Process Realizability

Samson Abramsky

1 Introduction

Realizability has proved to be a fruitful approach to the semantics of computation, see e.g. [AL91, Cro93, Lon95, AC98]. The *scope* of realizability methods has been limited to *Intuitionistic Logic* (with some extensions to Classical Logic), on the logical side, and to *functional computation* on the computational side. Our aim in the present work is to explore the possibilities for broadening the scope of realizability:

- beyond Intuitionistic Logic, to Classical Linear Logic, and more;
- beyond functional computation, to encompass concurrent and non-deterministic computation.

Why do this? We shall mention just one, fairly concrete motivation. Consider the well-established paradigm of extracting *functional programs* from (Intuitionistic or Classical) *proofs*, using the Curry-Howard isomorphism or realizability [GLT89, BS94]. Can we analogously find a suitable combination of a logic and a realizability universe such that we can extract interesting *concurrent programs*—communication protocols, distributed algorithms, security protocols—from proofs of their specifications?

Two important caveats should be registered here. The first is that we don't envisage the extraction of programs from proofs as a practical programming methodology. However, in the case of functional computation, the well-understood paradigm of program extraction from proofs is a key component of our foundational understanding of functional programming; the objective here is to attain a comparably well-founded paradigm for concurrent programming. The second caveat is that we don't—as yet—claim to be able to extract *interesting* concurrent programs, in the above sense, from proofs. However, we *do* see the ideas which we shall now put forward as a step in this direction.

Note to the reader This paper aims to give a readable and reasonably accessible account of some ideas linking the currently still largely separate worlds of concurrency theory and process algebra, on the one hand, and type theory, categorical models and realizability on the other. Background in process algebra may be found in standard texts such as [Hoa85, Hen88, Mil89, Ros97]; while background in realizability, categorical models etc. is provided by texts such as [GLT89, AL91, Cro93, AC98, BW99]. A modest background in either or both of these fields should suffice to understand the main ideas. Most of the detailed verification of properties of the formal definitions we will present is left as a series of exercises. The diligent reader who attempts a number of these should get some feeling for the interplay between concrete process-theoretic notions, and more abstract logical and categorical ideas, which is characteristic of this topic. It is this interplay which makes the topic a fascinating one for the author; I hope this brief introduction, to a field which is still wide open for further development, succeeds in conveying something of this fascination to the reader.

2 CCS with simultaneous actions

Our universe of realizers will be a minor extension of one of the most standard and widely-used process calculi, namely Milner's CCS [Mil89]. The extension is to allow "compound" actions, consisting of the simultaneous performance of several "atomic" actions. This idea of compound actions is present in the synchronous process calculus SCCS [Mil89]; the point here is to introduce this as an extension of the asynchronous calculus CCS. Our reason for using this extension is that it will allow us to realize identities and other canonical isomorphisms as "wires", with typical behaviour

$$\alpha \,\underline{\hspace{2em}}\, \beta$$

in which two signals are propagated simultaneously, at the two "ends of the wire", so to speak. This extension is not new; it was introduced in the present author's work on asynchronous interaction categories [Abr94a, AGN96], for similar reasons. Much of what we will do here can be seen as a recasting of the work on interaction categories into a realizability framework. Indeed, the essential ideas on the process interpretation of proofs go back to a 1991 lecture on "Proofs as Processes" (see [Abr94b]).

2.1 Names, co-names and actions

As usual with CCS, we introduce two disjoint, countable sets $\mathcal{N}$ of *names*, and $\bar{\mathcal{N}}$ of *co-names*, with a bijection $\bar{(\ )} : \mathcal{N} \xrightarrow{\cong} \bar{\mathcal{N}}$, which we extend to an involution

$$\bar{(\ )} : \mathcal{N} + \bar{\mathcal{N}} \xrightarrow{\cong} \bar{\mathcal{N}} + \mathcal{N}.$$

We use α, β, γ to range over names, and write $\bar{\alpha}$, $\bar{\beta}$, $\bar{\gamma}$ for the corresponding co-names, with $\bar{\bar{\alpha}} = \alpha$ etc. We write $\mathcal{L} = \mathcal{N} \cup \bar{\mathcal{N}}$ for the set of *labels*, ranged over by λ, μ, ν. An *action* will be a finite set of labels. We write Act for the set of actions, ranged over by a, b, c. In particular, we write τ for the empty set $\varnothing$. The interpretation of an action $a = \{\lambda_1, \ldots, \lambda_n\}$ is that the simultaneous performance of the actions in a is observed; thus τ can be viewed as a "silent" or "unobservable" or "internal" action. The involution $\bar{(\ )}$ is extended pointwise to actions:

$$\bar{a} = \{\bar{\lambda} \mid \lambda \in a\}.$$

Note that $\bar{\bar{a}} = a$, and that $\bar{\tau} = \tau$.

2.2 Guarded Terms

We now introduce a class of *guarded* process terms, with the following syntax.

$$P \ ::= \ a.P \mid \Sigma_{i \in I} a_i.P_i \ (\forall i \in I.\, a_i \neq \tau) \mid P \mid Q \mid G\backslash L \mid G[f] \mid X \mid \mathsf{rec}\, \mathsf{X}.\, \mathsf{P}.$$

Here I ranges over countable index sets, L ranges over subsets of $\mathcal{L}$ (*i.e.* *sorts*), X ranges over a set of process variables, and f over *renamings*, i.e. $\bar{(\ )}$-preserving injective functions on $\mathcal{L}$. (In fact, we shall allow *partial* injective functions as renamings, with the proviso that the *sort* of the process to which the renaming is applied is contained in the domain of the function. For details—which are easy and standard—see [Mil89]). Renamings are extended pointwise to actions. As usual, the empty sum is written as 0, and the binary case as $a.P + b.Q$.

2.3 Transitions

We define the labelled transitions $\xrightarrow{a}$, $(a \in \mathsf{Act})$ by the following inductive definition.

$$\frac{}{a.P \xrightarrow{a} P} \qquad\qquad \frac{}{\Sigma_{i\in I} a_i.P_i \xrightarrow{a_j} P_j \;\; (j \in I)}$$

$$\frac{P \xrightarrow{a} Q}{P\backslash L \xrightarrow{a} Q\backslash L} \;\; (a \cap (L \cup \bar{L}) = \varnothing) \qquad\qquad \frac{P \xrightarrow{a} Q}{P[f] \xrightarrow{f(a)} Q[f]}$$

$$\frac{P[\mathsf{rec\,X.\,P/X}] \xrightarrow{\mathsf{a}} \mathsf{Q}}{\mathsf{rec\,X.\,P} \xrightarrow{\mathsf{a}} \mathsf{Q}}$$

$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad\qquad \frac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'}$$

$$\frac{P \xrightarrow{a \dot{\cup} b} P' \quad Q \xrightarrow{\bar{b} \dot{\cup} c} Q'}{P \mid Q \xrightarrow{a \dot{\cup} c} P' \mid Q'}$$

Here $a \mathbin{\dot{\cup}} b$ means that a and b are disjoint; this, together with $b \mathbin{\dot{\cup}} c$ and $a \mathbin{\dot{\cup}} c$ should be viewed as *constraints* on the applicability of the rule. These rules are completely standard, except for the last, which generalizes the usual rule

$$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

(Take $a = c = \varnothing$, $b = \{\lambda\}$).

Note that we can define the wire $W_{\lambda,\mu}$ which repeatedly performs the compound action

$$\lambda \;\underline{\qquad}\; \beta$$

as the term

$$W_{\lambda,\mu} \;\stackrel{\Delta}{=}\; \mathsf{rec\,X.}\,\{\lambda, \mu\}.\mathsf{X}.$$

2.4 Failures Equivalence

Let Act_+ be the set of non-empty actions. We define the observable transition relations $\stackrel{a}{\Longrightarrow}$, for $a \in \mathsf{Act}_+$, as $\xrightarrow{\tau}^{\star}\xrightarrow{a}\xrightarrow{\tau}^{\star}$, and extend this to $\stackrel{s}{\Longrightarrow}$ for strings $s \in \mathsf{Act}_+^{\star}$ in the usual fashion. We define the set of *failures* of a process P by

$$\mathcal{F}(P) = \{(s, X) \mid \; s \in \mathsf{Act}_+^{\star} \;\wedge\; X \subseteq \mathsf{Act}_+ \;\wedge\; \exists Q.\,(P \stackrel{s}{\Longrightarrow} Q \;\wedge\; \forall a \in X.\, Q \stackrel{a}{\not\Longrightarrow})\}.$$

We define failures equivalence by

$$P \approx_f Q \;\; \stackrel{\Delta}{\Longleftrightarrow} \;\; \mathcal{F}(P) = \mathcal{F}(Q).$$

Proposition 2.1 *Failures equivalence is a congruence on guarded terms.*

Discussion Our reason for working with failures equivalence ([BHR84, BR83]) is that it, or one of its variants such as the testing equivalences of Hennessy and De Nicola [DNH83], or the Failures-Divergences model of Brookes and Roscoe [BR84], seem to be the finest equivalences which will suffice for our purposes. In particular, the realizability for the additive connectives of Linear Logic will not work in a fully satisfactory way if we use a finer equivalence such as weak congruence or weak bisimulation. It is worth noting that failures equivalence (or more accurately, the refined Failures-Divergences equivalence) is the standard equivalence for CSP, the other widely used process calculus.

Our reason for using guarded sums is to give a slightly simplified treatment of non-determinism. We could equally well have introduced the usual external and internal choice constructs as in [Hoa85, Hen88, Ros97].

The representation of processes in terms of their failures gives rise to a fully abstract model for failures equivalence, on which the process operations can be defined in a denotational, compositional fashion [BHR84, Ros97]. We have presented the semantics in an operational style to be concrete and simple, but in many ways the denotational presentation is more elegant and illuminating. It is also worth noting that we could equally well work with CSP as our process calculus rather than CCS, using the same underlying denotational model.

2.5 Some basic combinators

Since processes are untyped, we will build a type-free universe of realizers. As usual, this will require a little coding (*cf.* the Kleene algebra K_0 [Kle45], graph models [Sco76] etc.), but in our setting this will take a very simple form. We simply split the set of names into two infinite disjoint sets

$$\mathcal{N} = \mathcal{N}_l \mathbin{\dot\cup} \mathcal{N}_r$$

and fix bijections

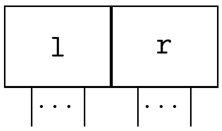
$$\mathtt{l} : \mathcal{N} \xrightarrow{\cong} \mathcal{N}_l \qquad \mathtt{r} : \mathcal{N} \xrightarrow{\cong} \mathcal{N}_r$$

and extend these in a $(\bar{\ })$-respecting way to the set of labels:

$$\mathcal{L} = \mathcal{L}_l \mathbin{\dot\cup} \mathcal{L}_r$$

$$\mathtt{l} : \mathcal{L} \xrightarrow{\cong} \mathcal{L}_l \qquad \mathtt{r} : \mathcal{L} \xrightarrow{\cong} \mathcal{L}_r.$$

This means that we can view an arbitrary process P as having its interface to its environment split into two disjoint parts:

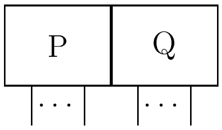


We can use this splitting of the name space to define some combinators which will play a fundamental role in our notion of process realizability.

Firstly, we have a tensor product which will express *disjoint (non-communicating) parallel composition*:

$$P \otimes Q \stackrel{\Delta}{=} P[\mathtt{l}] \mid Q[\mathtt{r}].$$

The use of the relabelling functions *forces* the two processes to be disjoint:

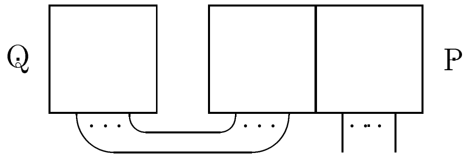


Secondly, we can define a notion of *application*. As usual in a type-free setting, we have to be able to see arbitrary elements of our universe either as "functions" or as "arguments", as required. The basic splitting of our name space allows us to see P as a "function" in *two, entirely symmetrical ways.* We can see the left part of the name space as the "attachment point" for an argument, with the right part left free to communicate the "result"; or we can attach on the right and transmit the result through the left.

The first view leads to a *left* (or *forwards*) application:

$$\langle Q|P \ \triangleq\ ((Q[\mathtt{l}] \mid P)\backslash\mathcal{L}_l)[\mathtt{r}^{-1}]$$

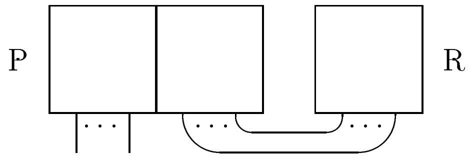
which we can visualize as follows:



We use a Dirac-style bra-ket notation (*cf.* [Dir67]) to denote the left application of P to Q. The idea is that Q is relabelled into the left part of P's name space, and we then restrict on the left name space so that P and Q are forced to interact there; the resulting observable behaviour is purely that produced by P in the right part of its name space. Finally, we "normalize" by relabelling back into the global name space, using the inverse of the bijection $\mathtt{r}$.

Symmetrically, we can define a *right* (or *reverse*) application:

$$P|R\rangle \ \triangleq\ ((P \mid R[\mathtt{r}])\backslash\mathcal{L}_r)[\mathtt{l}^{-1}]$$



3 Realizability

We will now define a notion of realizability for formulas built from the propositional connectives of (Classical) Linear Logic: the multiplicatives $\otimes$ and $⅋$, the additives & and $\oplus$, and the exponentials ! and ?. More precisely, we shall define two relations

$$P \Vdash_+ A \quad \text{and} \quad P \Vdash_- A$$

between process terms P and formulas A, by induction on the construction of A. We shall read $P \Vdash_+ A$ as: "P is a realizer/strategy/proof of A", or "P is a value of type A"; and $P \Vdash_- A$ as: "P is a counter-realizer/counter-strategy/refutation of A", or "P is an A-consuming context". This builds a classical (involutive) duality, and specifically an interpretation of the Linear negation $(\_)^\perp$, into our realizability interpretation. In particular, we will have:

$$\begin{aligned} P \Vdash_+ A^\perp \ &\equiv\ P \Vdash_- A \\ P \Vdash_- A^\perp \ &\equiv\ P \Vdash_+ A. \end{aligned}$$

3.1 Multiplicatives

For each connective, we must define both the positive and negative notions of realizability. However, once this is done for one connective, the notions for the de Morgan duals are also determined.
The definitions for the tensor product are as follows:

$$P \Vdash_+ A \otimes B \;\equiv\; P \approx_f P_1 \otimes P_2 \;\wedge\; P_1 \Vdash_+ A \;\wedge\; P_2 \Vdash_+ B.$$

Note that, because of the disjoint relabelling in the definition of the tensor combinator on processes:

$$P_1 \otimes P_2 \approx_f Q_1 \otimes Q_2 \implies P_1 \approx_f Q_1 \;\wedge\; P_2 \approx_f Q_2$$

so the decomposition in the above clause is unique up to failures equivalence (in fact, up to weak bisimulation).

$$P \Vdash_- A \otimes B \;\equiv\; \forall Q.\, (Q \Vdash_+ A \Rightarrow \langle Q|P \Vdash_- B) \;\wedge\; \forall R.\, (R \Vdash_+ B \Rightarrow P|R\rangle \Vdash_- A).$$

This is a symmetrized version of the familiar "logical relations" or realizability condition. P counter-realizes the multiplicative conjunction $A \otimes B$ if it carries every realizer of A, under forwards application, to a counter-realizer for B, and every realizer of B, under reverse application, to a counter-realizer for A.
Applying de Morgan duality, this yields the more familiar-looking definition for linear implication:

$$A \multimap B \;\stackrel{\Delta}{=}\; A^{\perp} \invamp B \;\stackrel{\Delta}{=}\; (A \otimes B^{\perp})^{\perp}.$$

$$P \Vdash_+ A \multimap B \;\equiv\; \forall Q.\, (Q \Vdash_+ A \Rightarrow \langle Q|P \Vdash_+ B) \;\wedge\; \forall R.\, (R \Vdash_- B \Rightarrow P|R\rangle \Vdash_- A).$$

P realizes the linear implication $A \multimap B$ if it carries realizers of A to realizers of B, and counter-realizers of B to counter-realizers of A.

The reading of the clause for negative realizability for the linear implication is also interesting:

$$P \Vdash_- A \multimap B \;\equiv\; P \approx_f Q \otimes R \;\wedge\; Q \Vdash_+ A \;\wedge\; R \Vdash_- B.$$

This can be read as saying that P realizes a context for consuming a "linear function" f of type $A \multimap B$ if it decomposes as an input of type A to be plugged into f, and a context of type B for consuming the corresponding output.

It is interesting to note that the combinators $\otimes$, $\langle \cdot | \cdot$ and $\cdot | \cdot \rangle$ which we introduced in order to realize the multiplicative connectives are defined purely in terms of the *static* operators of CCS in Milner's classification [Mil89], namely parallel composition, restriction and relabelling. As we shall now see, the additive connectives will be realized using only the *dynamic* operators of CCS (prefixing and summation), while the exponentials will require a combination of the two, together with recursion.

3.2 Additives

Firstly, we fix once and for all two distinct names, say α and β. These will be used to distinguish the left and right cases in the additive choice constructs $A \& B$ and $A \oplus B$. For the additive product or conjunction $A \& B$, the Opponent or Environment will make the choice, and the realizers for $A \& B$ will have the form $\alpha.P + \beta.Q$, where P is a realizer for A and Q is a realizer for B. For the additive sum or disjunction $A \oplus B$, Player or System makes the choice, and realizers for $A \oplus B$ either have the form $\bar{\alpha}.R$, where R is a realizer for A, or $\bar{\beta}.S$, where S is a realizer for B.

This leads to the following formal definitions:

$$\begin{array}{lcl}
P \Vdash_+ A\&B & \equiv & P \approx_f \alpha.Q + \beta.R \ \wedge \ Q \Vdash_+ A \ \wedge \ R \Vdash_+ B \\
P \Vdash_- A\&B & \equiv & (P \approx_f \bar{\alpha}.Q \ \wedge \ Q \Vdash_- A) \vee (P \approx_f \bar{\beta}.R \ \wedge \ R \Vdash_- B) \\
P \Vdash_+ A \oplus B & \equiv & (P \approx_f \bar{\alpha}.Q \ \wedge \ Q \Vdash_+ A) \vee (P \approx_f \bar{\beta}.R \ \wedge \ R \Vdash_+ B) \\
P \Vdash_- A \oplus B & \equiv & P \approx_f \alpha.Q + \beta.R \ \wedge \ Q \Vdash_- A \ \wedge \ R \Vdash_- B.
\end{array}$$

3.3 Exponentials

As already indicated, to interpret the exponentials we combine additive and multiplicative features with recursion on processes, and also induction to define the realizability relation. This use of recursion and induction will be set in a more general context in our later discussion of inductive and coinductive types.
As for the additives, we fix some global names: ω (for weakening), δ (dereliction) and γ (contraction). We define a process combinator

$$!P \ \stackrel{\Delta}{=} \ \mathsf{rec}\,\mathsf{X}.\,\omega.\mathsf{0} + \delta.\mathsf{P} + \gamma.(\mathsf{X}[\mathtt{l}] \mid \mathsf{X}[\mathtt{r}]).$$

We can then define positive realizability for !:

$$P \Vdash_+ !A \ \equiv \ P \approx_f !Q \ \wedge \ Q \Vdash_+ A.$$

We can think of $!P$ as a process which the environment can request to:

- deliver one copy of P (dereliction)
- be discarded (weakening)
- make two copies of itself (contraction).

The negative realizability for $!A$ can be defined as follows.

$$\begin{array}{lcl}
P \Vdash_- !A \equiv & & (P \approx_f \omega.0) \vee (P \approx_f \bar{\delta}.Q \ \wedge \ Q \Vdash_- A) \\
& & \vee \ (P \approx_f \bar{\gamma}.Q \ \wedge \ (\forall R \Vdash_+ A.\, \langle\, !R \,|\, Q \Vdash_- !A \ \wedge \ Q \,|\, !R \rangle \Vdash_- !A)).
\end{array}$$

Note that for the first time, the realizability relation is not being defined purely by structural induction on the formula. Rather, $P \Vdash_- !A$ is being defined inductively, as the least fixed point of the evident monotone operator on sets of processes which can be extracted from the above definition, keeping $!A$ fixed. Note, however, that the universal quantifier $\forall R \Vdash_+ A$ is ranging over realizers for A, which we *can* take to be already defined by structural induction on $!A$. This is crucial for monotonicity.

The realizability relation for the dual connective ? is defined by De Morgan duality from that for !, since $?A = (!A^{\perp})^{\perp}$. To understand the inductive definition, think of it as defining the set of all $!A$-consuming contexts, which request a number of copies of a realizer for A, and then consume these copies. Recall that $?A$ is the "$\invamp$-monoid generated by A", just as $!A$ is the "$\otimes$-comonoid cogenerated by A". Thus the inductive clause for contraction parallels that for Par, just as the case for contraction in the recursive definition of $!P$ parallels that for Tensor. This also says that there is no communication between the copies of P in $!P$, while the counter-realizers for $!A$ can use the information obtained from each copy in interacting with the others, as for Par.

3.4 Interpretation of Proofs

We now show how to assign a realizer for A to each proof of A in Linear Logic. We will then be able to extract concurrent processes as realizers from proofs.

We extend realizability to sequents $\Gamma = A_1, \ldots, A_k$, treating Γ as $\invamp \Gamma$ in the obvious fashion. We now indicate how to assign realizers to sequent proofs in Linear Logic.

3.4.1 Identity Axioms

$$\overline{\vdash A^{\perp}, A}$$

All instances of the Identity Axioms are realized by the process

$$I \ \triangleq\ \mathsf{rec}\,\mathsf{X}.\ \sum_{\mathsf{a}\in\mathsf{Act}_{+}} (\mathtt{l}(\mathsf{a}) \cup \overline{\mathtt{r}(\mathsf{a})}).\mathsf{X}$$

This behaves like a wire for any choice of actions

$$a \underline{\qquad\quad} \bar{a}$$

where the action at the left hand end of the wire corresponds to $\mathtt{l}(a)$, and that on the right to $\overline{\mathtt{r}(a)} = \mathtt{r}(\bar{a})$. To check that I is indeed a realizer for $A^{\perp} \invamp A = A \multimap A$ amounts to verifying the process-algebraic fact that

$$\forall P.\, \langle P|I \approx_f P \approx_f I|P\rangle.$$

Exercise Verify!

3.4.2 Cut Rule

$$\frac{P \Vdash_{+} \Gamma, A \qquad Q \Vdash_{+} A^{\perp}, \Delta}{P; Q \Vdash_{+} \Gamma, \Delta}$$

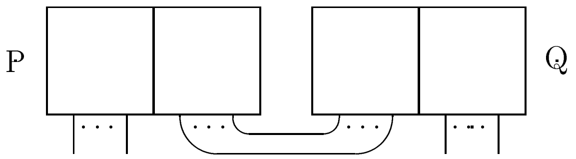
Here we are inductively assuming that we have already assigned a (positive) realizer P to the proof of Γ, A, and a realizer Q to the proof of $A^{\perp}, \Delta$. We must construct a realizer $P; Q$ for Γ, Δ. This composition combinator will simultaneously generalize the left and right application combinators we have previously introduced.
We fix a three-fold decomposition of the name space:

$$\mathcal{N} = \mathcal{N}_1 \,\dot{\cup}\, \mathcal{N}_2 \,\dot{\cup}\, \mathcal{N}_3$$

with bijections $\phi_{i,j} : \mathcal{N} \xrightarrow{\cong} \mathcal{N}_i \,\dot{\cup}\, \mathcal{N}_j$ such that $\phi_{i,j}(\mathcal{N}_l) = \mathcal{N}_i$, $\phi_{i,j}(\mathcal{N}_r) = \mathcal{N}_j$. We can then define

$$P; Q \ \triangleq\ ((P[\phi_{1,2}] \mid Q[\phi_{2,3}])\backslash\mathcal{N}_2)[\phi^{-1}_{1,3}].$$

The picture is



The key process-algebraic facts are

$$\begin{array}{lll} (P;Q);R & \approx_f & P;(Q;R) \\ P;I & \approx_f & P \\ I;P & \approx_f & P \\ \langle P|(Q;R) & \approx_f & \langle\langle P|Q|R \\ (P;Q)|R\rangle & \approx_f & P|Q|R\rangle\rangle \end{array}$$

Exercise Verify!
From these, the fact that $P;Q$ realizes Γ, Δ follows easily. For example, suppose that $R \Vdash_- \Gamma$. We must show that $\langle R|P;Q \Vdash_+ \Delta$. But

$$P \Vdash_+ \Gamma, A \implies \langle R|P \Vdash_+ A$$

$$Q \Vdash_+ A^\perp, \Delta \implies \langle\langle R|P|Q \Vdash_+ \Delta$$

and the result follows since

$$\langle R|P;Q \approx_f \langle\langle R|P|Q.$$

This last step uses the fact that, as already stated, we are taking processes modulo failures equivalence as our realizers. In fact, all the equivalences stated so far are valid with respect to much finer notions, in particular for weak bisimulation.

3.4.3 Multiplicatives

The construction of realizers for the multiplicative rules requires nothing more than renaming.

$$\frac{P \Vdash_+ \Gamma, A \quad Q \Vdash_+ \Delta, B}{P \otimes Q[\phi] \Vdash_+ \Gamma, \Delta, A \otimes B} \qquad \frac{P \Vdash_+ \Gamma, A, B}{P[\psi] \Vdash_+ \Gamma, A \parr B}$$

for suitable renamings ϕ, ψ.

Exercise Define ϕ and ψ, and verify that the required realizability relations do hold.

3.4.4 Additives

$$\frac{P \Vdash_+ \Gamma, A \quad Q \Vdash_+ \Gamma, B}{\mathtt{r}(\alpha).P + \mathtt{r}(\beta).Q \Vdash_+ \Gamma, A \& B}$$

$$\frac{P \Vdash_+ \Gamma, A}{\overline{\mathtt{r}(\alpha)}.P \Vdash_+ \Gamma, A \oplus B} \qquad \frac{Q \Vdash_+ \Delta, B}{\overline{\mathtt{r}(\beta)}.Q \Vdash_+ \Delta, A \oplus B}.$$

We can define the process combinators

$$\langle P, Q\rangle \triangleq \mathtt{r}(\alpha).P + \mathtt{l}(\beta).Q.$$

$$\mathtt{l}(P) \triangleq \overline{\mathtt{l}(\alpha)}.P \quad \mathtt{r}(Q) \triangleq \overline{\mathtt{r}(\beta)}.Q.$$

The key process algebraic facts we need to show the soundness of the above rules are

$$\begin{array}{lll} \langle P, Q\rangle; \mathtt{l}(R) & \approx_f & P;R \\ \langle P, Q\rangle; \mathtt{r}(S) & \approx_f & Q;S \\ \langle R|\langle P, Q\rangle & \approx_f & \langle\langle R|P, \langle R|Q\rangle. \end{array}$$

Exercise Verify these equations. Show that the first two hold with respect to weak bisimulation, but the third does not. Validating this last equation is one main reason for working with failures equivalence in this paper. (Those familiar with the failures-divergences model will note that this equation only holds in that model on the assumption that R is not the immediately divergent process. It is possible to adapt our treatment to accomodate divergences. We have not done so here to simplify the presentation.)

Exercise Using the above equations, show the soundness of the realizability assignments for the additive rules.

Exercise Work out the realizability assignments for the exponentials ! and ? of Linear Logic, and prove their soundness.

3.5 Cut-Elimination

Thus for each sequent proof Π in Linear Logic, we can assign a process term $[\![\Pi]\!]$. Moreover, we have the following result expressing the soundness of our assignment with respect to Cut Elimination.

Proposition 3.1 *If* Π *reduces to* Π' *under cut-elimination, then* $[\![\Pi]\!] \approx_f [\![\Pi']\!]$.

By virtue of these results, we can claim to have modelled Cut-elimination by process interaction. We can also prove that Cut-elimination *terminates* in our model. We outline the argument. For a detailed account in the interaction categories setting, see [AGN99].

Firstly, we say that a process P *diverges*, written $P\Uparrow$, if there is a sequence $(P_n \mid n \in \omega)$ with $P = P_0$ and $P_n \xrightarrow{\tau} P_{n+1}$ for all $n \in \omega$. We say that P is *convergent*, written $P\Downarrow$, if it does not diverge.
Now we define

$$P \perp Q \;\; \stackrel{\Delta}{\Longleftrightarrow} \;\; ((P \mid Q)\backslash\mathcal{N})\Downarrow.$$

Thus if we "close the system", so that P and Q can only interact with each other, there must be no possible divergences.
We say that a formula A is *total* if

$$\forall P \Vdash_{+} A.\, \forall Q \Vdash_{-} A.\; P \perp Q,$$

and that it is *inhabited* if it has both positive and negative realizers. Note that a realizer for a total and inhabited type is convergent.

Proposition 3.2 *If A and B are total and inhabited, then so are $A^{\perp}$, $A \otimes B$, $A \& B$, $!A$ (and hence also the other connectives).*

This means that if we start from total and inhabited interpretations of the atomic formulas, then the process we extract from any proof will be convergent. The key case is Cut, where the totality condition plays an analogous role to the computability predicate in a Tait-style proof of strong normalization [Tai67, GLT89]. The fact that our process realizers are convergent is analogous to the fact that a functional realizer extracted by standard realizability from a proof say in second-order logic will compute a total functional.

4 The Realizability Category

We now turn to a more semantic view of process realizability, in the same general spirit as the by now standard idea of constructing categories of assemblies or realizability toposes (for which see *e.g.* [AL91, Cro93, Lon95, AC98]).

For each formula A, we can define

$$\begin{array}{rcl} S_A & = & \{P \mid P \Vdash_+ A\} \\ S^*_A & = & \{Q \mid Q \Vdash_- A\} \end{array}$$

We can then define the "realizability semantics" for a formula A as

$$[\![A]\!] = (S_A, S^*_A).$$

The advantage of this point of view is that we can now abstract to consider *any* pair (S, S^*) of sets of processes as a "type" of realizers and counter-realizers, and define the action of the various Linear connectives over these types. Thus for example

$$(S, S^*)^\perp = (S^*, S)$$

while $(S, S^*) \multimap (T, T^*)$ is the pair

$$(\{P \mid \forall Q \in S. \langle Q|P \in T \;\wedge\; \forall R \in T^*. P|R\rangle \in S^*\}, \{P \otimes Q \mid P \in S \;\wedge\; Q \in T^*\})$$

etc. This idea needs to be refined slightly to yield a satisfactory result. In particular, in order to build in a suitable "modulus of extensionality", we shall work with *partial equivalence relations* on processes, rather than simply sets of processes. Recall that a partial equivalence relation is a symmetric, transitive relation.

We shall take a type to be a pair (E, E^*) of partial equivalence relations on processes, where processes are identified up to failures equivalence. The interpretations of the linear connectives lift to operations on partial equivalence relations in a straightforward manner. For example, $E_{A \multimap B}$ consists of all pairs (P, Q) such that

$$\forall (R, S) \in E_A. (\langle R|P, \langle S|Q) \in E_B \;\wedge\; \forall (T, U) \in E^*_B. (P|T\rangle, Q|U\rangle) \in E^*_A.$$

We proceed to define a realizability category $\mathcal{C}$. The objects

$$A = (E_A, E^*_A)$$

are pairs of partial equivalence relations on processes. A morphism $f : A \to B$ is a partial equivalence class $[P]$ of $E_{A \multimap B}$. This partial equivalence class induces a pair of maps (f^+, f^-), where f maps partial equivalence classes of E_A to partial equivalence classes of E_B, and f^- maps partial equivalence classes of E^*_B to partial equivalence classes of E^*_A:

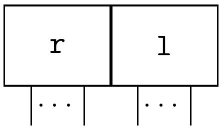
$$f^+([Q]) = [\langle Q|P], \quad f^-([R]) = [P|R\rangle].$$

Conversely, any such pair of maps which is "tracked" by a process P in this way determines a unique partial equivalence class of $E_{A \multimap B}$.

The further structure of the category unfolds as essentially a recapitulation of our account of realizability for linear proofs. For example, identities and composition are realized as for Axiom and Cut. Our constructions for $\otimes$ and $\multimap$ give $\mathcal{C}$ the structure of a symmetric monoidal closed category. There is a duality

$$\frac{f : A \to B}{f^\perp : B^\perp \to A^\perp}$$

where if $f = (f^+, f^-)$, $f^\perp = (f^-, f^+)$. At the process level, the duality just amounts to interchanging the left-right partition of the name space



The additive connectives &, $\oplus$ give products and coproducts in $\mathcal{C}$, and in fact $\mathcal{C}$ has all (countable) limits and colimits. $!A$ gives the cofree cocommutative comonoid on A.

Proposition 4.1 *$\mathcal{C}$ is a model of Linear Logic.*

Exercise Verify some of this structure. For example, show that & gives the categorical product in $\mathcal{C}$.

4.1 Quantifiers

This construction is easily extended to yield a model of second-order Linear Logic.

A formula $A[X]$ with a second-order propositional variable X can be interpreted as a function F_A on the objects of $\mathcal{C}$ in the obvious fashion. We then define

$$E_{\forall X.\,A[X]} = \bigcap\{E_{F_A(B)} \mid B \in \mathsf{Ob}\,\mathcal{C}\}$$

$$E^*_{\forall X.\,A[X]} = (\bigcup\{E^*_{F_A(B)} \mid B \in \mathsf{Ob}\,\mathcal{C}\})^+.$$

The clause for the negative realizers is really for the second-order existential:

$$(\forall X.\,A[X])^\perp = \exists X.\,A[X]^\perp.$$

The transitive closure is used, since the union of a family of per's need not be transitive. This "information loss" is typical of the behaviour of second-order existentials.

First-order quantifiers are handled nicely by value-passing at the process algebra level. We take the set $\mathcal{V}$ of values to be the domain of quantification. The realizability definitions are similar to those for the additives. We fix an atomic name σ.

$$P \Vdash_+ \forall x.\,A \;\equiv\; P \approx_f \sigma x.\,Q \;\wedge\; \forall v \in \mathcal{V}.\,Q[v/x] \Vdash_+ A[v/x]$$

$$P \Vdash_- \forall x.\,A \;\equiv\; P \approx_f \bar{\sigma} v.R \;\wedge\; R \Vdash_- A[v/x].$$

In terms of operations on partial equivalence relations:

$$E_{\forall x.\,A} = \{(\sigma x.P, \sigma x.Q) \mid \forall v \in \mathcal{V}.\,(P[v/x], Q[v/x]) \in E_{A[v/x]}\}$$

$$E^*_{\forall x.\,A} = \{(\bar{\sigma} v.P, \bar{\sigma} v.Q) \mid (P, Q) \in E^*_{A[v/x]}\}.$$

4.2 Inductive and Co-inductive types

Inductive and coinductive types can be canonically interpreted in $\mathcal{C}$ as initial T-algebras and final T-coalgebras for endofuctors $T : \mathcal{C} \to \mathcal{C}$. We look at a basic example by way of illustration. Firstly, we define a unit type I with

$$E_I = E^*_I = \{(0, 0)\}.$$

We fix a type A, and define a type of A-lists by

$$L = \mu X.\,I \oplus (A \otimes X).$$

What are the realizers for L? The empty list is realized by $\bar{\alpha}.0$. Given a realizer P for A, which may be taken as realizing the "value" $v = [P]$, the unit list $[v]$ is realized by $\bar{\beta}.(P \otimes \bar{\alpha}.0)$. Inductively, if an A-list l is realized by Q, and an A-value v is realized by P, then the list $v :: l$ is realized by $\bar{\beta}.(P \otimes Q)$. Thus we get an inductive definition of the positive realizers for L, which may be compared to that for $?A$. It is worth noting that these lists are truly *linear*—to "read" them (by interacting with the process realizing the list) is to consume them.
What of the counter-realizers for L? These will be the A-list consuming contexts. Such a context can be defined as follows. Let $(Q_n \mid n \in \omega)$ be a family of processes, with

$$Q_n \Vdash_{-} \underbrace{A \otimes \cdots \otimes A}_{n}, \quad (n \in \omega).$$

Define a family $(P_n \mid n \in \omega)$ by simultaneous recursion:

$$P_i = \alpha.Q_i + \beta.P_{i+1} \quad (i \in \omega).$$

Then P_0 is a counter-realizer for L.

Exercise Work out the details of this example, to give an explicit description of the initial T-algebra for the endofunctor

$$TX = I \oplus (A \otimes X).$$

Exercise Similarly, analyze the coinductive type of A-streams:

$$S = \nu X.\, I \,\&\, (A \otimes X).$$

We can use the example of lists to give a useful intuition for the symmetric condition on realizers for the linear implication. Consider a morphism

$$f : \mathsf{List(A)} \to \mathsf{List(B)}.$$

This means that we have a realizer P for a pair of maps (f^{+}, f^{-}). In the forwards direction, P induces a function f^{+} mapping A-lists to B-lists. In the backwards direction, P induces a *context-transformer* f^{-} mapping contexts whch consume B-lists to contexts which consume A-lists. E.g. given the definition

$$\begin{array}{lcl} \mathtt{f}\,[\,] & = & [\,] \\ \mathtt{f}\ \mathsf{a} :: \mathsf{b} :: \mathtt{xs} & = & (a+b) :: xs \end{array}$$

we have the usual function

$$f^{+}([1,2]) = [3]$$

and also

$$f^{-}(\mathsf{hd}([\cdot]) = \mathsf{hd}(\mathsf{hd}[\cdot]).$$

The context-transformer part of the interpretation of f in our realizability semantics is *intensional* information about the behaviour of f as an *algorithm*, rather than merely a set-theoretical function. This opens up the possibility of accurate realizability models for non-functional languages.

References

[Abr94a] S. Abramsky. Interaction Categories and Communicating Sequential Processes. In *A Classical Mind. Essays in Honour of C. A. R. Hoare*, ed. A. W. Roscoe, 1–16. Prentice Hall 1994.

[Abr94b] S. Abramsky. Proofs as Processes. *TCS* vol. 135, 5–9, 1994.

[AGN96] S. Abramsky, S. J. Gay and R. Nagarajan. Interaction Categories and the Foundations of Typed Concurrent Programming. In M. Broy, ed. *Deductive Program Design: Proceedings of the 1994 Marktoberdorf Summer School*, 35–113. Springer, 1996.

[AGN99] S. Abramsky, S. J. Gay and R. Nagarajan. A specification structure for deadlock-freedom of synchronous processes. *TCS*, vol. 222, 1–53, 1999.

[AC98] R. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.

[AL91] A. Asperti and G. Longo. *Categories, Types and Structures*. MIT Press, 1991.

[BW99] M. Barr and C. Wells. *Category Theory for Computing Science. Third Edition*. Les Publications CRM, Montreal, 1999.

[BR83] S. Brookes and W. Rounds. Behavioural Equivalences induced by Programming Logics. In *Proceedings of ICALP '83*, Springer Lecture Notes in Computer Science vol. 154, 97–108, 1983.

[BHR84] S. Brookes, C. A. R. Hoare and A. W. Roscoe. A Theory of Communicating Sequential Processes. *JACM* vol. 31, no. 7, 560–599, 1984.

[BR84] S. Brookes and A. W. Roscoe. An Improved Failures Model for Communicating Processes. In Springer Lecture Notes in Computer Science vol. 197, 281–305, 1984.

[Cro93] R. L. Crole. *Categories for Types*. Cambridge University Press, 1993.

[DNH83] R. DeNicola and M. Hennessy. Testing Equivalence for Processes. *TCS* vol. 34, 83–133, 1983.

[Dir67] P. A. M. Dirac. *Principles of Quantum Mechanics*. Oxford University Press, 1967.

[GLT89] J.-Y. Girard, Y. Lafont and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Kle45] S. C. Kleene. On the interpretation of intuitionistic number theory. *JSL* vol. 10, 1945.

[Lon95] J. R. Longley. *Realizability Toposes and Language Semantics*. Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1995.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[BS94] U. Berger and H. Schwichtenberg. Program extraction from classical proofs. In D. Leivant, ed. *Logic and Computational Complexity*, Springer Lecture Notes in Computer Science vol. 960, 77–97, 1995.

[Sco76] D. S. Scott. Data Types as Lattices. *SIAM J. on Computing*, vol. 5, 522–587, 1976.

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I. *JSL* vol. 32, 198–212, 1967.