Lecture Notes on
Semi-Axiomatic Sequent Calculus

15-836: Substructural Logics
Frank Pfenning

Lecture 14
October 26, 2023

1 Introduction

Message-passing communication in MPASS, which is based on the linear sequent calculus, is *synchronous* in the sense that both sending and receiving are potentially blocking actions. This is often a convenient abstraction, but under the hood communication is usually *asynchronous* in the sense that sending does not block but receiving does. In other formalisms for concurrency such as the π-calculus, we have synchronous [Milner et al., 1992] and asynchronous [Boudol, 1992] versions. So it is natural to look for an *asynchronous* calculus based on the interpretation of linear propositions as session types. It turns out that such a calculus exists and uncovers several new connections, in particular to *futures* in functional programming languages [Halstead, 1985] that are usually thought of as a form of shared memory concurrency. In this lecture we will develop an asynchronous message-passing calculus.

We know that in the untyped setting, the synchronous π-calculus is more expressive than the asynchronous one [Palamidessi, 2003]. In the setting of session types, they turn out to have the same expressive power [Pfenning and Griffith, 2015], so we have to decide which formulation we would like to take as fundamental. Because of its (relative) proximity to an implementation and its connection to futures, we prefer the asynchronous version as long as we can still relate it to proof theory. It turns out that going down this path will also allow us to generalize from linear to structural and then general adjoint types, which seems difficult to do directly for the synchronous version.

2 The Origin of Synchronous Communication

We refresh our memory about synchronous communication in MPASS. We use internal choice as an example. When the provider sends a label k, the type of the channel changes from $\oplus\{\ell : A_\ell\}_{\ell\in L}$ to A_k.

$$\begin{array}{lll} & \mathsf{proc}(\mathbf{send}\ a\ k\ ;\ P), & \mathsf{proc}(\mathbf{recv}\ a\ (\ell \Rightarrow Q_\ell)_{\ell\in L}) \\ \longrightarrow & \mathsf{proc}(P) & \mathsf{proc}(Q_k) \qquad\qquad (k \in L) \end{array}$$

This communication is synchronous because sender and receive proceed to their respective continuations at once. This ultimately comes from the linear sequent calculus where the principal cases of cut reduction replace a cut of proposition $A \oplus B$ either by a cut of A or of B, with subderivations on both premises of the cut.

Let's also recall the typing rules where the change in type of the communication channel is clearly visible.

$$\frac{k \in L \quad \Delta \vdash P :: (x : A_k)}{\Delta \vdash \mathbf{send}\ x\ k\ ;\ P :: (x : \oplus\{\ell : A_\ell\}_{\ell\in L})}\ \oplus R$$

$$\frac{\Delta, x : A_\ell \vdash Q_\ell \quad (\forall \ell \in L)}{\Delta, x : \oplus\{\ell : A_\ell\}_{\ell\in L} \vdash \mathbf{recv}\ k\ (\ell \Rightarrow Q_\ell)_{\ell\in L} :: (z : C)}\ \oplus L$$

We confirm that the synchronous nature of communication directly derives from the synchronous nature of cut reduction. Writing out a binary case (as is customary in logic):

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}' \\ \Delta \vdash A\end{array}}{\Delta \vdash A \oplus B}\ \oplus R_1 \qquad \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ \Delta', A \vdash C & \Delta', B \vdash C\end{array}}{\Delta', A \oplus B \vdash C}\ \oplus L}{\Delta, \Delta' \vdash C}\ \mathsf{cut}_{A\oplus B}$$

$$\longrightarrow \qquad \dfrac{\begin{array}{cc}\mathcal{D}' & \mathcal{E}_1 \\ \Delta \vdash A & \Delta', A \vdash C\end{array}}{\Delta, \Delta' \vdash C}\ \mathsf{cut}_A$$

3 Continuation Channels instead of Continuation Processes

We cannot simply drop the continuation process P to make communication asynchronous, because messages could be received out of order and progress would be violated. For example:

$\mathsf{bin} = \oplus\{\mathsf{b0} : \mathsf{bin}, \mathsf{b1} : \mathsf{bin}, \mathsf{e} : \mathbf{1}\}$
$\mathit{one}\ (x : \mathsf{bin}) = \mathbf{send}\ x\ \mathsf{b1}\ ;\ \mathbf{send}\ x\ \mathsf{e}\ ;\ \mathbf{send}\ x\ (\,)$

If we now match up a process that sends a binary 1 and one that receives a binary number, all several kinds of mismatches can occur.

$$\begin{array}{l}\mathsf{proc}(\mathbf{call}\ \mathit{one}\ a), \mathsf{proc}(\mathbf{recv}\ a\ (\mathsf{b0} \Rightarrow Q_0 \mid \mathsf{b1} \Rightarrow Q_1 \mid \mathsf{e} \Rightarrow Q_e) \\ \longrightarrow^* \mathsf{proc}(\mathbf{send}\ a\ \mathsf{b1}), \mathsf{proc}(\mathbf{send}\ a\ \mathsf{e}), \mathsf{proc}(\mathbf{send}\ a\ (\,)), \\ \qquad \mathsf{proc}(\mathbf{recv}\ a\ (\mathsf{b0} \Rightarrow Q_0 \mid \mathsf{b1} \Rightarrow Q_1 \mid \mathsf{e} \Rightarrow Q_e))\end{array}$$

Each of the messages could interact with the receiver, which could be an immediate "message not understood" problem (when the message is $(\,)$), or a later one (when the message is e).

The way we solve this problem is to replace the *continuation process* of the sender by a *continuation channel*. Ignoring for the moment where these continuation channels would come from, we might write

$$\begin{array}{l}\mathsf{bin} = \oplus\{\mathsf{b0} : \mathsf{bin}, \mathsf{b1} : \mathsf{bin}, \mathsf{e} : \mathbf{1}\} \\ \mathit{one}\ (x : \mathsf{bin}) = \mathbf{send}\ x\ \mathsf{b1}(x')\ ;\ \mathbf{send}\ x'\ \mathsf{e}(x'')\ ;\ \mathbf{send}\ x''\ (\,) \qquad \text{\% approximately}\end{array}$$

We need to fix this later to account for the creation of the continuation channels. The receiver then not only selects the branch, but also receives a continuation channel for further communication.

$$\mathsf{proc}(\mathbf{recv}\ x\ (\mathsf{b0}(x') \Rightarrow Q_0(x') \mid \mathsf{b1}(x') \Rightarrow Q_1(x') \mid \mathsf{e}(x') \Rightarrow Q_e(x'))$$

The idea is that x is used in only one place, and then x' next, and then x'', etc. so channels and their types cannot be confused. This technique is due to Kobayashi et al. [1996].

Revisiting our rules, they now become:

$$\frac{k \in L}{x' : A_k \vdash \mathbf{send}\ x\ k(x') :: (x : \oplus\{\ell : A_\ell\}_{\ell \in L})}\ \oplus X$$

$$\frac{\Delta, x' : A_\ell \vdash Q_\ell(x') \quad (\forall \ell \in L)}{\Delta, x : \oplus\{\ell : A_\ell\}_{\ell \in L} \vdash \mathbf{recv}\ k\ (\ell(x') \Rightarrow Q_\ell(x'))_{\ell \in L} :: (z : C)}\ \oplus L$$

Note that the right rule has become an axiom, that is, it has no logical premises. This makes sense intuitively because when a message is received it should be consumed by the recipient. The left rule only changes in the sense that x in the premises has become the continuation channel x'.

In the reduction rule we see that the channel a no longer changes type, but communication is transferred from $a : \oplus\{\ell : A_\ell\}_{\ell \in L}$ to the continuation channel $a' : A_k$.

$$\begin{array}{rll} & \mathsf{proc}(\mathbf{send}\ a\ k(a')), & \mathsf{proc}(\mathbf{recv}\ a\ (\ell(x') \Rightarrow Q_\ell(x'))_{\ell \in L}) \\ \longrightarrow & & \mathsf{proc}(Q_k(a')) \qquad (k \in L)\end{array}$$

4 Back to Logic

We have seen that the dynamics, if it holds up to scrutiny in the logic, would allow for asynchronous communication. Extracting the binary logical rules we have:

$$\frac{}{A \vdash A \oplus B}\ \oplus X_1 \qquad \frac{}{B \vdash A \oplus B}\ \oplus X_2 \qquad\qquad \frac{\Delta, A \vdash C \quad \Delta, B \vdash C}{\Delta, A \oplus B \vdash C}\ \oplus L$$

The reductions:

$$\frac{\dfrac{}{A \vdash A \oplus B}\ \oplus X_1 \qquad \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ \Delta', A \vdash C & \Delta', B \vdash C\end{array}}{\Delta', A \oplus B \vdash C}\ \oplus L}{\Delta', A \vdash C}\ \mathsf{cut}_{A \oplus B} \quad \longrightarrow \quad \begin{array}{c}\mathcal{E}_1 \\ \Delta', A \vdash C\end{array}$$

$$\frac{\dfrac{}{B \vdash A \oplus B}\ \oplus X_2 \qquad \dfrac{\begin{array}{cc}\mathcal{E}_1 & \mathcal{E}_2 \\ \Delta', A \vdash C & \Delta', B \vdash C\end{array}}{\Delta', A \oplus B \vdash C}\ \oplus L}{\Delta', A \vdash C}\ \mathsf{cut}_{A \oplus B} \quad \longrightarrow \quad \begin{array}{c}\mathcal{E}_2 \\ \Delta', B \vdash C\end{array}$$

In both cases, the cut disappears (which corresponds to a message receipt) and only the recipient continues computation. The channel substitution is hidden in this proof notation. For example, in first of the two reduction we would replace the $x : A$ resulting from the case split by the $x' : A$ label in the conclusion which is the same as in the first premise. A symmetric case arises for the second reduction.

5 Generalizing to Other Connectives

We have seen that for internal choice the right rules turned into axioms (representing messages) and the left rule remained unchanged. In general, those rules that carry information should become messages (and thus axioms) while invertible rules should remain as they are. This means for positive connectives the right rules become axioms while the left rules remain.

We call the result the *semi-axiomatic sequent calculus* (SAX) because half the rules are turned into axioms while the other half remains the same.

$$\frac{}{A, B \vdash A \otimes B}\ \otimes X \qquad\qquad \frac{\Delta, A, B \vdash C}{\Delta, A \otimes B \vdash C}\ \otimes L$$

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}X \qquad\qquad \frac{\Delta \vdash C}{\Delta, \mathbf{1} \vdash C}\ \mathbf{1}L$$

Because $\mathbf{1}R$ is already an axiom, it does not change but we have given it a new name, for uniformity. The negatives are symmetric in the sense that the left rules become axioms (they are the ones carrying information) while the right rules remain (they are invertible).

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \mathbin{\&} B}\ \&R \qquad \frac{}{A \mathbin{\&} B \vdash A}\ \&X_1 \quad \frac{}{A \mathbin{\&} B \vdash B}\ \&X_2$$

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ \multimap R \qquad \frac{}{A, A \multimap B \vdash B}\ \multimap X$$

We also have identity and cut as usual.

$$\frac{}{A \vdash A}\ \mathsf{id} \qquad \frac{\Delta \vdash A \quad \Delta', A \vdash C}{\Delta, \Delta' \vdash C}\ \mathsf{cut}$$

Before we return to the computational meaning of these rules, we should ask the obvious questions: (1) if we replace the positive right and negative left rules by axioms do the same judgments hold, and (2) do cut and identity elimination still hold?

6 Relating Sequent Calculus to SAX

We conjecture that the ordinary and semi-axiomatic sequent calculi prove the same sequents. To show this, we will demonstrate how to derive the rules of each calculus in the other.

First, translating from SAX to the sequent calculus. By showing that the SAX rules are derivable in the sequent calculus we can conclude that SAX is *sound*. We only show two examples.

$$\frac{\dfrac{}{A \vdash A}\ \mathsf{id}_A}{A \vdash A \oplus B}\ \oplus R_1 \qquad \frac{\dfrac{}{A \vdash A}\ \mathsf{id}_A \quad \dfrac{}{B \vdash B}\ \mathsf{id}_B}{A, A \multimap B \vdash B}\ \multimap L$$

So, in general to derive the new axioms we just need identity, while the negative right and positive left rules remain unchanged.

Second, translating from the sequent calculus to SAX. We show that the sequent calculus rules are derivable in SAX. Again the rules that don't change are trivial. We show two other examples.

$$\frac{\Delta \vdash A \quad \dfrac{}{A \vdash A \oplus B}\ \oplus X_1}{\Delta \vdash A \oplus B}\ \mathsf{cut}_A \qquad \frac{\dfrac{\Delta \vdash A \quad \dfrac{}{A, A \multimap B \vdash B}\ \multimap X}{\Delta, A \multimap B \vdash B}\ \mathsf{cut}_A \qquad \Delta', B \vdash C}{\Delta, \Delta', A \multimap B \vdash C}\ \mathsf{cut}_B$$

So for this direction we need cut. We formulate this as a theorem.

Theorem 1 (Soundness and Completeness of SAX) $\Delta \vdash A$ *in the sequent calculus iff* $\Delta \vdash A$ *in SAX.*

Proof: We prove in each direction that the rules in the other calculus are derivable. This could be formalized as in induction over the structure of the given derivation.

From left to right we insert suitable cuts (as exemplified above) and from right to left we insert suitable identities (as exemplified above). □

7 Cut Elimination for SAX

The fact that the translation requires us to insert cuts suggests that SAX does not satisfy a traditional cut elimination result. You may want to construct a counterexample for yourself before moving on.

Here is a simple one for distinct atoms P, Q, and R.

$$Q \vdash (P \oplus Q) \oplus R$$

While easily provable in the sequent calculus, in SAX we are stuck right away. It is not the form of an axiom, so we can only proceed with cut.

This is profoundly saddening if you are a logical fundamentalist and proof-theorist. But it turns out that it is also an opportunity for new discoveries!

If you look at the sample cases where cut had to be inserted in the previous section, you see that the cuts have a special form: they only eliminate a *subformula* of the sequent we are trying to prove. In the first example we have A as a subformula of $A \oplus B$, in the second we have both A and B as subformulas of $A \multimap B$. In general such cuts are called *analytic cuts*. Here we call them *snips*, which is an even more restricted class than analytic cuts in the sense that one of the two premises of a snip must be an axiom or another snip. We elide a precise definition for now, but we will come back to it in a future lecture.

We can easily see that we can eliminate all cuts that are not snips. We do this by translating a SAX derivation to the sequent calculus, eliminating cut, and then translating back. This back translation will only require snips if the given sequent derivation is cut-free to start with.

This, however, is not fully satisfying since we would like to relate the rules of computation to cut reduction. Looking ahead, there is indeed a direct cut elimination algorithm utilizing the cut reductions in SAX that we have shown, leaving only snips. For a structural version of SAX, this is proved by DeYoung et al. [2020]—the linear version is significantly easier.

8 Completing Process Assignment and Dynamics

We now take the logical rules back into typing rules, adding continuation channels to all messages. Our convention to assign the name x' to the continuation of a

channel x. First, the positive connectives.

$$\frac{k \in L}{x' : A_k \vdash \mathbf{send}\ x\ k(x') :: (x : \oplus\{\ell : A_\ell\}_{\ell \in L})}\ \oplus X$$

$$\frac{\Delta, x' : A_\ell \vdash Q_\ell(x') \quad (\forall \ell \in L)}{\Delta, x : \oplus\{\ell : A_\ell\}_{\ell \in L} \vdash \mathbf{recv}\ x\ (\ell(x') \Rightarrow Q_\ell(x'))_{\ell \in L} :: (z : C)}\ \oplus L$$

$$\frac{}{y : A, x' : B \vdash \mathbf{send}\ x\ (y, x') :: (x : A \otimes B)}\ \otimes X$$

$$\frac{\Delta, y : A, x' : B \vdash Q(y, x') :: (z : C)}{\Delta, x : A \otimes B \vdash \mathbf{recv}\ x\ ((y, x') \Rightarrow Q(y, x')) :: (z : C)}\ \otimes L$$

$$\frac{}{\cdot \vdash \mathbf{send}\ x\ (\,) :: (x : \mathbf{1})}\ \mathbf{1}X \qquad \frac{\Delta \vdash Q :: (z : C)}{\Delta, x : \mathbf{1} \vdash \mathbf{recv}\ x\ (() \Rightarrow Q) :: (z : C)}\ \mathbf{1}L$$

Now the negatives.

$$\frac{\Delta \vdash P_\ell(x') :: (x' : A_\ell) \quad (\forall \ell \in L)}{\Delta \vdash \mathbf{recv}\ x\ (\ell(x') \Rightarrow P_\ell(x')) :: (x : \&\{\ell : A_\ell\}_{\ell \in L})}\ \&R$$

$$\frac{k \in L}{x : \&\{\ell : A_\ell\}_{\ell \in L} \vdash \mathsf{send}\ x\ k(x') :: (x' : A_k)}\ \&X$$

$$\frac{\Delta, y : A \vdash P(y, x') :: (x' : B)}{\Delta \vdash \mathbf{recv}\ x\ ((y, x') \Rightarrow P(y, x')) :: (x : A \multimap B)}\ \multimap R$$

$$\frac{}{y : A, x : A \multimap B \vdash \mathbf{send}\ x\ (y, x') :: (x' : B)}\ \multimap X$$

Cut and identity do not change from the sequent calculus.

$$\frac{}{y : A \vdash \mathbf{fwd}\ x\ y :: (x : A)}\ \mathsf{id} \qquad \frac{\Delta \vdash P(x) :: (x : A) \quad \Delta', x : A \vdash Q(x) :: (z : C)}{\Delta, \Delta' \vdash x_A \leftarrow P(x)\ ;\ Q(x) :: (z : C)}\ \mathsf{cut}$$

We refactor the dynamics as before.

$$
\begin{array}{llcll}
\text{Messages} & M & ::= & k(x') \mid (y, x') \mid ( ) & \\
\text{Continuations} & K & ::= & (\ell(x') \Rightarrow P_\ell(x'))_{\ell \in L} & (\oplus, \&) \\
 & & \mid & ((y, x') \Rightarrow P(y, x')) & (\otimes, \multimap) \\
 & & \mid & (( ) \Rightarrow P) & (\mathbf{1}) \\
\text{Processes} & P & ::= & x \leftarrow P(x) \,;\, Q(x) & \mathsf{cut} \\
 & & \mid & \mathbf{fwd}\ x\ y & \mathsf{id} \\
 & & \mid & \mathbf{send}\ x\ M & \\
 & & \mid & \mathbf{recv}\ x\ K & \\
 & & \mid & \mathbf{call}\ p\ x\ y_1 \ldots y_n &
\end{array}
$$

The dynamics in this refactored form relies on the $M \triangleright K$ operation defined just below. Recall that a global signature Σ contains (possibly mutually recursive) type and process definitions.

$$
\begin{array}{lcll}
\mathsf{proc}(x \leftarrow P(x) \,;\, Q(x)) & \longrightarrow & \mathsf{proc}(P(a)), \mathsf{proc}(Q(a)) & (a \text{ fresh}) \\
\mathsf{proc}(P(b)), \mathsf{proc}(\mathbf{fwd}\ a\ b) & \longrightarrow & \mathsf{proc}(P(a)) & \\
\mathsf{proc}(\mathbf{send}\ a\ M), \mathsf{proc}(\mathbf{recv}\ a\ K) & \longrightarrow & \mathsf{proc}(M \triangleright K) & \\
\mathsf{proc}(\mathbf{call}\ p\ a\ b_1 \ldots b_n) & \longrightarrow & \mathsf{proc}(P(a, b_1, \ldots, b_n)) & \\
 & & \text{for } p\ x\ y_1 \ldots y_n = P(x, y_1, \ldots, y_n) \in \Sigma &
\end{array}
$$

$$
\begin{array}{lcll}
k(a') \triangleright (\ell(x') \Rightarrow P_\ell(x'))_{\ell \in L} & = & P_k(a') & (k \in L) \\
(b, a') \triangleright ((y, x') \Rightarrow P(y, x')) & = & P(b, a') & \\
( ) \triangleright (( ) \Rightarrow P) & = & P &
\end{array}
$$

9 Example Revisited

Recall the earlier example, which wasn't quite right because we could not explain where the continuation channels would come from.

$$
\begin{array}{l}
\mathsf{bin} = \oplus\{\mathsf{b0} : \mathsf{bin}, \mathsf{b1} : \mathsf{bin}, \mathsf{e} : \mathbf{1}\} \\
\mathit{one}\ (x : \mathsf{bin}) = \mathbf{send}\ x\ \mathsf{b1}(x') \,;\, \mathbf{send}\ x'\ \mathsf{e}(x'') \,;\, \mathbf{send}\ x''\ ( ) \qquad \text{\% approximately}
\end{array}
$$

In SAX we have to explicitly allocate the continuation channels via cut. Because of the orientation of the cut, this requires us to reverse the textual order of the send actions.

$$
\begin{array}{l}
\mathit{one}\ (x : \mathsf{bin}) = x'' \leftarrow \mathbf{send}\ x''\ ( ) \,; \\
\qquad\qquad\qquad\quad x' \leftarrow \mathbf{send}\ x'\ \mathsf{e}(x'') \,; \\
\qquad\qquad\qquad\quad \mathbf{send}\ x\ \mathsf{b1}(x')
\end{array}
$$

Because of the concurrent nature of cut, the order is not significant for the computation of this process and we are left with three messages. These messages form a queue, with the continuation channels acting as "pointers".

$$\mathsf{proc}(\mathbf{call}\ \mathit{one}\ a) \longrightarrow^* \mathsf{proc}(\mathbf{send}\ a''\ (\,)), \mathsf{proc}(\mathbf{send}\ a'\ \mathsf{e}(a'')), \mathsf{proc}(\mathbf{send}\ a\ \mathsf{b1}(a'))$$

Syntactically, the opposite order of sends in the definition would be closer to MPASS. This can be achieved with a "reverse cut" where the client $Q(x)$ precedes the provider $P(x)$.

$$\frac{\Delta', x : A \vdash Q(x) :: (z : C) \quad \Delta \vdash P(x) :: (x : A)}{\Delta, \Delta' \vdash x \rightarrow Q(x)\ ;\ P(x) :: (z : C)}\ \mathsf{cut}^R$$

Since we just reverse the premises of the cut, this is just a syntactic convenience and does not change the essence of the language. Then we could write:

$$\begin{array}{l} \mathit{one}\ (x : \mathsf{bin}) = x' \rightarrow \mathbf{send}\ x\ \mathsf{b1}(x')\ ; \\ \qquad\qquad\quad\ x'' \rightarrow \mathbf{send}\ x'\ \mathsf{e}(x'')\ ; \\ \qquad\qquad\quad\ \mathbf{send}\ x''\ (\,) \end{array}$$

We might decide to pick a different concrete syntax for this, to be discussed in the next lecture.

During lecture, we also briefly discussed an alternative where x' is somehow computed from x, and that the sender and recipient agree on this computation. This could be concrete "address arithmetic" (like: from x we go to $x + 1$) or more abstract (like: from x we go to $x.\overline{\mathsf{b1}}$). This actually has a quite sensible *logical* interpretation in terms of *snips* from Section 7 so we will come back to it, probably two lectures from now. With this case we might write the process one as follows:

$$\mathit{one}\ (x : \mathsf{bin}) = \mathbf{send}\ x\ \mathsf{b1}(\_)\ ;\ \mathbf{send}\ x.\overline{\mathsf{b1}}\ \mathsf{e}(\_)\ ;\ \mathbf{send}\ x.\overline{\mathsf{b1}}.\overline{\mathsf{e}}\ (\,) \qquad \%\ \text{with snips}$$

We have elided the continuation channels in the send actions because they can be computed from the channel x.

References

Gérard Boudol. Asynchrony and the π-calculus. Rapport de Recherche 1702, INRIA, Sophia-Antipolis, 1992.

Henry DeYoung, Frank Pfenning, and Klaas Pruiksma. Semi-axiomatic sequent calculus. In Z. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pages 29:1–29:22, Paris, France, June 2020. LIPIcs 167.

Robert H. Halstead. Multilisp: A language for parallel symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–539, October 1985.

Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In H.-J. Boehm and G. Steele, editors, *Proceedings of the 23rd Symposium on Principles of Programming Languages (POPL'96)*, pages 358–371, St. Petersburg Beach, Florida, USA, January 1996. ACM.

Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, September 1992. Parts I and II.

Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. *Mathematical Structures in Computer Science*, 13(5): 685–719, 2003.

Frank Pfenning and Dennis Griffith. Polarized substructural session types. In A. Pitts, editor, *Proceedings of the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2015)*, pages 3–22, London, England, April 2015. Springer LNCS 9034. Invited talk.