

Computation-by-Interaction with Effects

Ulrich Schöpp

Ludwig-Maximilians-Universität München, Munich, Germany
Ulrich.Schoepp@ifi.lmu.de

Abstract. A successful approach in the semantics of programming languages is to model programs by interaction dialogues. While dialogues are most often considered abstract mathematical objects, it has also been argued that they are useful for actual computation. A manual implementation of interaction dialogues can be complicated, however. To address this issue, we consider a general method for extending a given language with a metalanguage that supports the implementation of dialogues. This method is based on the construction by Dal Lago and the author of the programming language INTML, which applies interaction dialogues to sublinear space computation. We show that only few assumptions on the programming languages are needed to implement a useful INTML-like metalanguage. We identify a weak variant of the Enriched Effect Calculus (EEC) of Egger, Møgelberg & Simpson as a convenient setting for capturing the structure needed for the construction of the metalanguage. In particular, function types are not needed for the construction and iteration by means of a Conway operator is sufficient. By using EEC we show how computational effects can be accounted for in the implementation of interaction dialogues.

In game semantics and related areas of programming language semantics there is a long tradition of modelling programs by interaction dialogues. Programs are modelled as entities that may engage in a dialogue with their environment. The interpretation of a program explains what kinds of queries it can receive and how it may answer. Large programs are composed of smaller ones that interact with each other, so that the whole execution of a program may be considered an interaction process. The question/answer dialogues that make up such models tend to have very concrete nature, which has led to interesting applications, for example in algorithmic game semantics.

The premise of this paper is that interaction dialogues are useful not only for interpreting programming languages, but also as an actual implementation method. There are many examples where dialogues have been used for the implementation of programs, e.g. [17, 6, 13]. Two recent examples provide the main motivation for the work reported here. First, Ghica introduces the Geometry of Synthesis [6] as a method of hardware synthesis. His approach is to construct a game model by implementing interaction dialogues by digital circuits and then to interpret a variant of Idealized Algol in the thus constructed game model. With this approach one can write a program in a high-level language (Idealized Algol) and by interpretation in the game model have it translated to a low-level language for digital circuits (Verilog). In this way, the implementation of dialogues is used as a method for hardware synthesis.

A similar example has been studied by Dal Lago and the author in the context of computation with sublinear space [13]. There the problem is how to write programs

that operate on data too large to fit into memory. To access values that do not fit into memory one needs to query them piece by piece. Such computation can naturally be organised into question and answer dialogues. This observation has led to the design of the programming language INTML for writing programs that can access and manipulate large data [13]. The approach is to start from a simple low-level language, whose programs can be evaluated using limited space, and use it to implement dialogue-based computation. The implementation of dialogues is considered as the construction of a game model, which is then used to interpret the language INTML. The result is that one can use the higher-order functional language INTML to implement dialogues in a simple low-level language that allows easy analysis of space usage.

These examples can be seen as game semantics turned around. Rather than interpreting a given programming language in a game model, one implements a game model in a given language. In the thus constructed model one can then interpret a new programming language. As the game model has been implemented in the original language, this new language may then be seen as a metalanguage for programming dialogues in the original language. The value of this approach is that even weak low-level languages suffice to construct rich game models that can interpret sophisticated metalanguages.

Having argued that it is useful to implement computation by question and answer dialogues, we turn to the problem of implementing dialogues in a given programming language. Motivated by the examples above, we focus in particular on weak languages that allow circuit synthesis or simple resource analysis or the like. From game semantics it is known that the concrete details of interaction dialogues can be complicated, so that in theoretical work it is standard practice to identify useful structure abstractly, e.g. products or function spaces, and to work with this abstract structure. Here we consider how a similar abstraction can be attained for the implementation of interaction dialogues.

We do this by reconsidering the ideas of INTML [13] in a more general context. The approach is to extend a given programming language with constructs that support the implementation of dialogues. These extensions are definitional in the sense that any program written with them could have been written without them, only perhaps in a more complicated way. The approach is thus to extend a given language with a language for metaprogramming. Previous evidence suggests that INTML captures useful language constructs for the implementation of interaction dialogues [14].

We show that very little structure is needed to carry out the construction of an INTML-like metalanguage and that computational effects can be allowed without affecting the metalanguage. Computational effects are interesting in this context, as can be seen from the examples above. Ghica's Geometry of Synthesis uses stateful circuits, while for sublinear space programming it is interesting to consider nondeterminism, perhaps in an effort to characterise the complexity class $NLOGSPACE$ by a programming language. Beyond those examples, recent work on quantum λ -calculus [9] is based on computation by interaction with a quantum effect. Other effects, such as name generation, may be useful in the context of nominal game semantics [20]. There also seems to be a relation to Levy's Jump-with-Argument [16], which we intend to study in future work.

To give a rough idea of the metalanguage, assume given some programming language in which we want to implement the dialogues of a game model, e.g. PCF. The metalanguage extends this language with a new class of types for interactive

programming by dialogues. The new interaction types are formed by the grammar $X, Y ::= [A] \mid 1 \mid X \otimes Y \mid B \cdot X \multimap Y$, in which A and B range over types from the original language. Such an interaction type specifies an interface that tells which kinds of questions one may ask of programs of this type and which kinds of answer one may receive. The type $[A]$ specifies that one may be asked the single question ‘please compute a value’ and that one may reply with any value of type A . The type $X \otimes Y$ contains pairs of values; it combines the interfaces of X and Y so that one may interact with either component of the pair as if one had two values of type X and Y side by side. The type $B \cdot X \multimap Y$ contains functions from X to Y . These functions are evaluated in an interactive manner, i.e. information about the function argument is obtained by sending questions according to its interface.

The type $B \cdot X \multimap Y$ imposes a linearity restriction on the use of its argument: it may be used B -many times, which means that there is one copy for each value of type B . The values of type B thus serve as addresses for the copies of X . The linearity constraint allows us to construct the metalanguage even in weak low-level languages that can only represent a limited number of addresses, e.g. languages with only finite types, as one would use for circuits. In strong languages like PCF, full copying can be allowed.

Some words are in order about why we study language extensions for metaprogramming as opposed to deriving completely new languages from the game models. While computation by dialogue is a useful mode of computation, it seems that not necessarily all computations should be done in this way. With a metalanguage one has the option of mixing computation by interaction with the usual computation of a given language. For example, it should be useful to have two function types, one that is evaluated using dialogues as in game semantics and the other one using standard call by value, say.

1 Weak Effect Calculus

Before describing the metalanguage in the next section, we define a weak effect calculus to capture the assumptions we make on the base programming language. An effect calculus suggests itself, as for the construction of the metalanguage we need possibly non-terminating loops and so must account for the effect of non-termination at least.

We define the Weak Effect Calculus (WEC), a weak variant of the Enriched Effect Calculus (EEC), which was introduced by Egger, Møgelberg & Simpson [4] as a type theory for studying computational effects. The Enriched Effect Calculus develops Moggi’s computational metalanguage [19] and can also be understood as a reformulation and extension of Levy’s Call-by-Push-Value (CBPV) [16]. The choice of (a variant of) EEC as a basis for the construction of the metalanguage is motivated mainly by its clean separation of values and computations as well as the presence of copower types, which are particularly useful.

The Weak Effect Calculus (WEC) is obtained by taking the fragment of EEC without function types and products of computation types and adding sum types for values and a Conway operator for iteration:

$$\begin{array}{ll} \text{Value types} & A, B ::= \alpha \mid 0 \mid A + B \mid 1 \mid A \times B \mid \underline{A} \\ \text{Computation types} & \underline{A}, \underline{B} ::= \underline{\alpha} \mid \underline{0} \mid \underline{A} \oplus \underline{B} \mid A \cdot \underline{B} \mid !A \end{array}$$

It may be useful to think of value types simply as sets and computation types as sets with an additional element \perp intended to represent non-termination.

For value types we choose the usual sum and product types. For computation types we take sums ($\underline{0}$ and $\underline{A} \oplus \underline{B}$), copowers $A \cdot \underline{B}$ and computations $!A$. The type $A \cdot \underline{B}$ can be thought of as a type of pairs of a value of type A and a computation of type \underline{B} . The computation type $!A$ consists of computations that when executed may return a value of type A . It plays the role of TA in Moggi's computational λ -calculus. Further types could be added without affecting the results in this paper.

Like the Enriched Effect Calculus, WEC has two kinds of judgements, $\Gamma \mid - \vdash f : A$ and $\Gamma \mid x : \underline{B} \vdash g : \underline{C}$. Both contain a context Γ that assigns *value types* to variables. In addition there is a *stoup* that may either be empty or that may consist of a single variable declaration of computation type. The first judgement declares f to be a value of value type A . The second judgement declares g to be a computation. The term g therein may be thought of as an evaluation context whose hole is identified by x . An operational intuition is that the evaluation of g starts with the complete evaluation of x and then continues to evaluate g . In the above-mentioned interpretation of computation types as sets with an element \perp for non-termination, g appears as a strict function.

The terms and typing rules for WEC are given in Fig. 1. Therein, Δ ranges over stoups and may be either empty – or a variable declaration $x : \underline{B}$. The rules are subject to the condition that only judgements of one of the two forms above can be derived. For example, in the elimination rule for $\underline{0}$ the stoup Δ can only be empty if A is a computation type.

In addition to the usual terms for the various types from EEC, WEC also contains a term ($\text{let } x = f \text{ loop } g$) for iteration. The operational intuition is that first f is evaluated, its result is bound to x and then g is evaluated. If the result is $\text{inl}(h)$, then h is the result of ($\text{let } x = f \text{ loop } g$). If the result is $\text{inr}(f')$, then the computation continues as ($\text{let } x = f' \text{ loop } g$). In this way, the term ($\text{let } x = f \text{ loop } g$) represents a looping computation that may, in particular, fail to terminate.

This form of iteration will be enough to support the construction of a metalanguage in the next section. For example for the space usage analysis results in [13] it is essential that such a form of iteration suffices and full recursion is not needed.

In WEC the meaning of terms is explained by an equational theory. Except for the term ($\text{let } x = f \text{ loop } g$) the equations of WEC are just as for EEC, see [4, 5]. The equations for the term ($\text{let } x = f \text{ loop } g$) are those of a uniform Conway operator [21]. A Conway operator is a mapping $(-)^{\dagger}$ that takes a morphism $\underline{A} \rightarrow \underline{B} \oplus \underline{A}$ in some category, where \oplus denotes the coproduct, to a morphism $\underline{A} \rightarrow \underline{B}$ in the same category, subject to a number of equations¹. The term ($\text{let } x = f \text{ loop } g$) is syntax for such a Conway operator, for given a term $\Gamma \mid x : \underline{A} \vdash g : \underline{B} \oplus \underline{A}$ one can form $\Gamma \mid y : \underline{A} \vdash \text{let } x = y \text{ loop } g : \underline{B}$.

There are six equations for such a Conway operator: the fixpoint property, naturality, dinaturality, diagonal and uniformity. The fixpoint property is expressed by:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : \underline{B} \oplus \underline{A}}{\Gamma \mid \Delta \vdash (\text{let } x = f \text{ loop } g) = \text{case } g[f/x] \text{ of } \begin{cases} \text{inl}(x) \Rightarrow x \\ \text{inr}(y) \Rightarrow \text{let } x = y \text{ loop } g \end{cases} : \underline{A}}$$

¹ In loc. cit. Conway operators studied in the dual setting, taking $\underline{B} \times \underline{A} \rightarrow \underline{A}$ to $\underline{B} \rightarrow \underline{A}$.

$$\begin{array}{c}
 \frac{}{\Gamma, x: A \mid - \vdash x: A} \quad \frac{}{\Gamma \mid x: \underline{A} \vdash x: \underline{A}} \quad \frac{}{\Gamma \mid - \vdash *: 1} \\
 \frac{\Gamma \mid - \vdash f: A \quad \Gamma \mid - \vdash g: B}{\Gamma \mid - \vdash \langle f, g \rangle: A \times B} \quad \frac{\Gamma \mid - \vdash f: A \times B}{\Gamma \mid - \vdash \text{fst}(f): A} \quad \frac{\Gamma \mid - \vdash f: A \times B}{\Gamma \mid - \vdash \text{snd}(f): B} \\
 \frac{\Gamma \mid - \vdash f: 0}{\Gamma \mid \Delta \vdash \text{image}(f): A} \quad \frac{\Gamma \mid - \vdash f: A}{\Gamma \mid - \vdash \text{inl}(f): A + B} \quad \frac{\Gamma \mid - \vdash f: B}{\Gamma \mid - \vdash \text{inr}(f): A + B} \\
 \frac{\Gamma \mid - \vdash f: A + B \quad \Gamma, x: A \mid \Delta \vdash g: C \quad \Gamma, y: B \mid \Delta \vdash h: C}{\Gamma \mid \Delta \vdash \text{case } f \text{ of } \text{inl}(x) \Rightarrow g \mid \text{inr}(y) \Rightarrow h: C} \\
 \frac{\Gamma \mid \Delta \vdash f: \underline{0}}{\Gamma \mid \Delta \vdash \text{image}(f): \underline{A}} \quad \frac{\Gamma \mid \Delta \vdash f: \underline{A}}{\Gamma \mid \Delta \vdash \text{inl}(f): \underline{A} \oplus \underline{B}} \quad \frac{\Gamma \mid \Delta \vdash f: \underline{B}}{\Gamma \mid \Delta \vdash \text{inr}(f): \underline{A} \oplus \underline{B}} \\
 \frac{\Gamma \mid \Delta \vdash f: \underline{A} \oplus \underline{B} \quad \Gamma \mid x: \underline{A} \vdash g: \underline{C} \quad \Gamma \mid y: \underline{B} \vdash h: \underline{C}}{\Gamma \mid \Delta \vdash \text{case } f \text{ of } \text{inl}(x) \Rightarrow g \mid \text{inr}(y) \Rightarrow h: \underline{C}} \\
 \frac{\Gamma \mid - \vdash f: A \quad \Gamma \mid \Delta \vdash g: \underline{B}}{\Gamma \mid \Delta \vdash f \cdot g: A \cdot \underline{B}} \quad \frac{\Gamma \mid \Delta \vdash f: A \cdot \underline{B} \quad \Gamma, x: A \mid y: \underline{B} \vdash g: \underline{C}}{\Gamma \mid \Delta \vdash \text{let } x \cdot y = f \text{ in } g: \underline{C}} \\
 \frac{\Gamma \mid - \vdash f: A}{\Gamma \mid - \vdash !f: !A} \quad \frac{\Gamma \mid \Delta \vdash f: !A \quad \Gamma, x: A \mid - \vdash g: \underline{B}}{\Gamma \mid \Delta \vdash \text{let } !x = f \text{ in } g: \underline{B}} \\
 \frac{\Gamma \mid \Delta \vdash f: \underline{A} \quad \Gamma \mid x: \underline{A} \vdash g: \underline{B} \oplus \underline{A}}{\Gamma \mid \Delta \vdash \text{let } x = f \text{ loop } g: \underline{B}}
 \end{array}$$

Fig. 1. Typing Rules of the Weak Effect Calculus

The other four equations are just syntactic formulations of the corresponding equations in [21]. We omit them, as they are needed only to construct a uniform trace [7]. We could have added a uniform trace to WEC directly, but Conway operators appear more natural in the syntax, for example for giving an operational semantics.

1.1 Implementing Interaction

Suppose now we want to implement in WEC a way of computation by interaction, where the interface of an entity is given by a pair $(\underline{X}^-, \underline{X}^+)$ of computation types. Think of \underline{X}^- as the type of questions that may be asked of the entity and \underline{X}^+ as the type of possible answers.

A term of type $\Gamma \mid z: \underline{Y}^- \oplus \underline{X}^+ \vdash f: \underline{Y}^+ \oplus \underline{X}^-$ then implements a strategy of answering questions for an entity with interface $Y = (\underline{Y}^-, \underline{Y}^+)$ when given the ability to ask questions of an entity with interface $X = (\underline{X}^-, \underline{X}^+)$. For, suppose we have a term $\Gamma \mid x: \underline{X}^- \vdash e: \underline{X}^+$ that answers questions for X . Then we can define a term $\Gamma \mid x: \underline{Y}^- \vdash g: \underline{Y}^+$ that answers questions for Y . Concretely, we can define g to be the term $\text{let } z = \text{inl}(y) \text{ loop case } f \text{ of } \text{inl}(y) \Rightarrow \text{inl}(y) \mid \text{inr}(x) \Rightarrow \text{inr}(\text{inl}(e))$. Of course, this term is not very easy to read, nor are such terms easy to write. The metalanguage in the next section provides language constructs for writing such programs.

2 A Metalanguage for Interactive Computation

We now introduce $\text{INTML}[\text{WEC}]$, a metalanguage for implementing interactive computation in WEC . Formally we do this by introducing a new class of interaction types as well as a new typing judgement for interactive computations. We will then show that these extensions are in fact definitional, i.e. can be implemented in the original calculus. In this way, the new constructs can be seen as constructs for metaprogramming WEC .

As outlined in the Introduction, we add four kinds of interaction types:

$$\textbf{Interaction type } X, Y ::= [A] \mid 1 \mid X \otimes Y \mid A \cdot X \multimap Y$$

Each interaction type X represents a pair $(\underline{X}^-, \underline{X}^+)$ of computation types that specifies the interface the value of type X :

$$\begin{aligned} 1^- &= \underline{0} & [A]^- &= !1 & (X \otimes Y)^- &= \underline{X}^- \oplus \underline{Y}^- & (A \cdot X \multimap Y)^+ &= A \cdot \underline{X}^- \oplus \underline{Y}^+ \\ 1^+ &= \underline{0} & [A]^+ &= !A & (X \otimes Y)^+ &= \underline{X}^+ \oplus \underline{Y}^+ & (A \cdot X \multimap Y)^- &= A \cdot \underline{X}^+ \oplus \underline{Y}^- \end{aligned}$$

Interaction sequents have the form $\Gamma \mid x_1 : A_1 \cdot X_1, \dots, x_n : A_n \cdot X_n \vdash^i t : Y$. The context Γ maps variables to value types, as before. In the second part of the context, each variable x_i appears with multiplicity A_i , which is a value type. This means that x_i represents A_i -many copies of X_i , one for each value of A_i . The term t explains how to answer questions for Y given the ability to ask questions of the various copies of X_i .

The typing rules are given in Figs. 2, 3 and 4. In the rules we write $A \cdot \Phi$ for the context obtained by replacing each declaration $x : B \cdot X$ in Φ with $x : (A \times B) \cdot X$. In rule (STRUCT) we use a relation $A \leq B$ that informally expresses that B -many copies of X are more than A -many copies. Since A and B are value types, we formalise this by requiring there to exist a section-retraction pair between A and B .

$$\begin{array}{c} \text{VAR} \frac{}{\Gamma \mid x : 1 \cdot X \vdash^i x : X} \quad \text{STRUCT} \frac{\Gamma \mid \Phi, x : A \cdot X \vdash^i t : Y}{\Gamma \mid \Phi, x : B \cdot X \vdash^i t : Y} \quad A \leq B \\ \text{WEAK} \frac{\Gamma \mid \Phi \vdash^i t : Y}{\Gamma \mid \Phi, x : 1 \cdot X \vdash^i t : Y} \quad \text{EXCH} \frac{\Gamma \mid \Phi, x : A \cdot X, y : B \cdot Y, \Psi \vdash^i t : Z}{\Gamma \mid \Phi, y : B \cdot Y, x : A \cdot X, \Psi \vdash^i t : Z} \\ \text{COPY} \frac{\Gamma \mid \Phi \vdash^i s : X \quad \Gamma \mid \Psi, x : A \cdot X, y : B \cdot X \vdash^i t : Z}{\Gamma \mid \Psi, (A + B) \cdot \Phi \vdash^i \text{copy } s \text{ as } x, y \text{ in } t : Z} \end{array}$$

Fig. 2. Typing Rules of $\text{INTML}[\text{WEC}]$: Structural Rules

The terms of $\text{INTML}[\text{WEC}]$ represent strategies for computation by interaction. For most part the typing rules define a simply-typed λ -calculus with restricted copying. This λ -calculus interacts with the base language WEC by the rules ([I]) and ([E]). The operational intuition for the latter is: when asked to compute an answer in $[B]$, first ask s for its value. Upon receipt of an answer, which must be a value of type A , bind the result to x and query t . The answer, a value of type B , is then passed on to answer the initial request.

$$\begin{array}{c}
 \begin{array}{c}
 \text{[]I} \frac{\Gamma \mid - \vdash f : !A}{\Gamma \mid - \vdash^i [f] : [A]} \quad \text{[]E} \frac{\Gamma \mid \Phi \vdash^i s : [A] \quad \Gamma, x : A \mid \Psi \vdash^i t : [B]}{\Gamma \mid \Phi, A \cdot \Psi \vdash^i \text{let } [x] = s \text{ in } t : [B]} \quad 1 \leq A \\
 \\
 \otimes \text{I} \frac{\Gamma \mid \Phi \vdash^i s : X \quad \Gamma \mid \Psi \vdash^i t : Y}{\Gamma \mid \Phi, \Psi \vdash^i \langle s, t \rangle : X \otimes Y} \quad \otimes \text{E} \frac{\Gamma \mid \Phi \vdash^i s : X \otimes Y \quad \Gamma \mid \Psi, x : A \cdot X, y : A \cdot Y \vdash^i t : Z}{\Gamma \mid \Psi, A \cdot \Phi \vdash^i \text{let } \langle x, y \rangle = s \text{ in } t : Z} \\
 \\
 \multimap \text{I} \frac{\Gamma \mid \Phi, x : A \cdot X \vdash^i t : Y}{\Gamma \mid \Phi \vdash^i \lambda x. t : A \cdot X \multimap Y} \quad \multimap \text{E} \frac{\Gamma \mid \Psi \vdash^i s : X \quad \Gamma \mid \Phi \vdash^i t : A \cdot X \multimap Y}{\Gamma \mid \Phi, A \cdot \Psi \vdash^i t s : Y} \quad 1 \leq A \\
 \\
 \text{II} \frac{}{\Gamma \mid - \vdash^i * : 1} \quad \text{DIRECT} \frac{\Gamma \mid x : \underline{X}^- \vdash f : \underline{X}^+}{\Gamma \mid - \vdash^i \text{direct}(x.f) : X}
 \end{array}
 \end{array}$$

Fig. 3. Typing Rules of INTML[WEC]: Introductions, Eliminations and Direct Definition

$$\begin{array}{c}
 \text{0E}^i \frac{\Gamma \mid - \vdash f : 0}{\Gamma \mid \Phi \vdash^i \text{image}(f) : X} \quad \text{[]E}^c \frac{\Gamma \mid - \vdash^i s : [A] \quad \Gamma, x : A \mid - \vdash f : \underline{B}}{\Gamma \mid - \vdash \text{let } [x] = s \text{ in } f : \underline{B}} \\
 \\
 \text{+E}^i \frac{\Gamma \mid - \vdash f : A + B \quad \Gamma, x : A \mid \Phi \vdash^i s : X \quad \Gamma, y : B \mid \Phi \vdash^i t : X}{\Gamma \mid \Phi \vdash^i \text{case } f \text{ of } \text{inl}(x) \Rightarrow s \mid \text{inr}(y) \Rightarrow t : X}
 \end{array}$$

Fig. 4. Typing Rules of INTML[WEC]: Cross-Eliminations

In this way the computational effects from WEC become available in INTML[WEC]. The term $\text{let } [x] = [f] \text{ in } t$ represents a computation that first executes the computation $f : !A$ thus performing its effects and then continues with the execution of t .

That INTML[WEC] allows only restricted copying is important in order to be able to include weak low-level languages as base languages. With further assumptions about the base language, it is possible to allow full copying. Indeed, if there exists a value type G with $G \times G \leq G$ and $G + G \leq G$ and $A \leq G$ for any other value type A , then INTML[WEC] allows full copying. For instance, one could take for G a type of natural numbers that can encode the values of all value types. This approach is familiar from Geometry of Interaction Situations [1]. Another option, naturally suggested by the requirements $G \times G \leq G$ and $G + G \leq G$, is to use for G a type of trees. Such a type of trees is used by Mackie [17] in an interactive implementation of PCF.

However, fine-grained control of copying is an important feature of INTML[WEC]. This form of copying works even when there is no type G as above. In particular, the metalanguage is well-behaved even for very basic base languages, for example ones having only finite types. For applications such as programming with sublinear space or circuit synthesis it is essential to be able to account for such weak languages. Furthermore, while the multiplicity annotations $A \cdot (-)$ in the types complicate the type system, previous experience with INTML suggest that the type system nevertheless remains manageable [14], as type inference is possible.

INTML[WEC] improves over INTML not only in generality but also in terms of its equational theory. While in INTML only equations between closed terms are given and

$$\begin{array}{l}
(1-\beta) \Gamma \mid \Phi \vdash^i s = * : 1 \text{ if } \Gamma \mid \Phi \vdash^i s : 1 \\
(\otimes-\beta) \Gamma \mid \Phi, \Psi \vdash^i (\text{let } \langle x_1, x_2 \rangle = \langle t_1, t_2 \rangle \text{ in } x_k) = t_k : X_k \text{ for } k = 1, 2 \\
\quad \text{if } \Gamma \mid \Phi \vdash^i t_1 : X_1 \text{ and } \Gamma \mid \Psi \vdash^i t_2 : X_2 \\
(\otimes-\eta) \Gamma \mid \Phi, A \cdot \Psi \vdash^i (\text{let } \langle x, y \rangle = s \text{ in } t[\langle x, y \rangle / z]) = t[s/z] : Z \\
\quad \text{if } \Gamma \mid \Phi \vdash^i s : X \otimes Y \text{ and } \Gamma \mid \Psi, x : A \cdot X, y : A \cdot Y \vdash^i t : Z \\
(\multimap-\beta) \Gamma \mid \Phi, A \cdot \Psi \vdash^i (\lambda x. s) t = s[t/x] : Y \\
\quad \text{if } \Gamma \mid \Phi, x : A \cdot X \vdash^i s : Y \text{ and } \Gamma \mid \Psi \vdash^i t : X \\
(\multimap-\eta) \Gamma \mid \Phi \vdash^i (\lambda x. t x) = t : A \cdot X \multimap Y \text{ if } \Gamma \mid \Phi \vdash^i t : A \cdot X \multimap Y \text{ and } x \notin \Phi. \\
([\]-\beta) \Gamma \mid A \cdot \Phi \vdash^i (\text{let } [x] = [!v] \text{ in } t) = t[v/x] : [B], \\
\quad \Gamma \mid - \vdash (\text{let } [x] = [!v] \text{ in } f) = f[v/x] : \underline{C} \\
\quad \text{if } \Gamma \mid - \vdash v : A \text{ and } \Gamma, x : A \mid \Phi \vdash^i t : [B] \text{ and } \Gamma, x : A \mid - \vdash f : \underline{C} \\
(0^i-\beta) \Gamma \mid \Phi \vdash^i \text{image}(f) = s[f/x] : X \text{ if } \Gamma \mid - \vdash f : 0 \text{ and } \Gamma, x : 0 \mid \Phi \vdash^i s : X \\
(+^i-\beta) \Gamma \mid \Phi \vdash^i (\text{case inl}(f) \text{ of inl}(x) \Rightarrow s \mid \text{inr}(y) \Rightarrow t) = s[f/x] : X, \\
\quad \Gamma \mid \Phi \vdash^i (\text{case inr}(f) \text{ of inl}(x) \Rightarrow s \mid \text{inr}(y) \Rightarrow t) = t[f/y] : X \\
\quad \text{if } \Gamma \mid - \vdash f : A \text{ and } \Gamma, x : A \mid \Phi \vdash^i s : X \text{ and } \Gamma, y : B \mid \Phi \vdash^i t : X \\
(\text{CP}) \Gamma \mid \Psi \vdash^i \text{copy } s[t/x] \text{ as } y, y' \text{ in } u = \text{copy } t \text{ as } x, x' \text{ in } u[s/y, s[x'/x]/y'] : Z \\
\quad \text{if } \{ \Gamma \mid \Phi_i \vdash^i t_i : X_i \}_{i=1}^n \text{ and } \Gamma \mid x_1 : A_1 \cdot X_1, \dots, x_n : A_n \cdot X_n \vdash^i s : Y \text{ and} \\
\quad \Gamma \mid \Psi, y : B \cdot Y, y' : C \cdot Y \vdash^i u : Z \text{ and } \Psi = (B + C) \cdot (A_1 \cdot \Phi_1, \dots, A_n \cdot \Phi_n)
\end{array}$$

Fig. 5. Equations of INTML[WEC]

justified by a semantic interpretation, here we include equations between open terms, as one would expect from a well-behaved type theory, see Fig. 5.

A simply-typed λ -calculus alone is not very expressive as a programming language, of course. The intention in INTML[WEC] is that further constructs can be added by the programmer by direct implementation of combinators. Rule (DIRECT) allows a programmer to define combinators directly by implementing a strategy for it in the base language. Game semantics is a rich source of such strategies. For example, it is possible to implement combinators for loops, for control operators, or for locally scoped state:

$$\begin{array}{l}
\text{loop} : \alpha \cdot (\gamma \cdot [\alpha] \multimap [\alpha + \beta]) \multimap [\alpha] \multimap [\beta] \\
\text{callcc} : (\gamma \cdot ([\alpha] \multimap [\beta]) \multimap [\alpha]) \multimap [\alpha] \\
\text{newvar} : \alpha \cdot (\delta \cdot ((\gamma \cdot [\alpha] \multimap [1]) \otimes [\alpha]) \multimap [\beta]) \multimap [\beta]
\end{array}$$

These and other combinators can be implemented by direct definition, e.g. $\text{loop} = \text{direct}(x.f)$, for suitable f . The definitions of loop and callcc are described in [13]. In essence, callcc implements a game semantic strategy described also by Laird [15]. The combinator newvar represents a memory cell of type α . It is intended to be used as $\text{newvar}(\lambda \langle \text{write}, \text{read} \rangle. t)$. In t the memory cell can be read by means of $\text{let } [x] = \text{read in } s$ and written with value v by $\text{let } [*] = (\text{write } [v]) \text{ in } s$.

Other than congruences there are no equations for direct in INTML[WEC], as this term allows one to implement arbitrary strategies. Equations for specific direct-terms such as loop , callcc or newvar have to be considered on a case-by-case basis.

The above combinators are good examples why it is useful to have value types as bounds for copying, as opposed to natural numbers, say. In newvar , for example, the content of the memory cell is encoded in the number of the argument. Hence, there are α -many copies of the argument, as indicated by the type.

Example. To give a simple concrete example of the use of the metalanguage, we consider the Kierstead terms, which are often used to illustrate the need for justification pointers in Hyland-Ong games [10] for modelling higher-order λ -calculus. With explicit copying, these terms can be given the following types, where α can be any nonempty value type:

$$\begin{aligned} t_1 &= \lambda f. \text{copy } f \text{ as } f_1, f_2 \text{ in } f_1 (\lambda x. f_2 (\lambda y. y)) : (1 + \alpha) \cdot (\alpha \cdot (1 \cdot X \multimap X) \multimap X) \multimap X \\ t_2 &= \lambda f. \text{copy } f \text{ as } f_1, f_2 \text{ in } f_1 (\lambda x. f_2 (\lambda y. x)) : (1 + \alpha) \cdot (\alpha \cdot (\alpha \cdot X \multimap X) \multimap X) \multimap X \end{aligned}$$

What is encoded by the justification pointers in Hyland-Ong games is here, as in Abramsky-Jagadeesan-Malacaria games, encoded in the copy of the function argument.

That these two terms do indeed implement different strategies can be shown by constructing an argument for which t_1 and t_2 give different results. To do this, define the value type $\text{bool} = 1 + 1$ with abbreviations tt and ff for its two elements. It is easy to define in WEC a term $x : \text{bool}, y : \text{bool} \mid - \vdash \text{nor}(x, y) : !\text{bool}$ that returns tt when both x and y are ff and ff otherwise. If we then define the function f of type $(1 + \text{bool}) \cdot (\beta \cdot [\text{bool}] \multimap [\text{bool}]) \multimap [\text{bool}]$ to be $\lambda g. \text{copy } g \text{ as } g_1, g_2 \text{ in let } [x_1] = g_1 [\text{ff}] \text{ in let } [x_2] = g_2 [\text{tt}] \text{ in } [\text{nor}(x_1, x_2)]$, then we have $t_1 f = [\text{tt}]$ and $t_2 f = [\text{ff}]$.

3 Models of the Weak Effect Calculus

Having defined INTML[WEC] as a metalanguage for interaction, our goal is now to show that the Weak Effect Calculus from Sec. 1 can implement this language. We shall do so by a semantic argument, showing that from any model of WEC we can construct a model for INTML[WEC]. A translation from INTML[WEC] to WEC follows by applying this result to a term model of WEC.

We briefly define the semantic structure needed to model WEC. These definitions are a straightforward adaptation of those for EEC in [4]. The notion of a model of WEC uses basic enriched category theory [12].

Values are modelled in a small category \mathbb{V} in a standard way. To account for the indexing of computations over value contexts, computations are modelled in a $\widehat{\mathbb{V}}$ -enriched category \mathbb{C} , where $\widehat{\mathbb{V}} = \mathbf{Set}^{\mathbb{V}^{\text{op}}}$ is the category of presheaves over \mathbb{V} [18]. The $\widehat{\mathbb{V}}$ -enrichment of \mathbb{C} accounts for the indexing over value contexts, as the hom-object $\mathbb{C}(\underline{A}, \underline{B})$ is now a presheaf, i.e. a functor from \mathbb{V}^{op} to \mathbf{Set} . For each value type Γ we have a set $\mathbb{C}(\underline{A}, \underline{B})(\Gamma)$. The terms of type $\Gamma \mid x : \underline{A} \vdash f : \underline{B}$ appear as elements of this set.

The category \mathbb{V} itself can also be seen as a $\widehat{\mathbb{V}}$ -enriched category by $\mathbb{V}(A, B) = (\mathbf{y}B)^{(\mathbf{y}A)}$, where $\mathbf{y}A = \mathbb{V}(-, A)$ is the Yoneda embedding. Using the Yoneda lemma one can see that this enrichment can be spelled out as $\mathbb{V}(A, B)(\Gamma) \simeq \mathbb{V}(\Gamma \times A, B)$.

A model for WEC then consists of a $\widehat{\mathbb{V}}$ -enriched adjunction $F \dashv U : \mathbb{V} \rightarrow \mathbb{C}$ with the following structure:

1. In \mathbb{V} : finite products and finite coproducts which distributive over products; this induces $\widehat{\mathbb{V}}$ -enriched finite products and $\widehat{\mathbb{V}}$ -enriched finite coproducts in \mathbb{V} .
2. In \mathbb{C} : $\widehat{\mathbb{V}}$ -enriched finite coproducts and copowers indexed by representables. The latter means that for each object A in \mathbb{V} and each object \underline{B} in \mathbb{C} there is an object $A \cdot \underline{B}$ in \mathbb{C} and an isomorphism $\mathbb{C}(\underline{B}, \underline{C})^{\mathbf{y}A} \simeq \mathbb{C}(A \cdot \underline{B}, \underline{C})$ that is $\widehat{\mathbb{V}}$ -natural in \underline{C} . As in the syntax of WEC, we write \oplus and $\underline{0}$ for binary coproducts and initial object.

3. In \mathbb{C}^{op} : a uniform parametrised Conway operator in the sense that there is a map $(-)^{\dagger}: \mathbb{C}(\underline{A}, \underline{B} \oplus \underline{A}) \rightarrow \mathbb{C}(\underline{A}, \underline{B})$ in $\widehat{\mathbb{V}}$ that when considered as a map of type $\mathbb{C}^{\text{op}}(\underline{B} \oplus \underline{A}, \underline{A}) \rightarrow \mathbb{C}^{\text{op}}(\underline{B}, \underline{A})$ satisfies the equations of a uniform Conway operator [21]. Note that \oplus is a product in \mathbb{C}^{op} .
4. We require this structure to be such that the canonical maps $\mathbb{C}(X, Y)^{y_0} \rightarrow 1$ and $\mathbb{C}(X, Y)^{y(A+B)} \rightarrow \mathbb{C}(X, Y)^{y^A} \times \mathbb{C}(X, Y)^{y^B}$ are isomorphisms. This requirement is used to model the elimination of 0 and $A + B$ over computations.

Value and computation types are interpreted in this structure as objects of \mathbb{V} and \mathbb{C} respectively. If a computation type is used as a value type, then this is modelled by an application of the functor U . In the other direction, the type $!A$ is interpreted by FA . A value sequent $\Gamma \mid - \vdash f: A$ is interpreted as an element of $\mathbb{V}(1, A)(\Gamma)$. A computation sequent $\Gamma \mid x: \underline{A} \vdash g: \underline{B}$ appears in the model as an element of $\mathbb{C}(\underline{A}, \underline{B})(\Gamma)$. While value sequents $\Gamma \mid - \vdash f: \underline{B}$ defining a term of computation type are interpreted as elements of $\mathbb{V}(1, U\underline{B})(\Gamma)$, by the adjunction $F \dashv U$ their interpretation is in one-to-one correspondence with $\mathbb{C}(F1, \underline{B})(\Gamma)$, so that they may also be seen as computations.

A simple example of a model can be obtained by letting \mathbb{V} be the category of finite sets and \mathbb{C} be the $\widehat{\mathbb{V}}$ -category of finite pointed sets and strict functions, i.e. an object of \mathbb{C} is a finite set \underline{A} with a distinguished element \perp and $\mathbb{C}(\underline{A}, \underline{B})(\Gamma)$ consists of all functions $f: \Gamma \times \underline{A} \rightarrow \underline{B}$ that satisfy $f(\gamma, \perp) = \perp$ for any $\gamma \in \Gamma$.

A second example is a term model. The objects of \mathbb{V} and \mathbb{C} are the value types and computation types respectively. The morphisms from A to B in \mathbb{V} are terms $x: A \mid - \vdash f: B$, identified up to equality. Likewise, $\mathbb{C}(\underline{A}, \underline{B})(C)$ consists of terms $x: C \mid y: \underline{A} \vdash g: \underline{B}$ identified up to provable equality.

As the notation suggests, we use the copower to interpret copying in INTML[WEC]. We have found that the copower identifies just the right structure for this purpose. For any object A define \mathbb{C}^A to be the $\widehat{\mathbb{V}}$ -category with the same objects as \mathbb{C} and with hom-objects $\mathbb{C}^A(\underline{B}, \underline{C}) = \mathbb{C}(\underline{B}, \underline{C})^{y^A}$. Just as \mathbb{C} models sequents of the form $\Gamma \mid y: \underline{B} \vdash f: \underline{C}$, \mathbb{C}^A models sequents of the form $\Gamma, x: A \mid y: \underline{B} \vdash f: \underline{C}$. There is a canonical $\widehat{\mathbb{V}}$ -functor $W_A: \mathbb{C} \rightarrow \mathbb{C}^A$, that amounts to weakening. Then, the copower extends to a $\widehat{\mathbb{V}}$ -functor $A \cdot (-): \mathbb{C}^A \rightarrow \mathbb{C}$ that is left adjoint to W_A (in a $\widehat{\mathbb{V}}$ -enriched sense). As a left adjoint, $A \cdot (-)$ preserves sums. Moreover, we have a canonical isomorphism $A \cdot FB \simeq F(A \times B)$, as $A \times (-)$ is a copower in \mathbb{V} and copowers, being a particular form of colimit, are preserved by the left adjoint F .

Lemma 1. *In \mathbb{C} there are the isomorphisms $1 \cdot \underline{C} \simeq \underline{C}$ and $(A \times B) \cdot \underline{C} \simeq A \cdot (B \cdot \underline{C})$ and $0 \cdot \underline{C} \simeq 0$ and $(A + B) \cdot \underline{C} \simeq A \cdot \underline{C} + B \cdot \underline{C}$, which are all natural in A, B and \underline{C} .*

To establish the last two isomorphisms we use the assumption in point 4 of the definition of a model above.

3.1 Trace and Int-Construction

The first step in constructing a model of INTML[WEC] from a model of WEC is to apply the Int-construction [11] to the category of computations. This construction is well-known in the context of game semantics; it has been used by Abramsky and Jagadeesan to model AJM-games [2].

Lemma 2. *The $\widehat{\mathbb{V}}$ -category \mathbb{C} has a uniform trace with respect to coproducts as monoidal structure, in the sense that there is a map $Tr_{\underline{B}, \underline{C}, \underline{D}}: \mathbb{C}(\underline{B} \oplus \underline{C}, \underline{D} \oplus \underline{C}) \rightarrow \mathbb{C}(\underline{B}, \underline{D})$ in $\widehat{\mathbb{V}}$ that satisfies the usual equations for a uniform trace [7].*

Such a uniform trace can be constructed from a uniform Conway operator, as has been shown (in a dual setting) by Hasegawa [7].

Lemma 3. *If \mathbb{C} has a uniform trace with respect to coproducts then so does \mathbb{C}^A and both $\widehat{\mathbb{V}}$ -functors $W_A: \mathbb{C} \rightarrow \mathbb{C}^A$ and $A \cdot (-): \mathbb{C}^A \rightarrow \mathbb{C}$ preserve the trace.*

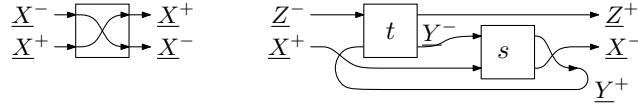
For the proof that $A \cdot (-)$ preserves the trace we need the assumption of uniformity.

It is useful to use a graphical notation for working with the traced monoidal category \mathbb{C} . We denote an element $f \in \mathbb{C}(\underline{B} \oplus \underline{C}, \underline{D} \oplus \underline{C})(\Gamma)$ as in the box on left below and use similar standard notation for the traced monoidal structure. For example, the result of applying the trace to f is shown next to f below. However, note that these diagrams are now used to work in $\widehat{\mathbb{V}}$, so that care is needed to verify, e.g., naturality conditions.

For the copower functor we use a box-notation as shown in the equation on the right below. In that equation $g \in \mathbb{C}^A(\underline{B} \oplus \underline{C}, \underline{D} \oplus \underline{E})(\Gamma) \simeq \mathbb{C}(\underline{B} \oplus \underline{C}, \underline{D} \oplus \underline{E})(\Gamma \times A)$. The equation expresses that $A \cdot (-)$ preserves the trace. The box-notation is justified, as $A \cdot (-)$ is a monoidal functor with respect to coproducts.



The Int-construction then defines a $\widehat{\mathbb{V}}$ -category $\text{Int}(\mathbb{C})$ as follows. An object X is a pair $(\underline{X}^-, \underline{X}^+)$ of \mathbb{C} -objects. The hom-objects for $\text{Int}(\mathbb{C})$ are defined by $\text{Int}(\mathbb{C})(X, Y) = \mathbb{C}(\underline{Y}^- \oplus \underline{X}^+, \underline{Y}^+ \oplus \underline{X}^-)$. The definition of the identity $1 \rightarrow \text{Int}(\mathbb{C})(X, X)$ and the composition $\text{Int}(\mathbb{C})(Y, Z) \times \text{Int}(\mathbb{C})(X, Y) \rightarrow \text{Int}(\mathbb{C})(X, Z)$ are best understood when given as graphical diagrams in \mathbb{C} :



Lemma 4. *$\text{Int}(\mathbb{C})$ is a $\widehat{\mathbb{V}}$ -category with a monoidal closed structure (I, \otimes, \multimap) with $I = (0, 0)$, $X \otimes Y = (\underline{X}^- + \underline{Y}^-, \underline{X}^+ + \underline{Y}^+)$ and $X \multimap Y = (\underline{X}^+ + \underline{Y}^-, \underline{X}^- + \underline{Y}^+)$.*

The structure in this lemma is well-known, see e.g. [8].

To model copying in INTML[WEC] we use in addition a functor $X^{\otimes A}$ that informally captures an A -fold tensor $X \otimes \cdots \otimes X$. To define it formally, define a category \mathbb{V}_{sr} of all section-retraction-pairs in \mathbb{V} , i.e. a morphism from A to B is a pair $\langle s, r \rangle \in \mathbb{V}(A, B) \times \mathbb{V}(B, A)$ with $r \circ s = id$. Note that \mathbb{V}_{sr} is again canonically $\widehat{\mathbb{V}}$ -enriched.

We define the object $X^{\otimes A}$ to be $(A \cdot \underline{X}^-, A \cdot \underline{X}^+)$. This definition can be extended to a functor in each argument, i.e. to $(-)^{\otimes A}: \text{Int}(\mathbb{C}) \rightarrow \text{Int}(\mathbb{C})$ and $X^{\otimes (-)}: (\mathbb{V}_{\text{sr}})^{\text{op}} \rightarrow \text{Int}(\mathbb{C})$. That \mathbb{V}_{sr} consists of section-retraction pairs ensures functoriality of this definition. We note that these two functors do *not* combine to a bifunctor $(-)^{\otimes (-)}$.

As $X^{\otimes A}$ amounts to repeated multiplication, there are isomorphisms that correspond to well-known rules of high-school arithmetic:

$$\begin{aligned} X^{\otimes 1} &\simeq X & X^{\otimes(A \times B)} &\simeq (X^{\otimes A})^{\otimes B} & I^{\otimes A} &\simeq I \\ X^{\otimes 0} &\simeq I & X^{\otimes(A+B)} &\simeq X^{\otimes A} \otimes X^{\otimes B} & (X \otimes Y)^{\otimes A} &\simeq X^{\otimes A} \otimes Y^{\otimes A} \end{aligned} \quad (1)$$

These isomorphisms follow from Lemma 1. They are natural in A , B , X and Y .

4 Interpreting the Metalanguage in WEC

Starting from $\text{Int}(\mathbb{C})$, we now build a model that can interpret $\text{INTML}[\text{WEC}]$. While the terms of $\text{INTML}[\text{WEC}]$ can already be interpreted in $\text{Int}(\mathbb{C})$, this interpretation validates only equations between closed terms. For example, given $\Sigma \mid \Phi \vdash^i s: X$ and $\Sigma \mid \Psi \vdash^i t: Y$ it does not have to be the case that $\Sigma \mid \Phi, \Psi \vdash^i \text{let } \langle x, y \rangle = \langle s, t \rangle \text{ in } x: X$ and $\Sigma \mid \Phi, \Psi \vdash^i s: X$ receive the same denotation. This is because we may start a dialogue with these terms by sending messages to the variables of t , i.e. those in Ψ . In the first term these messages are processed by t , while in the second term they are just discarded. We would like to discount such differences, as they appear only if one answers questions that have never been asked.

Another reason why $\text{Int}(\mathbb{C})$ does not justify open equations is that in the interpretation of rule (STRUCT) an arbitrary section-retraction pair between A and B may be chosen and different such choices can be observed in $\text{Int}(\mathbb{C})$. However, different choices cannot affect the final result of computations, so that we would like to consider their implementation details that should not be taken into account when considering program equality.

In order to explain in which sense we consider the behaviour of programs equal, we now take a quotient of the model with respect to a form of logical relations. This quotient is similar to the ones taken in AJM-games [2], but with WEC with unspecified effects as a basis, we cannot use an equivalence relation on traces of values as in [2].

The following definition is to be understood internally in the presheaf topos $\widehat{\mathbb{V}}$. In it we denote by $|\mathbb{C}|$ the discrete category with the same objects as \mathbb{C} .

Definition 1. A Kripke partial equivalence relation with arity $\mathcal{I}: |\mathbb{C}| \rightarrow \text{Int}(\mathbb{C})$ over an object X in $\text{Int}(\mathbb{C})$ is a family of partial equivalence relations

$$(R(\underline{A}) \subseteq \text{Int}(\mathbb{C})(\mathcal{I}\underline{A}, X) \times \text{Int}(\mathbb{C})(\mathcal{I}\underline{A}, X))_{\underline{A} \in |\mathbb{C}|} .$$

In the following we use such Kripke partial equivalence relations with respect to the arity \mathcal{I} defined by $\mathcal{I}(\underline{A}) = (\underline{A}, 0)$. Notice that an element of $\text{Int}(\mathbb{C})(\mathcal{I}\underline{A}, X)$ corresponds to a map in $\mathbb{C}(\underline{X}^-, \underline{X}^+ \oplus \underline{A})$. We use such maps, as opposed to just $\mathbb{C}(\underline{X}^-, \underline{X}^+)$ in order to avoid making the quotient too strong. Without the presence of \underline{A} a morphism modelling `callcc` would be ruled out, for example.

We fix some notation. First note that, for any \underline{A} , there is a canonical morphism $\Delta_{\underline{A}}: \mathcal{I}(\underline{A}) \rightarrow \mathcal{I}(\underline{A}) \otimes \mathcal{I}(\underline{A})$. Similarly, there is a canonical morphism $\Delta_{\underline{A}}^{\otimes B}: \mathcal{I}(\underline{A}) \rightarrow \mathcal{I}(\underline{A})^{\otimes B}$. Given $e: \mathcal{I}\underline{A} \rightarrow X$ we write short $e^{\otimes B}$ for $e^{\otimes B} \circ \Delta_{\underline{A}}^{\otimes B}: \mathcal{I}\underline{A} \rightarrow X^{\otimes B}$.

Given a small model of WEC, we construct a $\widehat{\mathbb{V}}$ -category \mathbb{I} , which can model INTML[WEC]. The construction works by restricting and quotienting $\text{Int}(\mathbb{C})$ in the style of models based on partial equivalence relations.

The objects of \mathbb{I} are pairs (X, R_X) of an *underlying object* X in $\text{Int}(\mathbb{C})$ and a Kripke partial equivalence relation over X . The $\widehat{\mathbb{V}}$ -object $\mathbb{I}(X, Y)$ of morphisms from X to Y consists of the quotient of $\text{Int}(\mathbb{C})(X, Y)$ under the partial equivalence relation \sim defined as follows in the internal logic of $\widehat{\mathbb{V}}$:

$$f \sim g \iff \forall \underline{A}. \forall (e, e') \in R_X(\underline{A}). (f \circ e, g \circ e') \in R_Y(\underline{A})$$

We define in \mathbb{I} objects 1 , $[B]$, $X \otimes Y$, $X \multimap Y$ and $X^{\otimes A}$ as follows. The underlying object of 1 is I and the underlying objects of the others are the $\text{Int}(\mathbb{C})$ -object of the same name. The relations are defined by:

$$\begin{aligned} R_1(\underline{A}) &= \top \\ R_{[B]}(\underline{A}) &= \{(e, e) \mid e \in \text{Int}(\mathbb{C})(\underline{I}\underline{A}, [B])\} \\ R_{X \otimes Y}(\underline{A} \oplus \underline{B}) &= \{(e_1 \otimes e_2, e'_1 \otimes e'_2) \mid (e_1, e'_1) \in R_X(\underline{A}) \wedge (e_2, e'_2) \in R_Y(\underline{B})\} \\ R_{X \multimap Y}(\underline{A}) &= \{(f, f') \mid \forall \underline{B}. \forall (e, e') \in R_X(\underline{B}). (\varepsilon \circ \langle f, e \rangle, \varepsilon \circ \langle f', e' \rangle) \in R_Y(\underline{A} \oplus \underline{B})\} \\ R_{X^{\otimes B}}(\underline{A}) &= \{(e_1^{\otimes B}, e_2^{\otimes B}) \mid (e_1, e_2) \in R_X(\underline{A})\} \end{aligned}$$

We use these constructions for the interpretation of INTML[WEC] in \mathbb{I} . The types 1 , $[A]$ and $X \otimes Y$ are interpreted by the corresponding objects. The function type $A \cdot X \multimap Y$ is interpreted by the object $(X^{\otimes A}) \multimap Y$. A term $\Gamma \mid x_1 : A_1 \cdot X_1, \dots, x_n : A_n \cdot X_n \vdash^i t : Y$ is interpreted as an element of $\mathbb{I}(X_1^{\otimes A_1} \otimes \dots \otimes X_n^{\otimes A_n}, Y)(\Gamma)$.

Lemma 5. $[-]$ is a $\widehat{\mathbb{V}}$ -functor from $Kl(T)$ to \mathbb{I} , where $Kl(T)$ is the $\widehat{\mathbb{V}}$ -category with same objects as \mathbb{V} and with $Kl(T)(A, B) = \mathbb{C}(FA, FB)$.

Lemma 6. $(1, \otimes, \multimap)$ defines a symmetric monoidal closed structure in \mathbb{I} whose unit 1 is terminal.

The definition of $X^{\otimes A}$ in \mathbb{I} is such that it informally represents A -many copies of the *same* element of X . As reordering such tuples of several copies of the same element has no effect, we may replace \mathbb{V}_{sr} by a preorder \leq on \mathbb{V} -objects. It is defined such that $A \leq B$ holds if and only if there is a morphism from A to B in \mathbb{V}_{sr} . We write \mathbb{V}_{\leq} for the category arising from this preorder.

Lemma 7. The definition of $X^{\otimes B}$ extends to a \mathbb{V} -functor $(-)^{\otimes(-)} : \mathbb{I} \times \mathbb{V}_{\leq}^{\text{op}} \rightarrow \mathbb{I}$ and there are morphisms that are natural in A, B, X and Y .

$$\begin{aligned} X^{\otimes 1} &\cong X & X^{\otimes(A \times B)} &\cong (X^{\otimes A})^{\otimes B} & 1^{\otimes A} &\cong 1 \\ X^{\otimes 0} &\cong 1 & X^{\otimes(A+B)} &\cong X^{\otimes A} \otimes X^{\otimes B} & (X \otimes Y)^{\otimes A} &\cong X^{\otimes A} \otimes Y^{\otimes A} \end{aligned}$$

The next lemma follows immediately from the definition of the morphisms of \mathbb{I} as equivalence classes under \sim .

Lemma 8. The functor $U : \mathbb{I} \rightarrow \widehat{\mathbb{V}}$ that maps an object X to the \mathbb{V} -set of equivalence classes of $\bigcup_{\underline{A}} R_X(\underline{A})$ and a morphism $[f]_{\sim} : X \rightarrow Y$ to the function $[e] \mapsto [f \circ e]$ is faithful and maps the monoidal structure $(1, \otimes, \multimap)$ to $(1, \times, \Rightarrow)$.

Note in particular that U sends X and $X^{\otimes A}$ to isomorphic objects. The isomorphisms from Lemma 7 are all mapped to the identity; $X^{\otimes(A+B)} \rightarrow X^{\otimes A} \otimes X^{\otimes B}$ is mapped to the diagonal. When reasoning about the identity of maps in \mathbb{I} we therefore do not need to consider the explicit duplication by means of $X^{\otimes A}$.

The next two lemmas capture the structure needed to interpret rules ($[]I$), ($[]E$) and the β -equation (let $[y] = [!x]$ in t) = $t[x/y]$.

Lemma 9. *There is an isomorphism $\varphi: \mathbb{I}(1, [B]) \xrightarrow{\cong} Kl(T)(1, B)$ that is natural in B .*

Proof. Note that $\mathbb{I}(1, [B])$ is isomorphic to $\text{Int}(\mathbb{C})(1, [B])$, which by definition is isomorphic to $\mathbb{C}(\underline{0} \oplus F1, FB \oplus \underline{0})$. By definition of $Kl(T)(1, B)$ the result follows. \square

Lemma 10. *There exists a map $\psi: \mathbb{I}(X, [B])^{\mathfrak{y}A} \rightarrow \mathbb{I}(X^{\otimes A} \otimes [A], [B])^{\mathfrak{y}A}$ that is natural in X and that any square of the following form commutes.*

$$\begin{array}{ccc} \mathbb{I}(X, [B])^{\mathfrak{y}A} & \xrightarrow{\langle i \circ \psi, r \rangle} & \mathbb{I}(X^{\otimes A} \otimes [A], [B])^{\mathfrak{y}A} \times \mathbb{I}(1, [A])^{\mathfrak{y}A} \\ \langle id, l \rangle \downarrow & & \downarrow \\ \mathbb{I}(X, [B])^{\mathfrak{y}A} \times \mathbb{I}(X^{\otimes A}, X)^{\mathfrak{y}A} & \longrightarrow & \mathbb{I}(X^{\otimes A}, [B])^{\mathfrak{y}A} \end{array}$$

Therein r and l are the canonical composites $r: 1 \rightarrow Kl(1, A)^{\mathfrak{y}A} \rightarrow \mathbb{I}(1, [A])^{\mathfrak{y}A}$ and $l: 1 \rightarrow \mathbb{V}(1, A)^{\mathfrak{y}A} = \mathbb{V}_{sr}(1, A)^{\mathfrak{y}A} \rightarrow \mathbb{I}(X^{\otimes A}, X^{\otimes 1})^{\mathfrak{y}A} \simeq \mathbb{I}(X^{\otimes A}, X)^{\mathfrak{y}A}$ and the unlabelled maps are the canonical maps of their type.

We spell out the proof since it gives a good example of how to work with \mathbb{I} and because it illustrates that the copower fits in very well with the string diagrams.

Proof. Since morphisms in \mathbb{I} are equivalence classes, we define ψ on representatives and observe that the definition does not depend on the choice of representative. Given $f \in \text{Int}(\mathbb{C})(X, [B])^{\mathfrak{y}A}(\Gamma)$, define $\psi(f)$ to be the equivalence class of the following morphism in $\text{Int}(\mathbb{C})(X^{\otimes A} \otimes [A], [B])^{\mathfrak{y}A}(\Gamma)$.

Here η and ε denote the unit and counit of the adjunction $A \cdot (-) \dashv W_A$. The box labelled with \simeq denotes the canonical isomorphism arising as F preserves copowers.

To show that the square in the lemma commutes, let $f \sim f'$ be two representatives of an equivalence class in $\mathbb{I}(X, [B])^{\mathfrak{y}A}(\Gamma)$. The two composites in the square then give the following two maps in $\mathbb{I}(X^{\otimes A}, [B])^{\mathfrak{y}A}(\Gamma)$.

We have to show that they are \sim -related, which amounts to showing that $(e, e') \in R'_X(\underline{C})$ implies the following equality:

We can simplify the left morphism left by joining the two $A \cdot (-)$ -boxes, using functoriality and naturality of the trace. We obtain the morphism on the left below. By noting that the adjunction $A \cdot (-) \dashv W_A$ gives us $W_A(\varepsilon) \circ W_A(A \cdot h) \circ \eta$ for any h by naturality of η and the triangular identity, we obtain the equality below (in which W_A is implicit).

The right morphism in the equation that we have to show can similarly be simplified. We obtain the same result with f' instead of f . The result then follows from $f \sim f'$. \square

For the interpretation of $+$ -cross-elimination we use the following lemma.

Lemma 11. *The canonical $\mathbb{I}(X, Y)^{\mathfrak{y}(A+B)} \rightarrow \mathbb{I}(X, Y)^{\mathfrak{y}A} \times \mathbb{I}(X, Y)^{\mathfrak{y}B}$ is isomorphic.*

The interpretation of INTML[WEC] is now defined such that the types and terms of WEC are interpreted in \mathbb{V} and \mathbb{C} , as in [4]. The terms of the metalanguage are interpreted in \mathbb{I} .

Theorem 1 (Soundness). *The construction of \mathbb{I} as quotient of $\text{Int}(\mathbb{C})$ yields a model of INTML[WEC]. That is, INTML[WEC] can be soundly interpreted in \mathbb{I} , given that rule (DIRECT) is restricted to define only morphisms in \mathbb{I} .*

The proof uses standard lemmas for substitution and for showing that two derivations of the same sequent have the same interpretation. A translation of INTML[WEC] to WEC is obtained by starting the construction of \mathbb{I} with the term model of WEC.

5 Conclusion

We have shown that a simple variant of the enriched effect calculus provides enough structure to support the construction of a metalanguage INTML[WEC] for interaction. This improves on the previous construction of INTML in a number of ways. We show that any computational effect that justifies the equations of WEC can be added to the base language. We justify equations between open terms in INTML[WEC], not just closed equations as in [13], by means of a quotient in $\text{Int}(\mathbb{C})$. In contrast to the construction in [13], we do not need to consider an operational semantics of the base language and make the construction using an equational theory only. Finally, we show that the structure of INTML can be accounted for in an enriched setting by a relatively simple

model construction. In particular the copower type from EEC turns out to be the right structure for modelling bounded copying in the metalanguage.

Acknowledgments. Rasmus Møgelberg first noticed a similarity of EEC and INTML and suggested to study their relationship. INTML was developed with Ugo Dal Lago [13]. I thank the anonymous referees for their interesting comments and suggestions.

References

1. S. Abramsky, E. Haghverdi, and P. J. Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.
2. S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Inf. Comput.*, 111(1):53–119, 1994.
3. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.
4. J. Egger, R. E. Møgelberg, and A. Simpson. Enriching an effect calculus with linear types. In *CSL*, volume 5771 of *LNCS*, pp. 240–254. Springer, 2009.
5. J. Egger, R. E. Møgelberg, and A. Simpson. Linearly-used Continuations in the Enriched Effect Calculus. In *FOSSACS*, volume 6014 of *LNCS*, pp. 18–32. Springer, 2010.
6. D. R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In *POPL*, pp. 363–375. ACM, 2007.
7. M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. Distinguished Dissertation Series. Springer-Verlag, 1999.
8. M. Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 19(2):217–244, 2009.
9. I. Hasuo and N. Hoshino. Semantics of Higher-Order Quantum Computation via Geometry of Interaction In *LICS*, 2011.
10. J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.
11. A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Cambridge Philos. Soc.*, 119(3):447–468, 1996.
12. G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *Lecture Notes in Mathematics*. Cambridge University Press, 1982.
13. U. Dal Lago and U. Schöpp. Functional programming in sublinear space. In *ESOP*, volume 6012 of *LNCS*, pp. 205–225. Springer, 2010.
14. U. Dal Lago and U. Schöpp. Type inference for sublinear space functional programming. In *APLAS*, volume 6461 of *LNCS*, pp. 376–391. Springer, 2010.
15. J. Laird. Full abstraction for functional languages with control. In *LICS*, pp. 58–67, 1997.
16. P. B. Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004.
17. I. Mackie. The geometry of interaction machine. In *POPL*, pp. 198–208, 1995.
18. S. MacLane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1994.
19. E. Moggi. Computational lambda-calculus and monads. In *LICS*, pp. 14–23. IEEE Computer Society, 1989.
20. Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic nominal game semantics. In *ESOP*, volume 6602 of *LNCS*, pp. 419–438. Springer, 2011.
21. A. Simpson and G. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS*, pp. 30–41, 2000.

A Equations of EEC

We list the equations of EEC that have been omitted from the main text. We state the standard equations without typing information with the understanding that these equations apply only if both sides are typeable:

$$\begin{array}{ll}
 \text{fst}(\langle f, g \rangle) = f & f = * \text{ if } f : 1 \\
 \text{snd}(\langle f, g \rangle) = g & \text{image}(f) = g[f/x] \text{ if } x : 0 \text{ or } x : \underline{0} \\
 \langle \text{fst}(f), \text{snd}(f) \rangle = f & \text{case inl}(f) \text{ of } \begin{cases} \text{inl}(x) \Rightarrow g \\ \text{inr}(y) \Rightarrow h \end{cases} = g[f/x] \\
 (\text{let } x \cdot y = f \cdot g \text{ in } h) = h[f/x, g/y] & \text{case inr}(f) \text{ of } \begin{cases} \text{inl}(x) \Rightarrow g \\ \text{inr}(y) \Rightarrow h \end{cases} = h[f/y] \\
 (\text{let } x \cdot y = f \text{ in } g[x \cdot y/z]) = g[f/z] & \text{case } f \text{ of } \begin{cases} \text{inl}(x) \Rightarrow g[\text{inl}(x)/z] \\ \text{inr}(y) \Rightarrow g[\text{inr}(y)/z] \end{cases} = g[f/z] \\
 (\text{let } !x = !f \text{ in } g) = g[f/x] & \\
 (\text{let } x = f \text{ in } g[!x/y]) = g[f/y] &
 \end{array}$$

The uniform Conway operator in our variant of EEC has the following equations. Fixpoint:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : \underline{B} \oplus \underline{A}}{\Gamma \mid \Delta \vdash (\text{let } x = f \text{ loop } g) = \text{case } g[f/x] \text{ of } \begin{cases} \text{inl}(x) \Rightarrow x \\ \text{inr}(y) \Rightarrow \text{let } x = y \text{ loop } g \end{cases} : \underline{A}}$$

Naturality:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : \underline{B} \oplus \underline{A} \quad \Gamma \mid y : \underline{B} \vdash h : \underline{C}}{\Gamma \mid \Delta \vdash h[\text{let } x = f \text{ loop } g/y] = \text{let } x = f \text{ loop case } g \text{ of } \begin{cases} \text{inl}(y) \Rightarrow \text{inl}(h) \\ \text{inr}(z) \Rightarrow \text{inr}(z) \end{cases} : \underline{C}}$$

Dinaturality:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : \underline{C} \oplus \underline{B} \quad \Gamma \mid y : \underline{B} \vdash h : \underline{C} \oplus \underline{A}}{\Gamma \mid \Delta \vdash \text{let } x = f \text{ loop case } g \text{ of } \text{inl}(z) \Rightarrow \text{inl}(z) \mid \text{inr}(y) \Rightarrow h = \text{case } g[f/x] \text{ of } \begin{cases} \text{inl}(z) \Rightarrow z \\ \text{inr}(y) \Rightarrow \text{let } y = y \text{ loop case } h \text{ of } \begin{cases} \text{inl}(x) \Rightarrow \text{inl}(x) \\ \text{inr}(y) \Rightarrow g \end{cases} \end{cases} : \underline{C}}$$

Diagonal:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : (\underline{C} \oplus \underline{A}) \oplus \underline{A}}{\Gamma \mid \Delta \vdash (\text{let } y = f \text{ loop } (\text{let } x = y \text{ loop } g)) = \text{let } x = f \text{ loop case } g \text{ of } \begin{cases} \text{inl}(z) \Rightarrow z \\ \text{inr}(y) \Rightarrow \text{inr}(y) \end{cases} : \underline{C}}$$

Uniformity:

$$\frac{\Gamma \mid \Delta \vdash f : \underline{A} \quad \Gamma \mid x : \underline{A} \vdash g : \underline{C} \oplus \underline{A} \quad \Gamma \mid y : \underline{B} \vdash h : \underline{C} \oplus \underline{B} \quad \Gamma \mid x : \underline{A} \vdash k : \underline{B}}{\Gamma \mid \Delta \vdash (\text{let } x = f \text{ loop } g) = (\text{let } y = k[f/x] \text{ loop } h) : \underline{C}}$$

B Combinators Defined by Direct Definition

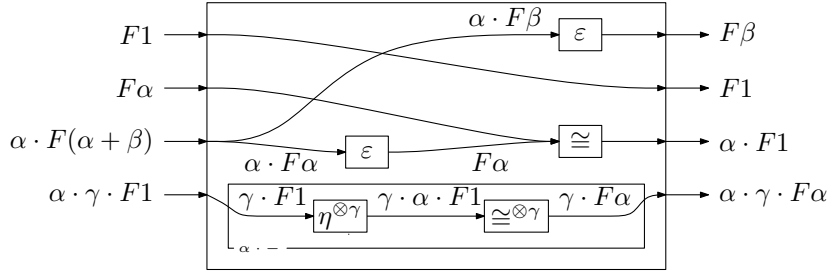
In the soundness theorem we have restricted the rule (DIRECT) so that it may only be used with terms that do indeed define morphisms of \mathbb{I} . To show that this restriction does not rule out interesting combinations, we outline how it can be seen that the combinators `loop` and `callcc` mentioned in Sec. 2 do appear in \mathbb{I} .

B.1 Loop

First we consider the `loop`-combinator for iteration.

$$\text{loop}: \alpha \cdot (\gamma \cdot [\alpha] \multimap [\alpha + \beta]) \multimap [\alpha] \multimap [\beta]$$

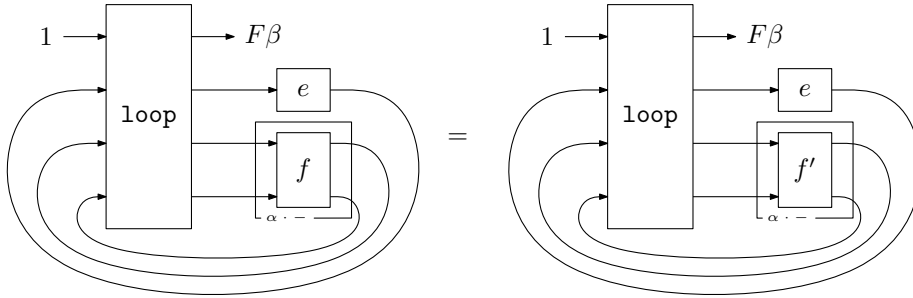
Such a combinator can be defined directly as $\text{loop} = \text{direct}(x.l)$, where l is a term defining the following \mathbb{C} -morphism:



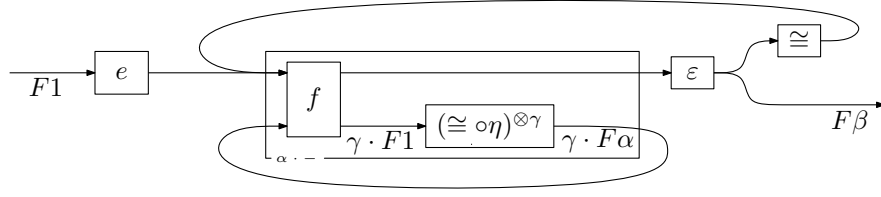
The term `loop` denotes the corresponding morphism in \mathbb{I} .

It is clear that `loop` is a morphism in $\text{Int}(\mathbb{C})$. To show that it is in fact also a morphism in \mathbb{I} , we must show that it preserves the relation \sim . This amounts to showing that related elements are mapped to related elements. Let us for simplicity show only that relations of the form $R(\underline{0})$ are preserved. The general case $R(\underline{A})$ for arbitrary \underline{A} is no more difficult, but the notation is a little more complicated.

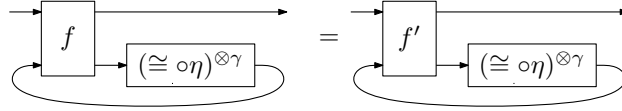
Since on objects of the form $[B]$, the relation is equality, this means we have to show the following equation for arbitrary related f and f' .



The morphism on the left can be simplified to:



As the morphism on the right can similarly be simplified and because the copower-box preserves the trace, it therefore suffices to show the following equality.

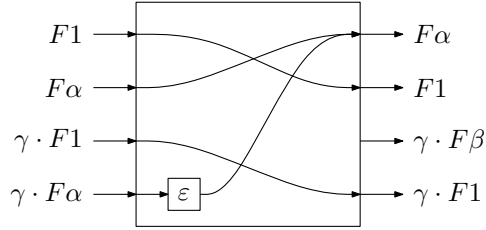


But this follows from the assumption that f and f' are related.

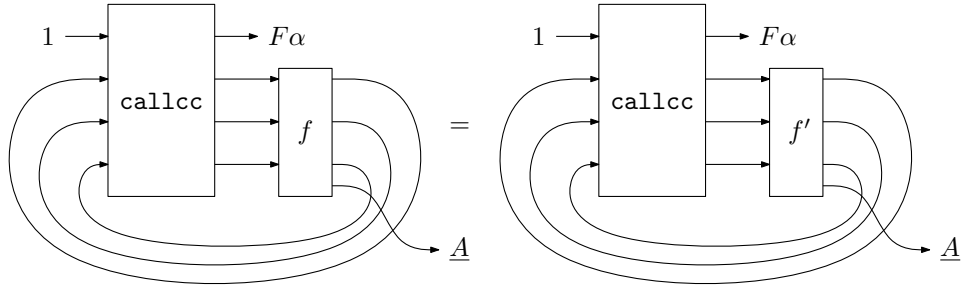
B.2 Callcc

$$\text{callcc}: (\gamma \cdot ([\alpha] \multimap [\beta]) \multimap [\alpha]) \multimap [\alpha]$$

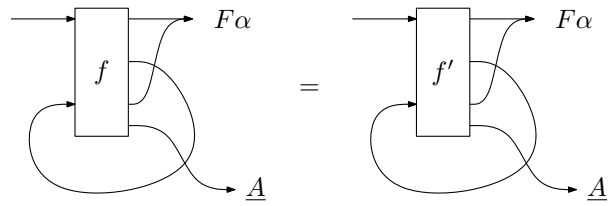
In this type we have omitted $1 \cdot (-)$ for simplicity, which we shall also do in the following. This combinator can be defined directly as $\text{callcc} = \text{direct}(x.c)$, where c is a term defining the following \mathbb{C} -morphism:



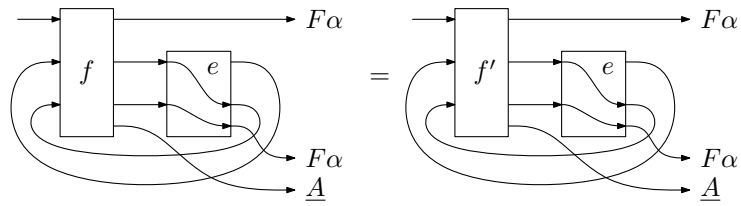
To show that this defines a morphism in \mathbb{I} , we have to show the following equality for any two related arguments f and f' . Here we consider the general case of relations of the form $R(\underline{A})$ for arbitrary \underline{A} .



By unfolding of the definition of callcc and simplification, this goal reduces to:



But we know that f and f' are related, so we know that they map related arguments to related results. By choosing e as shown below, we therefore have the following equality:



But from the the required equality follows immediately.

The combinator `newvar` can be justified in particular models of EEC, such as the one consisting of sets and functions. Therein it can be justified by considering traces of values, in the style of game semantics. It appears that reasoning with such traces is sound only for instances of EEC with certain particular effects, for example if non-termination is the only effect. Indeed our intended used of `newvar` were in such a variant of EEC. We do not know yet if the combinator can be justified in general, or if further assumptions are needed about the base language.