A Purely Logical Account of Sequentiality in Proof Search

Paola Bruscoli
Technische Universität Dresden
Fakultät Informatik - 01062 Dresden - Germany
Paola.Bruscoli@Inf.TU-Dresden.DE

Abstract *A strict correspondence between the proof-search space of a logical formal system and computations in a simple process algebra is established. Sequential composition in the process algebra corresponds to a logical relation in the formal system—in this sense our approach is purely logical, no axioms or encodings are involved. The process algebra is a minimal restriction of* CCS *to parallel and sequential composition; the logical system is a minimal extension of multiplicative linear logic. This way we get the first purely logical account of sequentiality in proof search. Since we restrict attention to a small but meaningful fragment, which is then of very broad interest, our techniques should become a common basis for several possible extensions. In particular, we argue about this work being the first step in a two-step research for capturing most of* CCS *in a purely logical fashion.*

1 Introduction

One of the main motivations of logic programming is the idea of using a high level, logical specification of an algorithm, which abstracts away from many details related to its execution. As Miller pointed out, logical operators can be interpreted as high level search instructions, and the sequent calculus can be used to give a very clear and simple account of logic programming [13].

In traditional logic programming, one is mainly interested in the result of a computation, and computing is essentially the exploration of a search space. Recently, Miller's methods have been extended to so-called resource-conscious logics, like linear logic [4, 12], and researchers designed several languages based on them [2, 10, 12]. These logics allow to deal directly with notions of *resources*, *messages*, *processes*, and so on; in other words, it is possible to give a proof-theoretical account of concurrent computations, in the logic programming spirit.

A concurrent computation is not as much about getting a result, as it is about establishing certain communication patterns, protocols, and the like. Hence we might wonder to which extent logic can be useful in the specification of concurrent programs. Differently stated, if concurrent programs are essentially protocols, subject mainly to an operational view of computation, can logic contribute to their design? We are not concerned here about the use of logics to prove properties of programs, like, say, Hennessy-Milner logic for CCS. We want to use logic in the design of languages for concurrent computation, in order to obtain some useful *inherent* properties, at the object level, so to speak.

In this paper I will present a very simple process algebra and I will argue about its proof-theoretical understanding in terms of proof-search. We will work within the calculus of structures [7], which is a recent generalisation of the sequent

calculus [3]. Guglielmi and Tiu showed how it is possible to design, in the calculus of structures, a simple logical system which possesses a self-dual non-commutative operator [7], and how this system can not be defined in the sequent calculus [16]. This non-commutative operator, called *seq*, has a resemblance to the prefix combinator of CCS [14]; it is a form of sequential composition, similar to other sequential constructs in other languages. (We should not forget that sequential composition has a longer history than parallel forms of composition, which more naturally correspond to the usual commutative logical operators.)

We will consider the simplest system containing seq, called system SBV: it is not very expressive (it is decidable), but contains the hard part of our problem. Beyond seq, SBV has two commutative logical operators, corresponding to linear logic's *par* and *times*. Several steps have to be made before a real language can be designed starting from SBV:

1 The correspondence between seq and a form of sequentiality studied independently must be established.

2 The search space for proofs must be narrowed enough to get the desired behaviour at run-time.

3 SBV must be extended to a Turing-equivalent fragment and the two properties above must be preserved.

In this paper we will deal with 1 and, partially, with 2, and I will argue about the possibility of completing the program in future work. Let us see in more detail what the three issues above are about.

Point 1: I believe that logic, in the sense of the formal study of language, should give an account of *existing* languages (as opposed to the creation of new *ad hoc* ones). As mathematical logic formalised mathematical reasoning, logic for computer science should deal with natural languages of computer science. Of course, computer science is young, and we should not expect the same kind of maturity that the language of mathematicians had reached when logic began. That said, I will consider CCS a natural language to close as much as possible on. As we will see, one of the main problems we have to deal with is the difference between the logical notion of sequentiality of seq, and the operational one of CCS's prefix combinator.

Point 2: In the calculus of structures, even more than in the sequent calculus, the bottom-up construction of proofs is a very non-deterministic process; this is due to the fact that inference rules may be applied anywhere deep in a structure. If this non-determinism is not tamed, our ability to design concurrent algorithms is severely hampered. Here I will solve part of this problem: to establish the operational correspondence between seq and prefix we have to coerce the search for proofs, otherwise the order induced by seq is not respected by the computational interpretation of proofs. This aspect is solved *logically*: I will show a system, called BVL, which is equivalent to SBV but which generates only those proofs that correspond to computations respecting the time-order induced by the prefixing. I will show the correspondence to CCS of this intermediate system.

Still, BVL generates more proofs than desirable for just an operational account, and the best answer to this problem should come by further applying methods inspired by Miller's uniform proofs. We will not deal with this in the present

paper, although I argue that this operation is entirely feasible because: 1) the calculus of structures is *more general* than the sequent calculus, so the methods for the sequent calculus should work as well; 2) our system is an extension of multiplicative linear logic, which so far has been the most successful logical system *vis-à-vis* the uniform proofs [12].

Point 3: Recent work by Guglielmi and Straßburger provides the extension: they designed a Turing-equivalent system, called SNEL, which conservatively extends SBV with exponentials [9]. Since we find there the usual exponential of linear logic, it should be possible to map fixpoint operators by simple, known replication techniques. SNEL is also a conservative extension of MELL, the multiplicative-exponential fragment of linear logic, amenable to the uniform proof reduction mentioned above. CCS choice operator requires additives: a presentation of full linear logic is provided in [15]; then we can borrow techniques from [11].

For those reasons, this paper establishes the first of what I believe is a two-step move towards the first abstract logic programming system directly corresponding to CCS and similar process algebras. More in detail, these are the contributions of this paper:

1 A logical system in the calculus of structures, BVL, which is equivalent to SBV and which shows a general technique for limiting non-determinism in the case of a non-commutative self-dual logical operator. This is a purely proof-theoretical result (Section 3).

2 A simple process algebra, $\mathsf{PA}_{\mathsf{BV}}$, corresponding to CCS restricted to the sequential and parallel operators, which is exactly captured by BVL: 1) Every terminating computation in it corresponds to a proof of BVL. 2) For every (legal) expression provable in BVL there is a corresponding terminating computation (Section 4).

Compared to some previous work, notably by Miller [11] and Guglielmi [5, 6], my approach has a distinctive, important feature: sequentiality is not obtained through axioms, or through an encoding, rather it is realised by a logical operator in the system. Despite the simplicity of the system, getting cut elimination has proved extremely difficult (it turned out to be impossible in the sequent calculus) and required the development of the calculus of structures.

This effort gives us an important property in exchange. As I will argue later in the paper, we will be able to manipulate proofs at various levels of abstraction: 1) There is the concrete level of BVL, where a proof closely corresponds to a computation. 2) More abstractly, we can use a restriction of SBV called BV, where we are free to exchange messages disregarding the actual ordering of the computation; here, for example, we could verify what happens towards the end of a computation without being forced to execute its beginning. 3) Even more abstractly, we could use in addition a new admissible rule which allows us to separate certain threads of a computation when performing an analysis. 4) Finally, we can use cut rules (in various forms), so reducing dramatically the search space.

As is typical in the calculus of structures, there is in fact a whole hierarchy of equivalent systems, generated as a consequence of the more general kind of cut elimination we have in this formalism. The smallest system is the concrete one,

corresponding to computations; all the others can be used for analysis, verification, and the like.

2 Basic Definitions

In this section I will shortly present definitions and results that the reader can find in more extensive details in [7] and [8]. I call *calculus* a formalism, like natural deduction or the sequent calculus, for specifying logical systems. A *system* in the calculus of structures is defined by a language of structures, an equational theory over structures, and a collection of inference rules. The equational theory serves just the purpose of handling simple decidable properties, like commutativity or idempotency of logical operators, something that in the sequent calculus is usually implicitly assumed. It also defines negation, as is typical in linear logic.

Let us first define the language of structures of BV. Intuitively, $[S_1, \ldots, S_h]$ corresponds to a sequent $\vdash S_1, \ldots, S_h$ or, equivalently to the formula $S_1 \parr \cdots \parr S_h$. The structure $(S_1, \ldots, S_h)$ corresponds to $S_1 \otimes \cdots \otimes S_h$. The structure $\langle S_1; \ldots; S_h \rangle$ has no correspondence in linear logic, it should be considered the *sequential* or *non-commutative* composition of $S_1, \ldots, S_h$.

2.1 Definition We consider a set $\mathcal{A}$ of countably many *positive atoms* and *negative atoms*, denoted by a, b, c, ... *Structures* are denoted with S, P, Q, R, T, U and V. The structures of the *language* BV are generated by

$$S ::= \circ \mid a \mid \langle \underbrace{S; \ldots; S}_{>0} \rangle \mid [\underbrace{S, \ldots, S}_{>0}] \mid (\underbrace{S, \ldots, S}_{>0}) \mid \bar{S} \quad ,$$

where $\circ$, the *unit*, is not an atom; $\langle S_1; \ldots; S_h \rangle$ is a *seq structure*, $[S_1, \ldots, S_h]$ is a *par structure* and $(S_1, \ldots, S_h)$ is a *copar structure*; $\bar{S}$ is the *negation* of the structure S. The notation $S\{\ \}$ stands for a structure with a hole that is not in the scope of a negation, and denotes the *context* of the structure R in $S\{R\}$; we also say that the structure R is a *substructure* of $S\{R\}$. We drop contextual parentheses whenever structural parentheses fill exactly the hole: for instance $S[R, T]$ stands for $S\{[R, T]\}$.

Inference rules assume a peculiar shape in our formalism: they typically have the form $\rho \dfrac{S\{T\}}{S\{R\}}$, which stands for a scheme ρ, stating that if a structure matches R, in a context $S\{\ \}$, then it can be replaced by T without acting in the context at all (and analogously if one prefers a top-down reading). A rule is a way to implement any axiom $T \Rightarrow R$, where $\Rightarrow$ stands for the implication we model in the system, but it would be simplistic to regard a rule as a different notation for axioms. The entire design process of rules is done for having cut elimination and the subformula property; these proof theoretical properties are foundational for proof search and abstract logic programming. A *derivation* is a composition of instances of inference rules and a *proof* is a derivation free from hypotheses: the shape of rules confers to derivations (but not to proofs) a vertical symmetry.

2.2 Definition An (*inference*) *rule* is any scheme $\rho\,\dfrac{T}{R}$, where ρ is the *name* of the rule, T is its *premise* and R is its *conclusion*; at most one between R and T may be missing. A set of rules defines a (*formal*) *system*, denoted by $\mathscr{S}$. A *derivation* in a system $\mathscr{S}$ is a finite chain of instances of rules of $\mathscr{S}$, is denoted by Δ and can consist of just one structure. Its topmost and bottommost structures are respectively called *premise* and *conclusion*. A derivation Δ in $\mathscr{S}$ whose premise is T and conclusion is R is denoted by $\Delta\dfrac{\|}{\begin{matrix}T\\R\end{matrix}}\mathscr{S}$.

It is customary in the calculus of structures first to define symmetric systems, returning just derivations, and only afterwards to break the symmetry by adding an (asymmetric) axiom. Symmetric systems are obtained by considering for each rule also its *corule*, defined by swapping and negating premise and conclusion. Hence, we typically deal with *pairs* of rules, $\rho{\downarrow}\,\dfrac{S\{T\}}{S\{R\}}$ (down version) and $\rho{\uparrow}\,\dfrac{S\{\bar{R}\}}{S\{\bar{T}\}}$ (up version), that make the system closed by contraposition. When the up and down versions coincide, the rules are self-dual, and in this case we will omit the arrows.

We now define system BV by extracting it from its symmetric version SBV. In SBV we distinguish a fragment, called *interaction*, which deals solely with negation; the rest of the system, the *structure* fragment, deals with logical relations. In analogy with sequent calculus presentations, the interaction fragment corresponds to the rules dealing with identity and cut, and the structure fragment to logical (and structural) rules. Note that in the calculus of structures rules are defined on complex contexts: *pairs* of logical relations are taken simultaneously into account.

2.3 Definition The structures of the language BV are equivalent modulo the relation $=$, defined at the left of Fig. 1. By $\vec{R}$, $\vec{T}$ and $\vec{U}$ we denote finite, non-empty sequences of structures (sequences may contain ',' or ';' separators as appropriate in the context). Structures whose only negated substructures are atoms are said to be *in normal form*. At the right of the figure *system* SBV is shown (*symmetric basic system* V). The rules $\mathsf{ai}{\downarrow}$, $\mathsf{ai}{\uparrow}$, s, $\mathsf{q}{\downarrow}$ and $\mathsf{q}{\uparrow}$ are called respectively *atomic interaction*, *atomic cut*, *switch*, *seq* and *coseq*. The *down fragment* of SBV is $\{\mathsf{ai}{\downarrow},\mathsf{s},\mathsf{q}{\downarrow}\}$, the *up fragment* is $\{\mathsf{ai}{\uparrow},\mathsf{s},\mathsf{q}{\uparrow}\}$.

It helps intuition always to consider structures in normal form, where not otherwise indicated. There is a straightforward two-way correspondence between structures not involving seq and formulae of multiplicative linear logic (MLL) in the version including mix and nullary mix: for example $[(a,\bar{b}),c,\bar{d}]$ corresponds to $((a\otimes b^{\perp})\parr c\parr d^{\perp})$, and vice versa. Units are mapped into $\circ$, since $1\equiv\perp$, when mix and nullary mix are present [1]. The reader can check that the equations in Fig. 1 correspond to equivalences in MLL plus mix and nullary mix, disregarding seq, and that rules correspond to valid implications.

Our three logical relations share a common self-dual unit $\circ$, which can be regarded as the empty sequence; it gives us flexibility in the application of rules.

Associativity

$$\langle \vec{R}; \langle \vec{T} \rangle; \vec{U} \rangle = \langle \vec{R}; \vec{T}; \vec{U} \rangle \qquad [\vec{R}, [\vec{T}]] = [\vec{R}, \vec{T}] \qquad (\vec{R}, (\vec{T})) = (\vec{R}, \vec{T})$$

Commutativity

$$[\vec{R}, \vec{T}] = [\vec{T}, \vec{R}] \qquad (\vec{R}, \vec{T}) = (\vec{T}, \vec{R})$$

Unit

$$\langle \circ; \vec{R} \rangle = \langle \vec{R}; \circ \rangle = \langle \vec{R} \rangle \qquad [\circ, \vec{R}] = [\vec{R}] \qquad (\circ, \vec{R}) = (\vec{R})$$

Singleton

$$\langle R \rangle = [R] = (R) = R$$

Negation

$$\bar{\circ} = \circ \qquad \overline{\langle R; T \rangle} = \langle \bar{R}; \bar{T} \rangle \qquad \overline{[R, T]} = (\bar{R}, \bar{T}) \qquad \overline{(R, T)} = [\bar{R}, \bar{T}] \qquad \bar{\bar{R}} = R$$

Contextual Closure

if $R = T$ then $S\{R\} = S\{T\}$

$$\mathsf{ai}{\downarrow}\,\frac{S\{\circ\}}{S[a, \bar{a}]} \qquad \mathsf{ai}{\uparrow}\,\frac{S(a, \bar{a})}{S\{\circ\}}$$

Interaction

Structure

$$\mathsf{s}\,\frac{S([R, U], T)}{S[(R, T), U]}$$

$$\mathsf{q}{\downarrow}\,\frac{S\langle [R, U]; [T, V] \rangle}{S[\langle R; T \rangle, \langle U; V \rangle]} \qquad \mathsf{q}{\uparrow}\,\frac{S(\langle R; U \rangle, \langle T; V \rangle)}{S\langle (R, T); (U, V) \rangle}$$

Fig. 1 **Left:** *Syntactic equivalence = for* BV **Right:** *System* SBV

For example, consider the following derivation:

$$\begin{array}{l} \mathsf{q}{\uparrow}\,\dfrac{(a, b)}{\langle a; b \rangle} \\ \mathsf{q}{\downarrow}\,\dfrac{}{[a, b]} \end{array} \quad = \quad \mathsf{q}{\uparrow}\,\dfrac{(\langle a; \circ \rangle, \langle \circ; b \rangle)}{\langle [a, \circ]; [\circ, b] \rangle = \langle (a, \circ); (\circ, b) \rangle} \\ \mathsf{q}{\downarrow}\,\dfrac{}{[\langle a; \circ \rangle, \langle \circ; b \rangle]} \quad .$$

Looking at the rules of system SBV, we note that all of them, apart from the cut rule, guarantee the subformula property: the premise only involves substructures of the structures of the conclusion.

The rules $\mathsf{i}{\downarrow}\,\dfrac{S\{\circ\}}{S[R, \bar{R}]}$ and $\mathsf{i}{\uparrow}\,\dfrac{S(R, \bar{R})}{S\{\circ\}}$ define respectively general forms of *interaction* and *cut*: as shown in [7, 8], they are admissible, respectively, for the down and up fragment of SBV.

So far we have dealt with SBV, a top-down symmetric system, lacking any notion of proof. Particularly relevant for provability is a study of permutability and admissibility of rules: the symmetric system is simplified into an equivalent minimal one, by discarding the entire fragment of up rules. Behind this, is that $T \Rightarrow R$ and $\bar{R} \Rightarrow \bar{T}$ are equivalent statements in many logics. Related to this phenomenon, systems in the calculus of structures have two distinctive features:

1 The cut rule splits into several up rules, and since we can eliminate up rules successively and independently one from the other, the cut elimination argument becomes modular. In our case $\mathsf{i}{\uparrow}$ can be decomposed into $\mathsf{ai}{\uparrow}, \mathsf{s}$ and $\mathsf{q}{\uparrow}$, in every derivation.

2 Adding up rules to the minimal system, while preserving provability, allows to define a broader range of equivalent systems than what we might expect in more traditional calculi, like sequent calculus (or natural deduction).

2.4 Definition The following (logical axiom) rule is called *unit*: $\circ{\downarrow}\,\dfrac{}{\circ}$. The system in Fig. 2 is called *system* BV (*basic system* V).

Note that system BV is cut-free, and every rule has the subformula property.

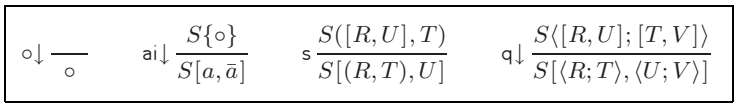

Fig. 2 *System* BV

2.5 Definition A *proof* is a derivation whose topmost inference rule is an instance of the unit rule. Proofs are denoted with Π. A formal system $\mathscr{S}$ *proves* R if there is in $\mathscr{S}$ a proof Π whose conclusion is R, written $\begin{array}{c}\Pi\\ \| \\ S\end{array}\!\!\mathscr{S}$. Two systems are *equivalent* if they prove the same structures.

Observe that $\circ\!\downarrow$ can only occur once in a derivation, and only at the top.

This is the cut elimination theorem, in a much more general form than possible in the sequent calculus:

2.6 Theorem *All the following systems are equivalent*: BV, $\mathsf{BV} \cup \{\mathsf{q}\uparrow\}$, $\mathsf{BV} \cup \{\mathsf{ai}\uparrow\}$, $\mathsf{BV} \cup \{\mathsf{i}\uparrow\}$, *and* $\mathsf{SBV} \cup \{\circ\!\downarrow\}$.

In addition, and according to the correspondence mentioned above, we have that BV is a conservative extension of MLL plus mix and nullary mix.

3 Restricting Interaction

In this section we will see a system equivalent to BV, and so to all systems equivalent to it, in which interaction is limited to certain contexts only. This limitation will be instrumental in showing the correspondence to CCS. Intuitively, in CCS interaction happens in the order induced by prefixing; by restricting interaction in BV, we force this ordering. Some proofs in the following are very sketchy, due to length constraints. I tried to put the emphasis on the techniques that are closer to our process algebra.

3.1 Definition The structure context $S\{\ \}$ is a *right context* if there are no structure $R \neq \circ$ and no contexts $S'\{\ \}$ and $S''\{\ \}$ such that $S\{\ \} = S'\langle R; S''\{\ \}\rangle$. Right contexts are also denoted by $S\{\ \}^{\llcorner}$, where the $\llcorner$ stands for (hole at the) left. We tag with $^{\llcorner}$ structural parentheses instead of contextual ones whenever possible: for example $S[R,T]^{\llcorner}$ stands for $S\{[R,T]\}^{\llcorner}$.

For example $S_1\{\ \}^{\llcorner} = [a,b,\langle\{\ \};c\rangle]$, $S_2\{\ \}^{\llcorner} = (a,\{\ \},b)$ and $S_3\{\ \}^{\llcorner} = \langle[a,\{\ \}];b\rangle$ are right contexts, whilst $[a,(b,\langle c;\{\ \}\rangle)]$ and $\langle(a,[b,c]);\{\ \}\rangle$ are not.

3.2 Definition The next rule is called *left atomic interaction*: $\mathsf{ai}\!\downarrow_{\llcorner} \dfrac{S\{\circ\}^{\llcorner}}{S[a,\bar{a}]^{\llcorner}}$; $[a,\bar{a}]$ is its *redex*. The system $\{\circ\!\downarrow, \mathsf{ai}\!\downarrow_{\llcorner}, \mathsf{q}\!\downarrow, \mathsf{s}\}$ is called *system* BVL.

Trivially, instances of $\mathsf{ai}\!\downarrow_{\llcorner}$ are instances of $\mathsf{ai}\!\downarrow$, and hence any proof in BVL is also a proof in BV.

We introduce some terminology for our coming analysis of permutability.

3.3 Definition A rule ρ *permutes by* $\mathscr{S}$ *over* ρ' if for all $\rho'\dfrac{Q}{U}\ \rho\dfrac{Q}{V}$ there is $\begin{array}{c}\rho\dfrac{U}{P}\\ \| \\ P\end{array}\!\!\mathscr{S}\cup\{\rho'\}$, for some V.

3.4 Lemma *The rule* $\mathsf{ai}{\downarrow}$ *permutes by* $\{\mathsf{q}{\downarrow}\}$ *over* $\mathsf{ai}{\downarrow_{\mathsf{L}}}$.

Proof Consider $\Delta = \dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{Q}{S\{\circ\}}}{\mathsf{ai}{\downarrow}\dfrac{}{S[a,\bar{a}]}}$. We reason about the position of the redex of $\mathsf{ai}{\downarrow_{\mathsf{L}}}$ in $S\{\circ\}$. The following cases exhaust all possibilities:

1 The redex of $\mathsf{ai}{\downarrow_{\mathsf{L}}}$ is inside context $S\{\ \}$:

$$\dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}}{S\{\circ\}}}{\mathsf{ai}{\downarrow}\dfrac{}{S[a,\bar{a}]}} \quad \text{yields} \quad \dfrac{\mathsf{ai}{\downarrow}\dfrac{S'\{\circ\}}{S'[a,\bar{a}]}}{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{}{S[a,\bar{a}]}} \quad .$$

2 Otherwise, there are only three possibilities:

1 $S\{\ \} = S'[b, \langle\{\ \}; \bar{b}\rangle]$, for some b; in this case

$$\Delta = \mathsf{ai}{\downarrow}\dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[b,\bar{b}]^{\mathsf{L}}}}{S'[b,\langle[a,\bar{a}];\bar{b}\rangle]^{\mathsf{L}}} \quad \text{trivially yields} \quad \mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[b,\bar{b}]^{\mathsf{L}}} \quad .$$

2 $S\{\ \} = S'[b, \langle\bar{b}; \{\ \}\rangle]$, for some b; in this case

$$\Delta = \mathsf{ai}{\downarrow}\dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[b,\bar{b}]^{\mathsf{L}}}}{S'[b,\langle\bar{b};[a,\bar{a}]\rangle]^{\mathsf{L}}} \quad \text{yields} \quad \dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[a,\bar{a}]^{\mathsf{L}}}}{\mathsf{q}{\downarrow}\dfrac{}{S'[b,\langle\bar{b};[a,\bar{a}]\rangle]^{\mathsf{L}}}} \quad .$$

3 $S\{\ \} = S'[b, (\{\ \}, \bar{b})]$, for some b; in this case

$$\Delta = \mathsf{ai}{\downarrow}\dfrac{\mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[b,\bar{b}]^{\mathsf{L}}}}{S'[b,([a,\bar{a}],\bar{b})]^{\mathsf{L}}} \quad \text{trivially yields} \quad \mathsf{ai}{\downarrow_{\mathsf{L}}}\dfrac{S'\{\circ\}^{\mathsf{L}}}{S'[b,\bar{b}]^{\mathsf{L}}} \quad .$$

□

3.5 Lemma *The rule* $\mathsf{ai}{\downarrow}$ *permutes by* $\{\mathsf{q}{\uparrow},\mathsf{s}\}$ *over the rules* $\mathsf{q}{\downarrow}$, $\mathsf{q}{\uparrow}$ *and* s.

Proof We first prove that for every $S\{\ \}$ and R there exists a derivation $\dfrac{(S\{\circ\},R)}{S\{R\}}\,\|_{\{\mathsf{q}\uparrow,\mathsf{s}\}}$ (easy structural induction on $S\{\ \}$); then for every $\rho \in \{\mathsf{q}{\downarrow},\mathsf{q}{\uparrow},\mathsf{s}\}$ we have:

$$\dfrac{\rho\dfrac{Q}{S\{\circ\}}}{\mathsf{ai}{\downarrow}\dfrac{}{S[a,\bar{a}]}} \quad \text{yields} \quad \dfrac{\mathsf{ai}{\downarrow}\dfrac{Q}{(Q,[a,\bar{a}])}}{\rho\dfrac{}{(S\{\circ\},[a,\bar{a}])}}\,\Big\|_{\{\mathsf{q}\uparrow,\mathsf{s}\}}\,S[a,\bar{a}] \quad .$$

□

Then, trivially, from Lemmas 3.4 and 3.5:

3.6 Theorem *The rule* $\mathsf{ai}{\downarrow}$ *permutes by* $\{\mathsf{q}{\downarrow},\mathsf{q}{\uparrow},\mathsf{s}\}$ *over* $\mathsf{ai}{\downarrow_{\mathsf{L}}},\mathsf{q}{\downarrow},\mathsf{q}{\uparrow}$ *and* s.

3.7 Theorem *If there is a proof for R in* BV*, then there is a proof for R in* $\mathsf{BVL} \cup \{\mathsf{q}\uparrow\}$.

Proof The topmost instance of $\mathsf{ai}\downarrow$ in a proof is also an instance of $\mathsf{ai}\downarrow_{\mathsf{L}}$. Transform the given proof as follows: Take the topmost instance of an $\mathsf{ai}\downarrow$ rule which is not already an $\mathsf{ai}\downarrow_{\mathsf{L}}$ instance and permute it up, by Theorem 3.6, until it becomes an instance of $\mathsf{ai}\downarrow_{\mathsf{L}}$ (which always happens when the instance reaches the top of a proof). Proceed inductively. □

For example, the proof on the left, where we have already renamed the topmost instance of $\mathsf{ai}\downarrow$ as $\mathsf{ai}\downarrow_{\mathsf{L}}$, is successively transformed as follows:

$$\mathsf{ai}\downarrow\dfrac{\mathsf{ai}\downarrow\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[c,\bar{c}]}}{[c,\langle\bar{c};[b,\bar{b}]\rangle]}}{[c,\langle\bar{c};[b,(\bar{b},[a,\bar{a}])]\rangle]} \quad\rightarrow\quad \mathsf{ai}\downarrow\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[b,\bar{b}]}}{\langle[c,\bar{c}];[b,\bar{b}]\rangle}}{[c,\langle\bar{c};[b,(\bar{b},[a,\bar{a}])]\rangle]} \quad\rightarrow$$

$$\mathsf{q}\downarrow\dfrac{\mathsf{ai}\downarrow\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[b,\bar{b}]}}{\langle[c,\bar{c}];[b,\bar{b}]\rangle}}{\langle[c,\bar{c}];[b,(\bar{b},[a,\bar{a}])]\rangle}}{[c,\langle\bar{c};[b,(\bar{b},[a,\bar{a}])]\rangle]} \quad\rightarrow\quad \mathsf{q}\downarrow\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\mathsf{ai}\downarrow_{\mathsf{L}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[b,\bar{b}]}}{[b,(\bar{b},[a,\bar{a}])]}}{\langle[c,\bar{c}];[b,(\bar{b},[a,\bar{a}])]\rangle}}{[c,\langle\bar{c};[b,(\bar{b},[a,\bar{a}])]\rangle]} \quad .$$

We need to refine the preceding theorem such that we can get rid of the $\mathsf{q}\uparrow$ rule in our system.

3.8 Theorem *If there is a proof for R in* BV*, and no copar structure appears in R, then there is a proof for R in* BVL.

Proof Take the given proof for R and transform it into one in $\mathsf{BVL} \cup \{\mathsf{q}\uparrow\}$, by Theorem 3.7. Since no copar appears in R, the bottommost instance of $\mathsf{q}\uparrow$ in the proof must necessarily be as in

$$\mathsf{q}\uparrow\dfrac{\dfrac{\dfrac{\vdots}{S(T,U)}}{S\langle T;U\rangle}}{R} \quad .$$

Transform the proof by upwardly changing (T,U) into $\langle T;U\rangle$, and correspondingly transforming s instances into $\mathsf{q}\downarrow$ instances. This eliminates one instance of $\mathsf{q}\uparrow$. Possibly, some instances of $\mathsf{ai}\downarrow_{\mathsf{L}}$ become simple $\mathsf{ai}\downarrow$. Rearrange them until all are again $\mathsf{ai}\downarrow_{\mathsf{L}}$ and repeat the procedure until all $\mathsf{q}\uparrow$ instances are eliminated. □

At this time I don't know whether it is possible to lift the restriction on R containing no copars. I believe that it is possible, but the proof does not look easy.

Laws for expressions

$$E \mid \circ = E$$
$$E \mid E' = E' \mid E$$
$$E \mid (E' \mid E'') = (E \mid E') \mid E''$$

Law for action sequences

$$\alpha_1; \ldots; \alpha_{i-1}; \circ; \alpha_i; \ldots; \alpha_n = \alpha_1; \ldots; \alpha_n$$

$$\mathsf{Cp}\ \frac{}{a.E \mid F \xrightarrow{a} E \mid F} \qquad \mathsf{Cs}\ \frac{E \xrightarrow{a} E' \quad F \xrightarrow{\bar{a}} F'}{E \mid F \xrightarrow{\circ} E' \mid F'}$$

Fig. 3 **Left:** *Syntactic equivalences for* $\mathsf{PA}_{\mathsf{BV}}$ **Right:** *Transition rules for* $\mathsf{PA}_{\mathsf{BV}}$

4 Relations with a Simple Process Algebra

4.1 Completeness

We now introduce some definitions and notation for a simple process algebra $\mathsf{PA}_{\mathsf{BV}}$ corresponding to the CCS fragment of prefixing and parallel composition.

4.1.1 Definition Let $\mathcal{L} = (\mathcal{A}/{=}) \cup \{\circ\}$ be the set of *labels* or *actions*, where $\circ$ is called the *internal* (or *silent*) action; we denote actions by α. The *process expressions* of $\mathsf{PA}_{\mathsf{BV}}$, denoted by E and F, are generated by

$$E ::= \circ \mid a.E \mid (E \mid E) \quad ,$$

where the combinators '.' and '|' are called respectively *prefix* and *composition*, and prefix is stronger than composition. We will consider expressions equivalent up to the laws defined at the left in Fig. 3. We denote the set of expressions by $\mathcal{E}_{\mathsf{PA}}$. At the right of Fig. 3 the transition rules of $\mathsf{PA}_{\mathsf{BV}}$ are defined: Cp is called *prefix* and Cs is called *synchronisation*.

Operational semantics is given by way of the labelled transition system $(\mathcal{E}_{\mathsf{PA}}, \mathcal{L}, \{\xrightarrow{\alpha} : \alpha \in \mathcal{L}\})$. We introduce some basic terminology and notation.

4.1.2 Definition In the *computation* $E \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} F$ we call $\alpha_1; \ldots; \alpha_n$ an *action sequence* of E; action sequences are considered equivalent up to the law at the left in Fig. 3; action sequences are denoted by $\vec{s}$; if $n = 0$ then E is the empty computation, its action sequence is empty and is denoted by ϵ. *Terminating* computations are those whose last expression is $\circ$. A computation $E \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} F$ can also be written $E \xrightarrow{\alpha_1;\ldots;\alpha_n} F$.

The reader will have no trouble in verifying that our process algebra indeed is equivalent to the fragment of CCS with prefix and parallel composition, as is presented, for example, in [14]. We make no distinction between 0 and τ, they both are collapsed into the unit $\circ$.

4.1.3 Definition The function $\underline{\cdot}_{\mathsf{S}}$ maps the expressions in $\mathcal{E}_{\mathsf{PA}}/{=}$ and the action sequences in $\mathcal{L}^*/{=}$ into structures of BV according to the following inductive definition:

$$\underline{\circ}_{\mathsf{S}} = \circ \quad , \qquad \underline{\epsilon}_{\mathsf{S}} = \circ \quad ,$$
$$\underline{a.E}_{\mathsf{S}} = \langle a; \underline{E}_{\mathsf{S}} \rangle \quad , \qquad \underline{a}_{\mathsf{S}} = a \quad ,$$
$$\underline{E \mid F}_{\mathsf{S}} = [\underline{E}_{\mathsf{S}}, \underline{F}_{\mathsf{S}}] \quad ; \qquad \underline{\alpha_1; \ldots; \alpha_{n_{\mathsf{S}}}} = \langle \underline{\alpha_1}_{\mathsf{S}}; \ldots; \underline{\alpha_n}_{\mathsf{S}} \rangle \quad .$$

4.1.4 Theorem *For every computation* $E_0 \xrightarrow{\vec{s}} E_n$ *there is a derivation* $\dfrac{\underline{E_n}_{\mathsf{S}}}{\big\|_{\mathsf{BVL}}}$. $[\underline{E_0}_{\mathsf{S}}, \underline{\vec{s}}_{\mathsf{S}}]$

Proof By induction on n. If $n = 0$ take the derivation $\underline{E_0}_{\mathsf{S}}$. The inductive cases are:

1 $E_0 \xrightarrow{a} E_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} E_n$: It must be $E_0 = a.E \mid F$, for some E and F, and $E_1 = E \mid F$. Let $S = \underline{\overline{\alpha_2; \ldots; \alpha_n}}_{\mathsf{S}}$; we can build:

$$\dfrac{\dfrac{\dfrac{\underline{E_n}_{\mathsf{S}}}{\big\|_{\mathsf{BVL}}}}{[\underline{E}_{\mathsf{S}}, \underline{F}_{\mathsf{S}}, S]}\,\mathsf{ai}{\downarrow_{\mathsf{L}}}}{[\langle [a, \bar{a}]; [\underline{E}_{\mathsf{S}}, S]\rangle, \underline{F}_{\mathsf{S}}]}\,\mathsf{q}{\downarrow}}{[\langle a; \underline{E}_{\mathsf{S}}\rangle, \underline{F}_{\mathsf{S}}, \langle \bar{a}; S\rangle]} \quad .$$

2 $E_0 \xrightarrow{\circ} E_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} E_n$: It must be $E_0 = E \mid F$, $E_1 = E' \mid F'$, $E = a.E'' \mid F''$, $E' = E'' \mid F''$, $F = \bar{a}.E''' \mid F'''$ and $F' = E''' \mid F'''$. Let $S = \underline{\overline{\alpha_2; \ldots; \alpha_n}}_{\mathsf{S}}$; we can build:

$$\dfrac{\dfrac{\dfrac{\underline{E_n}_{\mathsf{S}}}{\big\|_{\mathsf{BVL}}}}{[\underline{E''}_{\mathsf{S}}, \underline{F''}_{\mathsf{S}}, \underline{E'''}_{\mathsf{S}}, \underline{F'''}_{\mathsf{S}}, S]}\,\mathsf{ai}{\downarrow_{\mathsf{L}}}}{[\langle [a, \bar{a}]; [\underline{E''}_{\mathsf{S}}, \underline{E'''}_{\mathsf{S}}]\rangle, \underline{F''}_{\mathsf{S}}, \underline{F'''}_{\mathsf{S}}, S]}\,\mathsf{q}{\downarrow}}{[\langle a; \underline{E''}_{\mathsf{S}}\rangle, \underline{F''}_{\mathsf{S}}, \langle \bar{a}; \underline{E'''}_{\mathsf{S}}\rangle, \underline{F'''}_{\mathsf{S}}, S]} \quad .$$

□

4.1.5 Corollary *For every terminating computation in* $\mathsf{PA}_{\mathsf{BV}}$ *there exists a proof in* BVL.

4.2 Soundness

Now comes the tricky part. We want to map provable structures of BV to terminating computations of $\mathsf{PA}_{\mathsf{BV}}$ and, of course, we need a linguistic restriction on BV, which be determined by the grammar for expressions and action sequences of $\mathsf{PA}_{\mathsf{BV}}$. This restriction provides the legal set of structures we may use.

4.2.1 Definition The set $\mathcal{E}_{\mathsf{BV}}$ of *process structures* is the set of structures obtained by

$$P ::= \circ \mid \langle a; P\rangle \mid [P, P] \quad .$$

The function $\underline{\cdot}_{\mathsf{E}}$ maps the structures in $\mathcal{E}_{\mathsf{BV}}/=$ into expressions in $\mathcal{E}_{\mathsf{PA}}/=$ as follows:

$$\underline{\circ}_{\mathsf{E}} = \circ \quad ,$$

$$\underline{\langle a; P\rangle}_{\mathsf{E}} = a.\underline{P}_{\mathsf{E}} \quad ,$$

$$\underline{[P, Q]}_{\mathsf{E}} = \underline{P}_{\mathsf{E}} \mid \underline{Q}_{\mathsf{E}} \quad .$$

4.2.2 Theorem *Given the process structure P and the proof* $\dfrac{}{[P, \langle a_1; \ldots; a_n\rangle]}\,\mathsf{BVL}$, *for $n \geq 0$, there exists a computation $P_0 \xrightarrow{\vec{s}} \circ$, where $P_0 = \underline{P}_{\mathsf{E}}$ and $\underline{\vec{s}}_{\mathsf{S}} = \overline{\langle a_1; \ldots; a_n\rangle}$.*

Proof By induction on the size of P. If $P = \circ$ then P_0 is the computation. Otherwise, consider the given proof, where the bottommost instance of $\mathsf{ai}{\downarrow_{\mathsf{L}}}$ has been singled out:

$$\dfrac{\dfrac{\dfrac{}{S\{\circ\}}\,\mathsf{BVL}}{S[b,\bar{b}]^{\mathsf{L}}}\,\mathsf{ai}{\downarrow_{\mathsf{L}}}}{[P, \langle a_1; \ldots; a_n\rangle]}\,\Delta \Big\| \mathsf{BVL}\setminus\{\mathsf{ai}{\downarrow_{\mathsf{L}}}\}$$

Let us mark into Δ all occurrences of b and $\bar{b}$, as in $b^\bullet$ and $\bar{b}^\bullet$. Only two possibilities might occur:

1 One marked atom occurs in P an another occurs in $\langle a_1; \ldots; a_n\rangle$: In this case it must be $P = [\langle b^\bullet; P'\rangle, P'']$, for some P' and P'', and $a_1 = \bar{b}^\bullet$. Any other possibility would result in violating the condition of $S\{\ \}^{\mathsf{L}}$ being a right context (to see this, check carefully the rules of $\mathsf{BVL} \setminus \{\mathsf{ai}{\downarrow_{\mathsf{L}}}\}$ and see how they always respect seq orderings). Then replace all marked atoms by $\circ$, and remove all trivial occurrences of rule instances that result from this, including the $\mathsf{ai}{\downarrow_{\mathsf{L}}}$ instance. We still have a proof and $[P', P'']$ is a process structure, so we can apply the induction hypothesis on the proof

$$\dfrac{}{[P', P'', \langle a_2; \ldots; a_n\rangle]}\,\mathsf{BVL}$$

We get $b.\underline{P'}_{\mathsf{E}} \mid \underline{P''}_{\mathsf{E}} \xrightarrow{b} \underline{P'}_{\mathsf{E}} \mid \underline{P''}_{\mathsf{E}} \xrightarrow{\vec{s}'} \circ$, where $\underline{\vec{s}'}_{\mathsf{S}} = \overline{\langle a_2; \ldots; a_n\rangle}$.

2 Both marked atoms occur in P: It must be $P = [\langle b^\bullet; P'\rangle, \langle \bar{b}^\bullet; P''\rangle, P''']$, for the same reasons as above. By substituting $b^\bullet$ and $\bar{b}^\bullet$ by $\circ$, analogously as above, we can get, by induction hypothesis, the computation $b^\bullet.\underline{P'}_{\mathsf{E}} \mid \bar{b}^\bullet.\underline{P''}_{\mathsf{E}} \mid \underline{P'''}_{\mathsf{E}} \xrightarrow{\circ} \underline{P'}_{\mathsf{E}} \mid \underline{P''}_{\mathsf{E}} \mid \underline{P'''}_{\mathsf{E}} \xrightarrow{\vec{s}} \circ$.

□

This is the main result of this paper:

4.2.3 Corollary *The same statement of Theorem 4.2.2 holds for system* $\mathsf{SBV} \cup \{\circ\downarrow\}$ *instead of* BVL.

Proof It follows from Theorems 4.2.2, 2.6 and 3.8. □

The next example shows an application of the marking procedure and the extraction of the computation stepwise from the intermediate proofs. We start with a process structure $[a, \langle a; [\bar{a}, c]\rangle]$ and action sequence $a; c; \circ$. At each step the intermediate proof is obtained by removing marked occurrences and trivial

applications of rules; the associated computation is indicated below:

$$\mathsf{ai}{\downarrow_{\llcorner}}\dfrac{\mathsf{ai}{\downarrow_{\llcorner}}\dfrac{\mathsf{ai}{\downarrow_{\llcorner}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[a,\bar{a}]}}{\langle[c,\bar{c}];[a,\bar{a}]\rangle}}{\mathsf{q}{\downarrow}\dfrac{\langle[a^\bullet,\bar{a}^\bullet];[c,\bar{c}];[a,\bar{a}]\rangle}{\mathsf{q}{\downarrow}\dfrac{\langle[a^\bullet,\bar{a}^\bullet];[a,\bar{a},c,\bar{c}]\rangle}{\mathsf{q}{\downarrow}\dfrac{[a,\langle[a^\bullet,\bar{a}^\bullet];[\bar{a},c,\bar{c}]\rangle]}{[a,\langle a^\bullet;[\bar{a},c]\rangle,\langle\bar{a}^\bullet;\bar{c}\rangle]}}}} \;\rightarrow\; \mathsf{ai}{\downarrow_{\llcorner}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[a,\bar{a}]}}{\mathsf{q}{\downarrow}\dfrac{\langle[c^\bullet,\bar{c}^\bullet];[a,\bar{a}]\rangle}{[a,\bar{a},c^\bullet,\bar{c}^\bullet]}} \;\rightarrow\; \mathsf{ai}{\downarrow_{\llcorner}}\dfrac{\circ\downarrow\dfrac{}{\circ}}{[a^\bullet,\bar{a}^\bullet]} \;\rightarrow\; \circ\downarrow\dfrac{}{\circ} \;\;;$$

$$a.\circ \mid a.(\bar{a}.\circ \mid c.\circ) \xrightarrow{a} a.\circ \mid \bar{a}.\circ \mid c.\circ \xrightarrow{c} a.\circ \mid \bar{a}.\circ \xrightarrow{\circ} \circ \quad .$$

4.3 Comments

Let us summarise the results presented above.

1 Every computation can be put in an easy correspondence to a derivation in SBV, which essentially mimics its behaviour by way of seq and left atomic interaction rules. This result is certainly not unexpected, given that prefixing in CCS is subsumed by the more general form of ordering by seq that we have in SBV.

2 Every proof in SBV$\cup\{\circ\downarrow\}$ over a process structure corresponds to a terminating computation. This result is much harder than 1 and it was not obvious. The difficulty, of course, is in the fact that the logical system could perform in principle many more derivations than just those corresponding to computations. It actually does so, but now we know that for each of them there is a terminating computation. The source for the potential applications of this work stems from this result.

The use of point 2, i.e., soundness of SBV with respect to our process algebra, should be the following. BVL, or better yet a further, equivalent restriction along the lines of Miller's uniform proofs, faithfully performs our computations. Here we only have exactly the nondeterminism inherent in the operational semantics of our process algebra. But we can also use the more powerful systems that we know are equivalent to BVL. If we remove the restriction on atomic interactions to be left, as in BV, we can perform communications in any order we like: the time structure of the process is still retained by the logic, but we are not committed to the execution time.

Further, we can add the admissible rule $\mathsf{q}{\uparrow}$: its use allows strongly to limit nondeterminism, so making choices that, if well guided, could reduce dramatically the search space for, say, a verification tool. In addition we can also allow cut rules, in their various forms. These are notoriously extremely effective in reducing exponentially the search space for proofs, provided one knows exactly which structure to use in cuts. As Theorems 2.6 and 3.8 point out, several different systems

are equivalent to BVL. Extending our system to SNEL, an extension of SBV with exponentials studied in [9], will bring in an even larger range of possibilities.

The reader might have noticed that there is little use of the switch rule s when dealing with process structures. This is due to the fact that process structures do not contain copars. The rule s is essential in at least two scenarios:

1 When using the q↑ and cut rules.

2 In the presence of recursion. As I said already, in a coming extension to our system it will be possible to deal with fixpoint constructions. Very briefly, we will deal with structures like $?(\bar{P}, Q)$, which specifies the unlimited possibility of rewriting process P by process Q. For this construct to work, copar and s are essential.

In my opinion, the only really significant challenge remaining in order to capture exactly CCS in a logical system is coping with the silent transition τ. Its algebraic behaviour is rather odd, so I would expect a correspondingly odd logical system, if logical purity is to be maintained. A more sensible approach could be either to give up to perfect correspondence to CCS, or modeling τ by axioms and then studying the impact of this axiomatisation on the properties of interest (cut elimination, mainly).

5 Conclusions

This paper intends to be a contribution to the principled design of logic languages for concurrency. We examined a stripped down version of CCS, having only prefixing and parallel composition, called $\mathsf{PA}_{\mathsf{BV}}$. This very simple process algebra presents a significant challenge to its purely logical account in the proof search paradigm, because of its commutative/non-commutative nature. To the best of my knowledge, the only formal system presenting at the same time commutative, non-commutative and linear operators, necessary to give account of the algebraic nature of $\mathsf{PA}_{\mathsf{BV}}$, is system SBV. Still, there is a nontrivial mismatch, in SBV, between its form of sequentiality and CCS's one.

In this paper I showed how to close this gap, through a purely logical restriction of SBV, and I showed how to represent $\mathsf{PA}_{\mathsf{BV}}$ in SBV. I argued that this process algebra can be extended to a Turing-equivalent one, comprising much of CCS, while still maintaining a perfect correspondence to the purely logical formal system studied in [9]. Further steps, to enhance expressivity, are possible in even more extended formal systems, by way of additives, along the lines of [15].

References

[1] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, June 1994.

[2] Jean-Marc Andreoli and Remo Pareschi. Linear Objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.

[3] Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[4] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[5] Alessio Guglielmi. Concurrency and plan generation in a logic programming language with a sequential operator. In P. Van Hentenryck, editor, *Logic Programming, 11th International Conference, S. Margherita Ligure, Italy*, pages 240–254. The MIT Press, 1994.

[6] Alessio Guglielmi. Sequentiality by linear implication and universal quantification. In Jörg Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 160–174. Springer-Verlag, 1995.

[7] Alessio Guglielmi. A system of interaction and order. Technical Report WV-01-01, Dresden University of Technology, 2001. On the web at: http://www.ki.inf.tu-dresden.de/~guglielm/Research/Gug/Gug.pdf.

[8] Alessio Guglielmi and Lutz Straßburger. Non-commutativity and MELL in the calculus of structures. In L. Fribourg, editor, *CSL 2001*, volume 2142 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 2001. On the web at: http://www.ki.inf.tu-dresden.de/~guglielm/Research/GugStra/GugStra.pdf.

[9] Alessio Guglielmi and Lutz Straßburger. A non-commutative extension of MELL in the calculus of structures. Technical Report WV-02-03, Dresden University of Technology, 2002. On the web at: http://www.ki.inf.tu-dresden.de/~guglielm/Research/NEL/NELbig.pdf, submitted.

[10] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, May 1994.

[11] Dale Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *1992 Workshop on Extensions to Logic Programming*, volume 660 of *Lecture Notes in Computer Science*, pages 242–265. Springer-Verlag, 1993.

[12] Dale Miller. Forum: A multiple-conclusion specification logic. *Theoretical Computer Science*, 165:201–232, 1996.

[13] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[14] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[15] Lutz Straßburger. A local system for linear logic. Technical Report WV-02-01, Dresden University of Technology, 2002. On the web at: http://www.ki.inf.tu-dresden.de/~lutz/lls.pdf.

[16] Alwen Fernanto Tiu. Properties of a logical system in the calculus of structures. Technical Report WV-01-06, Dresden University of Technology, 2001. On the web at: http://www.cse.psu.edu/~tiu/thesisc.pdf.

Typeset with Xy-pic