1995

Syntactic Control of Interference Revisited

Peter W. O'Hearn
Syracuse University

A. J. Power
University of Edinburgh

M. Takeyama
University of Edinburgh

R. D. Tennent
Queen's University

Syntactic Control of Interference Revisited

P. W. O'Hearn [1]

Syracuse University, Syracuse, New York, U.S.A. 13244

A. J. Power [2] and M. Takeyama [3]

University of Edinburgh, Edinburgh, Scotland EH9 3JZ

R. D. Tennent [4]

Queen's University, Kingston, Canada K7L 3N6

Dedicated to John C. Reynolds, in honor of his 60th birthday

Abstract

In "Syntactic Control of Interference" (POPL, 1978), J. C. Reynolds proposes three design principles intended to constrain the scope of imperative state effects in Algol-like languages. The resulting linguistic framework seems to be a very satisfactory way of combining functional and imperative concepts, having the desirable attributes of *both* purely functional languages (such as PCF) *and* simple imperative languages (such as the language of **while** programs).

However, Reynolds points out that the "obvious" syntax for interference control has the unfortunate property that β-reductions do not always preserve typings. Reynolds has subsequently presented a solution to this problem (ICALP, 1989), but it is fairly complicated and requires intersection types in the type system. Here, we present a much simpler solution which does not require intersection types.

We first describe a new type system inspired in part by linear logic and verify that reductions preserve typings. We then define a class of "bireflective" models, which provide a categorical analysis of structure underlying the new typing rules; a companion paper "Bireflectivity," in this volume, exposes wider ramifications of this structure. Finally, we describe a concrete model for an illustrative programming language based on the new type system; this improves on earlier such efforts in that states are not assumed to be structured using locations.

Contents

1 Introduction

It has long been known that a variety of anomalies can arise when a programming language combines assignment with a sufficiently powerful procedure mechanism.
J. C. Reynolds (1978)

In an imperative programming language, a term C is said to *interfere* with a term E if executing (or, as appropriate, assigning to or calling) C can affect the outcome of E. For example, command $x := a$ interferes with expression $x + 1$, but not *vice versa*.

In purely functional languages, there is *no* interference between terms, and it is usually taken for granted [4,17,16] that this explains why reasoning

[1] Current address: Department of Computer Science, Queen Mary and Westfield College, London, England E1 4NS. This author was supported by NSF grant CCR-92110829.
[2] This author gratefully acknowledges the support of ESPRIT Basic Research Action 6453: Types for proofs and programs.
[3] Current address: Department of Computing Science, Chalmers University of Technology, Eklandagatan 86, 412 96 Gteborg, Sweden. This author was supported by an Edinburgh University Postgraduate Studentship.
[4] This author was supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

about purely functional programs is relatively straightforward. However, for "simple" imperative languages without full procedures, Hoare's logic [13] (and total-correctness variants of it) are quite satisfactory. This suggests that it is simplistic to attribute the serious difficulties that arise in reasoning about programs in conventional procedural languages to the presence of interference.

J. C. Reynolds [37,38,40] has provided a more refined analysis. He argues that conventional procedural languages are problematical primarily because they permit *covert* interference, that is to say, interference that is not syntactically obvious. For example, if identifiers x and y are *aliases* (denote the same storage variable), then $y := a$ interferes with $x + 1$, and this is problematic because the interference is not obvious from inspecting these phrases. In general, alias detection in a conventional higher-order procedural language requires complex interprocedural data-flow analysis of an entire program.

Similarly, if a procedure accesses a non-local variable and the value of that variable can be changed between calls of the procedure, then identical calls of the procedure may have different effects. Covert interference via non-local variables can also result in subtle bugs in the use of procedural parameters. For example, suppose $\mathit{Traverse}(p)$ applies procedural parameter p to every node of a data structure and *Remove* has the effect of deleting the node to which it is applied; then a call such as $\mathit{Traverse}(\mathit{Remove})$ will often fail to have the effect the programmer intends because removing a node can interfere with a traversal.

The problem of covert interference also affects language designers. For example, programmers expect that, immediately after assigning a value to a variable, the variable has the value just assigned; but this "obvious" property fails for so-called "bad" variables, such as the subscripted variable $A(A(i))$ whose sub-expression $A(i)$ is interfered with by the array variable A when $A(i) = i$. A language designer might want to forbid bad variables syntactically, but covert interference makes this very difficult; for example, $A(j)$ is a bad variable if j is an alias for $A(i)$. Similar difficulties arise for a language designer trying to provide a "block expression" (a command within an expression) without allowing side effects to non-local variables, trying to provide secure features for unions of types, or trying to allow concurrent composition of non-interfering commands.

The difficulties created by covert interference are especially evident if one considers reasoning principles. For example, in "specification logic," a Hoare-like logic for full Algol-like languages [38,40], the axiom for assignments is

$$\mathbf{gv}(V) \ \&\ V \# P \ \Rightarrow\ \{P(E)\}\, V := E\, \{P(V)\}$$

The consequent is essentially the familiar axiom from [13], but assumption $V \# P$ asserts that assignments to variable V do not covertly interfere with the pre and post-conditions, and assumption $\mathbf{gv}(V)$ asserts that V is a "good" variable. Similarly, the "Constancy" axiom in specification logic differs from the corresponding axiom in Hoare's logic in that a simple syntactic side condition must be replaced by a non-interference assumption. Finally, because of possible covert interference, procedure specifications must be more com-

plex: explicit assumptions about what procedures do *not* do are required (*cf.* the "frame problem" in artificial intelligence [6]) in order to discharge non-interference assumptions in the context of procedure calls. All of these complexities are clearly evident in the examples in [38].

For these reasons, many language designers have argued that programming languages should be designed so that it is *easy* for programmers and compilers to verify that program phrases do not interfere; some early examples are [5,14,45]. In [37], three general design principles intended to facilitate verification of non-interference are proposed.

(i) There should be no "anonymous" channels of interference; then the problem of verifying that C doesn't interfere with E reduces to showing that no free identifier of C interferes with any free identifier of E.

(ii) Distinct identifiers should not interfere; then if two sets of identifiers are disjoint, they are guaranteed not to interfere.

(iii) Some types of phrases, such as side effect-free expressions, are "passive" (do not interfere with anything), and so the disjointness requirement can be relaxed to allow sharing of identifiers used only passively.

In summary, to verify in this setting that C does not interfere with E, it is sufficient to ensure that no actively occurring free identifier of C is also free in E.

But of course the programming language must be designed so that there are no anonymous channels of interference and, in every context, distinct identifiers do not interfere. The first requirement is straightforward, but to achieve the second, it is proposed that the following basic constraint be imposed on procedure calls $P(A)$: the procedure part P and the argument part A should be mutually non-interfering (and similarly for defined language constructs, such as local definitions, that have *implicit* procedure calls). Note the elegant circularity of the approach: the syntactic restriction ensures that distinct identifiers do not interfere, and this property makes it feasible to implement the restriction using the syntactic criterion described in the preceding paragraph.

The syntax of an Algol-like programming language designed according to these principles is described in [37]. This design is extremely successful in most respects, combining the desirable attributes of *both* purely functional languages (such as PCF) *and* simple imperative languages (such as the language of **while** programs); however, a problem in the treatment of passivity is noted. In the approach used to incorporate the third principle (allowing sharing of passive identifiers), the syntax is such that the *subject-reduction* property fails; i.e., reductions may fail to preserve typing. Reynolds subsequently presented a solution to this problem in [41], but it is fairly complicated and requires intersection types [7] in the type system. We feel that the methods of interference control should be applicable relatively independently of the specifics of intersection types (which of course have substantial other merits).

In this work, we present a very simple and intuitive alternative solution to the problem of passive uses. Our solution does not require intersection types, allowing interference control to be investigated without unnecessary syntactic

or semantic complexity. Also, it would be conceivable to apply these methods in contexts, such as ML-like or Haskell-like languages, where the addition of intersection types would be far from trivial.

The type system presented here was actually worked out by the first author in 1991, but lay dormant for a number of years because it contained features for which no satisfactory semantic explanation was known. More precisely, at that time it would have been straightforward to formulate a type soundness theorem, based on an operational semantics, or a simple denotational model (with an adequacy theorem) that correctly predicted behaviour of complete programs. The perceived difficulty, however, was not whether *some* model existed, but rather that the typing rules for passivity exhibited intricate interactions, which, in the absence of a semantic analysis deeper than that provided by adequacy or type soundness, appeared discomfortingly *ad hoc*. In particular, the type system hinges on a treatment of "passively occuring" identifiers; i.e., identifiers, possibly of active type, that, in some contexts, are only *used* passively. This treatment is subtle, but crucial for treating types that *combine* passivity and activity, such as types for storage variables or products of passive and active types.

So, a central role is played in this paper by a semantic analysis of passivity, couched in terms of a new categorical concept of *bireflectivity*. The bireflective semantics exposes structural properties underlying our type-theoretic treatment, where the typing rules for passivity correspond to certain adjunctions. This provides a measure of relief for our previous fears of the potential *ad hoc* nature of the typing rules; further support is provided by a companion paper *Bireflectivity*, in this volume, which introduces bireflective subcategories and studies their mathematical properties.

To ground this analysis we describe a concrete model in which a subcategory of passive objects is built using semantic entities that, in a precise sense, can read from, but not write to, the computer store. The model improves on earlier efforts [42,25] in that states are not assumed to be structured using "locations." As a result, we obtain a much cleaner model in which the "disjointness" of identifiers is clearly visible. Distinct identifiers get associated with separate state-sets, and the sharing of passively-used state is explained through semantic "contraction" mappings.

We are grateful to Uday Reddy for numerous discussions that influenced the content and presentation of this paper. In fact, the revival of the type system came about originally as a result of his model of passivity in [35], follow-up correspondence in which he pointed out that our rules of Passification and Activation corresponded to a monad structure in his model, and his challenge to look for similar structure in Tennent's model of specification logic [43]. This challenge led to the identification of bireflective category structure which, it finally turns out, is subtly different from the structure in Reddy's model (see Section 3.4). A crucial step forward in this development was the utilization of Day's tensor product construction, the relevance of which was suggested by Andy Pitts.

A preliminary version of this paper appeared in [26].

2 Syntax

2.1 Passive Uses

The treatment of passivity in [37] is based on designating certain phrase types (such as "state reader" expression types) as being *passive*, and then, for any phrase R, determining

> the set of identifiers which have at least one free occurrence in R which is outside of any subphrase of passive type.

These are considered to be the *actively occurring* free identifiers of R. Unfortunately, this definition, being context-independent, cannot take account of the fact that, when R itself occurs within a passive phrase, *none* of its free identifiers can be used actively. This means that the syntactic constraints on procedure calls are unnecessarily restrictive, which results in anomalies when types combine passive and non-passive capabilities.

For example, a storage variable is used *passively* when it is read from, as on the right-hand side of an assignment statement, and *actively* when it is assigned to. Suppose that identifiers x and w are of type $\mathbf{var}[\tau]$ (i.e., they are τ-valued variable identifiers, with τ a data type such as **int** or **bool**), and consider the following command:

$$\Big(\lambda z{:}\,\tau.\ x := (\lambda y{:}\,\tau.\, w)z\Big)\,(w) \tag{1}$$

where typings of the form $\iota{:}\,\tau$ indicate that ι is a τ-valued *expression* identifier. Although w occurs in both the procedure and argument parts of the outer call, the phrase is *legal* because both occurrences are in expressions and hence regarded as passive. However, the command β-reduces to

$$x := (\lambda y{:}\,\tau.\, w)w \tag{2}$$

in which the right-hand side is *illegal*, according to Reynolds's treatment, because variable identifier w is deemed to occur actively in the procedure (which has type $\tau \to \mathbf{var}[\tau]$), and also occurs in the argument. But the procedure call is actually an *expression*, and so there cannot be any interference via w; indeed, the assignment β-reduces to the legal $x := w$.

It can be argued that the anomaly in this example could be avoided if dereferencing coercions were explicit; however, more complex examples, as in [37], show that the problem is a fundamental one. (An example of this kind from *loc. cit.* will be discussed in Section 2.4). The problem arises essentially because the context-independent notion of active occurrence cannot be sensitive to situations in which the context ensures passive use of potentially non-passive entities. To avoid the anomalies, it is necessary to consider when identifiers occur actively in *instances* of phrases, taking context into account.

2.2 The SCIR Type System

The phrase types are built from certain primitive types $\langle\text{prim}\rangle$ as follows:

$$\theta ::= \langle\text{prim}\rangle \mid \theta \otimes \theta' \mid \theta \times \theta' \mid \theta' \to \theta \mid \theta' \to_P \theta.$$

A subset $\langle \text{prim}_p \rangle$ of the primitive types is singled out as passive, and this generates the passive types as follows:

$$\phi ::= \langle \text{prim}_p \rangle \mid \phi \otimes \phi' \mid \phi \times \phi' \mid \theta \to \phi \mid \theta' \to_P \theta.$$

There are two products: $\theta \times \theta'$, for which the components can interfere, and $\theta \otimes \theta'$, for which the components must be non-interfering. There are also two exponentials: $\theta' \to \theta$, which is the type of ordinary procedures (which cannot interfere with, or be interfered with by, their arguments), and $\theta' \to_P \theta$, which is the type of passive procedures. A passive procedure does not assign to any global variables (though a *call* of a passive procedure may be active, if the argument of the call is).

We propose a syntax based on typing judgements $\Pi \mid \Gamma \vdash P{:}\theta$ in which the usual typing context on the left of the turnstile is partitioned into a "passive" zone Π and an "active" zone Γ. No identifier can be in both the passive and the active zones. Intuitively, if an identifier is in the passive zone, it can only be *used* passively, even if the type of the identifier is non-passive. The typing rules will be arranged so that when a phrase under a type assignment is placed in a context, that context must prevent identifiers in the passive zone from being used actively.

This use of zones is reminiscent of Girard's LU [12], with the passive/active distinction here being similar to the classical/linear distinction there; however, the *permeability* rules, that govern movement across the zone separator $\mid$, do not appear in LU nor, as far as we are aware, in other previous systems. These rules are the most distinctive aspect of the treatment of passivity here. See Section 2.6 for further discussion.

The rules concerning identifiers and contexts are in Table 1. Identifiers are initially introduced in the active zone, but may change zones with the help of the permeability rules of Passification[5] and Activation. Movement to the passive zone is accomplished using Passification, when the phrase on the right-hand side of the turnstile is of passive type. This is the only way that an identifier can move to the passive zone. On the other hand, a passive identifier can always be activated using the Activation rule. Notice that θ is unrestricted in the Passification rule, and that the change-of-zone is not accompanied by a change-of-type for the assumption; this is a key difference from the otherwise similar use of zones in LU.

Weakening and Exchange can be used in either zone. When type assignments are concatenated, as in the Weakening rule, we implicitly assume that the domains are disjoint. $\widetilde{\Pi}$ and $\widetilde{\Gamma}$ are permutations of Π and Γ, respectively.

Contraction can only be used in the passive zone. This is the essential restriction that implements the requirement that distinct identifiers do not interfere. We are using the notation $[P](\iota' \mapsto Q)$ to denote the result of substituting Q for free occurrences of ι' in P.

[5] This fabricated word seems more attractive as a name for this rule than alternatives such as Passivation or Deactivation.

Table 1 Identity and Structural Rules

IDENTITY

$$\frac{}{\mid \iota\colon\theta \vdash \iota\colon\theta}\ \text{Axiom}$$

STRUCTURE

$$\frac{\Pi \mid \iota\colon\theta, \Gamma \vdash P\colon\phi}{\Pi, \iota\colon\theta \mid \Gamma \vdash P\colon\phi}\ \text{Passification} \qquad \frac{\Pi, \iota\colon\theta \mid \Gamma \vdash P\colon\theta'}{\Pi \mid \iota\colon\theta, \Gamma \vdash P\colon\theta'}\ \text{Activation}$$

$$\frac{\Pi \mid \Gamma \vdash P\colon\theta}{\Pi, \Pi' \mid \Gamma, \Gamma' \vdash P\colon\theta}\ \text{Weakening} \qquad \frac{\Pi \mid \Gamma \vdash P\colon\theta}{\widetilde{\Pi} \mid \widetilde{\Gamma} \vdash P\colon\theta}\ \text{Exchange}$$

$$\frac{\Pi, \iota\colon\theta, \iota'\colon\theta \mid \Gamma \vdash P\colon\theta'}{\Pi, \iota\colon\theta \mid \Gamma \vdash [P](\iota' \mapsto \iota)\colon\theta'}\ \text{Contraction}$$

Rules for the type constructors are given in Table 2. Note that the active zone in rule $\to_P I$ is empty. Also, note that the type assignments for the procedure and the argument parts of procedure calls (rule $\to E$) must be disjoint; however, Contraction allows sharing of identifiers from the passive zone. Similar remarks apply to the introduction rule for $\otimes$.

In the preliminary version of this paper we used a rule for $\otimes$-elimination based on projections:

$$\frac{\Pi \mid \Gamma \vdash P\colon\theta_0 \otimes \theta_1}{\Pi \mid \Gamma \vdash \pi_i^{\otimes}\ P\colon\theta_i}\ \otimes E_i\ (i = 0, 1)$$

This rule was used on the grounds that projections are definable in the presence of Weakening, and the erroneous remark was made that the two forms of elimination would thus be equivalent.

The formulation with projections has two problems. First, it is not possible in general to unpack a term of type $\theta_0 \otimes \theta_1$ into non-interfering components. Second, it is not possible to mimic the isomorphism taking $f\colon\theta_0 \to \theta_1 \to \theta_2$ to

$$\lambda x\colon\theta_0 \otimes \theta_1\,.\,\mathbf{let}\ x \otimes y\ \mathbf{be}\ z\ \mathbf{in}\ fxy\colon\theta_0 \otimes \theta_1 \to \theta_2$$

These remarks do not invalidate any of the technical results in [26]; however, we now regard the formulation using projections as a language-design mistake.

Table 2 Rules for Type Constructors

$$\frac{\Pi \mid \Gamma \vdash P\colon \theta_0 \quad \Pi \mid \Gamma \vdash Q\colon \theta_1}{\Pi \mid \Gamma \vdash \langle P, Q\rangle\colon \theta_0 \times \theta_1}\ \times I \qquad \frac{\Pi \mid \Gamma \vdash P\colon \theta_0 \times \theta_1}{\Pi \mid \Gamma \vdash \pi_i\, P\colon \theta_i}\ \times E_i\ (i = 0, 1)$$

$$\frac{\Pi_0 \mid \Gamma_0 \vdash P\colon \theta_0 \quad \Pi_1 \mid \Gamma_1 \vdash Q\colon \theta_1}{\Pi_0, \Pi_1 \mid \Gamma_0, \Gamma_1 \vdash P \otimes Q\colon \theta_0 \otimes \theta_1}\ \otimes I \qquad \frac{\Pi \mid \Gamma \vdash P\colon \theta_0 \otimes \theta_1 \quad \Pi' \mid \Gamma', \iota_0\colon \theta_0, \iota_1\colon \theta_1 \vdash Q\colon \theta}{\Pi, \Pi' \mid \Gamma, \Gamma' \vdash \mathbf{let}\ \iota_0 \otimes \iota_1\ \mathbf{be}\ P\ \mathbf{in}\ Q\colon \theta}\ \otimes E$$

$$\frac{\Pi \mid \Gamma, \iota\colon \theta' \vdash P\colon \theta}{\Pi \mid \Gamma \vdash \lambda\iota\colon \theta'.\ P\colon \theta' \to \theta}\ \to I \qquad \frac{\Pi_0 \mid \Gamma_0 \vdash P\colon \theta' \to \theta \quad \Pi_1 \mid \Gamma_1 \vdash Q\colon \theta'}{\Pi_0, \Pi_1 \mid \Gamma_0, \Gamma_1 \vdash P(Q)\colon \theta}\ \to E$$

$$\frac{\Pi \mid \vdash Q\colon \theta' \to \theta}{\Pi \mid \vdash \mathbf{promote}\, Q\colon \theta' \to_P \theta}\ \to_P I \qquad \frac{\Pi \mid \Gamma \vdash Q\colon \theta' \to_P \theta}{\Pi \mid \Gamma \vdash \mathbf{derelict}\, Q\colon \theta' \to \theta}\ \to_P E$$

2.3 An Illustrative Programming Language

An illustrative Algol-like programming language is obtained by choosing appropriate primitive types and constants. We use a type **comm** of commands and types τ for τ-valued expressions:

$$\langle \text{prim} \rangle ::= \tau \mid \mathbf{comm}$$

where τ ranges over, say, **int** and **bool**. The only *passive* primitive types are the expression types τ.

The type $\mathbf{var}[\tau]$ of τ-valued variables abbreviates $(\tau \to \mathbf{comm}) \times \tau$. Dereferencing is implemented by the second projection; in examples, we will suppress explicit mention of this projection and assume a rule

$$\frac{\Pi \mid \Gamma \vdash V\colon \mathbf{var}[\tau]}{\Pi \mid \Gamma \vdash V\colon \tau}\ \text{Dereferencing}$$

We can consider constants representing various imperative constructs, such as

$:=_\tau\colon \mathbf{var}[\tau] \times \tau \to \mathbf{comm}$	assignment
$;\colon \mathbf{comm} \times \mathbf{comm} \to \mathbf{comm}$	sequential composition
$\|\|\colon \mathbf{comm} \otimes \mathbf{comm} \to \mathbf{comm}$	parallel composition
$\mathbf{if}_\theta\colon \mathbf{bool} \times \theta \times \theta \to \theta$	conditional
$\mathbf{Y}_\theta\colon (\theta \to_P \theta) \to \theta$	recursion
$\mathbf{new}_\tau\colon (\mathbf{var}[\tau] \to \mathbf{comm}) \to \mathbf{comm}$	local allocation
$\mathbf{do}_\tau\colon (\mathbf{var}[\tau] \to_P \mathbf{comm}) \to \tau$	block expression

The block-expression form requires some explanation; the call $\mathbf{do}_\tau(p)$ is evaluated by allocating a new local variable and applying p to it, as with the ordinary command block $\mathbf{new}_\tau(p)$, but then returning the final value of the local variable as the value of the expression. The passivity of $p\colon \mathbf{var}[\tau] \to_P \mathbf{comm}$ ensures that the block expression does not interfere with non-local variables, and so no "snap-back" effect is needed to restore their original values.

2.4 Examples

We illustrate the operation of the rules by presenting derivations of some typing judgements.

Consider first the (unreduced) example (1) discussed in Section 2.1. The assignment can be typed as follows:

$$\dfrac{\dfrac{\dfrac{}{\mid x\colon \mathbf{var}[\tau] \vdash x\colon \mathbf{var}[\tau]} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mid w\colon \mathbf{var}[\tau] \vdash w\colon \mathbf{var}[\tau]}}{\mid w\colon \mathbf{var}[\tau], y\colon \tau \vdash w\colon \mathbf{var}[\tau]}\,\text{Weakening}}{\mid w\colon \mathbf{var}[\tau] \vdash \lambda y\colon \tau.\, w\colon \tau \to \mathbf{var}[\tau]}\,{\to}I \quad \mid z\colon \tau \vdash z\colon \tau}{\mid w\colon \mathbf{var}[\tau], z\colon \tau \vdash (\lambda y\colon \tau.\, w)z\colon \mathbf{var}[\tau]}\,{\to}E}{\mid w\colon \mathbf{var}[\tau], z\colon \tau \vdash (\lambda y\colon \tau.\, w)z\colon \tau}\,\text{Dereferencing}}{w\colon \mathbf{var}[\tau] \mid z\colon \tau \vdash (\lambda y\colon \tau.\, w)z\colon \tau}\,\text{Passification}}{w\colon \mathbf{var}[\tau] \mid x\colon \mathbf{var}[\tau], z\colon \tau \vdash x := (\lambda y\colon \tau.\, w)z\colon \mathbf{comm}}\,:=$$

where the last step abbreviates use of the $:=$ constant, $\times I$, $\to E$ and Weakening. Note that after Dereferencing of the right-hand side, w can be moved to the passive zone. The typing is then completed as follows, using a Contraction:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{w\colon \mathbf{var}[\tau] \mid x\colon \mathbf{var}[\tau], z\colon \tau \vdash x := (\lambda y\colon \tau.\, w)z\colon \mathbf{comm}}}{w\colon \mathbf{var}[\tau] \mid x\colon \mathbf{var}[\tau] \vdash \lambda z\colon \tau.\, x := (\lambda y\colon \tau.\, w)z\colon \tau \to \mathbf{comm}}\,{\to}I \quad \dfrac{\dfrac{\dfrac{}{\mid w'\colon \mathbf{var}[\tau] \vdash w'\colon \mathbf{var}[\tau]}}{\mid w'\colon \mathbf{var}[\tau] \vdash w'\colon \tau}}{w'\colon \mathbf{var}[\tau] \mid\vdash w'\colon \tau}\,\text{Pass.}}{w, w'\colon \mathbf{var}[\tau] \mid x\colon \mathbf{var}[\tau] \vdash (\lambda z\colon \tau.\, x := (\lambda y\colon \tau.\, w)z)\,(w')\colon \mathbf{comm}}\,{\to}E}{w\colon \mathbf{var}[\tau] \mid x\colon \mathbf{var}[\tau] \vdash (\lambda z\colon \tau.\, x := (\lambda y\colon \tau.\, w)z)\,(w)\colon \mathbf{comm}}\,\text{Contraction}$$

The following shows how to derive a typing for the right-hand side of the "illegal" assignment (2) in Section 2.1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mid w\colon \mathbf{var}[\tau] \vdash w\colon \mathbf{var}[\tau]}}{\mid w\colon \mathbf{var}[\tau], y\colon \tau \vdash w\colon \mathbf{var}[\tau]}\,\text{Weak.}}{\mid w\colon \mathbf{var}[\tau] \vdash \lambda y\colon \tau.\, w\colon \tau \to \mathbf{var}[\tau]}\,{\to}I \quad \dfrac{\dfrac{}{\mid w'\colon \mathbf{var}[\tau] \vdash w'\colon \mathbf{var}[\tau]}}{\mid w'\colon \mathbf{var}[\tau] \vdash w'\colon \tau}\,\text{Dereferencing}}{\mid w, w'\colon \mathbf{var}[\tau] \vdash (\lambda y\colon \tau.\, w)w'\colon \mathbf{var}[\tau]}\,{\to}E}{\mid w, w'\colon \mathbf{var}[\tau] \vdash (\lambda y\colon \tau.\, w)w'\colon \tau}\,\text{Dereferencing}}{w, w'\colon \mathbf{var}[\tau] \mid\vdash (\lambda y\colon \tau.\, w)w'\colon \tau}\,\text{Passification}}{w\colon \mathbf{var}[\tau] \mid\vdash (\lambda y\colon \tau.\, w)w\colon \tau}\,\text{Contraction}}{\mid w\colon \mathbf{var}[\tau] \vdash (\lambda y\colon \tau.\, w)w\colon \tau}\,\text{Activation}$$

Even though the types of w and w' are active, Contraction can be applied when they are in the passive zone; but Dereferencing must be used before these identifiers can be passified. The assignment can then be typed as usual:

$$\dfrac{\mid x\colon \mathbf{var}[\tau] \vdash x\colon \mathbf{var}[\tau] \quad \dfrac{\vdots}{\mid w\colon \mathbf{var}[\tau] \vdash (\lambda y\colon \mathbf{var}[\tau].\, w)w\colon \tau}}{\mid x, w\colon \mathbf{var}[\tau] \vdash x := (\lambda y\colon \mathbf{var}[\tau].\, w)w\colon \mathbf{comm}}\,:=$$

The next example demonstrates that an identifier can be used both actively and passively. The following derivation involves shared passive use of a variable identifier x:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mid x\colon \mathbf{var}[\tau] \vdash x\colon \mathbf{var}[\tau]}}{\mid x\colon \mathbf{var}[\tau] \vdash x\colon \tau}\ \dfrac{}{x\colon \mathbf{var}[\tau] \mid\vdash x\colon \tau}\ \text{Pass.}}{\mid y\colon \mathbf{var}[\tau] \vdash y\colon \mathbf{var}[\tau]}\quad \dfrac{}{x\colon \mathbf{var}[\tau] \mid y\colon \mathbf{var}[\tau] \vdash y := x\colon \mathbf{comm}}\ :=\qquad \begin{array}{c}\vdots\\ x'\colon \mathbf{var}[\tau] \mid z\colon \mathbf{var}[\tau] \vdash z := x'\colon \mathbf{comm}\end{array}}{x, x'\colon \mathbf{var}[\tau] \mid y, z\colon \mathbf{var}[\tau] \vdash y := x \parallel z := x'\colon \mathbf{comm}}\ \parallel}{x\colon \mathbf{var}[\tau] \mid y, z\colon \mathbf{var}[\tau] \vdash y := x \parallel z := x\colon \mathbf{comm}}\ \text{Contraction}}{\mid x, y, z\colon \mathbf{var}[\tau] \vdash y := x \parallel z := x\colon \mathbf{comm}}\ \text{Activation}$$

where the derivation for $z := x'$ is similar to that for $y := x$, and the step for $\parallel$ uses the introduction rule for $\otimes$ followed by the elimination rule for $\rightarrow$ with the constant $\parallel$. This can then be combined with non-passive use of x, as in the following derivation:

$$\dfrac{\begin{array}{c}\vdots\\ \mid w\colon \tau, x\colon \mathbf{var}[\tau] \vdash x := w\colon \mathbf{comm}\end{array}\quad \begin{array}{c}\vdots\\ \mid x, y, z\colon \mathbf{var}[\tau] \vdash y := x \parallel z := x\colon \mathbf{comm}\end{array}}{\mid w\colon \tau, x, y, z\colon \mathbf{var}[\tau] \vdash x := w\,;\,(y := x \parallel z := x)\colon \mathbf{comm}}\ ;$$

We now consider the problematic example from [37, p. 44]. Suppose $n, y\colon \mathbf{var}[\mathbf{int}]$; then, the parallel command in

$$\pi_0\Big\langle n+1\,,\,(n := 0 \parallel y := \pi_0\langle n\,,\,n := 0\rangle)\Big\rangle$$

is illegal in the treatment of [37] because n is used on both sides of $\parallel$. However, the entire term is of type $\mathbf{int}$, and so these uses should be regarded as passive. To type this in our system, we can proceed as follows:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \mid n, n'\colon \mathbf{var}[\mathbf{int}] \vdash \pi_0\langle n+1\,,\,(n := 0 \parallel y := \pi_0\langle n'\,,\,n' := 0\rangle)\rangle\colon \mathbf{int}\end{array}}{n, n'\colon \mathbf{var}[\mathbf{int}] \mid y\colon \mathbf{var}[\mathbf{int}] \vdash \pi_0\langle n+1\,,\,(n := 0 \parallel y := \pi_0\langle n'\,,\,n' := 0\rangle)\rangle\colon \mathbf{int}}\ \text{Passification}}{n\colon \mathbf{var}[\mathbf{int}] \mid y\colon \mathbf{var}[\mathbf{int}] \vdash \pi_0\langle n+1\,,\,(n := 0 \parallel y := \pi_0\langle n\,,\,n := 0\rangle)\rangle\colon \mathbf{int}}\ \text{Contraction}}{\mid n, y\colon \mathbf{var}[\mathbf{int}] \vdash \pi_0\langle n+1\,,\,(n := 0 \parallel y := \pi_0\langle n\,,\,n := 0\rangle)\rangle\colon \mathbf{int}}\ \text{Activation}$$

The first line can be typed straightforwardly because the identifiers on either side of $\parallel$ are distinct.

Notice that the subterm $(n := 0 \parallel y := \pi_0\langle n\,,\,n := 0\rangle)$ does not itself have any typing in the SCIR type system. But it can nevertheless appear in a larger term because Contraction can be applied when a subterm with occurrences of n remaned apart appears within a passive phrase. This subtle interaction of Contraction and Passification is what allows the subject reduction problems from [37] to be solved. An equivalent type system that does not use Contraction explicitly can be formulated, but replaces this subtle interaction by explicitly accounting for the "semi-well-typed" status of phrases such as $(n := 0 \parallel y := \pi_0\langle n\,,\,n := 0\rangle)$, or more simply $(\lambda x.\, x; y)y$.

Finally, it is natural to ask about the relationship between the SCIR treatment of passivity in the SCI2 treatment in [37].

We have argued that a merit of our approach is that it shows that subtypes are not *necessary* for the treatment of passivity. But a compensating merit of

Table 3 β-reductions

$$\pi_0\langle P, Q\rangle \to_\beta P \qquad \pi_1\langle P, Q\rangle \to_\beta Q$$

$$\textbf{let } \iota_0 \otimes \iota_1 \textbf{ be } P \otimes Q \textbf{ in } M \to_\beta [M](\iota_0 \mapsto P, \iota_1 \mapsto Q)$$

$$(\lambda\iota{:}\theta.P)Q \to_\beta [P](\iota \mapsto Q) \qquad \textbf{derelict}(\textbf{promote}\, Q) \to_\beta Q$$

$$\frac{P \to_\beta Q}{C[P] \to_\beta C[Q]} \quad \text{for term-with-hole } C[\cdot]$$

SCI2 is that it can typecheck programs that SCIR cannot. One example is

$$\lambda c_1{:}\,\textbf{comm}\,.\,\lambda c_2{:}\,\textbf{comm}\,.\,\lambda c_3{:}\,\textbf{comm}\,.\,\pi_0\langle c_0, c_3\rangle \parallel \pi_0\langle c_2, c_3\rangle$$

$$:\textbf{comm} \to \textbf{comm} \to \textbf{comm} \to \textbf{comm}$$

In SCIR this program is not typable because the active identifier c_3 appears in both arms of the parallel composition, and because there are no passive phrases to allow use of the rule of Passification. But in SCI2 products are represented as records with named alternatives, and a forgetting-fields conversion can be applied which, in effect, assigns a (passive) unit type to c_3; this is reasonable, as c_3 is never used.

It can be argued that this points to an incompleteness in the SCIR type system, because c_3 is used passively in the example. A counter-argument is that the example has not so much to do with passive use but with the ability of subtyping to account for some circumstances when parts of a record are not used at all. We wonder if there is a precise relationship between SCI2 and a version (as yet unformulated) of SCIR with conjunctive types.

These details aside, we would like to emphasize that the central aspects of syntactic control of interference, including passivity, were already identified in [37], and we regard the type theoretic solution presented in this paper as a further development and analysis of ideas present there.

2.5 Typing and Reduction

The principal reductions for the SCIR type system are in Table 3. (A comprehensive treatment would require commuting conversions for $\otimes$ [1], which are omitted here.)

Theorem 2.1 (Subject Reduction) *If* $\Pi \mid \Gamma \vdash P{:}\theta$ *and* $P \to_\beta Q$ *then* $\Pi \mid \Gamma \vdash Q{:}\theta$. □

Typing is also preserved by various η laws.

To prove this result we will concentrate on the reduction from $(\lambda x.P)Q$ to $[P](\iota \mapsto Q)$. The proofs for **let** is similar, projections and Promotion/Dereliction elimination are easier, and the extension to subterms via the rule for $C[\cdot]$ is not difficult. We need two lemmas.

Lemma 2.2 *If* $\Pi \mid \Gamma \vdash \lambda\iota{:}\theta'.\, P{:}\theta' \to \theta$ *then* $\Pi \mid \Gamma, \iota{:}\theta' \vdash P{:}\theta$.

Proof. We have assumed (without loss of generality) that ι is not in Π or Γ. The result clearly holds if the last step of the derivation for $\lambda\iota{:}\theta'.P$ is an instance of $\to I$, and is preserved by any structural rules that might be used after $\to I$. □

Next is a generalized form of the "Cut" rule.

Lemma 2.3 *If* $\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n \mid \iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_m{:}\theta_m \vdash P{:}\theta$ *and, for all* $1 \le i \le m$, $\Pi_i \mid \Gamma_i \vdash Q_i{:}\theta_i$, *then*

$$\Pi_1, \ldots, \Pi_m, \Gamma_1, \ldots, \Gamma_n \mid \Gamma_{n+1}, \ldots, \Gamma_m \vdash [P](\iota_1 \mapsto Q_1, \ldots, \iota_m \mapsto Q_m){:}\theta \ .$$

Proof. The proof is by induction on the size of the derivation for P. We discuss only the key cases of structural rules that make use of the separation of a type assignment into zones.

Case Contraction: the last step is

$$\frac{\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n, \iota{:}\theta_n \mid \iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_m{:}\theta_m \vdash P'{:}\theta}{\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n \mid \iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_m{:}\theta_m \vdash P{:}\theta}$$

where $P = [P'](\iota \mapsto \iota_n)$. By the induction hypothesis,

$$\begin{aligned} &\Pi_1, \ldots, \Pi_m, \Pi, \Gamma_1, \ldots, \Gamma_n, \Gamma \mid \Gamma_{n+1}, \ldots, \Gamma_m \\ \vdash\quad &[P'](\iota_1 \mapsto Q_1, \ldots, \iota_m \mapsto Q_m, \iota \mapsto Q){:}\theta \end{aligned}$$

where $\Pi \mid \Gamma \vdash Q{:}\theta_n$ is a variant of $\Pi_n \mid \Gamma_n \vdash Q_n{:}\theta_n$ with fresh identifiers not appearing in any Π_i or Γ_i. Then, Π and Γ, being in the passive zone, can be contracted to Π_n and Γ_n, respectively, using Contractions (and Exchanges), and the resulting judgement is the desired conclusion.

Case Activation. The last rule is

$$\frac{\iota_1{:}\theta_1, \ldots, \iota_{n+1}{:}\theta_{n+1} \mid \iota_{n+2}{:}\theta_{n+2}, \ldots, \iota_m{:}\theta_m \vdash P{:}\theta}{\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n \mid \iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_m{:}\theta_m \vdash P{:}\theta}$$

By the induction hypothesis,

$$\Pi_1, \ldots, \Pi_m, \Gamma_1, \ldots, \Gamma_{n+1} \mid \Gamma_{n+2}, \ldots, \Gamma_m \vdash [P](\iota_1 \mapsto Q_1, \ldots, \iota_m \mapsto Q_m){:}\theta$$

Using a number of applications of Activation, we can move Γ_{n+1} to the right of $\mid$, obtaining the desired conclusion

$$\Pi_1, \ldots, \Pi_m, \Gamma_1, \ldots, \Gamma_n \mid \Gamma_{n+1}, \ldots, \Gamma_m \vdash [P](\iota_1 \mapsto Q_1, \ldots, \iota_m \mapsto Q_m){:}\theta \ .$$

Case Passification. The last rule is

$$\frac{\iota_1{:}\theta_1, \ldots, \iota_{n-1}{:}\theta_{n-1} \mid \iota_n{:}\theta_n, \ldots, \iota_m{:}\theta_m \vdash P{:}\phi}{\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n \mid \iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_m{:}\theta_m \vdash P{:}\phi}$$

By the induction hypothesis,

$$\Pi_1, \ldots, \Pi_m, \Gamma_1, \ldots, \Gamma_{n-1} \mid \Gamma_n, \ldots, \Gamma_m \vdash [P](\iota_1 \mapsto Q_1, \ldots, \iota_m \mapsto Q_m) : \phi \ .$$

Because ϕ is passive (as Passification was the last rule), we can use Passification a number of times to move Γ_n to the left of $|$, and we obtain the desired conclusion. □

We can now prove the following desired result: if $\Pi \mid \Gamma \vdash (\lambda\iota{:}\theta'.P)Q{:}\theta$ then $\Pi \mid \Gamma \vdash [P](\iota \mapsto Q){:}\theta$. For the proof, first note that if a derivation ends in an application $M(N)$ then there are only a number of possibilities for the last rule. These are: $\to E$ and the structural rules of Contraction, Exchange, Weakening, Passification, and Activation. Further, the structure of such a derivation must always consist, at the end, of an instance of $\to E$, followed by a number of applications of these other rules. The proof goes by induction on the size of this last part of the derivation, after the final elimination rule.

The basis case when the last rule is of the form

$$\frac{\Pi \mid \Gamma \vdash \lambda\iota{:}\theta'.\, P{:}\theta' \to \theta \quad \Pi' \mid \Gamma' \vdash Q{:}\theta'}{\Pi, \Pi' \mid \Gamma, \Gamma' \vdash (\lambda\iota{:}\theta'.\, P)Q{:}\theta}$$

follows directly from the two lemmas, taking

$\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n$	to be Π
$\iota_{n+1}{:}\theta_{n+1}, \ldots, \iota_{m-1}{:}\theta_{m-1}$	to be Γ
$\iota_m{:}\theta_m$	to be $\iota{:}\theta'$
$\Pi_i \mid \Gamma_i \vdash Q_i{:}\theta_i$	to be $\mid \iota_i{:}\theta_i \vdash \iota_i{:}\theta_i \quad (1 \le i < m)$
$\Pi_m \mid \Gamma_m \vdash Q_m{:}\theta_m$	to be $\Pi' \mid \Gamma' \vdash Q{:}\theta'$

The inductive steps of the proof of the theorem consist of straightforward verifications that the preservation of typing by a β-reduction is preserved by any use of structural rules. □

2.6 Relation and Non-Relation to Linear Logic

The SCIR type system was inspired by linear logic, specifically in the focus on a restricted use of Contraction. The specific presentation, based on zones, was influenced by LU, but the basic type system was worked out in May 1991 prior to seeing LU. Previously, the syntax worked by "marking" identifiers in typing contexts as being passively used, with Passification and Activation manipulating the marks; the zones are a notational variant of this. This was similar to the marking in [44], except that marking of identifiers was done *without* changing types.

In linear logic, Contraction and Weakening are allowed only for types of the form $!A$, whereas in SCIR Contraction is allowed only for passively-used identifiers (in the passive zone). Furthermore, the Dereliction and Promotion

rules for the passive type constructor $\rightarrow_P$ are obviously inspired by the corresponding linear logic rules for the "!" modality, though they have precursors in Reynolds's original (1978) presentation of SCI. These facts, supported by semantic models, were the basis for the analogy of passivity as "!", and SCI as affine linear logic, proposed in [24,25]. It was known then that the passivity $\sim$! analogy was not an exact correspondence, and that there were some properties of passivity not accounted for by "!".

For example, it would have been possible, in principle, to use a linear logic-based type system to design an alternate type system for SCI satisfying the subject reduction property. But if we had followed up the passivity $\sim$! analogy, the most obvious candidate syntax would have had a form of "boxing" [11]. For example, the Promotion rule for passive procedures would be something like (*cf.* [1])

$$\frac{x_1\colon A_1, \ldots, x_n\colon A_n \mid \vdash Q\colon \theta' \rightarrow \theta \quad \cdots \Delta_i \mid\vdash E_i\colon A_i \cdots}{\Delta_1, \ldots, \Delta_n \mid \vdash \mathbf{promote}\, E_1, \ldots, E_n\, \mathbf{for}\, x_1, \ldots, x_n\, \mathbf{in}\, Q\colon \theta' \rightarrow_P \theta} \rightarrow_P I$$

While this syntax is perhaps appropriate for "!" in linear logic, it seems overly heavy, with no conceivable justification, from the point of view of interference control.

More importantly, the concept of passivity involves a notion of *passive use*, which has additional properties beyond those for "!". These extra properties are embodied syntactically in the rules of Passification and Activation, which have the side benefit of allowing us to avoid these syntactic complications, retaining a relatively simple syntax possessing the subject reduction property. (Compare the implicit syntax mentioned above with that just given for Promotion!) These two rules do not correspond to any rules in linear logic, or LU; this difference will be seen again when we consider categorical models of the SCIR type system.

3 Semantics

The permeability rules of Passification and Activitation can exhibit subtle behaviour (as we saw in Section 2.4). To understand this behaviour, it is beneficial to have an analysis that exposes their essential structure in more abstract terms. To this end, in this section we define a class of categorical models of the SCIR type system. We do not attempt to formulate a most general possible notion of model. Rather, we focus on a particularly cohesive class, which we term "bireflective" models, that are sufficient to secure our basic aim of showing a *sound* interpretation which accounts for the permeability rules.

A concrete model for the programming language of Section 2.3 will be presented in Section 4.

3.1 Bireflective Models

As usual, the types and terms of the language are to be interpreted as objects and morphisms, respectively, of an appropriate semantic category $\mathbf{C}$. We require, first, that $\mathbf{C}$ come equipped with a symmetric monoidal closed structure $(I, \otimes, \multimap)$, and finite products. This enables us to interpret the non-interfering product, the interfering product, and function types in standard ways. For example, the closed structure will provide application maps

$$app(A, B)\colon (A \multimap B) \otimes A \to B$$

for all objects A and B, and, for every map $f\colon A \otimes B \to C$, a curried map

$$f^*\colon A \to (B \multimap C)$$

satisfying appropriate β and η laws.

Typing contexts Π, Γ to the left of $\vdash$ in any syntax judgement will be interpreted as products built using $\otimes$:

$$[\![\iota_1\colon\theta_1, \ldots, \iota_n\colon\theta_n]\!] \;=\; [\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!]$$

To interpret the Weakening rule, the tensor product $\otimes$ must allow for *projection* maps, $\pi_0^{\otimes}\colon A \otimes B \to A$ and $\pi_1^{\otimes}\colon A \otimes B \to B$. We therefore require the unit I for $\otimes$ be a terminal object 1 of $\mathbf{C}$; then π_0 is $(\mathrm{id}_A \otimes !_B)\,;\varrho$, where $!_B$ is the unique map from B to 1, and $\varrho\colon A \otimes 1 \to A$ is the unity isomorphism, and similarly for $\pi_1^{\otimes}$.

To treat passivity, we begin by assuming a full subcategory $\mathbf{P}$ of $\mathbf{C}$, to be thought of as the subcategory of passive objects. The typing context in the passive zone will be interpreted as a passive object. Thus, every judgement $\Pi \mid \Gamma \vdash P\colon\theta$ will be interpreted by a map

$$S[\![\Pi]\!] \otimes [\![\Gamma]\!] \longrightarrow [\![\theta]\!]$$

where $S[\![\Pi]\!]$ is an object of $\mathbf{P}$, and $[\![\Gamma]\!]$ and $[\![\theta]\!]$ are objects of $\mathbf{C}$. To treat both Contraction and Weakening in the passive zone, we simply require that $\otimes$ be a categorical product in $\mathbf{P}$. The interactions of permeability rules and rules for the passive function type are accounted for by making a further assumption on $\mathbf{P}$.

Definition 3.1 (Bireflective Subcategory) A *bireflective subcategory* of a category $\mathbf{C}$ is a full subcategory $\mathbf{P}$ of $\mathbf{C}$ with inclusion $J\colon\mathbf{P} \hookrightarrow \mathbf{C}$ that has left and right adjoints equal, say $S\colon\mathbf{C} \to \mathbf{P}$, with the composite

$$JSA \xrightarrow{\varepsilon'_A} A \xrightarrow{\eta_A} JSA$$

being the identity, where η is the unit of the adjunction $S \dashv J$ and ε'_A is the counit of $J \dashv S$.

This definition is from [10], where its categorical properties are studied. Our main concern here is to explain its connection to the SCIR type system.

The adjunction $S \dashv J$ is used to interpret the permeability rules of Passification and Activation. For Passification, consider first the special case in which there is only one identifier in the active zone and none in the passive zone:

$$\frac{\mid \iota{:}\theta \vdash M{:}\phi}{\iota{:}\theta \mid \vdash M{:}\phi}$$

The adjunction determines a transformation of maps

$$\frac{f{:}A \to JP}{\mathit{passify}(f){:}SA \to P}$$

where P is any object of $\mathbf{P}$, and A is an arbitrary $\mathbf{C}$-object. This interprets the indicated instance of the rule, and unit of the left adjunction gives us a natural family of maps $\eta_A{:}A \to SA$ to interpret an instance

$$\frac{\iota{:}\theta \mid \vdash M{:}\theta'}{\mid \iota{:}\theta \vdash M{:}\theta'}$$

of the Activation rule by pre-composition:

$$\frac{f{:}SA \to B}{\eta_A; f{:}A \to B}$$

Instances of these rules involving more than one contextual identifier can be dealt with by assuming that S be a strong monoidal functor; i.e., that it preserves tensor products up to (coherent) isomorphism: $S(A \otimes B) \cong SA \otimes SB$ and $S1 \cong 1$ [9,18].

The right adjunction $J \dashv S$ is utilized in the treatment of $\to_P$. Clearly, we would like $\to_P$ to behave like a function type. But, as evidenced by the introduction rule $\to_P I$, these functions are subject to constraints ensuring the passive use of free identifiers within them. If we set $A \to_P B = S(A \multimap B)$ then, using $J \dashv S$, this determines an adjunction

$$\frac{JP \otimes A \to B}{P \to [A \to_P B]}$$

where P is a passive object. (That is, $(\text{–}) \otimes A{:}\mathbf{P} \to \mathbf{C}$ is left adjoint to $S(A \multimap (\text{–}))$, for all $\mathbf{C}$-objects A.) Thus, we have an interpretation of $\to_P$ that takes into account both passive use and functional properties such as β and η.

The further requirement of bireflectivity—the coincidence of the left and right adjoints to J and the coherence condition—implies certain equations relating the left and right adjunctions. First, as the analysis in [10] shows, bireflectivity implies that the transformation of maps $f \mapsto \mathit{passify}(f)$ associated with the left adjunction $S \dashv J$ can be calculated using the counit $\varepsilon'_A{:}SA \to A$ (where $SA = JSA$) of the right adjunction $J \dashv S$:

$$\mathit{passify}(f) \;=\; \varepsilon'_A \,;\, f \tag{3}$$

where $f{:}A \to P$. Similarly, the transformation associated with the right adjunction

$$\frac{g{:}P \to A}{\mathit{promote}(g){:}P \to SA}$$

can be calculated using the unit $\eta_A : A \to SA$ (where $SA = JSA$) of the left adjunction:

$$promote(g) \;=\; g \,;\, \eta_A \;. \tag{4}$$

The simplifying effect of these equations is dramatic.

For instance, in [1] it is emphasized that naturality requirements lead to a syntactic treatment of promotion rules such as $\to_P I$ that involve binding, much like the rule discussed in Section 2.6. But by interpreting $\to_P I$ using composition on the right, as in equation (4), all necessary naturality requirements are met by the simpler form of syntax rule that we use. Similarly, the interpretation of the Passification rule can now be given simply by composing on the left as indicated by (3). This will be a great aid in establishing the connection between model and syntax, as given by the coherence theorem below.

Definition 3.2 (Bireflective Model) A *bireflective model of SCIR* is given by the following data:

(i) a symmetric monoidal closed category $(\mathbf{C}, 1, \otimes, \multimap)$ with finite products $(1, \times)$; and

(ii) a bireflective subcategory $J : \mathbf{P} \hookrightarrow \mathbf{C}$ in which $(1, \otimes)$ is a finite-product structure and the bireflector $S : \mathbf{C} \to \mathbf{P}$ is a strong symmetric monoidal functor for which $S \dashv J \dashv S$ are monoidal adjunctions.

Note that, since we have required that $\otimes$ be a cartesian product structure in the full subcategory $\mathbf{P}$, the category $\mathbf{P}$ is monoidal and the inclusion J is a strong monoidal functor with comparison morphisms $JP \otimes JQ \to J(P \otimes Q)$ and $1 \to J1$ being identities. An adjunction is monoidal when certain equations hold involving the units and counits and the comparison morphisms $SA \otimes SB \to S(A \otimes B)$ and $1 \to S1$ [9,18]. Monoidal functors and adjunctions are useful for treating rules involving typing contexts [1].

The conditions that S be strong monoidal and that $S \dashv J$ and $J \dashv S$ be monoidal adjunctions are equivalent to the condition that, for A and B in $\mathbf{C}$,

$$\begin{array}{ccc} A \otimes B & \xrightarrow{\eta_{A\otimes B}} & JS(A \otimes B) \\ {\scriptstyle \eta_A \otimes \eta_B}\downarrow & & \downarrow{\scriptstyle \epsilon'_{A\otimes B}} \\ JSA \otimes JSB & \xrightarrow[\epsilon'_A \otimes \epsilon'_B]{} & A \otimes B \end{array}$$

commutes, where η is the unit of $S \dashv J$ and ϵ' is the counit of $J \dashv S$.

To simplify the presentation, we assume that the counit ϵ of $S \dashv J$ is the identity, and identify P with JP in $\mathbf{C}$. Then the isomorphism $m_{A,B} : SA \otimes SB \to S(A \otimes B)$ associated with the strong monoidal functor S can be written

$$SA \otimes SB \xrightarrow{\epsilon'_A \otimes \epsilon'_B} A \otimes B \xrightarrow{\eta_{A\otimes B}} S(A \otimes B)$$

with inverse $\epsilon'_{A\otimes B}; \eta_A \otimes \eta_B$, and $m_1 : 1 \to S1$ is η_1.

Notice that the units of the monoidal and cartesian structures coincide. The adjunction $J \dashv S$ determines a co-monad on $\mathbf{C}$, and this is the aspect of

passivity that is similar (but not identical) to "!" from linear logic. The left adjoint to J determines additional structure, that of a monad.

Proposition 3.3 *$SP \cong P$ for all passive P, and hence S is idempotent.*

Proof. Standard for reflective subcategories; see [20]. □

Proposition 3.4 [10]

(i) $\mathbf{P}$ *is Cartesian closed.*

(ii) $P \times Q \cong P \otimes Q$ *when P and Q are $\mathbf{P}$-objects.*

(iii) $\mathbf{P}$ *is an "exponential ideal" of $\mathbf{C}$; i.e. $A \multimap P$ lies in $\mathbf{P}$ (up to isomorphism) when P is a $\mathbf{P}$-object and A is any $\mathbf{C}$-object.* □

Part 1 of the proposition corresponds to the following intuition: the passive fragment of SCIR has no interference constraints, and so a model of this fragment should be a model of the full typed λ-calculus. Parts 2 and 3 correspond to the syntactic classification of passive types. For instance, types of the form $\theta \to_P \phi$ and $\theta \to \phi$ are isomorphic, so that the two exponentials coincide for passive result types.

The adjunction $S \dashv J$ could be used to show that "passifying all variables" is bijective, but we also want to passify one variable at a time. That "passifying one variable is bijective" is the content of the following.

Lemma 3.5 *There is a bijection*

$$\frac{f : JQ \otimes A \otimes B \to JP}{(\mathrm{id} \otimes \varepsilon'_A \otimes \mathrm{id}); f : JQ \otimes JSA \otimes C \to JP}$$

where P and Q are passive objects.

Proof. Immediate from properties of monoidal functors and adjunctions, or it can also be proven directly using the fact that $\mathbf{P}$ is an exponential ideal. □

Example 3.6 This is essentially from [23], and is related to the functor-category model given later which is based on [43,27].

Let $\mathbf{N}$ be the category with a single object, $*$, and where the morphisms are natural numbers together with an extra number ∞. The composition $m\,;n$ is the minimum of m and n, with $m\,;\infty = \infty\,;m = m$. The functor category $\mathbf{Sets}^{\mathbf{N}}$ is a model of SCIR.

The category $\mathbf{P}$ of passive objects is the subcategory of constant functors, where each morphism in $\mathbf{N}$ gets mapped to an identity. Functor $S\colon \mathbf{Sets}^{\mathbf{N}} \to \mathbf{P}$ is given by $S(A)* = \{A(0)a \mid a \in A(*)\}$ and $SA(m)a = S(0)a$. The functors SA are constant because $0\,;0 = 0$. Given a map $f\colon A \dot{\to} P$, the corresponding map $f'\colon SA \dot{\to} P$ is given by $f'(*)a = (f(*))(A(0)a)$. The adjunction $J \dashv S$ is given by composing with the inclusion $SA \to A$.

To give some intuition, consider a "locations" functor $\mathrm{Loc}\colon \mathbf{N} \to \mathbf{Sets}$. $\mathrm{Loc}(*)$ is the set of natural numbers, together with an extra element $-$. For natural numbers n and m, $\mathrm{Loc}(n)m = m$ if $m < n$, and $-$ otherwise, and $\mathrm{Loc}(\infty)m = m$. One may think of function $\mathrm{Loc}(n)$ as "disallowing access"

to locations greater than or equal to n, by mapping these locations to $-$. $S(\mathrm{Loc})(*)$ has only one element, $-$.

In this category we can begin to see a glimpse of semantic structure related to side effects. But the category $\mathbf{P}$ does not quite match computational intuitions concerning passivity. It consists of constant functors, which are effectively stateless. State will be better treated in Section 4 by adopting a category of worlds with multiple objects (to account for local state) to use in place of $\mathbf{N}$.

3.2 Interpretation of the Typing Rules

In this section, we explain how typing rules are interpreted in any bireflective model of SCIR. Each of the primitive types θ is interpreted as an object $[\![\theta]\!]$ of $\mathbf{C}$, with passive primitive types interpreted as objects of sub-category $\mathbf{P}$. This then determines interpretations of non-primitive types, as follows:

$$[\![\theta \times \theta']\!] = [\![\theta]\!] \times [\![\theta']\!] \qquad [\![\theta \to \theta']\!] = [\![\theta]\!] \multimap [\![\theta']\!]$$

$$[\![\theta \otimes \theta']\!] = [\![\theta]\!] \otimes [\![\theta']\!] \qquad [\![\theta \to_P \theta']\!] = S([\![\theta]\!] \multimap [\![\theta']\!])$$

It is clear that each syntactically passive type is interpreted as an object in $\mathbf{P}$ (or an object isomorphic to an object in $\mathbf{P}$).

Each typing judgement $\Pi \mid \Gamma \vdash P{:}\theta$ is interpreted as a morphism from $S[\![\Pi]\!] \otimes [\![\Gamma]\!]$ to $[\![\theta]\!]$, where for any typing context $\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n$,

$$[\![\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n]\!] \;=\; [\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!]$$

and where by $S[\![\Pi]\!]$ we mean explicitly

$$S[\![\iota_1{:}\theta_1, \ldots, \iota_n{:}\theta_n]\!] \;=\; S[\![\theta_1]\!] \otimes \cdots \otimes S[\![\theta_n]\!]$$

In effect, we are bypassing the isomorphism $S(A \otimes B) \cong SA \otimes SB$ in the presentation, and we are glossing over associativity and unity isomorphisms. We are most concerned with an analysis of the rules of Passification, Activation, and Contraction, and so will concentrate for the most part on these.

The interpretation goes by induction on derivations, so we are assigning a meaning $[\![\Psi]\!]$ to each *proof* Ψ of a typing judgement.

The Axiom and the structural rules of Weakening and Exchange are treated in the standard way, using identities $\mathrm{id}_{[\![\theta]\!]}{:}\,[\![\theta]\!] \to [\![\theta]\!]$, weakenings $[\![\theta]\!] \to 1$, and symmetries $A \otimes B \to B \otimes A$, respectively.

For Activation, suppose $f{:}\,S[\![\Pi, \iota{:}\theta]\!] \otimes [\![\Gamma]\!] \to [\![\theta']\!]$; then we define the desired map from $S[\![\Pi]\!] \otimes [\![\iota{:}\theta, \Gamma]\!]$ to $[\![\theta']\!]$ as the following composite:

$$\begin{array}{ccc} S[\![\Pi]\!] \otimes S[\![\theta]\!] \otimes [\![\Gamma]\!] & \xrightarrow{\quad f \quad} & [\![\theta']\!] \\ \uparrow \mathrm{id} \otimes \eta[\![\theta]\!] \otimes \mathrm{id} & & \\ S[\![\Pi]\!] \otimes [\![\theta]\!] \otimes [\![\Gamma]\!] & & \end{array}$$

where $\eta(A) = \mathit{passify}^{-1}(\mathrm{id}_A)$ is the unit of the adjunction $S \dashv J$.

For Passification, suppose $f{:}\,S[\![\Pi]\!] \otimes [\![\iota{:}\theta, \Gamma]\!] \to [\![\phi]\!]$. The interpretation is

$$\begin{array}{ccc} S[\![\Pi]\!] \otimes [\![\theta]\!] \otimes [\![\Gamma]\!] & \xrightarrow{\quad f \quad} & [\![\theta']\!] \\ \uparrow{\scriptstyle \mathrm{id} \otimes \varepsilon'[\![\theta]\!] \otimes \mathrm{id}} & & \\ S[\![\Pi]\!] \otimes S[\![\theta]\!] \otimes [\![\Gamma]\!] & & \end{array}$$

where ε' is the counit of $J \dashv S$. This interpretation is possible because of equation (3).

For Contraction, suppose $f\colon S[\![\Pi, \iota\colon\theta, \iota'\colon\theta]\!] \otimes [\![\Gamma]\!] \to [\![\theta']\!]$; then we define the desired map from $S[\![\Pi, \iota\colon\theta]\!] \otimes [\![\Gamma]\!]$ to $[\![\theta']\!]$ as follows:

$$\begin{array}{ccc} S[\![\Pi]\!] \otimes S[\![\theta]\!] \otimes S[\![\theta]\!] \otimes [\![\Gamma]\!] & \xrightarrow{\quad f \quad} & [\![\theta']\!] \\ \uparrow{\scriptstyle \mathrm{id} \otimes \mathit{duplicate}(S[\![\theta]\!]) \otimes \mathrm{id}} & & \\ S[\![\Pi]\!] \otimes S[\![\theta]\!] \otimes [\![\Gamma]\!] & & \end{array}$$

Here, *duplicate* is the diagonal map for the cartesian structure in $\mathbf{P}$.

For rule $\to I$, suppose that $f\colon S[\![\Pi]\!] \otimes [\![\Gamma, \iota\colon\theta']\!] \to [\![\theta]\!]$; then the desired map is

$$f^*\colon S[\![\Pi]\!] \otimes [\![\Gamma]\!] \to \Big([\![\theta']\!] \multimap [\![\theta]\!]\Big)$$

where f^* is the currying of f, as discussed in Section 3.1. For rule $\to E$, suppose $f_0\colon S[\![\Pi_0]\!] \otimes [\![\Gamma_0]\!] \to \Big([\![\theta']\!] \multimap [\![\theta]\!]\Big)$ and $f_1\colon S[\![\Pi_1]\!] \otimes [\![\Gamma_1]\!] \to [\![\theta']\!]$; then the desired map is

$$\begin{array}{ccc} S[\![\Pi_0]\!] \otimes [\![\Gamma_0]\!] \otimes S[\![\Pi_1]\!] \otimes [\![\Gamma_1]\!] & \xrightarrow{\quad f_0 \otimes f_1 \quad} & \Big([\![\theta']\!] \multimap [\![\theta]\!]\Big) \otimes [\![\theta']\!] \\ \uparrow{\scriptstyle \gamma} & & \downarrow{\scriptstyle \mathit{app}([\![\theta']\!], [\![\theta]\!])} \\ S[\![\Pi_0]\!] \otimes S[\![\Pi_1]\!] \otimes [\![\Gamma_0]\!] \otimes [\![\Gamma_1]\!] & & [\![\theta]\!] \end{array}$$

where *app* is the application map discussed in Section 3.1 and γ is the evident isomorphism.

For rule $\to_P I$, suppose $f\colon S[\![\Pi]\!] \to [\![\theta \to \theta']\!]$; then the desired map is

$$\begin{array}{ccc} S[\![\Pi]\!] & \xrightarrow{\quad f \quad} & [\![\theta']\!] \multimap [\![\theta]\!] \\ & & \downarrow{\scriptstyle \eta([\![\theta']\!] \multimap [\![\theta]\!])} \\ & & S\Big([\![\theta']\!] \multimap [\![\theta]\!]\Big) \end{array}$$

where $\eta(A)\colon A \to SA$. This interpretation utilizes equation (4).

For rule $\to_P E$, suppose $f\colon S[\![\Pi]\!] \otimes [\![\Gamma]\!] \to S\Big([\![\theta']\!] \multimap [\![\theta]\!]\Big)$; then the desired map is

$$\begin{array}{ccc} S[\![\Pi]\!] \otimes [\![\Gamma]\!] & \xrightarrow{\ f\ } & S\big([\![\theta']\!] \multimap [\![\theta]\!]\big) \\ & & \Big\downarrow \varepsilon'([\![\theta']\!] \multimap [\![\theta]\!]) \\ & & [\![\theta']\!] \multimap [\![\theta]\!] \end{array}$$

where $\varepsilon'(A)\colon SA \to A$ is the counit of $J \dashv S$, definable as $promote^{-1}(\mathrm{id}_{SA})$.

The remaining rules, for tensor and categorical products, can be treated in an obvious way. Each constant is interpreted by a map out of the terminal object.

3.3 Coherence

Notice that the presence of structural rules in the type system allows for multiple proofs of a typing judgement, and it is important to show that this does not lead to semantic ambiguity. In this section we verify that the semantics is in fact *coherent*; i.e., all proofs of any syntax judgement have the same interpretation.

Theorem 3.7 (Coherence) *Let Ψ_0 and Ψ_1 be proofs of $\Pi \mid \Gamma \vdash P\colon\theta$; then $[\![\Psi_0]\!] = [\![\Psi_1]\!]$.*

The proof occupies the remainder of this section. It will be convenient to have a notation for certain composite proofs. Suppose Ψ is a proof of a judgement $\Pi \mid \Gamma \vdash Q\colon\theta$, and that we can extend Ψ by applications Ψ' of only the structural rules of Contraction, Exchange, Weakening, Activation and Passification to obtain a proof of $\Pi' \mid \Gamma' \vdash Q'\colon\theta$. We write $\Psi\,;\Psi'$ for the composite proof, and call Ψ' a *structural extension* of Ψ.

Notice that, because all structural rules are interpreted by composing on the left, the denotation of any proof $\Psi;\Psi'$ of $\Pi' \mid \Gamma' \vdash P' : \theta$ can be decomposed so that

$$[\![\Psi\,;\Psi']\!] \;=\; h \;;\; [\![\Psi]\!]$$

for a map $h\colon S[\![\Pi']\!] \otimes [\![\Gamma']\!] \to S[\![\Pi]\!] \otimes [\![\Gamma]\!]$ induced by structural rules in Ψ'. We often write $[\![\Psi']\!]$ to denote a map of this form induced by a proof extension. If Ψ' is empty then we declare $[\![\Psi']\!]$ to be the identity.

One important property to isolate is coherence of structural extensions.

Lemma 3.8 (Coherence of Structural Extensions) *Suppose that Ψ is a proof of $\Pi \mid \Gamma \vdash Q\colon\theta$, and that $\Psi\,;\Psi_1$ and $\Psi\,;\Psi_2$ are structural extensions that prove judgement $\Pi' \mid \Gamma' \vdash Q'\colon\theta$; then $[\![\Psi\,;\Psi_1]\!] = [\![\Psi\,;\Psi_2]\!]$.*

This is really a statement about the maps induced by structural extensions, and is independent of Ψ, Q, and Q'. A structural extension determines a function ρ from variables in $\Pi \mid \Gamma$ to those in $\Pi' \mid \Gamma'$ with $\rho(x)$ being the variable to which x contracts. (We omit a formal definition, which is a simple induction on derivations). The desired result, with data as in the statement

of the lemma, is then:

(∗) If structural extensions Ψ_1 and Ψ_2 determine the same ρ, then
  (A) $\varepsilon' ; [\![\Psi_1]\!] = \varepsilon' ; [\![\Psi_2]\!]$, where ε' here is an appropriate component of the counit of $J \dashv S$, and
  (B) $[\![\Psi_1]\!] = [\![\Psi_2]\!]$ if θ is non-passive (so neither derivation uses Passification).

It is easy to verify that this formulation (which now has more the flavour of a categorical coherence result) implies the Coherence of Structural Extensions. Note that we cannot generally ask for equality of the $[\![\Psi_i]\!]$ (because of Passification). In cases where θ is passive, we use (A) and the property $f = g : A \to JP$ iff $\varepsilon' ; f = \varepsilon' ; g$ to conclude the lemma.

We indicate the proof of (∗).

Proof. Given θ and a function ρ from $\Pi \mid \Gamma$ to $\Pi' \mid \Gamma'$, we can define a canonical extension Ψ_1 (that determines ρ) as follows.

(i) Passify all identifiers if θ is passive.

(ii) Perform all Contractions indicated by ρ.

(iii) Activate all variables in the intersection of the image of ρ and the domain of Γ'.

(iv) Perform appropriate Weakenings for variables not in the image of ρ.

Step (ii) assumes that all Contractions indicated by ρ are for identifiers in the passive zone (this is an assumption on ρ and θ).

We thus obtain an extension $\Psi_1 = P ; C ; A ; W$ consisting of Passifications, followed by Contractions, Activations, and Weakenings (with some Exchanges sprinkled throughout). We prove the property (∗), for Ψ_1 a canonical extension, by induction on the length of Ψ_2. We consider two sample cases.

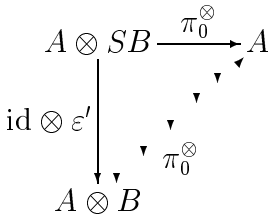
Base case: length 0. $[\![\Psi_2]\!]$ is the identity, whereas Ψ_1 is either empty or a sequence $P ; A$ of Passifications and Activations (if θ is passive). (B) is trivial, and (A) follows from the identity $\eta ; \varepsilon' ; f = f$, where $f : X \to JP$. This equation in turn follows from the identities $\varepsilon' ; f = \mathit{passify}(f)$ and $\eta ; \mathit{passify}(f) = f$, the former a consequence of bireflectivity and the latter of $S \dashv J$.

Case: last rule is Passification. Part (B) is trivial. For (A), the induction hypothesis gives us $\varepsilon' ; [\![\Psi_1']\!] = \varepsilon' ; [\![\Psi_2']\!]$, where $\Psi_1' = P' ; C' ; A' ; W'$ is canonical and $\Psi_2 = \Psi_2' ; p$ with p an instance of Passification. Suppose that x is the identifier moved by p. There are three subcases to consider:

1. no rule in Ψ_1' explicitly involves x,
2. x was introduced in the active zone through a Weakening step in W', or
3. x was moved into the active zone through an Activation step in A'.

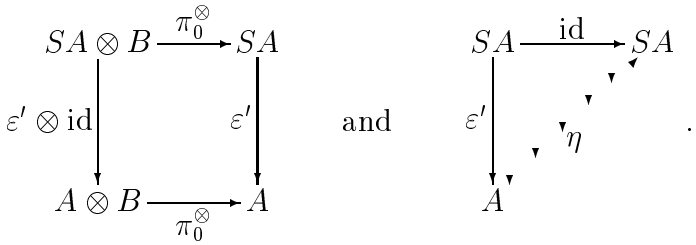
In subcase 1 we mean that x is not moved by Passification or Activation, or introduced via Contaction or Weakening. Clearly one of these three cases must apply: note that if x was involved in Contraction, Activation, or Passification, then subcase 3 would apply. Subcase 1 is straightforward since x is interpreted by an identity in $[\![\Psi_1']\!]$; we concentrate on 2 and 3.

For subcase 2, we can replace the instance of Weakening that introduces x in W' by another instance that puts x in the passive zone, giving us W''. Then $[\![W'']\!] = [\![W'; p]\!]$ because of the identity



Thus, $[\![\Psi_2]\!] = [\![P'; C'; A'; W'']\!]$, and $P'; C'; A'; W''$ is of the form prescribed above for the canonical extension. Simple permutations within each component P', A', C', W'' suffice to show that it is semantically equal to the prescribed extension (in any case, there is some trivial imprecision, involving order of rules, in the prescription (i)-(iv) for extensions).

For subcase 3, we first move p to the left of W', and then compose the resulting instance of Passification with the instance of A' that activates x; this composition yields the identity. The involved equations for this are



Thus $[\![P'; C'; A''; W'']\!] = [\![P'; C'; A'; W'; p]\!]$ where A'' has the mentioned occurrence of Activation removed (so later rules in A'' and W'' are slightly adjusted), and the desired result follows as in subcase 2.

Other rules are treated in a similar fashion, using the induction hypothesis and various identities to reduce a proof to a canonical extension. □

With coherence of structural extensions, we may deduce the desired theorem as a corollary of the following result.

Lemma 3.9 *Suppose* $\Pi \mid \Gamma \vdash P{:}\theta$ *is derivable both from* $\Pi_0 \mid \Gamma_0 \vdash P_0{:}\theta$ *and from* $\Pi_1 \mid \Gamma_1 \vdash P_1{:}\theta$*, using only the structural rules. Suppose further that, for* $i = 0, 1$, Ψ_i *is any proof of* $\Pi_i \mid \Gamma_i \vdash P_i{:}\theta$*; then*

$$[\![\Psi_0 \,;\, \Psi_0']\!] = [\![\Psi_1 \,;\, \Psi_1']\!] : S[\![\Pi]\!] \otimes [\![\Gamma]\!] \longrightarrow [\![\theta]\!]$$

for all structural extensions Ψ_i' *such that* $\Psi_i \,;\, \Psi_i'$ *proves* $\Pi \mid \Gamma \vdash P{:}\theta$*, for* $i = 0, 1$.

Note that, for $i = 0, 1$, $P = [P_i]\sigma_i$ for identifier substitutions σ_i introduced by Contractions.

Proof. The proof is by induction on the sum of the sizes of proofs Ψ_0 and Ψ_1.

The main base case is when Ψ_0 and Ψ_1 are both instances of the Axiom for identifiers. This case follows from the coherence of structural extensions.

The other base cases, for constants, are immediate if any constant $C{:}\theta$ is interpreted as a map $[\![C]\!]{:}\, 1 \to [\![\theta]\!]$.

If the last step in Ψ_0 is an instance of a structural rule then we prove the result as follows. Suppose that RR is the last rule applied in Ψ_0, and consider any appropriate structural extensions Ψ_i', $i = 0, 1$. We want to show

$$[\![\Psi_0 \,;\, \Psi_0']\!] = [\![\Psi_1 \,;\, \Psi_1']\!] {:}\, S[\![\Pi]\!] \otimes [\![\Gamma]\!] \longrightarrow [\![\theta]\!] \ .$$

Since the last rule in Ψ_0 is RR, which is one of the rules permissible in proof extensions, this means that $\Psi_0 \,;\Psi_0'$ is the same proof as $\Psi_2 \,;\Psi_2'$, where Ψ_2 is Ψ_0 with the final instance of RR stripped off and Ψ_2' is Ψ_0' with the corresponding instance of RR placed on the front. (We have simply moved the break-point ";" indicating a structural extension.) Since the proof Ψ_2 is smaller than Ψ_0, the induction hypothesis applies and we may conclude

$$[\![\Psi_2 \,;\, \Psi_2']\!] = [\![\Psi_1 \,;\, \Psi_1']\!] {:}\, S[\![\Pi]\!] \otimes [\![\Gamma]\!] \longrightarrow [\![\theta]\!] \ .$$

The result follows from the identity $\Psi_0 \,;\Psi_0' = \Psi_2 \,;\Psi_2'$. The case when Ψ_1 ends in a structural rule is symmetric.

The only remaining cases are when both Ψ_0 and Ψ_1 end in a non-structural rule for a type constructor. There are two groups of rules to consider: those that involve disjoint hypotheses, and those that do not.

For the latter group, we consider one example: $\times I$. Suppose the last rules of Ψ_0 and Ψ_1 are $\times I$, with proofs Ψ_{ij} of their premises.

$$\frac{\begin{array}{cc}\Psi_{00} & \Psi_{01} \\ \vdots & \vdots \\ \Pi_0 \mid \Gamma_0 \vdash P_0 : \theta & \Pi_0 \mid \Gamma_0 \vdash Q_0 : \theta\end{array}}{\Pi_0 \mid \Gamma_0 \vdash \langle P_0, Q_0 \rangle : \theta \times \theta'} \qquad \frac{\begin{array}{cc}\Psi_{10} & \Psi_{11} \\ \vdots & \vdots \\ \Pi_1 \mid \Gamma_1 \vdash P_1 : \theta & \Pi_1 \mid \Gamma_1 \vdash Q_1 : \theta\end{array}}{\Pi_1 \mid \Gamma_1 \vdash \langle P_1, Q_1 \rangle : \theta \times \theta'}$$

Let $h_k = [\![\Psi_k']\!] : S[\![\Pi_k]\!] \otimes [\![\Gamma_k]\!] \to S[\![\Pi]\!] \otimes [\![\Gamma]\!]$, $k = 0, 1$, be the maps induced by the structural extensions. Then by the induction hypothesis, $h_0 \,;[\![\Psi_{0j}]\!] = h_1 \,;[\![\Psi_{1j}]\!]$, $j = 0, 1$. The desired result is then immediate from the usual identity $h_k \,;\langle f, g\rangle = \langle h_k \,; f, h_k \,; g\rangle$. Other rules not involving disjoint contexts are proven similarly using the induction hypothesis and an additional identity: for $\otimes E$ and $\times E$, use $h \,;(f \,;\pi) = (h \,; f) \,;\pi$; for $\to I$, use $h \,; f^* = ((h \otimes \mathrm{id}) \,; f)^*$; for $\to_P I$, use $h \,;(f \,;\eta) = (h \,; f) \,;\eta$; for $\to_P E$, use $h \,;(f \,;\varepsilon') = (h \,; f) \,;\varepsilon'$.

For the rules involving disjoint contexts we consider $\otimes I$; $\to E$ and $\otimes E$ are similar. In the following, we will content ourselves with skimming over the details of some of the (long) syntactic constructions involved. The basic idea will be to postpone certain Contractions until the end, so that we can apply the induction hypothesis to disjoint terms, and conclude the desired result using the coherence of structural extensions.

Suppose the last rule in each of Ψ_0, Ψ_1 is $\otimes I$, i.e.

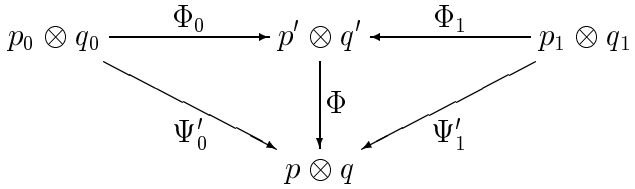
$$\frac{\begin{array}{cc}\Psi_{00} & \Psi_{01} \\ \vdots & \vdots \\ \Pi_0 \mid \Gamma_0 \vdash p_0 {:}\, \theta_0 & \Pi_0' \mid \Gamma_0' \vdash q_0 {:}\, \theta_1\end{array}}{\Pi_0, \Pi_0' \mid \Gamma_0, \Gamma_0' \vdash p_0 \otimes q_0 {:}\, \theta_0 \otimes \theta_1} \qquad \frac{\begin{array}{cc}\Psi_{10} & \Psi_{11} \\ \vdots & \vdots \\ \Pi_1 \mid \Gamma_1 \vdash p_1 {:}\, \theta_0 & \Pi_1' \mid \Gamma_1' \vdash q_1 {:}\, \theta_1\end{array}}{\Pi_1, \Pi_1' \mid \Gamma_1, \Gamma_1' \vdash p_1 \otimes q_1 {:}\, \theta_0 \otimes \theta_1}$$

We have structural extensions Ψ_0' and Ψ_1' to consider, where $\Psi_i \,;\Psi_i'$ proves

$\Pi \mid \Gamma \vdash p \otimes q : \theta_0 \otimes \theta_1$. Since identifiers in p_i and q_i are disjoint, there are (other) structural extensions Φ_i, Φ possessing the following properties.

- $\Psi_i \,;\, \Phi_i$ proves a sequent $\Pi' \mid \Gamma' \vdash p' \otimes q'$, for $i = 0, 1$, where p' and q' have no free identifiers in common, and
- $\Psi_i \,;\, \Phi_i \,;\, \Phi$ proves $\Pi \mid \Gamma \vdash p \otimes q$, for $i = 0, 1$.

That is, we are performing just enough Contractions to identify p_0 and p_1, and q_0 and q_1, postponing the identification of identifiers in both p's and q's until the Φ stage. The reader may wish to use the following picture (where the contexts have been omitted):



Next, from Φ_i we can obtain proofs Φ_{pi} and Φ_{qi} such that $[\![\Phi_{pi}]\!] \otimes [\![\Phi_{qi}]\!] = [\![\Phi_i]\!]$:

$$\dfrac{\begin{array}{cc} \Pi_i \mid \Gamma_i \vdash p_i : \theta_i & \Pi'_i \mid \Gamma'_i \vdash q_i : \theta_i \\ \vdots\ \Phi_{pi} & \vdots\ \Phi_{qi} \\ \Pi_{pi} \mid \Gamma_{pi} \vdash p' : \theta_0 & \Pi_{qi} \mid \Gamma_{qi} \vdash q' : \theta_1 \end{array}}{\Pi' \mid \Gamma' \vdash p' \otimes q' : \theta_0 \otimes \theta_1}$$

These are obtained by copying instances of rules that concern p or q, as appropriate. Finally, we may apply the induction hypothesis to conclude the middle equality in the following

$$\begin{aligned} [\![\Psi_0 \,;\, \Phi_0]\!] &= [\![\Psi_{00} \,;\, \Phi_{p0}]\!] \otimes [\![\Psi_{01} \,;\, \Phi_{q0}]\!] \\ &= [\![\Psi_{10}; \Phi_{p1}]\!] \otimes [\![\Psi_{11} \,;\, \Phi_{q1}]\!] \;= [\![\Psi_1 \,;\, \Phi_1]\!] \end{aligned}$$

where we have suppressed some symmetry isos. The outer two equalities follow from the identity $(h \otimes h');(f \otimes g) = (h;f) \otimes (h';g)$ and the indicated construction of Φ_{pi} and Φ_{qi}. The desired result $[\![\Psi_0;\Psi'_0]\!] = [\![\Psi_1;\Psi'_1]\!]$ then follows immediately from the coherence of structural extensions, using $[\![\Phi_i \,;\, \Phi]\!] = [\![\Psi'_i]\!]$. □

The Coherence theorem then follows directly by taking $\Pi_0 = \Pi_1 = \Pi$, $\Gamma_0 = \Gamma_1 = \Gamma$, and $P_0 = P_1 = P$.

Having established that the semantics is well-defined, we can note that it satisfies the reductions listed in Table 3.

Proposition 3.10 *The reductions in Table 3 preserve equality in any bireflective model of SCIR.* □

For instance, the equivalence $\mathbf{derelict}(\mathbf{promote}\, M) \equiv M$ follows from the identity $f = f;\eta_A;\varepsilon'_A$ where $f : JP \to A$, which is true by virtue of $J \dashv S$ and equation (4). A fuller treatment of equivalences will not be given here. However, it is worth noting that many additional equations beyond these β laws are valid in bireflective models. As one example, one can synthesize an

equivalence from the law of monoidal functors

$$\begin{array}{ccc} SA \otimes SB & \xrightarrow{m_{A,B}} & S(A \otimes B) \\ \downarrow{\scriptstyle \gamma_{SA,SB}} & & \downarrow{\scriptstyle S(\gamma_{A,B})} \\ SB \otimes SA & \xrightarrow[m_{B,A}]{} & S(B \otimes A) \end{array}$$

by replacing $S(-)$ by $C \to_P (-)$, where C is a passive type. For instance, the map $m_{A,B}$ would be replaced by the term

$$\begin{array}{l} \lambda f{:}(C \to_P A) \otimes (C \to_P B)\,. \\ \quad \textbf{let } f_0 \otimes f_1 \textbf{ be } f \textbf{ in} \\ \quad \textbf{promote}(\lambda x : C\,.\,(\textbf{derelict}\, f_0 x) \otimes (\textbf{derelict}\, f_1 x)) \end{array}$$

See [1,3] for discussion.

Verifying coherence proved to be quite a lot of detailed work, even with certain isomorphisms left implicit and with the skimming over of some syntactic constructions. We wonder whether type theoretic coherence could be better approached in a more general setting; see [34] for discussion and references.

3.4 Discussion: Non-Bireflective Models

We have included the qualification "bireflective" in Definition 3.2 because there are models of the SCIR type system in which the left and right adjoints to the inclusion do not coincide. The first, and foremost, examples are given by the models in [35,36]. Others are given, for example, by arrow categories $\mathbf{C}^{\to}$. The models that we know of have the form of two categories and three functors between them, like so:

$$\mathbf{P} \underset{R}{\overset{L}{\underset{\xrightarrow{\quad I \quad}}{\rightleftarrows}}} \mathbf{C}$$

with I fully faithful and $L \dashv I \dashv R$. Additional conditions that a more general (not necessarily bireflective) model of SCIR should satisfy have not been formulated. Coherence is the minimal requirement for any general notion of model of SCIR, and is particularly subtle because of the intricate interactions between the permeability rules and other rules.

4 A Functor-Category Model

In this section, we present a concrete model of the illustrative Algol-like programming language of Section 2.3, This confirms that the categorical analysis using bireflectivity is consistent with a more concrete reading of passivity in terms of read-only access to the computer store.

We emphasize that the aim of the model is not simply to characterize behaviour of complete programs, i.e., closed terms of type **comm**. Such a model could be obtained using a standard "marked stores" model [21] in the category of cpo's and continuous functions, with a trivial bireflective subcategory

structure given by the identity functor on the category. To see why this is so consider first that, if we map $\otimes$ to $\times$ and $\rightarrow_P$ to $\rightarrow$, any term in SCIR is typable in simply-typed λ-calculus. Then a standard model for Idealized Algol can be used, allowing for side effects in expressions (to account for the block expression **do**) and interpreting parallel composition as if it were sequential. But while such a model would correctly predict observable behaviour and would satisfy an adequacy correspondence, with a suitable operational semantics, it would not make manifest the principle that distinct identifiers don't interfere. Furthermore, the passive function type $\rightarrow_P$ would be semantically equivalent to $\rightarrow$, and the model would not show the sense in which expressions, and in particular the block expression, are free from side effects. That is, the semantics would fail to elucidate the most important aspects of the language.

We desire a semantics that makes the consequences of the syntactic restrictions clear. For instance, if the principle that distinct identifiers don't interfere is built into the semantics, so the *only* environments are ones adhering to the principle, then it will be evident that $C_1 \parallel C_2$ is deterministic. It will then turn out that $x; y$ and $y \parallel x$ are equivalent, but this fact, which could in hindsight be assumed by a semantics, is not so interesting as the reason for it; namely, that x and y don't interfere. Similarly, we desire a semantics in which freedom from side effects is built into passive types, so that the side-effect freeness of the block expression is a constraint imposed by the types themselves rather than a property to be proven about valuations.

The main challenge is to define non-interference and passivity for entities such as commands, expressions and procedures, which are conventionally modelled as input-to-output *functions*. In [43,27], the similar problems that arise in treating the non-interference predicates in specification logic are addressed by using a category-theoretic form of *possible-world* semantics [39,31]. Each phrase type θ is interpreted as a *functor* $[\![\theta]\!]$ from a suitable (small) category of "possible worlds" to a category of domains, and any phrase P is interpreted as a *natural transformation* $[\![P]\!]$ of such functors. We will show that the *same* category of functors and natural transformations can be used to provide a satisfactory model of the SCIR-based programming language.

4.1 The Category of Worlds

A category of possible worlds appropriate to treating non-interference and passivity in Algol-like languages is defined as follows.

- The objects are sets (we require a small collection), thought of as sets of states. The set of all worlds is assumed to be closed under the following:
  - if V_τ is the set of values appropriate to a data type τ, V_τ is a world;
  - if X and Y are worlds, so is their set product $X \times Y$; and
  - if X is a world, so is any $Y \subset X$.
- A map from X to Y is a pair (f, Q), where Q is an equivalence relation on X and f is a function from X to Y whose restriction to each Q-equivalence class is an injection. Intuitively, X is a world "derived" from Y, f maps

states in X back into Y, and Q is an equivalence relation on states which must be preserved by execution in world X.

The composition of maps $(f, Q)\colon X \to Y$ and $(g, R)\colon Y \to Z$ is the map $(h, P)\colon X \to Z$ such that $h = f\,;g$ and $x\,P\,x'$ iff $x\,Q\,x'$ and $f(x)\,R\,f(x')$. The identity map id_X on world X is (I_X, T_X), where I_X is the identity function on set X and T_X is the everywhere-true binary relation on X. We will designate this category as $\mathbf{X}$; however, it is the *opposite* of the category of worlds used in [43,27].

Any one-element set is a *terminal* object in $\mathbf{X}$; the unique map from X to, say, $\{*\}$ is $(\lambda x.*, =_X)$. We can also define a tensor product as follows; for objects X and Y, $X \otimes Y = X \times Y$ (the usual cartesian product of sets), and $(f, Q) \otimes (g, R) = (f \times g, Q \times R)$, where $(f \times g)\langle x, y\rangle = \langle f(x), g(y)\rangle$ and $\langle x, y\rangle (Q \times R) \langle x', y'\rangle$ if and only if $x\,Q\,x'$ and $y\,R\,y'$. This is the basis for a symmetric monoidal structure on $\mathbf{X}$, with the designated terminal object as the unit; for example, the symmetry map from $X \otimes Y$ to $Y \otimes X$ consists of the exchange function and the total relation on $X \times Y$.

Projection maps $\pi_0\colon X \otimes Y \to X$ and $\pi_1\colon X \otimes Y \to Y$ can be defined to consist of: the usual projection functions on $X \times Y$, and equivalence relations that relate $\langle x, y\rangle$ pairs having the same y or x components, respectively. These maps are termed "expansions" in [43,27], where the opposite category is considered, and similar maps are treated in [31].

We can also define a natural family of *diagonal* maps $\delta_X\colon X \to X \otimes X$ whose components are: the diagonal function on X and the total relation on X. Note, however, that $\delta_X\,;\pi_i \neq \mathrm{id}_X$, and $\otimes$ is not a categorical product.

4.2 Semantic Category and Basic Functors

The semantic category for our model is the category $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ of contravariant functors from the category of possible worlds to $\mathbf{D}$, where $\mathbf{D}$ is the category of ω-cpos (i.e., possibly bottom-less ω-complete posets and continuous functions), with all natural transformations as the maps. This is essentially the same semantic category used in [43,27]. Finite products in $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ can be obtained pointwise from the familiar products in $\mathbf{D}$.

We now consider interpretations in $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ for the basic types (expressions and commands) in the programming language. First, we define the "domain-of-states" functor, St, to be the *covariant* functor from $\mathbf{X}$ to $\mathbf{D}$ such that $St(X) = X$, discretely-ordered, and $St(f, Q) = f$. Contravariant functors for *expression* types can then be defined pointwise as follows:

$$[\![\tau]\!]X \;=\; St(X) \to (V_\tau)_\perp \quad \text{and} \quad [\![\tau]\!]f\,e \;=\; St(f)\,;e$$

where V_τ is the set of values associated with τ; i.e., $V_{\mathbf{int}}$ is the set of integers and $V_{\mathbf{bool}}$ is the two-element set of truth values.

For the command type, if X is a world then $c \in [\![\mathbf{comm}]\!]X$ is a family of *partial* functions, indexed by all $\mathbf{X}$-maps with co-domain X, so that $c(f\colon Y \to X)$ is a partial function on $St(Y)$. The uniformity condition on the family is the following "semi-commutativity" requirement: for all $f\colon Y \to X$

and $g\colon Z \to Y$,

$$c(g\,;f)\,;St(g) \;\subseteq\; St(g)\,;c(f),$$

where the $\subseteq$ relation is graph inclusion of partial functions:

$$\begin{array}{ccc} Y & \xrightarrow{\;c(f)\;} & Y \\ {\scriptstyle St(g)}\uparrow & \supseteq & \uparrow{\scriptstyle St(g)} \\ Z & \xrightarrow[\;c(g\,;f)\;]{} & Z \end{array}$$

The semi-commutativity allows command meanings to become less-defined in more-restricted worlds; however, the family must *also* satisfy the following commutativity requirement arising from the equivalence-class component of **X**-maps. For any **X**-map $(f,Q)\colon Y \to X$ and $y \in St(Y)$, let

$$Y' \;=\; \{y' \in St(Y) \mid y\,Q\,y'\}$$

(i.e., the set of states Q-reachable from y); then

$$\begin{array}{ccc} Y & \xrightarrow{\;c(f,Q)\;} & Y \\ {\scriptstyle St(\lceil Y')}\uparrow & & \uparrow{\scriptstyle St(\lceil Y')} \\ Y' & \xrightarrow[\;c\left(Y'\,;(f,Q)\right)\;]{} & Y' \end{array}$$

must commute (and not just semi-commute), where $\lceil Y'\colon Y' \to Y$ is the **X**-map with components: the insertion function from Y' to Y, and the total relation on Y'. This requirement is imposed to ensure that, when $c(f,Q)$ has a defined result, it preserves the Q-equivalence class of its argument.

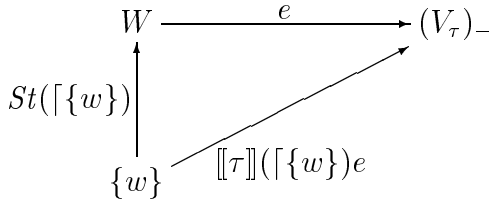
The morphism part of $[\![\mathbf{comm}]\!]$ is defined as follows: for any **X**-map $f\colon Y \to X$, command meaning $c \in [\![\mathbf{comm}]\!]X$, and **X**-map $g\colon Z \to Y$,

$$[\![\mathbf{comm}]\!]f\,c\,g \;=\; c(g\,;f)$$

This makes $[\![\mathbf{comm}]\!]$ a *contravariant* functor from **X** to **D**, as required.

We now discuss some examples to show how these functors interact with the **X**-maps defined in the preceding section.

Because of maps from *subsets* of state sets, expression meanings in the semantics cannot have side effects, not even "temporary" ones. For any world W and $w \in W$ we can restrict to the singleton set of states $\{w\}$ using the "restriction" map $\lceil\{w\}\colon\{w\} \to W$ whose components are: the insertion function and the total relation on $\{w\}$. Then, for any expression meaning $e \in [\![\tau]\!]W$, the value of e in state w is completely determined by the meaning $[\![\tau]\!](\lceil\{w\})e$ at world $\{w\}$:



where the vertical arrow is the insertion of $\{w\}$ into W. There can be no side effects during evaluation of $e(w)$ because, in world $\{w\}$, there are no other states to change to!

The behaviour of *commands* under restrictions is quite different. Consider the command meaning $c(\cdot) \in [\![\mathbf{comm}]\!](W \otimes Z)$ corresponding to an assignment statement $z := z + 1$, where z accesses the Z-valued component in $X \otimes Z$. The partial function for $c(\mathrm{id}_{W\otimes Z})$ maps $\langle w, n\rangle$ to maps $\langle w, n+1\rangle$. But we also need to define $c(f)$ for all other $\mathbf{X}$-maps f into $W \otimes Z$, including restriction maps. In particular, if we consider $c(\lceil\{\langle w, n\rangle\})$ then this component of c cannot produce an output state, because $\langle w, n+1\rangle$ is not an element of the world $\{\langle w, n\rangle\}$. More generally, $c(f)s$ can be defined only if $\langle w, n+1\rangle$ is in the range of $St(f)$. In contrast to the previous example, command meanings are not completely determined at singleton worlds, just because they may change the state.

Suppose now that we restrict to the world

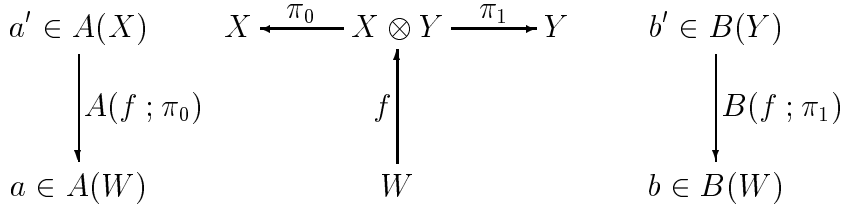
$$Y = \{\langle w, n\rangle \in W \otimes Z \mid n \text{ is even}\}$$

and consider the composite $z := z+1\,; z := z+1$, and its semantic counterpart $c\,;c$. Sequential composition is interpreted componentwise, so for command meanings c_1 and c_2, $(c_1\,;c_2)(f)$ is just the composition $c_1(f)\,;c_2(f)$ of the partial functions for the components. Thus, we get that $(c\,;c)(\mathrm{id}_{W\otimes Z})\langle w, n\rangle = \langle w, n\rangle$. However, $(c\,;c)(\lceil Y)\langle w, n\rangle$ is undefined, because $c(\lceil Y)\langle w, n\rangle$ is undefined. The attempt to "stray" out of Y, even at an intermediate state, leads to divergence.

4.3 Non-Interference

4.3.1 Tensor Product

Intuitively, meanings $a \in A(W)$ and $b \in B(W)$ are non-interfering if neither makes active use of any memory used by the other. We formalize this intuition as follows: $a \# b$ iff there exist worlds X and Y, an $\mathbf{X}$-map $f\colon W \to X \otimes Y$ and meanings $a' \in A(X)$, $b' \in B(Y)$ such that $A(f\,;\pi_0)a' = a$ and $B(f\,;\pi_1)b' = b$:



The idea is that a and b "come from" disjoint worlds X and Y, respectively. The archetypical example of this arises in the declaration of a new local variable: the new variable and non-local entities are non-interfering because they can be viewed as "coming from" the factors of a product world [27, Section 5].

The map f in the definition of $a \# b$ allows for sharing of passively-used memory, as in

$$\begin{array}{ccccc} X \otimes Z & \xleftarrow{\pi_0} & X \otimes Z \otimes Z \otimes Y & \xrightarrow{\pi_1} & Z \otimes Y \\ & & \uparrow{\scriptstyle \mathrm{id}_X \otimes \delta_Z \otimes \mathrm{id}_Y} & & \\ & & X \otimes Z \otimes Y & & \end{array}$$

The composite maps from $X \otimes Z \otimes Y$ to $X \otimes Z$ and $Z \otimes Y$ have the *equality* relation $=_Z$ as the equivalence-relation component on Z; this ensures that the shared memory Z can only be used passively. An example is discussed below.

We can now define a bifunctor $\otimes$ on $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ to interpret type assignments and the non-interfering product type constructor in the syntax. For any functors $A, B \colon \mathbf{X}^{\mathrm{op}} \to \mathbf{D}$ and world W,

$$(A \otimes B)(W) \;=\; \Big\{ \langle a, b\rangle \in (A \times B)(W) \;\Big|\; a \# b \Big\}$$

and the morphism part is defined as follows; for any $f \colon W \xrightarrow{\mathbf{X}} Y$,

$$(A \otimes B)(f)\langle a, b\rangle = \Big\langle A(f)a, B(f)b \Big\rangle \ .$$

If $\eta : A \dot{\to} A'$ and $\mu : B \dot{\to} B'$, then

$$(\eta \otimes \mu)(W)\langle a, b\rangle = \Big\langle \eta(W)a, \mu(W)b \Big\rangle \ .$$

To complete the monoidal structure on $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$, we define the unit to be a specified terminal object 1, which can be defined pointwise. These definitions make $(\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}, \otimes, 1)$ a symmetric monoidal category.

4.3.2 Sharing and Contraction

To illustrate the interaction between sharing and disjointness in the definition of $\otimes$, we consider a map

$$\| \colon [\![\mathbf{comm}]\!] \otimes [\![\mathbf{comm}]\!] \dot{\to} [\![\mathbf{comm}]\!]$$

for interpreting the deterministic parallel composition of non-interfering commands. Given $\langle c_1, c_2\rangle \in \Big([\![\mathbf{comm}]\!] \otimes [\![\mathbf{comm}]\!]\Big)(W)$, there exist c_1' and c_2' as follows:

$$\begin{array}{ccccc} c_1' \in [\![\mathbf{comm}]\!]X & & X \xleftarrow{\pi_0} X \otimes Y \xrightarrow{\pi_1} Y & & c_2' \in [\![\mathbf{comm}]\!]Y \\ \Big\downarrow{\scriptstyle [\![\mathbf{comm}]\!](f\,;\pi_0)} & & \Big\uparrow{\scriptstyle f} & & \Big\downarrow{\scriptstyle [\![\mathbf{comm}]\!](f\,;\pi_1)} \\ c_1 \in [\![\mathbf{comm}]\!]W & & W & & c_2 \in [\![\mathbf{comm}]\!]W \end{array}$$

Define $c_1' \otimes c_2' \in [\![\mathbf{comm}]\!](X \otimes Y)$ to be the component-wise product map; i.e., $(c_1' \otimes c_2')(g) = (c_1'\, g) \times (c_2'\, g)$, using the morphism part of the cartesian product $\times$ in the category of sets and partial functions. To get a meaning at world W we use map f, as follows:

$$\| \,(W)\langle c_1, c_2\rangle \;=\; [\![\mathbf{comm}]\!](f)(c_1' \otimes c_2') \ .$$

Here, X, Y, c_1' and c_2' are not uniquely determined, but the functoriality requirements on $[\![\mathbf{comm}]\!]$ are sufficient to ensure that this is a good definition.

The f map is what allows for a limited amount of sharing. To illustrate this, suppose $X = Y = Z \otimes Z$, $c_1'(\mathrm{id})\langle n_1, n_2\rangle = \langle n_2+1, n_2\rangle$ and $c_2'(\mathrm{id})\langle n_1, n_2\rangle = \langle n_1, n_1+3\rangle$. Then, we can form a composite command in which c_1' and c_2' operate on disjoint portions of the state:

$$c_1' \otimes c_2' \in [\![\mathbf{comm}]\!](Z \otimes Z \otimes Z \otimes Z) \ .$$

Sharing can be achieved via a diagonal map

$$\begin{array}{c} Z \otimes Z \otimes Z \otimes Z \\ \uparrow \ \mathrm{id}_Z \otimes \delta_Z \otimes \mathrm{id}_Z \\ Z \otimes Z \otimes Z \end{array}$$

yielding the meaning

$$c = [\![\mathbf{comm}]\!]\Big(\mathrm{id}_Z \otimes \delta_Z \otimes \mathrm{id}_Z\Big)(c_1' \otimes c_2') \ .$$

We find that $c(\mathrm{id})\langle n_1, n_2, n_3\rangle = \langle n_2+1, n_2, n_2+3\rangle$: the two middle components in the product $Z \otimes Z \otimes Z \otimes Z$ get identified, which is to say, shared, by the diagonal map. Intuitively,

- c_1' corresponds to a command $x := y + 1$
- c_2' corresponds to a command $z := y' + 3$, and
- c corresponds to the command $x := y + 1 \parallel z := y + 3$, obtained by parallel composition followed by Contraction of y and y',

where the identifiers correspond to evident components in Z^4 and Z^3.

Thus, the semantics of $\parallel$ is given by combining functions on disjoint state-sets, followed by sharing. This corresponds closely to how parallel commands are typed: first, commands with no identifiers in common are combined, and then sharing is introduced using the Contraction rule.

4.3.3 Exponential

An *exponential* construction right adjoint to $\otimes$ makes $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ a *closed* category; $(A \multimap B)(W)$ is defined to be the set (ordered pointwise) of families $q(X)\colon A(X) \to B(W \otimes X)$ of continuous functions indexed by worlds X, such that, for all $\mathbf{X}$-maps $f\colon Y \to X$, the following naturality diagram commutes:

$$\begin{array}{ccc} A(X) & \xrightarrow{q(X)} & B(W \otimes X) \\ {\scriptstyle A(f)}\downarrow & & \downarrow{\scriptstyle B(\mathrm{id}_W \otimes f)} \\ A(Y) & \xrightarrow{q(Y)} & B(W \otimes Y) \end{array}$$

$(A \multimap B)(W)$ is simply the (pointwise-ordered) hom-set $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}\Big(A, B(W \otimes -)\Big)$. Note that the argument of $q(X)$ is an element of $A(X)$; i.e., W is not involved, corresponding to the principle that a procedure and its argument are disjoint. The morphism part of $A \multimap B$ is defined as follows: for any $\mathbf{X}$-map $f\colon X \to W$, $(A \multimap B)(f)(q)(Y) = q(Y)\,;\, B(f \otimes \mathrm{id}_Y)$. If $\eta\colon A' \dot{\to} A$ and $\mu\colon B \dot{\to} B'$, then

$\eta \multimap \mu\colon (A \multimap B) \dashrightarrow (A' \multimap B')$ is given by

$$(\eta \multimap \mu)(W)(p \in (A \multimap B)W)(X) \;=\; \eta(X)\,;p(X)\,;\mu(W \otimes X)\ .$$

The application map $app(A,B)\colon (A \multimap B) \otimes A \dashrightarrow B$ is defined by

$$app(A,B)(W)\Big\langle q \in (A \multimap B)(W), a \in A(W)\Big\rangle \;=\; B(f)\Big(q'(Y)a'\Big)\ ,$$

where $f\colon W \to X \otimes Y$, $A(f\,;\pi_0)a' = a$, and $(A \multimap B)(f\,;\pi_1)q' = q$. Here, $f\colon W \to X \otimes Y$, $a' \in A(X)$ and $q' \in (A \multimap B)(Y)$ are not uniquely determined, but the naturality condition on procedure meanings is sufficient to ensure that this is a good definition. If $\eta\colon A \otimes B \dashrightarrow C$, the curried map $\eta^*\colon A \dashrightarrow (B \multimap C)$ is defined by

$$\eta^*(W)\Big(a' \in A(W)\Big)(X)\Big(b' \in B(X)\Big) \;=\; \eta(W \otimes X)\Big\langle A(\pi_0)a', B(\pi_1)b'\Big\rangle\ .$$

Proposition 4.1 $(\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}, \otimes, 1, \multimap)$ *is a symmetric monoidal closed category.*

Proof. The structure described is an instance of an abstract construction presented in [8]. □

4.4 Passivity

Intuitively, $a \in A(W)$ is passive if it doesn't interfere with anything. This can be defined rigorously using "state-change constraint" endomaps $\alpha_W\colon W \to W$ in $\mathbf{X}$ whose components are: the identity function on W and the equality relation on W. It is easily verified that the α_W are *idempotent* maps, and, furthermore, that they constitute a *natural* family of maps; i.e., α is a natural idempotent on the identity functor.

The importance of the α_W for treating passivity is that, because of the definition of $[\![\mathbf{comm}]\!]$, they preclude *any* state changes; hence $A(\alpha_W)$ applied to any $a \in A(W)$ "pacifies" it so that it cannot interfere with anything. For example, suppose that $c \in [\![\mathbf{comm}]\!]W$ is the denotation of $w := w + 1$; the second uniformity condition on command meanings ensures that $c(\alpha_W)s$ can be defined only if $c(\mathrm{id}_W)s = s$, and so, for this c, we obtain that $[\![\mathbf{comm}]\!](\alpha_W)c$ is everywhere-undefined.

The effect of state-change constraints on expression meanings is quite different. For each world W and $e \in [\![\tau]\!]W$, $[\![\tau]\!](\alpha_W)e = e$. State-change constraints have no effect here because expressions cannot cause side effects.

These examples suggest the following definition: $a \in A(W)$ is passive if and only if $A(\alpha_W)a = a$. For example, $[\![\mathbf{skip}]\!]W$ (a family of identity functions) and $[\![\mathbf{diverge}]\!]W$ (a family of everywhere undefined functions) are passive elements of $[\![\mathbf{comm}]\!]W$.

The following results establish the connections between passivity and noninterference.

Proposition 4.2 *If $p \in P(W)$ and $q \in Q(W)$ are passive, $p \# q$.*

Proof. If p and q are passive, $P(\alpha_W)p = p$ and $Q(\alpha_W)q = q$; but $\alpha_W = \delta_W\,;\pi_i$ for $i = 0, 1$, and so $p \# q$. □

Proposition 4.3 *$a \in A(W)$ is passive iff $a \# a$.*

Proof. The "only if" part follows from the preceding Proposition.

In the other direction, suppose that $a \# a$; then there exist worlds X and Y, $a_X \in A(X)$, $a_Y \in A(Y)$, and an $\mathbf{X}$-map $f: W \to X \otimes Y$ such that $A(f\,;\pi_0)a_X = a = A(f\,;\pi_1)a_Y$. Let Q_i for $i = 0, 1$ be the equivalence-relation components of $f\,;\pi_i$; then $(I_W, Q_i)\,;f\,;\pi_i = f\,;\pi_i$, and so we get, by functoriality of A, that $A(I_W, Q_i)a = a$ for $i = 0, 1$. This gives us that $A\Big((I_W, Q_0)\,;(I_W, Q_1)\Big)a = a$; but $(I_W, Q_0)\,;(I_W, Q_1) = \alpha_W$, and so a is passive. □

An *object* A of $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$ is passive iff, for every world W, every $a \in A(W)$ is passive. For example, a terminal object 1 is passive because it is a constant functor, and $[\![\tau]\!]$ is a passive object because, for any world W and $e \in [\![\tau]\!]W$,

$$\begin{aligned} [\![\tau]\!](\alpha_W)(e) &= St(\alpha_W)\,;e && \text{morphism part of } [\![\tau]\!] \\ &= e && St(\alpha_W) \text{ is the identity function} \end{aligned}$$

Let $\mathbf{P}$ be the full subcategory of passive objects of $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$. This determines a model of SCIR, which follows in fact as a special case of the abstract results of [10].

Theorem 4.4 [10] *Category* $\mathbf{D}^{\mathbf{X}^{\mathrm{op}}}$, *together with subcategory* $\mathbf{P}$, *comprise a bireflective model of SCIR.* □

The following data are thus obtained, allowing us to interpret the SCIR typing rules:

- the bireflector $S: \mathbf{D}^{\mathbf{X}^{\mathrm{op}}} \to \mathbf{P}$, which takes $A(X)$ to the sub-cpo of passive elements: $SAX = \{a \in A(X) \mid a \text{ is passive}\}$, and $SAfa = Afa$;
- the unit $\eta_A: A \to SA$ of $S \dashv J$, given by $\eta_A W a = A(\alpha_W)a$; and
- the counit $\varepsilon'_A: SA \to A$ of $J \dashv S$, given by the inclusion $SAW \hookrightarrow AW$.

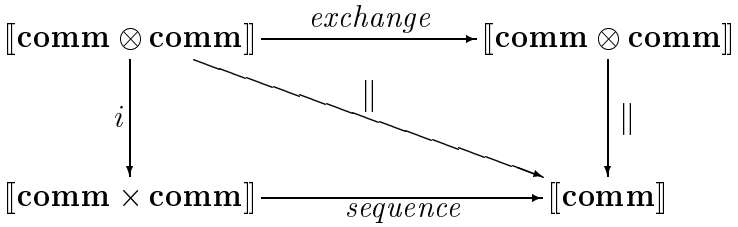
4.5 Interpretation of the Constants

We now present interpretations of selected constants. The interpretation of $\|$ has already been given in Section 4.3.2.

Sequential composition is given by a map

$$\mathit{sequence}: [\![\mathbf{comm} \times \mathbf{comm}]\!] \dot{\to} [\![\mathbf{comm}]\!] \ .$$

The definition is $\mathit{sequence}(W)\langle c_1, c_2\rangle(f) = c_1(f)\,;c_2(f)$, using composition of partial functions. One can show that the following diagram commutes



where $\|$ is the interpretation of parallel composition from Section 4.3.2, i is the evident inclusion, and *exchange* is the twist map exchanging the two components of $\otimes$. As a consequence, if commands C_1 and C_2 don't share any identifiers, we have the equivalences $C_1; C_2 \equiv C_2; C_1 \equiv C_1 \parallel C_2 \equiv C_2 \parallel C_1$, which wouold not hold in the absence of interference constraints.

For assignment, we define a map

$$\mathit{assign}\colon \Big((\llbracket\tau\rrbracket \multimap \llbracket\mathbf{comm}\rrbracket) \times \llbracket\tau\rrbracket\Big) \times \llbracket\tau\rrbracket \dashrightarrow \llbracket\mathbf{comm}\rrbracket\ .$$

Because of the presence of $\times$ instead of $\otimes$ on the left, we cannot simply use the *app* map to apply the procedure. To deal with this, we supply the "acceptor" component of a variable with a constant-function argument. Given $v \in V_\tau$, define $k_v \in \llbracket\tau\rrbracket 1$ to be the constant meaning such that $k_v(f)(w) = v$ for all $f\colon 1 \xrightarrow{\mathbf{X}} W$ and $w \in W$. We can then define the assignment map as follows.

$$\mathit{assign}(W)\left\langle\langle a, e\rangle, e'\right\rangle w \;=\; \begin{cases} i(a(1)(k_v)\langle w, *\rangle), \text{ if } e'(w) = v \neq - \\ \text{undefined}, \qquad\quad \text{if } e'(w) = - \end{cases}$$

where $i\colon W \otimes 1 \to W$ is the unity isomap.

The block-expression combinator $\mathbf{do}_\tau$ is treated by defining

$$do_\tau\colon S\Big(\llbracket\mathbf{var}[\tau]\rrbracket \multimap \llbracket\mathbf{comm}\rrbracket\Big) \dashrightarrow \llbracket\tau\rrbracket.$$

First, let $\langle a, e\rangle \in \big[\!\!\big[\mathbf{var}[\tau]\big]\!\!\big]V_\tau$ be the standard "local" variable meaning at world V_τ [27]. Then

$$do_\tau(W)\, p\, w \;=\; \begin{cases} v, \text{ if } p(V_\tau)\langle a, e\rangle\langle w, v_0\rangle = \langle w, v\rangle \\ -, \text{ if } p(V_\tau)\langle a, e\rangle\langle w, v_0\rangle \text{ is undefined} \end{cases}$$

where v_0 is a standard initial value for τ-typed variables. The passivity of p guarantees that $w' = w$ whenever $p(X)(e)\langle w, x\rangle = \langle w', x'\rangle$, so there is no need for a snap-back effect.

Finally, we show how the fixed-point combinator can be interpreted by defining a map

$$Y_\theta\colon S\Big(\llbracket\theta\rrbracket \multimap \llbracket\theta\rrbracket\Big) \dashrightarrow \llbracket A\rrbracket.$$

If $p \in S(\llbracket\theta\rrbracket \multimap \llbracket\theta\rrbracket)Z$ then we can obtain a function $p' : \llbracket\theta\rrbracket Z \to \llbracket\theta\rrbracket Z$ by composing $p[Z]\colon \llbracket\theta\rrbracket Z \to \llbracket\theta\rrbracket Z \otimes Z$ with the map $\llbracket\theta\rrbracket\delta_Z : \llbracket\theta\rrbracket Z \otimes Z \to \llbracket\theta\rrbracket Z$. $Y_\theta[Z]$ sends p to the least fixed-point of p'.

Other constants can be treated as in [31,43].

4.6 An Alternative Presentation

J. C. Reynolds has suggested (private communication) that an interference-controlled Algol-like language should be interpreted by families of continuous functions, indexed by assignments of state-sets to identifiers, with each identifier in the context interpreted by a meaning relative to its *own* state-set. In our framework, this would mean that a syntax judgement $\Pi \mid \Gamma \vdash P\colon \theta$ would be interpreted by a family of functions $\phi(\overline{W})$, indexed by assignments $\overline{W}$ of

worlds to identifiers, with the functionality of $\phi(\overline{W})$ being

$$\Big(\prod_{\iota\in\mathrm{dom}\,\Pi} S[\![\Pi\iota]\!](\overline{W}\iota)\Big)\times\Big(\prod_{\iota\in\mathrm{dom}\,\Gamma}[\![\Gamma\iota]\!](\overline{W}\iota)\Big)\;\longrightarrow\;[\![\theta]\!]\Big(\prod_{\iota\in\mathrm{dom}\,(\Pi,\Gamma)}\overline{W}\iota\Big)$$

Note that the products in the domain of $\phi(\overline{W})$ are *cpo* products, whereas the product in the co-domain is a *set* product. This form of semantic interpretation seems intuitively appealing because it makes the disjointness of distinct identifiers very explicit; but it is highly non-standard.

In this section, we show that we can define a bijection between the standard form of semantics discussed in earlier sections and this non-standard form. To simplify the treatment, we will consider natural transformations

$$\eta\colon A\otimes B\dot{\to}C$$

and families of functions

$$\phi(X,Y)\colon A(X)\times B(Y)\to C(X\otimes Y)$$

natural in $\mathbf{X}$-objects X and Y. From a natural transformation η, we can define a family $\phi(X,Y)$ of functions as follows:

$$\phi(X,Y)(a',b')\;=\;\eta(X\otimes Y)\big\langle A(\pi_0)a',B(\pi_1)b'\big\rangle$$

In the other direction,

$$\eta(W)\langle a,b\rangle\;=\;C(f)\big(\phi(X,Y)(a',b')\big)$$

where $f\colon W\xrightarrow{\mathbf{X}}X\otimes Y$, $a'\in A(X)$ and $b'\in B(Y)$ such that $A(f\,;\pi_0)a'=a$ and $B(f\,;\pi_1)b'=b$ must exist because $\langle a,b\rangle\in(A\otimes B)(W)$; the naturality requirement for $\phi(X,Y)$ ensures that $\eta(W)\langle a,b\rangle$ is uniquely determined. It is a routine exercise to verify that the mappings $\eta\mapsto\phi$ and $\phi\mapsto\eta$ just given are mutual inverses.

Uday Reddy has launched a criticism at semantics based on global states [36], and developed an alternate approach in which different identifiers denote independent "objects," where the state is implicitly represented in "histories of observations." We would claim that functor-category models, though they are not stateless, also represent a move away from the viewpoint of a common "global store" that programs act upon. For example, in the presentation sketched in this section, and implicitly in the standard presentation, each identifier is associated with its own state set, separate from the state-sets associated with other identifiers; intuitively, each identifier denotes an object acting upon a piece of local state.

5 Concluding Remarks

Syntactic control of interference is an important step toward the ideal of a "clean" form of imperative programming. It retains basic principles of Algol-like and functional programming, including equational laws such as the β law; this it has in common with recent work emanating from the functional-programming community (see, e.g., [32,22,19]). But interference control also begins to address some of the problems of state, such as aliasing. Functional

principles alone do not make state significantly easier to reason about, as is abundantly clear, for example, from specification logic. Controlling interference addresses some of the most prominent difficulties.

At present, syntactic control of interference has developed to the point where it possesses quite satisfactory type systems and models. Nevertheless, there are many issues that need to be addressed before the ideal of a clean *and* practical form of imperative programming can be realized. The following is a partial list of immediately relevant issues.

(i) Our example programming language does not have facilities for programming dynamically-reconfigurable data structures of the kind often implemented using pointers or references. Simple languages of this form can serve as a useful testbed for ideas on integrating imperative and functional programming, but extending the basic approach of SCI to support coding of dynamic data is clearly crucial. It is not obvious what the best way to do this might be.

(ii) A call-by-value version of SCI could have some interest. A challenge for such a design is to maintain a controlled form of side effects.

(iii) One motivation for interference control is that it should simplify reasoning about programs. To find evidence for this position, one might investigate a version of specification logic stripped of the pervasive # assumptions. A more ambitious program would be to set down axioms characterizing independence of identifiers, possibly using the parametricity ideas of [28], and to investigate the thesis that such a characterization simplifies the logical form of specifications needed for familiar objects or procedures.

(iv) The complexity of type checking and the possibility of type inference need to be investigated for the type system presented here.

(v) The semantic model presented here possesses two kinds of exponentials, one for the monoidal closed structure, and another, adjoint to $\times$, for cartesian closed structure. This raises the question of whether interference control and uncontrolled Algol can coexist harmoniously in one system, which might be useful in addressing difficulties with jumps and recursive definitions having active free identifiers. Various "unified logics" [12,2] have similar aims, combining intuitionistic, linear, and classical logics; we would want to combine intuitionistic and affine systems. An interesting point to note is that here the two kinds of closed structure coexist in the *same* category, so there is no need to pass to a separate category, such as a Kleisli category, to interpret the intuitionistic (i.e., Algol's) function types.

(vi) The hope for a "linear logic-based functional language" that can express state manipulation remains unrealized, or certainly not adequately realized; but the similarities with interference control, both in aims and in technical details, are alluring. Rather than taking functional programming as the starting point, a reasonable approach might be to modify syntactic control of interference so that it provides a range of types for expressing manipulation of state, instead of a single type **comm**.

References

[1] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. A term calculus for intuitionistic linear logic. In M. Bezen and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90, Utrecht, The Netherlands, March 1993. Springer-Verlag, Berlin.

[2] P. N. Benton. A mixed linear and non-linear logic: proofs, terms and models (preliminary report). Technical Report 352, University of Cambridge Computer Laboratory, October 1994.

[3] G.M. Bierman. What is a categorical model of intuitionistic linear logicΓ In *Proceedings of Second International Conference on Typed λ-calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 78–93, Edinburgh, Scotland, April 1995. Springer-Verlag, Berlin.

[4] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice-Hall International, London, 1988.

[5] P. Brinch Hansen. Structured multiprogramming. *Comm. ACM*, 15(7):574–78, 1972.

[6] F. Brown, editor. *The Frame Problem in Artificial Intelligence. Proceedings of the 1987 workshop.* Morgan Kaufmann, 1987.

[7] M. Coppo and M. Dezani. A new type assignment for λ-terms. *Archiv. Math. Logik*, 19:139–156, 1978.

[8] B. J. Day. On closed categories of functors. In S. Mac Lane, editor, *Reports of the Midwest Category Seminar*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38. Springer-Verlag, Berlin-New York, 1970.

[9] S. Eilenberg and G. M. Kelly. Closed categories. In S. Eilenberg et al., editors, *Proceedings of the Conference on Categorical Algebra*, pages 421–562, La Jolla, California, 1965. Springer-Verlag, New York, 1966.

[10] P. Freyd, P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Bireflectivity. In these proceedings.

[11] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, pages 1–102, 1987.

[12] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[13] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580 and 583, 1969.

[14] C. A. R. Hoare. Hints on programming-language design. In C. Bunyan, editor, *Computer Systems Reliability*, volume 20 of *State of the Art Report*, pages 505–34. Pergamon/Infotech, 1974. Also pages 193-216 of [15].

[15] C. A. R. Hoare and C. B. Jones, editors. *Essays in Computing Science.* Prentice Hall International, 1989.

[16] P. Hudak. Conception, evolution, and application of functional programming languages. *Computing Surveys*, 31:359–411, 1989.

[17] J. Hughes. Why functional programming matters. *The Computer Journal*, 32:98–107, 1989.

[18] G. M. Kelly. Doctrinal adjunctions. In G. M. Kelly, editor, *Category Seminar: Proceedings, Sydney Category Theory Seminar, 1972/73*, volume 420 of *Lecture Notes in Mathematics*, pages 257–280. Springer-Verlag, New York, 1974.

[19] J. Launchbury and S. Peyton Jones. State in Haskell. *Lisp and Symbolic Computation*, 8(4):293–341, December 1995.

[20] S. Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.

[21] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: preliminary report. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 191–203, San Diego, California, 1988. ACM, New York.

[22] M. Odersky, D. Rabin, and P. Hudak. Call by name, assignment, and the lambda calculus. In POPL [33], pages 43–56.

[23] P. W. O'Hearn. *The Semantics of Non-Interference: A Natural Approach.* Ph.D. thesis, Queen's University, Kingston, Canada, 1990.

[24] P. W. O'Hearn. Linear logic and interference control. In D. H. Pitt et al., editors, *Category Theory and Computer Science*, volume 530 of *Lecture Notes in Computer Science*, pages 74–93, Paris, France, September 1991. Springer-Verlag, Berlin.

[25] P. W. O'Hearn. A model for syntactic control of interference. *Mathematical Structures in Computer Science*, 3(4):435–465, 1993.

[26] P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Mathematical Foundations of Programming Semantics, Eleventh Annual Conference*, volume 1 of *Electronic Notes in Theoretical Computer Science*, Tulane University, New Orleans, Louisiana, March 29–April 1 1995. Elsevier Science. Also in [29], pages 189–226.

[27] P. W. O'Hearn and R. D. Tennent. Semantical analysis of specification logic, 2. *Information and Computation*, 107(1):25–57, 1993. Also in [29].

[28] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, May 1995. Also in [29], pages 109–164.

[29] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*, volume 2. Birkhauser, Boston, 1997.

[30] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*, volume 1. Birkhauser, Boston, 1997.

[31] F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* Ph.D. thesis, Syracuse University, Syracuse, N.Y., 1982.

[32] S. Peyton-Jones and P. Wadler. Imperative functional programming. In POPL [33], pages 71–84.

[33] *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Charleston, South Carolina, 1993. ACM, New York.

[34] A. J. Power. Why tricategoriesΓ To appear in *Information and Computation*.

[35] U. S. Reddy. Passivity and independence. In *Proceedings, 9th Annual IEEE Symposium on Logic in Computer Science*, pages 342–352, Paris, 1994. IEEE Computer Society Press, Los Alamitos, California.

[36] U. S. Reddy. Global states considered unnecessary: introduction to object-based semantics. *Lisp and Symbolic Computation*, February 1996. Special issue on State in Programming Languages. Also in [29], pages 227–296.

[37] J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, January 1978. ACM, New York. Also in [30], pages 273–286.

[38] J. C. Reynolds. *The Craft of Programming*. Prentice-Hall International, London, 1981.

[39] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372, Amsterdam, October 1981. North-Holland, Amsterdam. Also in [30], pages 67-88.

[40] J. C. Reynolds. Idealized Algol and its specification logic. In D. Néel, editor, *Tools and Notions for Program Construction*, pages 121–161, Nice, France, December 1981. Cambridge University Press, Cambridge, 1982. Also in [30], pages 125–156.

[41] J. C. Reynolds. Syntactic control of interference, part 2. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium*, volume 372 of *Lecture Notes in Computer Science*, pages 704–722, Stresa, Italy, July 1989. Springer-Verlag, Berlin.

[42] R. D. Tennent. Semantics of interference control. *Theoretical Computer Science*, 27:297–310, 1983.

[43] R. D. Tennent. Semantical analysis of specification logic. *Information and Computation*, 85(2):135–162, 1990. Also in [29].

[44] P. Wadler. A syntax for linear logic. In S. Brookes et al., editors, *Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 513–529, New Orleans, 1993. Springer-Verlag, Berlin.

[45] N. Wirth. On the design of programming languages. In J. L. Rosenfeld, editor, *Proceedings* IFIP *Congress 74*, pages 386–393, Stockholm, 1974. North-Holland, Amsterdam.