TR93-18*
Applications of Linear Logic to Computation: An Overview

Vladimir Alexiev† <vladimir@cs.ualberta.ca>
Department of Computing Science, 615 GSB
University of Alberta
Edmonton, Alberta T6G 2H1
Canada

December 1993‡

Abstract

This paper is an overview of existing applications of Linear Logic (LL) to issues of computation. After a substantial introduction to LL, it discusses the implications of LL to functional programming, logic programming, concurrent and object-oriented programming and some other applications of LL, like semantics of negation in LP, non-monotonic issues in AI planning, *etc.* Although the overview covers pretty much the state-of-the-art in this area, by necessity many of the works are only mentioned and referenced, but not discussed in any considerable detail. The paper does not presuppose any previous exposition to LL, and is addressed more to computer scientists (probably with a theoretical inclination) than to logicians. The paper contains over 140 references, of which some 80 are about applications of LL.

1 Linear Logic

Linear Logic (LL) was introduced in 1987 by Girard [62]. From the very beginning it was recognized as relevant to issues of computation (especially concurrency and state change), an evidence of which is that the paper appeared not in a journal of logic, but in *Theoretical Computer Science*. Also, it was recognized as a novel and important contribution: it was allotted a whole issue of the journal (about 100 pages) and was published with the following caveat "We warn the reader that because of the length and novelty of this paper, it was not passed through the normal review process".

The paper is organized as follows: this section provides an introduction to LL (no previous exposure to LL is needed). For other short introductions to LL, including intuitive motivations and full presentation of the sequent system, see *e.g.* [109, 148, 154]. A very complete yet brief reference of LL theory is [136]. Unfortunately it is in Japanese, but one can still read the formulas. No references are included.

Subsequent sections discuss in turn applications of LL in various areas of computing: functional programming, logic programming, concurrent and object-oriented programming, deductive planning.

A comprehensive (although somewhat out of date) coverage of both proof theory and semantics of LL is provided in the book by Anne Troelstra [151] (well-written and easy to read; above all the only book on LL to date), and a good book on proof-theoretical issues and their computational interpretation is *Proofs and Types* [73]. A WWW page about LL is maintained by Lincoln, `http://www.csl.sri.com/linear/sri-csl-ll.html`.

*Available from `ftp.cs.ualberta.ca: pub/TechReports/TR93-18`, file `TR93-18.ps.gz` or `TR93-18.ps.Z`. Also in *Bulletin of the IGPL* 2(1), March 1994. Comments are most welcome.

‡Last revision September 1994

1.1 Formulas as Resources

Classical Logic (CL) is not well suited for reasoning about a dynamic world, because it was not designed for such a task. The methodological setting of CL is "Platonic": there is an external (ideal or real) static world, and logical formulas try to describe and mirror it. Formulas are interpreted as eternal truth values which, once established, last forever and can be used again and again in derivations of other formulas. This is the ultimate source of a number of problems with the application of CL in areas which need the notion of state or non-monotonicity. Girard refers very critically to the existing approaches, using expressions like:

> The continuing shame referred to as The Frame Problem...[65]
>
> ...Reiter's Default Logic, better called "Deficient Logic", because it has been long known that procedures based on the impossibility to prove a formula cannot possibly be efficient. [65]
>
> The consideration [in non-monotonic logics] of weird classical models definitely cuts the bridge with formal systems but also with informal reasoning. [70]

Although clearly outspoken, these claims are not without a justification. The two main problems with most extensions of CL are

- The addition of "non-logical" inference rules usually spoils the proof theory of the formalism. To the contrary, LL has a clean proof theory which actually serves as the basis of its applications to computation.
- Since these formalisms simply extend CL, without addressing the problem of the staticity of CL, they end up reasoning about successive changed states of *whole theories*, which usually makes them computationally infeasible. So the best they can serve is as *descriptive* formalisms, rather then actually *programmable* approaches.

In LL formulas are treated as resources which are produced and consumed. Every formula should be used exactly once in a derivation: neither duplication (reuse) nor discarding formulas is permitted. As Phil Wadler describes it vividly,

> LL is a logic for the 90s. If you lean to the right, view it as a logic of realistic accounting: no more free assumptions. If you lean to the left, view it as an eco-logic: resources must be conserved. [154]

The technical means by which this is achieved is that two of the "structural" rules of CL

$$\frac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\ \mathsf{Contraction} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\ \mathsf{Weakening}$$

are abolished (thus sometimes LL is referred to as a "substructural logic"). Under a bottom-up reading (corresponding to goal-directed proof search), these rules correspond to duplication and discarding of formulas respectively. While in CL sequents are composed of "expandable" *sets* of formulas (meaning that any formula can be added and that duplicates are merged), in LL sequents are composed of *multisets* of formulas, *i.e.* the number of occurrences does matter. Since the third structural rule

$$\frac{\Gamma_1, \varphi, \psi, \Gamma_2 \vdash \Delta}{\Gamma_1, \psi, \varphi, \Gamma_2 \vdash \Delta}\ \mathsf{Exchange}$$

is retained, the order of formulas does not matter. However there exists *Non-Commutative LL* [5, 4, 155] (still rather young) which also abolishes Exchange and in which sequents are composed of *sequences* of formulas. Even more exotic varieties exist, *e.g.* Cyclic LL (with applications to Quantum Mechanics) [155], in which only cyclic rotations of the formulas are allowed.

LL is a refinement of CL which revises the very foundation of the logic. This is appealing compared to most existing approaches which extend CL with additional constructs (multiple worlds, additional non-logical inference rules, *etc*), because is seems less probable to adapt an unsuitable formalism by extension

than by revision. However the seemingly simple revisions which LL introduces (abolition of the structural rules) lead to radical changes of the logic, which make it more complex and which are described below.

It should not be unexpected that conjunction and disjunction cease to be idempotent (indeed idempotence is incompatible with our goal to regard formulas as resources). Further, LL introduces two kinds of connectives, *multiplicative* and *additive*, none of which is quite the same as the classical connectives (although additives *are* idempotent). (The names come from a semantic theory of LL (coherence spaces), where multiplicatives are modeled by the Cartesian product of the two operands, while additives are modeled by the direct sum (disjoint union).) The connectives are shown below, together with their names and neutral (identity) elements (*e.g.* $\varphi \otimes \mathbf{1} = \varphi$).

	multiplicative			additive			exponential	
conjunction	$\otimes$	times,	$\mathbf{1}$	$\&$	with,	$\top$	!	ofCourse
disjunction	$\parr$	par,	$\bot$	$\oplus$	plus,	$\mathbf{0}$	?	whyNot
implication	$\multimap$	lollipop						

Notes:

1. The symbol $\parr$ above is supposed to look like an upside-down ampersand.
2. Exponentials are explained below.
3. LL has also quantifiers $\forall$ and $\exists$, which are pretty much the same as in CL.
4. In my opinion, it is unfortunate that Girard has chosen the above notation, because one would expect that dual connectives will have similar symbols, which is not the case (his reasons were that $\otimes$ distributes over $\oplus$, which gives the nicely looking $(\varphi \oplus \psi) \otimes \xi = (\varphi \otimes \xi) \oplus (\psi \otimes \xi)$). Troelstra in [151] proposes the alternative notation

	multiplicative	additive
conjunction	$\star, 1$	$\sqcap, \top$
disjunction	$+, 0$	$\sqcup, \bot$

   which however hasn't enjoyed wide popularity.

The distinction multiplicative/additive stems from the fact that when sequents are multisets, inference rules can involve sharing of contexts (the passive formulas in the rule) in various degrees (in CL, Weakening and Contraction do not leave the possibility for such distinctions). LL considers two modes of sharing:

- No sharing at all: contexts are completely separate and are concatenated in the conclusion. This corresponds to multiplicatives, which in [73, Appendix B] are also called "cumulative" connectives. Relevantists call similar connectives "intensional", and Avron in [26] calls them "internal" because they correspond to the commas inside a sequent.

$$\frac{\Gamma_1 \vdash \varphi, \Delta_1 \quad \Gamma_2 \vdash \psi, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \varphi \otimes \psi, \Delta_1, \Delta_2} \otimes \mathcal{R} \qquad \frac{\Gamma_1, \varphi \vdash \Delta_1 \quad \Gamma_2, \psi \vdash \Delta_2}{\Gamma_1, \Gamma_2, \varphi \parr \psi \vdash \Delta_1, \Delta_2} \parr \mathcal{L}$$

- Complete sharing: contexts are required to be the same. This corresponds to additives, which are also called "identifying", "extensional" or "combining" connectives.

$$\frac{\Gamma \vdash \varphi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \,\&\, \psi, \Delta} \,\&\, \mathcal{R} \qquad \frac{\Gamma, \varphi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \oplus \psi \vdash \Delta} \oplus \mathcal{L}$$

Please note that in CL the above multiplicative/additive rules are equivalent (derivable from one another) because sequents are "expandable" sets.

The comma (multiset union) in sequents is multiplicative and is interpreted as conjunction on the left and as disjunction on the right. Linear implication is the internalisation of entailment in the sequent calculus, thus $\Gamma \vdash \Delta$ iff $(\otimes\, \Gamma) \multimap (\parr\, \Delta)$. In terms of inference rules,

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \otimes \psi \vdash \Delta} \otimes \mathcal{L} \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \parr \psi, \Delta} \parr \mathcal{R}$$

To complete the collection, there are a couple more rules for the additives:

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \varphi \,\&\, \psi \vdash \Delta} \,\&\, \mathcal{L}_1 \qquad \frac{\Gamma, \psi \vdash \Delta}{\Gamma, \varphi \,\&\, \psi \vdash \Delta} \,\&\, \mathcal{L}_2 \qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \varphi \oplus \psi, \Delta} \oplus \mathcal{R}_1 \qquad \frac{\Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \oplus \psi, \Delta} \oplus \mathcal{R}_2$$

The intuitive meaning of the connectives introduced so far is: $\varphi \otimes \psi$ means to have *both* data φ and ψ; $\varphi \,\&\, \psi$ means to have *a choice* between φ and ψ (observe in $\&\, \mathcal{L}_1$ and $\&\, \mathcal{L}_2$ that if we can prove $\Gamma, \varphi \vdash \Delta$ or $\Gamma, \psi \vdash \Delta$, we can pick the corresponding rule and prove $\Gamma, \varphi \,\&\, \psi \vdash \Delta$); $\varphi \oplus \psi$ means we have one of φ and ψ, but the choice is *external* (in $\oplus\, \mathcal{L}$, we have to ensure we can prove both $\Gamma, \varphi \vdash \Delta$ and $\Gamma, \psi \vdash \Delta$ because the choice is not ours); $\varphi \multimap \psi$ means that the datum φ is consumed (used exactly once) to produce the datum ψ.

1.2 Negation

Linear negation is denoted $(\cdot)^\perp$. It is introduced *syntactically*, starting from a set of propositional letters $\{p, q, \ldots\}$ *and* their negations $\{p^\perp, q^\perp, \ldots\}$ and defining negations of complex formulas by DeMorgan's dualities, $(\varphi \otimes \psi)^\perp = \varphi^\perp \parr \psi^\perp$ *etc.* As usual, implication can be defined through negation and (multiplicative) disjunction, $\varphi \multimap \psi = \varphi^\perp \parr \psi$, and regarded as shortcut notation.

Unlike Negation As Failure in logic programming, linear negation is involutive ($\varphi^{\perp\perp} = \varphi$).[1] Yet unlike classical negation, linear negation is constructive. This is possible because linear negation does not impose an isomorphism on a semantic structure for LL (while double negation is certainly an isomorphism). In the usual models of CL, Boolean lattices, negation corresponds to turning the lattice upside down and imposes an isomorphism.

Because of the involutiveness of negation, sometimes LL is presented in a one-sided (usually right-sided) sequent calculus, transforming $\Gamma \vdash \Delta$ to $\vdash \Gamma^\perp, \Delta$. This approach reduces in half the number of rules needed.

1.3 Exponentials

LL does not abolish Weakening and Contraction altogether, because this would render the logic too weak. Instead, their use is reintroduced in a "controlled environment" by explicitly marking the formulas for which Weakening and Contraction are applicable. Two unary connectives are introduced, ofCourse ! and whyNot ?. They are called *exponential* connectives, *modalities*, or *storage* operators and are dual to each other. ! is used for assumptions (antecedents in a sequent; resources) and ? is used for conclusions (succedents; results). The inference rules for ! are

$$\frac{\Gamma, !\varphi, !\varphi \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\ \text{Contraction} \qquad \frac{\Gamma \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\ \text{Weakening}$$

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, !\varphi \vdash \Delta}\ \text{Dereliction} \qquad \frac{!\Gamma \vdash \varphi, ?\Delta}{!\Gamma \vdash !\varphi, ?\Delta}\ \text{Promotion}$$

(!Γ and ?Δ in Promotion mean that every formula in Γ and Δ is exponential.)

Linear (non-exponential) formulas are similar to tokens in a dataflow machine or signals over an electric wire: they are created and immediately consumed. An exponential formula corresponds to a datum stored in memory. With this interpretation (and with a bottom-up reading), the first three rules above can be

[1] de Paiva considers non-involutive negation in [51, 52].

thought of as the actions of *duplication, discarding* and *reading* respectively. Promotion corresponds to *storing* a datum which is produced only from stored data (the Γ before Δ play the rôle of ! for formulas on the right side, which is justified by the duality of Γ and !). It can be proved that

$$!\varphi = \mathbf{1} \,\&\, \varphi \,\&\, (!\varphi \otimes !\varphi)$$

so indeed the three things one can do with a stored datum are to discard it ($\mathbf{1}$), to use (read) it (φ) or to duplicate it ($!\varphi \otimes !\varphi$), and the choice is ours.[2]

1.4 Proof Theory

In CL the process of deduction is only a means to establishing truth and has no importance by itself. Therefore proofs are not regarded as "first-class citizens". This has adverse consequences for the computational interpretations of CL, both for automatic proof search and for its use as a programming language. So CL is not very constructive. A number of approaches to make CL more constructive exist, based on both philosophical and computational grounds.

- An easy way to make CL more tractable is to consider a limited fragment of it. The most important example of this are *Horn clauses* and *resolution*. However this reduces the expressive power of logic, which may be unacceptable for certain applications. Attempts to restore the expressive power are disjunctive logic programs, programs with negation, *etc.*

- The traditional approach stemming from the desire to make mathematics more constructive is *intuitionism*. In this approach, proofs are considered as effective functions from the assumptions to the conclusion. This approach has important applications in Functional Programming through the Curry-Howard isomorphism (see Section 2). There exists *Intuitionistic LL* in which only one formula is allowed in the right-hand side of a sequent.

- A number of other substructural logics have been studied, for example affine (direct) logic (no Contraction), relevance (pertinent) logic (no Weakening) BCK logic, *etc.* (for references see *e.g.* [58]). Relevance logic has stemmed from the desire to exclude vacuous uses of implication (*e.g.* "If π is 3 then I am a tzar" is a true, but vacuous sentence). Since discarding of assumptions is not allowed, only relevant entailments are provable.

LL has a rich and very well-developed proof theory. Furthermore, its proof theory is applied directly for obtaining computational interpretations of the logic, unlike the case with CL, where the proof theory is largely "external" to the logic. One problem with proof search in general is that not only there are very many ways of proving the same proposition, but also that many of these proofs differ only in inessential details, like the exact order of rule applications. Technically speaking, proofs possess rich symmetries and permutabilities. These permutabilities need to be discovered and exploited, because otherwise the search space gets unmanageably large and a program may spend years generating variants which are essentially the same. Girard speaks of "the bureaucracy of taxonomy", having in mind that all these different ways of saying the same thing, if not taken care of, can get over and render a logic programming system unusable.

A number of features of LL have positive implications to the proof search process. One is that its richer suite of connectives conveys better the intended usage of formulas. To a certain extent, the proof is encoded directly in the formula to be proved. Andreoli and Pareschi [12, p.27 ff] speak of "syntax-directed proof search".

[2] As pointed out to me by Martí-Oliet, more precisely the equality symbol above should be replaced by $\circ\!-\!\circ$ (linear implication in both directions). He says "there is no notion of equality in LL. It is true that a double linear implication can be proved, but double implication is not the same as equality. ... As I think Girard himself says in one of his papers, the rules for the modalities do not determine them completely."

Another is that LL possesses the very desirable *Cut Elimination* property. The deduction rule Cut (not to be confused with PROLOG's !) is formulated in either of the following two forms

$$\frac{\Gamma_1 \vdash \Delta_1, \varphi \quad \varphi, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \mathsf{Cut} \qquad (\text{or } \frac{\Gamma_1 \vdash \Delta_1, \varphi \quad \Gamma_2 \vdash \Delta_2, \varphi^\perp}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2})$$

Cut elimination states that any proof using Cut can be transformed to one which does not use it (so Cut need not be present in the logic, but can be adopted as a derived rule). (Note: this holds only if there are no additional non-logical axioms. In the presence of non-logical (proper) axiom, *e.g.* the predicate definitions in a logic program, an analogous property states that Cut can be pushed all the way up to the leaves of the proof tree where one of its premises is a proper axiom.) Cut elimination for LL has been proved by Girard [62, 73] by largely the same method as Gentzen's *Hauptsatz* which proves it for CL.

Cut elimination is important both for reducing the space of proofs (only cut-free proofs will be searched for, see Section 3), and because the process of cut elimination (proof normalization) can be regarded as computation in a term rewriting system (Section 2). Logics which do not possess this property (not to mention systems contaminated with non-logical inference rules, in which it may be even impossible to formulate it) forfeit these possibilities.

Closely related is the *Subformula property* which says that if a formula is provable, there exists a proof for it in which all formula occurrence are subformulas of the conclusion (and the assumptions, if any). It is obviously useful for proof search, because it limits the set of formulas to be tried.

A construction which largely avoids the "bureaucracy of taxonomy" are *Proof Nets* (for some of the developments see [62, 30, 28, 49, 77, 18, 35], for a generalization see [100]). Proof nets were invented by Girard for LL, but they are not that peculiar to LL and can hopefully be adapted for other logics as well. A proof net is a graph with nodes marked by formulas (not sequents) and edges linking the premises of a rule occurrence to the conclusion. These links are directed, which is usually represented by placing the conclusion below the premises. All nodes without outgoing (downward) edges are the conclusions of the net, so a derivation can have more than one conclusion (of course, they can be gathered together by $\wp$). There are also *axiom links* connecting p and $p^\perp$, corresponding to axiom sequents $\vdash p, p^\perp$, which are not directed.

Some of the order (sequentiality) in the presentation of a proof is unavoidable; for example two introduction rules enlarging the same formula come necessarily in a particular order. But conventional proof trees introduce in addition lots of inessential sequentiality (though not that much as a linear presentation of the proof would do), because two introductions in the same branch of the tree have to be assigned a certain order, no matter whether they operate on the same formula or not (in other words, whether the order is essential or not). This is not the case with proof nets, because multiple conclusions are allowed and because dual propositions in an axiom need not be stuck together in the same node (sequent), but only need to be connected by an axiom link, which itself is undirected. Girard says "Proof nets are well-developed and work perfectly for the multiplicative fragment; work for full LL is underway." Some developments for the quantifiers are in [64, 69].

Proof-theoretic considerations have inspired Girard to formulate a programme called *Geometry of Interaction* [65, 66, 67, 3]. Briefly stated, it is the desire to obtain a good formalization of concurrency, analogously to what λ-calculus has achieved for sequential computation. The introduction of unnecessary sequentialization should be avoided when possible. Girard was not perfectly happy with denotational semantics, because it describes the behavior of finite programs by infinite static structures. Geometry of interaction tries to capture this behavior using finite dynamic structures. Girard compares denotational semantics to the part of mechanics called Statics, and desires what would be the Dynamics of computation.

Proof-theoretical results for LL and other resource logics can be found in Dirk Roorda's thesis [145].

1.5 Complexity of LL

Fortunately, we have a quite good understanding of the computational complexity of various fragments of LL, mainly due to Patrick Lincoln [108] and Max Kanovich. Unfortunately, this complexity is rather high.

- Propositional (quantifier-free) LL is undecidable, unlike Propositional CL. This is mainly due to the exponentials and is proven by encoding the behavior of certain simple counting machines [112, 113].
- Multiplicative-exponential LL (no additives and quantifiers) is EXPSPACE-hard, which follows from an encoding of Petri nets in that fragment [120, 122] (see Section 4.1).
- First order multiplicative-additive LL (no exponentials) is NEXPTIME-hard [114].
- Multiplicative-additive LL (no exponentials and quantifiers) is PSPACE-complete [112, 113].
- Multiplicative LL and Horn LL are NP-complete, which is proven by reduction to the 3-Partition problem [91, 92].
- Even severely limited parts of the fragments mentioned in the preceding item still remain NP-complete [115]. "Severely limited" means constant-only (no propositional letters at all, but the multiplicative/additive neutral elements are part of the language) or constant free, with only two or three predicate letters.

After seeing these complexity results, one inevitably asks oneself "So what can be the practical use of LLΓ Yet another fancy theoretical development." The following considerations are beneficial for LL in this respect:

- Usually the other name of "too complex" is "very expressive". Thus if we need an expressive logic (*e.g.* a logic for software specification, CASE or high-level modeling), we may be willing to accept the high complexity.
- Much of the (undecidable) complexity of LL comes from the introduction of the exponentials in order to regain the full power of CL. But "the gist" of LL, namely the concept that formulas are resources, is already present in the exponential-free fragment. For some applications this fragment can be sufficient, *e.g.* interconnection specifications for concurrent processes based on proof nets. Even more restricted systems —without negation— can be of practical interest, *e.g.* Lambek Calculus [104] for Natural Language processing and Action Logic [138] as a model of concurrency.
- The complexity results of the previous subsection do not account for differences which are practically important (but which do not amount simply to a different suite of connectives). For example, Intuitionistic LL has the same complexity as full LL, yet ILL corresponds to a perfectly feasible refinement of the λ-calculus. This fragment is used as a FP language (Section 2), proofs correspond to functional programs, and computation (evaluation) corresponds to proof reduction (as opposed to proof search). Thus undecidability (the impossibility of finding proofs) does not matter. On the other hand, when LL is used for LP (Section 3), various restrictions are placed both on the set of allowed formulas and on the space of proofs to be searched, in order to make the search more tractable.
- A remarkable work of Girard, Scedrov and Scott [74] provides a logical formalization of the class of functions computable in polynomial time using *Bounded LL*, where exponentials are indexed by natural numbers $!_n$ and Γ_n and can be duplicated only the specified number of times. The characteristic equation of ofCourse becomes

$$!_n\varphi = \&_{i=0}^{n}\, \varphi^i, \qquad \varphi^i = \underbrace{\varphi \otimes \cdots \otimes \varphi}_{i \text{ times}}.$$

1.6 Semantics of LL

If you think by now that LL is way too complex compared to CL, you will not get any happier when you hear about semantics. The complexity result about constant-only propositional multiplicative LL [115] mentioned in the last subsection means that no truth-table interpretation of LL is possible (unless P=NP). Also, no

simpleminded Tarski-style interpretation of the flavor "$M \models \varphi \wedge \psi$ iff $M \models \varphi$ and $M \models \psi$" is known for full LL. This is connected to the fact that linear negation does not impose an isomorphism on the semantic structure.

A notable exception to the above statement is the semantics proposed by Miller and Hodas [85] for their LP language based on LL (see Section 3). However, the semantics there is conceived for the pragmatic purpose of providing a canonical (intended) model for the language and it is not completely general, because it lacks negation (although soundness and completeness theorems for the corresponding fragment are proved). The semantics consists of a set of Kripke interpretations K_r (all defined on the same frame of worlds $\langle \mathcal{W}, \leq \rangle$) where $r \in R$ and $\langle R, +, 0 \rangle$ is a commutative monoid of "resources", corresponding to the multiset treatment of resources in LL. K_0 is intended to be the intuitionistic interpretation (no resources at all). The two semantic clauses below clearly show the Tarskian style of the interpretation and the distinction additive/multiplicative. Satisfaction of a formula under an interpretation K_r in a possible world w is defined inductively:

- $K_r, w \models \varphi \,\&\, \psi$ iff $K_r, w \models \varphi$ and $K_r, w \models \psi$.
- $K_r, w \models \varphi \otimes \psi$ iff $K_{r1}, w \models \varphi$ and $K_{r2}, w \models \psi$ for some resources $r1, r2 \in R$ such that $r1 + r2 = r$.

The accessibility relation between worlds is intended to cater for the interpretation of linear implication, and in order for this to be possible, it is required to be a partial order. Also, all K_r are required to be monotonic on worlds for atomic formulas, that is $w \leq w'$ implies $K_r(w) \subseteq K_r(w')$ (in fact $w \leq w'$ and $K_r, w \models \varphi$ implies $K_r, w' \models \varphi$ for *any* formula φ). Then the clause for $\multimap$ is

- $K_r, w \models \varphi \multimap \psi$ iff $K_{r'}, w' \models \varphi$ implies $K_{r+r'}, w' \models \psi$ for every $r' \in R$ and $w' \geq w$.

A similar notion appears in an interesting paper by Chau [44] defining a modal and substructural (*viz.* catenation) logic in terms of Gabbay's Labelled Deductive Systems. This semantics has at least the advantage that it is maybe the simplest one proposed for LL.

In his initial paper [62] Girard introduced two semantic frameworks. One is the *phase space* semantics. It has an algebraic origin, the domains of interpretation being algebraic structures, later called Girard structures [59]. They are commutative monoids P with a distinguished subset called the set of "antiphases" $\bot$. For a set $X \subseteq P$, its dual is defined as $X^\bot = \{y \mid \forall x \in X.\ xy \in \bot\}$. Then $(\cdot)^{\bot\bot}$ is monotone, inflationary ($X \subseteq X^{\bot\bot}$) and idempotent, *i.e.* a closure operation. The closed sets (ones with $X = X^{\bot\bot}$) are called "facts" and they are the entities which interpret LL formulas. It can be proved that the facts considered under set inclusion form a complete lattice and the monoid operation commutes with the lattice operations. The lattice's $\leq$ relation is used to interpret provability; theorems are characterized by $\mathbf{1} \leq \varphi$. Tensor is interpreted as $X \otimes Y = (XY)^{\bot\bot}$ and so on.

The other one is the *coherence space* semantics. First we define "webs" as undirected graphs (sets equipped with a reflexive symmetric relation $\frown$ called "coherence"). Coherence spaces are cliques of the web (subsets where every pair of elements is coherent), regarded as complete partial orders under set inclusion. Every coherence space interprets one formula. Given two spaces A and B, a multiplicative combination $A \otimes B$, $A \parr B$ or $A \multimap B$ is interpreted by taking the Cartesian product of the two corresponding webs (that is, $|A \otimes B| = |A \parr B| = |A \multimap B| = |A| \times |B|$) and stipulating certain (intuitive) conditions on coherence (*e.g.* for $\otimes$, $(a_1, b_1) \frown (a_2, b_2)$ is defined as $a_1 \frown a_2$ and $b_1 \frown b_2$). Additive combinations are interpreted by taking the disjoint union of the corresponding webs ($|A \,\&\, B| = |A \oplus B| = |A| + |B| = \{(0, a) \mid a \in |A|\} \cup \{(1, b) \mid b \in |B|\}$), again with certain coherence conditions. Exponential is interpreted by the set of all finite coherent subsets of the operand $|A!| = \wp(|A|)$. Negation does not play such important rôle here as in the phase semantics.

An important class of semantics are *categorial semantics*. They remove the unnecessary concreteness present in the other semantics and make a connection to a rich body of results existing in category theory. For example, the $\star$-autonomous categories which Seely has used as models of LL [150, 27] were studied by Barr, "in a truly categorial spirit", as early as 1975. Categorial notions (like adjoints, monads, Kleisli

categories) give useful insights for the design of logics and interpretations, as well as programming languages and abstract machines. Another semantics in this class are dePaiva's Dialectica categories [51, 52].

A recent new development which has important applications to models of concurrency based on LL are the *game semantics* [34, 103, 2]. A LL proof is interpreted as a history-independent winning strategy in a two-opponent zero-sum game.

Something which has been long missing for LL is the *Kripke-style* semantics developed by Allwein and Dunn [8, 7]. It is fairly general in that it starts from non-commutative (and even non-associative) operators and then commutative LL can be obtained by imposing additional conditions.

2 LL and Functional Programming

Probably the most developed area of applications of LL is functional programming. A fundamental idea here is the *Curry-Howard interpretation* (also variously called "isomorphism", or the *formulas-as-types paradigm*) [89]. It states that there is a correspondence between the proofs of an intuitionistic logic system and computations in a functional programming language. This has its basis in the early approach of intuitionists that a proof should be an effective function which given the assumptions, produces the conclusions. Thus, the proof of a conjunction is a pair of the proofs of the conjuncts, the proof of a disjunction is a similar pair plus a selection function which says for each concrete pair of disjuncts which proof to pursue, the proof of an implication is a function which transforms any proof of the antecedent to a proof for the consequent, *etc.*

The term "formulas-as-types" comes from the following idea. (It is easier to understand it for a system with introduction and elimination rules, *i.e.* natural deduction.) We can take the inference rules of a logic and "adorn" them with terms (this is also called *term assignment*):

$$\frac{\Gamma \vdash s : \varphi \quad \Gamma \vdash t : \psi}{\Gamma \vdash \langle s,t \rangle : \varphi \wedge \psi} \wedge\mathcal{I} \qquad \frac{\Gamma \vdash u : \varphi \wedge \psi}{\Gamma \vdash \pi_1(u) : \varphi} \wedge\mathcal{E}_1 \qquad \frac{\Gamma \vdash u : \varphi \wedge \psi}{\Gamma \vdash \pi_2(u) : \psi} \wedge\mathcal{E}_2$$

$$\frac{\Gamma, x : \varphi \vdash t : \psi}{\Gamma \vdash \lambda x.t : \varphi \rightarrow \psi} \rightarrow\mathcal{I} \qquad \frac{\Gamma \vdash s : \varphi \rightarrow \psi \quad \Gamma \vdash t : \varphi}{\Gamma \vdash st : \psi} \rightarrow\mathcal{E}$$

$$\frac{\Gamma \vdash s : \varphi \quad x : \varphi, \Delta \vdash t : \psi}{\Gamma, \Delta \vdash t[s/x] : \psi} \ \mathsf{Cut}$$

(where $\langle s,t \rangle$ is the pair of s and t, $\pi_1(u)$ is the left projection of u, $\lambda x.t$ is abstraction and st is application). Thus formulas play the rôle of types, and the proof system plays the rôle of a type inference calculus (for typed λ-calculus with products and sums). Introduction rules correspond to constructors, elimination rules correspond to destructors, Cut corresponds to substitution. When an elimination rule appears immediately below a corresponding introduction rule, we have a redex which can be rewritten to its reduct, thus eliminating the useless "detour" in the proof:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1 \\ \vdots \\ \Gamma, x : \varphi \vdash s : \psi\end{array}}{\Gamma \vdash \lambda x.s : \varphi \rightarrow \psi} \rightarrow\mathcal{I} \qquad \begin{array}{c}\Pi_2 \\ \vdots \\ \Delta \vdash t : \psi\end{array}}{\Gamma, \Delta \vdash (\lambda x.s)t : \psi} \rightarrow\mathcal{E} \quad \Rightarrow \quad \dfrac{\begin{array}{c}\Pi_2 \\ \vdots \\ \Delta \vdash t : \psi\end{array} \qquad \begin{array}{c}\Pi_1 \\ \vdots \\ \Gamma, x : \varphi \vdash s : \psi\end{array}}{\Gamma, \Delta \vdash s[t/x] : \psi} \ \mathsf{Cut}$$

This particular example is, of course, β-reduction.

The same idea can be restated in a Gentzen-style sequent system (with left and right rules), but we will have to restrict ourselves to intuitionistic systems (only one formula on the right side), due to the asymmetric character of functions—many inputs, but only one output. Also, the rôle of Cut becomes more important, because it is the only mechanism which allows formulas from the left and the right sides to interact. An

example involving conjunction is shown below:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\Pi_1\\ \vdots\\ \Gamma \vdash s : \varphi\end{matrix} \quad \begin{matrix}\Pi_2\\ \vdots\\ \Gamma \vdash t : \psi\end{matrix}}{\Gamma \vdash \langle s,t\rangle : \varphi \wedge \psi}\wedge\mathcal{R} \quad \dfrac{\begin{matrix}\Pi_3\\ \vdots\\ \Delta, x : \varphi \vdash u : \xi\end{matrix}}{\Delta, z : \varphi \wedge \psi \vdash u[\pi_1(z)/x] : \xi}\wedge\mathcal{L}}{\Gamma, \Delta \vdash u[\pi_1(z)/x][\langle s,t\rangle/z] : \xi}\,\mathsf{Cut}$$

$$\Rightarrow \qquad \dfrac{\begin{matrix}\Pi_1\\ \vdots\\ \Gamma \vdash s : \varphi\end{matrix} \quad \begin{matrix}\Pi_3\\ \vdots\\ \Delta, x : \varphi \vdash u : \xi\end{matrix}}{\Gamma, \Delta \vdash u[s/x] : \xi}\,\mathsf{Cut}$$

Although this is called Cut elimination, the cut does not disappear immediately, rather it is pushed up the proof tree and disappears only at its leaves (identity axioms $\varphi \vdash \varphi$). The process of cut elimination (proof normalization) corresponds exactly to computation in a term rewriting system. Thus we have a *computation as proof reduction* paradigm. For some generalizations of the Curry-Howard interpretation based on Gabbay's Labelled Deductive Systems (which themselves are generalization of term assignment), see [58].

Everything mentioned up to this point was about CL. When LL comes into play, it renders a refinement of the λ-calculus in which the programmer has finer control on evaluation order (lazy vs eager evaluation) and memory allocation (multithreaded data which demands garbage collection vs single threaded data which does not). Usually about 90% of the data created at runtime is referenced from only one place and is consumed right away (generation-based scavenging is based on this fact). Thus it is profitable to manage this data in a "short-term" memory and release it right after its use, avoiding the need for garbage collection. Long-term data can be stored explicitly using the exponential !. In LL the manipulations of stored data are given explicitly through the rules of Promotion (storing), Dereliction (reading), Contraction (duplication) and Weakening (discarding). All these uses can be either specified explicitly by the programmer, giving the compiler richer information for storage management, or inferred by a smart type-inference mechanism, thus using linear λ-calculus as a lower-level intermediate code.

Some early work in this direction has been done by Girard and Lafont [72] and Holmström [88]. Lafont introduced a Linear Abstract Machine [99] which implements a functional language without the use of garbage collection (but with much copying of terms). An important work is the one by Abramsky [1]. He describes a linear machine more faithful in spirit to the SECD machine of Landin. Wadler has also done work in this area, introducing the distinction between linear short-term memory and non-linear garbage collected memory [152, 153]. Other works are [31, 111]. Actually implemented linear functional languages are Linear ML (Chirimar, Gunter and Riecke [45, 46]), Lilac (Mackie [117]) and an implementation of Lisp [90].

Work in a rather different vein has been done on *optimal lambda reduction*. An optimal evaluator should utilize all the term sharing present in the lambda expression to be reduced, avoiding structure copying and thus avoiding work duplication. The performance criteria proposed fifteen years ago by Lévy remained unachieved for a long time, until recently Lamping [105] and Kathail [93] independently developed optimal reduction algorithms. However their solution is rather involved, and later Gonthier, Abadi and Lévy [76, 77] greatly simplified it by "reengineering" it in terms of LL proof nets. What is more important, this gave a better logical foundation to the algorithm, which proved very useful for generalizing it to richer typed λ-calculus with inductive types, performed by Asperti and Laneve [19, 22, 23, 24, 106]. Their *interaction systems* can be considered a generalization of Lafont's *interaction nets* [100] on the other hand, one which removes the requirement for linearity and allows sharing of terms. For further joint developments with Danos and Regnier, see [50, 25, 20].

3 LL and Logic Programming

The paradigm *computation as proof search* plays for generalized forms of logic programming a rôle, similar to the one which the paradigm "computation as proof reduction" plays for functional programming (described in the previous section). In order to obtain a feasible system, it is important to restrict attention to only a certain type of proofs which are easier to construct than general proofs and then to identify a sensible fragment of the logic for which this restricted type of proofs is complete. The Cut rule is infeasible from a computational viewpoint (under a bottom-up reading), because its premises contain a formula which does not appear in its conclusion. Presented with the conclusion, the search procedure will have to make a "wild guess" at this point to pick a formula (Cut compromises the subformula property). Therefore, it is highly desirable for a logical system to have the cut elimination property, which justifies that Cut can be pushed all the way up to the leaves of the proof tree. There one finds either logical axioms $\varphi \vdash \varphi$ at which Cut disappears, or non-logical axioms which correspond to program clauses and which can be handled by well-known indexing techniques.

An abstract foundation for generalized LP has been laid down by Miller *et al.* [133]. They define an abstract LP language as a set of formulas which can appear as program clauses and a set of allowed goals (queries), such that it is possible to perform goal-directed proof search (from the goal up). For this fragment, a restricted kind of proofs called *uniform proofs* should be complete (*i.e.* if a goal has a general proof, it has to have a uniform proof as well). In an intuitionistic setting, uniform proofs are defined as ones in which every occurrence of a non-atomic formula on the right side is the immediate result of a right-introduction rule. In other words, the search procedure reduces every compound goal to its parts and tries Cut (looks in the program) only when the goal is atomic. The authors instantiate these general ideas by introducing LP with hereditary Harrop formulas, the main innovation of which is that implication is allowed in goals. To prove an implication $\varphi \multimap \psi$ from a context Γ, the context is enriched with the antecedent and then ψ is tried in this new context Γ, φ. If contexts are taken to correspond to programs, this opens the possibility to formalize modular LP. Another possibility is to view the formulas in a context as the data fields of an object and obtain OO LP. However in the CL setting contexts always grow during the course of a computation (by making additional assumptions), and can shrink only upon backtracking. That is, something can be retracted from a context (or changed) only when the process which has created it terminates and thus cannot make use of the change. Building on this work, Hodas and Miller investigate LP in a linear setting [85]. Since LL allows for more refined control over contexts (multiplicity of formulas matters) and can model not only production of data but also consumption, this opens numerous possibilities for applications such as object-oriented programming, databases, natural language parsing *etc.*

Cut elimination is not the only property which should be investigated for a logic system and exploited for its "computerization". Another important features are the *permutabilities* (ways of commuting occurrences of inference rules) which the logic may have. These permutabilities render many proofs equivalent and it is profitable to select only one "canonic" proof for every equivalence class and restrict the search to only these canonical proofs. Since ! and $\otimes$ do not commute over all right rules, Hodas and Miller consider a fragment which does not contain them (some of these restrictions are lifted in the work of Harland and Pym [81, 82] which instead of uniform proofs consider *LR* proofs). The functionality of ! is regained by considering contexts (left-hand sides) consisting of two parts: intuitionistic (unbounded resources or methods in an OO setting) and linear (bounded resources or OO state attributes). (A very similar notion appears in the work of Girard on the Logic of Unity [68] which combines CL and LL.) Hodas has implemented a language based on this work, called Lolli [84].

In an early work Andreoli and Pareschi [12] investigate the permutability properties of LL and argue that it is better suited for LP that CL, for which severe restrictions are needed (Horn clauses) to make it amenable to effective proof search (resolution). The richer set of connectives of LL convey better the programmer's intentions to the LP system. To a certain degree, the proof is encoded directly in the goal, so the authors speak of "syntax-directed proof search". This work is the foundation of the language Logical Objects [13, 15] which is described in the following section.

A more recent work of Andreoli [9] investigates more deeply the proof-theoretical properties of LL. First he makes a distinction between *asynchronous* connectives which require no choice from the proof search procedure and introduce no nondeterminism or only "don't care" nondeterminism; and *synchronous* connectives which cause the search procedure to make a committed choice and introduce "don't know" nondeterminism which may lead to dead-ends and backtracking. This terminology comes from a concurrency view on computation, where the latter class of connectives introduces a synchronization (thus sequentiality) point for two parallel search processes.

The connectives $\&$, $\wp$, Γ and $\forall$ are asynchronous. In a one-sided (right-only) sequent calculus with two-part (dyadic) contexts (unbounded part Θ and bounded part Γ) their corresponding rules are

$$\frac{\Theta : \Gamma, \varphi \quad \Theta : \Gamma, \psi}{\Theta : \Gamma, \varphi \,\&\, \psi}\,[\&] \qquad \frac{\Theta : \Gamma, \varphi, \psi}{\Theta : \Gamma, \varphi \,\wp\, \psi}\,[\wp]$$

$$\frac{\Theta, \varphi : \Gamma}{\Theta : \Gamma, \Gamma\varphi}\,[\Gamma] \qquad \frac{\Theta : \Gamma, \varphi[c/x]}{\Theta : \Gamma, \forall x.\varphi}\,[\forall]$$

The [&] rule says that both conjuncts have to be proved (with unchanged context), eventually in parallel; the [$\wp$] rule is a simple syntactic rewriting; the [Γ] rule moves the exponential formula to the unbounded part of the context as soon as it appears (*i.e.* stores it away, but it can be retrieved from there at any time); the [$\forall$] rule says that the goal should be proved parameterized with an arbitrary constant c.

The connectives $\otimes$, $\oplus$, ! and $\exists$ are synchronous. Their rules are

$$\frac{\Theta : \Gamma, \varphi \quad \Theta : \Delta, \psi}{\Theta : \Gamma, \Delta, \varphi \otimes \psi}\,[\otimes] \qquad \frac{\Theta : \Gamma, \varphi}{\Theta : \Gamma, \varphi \oplus \psi}\,[\oplus_1] \qquad \frac{\Theta : \Gamma, \psi}{\Theta : \Gamma, \varphi \oplus \psi}\,[\oplus_2]$$

$$\frac{\Theta : \Gamma \vdash \varphi}{\Theta : \Gamma \vdash !\varphi}\,[!] \qquad \frac{\Theta : \Gamma, \varphi[t/x]}{\Theta : \Gamma, \exists x.\varphi}\,[\exists]$$

The [$\otimes$] rule involves a choice of splitting the context into two disjoint parts (but notice that the unbounded part Θ remains unchanged. Similar concerns about the need of "lazy splitting" of contexts were expressed in [85]); the [$\oplus$] rule introduces a choice as to which of the two disjuncts shall be tried; the [!] rule makes the irreversible decision to derelict the exponential (we don't have the option of storing $!\varphi$ in Θ because only Γ is the correct exponential marker for right-handed sequents); the [$\exists$] rule involves finding a concrete term t which will do the job.

In pursuit of the "true canonic" proofs, with maximum of permutabilities removed and minimal introduction of unnecessary sequentialities, Andreoli considers *focusing proofs*. They have the following features:

- If the goal contains asynchronous formulas (*i.e.* formulas with asynchronous top-level connective), they are immediately decomposed. This can be done either in any order, or in parallel.
- If the goal has only synchronous formulas, one of them is selected for processing. (Although all of them have to be proved after all, this choice will have implications on the depth of a dead-end tree to be pursued, up to infinite looping.) After the choice, the proof *focuses* on this formula and strips all layers of synchronous connectives from it. This is called a *critical focusing section* (the terminology comes from concurrent computing) and its purpose is to reach a dead-end (if we are doomed so) as soon as possible.

To formalize this, Andreoli introduces three-part (triadic) sequents where the first two parts are again the unbounded and bounded contexts, and the third one is the active part on which the proof procedure operates. However, this declarative formalization of a control strategy makes the presentation of the deductive system somewhat too "procedural". Also, since the [&] rule effectively copies the rest of the context in two branches of the proof, its execution as early as possible will duplicate the work to be done for the synchronous connectives, which is actually the really hard work. This may not matter if the two branches will be picked up by two processors working in parallel, but it matters in a sequential implementation. (It may happen

that the two synchronous proofs have to be different in the two branches, *e.g.* because the two conjuncts of & cause different unifications. In such case doing the & first is OK.)

An important achievement of this work is that focusing proofs are complete for *full* LL. Andreoli introduces a translation from full LL to a syntactically restricted language called LinLog for which focusing proofs are "natural", analogous to the normalization of CL formulas to clausal form.

A final remark is in place here: since removal of sequentialities and exploiting permutabilities are the very goal of the Geometry of Interaction, it can be expected that in the future Proof Nets will have important applications to LP. Some steps towards automatic construction of proof nets have been taken in [60], but the status of the work is still more one of a theorem prover than of a LP system.

For another survey of this area see [149].

4 LL and Models of Concurrency

CL is not constructive enough to make a good computational formalism without imposing certain restrictions on it. One of the main problems is that the reduction relation for full CL (obtained through the Curry-Howard correspondence) is not confluent (Church-Rosser). That is to say, the result of reducing a proof (term) depends on the order in which reductions are performed, so nondeterminism is introduced. To avoid this, a commonly employed restriction is to consider an intuitionistic system (with only one formula in the right side of the sequent) and dismiss negation. This introduces an asymmetry between assumptions (inputs) and conclusions (outputs), which is characteristic of functions. Indeed, probably the most important example of a non-commutative operation is application, where it makes a big difference which term is the function and which one is the argument; which one is active and which one is acted upon. This is the natural setting for functional programming (Section 2), but interaction between parallel processes needs a more symmetric setting.

Such a setting is provided by full LL. It possesses rich symmetries without being non-constructive. A very important feature in this respect is that linear negation is involutive, yet constructive. Two dual atoms φ and $\varphi^{\perp}$ are informally regarded as producer and consumer, question and answer, assumption and goal, information and anti-information, male and female jacks on electrical cables which can be plugged together by a Cut rule. When they meet, an interaction takes place, or the two jacks are plugged together to form an information path. The study of this approach is the essence of the Geometry of Interaction [65, 66, 67, 71, 3] (see [102] for a gentle introduction to the subject). Proof nets [62, 30] and interaction nets [100, 101, 102] are an important achievement of this research. [67, 71] introduces a formalism in which correctly typed programs are guaranteed to be deadlock-free (which corresponds in importance to the termination condition for the sequential case).

Girard has done early work in the direction of applications of LL to concurrency [63]. Pratt in [141, 140] describes in algebraic terms a duality of events and states. He says that events bear time and change information, while states bear information and change ("while away") time. One often speaks of events as "time-stamped"; it is natural to think of states as "information-stamped". Events are arranged in a poset with joins representing concurrence; states are arranged in a poset with meets representing choice. This duality of events and states, time and information, utilizes the Birkhoff-Stone duality from lattice theory. A connection with full LL also emerges. Pratt compares the transition from Petri nets to LL through the introduction of negation (non-monotonic operations) to his earlier work [138] on extending Kleene's algebra of regular expressions to Action logic. In [139] this work is carried further to the development of a geometric theory of concurrency, in the spirit of Geometry of Interaction. A good philosophical explanation of LL along these lines can be found in [142].

A number of concurrency models have been expressed in LL. Some of them are described in the following subsections.

4.1 LL and Petri Nets

One of the first models of concurrency represented in LL were *Petri nets* [17, 21, 80, 61, 37, 38, 53, 54, 120, 122, 123]. Category theory is used widely in these works, for example in [107] to show that high-level nets (whose markers are data structures) are also LL models. [39] relates two typical uses of category theory for Petri Nets, refinement (mapping a net to another one) and simulation (mapping possible executions of a net).

As an example, the Petri net on Figure 1 can be described by the following LL formulas (names before

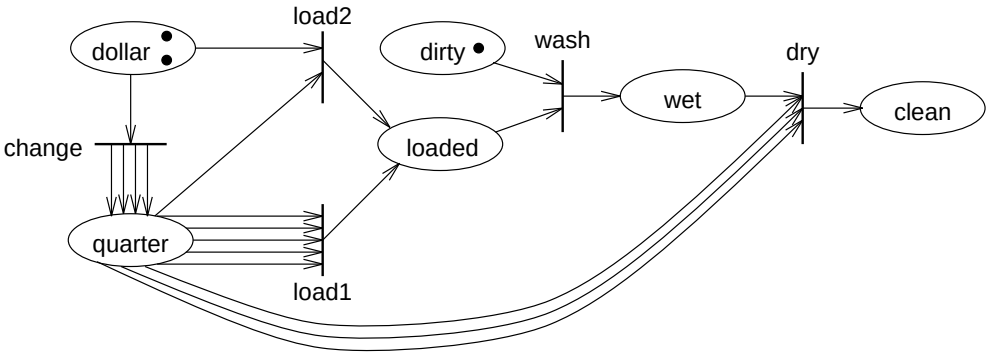

Figure 1: A Coin-Operated Washer-Dryer

the colons are simply labels and are not part of the formulas; $\mathsf{quarter}^3$ means $\mathsf{quarter} \otimes \mathsf{quarter} \otimes \mathsf{quarter}$):

$$\begin{array}{rrcl} \mathsf{change}: & !(\mathsf{dollar} & \multimap & \mathsf{quarter}^4) \\ \mathsf{load1}: & !(\mathsf{quarter}^5 & \multimap & \mathsf{loaded}) \\ \mathsf{load2}: & !(\mathsf{dollar} \otimes \mathsf{quarter} & \multimap & \mathsf{loaded}) \\ \mathsf{wash}: & !(\mathsf{loaded} \otimes \mathsf{dirty} & \multimap & \mathsf{wet}) \\ \mathsf{dry}: & !(\mathsf{quarter}^3 \otimes \mathsf{wet} & \multimap & \mathsf{clean}) \end{array}$$

The formulas need to be exponentiated because we would like to use them many times. They describe the structure of the Petri net, like methods (program code) describe the computations which an object can perform. Then we can add an initial marking, say $\mathsf{dollar} \otimes \mathsf{dollar} \otimes \mathsf{dirty}$, as shown on the figure (two tokens in the position dollar and a load of dirty clothes) and ask whether a certain final marking is reachable from it, say $\mathsf{clean} \otimes X$. Martí-Oliet and Meseguer have introduced a generalization of this model called "financial games" [121] by allowing negated atoms (loans). If a marker needed for a transition is not present in a certain position, we can introduce its negation in that position and still perform the transition. Later we are obliged to return the loan (usually the final marking should have no occurrence of negated atoms).

4.2 LL and Process Calculi

In the second half of his seminal paper [1], Abramsky gives a computational interpretation of *full LL* in terms of process interaction. It is very instructive to compare this to the interpretation of *intuitionistic LL* in terms of functional programming given in the first half of the paper. Abramsky describes his paper as "a promising new approach to the parallel implementation of functional programming . . . , typed concurrent programming in which correctness is guaranteed by the typing". He introduces an extension of sequents called *proof expressions* to record the set of Cuts which have been performed during the proof. In addition to formulas and terms, proof expressions feature multisets of *coequations* $s \bot t$ which record that a cut (interaction) involving $s : \varphi$ and $t : \varphi^\bot$ has taken place, or in other words establish a communication channel between

two processes. Certain conditions are imposed on the sets of coequations, like acyclicity (no closed loop of variables exist; this gives deadlock-freeness) and linearity (every variable appears exactly twice, as the two connectors of a communication channel). The mechanism which makes coequations interact is a set of rewriting rules including CHAM-like "stirring" (see Section 4.3).

LL deduction has been presented as process reduction in CSP by Monteiro [134]. Conversely, asynchronous versions of Milner's π-calculus have been presented as a LL theory in works of Miller [132] and Bellin and Scott [29], and a number of other issues on mobile processes are discussed by Okada [137]. See [118] for a survey.

The abovementioned approaches stay more or less in the "computation as proof reduction" paradigm (Section 2).[3] Other approaches stem from the "computation as proof search" paradigm (Section 3).[4] Saraswat has collaborated with Lincoln [146, 147] to linearize Saraswat's Concurrent Constraint Programming framework and extend it with state change (the old cc paradigm only allows monotonic accumulation of constraints). Their system [147] is higher-order and polymorphically typed using a version of Church's Simple Theory of Types. An implementation of part of this framework is available, the language Linear Janus.

Another work in the same vein[5] is the language ACL (Asynchronous Communication in LL) by Kobayashi and Yonezawa [97, 95]. Given a set of "message atoms" A_m and a set of "process atoms" A_P, the syntax of ACL is defined as follows:

C	$::=$	$A_P \circ\!\!- G$	Clause
G	$::=$	$\bot \mid \top \mid A_m \mid \Gamma A_m \mid A_P \mid G \parr G \mid R$	Goal
R	$::=$	$M \otimes G \mid R \oplus R$	message Reception
M	$::=$	$A_m^\bot \mid M \otimes M$	multiset of Messages to be received

Given an initial configuration (multiset) of process atoms and a program $!\mathbf{P}$ of modalized clauses, the system constructs a proof bottom-up by backward chaining. A process atom (name) is unfolded with the body of a matching clause. This body (goal) can proceed with one of the following actions:

Termination $\bot$ causes the process to terminate (disappear) since $\bot$ is the unit element for $\parr$, and $\parr$ is used to hold the context together.

Abort $\top$ terminates the whole configuration of processes, because $\overline{!\mathbf{P} \vdash \top, \Gamma}$ is an axiom of LL for any Γ. (In some approaches this can be interpreted as succesful termination of the program).

Message Send A_m corresponds to posting a message in the configuration. A corresponding receiver picks the message from the configuration and consumes it. ΓA_m is a special kind of "modal message" which does not disappear after it is read by a receiver, so it can be used for information sharing.

Call A_P calls another process, or in other words replaces the current process with A_P.

Parallel Composition $G \parr G$ starts two independent processes in parallel. The same connective is used for message send, *e.g.* $P \circ\!\!- m \parr Q$ means that P sends message m and then becomes Q. In fact the distinction between messages (A_m) and processes (A_P) is blurred in this LL setting, and this is used in [98] to pass processes as first-class messages.

Message Reception $m_1 \otimes \cdots \otimes m_n \otimes G$ waits until all the messages $m_1, \ldots, m_n$ become available, consumes them and then executes G. $R_1 \oplus R_2$ is "external choice": depending on the configuration, only the receptor R_i that can be satisfied is executed, and the other one is discarded. For example $(a^\bot \otimes A) \oplus (b^\bot \otimes B)$ in the presence of a continues as A, and in the presence of b continues as B.

[3] However "computation" here is understood as "process interaction" instead of the "term reduction" of FP.

[4] Or rather "computation as proof," because many of these approaches utilise don't care non-determinism (as opposed to don't know non-determinism) and so do not have complete proof search procedures. See [96] for a language using don't know non-determinism.

[5] However in a dual setting.

The LL inference corresponding to a process interaction (send-receive act) is

$$\dfrac{\dfrac{\dfrac{\overline{!\mathbf{P} \vdash m_i^{\perp}, m_i} \quad !\mathbf{P} \vdash A_i, A, \Gamma}{!\mathbf{P} \vdash m_i^{\perp} \otimes A_i, m_i, A, \Gamma}\otimes \text{ reception}}{!\mathbf{P} \vdash \oplus_j\ (m_j^{\perp} \otimes A_j), m_i, A, \Gamma}\oplus \text{ choice}}{!\mathbf{P} \vdash \oplus_j\ (m_j^{\perp} \otimes A_j), m_i \parr A, \Gamma}\parr \text{ send}$$

An important part of this work is a concrete phase-space semantics derived through a fixed-point construction. However in order for this construction to work, a limitation of only one defining clause per process is imposed.

4.3 LL and the Chemical Abstract Machine

There is a large body of work close in spirit to the general approach of the Chemical Abstract Machine of Berry and Boudol [32, 36]. This approach regards a system of distributed processes/agents/objects as a chemical solution where molecules wander around in a Brownian-like motion, energised by a "magic stirring" mechanism. When two matching molecules get in contact with their interlocking (dual) parts, a chemical reaction takes place which consumes (parts) of the molecules and replaces them with a resulting product. "Membranes" serve to separate the solution into hierarchical "subsolutions" and insulate parts which do not have to interact. A LL-style "realization" of the CHAM was proposed by Andreoli, Ciancarini and Pareschi [10], based on their earlier work on Linear Objects [13] (see also Section 5).

Communication is achieved by global broadcasting to all objects which is specified using a "tell marker". This is an extralogical feature of LO; a CHAM-style operational semantics for this language is given in [11]. No buffers or message queues are used; the message simply appears as a literal in every object. A message in which a particular object is not interested won't affect its behavior, it simply will never be picked by the object. This leads however to garbaging the objects with message leterals they will never access (called "context saturation" in [40]). This problem is solved in [16] by implementing a kind of "garbage collection" through abstract interpretation. A more thorough discussion of different communication mechanisms (blackboard vs broadcasting) can be found in [14]. Characteristic of their approach is that they use different LL connectives as the "separating membranes": the parts of an object are glued by multiplicative disjunction, and the whole "solution" is held together by additive conjunction.

See also [131, 124] for a more general development in Meseguer's Rewriting Logic. The paper [125] represents LL (among other logics) in Rewriting Logic.

5 LL and State-Oriented Programming

While one learns about LL, all of the time one has the feeling that LL is about state. Indeed, the developments in storage management for functional programming (Section 2) demonstrate this. O'Hearn [135] has undertaken further developments aimed at the famous "update in place" problem in FP, which efforts have lead more or less to a "logical reconstruction" of Reynolds' Syntactic Control of Interference. This is a type-inference scheme which is able to detect when two instructions do not interfere with each other and thus can be executed in parallel, as well as when a potentially large structure (*e.g.* an array) is referenced from only one place in the program, so can be updated *in situ*.

Uday Reddy [144, 143] builds upon this work to develop an explicit *LL Model of State*, aiming at incorporating state manipulation in FP and denotational description of (higher-order) imperative languages. (And I strongly believe that this work will be useful for the integration of state-oriented (OO) and Logic Programming). He points out that the notion of state originally present in LL is rudimentary, in that it is only usable for low-level implementation developments and does not supply a language for describing state manipulations. He says for his work "the model is significantly richer than that based on pure LL". Values in λ-calculus (formulas of CL) are static and eternal. LL takes the first necessary step, making values dynamic (consumable) and volatile. Then the exponentials regain back the lost staticity. But there is more to state:

it is dynamic, yet not volatile; stable, yet not eternal. Reddy calls this *regenerative state*: after every access to it the datum is consumed, but recreates itself automatically. He introduces a special modality † for it. As ! is the type constructor for static values, so is † the type constructor for regenerative values. Yet one more type constructor is introduced, *st* for read-write states. The characteristic recursive equations for the three modalities are compared below:

$$!\varphi = \mathbf{1} \,\&\, \varphi \,\&\, (!\varphi \otimes !\varphi) \qquad \dagger\varphi = \mathbf{1} \,\&\, (\varphi \otimes \dagger\varphi) \qquad st\ \varphi = \mathbf{1} \,\&\, (!\varphi \otimes st\ \varphi) \,\&\, (!\varphi \multimap st\ \varphi)$$

The component **1** allows discarding the datum, φ means to use it as a dynamic value and then discard it, $(!\varphi \otimes !\varphi)$ means to duplicate a static value, $(\varphi \otimes \dagger\varphi)$ means to read off a state obtaining a dynamic value and to regenerate the state, $(!\varphi \otimes st\ \varphi)$ reads off a static value (this is just a matter of convenience; a dynamic one could have been returned instead) and regenerates a *st* state, and finally $(!\varphi \multimap st\ \varphi)$ writes a state with a static value and returns the resulting state (here a linear value could not have been used because we don't have the right to capture a linear value).

Regenerative values (and for that matter *st* states) use for cloning *threading* instead of simple contraction. This is motivated by the fact that the two new owners of the clones should know which one is to use it first and which one second. Furthermore, this threading obviates the need for non-commutativity in the logic (indeed state develops in time, which is sequential and thus non-commutative). Reddy does not use non-commutative LL [5, 4, 155] but rather introduces a somewhat *ad-hoc* non-commutative multiplicative $\triangleright$ called *before*.[6] The paper contains a sequent proof system, term assignment and a coherent semantics for the model. The coherent semantics of $\triangleright$ shows very interesting properties, placing it somewhere between & and $\otimes$, thus $\triangleright$ possesses a mixture of conjunctive and disjunctive features. The webs (coherent spaces) of the new modalities are defined as $|\dagger\varphi| = |\varphi|^*$ (the set of all finite sequences) and $|st\ \varphi| = ss(|!\varphi| + |!\varphi|)$ (a certain subset of the disjoint sum of two copies of $|!\varphi|$, corresponding to reading and writing the state). Compare this to $|!\varphi| = \wp(|\varphi|)$, the set of finite coherent subsets of $|\varphi|$.

Reddy discusses the issue who will be willing to program in a system where you have to specify tediously every simple step. Not all weakenings, contractions and threadings need to be explicit in the term assignment system: the smarter the type inference algorithm, the less the burden on the programmer. The important point is that the formalism is present, so a theoretician can use it to design language and implementation features in a logical way. He says "a crucial point at which the language ceases to be functional and becomes imperative is when threading is made explicit". Overall, although the work is very interesting and important, I find the approach somewhat too complicated, especially concerning threading. Hopefully a simpler formalism can be found, at least for some restricted applications.

A famous work on Object-Oriented Logic Programming is the language Linear Objects (LO) developed by Andreoli and Pareschi at ECRC, Germany [13, 15]. They have written many papers on LO, their focus shifting initially from state representation (similar to an earlier proposal of John Conery [47]), later to concurrency issues (described in Section 4). In their approach, an object is a multiplicative disjunction of attribute literals, which as a whole is a free-floating additive conjunct in the "solution" containing all objects. I will describe here only one feature of their model, *builtin inheritance*. Unlike traditional OOP, methods in LO are not attached to objects (classes) explicitly by the programmer, but a method is applicable every time its head matches (through ACI-unification) a part of the object state. Then this part is consumed and replaced by the result of the method. Adding more attributes to the object (specialization) does not render any methods "of the superclass" inapplicable (so method overriding is somewhat problematic). This is similar to delegation-based object languages (*e.g.* Self), but constitutes an even more radical departure from "classical"[7] OOP, because delegates are not specified explicitly.

For a survey on representing state in OOLP see [6].

[6] The same operator appears in a paper by Monteiro [134] (see Section 4.2).

[7] That is, class-based OOP (pun unintended)

6 Other Applications

This section describes some less "populated" (though not necessarily less important) areas of application of LL which did not fit in the sections above.

6.1 LL for Logical Operational Semantics of PROLOG

A couple of papers by Serenella Cerrito give a logical account for some operational features of PROLOG. The first paper [41] is in a traditional vein and gives a LL semantics for allowed programs similar to Clark's completion, the only difference being that LL is used instead of CL. A program is *allowed* if every variable occurring in the head of a clause also occurs in a positive literal in the body, thus SLDNF (resolution with Negation-As-Failure (NAF)) will always be able to pick for evaluation a grounded negation, so no "floundering" goals will happen (NAF is "safe"). A completeness result for SLDNF with respect to this semantics is proved. This generalizes slightly the earlier semantical result which demanded the program to be not only allowed, but also hierarchical (stratified).

The second paper [42, 43] develops a suggestion of Girard that LL can be used to account for the difference between the "internal" logic of PROLOG (that is, the specific left-to-right selection strategy and depth-first search which PROLOG employs) and the "external" logical semantics which are ascribed to PROLOG. It gives a LL axiomatization of NAF by encoding precisely not only the clauses themselves, but also the order of literals in the bodies and the order of clauses in the program. For example, the program $\langle p \leftarrow a, b;\ p \leftarrow c\rangle$ (angle brackets indicate sequence) is encoded in LL as

$$a \otimes b \multimap p;\ (a^{\perp} \oplus (a \otimes b^{\perp})) \otimes c \multimap p.$$

The second clause accounts for all possible ways of the first clause to fail. Actually Cerrito gives a sequent encoding; also the failure conditions of the whole procedure of a given predicate letter are to be specified. It is curious to note that this axiomatization corresponds more closely to what PROLOG actually does than even SLDNF, an operational semantics. Although somewhat clumsy (*e.g.* the use of non-commutative LL would probably do the translation more natural), in my opinion this axiomatization compares favorably to many of the known semantics for NAF.

A paper by JiříZlatuška [156] gives a LL semantics to committed-choice guarded flat PROLOGs such as CONCURRENT PROLOG or PARLOG. He uses one-sided sequents and translates a guarded clause

$$(H \leftarrow G_1, \ldots, G_m | B_1, \ldots, B_n)$$

to the LL proper axiom

$$\vdash H \multimap ((G_1 \otimes \cdots \otimes G_m) \multimap (B_1 \parr \cdots \parr B_n)).$$

The guards are defined by proper axioms $\vdash G_i$ or $b \multimap (d \multimap \perp$ (b=builtin test, d=defining condition of the test). An alternative to translating the program to a set of proper axioms is to translate it to a *formula* $P = !(C_1 \,\&\, \cdots \,\&\, C_m)$, where $C_1, \ldots, C_m$ are the translations of the individual clauses. Then the computation from a set of goals (state) $p_1, \ldots, p_k$ to the set of goals $q_1, \ldots, q_l$ is represented by the provability of the sequent

$$\vdash P\perp, (p_1 \parr \cdots \parr p_k)\perp, q_1, \ldots, q_k.$$

The semantics is only restricted to deadlock-free computations.

6.2 LL and Non-Monotonic Issues in AI

One of the most difficult problems in Logics in AI is non-monotonic reasoning (NMR). NMR and the related issues of knowledge base *revision* are inherently very difficult, but I think that quite often another problem which is not that difficult is put in the same bag as NMR and is "forced" to behave badly. This is the problem of state change, knowledge base *update* and the related Frame problem. The difference between

revision and update is that the former involves modifying a knowledge base upon acquiring new knowledge without any change in the real world (thus some of the possible worlds present in our model can be found infeasible in light of the new knowledge); whereas the latter involves actualization of the knowledge to reflect a change in the real world and cannot discard any of the possible worlds [94]. I believe that LL is (will be) very useful for update, but probably not that much for revision.

6.2.1 :

Linear Deductive Planning An area where update and state change are prominent is *Deductive Planning*. Quite early Wolfgang Bibel has noticed that a modification of his connection method for automated proof can make Frame axioms unnecessary and makes a good planning formalism. The mentioned modification consists of a linearization of the graph, allowing every literal to have only one outgoing edge (to be used only once). He has been criticised on semantic grounds, namely that since a semantics for the linearization is not known, it constitutes no more than a clever trick of unclear virtue. Later Bibel *et al.* [33] gave a semantics for this method. Further development of this method was done by Fronhöfer [56, 57].

A series of works with an emphasis on an equational implementation of the idea is one by Josef Schneeberger, Steffen Hölldobler and his students at Techniche Hochschule Darmstadt [86, 87, 78, 79]. They represent the components of state as a multiset term held together by an Associative-Commutative-with Identity operation (ACI-operation) $\circ$, which can be thought of as multiset union. It is important that $\circ$ is not idempotent. The state of the planner is represented by the predicate $plan(I, P, G)$ where I is the multiset representing the initial (current) state, G is the multiset representing the goal state, and P is a list of actions which transform I to G. An action a with conditions $c_1, \ldots, c_n$ and effects $e_1, \ldots, e_m$ is represented as the clause

$$plan(I \circ e_1 \circ \cdots \circ e_m, [a|P], G) :- plan(I \circ c_1 \circ \cdots \circ c_n, P, G).$$

An additional clause $plan(G \circ X, [], G)$ gives termination (if the initial state is a subset of the final state then the empty plan suffices).

What is important here is that PROLOG terms have to be unified *modulo* the ACI operation *(ACI-unification)*. Hölldobler calls resolution with builtin ACI-unification *SLDE resolution*. These ideas are quite close to work of Meseguer [129, 128, 130] to formalize concurrent programming by non-equational (one-way) ACI-rewriting. The connection has been recently developed by Martí-Oliet and Meseguer [125, 124], see also recent work of Malcolm [119]. Techniques for ACI-rewriting are described in *e.g.* [110, 83, 116].

Hölldobler *et al.* [86, 78] prove that at least for deductive planning, three proposed approaches —the linear connection method, the ACI-rewriting approach and the LL-based approach (see below)— are equivalent. They also consider issues of change, action and specificity (selecting the method which matches largest part of the state) [79].

Masseron *et al.* [127] consider planning in a logic deduction setting. They first describe from philosophical grounds an abstract model of actions (basically relations with specific properties) and then "implement" this in LL. They consider a blocks world example and the three socks example. In the companion paper [126] Masseron lays down the beginnings of a geometric theory of (conjunctive only) actions, which turns out to be a simplification of Petri nets, with only one edge allowed to enter and exit a position (so positions are not drawn explicitly). Similar ideas in a logic programming setting are presented in [96] (see Section 4.2 for more information on ACL).

Linear deductive planning is a very illuminating example of the utility of LL for reasoning about change. With a relatively simple application of LL and parts of the Geometry of Interaction, researchers were able to largely overcome the Frame problem which was undefeated for such a long time maybe just because it was looked upon from the wrong angle.

6.2.2 :

LL and Hierarchies with Exceptions LL itself is monotonic, but it contains a kind of non-monotonicity which is more basic than the "artificially imposed" non-monotonicity present in, say, Default Logic. Namely, after performing a derivation the initial formula gets consumed and is no more available for other derivations. Due to the symmetry of LL, sometimes $\varphi^\perp$ can be regarded as a question whether φ holds, so adding it to a system can trigger such a derivation. A formally more clear non-monotonicity is the one of $\vdash$ (and $\perp\!\!\circ$) w.r.t. adding formulas to the antecedent: if $\Gamma \vdash \Delta$ then usually (in the absence of non-logical axioms) $\Gamma, \varphi \not\vdash \Delta$. This non-monotonicity can be used profitably for representing classification hierarchies with exceptions (inheritance with overriding).

The weary Tweety problem about penguins and birds is represented in a hierarchy with exceptions as follows:

$$p \rightarrow b,\ b \rightarrow f,\ p \rightsquigarrow f$$

(*p*enguins are *b*irds, *b*irds are *f*lying objects, yet *p*enguins are not *f*lying objects). Fouqueré and Vauzeilles [55] propose to code this by the following non-logical (proper) axioms in LL:

$$p \vdash p^+ \otimes b \otimes f^\perp \otimes !(f \perp\!\!\circ \mathbf{1}) \quad (1)$$

$$b \vdash b^+ \otimes f \quad (2)$$

$$f \vdash f^+ \quad (3)$$

A neutral atom f represents initial and intermediate data, f^+ represents a positive answer, and $f^\perp$ represents a negative answer. For every node f in the hierarchy, we have $f \vdash f^+$ to confirm that the node has been reached (positively), in addition for all the outgoing positive edges $f \rightarrow p_1, \ldots, f \rightarrow p_m$ we simply add to the sequent $p_1 \otimes \cdots \otimes p_m$. The representation of the negative edges $f \rightsquigarrow n_1, \ldots f \rightsquigarrow n_k$ is just a bit more complex: we add the negative conclusion $n_i^\perp$, but we also have to cancel the possibility for arriving at a positive conclusion. $(n_i \perp\!\!\circ \mathbf{1})$ serves for this purpose, because $n_i \otimes (n_i \perp\!\!\circ \mathbf{1}) \vdash \mathbf{1}$, and $\mathbf{1}$ is the identity for tensor (so $\varphi \otimes \mathbf{1} = \varphi$). We actually use the exponentiated version $!(n_i \perp\!\!\circ \mathbf{1})$ in order to be able to cancel any number of n_is which may be born along different paths from the source node. It can be seen that while the positive information is defeasible (f^+ is concluded only "at the node f", *i.e.* through the atom f, and it can be cancelled there by $!(f \perp\!\!\circ \mathbf{1})$), the exceptions are definitive. However, the authors mention an extension of the framework with double (counter-) exceptions. The authors also give a proofnet-like theory for "standard" proofs of simple sequents (their nets are bicolored node- and edge-colored directed graphs which closely resemble the topology of the original hierarchical network). They compare their framework to Default Logic and prove that for the particular domain, Default Logic is equivalent to the simple-sequent fragment of Intuitionistic LL.

Let's see how does this formalization work. Given p ("Tweety is a penguin"), we deduce the right-hand side of (1) and then by (2) easily obtain

$$p^+ \otimes b^+ \otimes f \otimes f^\perp \otimes !(f \perp\!\!\circ \mathbf{1}). \quad (4)$$

Now we apply dereliction (see Section 1.3) to remove the !, then linear Modus Ponens to "cancel" f with $f \perp\!\!\circ \mathbf{1}$ ($\mathbf{1}$ is "empty" in a multiplicative conjunctive setting) to reach the desired answer

$$p^+ \otimes b^+ \otimes f^\perp.$$

Unfortunately nothing can stop us from applying (3) to (4) and obtaining the contradictory

$$p^+ \otimes b^+ \otimes f^+ \otimes f^\perp \otimes !(f \perp\!\!\circ \mathbf{1}).$$

"Cancellation" no longer works, and clearly $f^+ \otimes f^\perp$ is useless. There is no logical connection between f, f^+ and $f^\perp$, so the authors use a syntactic criterion ("simple sequents") to weed out such useless conclusions. They mention that we can (incidentally) use $f^\perp$ for $f^\perp$, in which case LL will detect the contradiction for

us, but nevertheless *it would have already been committed*. The shortcoming is that at the point (4), nothing guides the proof search procedure in the right direction (which is to apply the "internal reduction" in (4) and not to apply the non-logical axiom (3)).

Although LL allows us to avoid a contradictory conclusion which in CL could have been obtained walking along two conflicting threads of reasoning (by consuming the shared resource which is at the head of the two threads), it gives us no clue as to which one is the "right" thread. This information has to be provided by extralogical means (if at all). Thus, although [55] manages to capture hierarchies with exceptions in a logical way (as opposed to Default Logic which is extra-logical) the Cut-elimination which LL enjoys is in this case only a marginal advantage over Default Logic.

Another shortcoming of this approach is a lack of modularity. A penguin is much more things than simply a bird: it is a black-and-white thing, a fat thing, a polar inhabitant, *etc*. So there will be many positive arrows going out of it. But we cannot represent these arrows in separate theory modules, they all need to be dumped in one formula. Thus the only mode of operation of such a system is to pose a query p and get back a whole bunch of properties. The reason why it is hard to separate the properties is that if we do this and then try to obtain concurrently the answers for two properties, we have to supply two copies of p to "feed" the two relevant threads. But there are no logical means to prevent the proof search procedure from running these two p's along the same path (using them not as we intended) and obtain a contradiction. And to begin with, we had to have extra-logical means to specify the two interesting paths anyway.

So in (relatively) static situations like inheritance networks, logic theory modules, *etc*., in my opinion LL does not give such a significant advantage over CL as for state change. Maybe module manipulation calculi like the one of Goguen and Meseguer based on equational logic (the family of languages OBJ, see *e.g.* [75]) are more appropriate here.

7 Final Remarks

Disclaimer: the distribution of the works to the different sections is somewhat arbitrary; often the same work had to be mentioned in more than one section. I tried to make the connections clear. After all everything is connected to everything else.[8]

Caveat: some of the exposition above may be inexact or even incorrect, and certainly in many places it is incomplete. I have read less than half of the referenced papers, and I understand well half of that half. (My personal interest is Object-Oriented Logic Programming). Comments and corrections are most welcome.

There are different opinions about LL and its applicability/importance to computing. They range from scepticism:

> I am much more sceptic than you about LL. It *might*, of course, turn out to be useful/important. There is, surely, a lot of activity concerning it (I contributed to it myself). But it can be much ado about (almost) nothing. So far there are a lot of promises, but they are still only promises. Time will tell if this is not just a matter of fashion. I hope it is not, but I suspect it is. [Answer on the Linear mailing list to a question by Jon Barwise]

through cautiousness:

> An interesting development in logic of some significance to theoretical computer science [81, 82].

to enthusiasm and excitement. Here is what Jon Barwise himself says after getting a marvelous philosophical explanation "why LL" from Vaughan Pratt [142].

> "May you live in exciting times" goes the old curse.
> I found Vaughan's message extremely interesting. That is to say, I found it helpful, exciting, and daunting, all at once.

[8] As Mark Twain puts it, "Old maidens are beneficial to livestock, because they usually have cats which chase mice which eat ground bees which pollinate alfalfa which is fed to cattle."

- Helpful because it really gives me a way to think about LL that I can understand.
- Exciting because it does indeed point to genuine connections between his view of LL and work in Situation Theory on information flow.
- Daunting because it shows I really do have to understand the work in LL in order to continue what I have been up to.

Notwithstanding the different opinions, LL and its applications have certainly made amazing progress for the 6 years since its inception. Definitely Barwise's words above do not hold exclusively for Barwise's own work, because LL has exhibited "genuine connections" to many fields of Computing Science. There even appears to exist something of a "linearization fashion", for example Curien linearised his Concrete Data Structures [48].

Whether this is only a momentary enthusiasm, time will show. At least I hope it won't have negative consequences to the field (like the hype around AI a decade ago), because the relative technicality of the subject makes it unsuitable for enthusing wide audiences and the press.

Acknowledgements

Narciso Martí-Oliet and two anonymous referees made valuable suggestions which significantly improved the paper. I am especially indebted to Narciso for correcting a number of typos.

I am grateful to Andrea Asperti for pointing me out the work on optimal lambda reduction.

Large part of the bibliography came from publicly available bibliographies collected by Andre Scedrov and Anne Troelstra.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Comput. Sci.*, 111:3–57, 1993. Earlier version appeared as Imperial College Technical Report DOC 90/20, Oct. 1990.

[2] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science (FST-TCS'92)*, pages 291–301, New Delhi, India, Dec. 1992. Full paper available as technical report DOC 92/24, Imperial College, London, Sep. 1992 and from `theory.doc.ic.ac.uk: theory/papers/Abramsky/gfc.dvi`.

[3] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Logic in Computer Science (LICS'92)*, pages 211–222, Santa Cruz, CA, June 1992. IEEE Computer Society Press. Full version to appear in *Information and Computation*.

[4] V. Abrusci. Noncommutative intuitionistic linear propositional logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 36:297–318, 1990.

[5] V. Abrusci. Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. *Journal of Symbolic Logic*, 56(4):1403–1451, Dec. 1991.

[6] V. Alexiev. Mutable object state for object-oriented logic programming: A survey. Technical Report TR93–15, University of Alberta, Aug. 1993. Available from `ftp.cs.ualberta.ca: pub/TechReports/TR93-15`, file `TR93-15.ps.Z` or `TR93-15.a4.ps.Z`.

[7] G. T. Allwein. A neigborhood model for linear logic's exponentials. Manuscript, Visual Inference Laboratory, Indiana University, Bloomington, IN, Nov. 1992.

[8] G. T. Allwein and J. M. Dunn. Kripke models for linear logic. *Journal of Symbolic Logic*, 58(2):514–545, 1993.

[9] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[10] J.-M. Andreoli, P. Ciancarini, and R. Pareschi. Interaction abstract machines. In G. Agha, K. Yonezawa, and P. Wegner, editors, *Research Directions in Concurrent Object Oriented Programming*. MIT Press, 1993.

[11] J.-M. Andreoli, L. Leth, R. Pareschi, and B. Thomsen. True concurrency semantics for a linear logic programming language with broadcast communication. In *Theory and Practice of Software Development (TAPSOFT'93)*, pages 182–198, 1993.

[12] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems: A linear logic approach. In P. Schroeder-Heister, editor, *Intl. Workshop on Extensions of Logic Programming*, number 475 in LNAI, pages 1–30, Tübingen, Germany, 1989.

[13] J.-M. Andreoli and R. Pareschi. LO and behold! Concurrent Structured Processes. In *ECOOP-OOPSLA'90*, Ottawa, Ontario, 1990. (*SIGPLAN Notices*, 25(10):44–56, Oct. 1990).

[14] J.-M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'91)*, pages 212–229, Nov. 1991. *ACM SIGPLAN Notices*, 26(11).

[15] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9(3-4):445–473, 1991. Shorter version appeared in D.H.D. Warren and P. Szeredi (eds), *Intl. Conf. on Logic Programming (ICLP'90)*, Jerusalem, Israel, June 1990, pages 495–510.

[16] J.-M. Andreoli, R. Pareschi, and T. Castagnetti. Abstract interpretation of linear logic programming. In *International Logic Programming Symposium (ILPS'93)*, pages (295–314 or 315–334)Γ MIT Press, 1993.

[17] A. Asperti. A logic for concurrency. Technical report, Dipartimento di Informatica, Universitá di Pisa, Nov. 1987.

[18] A. Asperti. A linguistic approach to deadlock. Rapport de Recherche LIENS-91-15, Ecole Normale Superieure de Paris, 1991. Submitted to *Mathemathical Structures in Computer Science*.

[19] A. Asperti. Linear logic, comonads, and optimal reductions. *Fundamenta Informaticae*, 1994. Special issue devoted to Categories in Computer Science. To appear.

[20] A. Asperti, V. Danos, C. Laneve, and L. Regnier. Paths in the lambda calculus. Three years of communications without understanding. Draft, submitted to LICS'94.

[21] A. Asperti, G.-L. Ferrari, and R. Gorrieri. Implicative formulae in the 'proofs as computations' analogy. In *Principles of Programming Languages (POPL'90)*, pages 59–71, San Francisco, CA, Jan. 1990. ACM.

[22] A. Asperti and C. Laneve. Interaction systems I: The theory of optimal reductions. Technical Report 1748, INRIA-Rocquencourt, Sept. 1992. Available from `cma.cma.fr: pub/papers/cosimo/IS1.ps.Z`.

[23] A. Asperti and C. Laneve. Interaction systems II: The practice of optimal reductions. Technical Report UBLCS-93-12, Universitá di Bologna, May 1993. Available from `ftp.cs.unibo.it: pub/TR/UBLCS/93-12.ps.Z`.

[24] A. Asperti and C. Laneve. Optimal reductions in interaction systems. In *Theory and Practice of Software Development (TAPSOFT'93)*, number 668 in LNCS, pages 485–500, 1993.

[25] A. Asperti and C. Laneve. Paths, computations and labels in the lambda-calculus. In *Rewriting Techniques and Applications (RTA '93)*, number 690 in LNCS, pages 152–167, 1993.

[26] A. Avron. Simple consequence relations. *Information and Computation*, 92(1):105–139, 1991.

[27] M. Barr. $\star$-Autonomous categories and linear logic. *Mathemathical Structures in Computer Science*, 1(2):159–178, July 1991.

[28] G. Bellin. Proof nets for multiplicative and additive linear logic. Draft, submitted to *Annals of Pure and Applied Logic.*Earlier version appeared as University of Edinburgh Report LFCS-91-161, May 1991. Available from `theory.doc.ic.ac.uk: theory/papers/Bellin`, Apr. 1993.

[29] G. Bellin and P. Scott. On the π-calculus and linear logic. Manuscript to be submitted to Proc. MFPS 8, Oxford, Nov. 1992.

[30] G. Bellin and J. van de Wiele. Proof nets and typed lambda calculus. I. Empires and kingdoms. Draft, available from `theory.doc.ic.ac.uk: theory/papers/Bellin/king.dvi`, May 1993.

[31] N. Benton, G. Bierman, V. de Paiva, and J. Hyland. Term assignment for intuitionistic linear logic. Manuscript, Sept. 1992.

[32] G. Berry and G. Boudol. The Chemical Abstract Machine. In *Principles of Programming Languages (POPL '90)*, pages 81–94, San Francisco, CA, Jan. 1990. ACM.

[33] W. Bibel, L. F. del Cerro, B. Fronhöfer, and A. Herzig. Plan generation by linear proofs: On semantics. In *German Workshop on Artificial Intelligence (GWAI'89)*, number 216 in Informatik-Fachberichte, Eringerfeld, Geseke, Germany, Sept. 1989. Springer-Verlag.

[34] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992. Special Volume dedicated to the memory of John Myhill.

[35] R. Blute. Proof nets and coherence theorems. In D. Pitt et al., editors, *Category Theory and Computer Science*, number 530 in LNCS, pages 121–137, Paris, Sept. 1991.

[36] G. Boudol. Some Chemical Abstract Machines. Technical Report BP 93, INRIA Sophia Antipolis, 1993.

[37] C. Brown. *Linear Logic and Petri Nets: Categories, Algebra and Proof.* PhD thesis, University of Edinburgh, 1990. Technical Report ECS-LFCS-91-128.

[38] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Logic in Computer Science (LICS'90)*, pages 208–219, Philadelphia, PA, June 1990. IEEE Computer Society Press.

[39] C. Brown and D. Gurr. Refinement and simulation of nets – a categorial characterisation. In K. Jensen, editor, *Applications and Theory of Petri Nets*, number 616 in LNCS, pages 76–92, Sheffield, UK, June 1992.

[40] S. Castellani and P. Ciancarini. Comparative semantics of LO. Technical Report UBLCS-94-7, University of Bologna, Apr. 1994. Availabile by anonymous FTP from `ftp.cs.unibo.it: /pub/TR/UBLCS/SemanticsOfLO.ps.gz`.

[41] S. Cerrito. A linear semantics for allowed logic programs. In *Logic in Computer Science (LICS'90)*, pages 219–227, Philadelphia, PA, June 1990. IEEE Computer Society Press.

[42] S. Cerrito. A linear axiomatization of negation as failure. *Journal of Logic Programming*, 12(1-2):1–24, Jan. 1992.

[43] S. Cerrito. Negation and linear completion. In L. del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*. Clarendon Press, June 1992.

[44] H. Chau. A proof search system for a modal substructural logic based on Labelled Deductive Systems. In A. Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR'92)*, number 624 in LNAI (subseries of LNCS), St. Petersburg, Russia, July 1992. Extended version available as technical report DOC 93/1, Imperial College, London, Apr. 1993 (17 p.).

[45] J. Chirimar, C. Gunter, and J. Riecke. Proving memory management invariants for a language based on linear logic. In W. Clinger, editor, *Lisp and Functional Programming (LFP'92)*, pages 139–150, San Francisco, CA, June 1992. In *ACM LISP Pointers* 5(1).

[46] J. Chirimar, C. Gunter, and J. Riecke. Reference counting as a computational interpretation of linear logic, 1992. Manuscript, available from `research.att.com: dist/riecke/linear-logic-journal.ps`.

[47] J. S. Conery. Logical objects. In R. A. Kowalski and K. A. Bowen, editors, *Fifth International Conference and Symposium on Logic Programming*, pages 420–434, 1988.

[48] P.-L. Curien. Concrete data structures, sequential algorithms, and linear logic. Message on the 'Linear' mailing list. Available from `theory.stanford.edu: pub/linear`, 4 June 1992. File `10.summary`.

[49] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[50] V. Danos and L. Regnier. Local and asynchronous beta-reduction. In *Logic in Computer Science (LICS'93)*, pages 296–306, Montreal, 1993. IEEE Computer Society Press.

[51] V. de Paiva. The Dialectica categories. In J. Gray and A. Scedrov, editors, *AMS-IMS-SIAM Joint Summer Research Conference* Categories in Computer Science and Logic, number 92 in Contemporary Mathematics, pages 47–62, Boulder, CO, June 1987. American Mathematical Society, 1989.

[52] V. de Paiva. A Dialectica-like model of linear logic. In D. Pitt et al., editors, *Category Theory and Computer Science*, number 389 in LNCS, pages 313–340, Manchester, Sept. 1989.

[53] U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *Colloquium on Trees in Algebra and Programming (CAAP'90)*, number 431 in LNCS, pages 147–161, Copenhagen, Denmark, May 1990.

[54] U. Engberg and G. Winskel. Completeness results for linear logic on Petri nets. In A. Borzyszkowski and S. Sokołowski, editors, *Mathematical Foundations of Computer Science (MFCS'93)*, number 711 in LNCS, pages 442–452, Gdańsk, Poland, Aug. 1993. Full version is DAIMI PB, Jan. 1993.

[55] C. Fouqueré and J. Vauzeilles. Linear logic and exceptions. Manuscript, Université Paris-Nord. Email address: {`cf,jv`}`lipn.univ-paris13.fr`, Dec. 1993.

[56] B. Fronhöfer. Linearity and plan generation. *New Generation Computing*, 5:213–225, 1987.

[57] B. Fronhöfer. Linear proofs and linear logic. In D. Pearce and G. Wagner, editors, *Logics in AI: European Workshop JELIA'92*, number 633 in LNAI (subseries of LNCS), pages 106–125, Berlin, Germany, Sept. 1992.

[58] D. Gabbay and R. de Queiroz. Extending the Curry-Howard interpretation to linear, relevance and other resource logics. *Journal of Symbolic Logic*, 57(4):1319–1365, Dec. 1992.

[59] J. Gallier. Constructive logics. Part II: Linear logic and proof nets. Research Report PRL-RR-9, Digital Equipment Corporation, Paris Research Lab, May 1991.

[60] D. Galmiche and G. Perrier. A procedure for automatic proof nets construction. In A. Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR'92)*, number 624 in LNAI (subseries of LNCS), pages 42–53, St. Petersburg, Russia, July 1992.

[61] V. Gehlot and C. Gunter. Normal process representatives. In *Logic in Computer Science (LICS'90)*, pages 200–207, Philadelphia, PA, June 1990. IEEE Computer Society Press.

[62] J.-Y. Girard. Linear logic. *Theoretical Comput. Sci.*, 50:1–102, 1987.

[63] J.-Y. Girard. Linear logic and parallelism. In M. Venturini Zilli, editor, *Mathematical Models for the Semantics of Parallelism*, number 280 in LNCS, pages 166–182. Springer Verlag, 1987. Advanced School, Rome, September 1986.

[64] J.-Y. Girard. Quantifiers in linear logic. In *Temi e prospettivi della logica e della filosofia della szienca contemporanee (SILFS)*, volume 1, pages 11–33, Cesena, Italy, Jan. 1987. CLUEB, Bologna, Italy.

[65] J.-Y. Girard. Towards a geometry of interaction. In J. Gray and A. Scedrov, editors, *AMS-IMS-SIAM Joint Summer Research Conference* Categories in Computer Science and Logic, number 92 in Contemporary Mathematics, pages 69–108, Boulder, CO, June 1987. American Mathematical Society, 1989.

[66] J.-Y. Girard. Geometry of interaction I: Interpretation of system **f**. In R. Ferro et al., editors, *Logic Colloquium '88*, pages 221–260, Padova, Italy, Aug. 1989. North-Holland.

[67] J.-Y. Girard. Geometry of interaction II: Deadlock-free algorithms. In P. Martin-Löf and G. Mints, editors, *Intl. Conf. on Computer Logic (COLOG'88)*, number 417 in LNCS, pages 76–93, 1990.

[68] J.-Y. Girard. On the unity of logic. Prépublication No. 27, Equipe de Logique Mathématique, Université Paris 7, 1991. To appear in *Annals of Pure and Applied Logic* (Proceedings ALC'90).

[69] J.-Y. Girard. Quantifiers in linear logic II. In G. Corsi and G. Sambin, editors, *Nuovi problemi della logica e della filosofia della scienza (SILFS)*, volume II, Viareggio, Italy, Jan. 1990, 1991. CLUEB, Bologna, Italy. Also available as Equipe de Logique Mathématique Prépublication No. 19, Université Paris 7.

[70] J.-Y. Girard. Logic and exceptions: A few remarks. *Journal of Logic and Computation*, 2:111–118, 1992.

[71] J.-Y. Girard. Geometry of interaction III: The general case. In *Linear Logic Workshop*, Cornell University, June 1993. MIT Press, to appear.

[72] J.-Y. Girard and Y. Lafont. Linear logic and lazy computation. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Theory and Practice of Software Development (TAPSOFT'87), Vol. 2*, number 250 in LNCS, pages 52–66, Mar. 1987.

[73] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1988.

[74] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Comput. Sci.*, 97:1–66, 1992. Also available as technical report MS-CIS-91-59, University of Pennsylvania. An extended abstract appeared in S.R. Buss and P.J. Scott (eds.), *Mathematical Sciences Institute Workshop on Feasible Mathematics*, Ithaca, NY, June 1989, Birkhauser 1990.

[75] J. Goguen, D. Coleman, and R. Gallimore, editors. *Applications of Algebraic Specification Using OBJ*. Cambridge University Press, 1993.

[76] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Principles of Programming Languages (POPL'92)*, pages 15–26, Albuquerque, NM, Jan. 1992. ACM.

[77] G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *Logic in Computer Science (LICS'92)*, pages 223–234, Santa Cruz, CA, June 1992. IEEE Computer Society Press.

[78] G. Große, S. Hölldobler, and J. Schneeberger. Linear deductive planning. Technical Report AIDA-92-08, Intellectik, Informatik, Techniche Hochschule Darmstadt, 1992.

[79] G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Equational logic programming, actions, and change. In *Joint Intl. Conf. and Symp. on Logic Programming (JICSLP'92)*, 1992. Also in *Logic and Change*, a workshop at GWAI'92.

[80] C. Gunter and V. Gehlot. Nets as tensor theories. In G. D. Michelis, editor, *10-th Intl. Conf. on Application and Theory of Petri Nets*, pages 174–191, Bonn, Germany, 1989. Updated paper appeared as Technical Report MS-CIS-89-68 Logic & Computation 17, Department of Computer and Information Science, University of Pennsylvania, Oct. 1989.

[81] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming (extended abstract). In V. Saraswat and K. Ueda, editors, *Intl. Symposium on Logic Programming (SLP'91)*, pages 304–318, 1991. The full paper is available as University of Edinburgh Technical Report ECS-LFCS-90-124, Nov. 1990.

[82] J. Harland and D. Pym. On resolution in fragments of classical linear logic (extended abstract). In A. Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR'92)*, number 624 in LNAI (subseries of LNCS), pages 30–41, St. Petersburg, Russia, July 1992.

[83] M. Henz. Term rewriting in associative commutative theories with identities. Master's thesis, State University of New York at Stony Brook, Dec. 1991. Available by anonymous FTP from `duck.dfki.uni-sb.de: pub/papers/MT-Henz.ps.Z`.

[84] J. Hodas. Lolli: An extension of λ-Prolog with linear logic context management. In *1992 λProlog Workshop*, 1992. Available from `ftp.cis.upenn.edu: pub/Lolli`.

[85] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 110(2):327–365, May 1994. Available by WWW from `http://www.cis.upenn.edu/ ~dale` or by FTP from `ftp.cis.upenn.edu: pub/papers/miller/ic94.dvi.Z`. An extended abstract appeared in LICS'91:32–42, July 1991.

[86] S. Hölldobler. On deductive planning and the frame problem. In A. Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR'92)*, number 624 in LNAI (subseries of LNCS), pages 13–29, St. Petersburg, Russia, July 1992.

[87] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[88] S. Holmström. Linear functional programming. In T. Johnsson, S. P. Jones, and K. Karlsson, editors, *Workshop on Implementation of Lazy Functional Languages*, pages 13–32, 1988.

[89] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[90] H. B. Jr. Lively linear lisp—'look ma, no garbage!'. *SIGPLAN Notices*, 27(8):89–98, Aug. 1992.

[91] M. Kanovich. The multiplicative fragment of linear logic is NP-complete. ITLI Prepublication Series X-91-13, University of Amsterdam, 1991.

[92] M. Kanovich. Horn programming in linear logic is NP-complete. In *Logic in Computer Science (LICS'92)*, pages 200–210, Santa Cruz, CA, June 1992. IEEE Computer Society Press. Also University of Amsterdam ITLI Prepublication Series X-91-14.

[93] V. Kathail. *Optimal Interpreters for Lambda-Calculus Based Functional Languages*. PhD thesis, MIT, 1990.

[94] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In J. Allen, R. Fikes, and E. Sandewall, editors, *Knowledge Representation and Reasoning (KR'91)*, pages 387–394, Boston, MA, Apr. 1991.

[95] N. Kobayashi and A. Yonezawa. ACL — a concurrent linear logic programming paradigm. In *International Logic Programming Symposium (ILPS'93)*, pages 279–294. MIT Press, 1993.

[96] N. Kobayashi and A. Yonezawa. Reasoning on actions and change in a linear logic programming. Technical report, University of Tokyo, 1993. Draft, available by FTP from `camille.is.s.u-tokyo.ac.jp /pub/papers/`.

[97] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, (3), 1994. Short version appeared in *Joint Intl. Conf. and Symp. on Logic Programming (JICSLP'92)*, Washington, DC, Nov. 1992, Workshop on Linear Logic and Logic Programming.

[98] N. Kobayashi and A. Yonezawa. Typed higher-order concurrent linear logic programming. Technical Report 94-12, University of Tokyo, July 1994. Available by FTP from `camille.is.s.u-tokyo.ac.jp /pub/papers/ TR94-12-hacl-a4.ps.Z`.

[99] Y. Lafont. The linear abstract machine. *Theoretical Comput. Sci.*, 59(1,2):157–180, 1988. Some corrections in volume 62:327–328.

[100] Y. Lafont. Interaction nets. In *Principles of Programming Languages (POPL'90)*, pages 95–108, San Francisco, CA, Jan. 1990. ACM.

[101] Y. Lafont. The paradigm of interaction (short version), 1991.

[102] Y. Lafont. From proof nets to interaction nets. In *Linear Logic Workshop*, Cornell University, Mar. 1994. MIT Press, to appear.

[103] Y. Lafont and T. Streicher. Game semantics for linear logic. In *Logic in Computer Science (LICS'91)*, pages 43–50, Amsterdam, July 1991. IEEE Computer Society Press.

[104] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–169, 1958.

[105] J. Lamping. An algorithm for optimal lambda calculus reductions. In *Principles of Programming Languages (POPL'90)*, pages 16–30, San Francisco, CA, Jan. 1990. Also Xerox PARC technical report, 1989.

[106] C. Laneve. *Optimality and Concurrency in Interaction Systems*. PhD thesis, Dip. Informatica, Universitá di Pisa, Mar. 1993. Also technical report TD–8/93.

[107] J. Lilius. High-level nets and linear logic. In K. Jensen, editor, *Applications and Theory of Petri Nets*, number 616 in LNCS, pages 310–327, Sheffield, UK, June 1992.

[108] P. Lincoln. *Computational Aspects of Linear Logic*. PhD thesis, Stanford University, 1992.

[109] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[110] P. Lincoln and J. Christian. Adventures in associative-commutative unification. In *Ninth Conference on Automated Deduction (CADE'88)*, number 310 in LNCS, 1988.

[111] P. Lincoln and J. Mitchell. Operational aspects of linear lambda caclulus. In *Logic in Computer Science (LICS'92)*, pages 235–246, Santa Cruz, CA, June 1992. IEEE Computer Society Press.

[112] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *Foundations of Computer Science (FOCS'90)*, volume II, pages 662–671, St. Louis, MO, Oct. 1990. Also SRI International Technical Report SRI-CSL-90-08, Aug. 1990.

[113] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992. Special Volume dedicated to the memory of John Myhill.

[114] P. Lincoln and A. Scedrov. First order linear logic without modalities is NEXPTIME-hard. Manuscript, Sept. 1992. Available from `ftp.cis.upenn.edu: pub/papers/scedrov/mall1.dvi`.

[115] P. Lincoln and T. Winkler. Constant-only multiplicative linear logic is NP-complete. Manuscript, Sept. 1992. Available from `ftp.csl.sri.com: pub/lincoln/comult-npc.dvi`.

[116] D. Lugiez and J. Moysset. Complement problems and tree automata in AC-like theories. In P. Enjalbert, A. Finkel, and K. Wagner, editors, *Proceedings STACS 93*, volume 665 of *Lecture Notes in Computer Science*, pages 515–524. Springer Verlag, Feb. 1993. Available by anonymous FTP from `duck.dfki.uni-sb.de: pub/ccl/inria-lorraine/stacs93.ps.Z`.

[117] I. Mackie. LILAC—a functional programming language based on linear logic. Master's thesis, Department of Computing, Imperial College, London, 1991.

[118] I. Mackie. λ-calculus, linear logic and the π-calculus: A survey, Feb. 1993.

[119] G. Malcolm. Equational specification of systems of interacting objects. Talk at the BCS-FACS Christmas Meeting *Formal Aspects of Object-Oriented Systems*, Dec. 16-17 1993.

[120] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. In D. Pitt et al., editors, *Category Theory and Computer Science*, number 389 in LNCS, pages 313–340, Manchester, Sept. 1989.

[121] N. Martí-Oliet and J. Meseguer. An algebraic axiomatization of linear logic models. In G. Reed, A. Roscoe, and R. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 335–357. Clarendon Press, Oxford, 1991. Proceedings of the Oxford Topology Symposium, June 1989.

[122] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. *Mathemathical Structures in Computer Science*, 1:66–101, 1991. Revised version of paper in LNCS 389.

[123] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *Intl. Journal on Foundations of Computer Science*, 2(4):297–399, Dec. 1991.

[124] N. Martí-Oliet and J. Meseguer. Action and change in rewriting logic. Technical Report (to appear), Computer Science Laboratory, SRI International, 1993.

[125] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, Computer Science Laboratory, SRI International, Aug. 1993.

[126] M. Masseron. Generating plans in linear logic: II A geometry of conjunctive actions (note). *Theoretical Comput. Sci.*, 113, June 1993.

[127] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic: I Actions as proofs. *Theoretical Comput. Sci.*, 113, June 1993. Also in Proc. 10-th Conf. *Foundations of Software Technology and Theoretical Computer Science (FST-TCS'90)*, Banglore, India, LNCS 472, 1990.

[128] J. Meseguer. A logical theory of concurrent objects. In *ECOOP/OOPSLA'90*, Ottawa, Ontario, 1990. (*SIGPLAN Notices*, 25(10):101–115, Oct. 1990).

[129] J. Meseguer. Rewriting as a unified model of concurrency. In *CONCUR'90: Intl. Conf. on Concurrency Theory*, number 458 in LNCS, pages 384–400, Amsterdam, Aug. 1990. Also Technical Report SRI-CSL-90-02, SRI International, Feb. 1990.

[130] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992. Also Technical Report SRI-CSL-91-05, SRI International, Feb. 1991.

[131] J. Meseguer. A logical theory of concurrent objects and its realization in the MAUDE language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*. MIT Press, 1992(Γ).

[132] D. Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Extensions of Logic Programming (ELP'92*, 1992. Also University of Pennsylvania technical report MS-CIS-92-48, available from `ftp.cis.upenn.edu: pub/papers/miller/pic.dvi.Z`.

[133] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991. Special issue on LICS'87.

[134] E. Monteiro. Linear logic as CSP. *Journal of Logic and Computation*, 4(4):405–421, 1994.

[135] P. O'Hearn. Linear logic and interference control (preliminary report). In D. Pitt et al., editors, *Category Theory and Computer Science*, number 530 in LNCS, pages 74–93, Paris, Sept. 1991.

[136] M. Okada. Linear logic, Aug. 1992. Tutorial presented at the University of Tokyo (in Japanese). Available by FTP from `camille.is.s.u-tokyo.ac.jp /pub/okadasemi/ okadaall.ps.Z`.

[137] M. Okada. Mobile linear logic as a framework for asynchronous and synchronous mobile communication calculi (preliminary report). Technical report, Concordia University, 17 Feb. 1993.

[138] V. Pratt. Action logic and pure induction. In J. van Eijck, editor, *Logics in AI: European Workshop JELIA'90*, number 478 in LNCS, pages 97–120, Amsterdam, Sept. 1990.

[139] V. Pratt. Arithmetic+logic+geometry=concurrency. In *First Latin American Symposium on Theoretical Informatics (LATIN'92)*, number 583 in LNCS, pages 430–447, São Paulo, Brazil, Apr. 1992.

[140] V. Pratt. The duality of time and information. In W. Cleaveland, editor, *CONCUR'92: Third Intl. Conf. on Concurrency Theory*, number 630 in LNCS, pages 237–253, Stony Brook, NY, Aug. 1992.

[141] V. Pratt. Event spaces and their linear logic. In *Algebraic Methodology and Software Technology (AMAST'91)*, Workshops in Computing, pages 1–23, Iowa City, IA, 1991, 1992. Springer-Verlag.

[142] V. Pratt. Linear logic semantics (answer to Jon Barwise). Message on the 'Linear' mailing list. Available from `theory.stanford.edu: pub/linear`, 25 Feb. 1992. File `08.summary`.

[143] U. S. Reddy. Global state considered unnecessary: Semantics of interference-free imperative programming. In *ACM SIGPLAN Workshop on State in Programming Languages (SIPL'93)*, Copenhagen, Denmark, June 1993. Available as Yale University technical report YALEU/DCS/RR-968.

[144] U. S. Reddy. A linear logic model of state, Oct. 1993. Manuscript, 29 pages. Available from `cs.uiuc.edu: pub/reddy/papers/ state.full.ps.Z`. A shorter preliminary version is in `state.dvi`, July 1992, 17 pages.

[145] D. Roorda. *Resource Logics: Proof-theoretical Investigations*. Dissertation, University of Amsterdam, Sept. 1991.

[146] V. Saraswat. A brief introduction to linear concurrent constraint programming, Apr. 1993. Available from `parcftp.xerox.com: pub/ccp/lcc/lcc-intro.dvi.Z`.

[147] V. Saraswat and P. Lincoln. Higher-order, linear, concurrent constraint programming, July 1992. Available from `parcftp.xerox.com: pub/ccp/lcc/hlcc.dvi.Z`.

[148] A. Scedrov. A brief guide to linear logic. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 41:154–165, June 1990.

[149] A. Scedrov. Linear logic and computation: A survey. In H. Schwichtenberg, editor, *Proof and Computation*, Marktoberdorf Summer School, Germany, 1993. Springer-Verlag, 1994. To appear. Available from `ftp.cis.upenn.edu: pub/papers/scedrov/mdorf93.dvi`.

[150] R. Seely. Linear logic, $\star$-autonomous categories and cofree coalgebras. In J. Gray and A. Scedrov, editors, *AMS-IMS-SIAM Joint Summer Research Conference* Categories in Computer Science and Logic, number 92 in Contemporary Mathematics, pages 371–382, Boulder, CO, June 1987. American Mathematical Society, 1989.

[151] A. Troelstra. *Lectures on Linear Logic*. Number 29 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University, 1992.

[152] P. Wadler. Linear types can change the world! In *IFIP TC 2 Working Conference on Programming Concepts and Methods*, pages 546–566, Sea of Galilee, Israel, Apr. 1990. Elsevier Science Pub.

[153] P. Wadler. Is there a use for linear logicΓ In *Symp. on Partial Evaluation and Semantics-Based Program Manipulation*, pages 255–273, Sept. 1991. In *SIGPLAN Notices*.

[154] P. Wadler. A taste of linear logic. In A. Borzyszkowski and S. Sokołowski, editors, *Mathematical Foundations of Computer Science (MFCS'93)*, number 711 in LNCS, Gdańsk, Poland, Aug. 1993.

[155] D. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1):41–64, Mar. 1990.

[156] J. Zlatuška. Committed-choice concurrent logic programming in linear logic. In G. Gottlob et al., editors, *Computational Logic and Proof Theory. Third Kurt Gödel Colloquium, KGC'93*, number 713 in LNCS, pages 337–348, Brno, Czech Republic, Aug. 1993.