our model preserve all joins, this lemma extends to inequational full abstraction: if $M \lesssim N$ then $[\![M]\!]; [\![\mathtt{mono}]\!] \subseteq [\![N]\!]; [\![\mathtt{mono}]\!]$ and so $([\![M]\!] \cup [\![N]\!]); [\![\mathtt{mono}]\!] = ([\![M]\!]; [\![\mathtt{mono}]\!]) \cup ([\![N]\!]; [\![\mathtt{mono}]\!]) = [\![N]\!]; [\![\mathtt{mono}]\!]$. So $[\![M]\!] \cup [\![N]\!] = [\![N]\!]$ and so $[\![M]\!] \subseteq [\![N]\!]$.)

Lemma 4.3 *If equivalence is decidable at type T, and there is a definable monomorphism from S to T, then equivalence is decidable at type S.*

Proof: For any $M, N : S$ we have $M \simeq N$ iff $[\![M]\!] = [\![N]\!]$ iff $[\![M]\!]; [\![\mathtt{mono}]\!] = [\![N]\!]; [\![\mathtt{mono}]\!]$ iff $\mathtt{mono}\, N \simeq \mathtt{mono}\, N$. □

Thus we can prove that equivalence is decidable at any type S by showing that there is a definable monomorphism from S to some first-order type.

First, we observe that we may restrict attention to *functional* types — i.e. types which do not contain any instances of the product.

Definition 4.4 *A* definable retraction *between types S and T is a pair of terms* $(\mathtt{in} : S \multimap T, \mathtt{out} : T \multimap S)$ *such that* $[\![\lambda x.\mathtt{in}\,(\mathtt{out}\, x)]\!] = [\![\lambda x.x]\!]$. *We write* $S \trianglelefteq T$ *if such a retraction exists.*

Clearly, if $(\mathtt{in}, \mathtt{out})$ is a definable retraction, then $\mathtt{in}$ is a definable monomorphism. We also note that given definable retractions $(\mathtt{in_1}, \mathtt{out_1})$ from S_1 to T_1 and $(\mathtt{in_2}, \mathtt{out_2})$ from S_2 to T_2, $(\lambda f.\lambda x.\mathtt{in_2}\,(f\,(\mathtt{out_1}\,x)), (\lambda f.\lambda x.\mathtt{out_2}\,(f\,(\mathtt{in_1}\,x)))$ and $\lambda x.\langle \mathtt{in_1}\,(\pi_1 x), \mathtt{in_2}\,(\pi_2 x)\rangle, \lambda x.\langle \mathtt{out_1}\,(\pi_1 x), \mathtt{out_2}\,(\pi_2 x)\rangle)$ are definable retractions from $S_1 \multimap S_2$ to $T_1 \multimap T_2$ and from $S_1 \& S_2$ to $T_1 \& T_2$ respectively.

Proposition 4.5 *Given functional types S, T there is a functional type* $\mathrm{prod}(S,T)$ *(of order* $\max\{o(S), o(T), 1\}$*) such that $S \& T$ is a definable retract of* $\mathrm{prod}(S,T)$.

Proof: is by induction on $o(S)+o(T)$. For the base case, suppose $S = \underline{m}$ and $T = \underline{n}$. We define $\mathrm{prod}(S,T) =_{df} \underline{2} \multimap \underline{\max\{m,n\}}$, $\mathtt{in} = \lambda x.\lambda y.\mathtt{case}\,\langle \mathsf{in}_m\,y, \mathsf{in}_n\,x\rangle$ and $\mathtt{out} = \lambda x.\langle \mathsf{proj}_m\,(x\,\mathtt{0}), \mathsf{proj}_n\,(x\,\mathtt{1})\rangle$, where $\mathsf{in}_n = \lambda x.\mathtt{case}\langle x, \langle \mathtt{0}, \ldots, \mathtt{n}\rangle\rangle$ and $\mathsf{proj}_n = \lambda x.\mathtt{case}\langle x, \langle \mathtt{0}, \ldots, \mathtt{n}, \Omega\rangle\rangle$.

For the induction case, suppose $T = U \multimap V$. Then let $\mathrm{prod}(S,T) = U \multimap \mathrm{prod}(S,V)$: $S \& T \trianglelefteq (U \multimap S) \& (U \multimap V) \cong U \multimap (S \& V) \trianglelefteq U \multimap \mathrm{prod}(S,V) =_{df} \mathrm{prod}(S,T)$. If T is a base type and $S = U \multimap V$ then $\mathrm{prod}(S,T) =_{df} U \multimap \mathrm{prod}(V,T)$. $\square$

Corollary 4.6 *For every type T there exists a functional type $\overline{T}$ of order at most $o(T)+1$ with a definable retraction from T to $\overline{T}$.*

We now define monomorphisms from higher to lower-order functional types, using the observation that we may represent a sequence of tuples of sequences $\langle \vec{s}_{11}, \ldots, \vec{s}_{1n}\rangle \ldots \langle \vec{s}_{k1}, \ldots, \vec{s}_{kn}\rangle$ as a tuple of sequences $\langle \vec{s}_{11} @ \ldots @ \vec{s}_{k1}, \ldots, \vec{s}_{1n} @ \ldots @ \vec{s}_{kn}\rangle$ (uniquely), where @ is a symbol not occurring in any of the $\vec{s}_{ij}$.

For any types $S, T, \underline{n}$, we now define a monomorphism $\mathtt{mono1}$ from $(\underline{n} \multimap S) \multimap T$ to $(\underline{n+1}) \multimap S \multimap T$. Recalling the equality test $\mathtt{eq} : \underline{n} \times \underline{n} \to \underline{1}$, let

$$\mathtt{mono1} = \lambda f.\lambda x.\lambda y.f\,(\lambda z.(\mathtt{while0}\,(\mathtt{case}\,\langle x, \langle \mathtt{eq}\,\langle \mathtt{0}, z\rangle, \ldots, \mathtt{eq}\,\langle \mathtt{n}-\mathtt{1}, z\rangle, \mathtt{1}\rangle\rangle)); y)$$

Lemma 4.7 $\mathtt{mono1}$ *is a definable monomorphism.*

Proof: $x : \underline{n+1}, y : S, z : \underline{n} \vdash \mathtt{while0}\,(\mathtt{case}\,\langle x, \langle \langle \mathtt{eq}\,\langle \mathtt{i}, z\rangle \mid i < n\rangle, \mathtt{1}\rangle\rangle); y : S$ evaluates by reading a value v from x — followed by a value u from z, and then diverges if $v \neq u$ — until $v = n$, when a value is read from y and returned as a result. So it has the denotation:

$$\{\langle \langle \vec{j} n, e, \vec{j}\rangle, e\rangle \mid \vec{j} \in [\![\underline{n}]\!]^* \wedge e \in [\![S]\!]\}.$$

Thus the denotation of $\mathtt{mono1}$ relates each element $\langle \langle \vec{j}_1, s_1\rangle \ldots \langle \vec{j}_m, s_m\rangle, t\rangle$ to the *unique* element $\langle \vec{j}_1 n \ldots n \vec{j}_m n, \langle s_1 \ldots s_n, t\rangle\rangle$, and is therefore a monomorphism. $\square$

For any types S, T, U, V, we now define a monomorphism $\mathtt{mono2}$ from $((S \multimap T) \multimap U) \multimap V$ to $(S \& \underline{1}) \multimap (T \multimap U) \multimap V$:

$$\mathtt{mono2} = \lambda f.\lambda x.\lambda g.f\ \lambda y.(g\,(\pi_2(x); (y\,\pi_1(x))))$$

Lemma 4.8 $\mathtt{mono2}$ *is a definable monomorphism.*

Proof: The term $x : S \& \underline{1}, y : S \multimap T \vdash (\pi_2\, x); (y\,(\pi_1\, x)) : T$ has the denotation

$$\{\langle\langle 0^r \vec{s^l}, \langle \vec{s}, e\rangle, e\rangle \mid \vec{s} \in [\![S]\!]^* \wedge e \in [\![T]\!]\}$$

So denotation of $x : S \& \underline{1}, y : S \multimap T, g : T \multimap U \vdash g\,(\pi_2(x); (y\,\pi_1(x))) : U$ relates each element $\langle 0^r \vec{s_1^l} 0^r \ldots 0^r \vec{s_n^l}, \langle \vec{s_1}, e_1\rangle \ldots \langle \vec{s_n}, e_n\rangle, \langle e_1, \ldots e_n, u\rangle\rangle$ to u for $\vec{s_1}, \ldots, \vec{s_n} \in [\![S]\!]^*$, $e_1, \ldots, e_n \in [\![T]\!]$ and $u \in [\![U]\!]$.

Thus $[\![\mathtt{mono2}]\!]$ relates each element:

$$\langle\langle s_{11}, e_{11}\rangle \ldots \langle s_{1m_1}, e_{1m_1}\rangle, u_1\rangle \ldots \langle\langle s_{n1}, e_{n1}\rangle \ldots \langle s_{nm_n}, e_{nm_n}\rangle, u_n\rangle, v\rangle$$

to the *unique* element:

$$\langle 0^r s^l_{11} \ldots 0^r s^l_{1m_1} \ldots 0^r s^l_{n1} \ldots 0^r s^l_{nm_n}, \langle\langle e_{11} \ldots e_{1m_1}, u_1\rangle \ldots \langle e_{n1} \ldots e_{nm_n}, u_n\rangle, v\rangle\rangle$$

and is therefore a monomorphism. $\square$

Proposition 4.9 *For any functional type R of order at most $n+2$ there is a functional type $\widehat{R}$ of order at most $n+1$ and a definable monomorphism from R to $\widehat{R}$.*

Proof: By structural induction on R. For the induction step, if $n = 0$ then if R has order 2 it is isomorphic to a type of the form $(\underline{m} \multimap S) \multimap T$. By Lemma 4.7 there is a definable monomorphism $\mathtt{mono}_1$ from R to $\underline{m+1} \multimap S \multimap T$. By induction hypothesis there is a definable monomorphism $\mathtt{mono}_2$ from $S \multimap T$ to $\widehat{S \multimap T}$, and hence $\lambda x.\lambda y.\mathtt{mono}_2\,((\mathtt{mono}_1\, x)\, y)$ is a definable monomorphism from R to $\underline{m+1} \multimap \widehat{S \multimap T}$.

If $n > 0$ then R is isomorphic to a type of the form $((S \multimap T) \multimap U) \multimap V$, where $o(S) \leq n$. By Lemma 4.8 there is a definable monomorphism from R to $S \& \underline{1} \multimap (T \multimap U) \multimap V$ and hence — using the induction hypothesis — to $\widehat{S \& \underline{1}} \multimap \widehat{(T \multimap U)} \multimap V$. $\square$

Hence we may prove the following by a simple induction on order.

Proposition 4.10 *For every type of SCI there exists a definable monomorphism into a first-order type.*

By applying Proposition 2.9 and Lemma 4.2 we may now prove McCusker's theorem.

Theorem 4.11 *The relational model of basic SCI is fully abstract.*

Hence by Corollary 3.10 and Lemma 4.3.

Theorem 4.12 *Observational equivalence in finitary basic SCI is decidable.*

5 Observably Sequential SCI

We will now show that observational approximation is decidable in an "observably sequential" version of SCI containing a simple form of non-local control in the form of backwards jumps to labelled program points. (This breaks the equivalence between parallel and sequential composition which holds in SCI — in particular, jumps may be regarded as a way for one thread of a parallel composition to interfere with the other by terminating it.)

We form SSCI by extending the syntax of SCI with a single constant $\mathtt{label} : (\underline{0} \multimap \underline{0}) \multimap \underline{1}$. This acts as a handler for catching (statically bound) exceptions: if the variable $k : \underline{0}$ is encountered in evaluating M then $\mathtt{label}\,\lambda k.M$ evaluates to $\mathtt{0}$.

To define the operational semantics of SSCI, we extend the notion of evaluation context as follows:

$$\begin{aligned} E[\_] ::= \quad & [\_] \mid E[\_]\,M \mid \pi_i\,E[\_] \mid \mathtt{case}\,E[\_] \mid \mathtt{case}\,\langle E[\_], M\rangle \mid \\ & \mathtt{while0}\,E[\_] \mid \mathtt{label}\,E[\_] \mid \mathtt{label}\,\lambda k.E[\_] \end{aligned}$$

Writing $E_k[\_]$ for an evaluation context which does not bind k, and assuming that all substitutions are capture-avoiding, we extend the reduction rules for terms in environments (Table 1) with the following rule for `label`:

$$E[\mathtt{label}\,\lambda k.E'_k[k]], \mathcal{E} \longrightarrow E[0], \mathcal{E}$$

We may use labelled jumps to distinguish SCI-equivalent terms such as $\lambda x.\lambda y.x; y$ and $\lambda x.\lambda y.y; x$ (for example, taking $C[\_] = \mathtt{label}\,k.((\_]\,k)\,\Omega)$, we have $C[\lambda xy.x; y] \Downarrow$ and $C[\lambda xy.y; x] \not\Downarrow$).

More generally, using imperative variables we can express Cartwright and Felleisen's `catch` operators [6]. For each n, $\mathtt{catch}_n : (T_0 \multimap \ldots \multimap T_{n-1} \multimap \underline{m}) \multimap \underline{m+n}$ returns `i` if its argument is strict in its ith argument, or $\mathtt{n} + \mathtt{j}$ if it is constant with value j. We define $\mathtt{catch}_0 = \lambda x.x$, and $\mathtt{catch}_{n+1} =$

$$\lambda f.\mathtt{new}\,\lambda x.(\mathtt{label}\,\lambda k.(x := ((\mathtt{catch}_n\,(f\,k)) + 1); k); !x).$$

We will now give a fully abstract games model of observably sequential SCI, and show that denotations may be represented as regular languages. It is based on Abramsky and McCusker's semantics of Idealized Algol [3] (using Hyland-Ong style dialogue games [10]). We allow the interpretation of backwards jumps by abandoning the notion of questions and answers — and hence the "bracketing condition" — as in [12].

Since only one thread of computation relating to each argument may be "open" at a time in SSCI programs, we may omit explicit justification pointers from our model (as proposed by Abramsky for "serially re-entrant Idealized Algol"[2]). As in McCusker and Wall's semantics of SCI [20] (and the author's semantics of linearly used continuations [15]) we add an equivalence relation $\sim$ "interference" to the notion of HO-arena in order to indicate which moves may not occur in the same thread.

Definition 5.1 *A SCI* arena *is a directed acyclic graph* $(M_A, \vdash_A)$ *— in the form of a set of nodes or* moves M_A*, and a set of directed edges (or* enabling relation*)* $\vdash_A \subseteq M_A \times M_A$ *— with a labelling function* $\lambda_A : M_A \to \{P, O\}$*, partitioning the moves between Player and Opponent, and an equivalence relation* $\sim \subseteq M_A \times M_A$ *such that:*

- *If* $m \vdash n$ *then* $\lambda(m) = \overline{\lambda(n)}$.
- *If* $m \sim m'$ *then* $\lambda(m) = \lambda(n)$.
- *If* $m \vdash n$ *and* $m' \vdash n$ *then* $m \sim m'$.

We write M_A^I *for the set of root nodes of* $(M_A, \vdash_A)$ *(the "initial moves"). A is* negative *if every move in* M_A^I *is an Opponent move.*

We write $s \sqsubseteq t$ for "s is a prefix of t" (and $s \sqsubseteq^E t$ if s is an even prefix of t). For any sequence s of moves, we define a subsequence $\mathsf{open}(s)$ — the "stack of open moves" — containing at most one move from each $\sim$ equivalence class.

- $\mathsf{open}(\varepsilon) = \varepsilon$,
- $\mathsf{open}(sm) = tm$, if there exists $tm' \sqsubseteq \mathsf{open}(s)$ such that $m' \sim m$, (we say that m *closes* m').
- $\mathsf{open}(sm) = \mathsf{open}(s)m$, otherwise.

It follows from this definition that the stack of open moves contains at most one representative of each $\sim$-equivalence class (and so at most one enabler for any given move). A stack of open moves is a sequence with this property. Following [10,17], we define the *view* $\ulcorner s \urcorner$ of a stack of open moves s as follows:[3]:

[3] We do not distinguish Player and Opponent views in this notation — the view of

- $\ulcorner \varepsilon \urcorner = \varepsilon$
- $\ulcorner sm \urcorner = m$, if s contains no enabling move for m,
- $\ulcorner smtn \urcorner = \ulcorner s \urcorner mn$ if $m \vdash n$.

The *stack-view* of t is $\ulcorner \mathsf{open}(t) \urcorner$. An alternating sequence s satisfies the *stack-visibility* condition if:

- the stack-view at every non-initial move b in s contains a (unique) enabling move for b — the *implicit justifier* of b — i.e. if $tb \sqsubseteq s$ and b is non-initial then there exists $ra \sqsubseteq \ulcorner \mathsf{open}(t) \urcorner$ such that $a \vdash b$.
- if b closes a in s then a occurs in the stack-view at b — i.e. if $tb \sqsubseteq s$ and there exists $ra \sqsubseteq \mathsf{open}(t)$ such that $a \sim b$ then $\ulcorner r \urcorner a \sqsubseteq \ulcorner \mathsf{open}(t) \urcorner$.

The final constraint required to define our notion of *legal sequence* is a *non-interference condition*. We say that two (occurrences of) moves are co-justified if they have the same justifier, or are both initial.

Definition 5.2 *A sequence t satisfies the non-interference condition if for any $sb \sqsubseteq s'b' \sqsubseteq t$ and co-justified moves a, a' ocurring (respectively) in $\ulcorner \mathsf{open}(s) \urcorner$ and $\ulcorner \mathsf{open}(s') \urcorner$: if $b \sim b'$ and $\lambda(a) = \overline{\lambda(b)}$ then $a \sim a'$.*

Note that since the justifier of a and a' immediately precedes a in $\ulcorner \mathsf{open}(s) \urcorner$ and a' in $\ulcorner \mathsf{open}(s') \urcorner$, we have either $a = a'$ (in which case the condition holds trivially) or else it is the last move to occur in both views, and so a and a' are the first moves on which the views differ. In this case we shall say that a, a' are branching moves for s, s', and we may restate the non-interference condition equivalently thus: for any $sb \sqsubseteq s'b' \sqsubseteq t$ such that $b \sim b'$, if a, a' are the branching moves for s, s' then $a \sim a'$.

The force of the condition to prevent potentially interfering behaviour is will be discussed later (see Figure 1).

Definition 5.3 *The set L_A of legal sequences of the arena A consists of the finite, alternating sequences over A, starting with an Opponent move, and satisfying the stack-visibility and non-interference conditions.*

A sequence s is single-threaded if it contains at most one initial move. For any set of sequences S, we write $S^\natural$ for the set of single-threaded sequences in S.

We form a category in which the objects are negative SCI-arenas, and morphisms from A to B are strategies over the arena $A \to B$, which is defined:

$$A \to B = (M_A + M_B, [\overline{\lambda_A}, \lambda_B], (\vdash_A + \vdash_B), \sim_A + \sim_B)$$

(note that this is not a negative arena). As usual, a strategy for the arena

any sequence is that of the participant about to move.

A is specified as a non-empty, even-prefix closed subset of L_A satisfying the determinacy condition:

$$sa, sb \in \sigma \implies b = c$$

In addition, we follow [5] in requiring that strategies satisfy a *thread-independence* condition. We say that two moves are hereditarily co-justified if there is some move (which we may assume to be initial) which hereditarily justifies both of them. We define $\mathsf{thread}(s)$ for odd-length legal sequences $s \in L_{A \to B}$ as follows:

- $\mathsf{thread}(sa) = a$, if a is initial,
- $\mathsf{thread}(satb) = \mathsf{thread}(s)ab$, if a is the most recent move hereditarily co-justified with b.

If s is legal then $\mathsf{thread}(s)$ is a legal sequence by the visibility condition. A strategy $\sigma : A \to B$ is *thread-independent* if:

$$\text{If } sab, t \in \sigma,\ ta \in L_{A \to B} \text{ and } \mathsf{thread}(sa) = \mathsf{thread}(ta) \text{ then } tab \in \sigma.$$

Given $\sigma : A \to B$ and $\tau : B \to C$, let $\sigma ; \tau : A \to C =$

$$\{s \in (M_A + M_C)^* \mid \exists t \in (M_A + M_B + M_C)^*. t{\restriction}A, B \in \sigma \wedge t{\restriction}B, C \in \tau \wedge s = t{\restriction}A, C\}$$

(where $s \restriction A, B$ denotes the restriction of the sequence s to $M_A \cup M_B$). To prove that this is a well-defined strategy on $A \to C$, it suffices to show that it is a set of legal sequences, since preservation of determinacy and thread-independence are shown elsewhere [10,17,9]. Writing $I_{A,B,C}$ for the set of $t \in (M_A + M_B + M_C)^*$ such that $t{\restriction}A, B \in L_{A \to B}$ and $t{\restriction}B, C \in L_{B,C}$, it suffices to prove that $t{\restriction}A, C \in L_{A \to C}$ for all $t \in I_{A,B,C}$. To do this we adapt the notion of "core view" introduced by McCusker [17]. The (core) stack-view of $s \in I_{A,B,C}$ may be defined as follows:

- $\ulcorner \varepsilon \urcorner_c = \varepsilon$,
- $\ulcorner sm \urcorner_c = m$ if m is an initial move,
- $\ulcorner smtm't'n \urcorner_c = \ulcorner s \urcorner_c m'n$, if n is a move in A, C, and m' justifies n and closes m'.
- $\ulcorner smtn \urcorner_c = \ulcorner s \urcorner_c m'n$, if n is a move in A, C, and m' justifies n and does not close any move.
- $\ulcorner sm \urcorner_c = \ulcorner s \urcorner_c m$ if m is a move in B.

Lemma 5.4 *For any $sab \in I_{A,B,C}$:*

- $\ulcorner sab{\restriction}A, C \urcorner = \ulcorner sab \urcorner_c {\restriction} A, C$,
- *If b is a Player move in A, B then $\ulcorner \mathsf{open}(sa) \restriction A, B \urcorner$ is a subsequence of $\ulcorner sa \urcorner_c$. If b is a Player move in B, C then $\ulcorner \mathsf{open}(sa){\restriction}B, C \urcorner$ are subsequences of $\ulcorner sa \urcorner_c$.*

PROOF: The first part follows directly from the definition of core views.

We prove the second by induction on the length of sa. If a is a move in A or C, then the induction case follows directly from the definition of core view. So the key case is the one in which a is a move in B. If b is a player move in B, C then a is a Player move in A, B. Suppose $sa = rmr'm'r'a$, where m' justifies a and closes m. Then by the stack-visibility condition (first part), m' occurs in $\ulcorner \mathsf{open}(s{\restriction}A, B) \urcorner$ and hence by induction hypothesis in $\ulcorner t \urcorner$. Since m' is a Player move in B, C, by the stack-visibility condition (second part) m occurs in $\ulcorner rmr' {\restriction} B, C \urcorner$, and so $\ulcorner \mathsf{open}(r \restriction B, C) \urcorner \sqsubseteq \ulcorner \mathsf{open}(rmr' \restriction B, C) \urcorner$. By inductive hypothesis $\ulcorner r{\restriction}B, C \urcorner$ is a subsequence of $\ulcorner r \urcorner_c$, and hence of $\ulcorner t \urcorner_c$. So $\ulcorner sa{\restriction}B, C \urcorner = \ulcorner r{\restriction}B, C \urcorner ma$ is a subsequence of $\ulcorner t \urcorner_c$ as required. $\square$

Lemma 5.5 *If $t \in I_{A,B,C}$ then $t{\restriction}A, C$ satisfies stack-visibility.*

PROOF: Suppose $sa \sqsubseteq t$, where a is a (non-initial) Player move in C. Then by stack-visibility there is an enabling move for a in $\ulcorner \mathsf{open}(s{\restriction}B, C) \urcorner$ and hence by Lemma 5.4, in $\ulcorner s \urcorner_c$, and so by Lemma 5.4, in $\ulcorner \mathsf{open}(s{\restriction}A, C) \urcorner = \ulcorner s \urcorner_c {\restriction} A, C$. Similarly, if a closes a move a' then a' occurs in $\ulcorner s{\restriction}B, C \urcorner$ and hence in $\ulcorner s \urcorner_c$ and $\ulcorner \mathsf{open}(s{\restriction}A, C) \urcorner$. $\square$

The set of *generalized Player moves* of $s \in I_{A,B,C}$ consists of those moves which are Player moves in either $A \to B$ or $B \to C$ (i.e. Player moves of C, Opponent moves of A and all moves of B). Generalized Opponent moves are Opponent moves of $A \to C$. We shall say that $t \in I_{A,B,C}$ satisfies non-interference with respect to core views if for any generalized Player moves b, b' with $sb \sqsubseteq s'b' \sqsubseteq t$ and co-justified generalized Opponent moves a, a' ocurring (respectively) in $\ulcorner s \urcorner_c$ and $\ulcorner s' \urcorner_c$: if $b \sim b'$ then $a \sim a'$. As in the original definition we may assume that a, a' are the branching moves for s and s'. (Note that they must be moves in A, C).

Lemma 5.6 *If $t \in I_{A,B,C}$ satisfies non-interference with respect to core views then $t{\restriction}A, C$ satisfies non-interference.*

PROOF: Suppose we have Player moves b, b' from $A \to C$ with $sb \sqsubseteq s'b' \sqsubseteq t$ and branching moves a, a' for $s \restriction A, C$ and $s' \restriction A, C$. Then a, a' are generalized Opponent moves and by by Lemma 5.4, they occur in $\ulcorner s \urcorner_c$ and $\ulcorner s' \urcorner_c$ (respectively) and so $b \sim b'$ implies $a \sim a'$ as required. $\square$

Lemma 5.7 *Every sequence $t \in I_{A,B,C}$ satisfies non-interference with respect to core views.*

PROOF: By induction on the length of t. Suppose we have generalized Player moves $b \sim b'$ with $sb \sqsubseteq s'b' = t$. Let a, a' be the branching moves for s, s'.

Suppose b, b' are Player moves in B, C. Let c, c' be the branching moves of

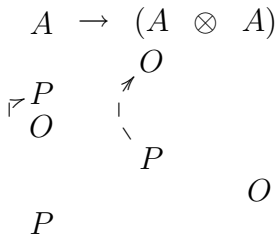

Fig. 1. A sequence violating the non-interference condition

$s{\restriction}B, C$ and $s'{\restriction}B, C$. Then $c \sim c'$ by the non-interference condition for $s{\restriction}B, C$. If c, c' are moves in C then they occur in $\ulcorner s \urcorner_c$ and $\ulcorner s' \urcorner_c$ by Lemma 5.4, and so by definition of core view they must also be the branching moves for s, s' — i.e. $a = c$ and $a' = c'$ and so we are done. If c, c' are moves in B then they are generalized Player moves of t, and so we have $rc, r'c' \sqsubset t$. Moreover, c and c' appear in $\ulcorner s \urcorner_c$ and $\ulcorner s' \urcorner_c$ (respectively) by Lemma 5.4. Thus $\ulcorner r \urcorner_c \sqsubseteq \ulcorner s \urcorner_c$ and $\ulcorner r' \urcorner_c \sqsubseteq \ulcorner s' \urcorner_c$ and so a, a' are the branching moves for r, r'. So we may apply the induction hypothesis to obtain $c \sim c'$ as required. $\square$

The proof of associativity of composition is standard, as is the existence of the standard "copycat" identities. Thus we have defined a category $\mathcal{G}$ of arenas and strategies. We may define Cartesian and symmetric monoidal (multiplicative) products on $\mathcal{G}$ as in [15]: initial moves of A interfere with those from B in $A\&B$ but not in $A \otimes B$.

- $A\&B = (M_A + M_B, [\lambda_A, \lambda_B], \vdash_A + \vdash_B, (\sim_A + \sim_B) \cup ((M_A^I)^l \times (M_B^I)^r) \cup ((M_B^I)^l \times (M_A^I)^r))$
- $A \otimes B = (M_A + M_B, [\lambda_A, \lambda_B], (\vdash_A + \vdash_B), \sim_A + \sim_B)$

From $\sigma : A \to B, \tau : A \to C$, we may form $\langle \sigma, \tau \rangle =$
$\{s \in L_{A \to B\&C} \mid \forall t \sqsubseteq^E s.\mathsf{thread}(t{\restriction}A, B) \in \sigma \vee \mathsf{thread}(t{\restriction}A, C) \in \tau\}$ and from $\sigma : A \to B, \tau : C \to D$, we may form $\sigma \otimes \tau =$
$\{s \in L_{A \otimes C \to B \otimes D} \mid \forall t \sqsubseteq^E s.t \restriction A, B \in \sigma \wedge t \restriction C, D \in \tau\}$. The unit I for both products is the empty game. In conjunction with the copycat natural isomorphisms and projections, we have:

Lemma 5.8 *$(\mathcal{G}, I, \otimes)$ is a SMC with a Cartesian product, &.*

Say that an SCI-arena A is *well-opened* if $m, m' \in M_A^I$ implies $m \sim m'$. Observe that if A is well-opened then the non-interference condition prevents us from forming the partial "diagonal" strategy from A to $A \otimes A$ which plays copycat between A and both components of $A \otimes A$ as long as it can do so without interleaving threads. This is illustrated in Figure 1, in which Player plays interfering moves in A in response to non-interfering moves in $A \otimes A$. As in the case of the relational model, the SMC $(\mathcal{G}, I, \otimes)$ is not symmetric monoidal closed, but does have exponentials by well-opened arenas, allowing us to interpret the types of SCI.

Lemma 5.9 *$\mathcal{G}$ has (well-opened) exponentials by all well-opened arenas.*

PROOF: If C is well-opened, then for any B we may define a negative, well-opened arena $B \multimap C$ as for $B \to C$, except that the initial moves of C enable the initial moves of B — i.e.:

$$B \multimap C = (M_B + M_C, [\overline{\lambda_B}, \lambda_C], (\vdash_B + \vdash_C) \cup ((M_B^I)^l \times (M_C^I)^r), \sim_B + \sim_C).$$

There is an isomorphism between legal sequences on $A \otimes B \to C$ and on $A \to (B \multimap C)$ obtained by adding justification pointers from each initial move in B to the first move in the current thread (which yields an isomorphism Λ between thread-independent strategies on $A \otimes B \to C$ and on $A \to (B \multimap C)$. $\Box$

Note that if A, B are well-opened then $A \& B$ is well-opened. Thus we may interpret each SCI type as a well-opened arena by setting the denotation of the ground type $\underline{n}$ to be the well-opened arena A_n with a single initial Opponent move q enabling n distinct Player moves $a_0, \ldots, a_{n-1}$.

Note that $A_0 \multimap A_0$ is isomorphic to A_1: we define the denotation of `label` to be this isomorphism. The remaining constants — the numerals, `case`, `while0` and `new` are interpreted as strategies essentially as in the game semantics of Idealized Algol [3]. It is straightforward to check that they satisfy the stack-visibility and non-interference conditions. In fact we may observe that, roughly speaking, our semantics is the restriction of Abramsky and McCusker's games model of Idealized Algol [3] to the implicitly justified sequences satisfying the non-interference condition: if we extend the latter with finitary expression types and products, then for any SCI type T, $L^{\natural}_{[\![T]\!]}$ is a subset of the (unbracketed) *plays* over the corresponding IA type-object. This correspondence extends to the semantics of terms: if we express `label`, `case` and `while0` as the corresponding macros in Idealized Algol + `catch`, then for any SCI term $M : T$, $[\![M]\!]^{\natural}_{\mathcal{G}} = [\![M]\!]_{IA_c} \cap L_{[\![T]\!]}$. Hence we may obtain soundness and adequacy results for our semantics with minimal modification to the proofs described in [3,4].

Proposition 5.10 *For any closed term $M : \underline{1}$, $M \Downarrow$ if and only if $[\![M]\!] \neq \bot$.*

5.2 Strategies as Finite State Automata

We now observe that denotational equivalence and approximation are decidable because the strategy denoting each term is a *regular language*. The key requirement is to show that the set of legal sequences over each SCI type-object is itself regular, as we may then follow [7] in showing that the composition of regular strategies is regular.

For each finite arena A, let $Op_A = \{\mathsf{open}(s) \mid s \in L_A\}$, which is a finite set. We will define a (single tape) finite state automaton α_A over the alphabet M_A which recognizes L_A, by defining the set of states to be $Op_A \times \mathcal{P}(M_A \times Op_A)$, and an initial state and transition function so that having read the legal sequence of moves r, α_A is in the state $\langle \mathsf{open}(r), X \rangle$, where $(m, u) \in X$ if and only if m occurs in $\mathsf{open}(r)$ and there is a prefix $v \sqsubseteq r$ such that the same occurrence of m appears in $\ulcorner \mathsf{open}(v) \urcorner = u$. This information is sufficent to determine the legality of the next move to be read.

Definition 5.11 *For each finite arena A, we define $\alpha_A = (Op_A \times \mathcal{P}(M_A \times Op_A), (\varepsilon, \varnothing), \delta_A)$, where:*
$\delta_A(b, (t, X)) = (\mathsf{open}(tb), X')$, if:

- *there exists $rc \sqsubseteq \ulcorner t \urcorner$ such that $c \vdash b$,*
- *if there exists $rb' \sqsubseteq s$ with $b \sim b'$ then $\ulcorner r \urcorner b' \sqsubseteq s$,*
- *if $(m, sb') \in X$ and $b \sim b'$ then if $rma \sqsubseteq s$ and $rma' \sqsubseteq \ulcorner t \urcorner$ then $a \sim a'$,*
- *$(m, u) \in X'$ if and only if ($m = b$ and $u = \ulcorner \mathsf{open}(tb) \urcorner$) or ($m \in |\mathsf{open}(tb)| - \{b\}$ and $(m, u) \in X$).*

Lemma 5.12 *The sequence s is accepted by α_A in state (t, X) if and only if:*

- *s is legal,*
- *$\mathsf{open}(s) = t$,*
- *If $ra \sqsubseteq t$, then $(a, u) \in X$ if and only if there exists $v \sqsubseteq s$, with $\ulcorner ra \urcorner \sqsubseteq u = \ulcorner \mathsf{open}(v) \urcorner$.*

Proof: This follows by induction on the length of s from the definition of α_A. $\square$

Corollary 5.13 *For any finitary SCI arena A, L_A is a regular language.*

We say that a strategy σ on a finite arena is regular if it is a regular language — i.e. there is a (deterministic) finite state automaton α^σ such that $\sigma = \mathcal{L}(\alpha^\sigma)$. We may use Corollary 5.13 to show that each of the copycat strategies which yield the symmetric monoidal and Cartesian structure of $\mathcal{G}$ are regular — for example the identity strategy $\mathsf{id}_A : A \to A$ is the intersection of the "copycat sequences" $(\Sigma_{m \in M_A} m^r m^l + m^l m^r)^*$ with $L_{A \to A}$. Following [7], we may show that the composition preserves regularity.

Proposition 5.14 *If σ, τ are regular strategies then $\sigma; \tau$ is a regular strategy.*

Proof: Assuming α^σ and α^τ are ε-free, we define the FSA $\alpha^{\sigma;\tau}$ by setting $S^{\sigma;\tau} = S^\sigma \times S^\tau$, $s_0^{\sigma;\tau} = (s_0^\sigma, s_0^\tau)$ and:
$\delta^{\sigma;\tau}(m, (s_1, s_2)) = (s_1', s_2)$, if $m \in M_A$ and $\delta^\sigma(m, s_1) = s'$,
$\delta^{\sigma;\tau}(m, (s_1, s_2)) = (s_1, s_2')$, if $m \in M_C$ and $\delta^\tau(m, s_2) = s_2'$,
$\delta^{\sigma;\tau}(\varepsilon, (s_1, s_2)) = (s_1', s_2')$, if there exists $m \in M_B$ such that $\delta^\sigma(m, s_1) = s_1'$ and

$\delta^\tau(m, s_2) = s_2'$. $\square$

The interpretations of the constants are regular strategies and thus:

Proposition 5.15 *The denotation of every term of finitary observably sequential SCI is a regular strategy.*

6 Full Abstraction

We will now prove that our interpretation of observably sequential SCI is (inequationally) fully abstract. As in the relational model of SCI, this is the case despite the fact that there are finitary strategies in the model which are not the denotations of terms. Moreover, these strategies may correspond to Idealized Algol terms which do intrinsically exhibit interference. For example, the strategy on the arena $(\underline{1} \multimap \underline{1} \multimap \underline{1}) \multimap \underline{2}$ which runs its argument f once, and returns `1` if the last argument tested by it was the leftmost one and 0 otherwise (and therefore sends $[\![\lambda x.\lambda y.x; y]\!]$ to 0, $[\![\lambda x.\lambda y.y; x]\!]$ to 1, and diverges on $[\![\lambda x.\lambda y.x; \Omega]\!]$ and $[\![\lambda x.\lambda y.y; \Omega]\!]$). To write this test in Idealized Algol requires that the two arguments supplied to f write to the same cell — as in $\lambda f.\mathtt{new}\, \lambda x.((f\,(x := 1))\, x := 0); !x.$, and so it cannot be expressed in SCI with or without `catch`.

As in the case of SCI, it is sufficient to establish full abstraction for functional types.

Lemma 6.1 *Every type T is a definable retract of the functional type $\overline{T}$*

Proof: As for Proposition 4.5. $\square$

We further simplify the set of types for which we need to prove full abstraction by reduction to the *zero* types: functional types generated from the empty type $\underline{0}$. Define the type $\underline{0}_k$ for each $k \geq 0$ by $\underline{0}_0 = \underline{0}$ and $\underline{0}_{k+1} = \underline{0} \multimap \underline{0}_k$.

Lemma 6.2 *$\underline{0}_k$ is definably isomorphic to $\underline{k}$.*

Proof: For each k, the arena denoted by $\underline{0}_k$ has a single initial Opponent move, which enables k distinct P-moves (in different $\sim$ equivalence classes). Thus it is isomorphic to the arena A_k. Moreover, the isomorphism is definable in SSCI as the terms $\mathtt{catch}_k : \underline{0}_k \multimap \underline{k}$ and $\lambda x.\lambda \vec{y}.\mathtt{case}\, \langle x, \langle y_0, \ldots, y_{k-1}\rangle : \underline{k} \multimap \underline{0}_k$. $\square$

Corollary 6.3 *Every functional type is definably isomorphic to a zero type.*

In the following we will assume that all arenas are the denotations of functional

types, and thus are well-opened and have the property that $m \sim m'$ implies $m = m'$. We now show that sufficient "observations" are definable to prove full abstraction.

Definition 6.4 *For any sequence s, let $|s|$ be the* multiset *of moves occurring in s. If $s \in L^{\natural}_{[\![T]\!]}$, we will say that a term $M_s : T$* tests for *s if $s \in [\![M_s]\!]$ and for all $t \in [\![M_s]\!]$, $|t| \subseteq |s|$.*

First, a key lemma (a form of "linear function extensionality" [1]). For any legal sequence s on $((A \multimap B) \multimap C) \multimap D$ there is a legal sequence $\phi(s)$ on $(B \multimap C) \multimap A \multimap D$ obtained by simply relabelling moves in A. Acting on strategies, this sends $[\![M : ((S \multimap T) \multimap U) \multimap V]\!]$ to $[\![\lambda x.\lambda y.M\, \lambda z.(x\,(z\,y))]\!]$. We shall now identify a set of legal sequences for which ϕ is an isomorphism.

Say that $m_1 m_2 s \in L^{\natural}_{A \multimap B}$ is *strict* if the (Player) move m_2 is from A, and *head-linear* if in addition there is no initial move from A in s.

Definition 6.5 *Given a strict, head-linear legal sequence $m_1 m_2 s \in L^{\natural}_{A \multimap (B \multimap C)}$, suppose $tb \sqsubseteq m_1 m_2 s$, where b is an initial move in B. The stack-view $\ulcorner \mathsf{open}(t) \urcorner$ contains the Player move m_2, and therefore some unique Opponent move justified by it. We will say that this is the indexing move for b.*

Lemma 6.6 *Any two occurrences of the initial move of B in $m_1 m_2 s$ have the same indexing move.*

Proof: Suppose b and b' are occurrences of the initial move of A in $m_1 m_2 s$, with indices c and c'. Then $b \sim b'$ and so by the non-interference condition $c \sim c'$ and so $c = c'$ as we have assumed that A is a function-type-object. $\square$

Thus if there is some initial move b of B in s with index m we may say that the index of B in s is m.

Lemma 6.7 *For any strict and head-linear legal sequence s on $(B \multimap C) \multimap A \multimap D$ in which the index of A is the initial move in B there is a (strict, head-linear) legal sequence $\phi^{-1}(s)$ on $((A \multimap B) \multimap C) \multimap D$ such that $\phi(\phi^{-1}(s)) = s$.*

Proof: We form $\phi^{-1}(s)$ by simply relabelling moves from A. $\square$

For any type $T_1 \multimap \ldots \multimap T_n \multimap \underline{0}$, and $i \leq n$, let $T^{[i]} = T_1 \multimap \ldots \multimap T_{i-1} \multimap T_{i+1} \multimap \ldots \multimap T_n \multimap \underline{0}$.

Proposition 6.8 *Let $S \multimap T$ be a zero type. For any even-length sequence $s \in L^{\natural}_{[\![T]\!]}$, there exists a term $M(s) : T$ which tests for s.*

Proof: We define $M(s)$ by induction on the length of s. For the base case, $s = \varepsilon$, let $M(s) = \Omega$. For the induction case, we first suppose $s = m_1 m_2 r$ is

strict and head-linear, and that $S = R_1 \multimap \ldots \multimap R_n \multimap \underline{0}$. Define $M(s)$ by induction on the arity of $S \multimap T$.

- If $T = \underline{0}$, then for each $i \leq n$, $r \upharpoonright [\![R_i]\!] = t_{i1}t_{i2}\ldots t_{ik_i}$, where each $t_{ij} \in L^{\natural}_{[\![R_i]\!]}$. For each $i \leq n$ we may define $x_i : \mathtt{var}[n+2] \vdash N_i : S_i =_{df} x_i := !x_i + 1; \mathtt{case}\,\langle !x_i, \langle \Omega, M(t_1), \ldots, M(t_n), \Omega\rangle\rangle$ for each i (by outer induction hypothesis), and thus:

$$M(s) =_{df} \lambda f.\mathtt{new}\,\lambda x_1 \ldots \mathtt{new}\,\lambda x_n.(f\,N_1 \ldots N_n)$$

  Then by definition of $[\![M(s)]\!]$, if $m_1m_2u \in [\![S \multimap T]\!]$ and $u \upharpoonright [\![R_i]\!] \sqsubseteq^E t_{i1}t_{i2}\ldots t_{ik_i}$ for each $i \leq n$ then $m_1m_2u \in [\![M(s)]\!]$, and so, in particular $s \in [\![M(s)]\!]$.

  Conversely, if $m_1m_2u \in [\![M(s)]\!]$ then $|u\upharpoonright[\![R_i]\!]| \subseteq |t_{i1}t_{i2}\ldots t_{ik_i}|$ for each i, and so $|m_1m_2u| \subseteq |s|$ and so $M(s)$ tests for s as required.
- For the induction case, suppose $T = U \multimap V$. If s contains no moves of U, then s relabels as a sequence s' on $S \multimap V$, and we may define $M(s) = \lambda x.\lambda y.M(s')\,x$. Otherwise, we assume that the index of U in s is the initial move in R_i. Then let $\widehat{s}$ be the (legal) sequence on $[\![(R_i \multimap S^{[i]}) \multimap T]\!]$ obtained by relabelling s.

  By Lemma 6.7 $\phi^{-1}(\widehat{s})$ is a legal sequence on $[\![((U \multimap R_i) \multimap S^{[i]}) \multimap V]\!]$, and by (inner) induction hypothesis, there is a term $M(\phi^{-1}(\widehat{s}))$ which tests for it. Hence we may define

$$M(s) = \lambda x.\lambda y.M(\phi^{-1}(\widehat{s})\,\lambda\vec{a}.(x\,(a_i\,y))a_1 \ldots a_{i-1}a_{i+1}\ldots a_n$$

Now suppose s is strict but not head-linear. Then $s = m_1m_2r_1m_2r_2$, where m_2 does not occur in r_1. We may define a head-linear legal sequence $m_1m_2r_1n$ in $[\![\underline{0} \multimap T]\!]$ by adding a single move n in $\underline{0}$. This is no longer than s, and so we may define a test $M(m_1m_2r_1n) : \underline{0} \multimap T$ for it. Since $m_1m_2r_2$ is legal and shorter than s, we may define a test $M(m_1m_2r_2)$ for it. Let

$$M(s) = \lambda\vec{x}.(\mathtt{label}\,\lambda k.(M(m_1m_2r_1n)\,k\vec{x})); (M(m_1m_2r_2)\,\vec{x})$$

$M(s)$ first tests for m_2r_1 and then jumps out using the `label` operation, and then tests for m_2r_2 — i.e. $M(s)$ tests for s as required.

Finally, suppose s is not strict. Then if $T = U_1 \multimap \ldots U_n \multimap \underline{0}$, suppose m_2 is the initial move in U_i. We obtain a strict sequence $\widehat{s} \in [\![U_i \multimap S \multimap T^{[i]}]\!]$ by relabelling, and thus a test for s — $\lambda y.\lambda\vec{x}.(M(\widehat{s})\,x_i)\,yx_1 \ldots x_{i_1}x_{i+1}\ldots x_n$. □

Theorem 6.9 *For any terms M, N, $[\![M]\!] \subseteq [\![N]\!]$ iff $M \lesssim N$.*

Proof: From left to right (soundness) this is a standard corollary of soundness and adequacy for the operational semantics. In proving the converse (completeness), we assume that M, N are closed terms of a zero-type, T. Suppose $[\![M]\!] \not\subseteq [\![N]\!]$ and let smn be a minimal length sequence in $[\![M]\!]^{\natural} - [\![N]\!]^{\natural}$. Let q be

the initial move in $M_{\underline{0}\multimap\underline{0}}$, and a its response. Then $qsmna$ is a legal sequence on $[\![T \multimap \underline{0} \multimap \underline{0}]\!]$. By Proposition 6.8, there exists $L : T \multimap \underline{0} \multimap \underline{0}$ such that $qsmna \in [\![L]\!]$ (hence $[\![L\,M]\!] \neq \bot$) and if $t \in [\![L]\!]$ then $|t| \subseteq |qsmna|$.

Suppose $[\![L\,N]\!] \neq \bot$. Then there exists $qta \in [\![L]\!]$ such that $t \in [\![N]\!]$. By minimality of smn, $s \in [\![N]\!]$, and hence by determinacy of $[\![L]\!]$ and $[\![N]\!]$, $qsm \sqsubseteq qt$, and so $|qsm| \subseteq |qta| \subseteq |qsmna|$ by definition of $[\![L]\!]$. But then $t = smn$, contradicting the assumption that $smn \notin [\![N]\!]$.

Hence $[\![\texttt{label}\,(L\,M)]\!] \neq \bot$ and $[\![\texttt{label}\,(L\,N)]\!] = \bot$, and so by soundness and adequacy $\texttt{label}\,(L\,M) \Downarrow$ and $\texttt{label}\,(L\,N) \not\Downarrow$ as required. □

By decidability of inclusion in the semantics (Proposition 5.15) we obtain decidability of observational approximation.

Theorem 6.10 *Observational equivalence and approximation are decidable in finitary observably sequential SCI.*

7 Conclusions and Further Directions

We may summarize our results via the following table:

	SCI	Obs. Sequential SCI	Erratic SCI
Equivalence	Decidable	Decidable	Undecidable
Approximation	Undecidable	Decidable	Undecidable

We have also shown that adding fixpoints (for closed terms of order three or greater) breaks termination in all of the above languages. More generally, we have extended "algorithmic programming language semantics" to an interference controlled setting, and lent weight to the thesis that observably sequential versions of programming languages are typically easier to model check. There are many possible variants and extensions of SCI, and thus many remaining open questions which could be investigated.

Contrary to a claim made in [13], it is not clear whether observational approximation is decidable in SCI without loops (i.e. without the command `while0`). The denotations of first-order terms of SCI without `while0` are finite sets, and therefore approximation for such terms is decidable at first-order. Moreover

we can reduce decidability of approximation at all higher types to the second-order case via (loop-free) definable monomorphisms. Although we can simulate any single tape FSA as a loop-free term of second-order type $((\underline{1}^n \multimap \underline{1}) \multimap \underline{1}$ this does not extend to the simulation of 2-tape FSA at second-order types as this would require sharing of variables. It may be possible, however, to represent second-order denotations as finite sets of single-tape automata.

Basic SCI can be extended with a notion of *passive* type, based on the principle that executing a term of passive type may not change the store, and therefore variables of passive type may be shared between procedures without causing interference through the store. As we observed in the introduction, we may (conservatively) embed purely functional languages such as PCF in these type systems by assigning purely passive types to their terms. Thus reasoning about them is at least as difficult as reasoning about PCF. The possibility exists of partial results, for example, by restricting the nesting of passive function types. McCusker and Wall have developed a fully abstract game semantics of SCIR (for which the basic fragment is similar to the games model described here), but there is no direct characterization of denotational equivalence in this (quotient) model.

Alternatively, it may be possible to construct relatively simple models of observably sequential SCI extended with passive types, modelled as in the effectively presentable game (or sequential algorithms semantics of SPCF) as non-repetitive sequences of moves.

We have considered the addition of two forms of side effect to SCI: non-determinism, and non-local control. Both of these demonstrate the sensitivity of observational equivalence to the presence of such effects. (We leave open the problem of decidability for SCI with control and non-determinism — if we introduce erratic choice into our games model of observably sequential SCI by dropping the determinacy condition for strategies, then full abstraction fails.) The most significant effects for which we have no results are concurrency and higher-order store. In the case of higher-order store, much work remains to be done to identify type systems for interference control as well as suitable semantics. In the case of concurrency, although we can represent parallel composition with no interference between threads in the relational model of SCI, once we allow one thread to be terminated by another, things become more complicated.

Other questions concern the expressive power of SCI and related languages: the correspondence between SCI terms and multitape finite state automata does not appear to fully capture this. Which functionals, for example, can be computed by SCI terms? Is there a characterisation of the programs which can be written in total SCI (i.e. SCI with primitive recursion rather than loops) and their equivalence? In the "observably sequential case" some approaches

are suggested by a companion paper [14] we have studied the expressiveness of a functional language with control (SPCF), subjected to the same affine typing discipline as SCI — in other words, observably sequential SCI without state. This is shown to be exactly as expressive as simply-typed SPCF itself — i.e. every term of SPCF is observationally equivalent (in SPCF) to one which is affinely typable. Since affinely typed SPCF is contained within observably sequential SCI, the latter can compute every observably sequential functional, and is arguably more expressive than SPCF.

Acknowledgements

I would like to acknowledge valuable discussions with Guy McCusker and Matthew Wall, and helpful comments from the anonymous referees.

References

[1] S. Abramsky. Axioms for full abstraction and full completeness. In *Essays in Honour of Robin Milner*. MIT Press, 1997.

[2] S. Abramsky. Beyond full abstraction: Model checking for algol-like languages, 2001. Lecture notes from the Marktoberdorf summer school.

[3] S. Abramsky and G. McCusker. Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions. In P.W. O'Hearn and R. Tennent, editors, *Algol-like languages*. Birkhauser, 1997.

[4] S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenburg and U. Berger, editors, *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School*. Springer–Verlag, 1998.

[5] S. Abramsky, K. Honda, G. McCusker. A fully abstract games semantics for general references. In *Proceedings of the 13th Annual Symposium on Logic In Computer Science, LICS '98*, 1998.

[6] R. Cartwright and M. Felleisen. Observable sequentiality and full abstraction. In *Proceedings of POPL '92*, 1992.

[7] D. Ghica and G. McCusker. The regular language semantics of second-order Idealised Algol. *Theoretical Computer Science*, 309:469 – 502, 2003.

[8] T. Harju and J. Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78:347–355, 1991.

[9] R. Harmer. *Games and Full Abstraction for Nondeterministic Languages*. PhD thesis, Imperial College London, 1999.

[10] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.

[11] E. Kinber. The inclusion problem for some classes of deterministic multitape automata. *Theoretical Computer Science*, 26:62–73, 1983.

[12] J. Laird. Full abstraction for functional languages with control. In *Proceedings of the Twelfth International Symposium on Logic In Computer Science, LICS '97*. IEEE Computer Society Press, 1997.

[13] J. Laird. Decidability in syntactic control of interference. In *Proceedings of ICALP '05*, number 3580 in LNCS, pages 904–916. Springer, 2005.

[14] J. Laird. On the expressiveness of affine programs with non-local control: The elimination of nesting in SPCF. In *Fundamenta Informaticae*, 77(4):511–531. IOS press, 2007.

[15] J. Laird. Game semantics and Linear CPS interpretation. *Theoretical Computer Science*, 333:199–224, 2005.

[16] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266(1 -2):341 –364, 2000.

[17] G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College London, 1996. Published by Cambridge University Press.

[18] G. McCusker. A fully abstract relational model of Syntactic Control of Interference. In *Proceedings of Computer Science Logic '02*, number 2471 in LNCS. Springer, 2002.

[19] G. McCusker. On the semantics of the bad-variable constructor in Algol-like languages. In *Proceedings of MFPS XIX*, ENTCS, 2003. To appear.

[20] G. McCusker and M. Wall. Categorical and game semantics for SCIR. In the proceedings of Games for Logics and Programming Languages, 2004.

[21] A. Murawski. On program equivalence in languages with ground-type references. In *Proceedings of LICS '03*. IEEE Press, 2003.

[22] A. Murawski and I. Walukiewicz. Third-order Idealized Algol with iteration is decidable. In *Proceedings of FoSSACS '05*, number 3411 in LNCS, pages 202–218. Springer, 2005.

[23] M. W. Minsky. Computation: Finite and Infinite Machines. Prentice-Hall, 1967.

[24] C.-H. L. Ong An approach to deciding observational equivalence of Algol-like languages. *Annals of Pure and Applied Logic*, 130:125–171, 2004.

[25] P. W. O'Hearn, A.J. Power, M. Takeyama and R.D. Tennent. Syntactic control of interference revisited. *Theoretical Computer Science*, 228(1-2):211–252, 1999.

[26] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.

[27] U. S. Reddy. Global state considered unnecesary: Object-based semantics for interference-free imperative programs. *Lisp and Symbolic Computation*, 9(1), 1996.

[28] J. Reynolds. Syntactic Control of Interference. In *Conf. Record 5^{th} ACM Symposium on Principles of Programming Languages*, pages 39–46, 1978.

[29] J. Reynolds. The essence of Algol. In *Algorithmic Languages*, pages 345–372. North Holland, 1981.