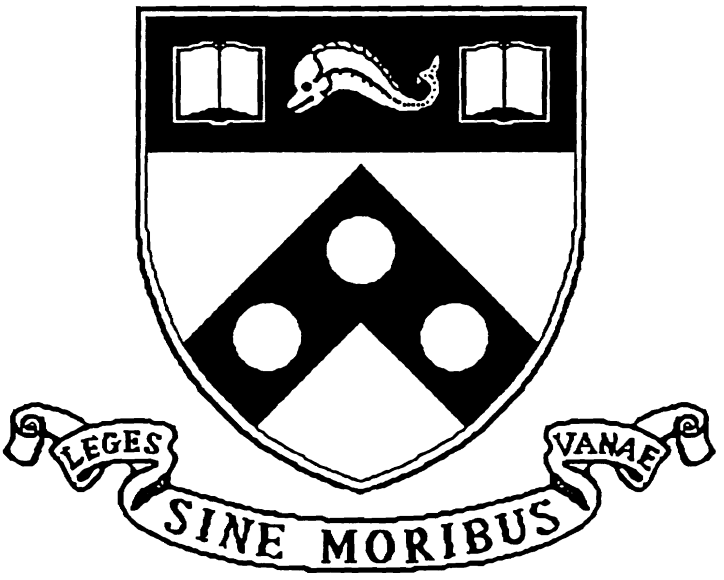
Proceedings of the Workshop on Linear Logic and Logic Programming
Washington, DC
14 November 1992

MS-CIS-92-80
LINC LAB 238

Edited by Dale Miller



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

November 1992

Proceedings of the Workshop on Linear Logic and Logic Programming

Washington, DC 14 November 1992
following the 1992 Joint International Conference and Symposium on Logic Programming

Declarative programming languages often fail to effectively address many aspects of control and resource management. Linear logic provides a framework for increasing the strength of declarative programming languages to embrace these aspects. Linear logic has been used to provide new analyses of Prolog's operational semantics, including left-to-right/depth-first search and negation-as-failure. It has also been used to design new logic programming languages for handling concurrency and for viewing program clauses as (possibly) limited resources. Such logic programming languages have proved useful in areas such as databases, object-oriented programming, theorem proving, and natural language parsing.

This workshop is intended to bring together researchers involved in all aspects of relating linear logic and logic programming.

Workshop organizers:

- Jean-Yves Girard, CNRS and University of Paris VII
- Dale Miller (chair), University of Pennsylvania, Philadelphia
- Remo Pareschi, ECRC, Munich

Table of Contents

Linear Logic and Logic programming: An Overview

Dale Miller

Computer Science Department
University of Pennsylvania
Philadelphia, PA 19104-6389

Workshop on Linear Logic and Logic Programming

14 November 1992
Washington, DC

Connections between Linear Logic and Logic Programming

Describing the External Logic of Prolog

S. Cerrito, "A Linear Semantics for Allowed Logic Programs", LICS90.

S. Cerrito, "A Linear Axiomatization of Negation as Failure", Journal of Logic Programming, January 1992.

J-Y. Girard, "Towards a Geometry of Interaction", in *Categories in Computer Science*, AMS series on Contemporary Mathematics, Vol 92, June 1987.

New Language Designs

Focus of this presentation.

A Proof System for Linear Logic

$$\frac{\Gamma' \longrightarrow \Delta', B \quad \Gamma'' \longrightarrow \Delta'', C}{\Gamma', \Gamma'' \longrightarrow \Delta', \Delta'', B \otimes C} \qquad \frac{B, C, \Delta \longrightarrow \Gamma}{B \otimes C, \Delta \longrightarrow \Gamma}$$

$$\frac{\Gamma \longrightarrow \Delta, B \quad \Gamma \longrightarrow \Delta, C}{\Gamma \longrightarrow \Delta, B \,\&\, C}$$

$$\frac{B, \Delta \longrightarrow \Gamma}{B \,\&\, C, \Delta \longrightarrow \Gamma} \qquad \frac{C, \Delta \longrightarrow \Gamma}{B \,\&\, C, \Delta \longrightarrow \Gamma}$$

$$\frac{B, \Delta' \longrightarrow \Gamma' \quad C, \Delta'' \longrightarrow \Gamma''}{B \parr C, \Delta', \Delta'' \longrightarrow \Gamma', \Gamma''} \qquad \frac{\Delta \longrightarrow B, C, \Gamma}{\Delta \longrightarrow B \parr C, \Gamma}$$

$$\frac{B, \Delta \longrightarrow \Gamma \quad C, \Delta \longrightarrow \Gamma}{B \oplus C, \Delta \longrightarrow \Gamma}$$

$$\frac{\Gamma \longrightarrow \Delta, C}{\Gamma \longrightarrow \Delta, B \oplus C} \qquad \frac{\Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, B \oplus C}$$

Here, Δ and Γ are multisets of propositional formulas, and comma denotes multiset union.

A Proof Systems for Linear Logic (continued)

$$\frac{\Gamma' \longrightarrow \Delta', B \quad C, \Gamma'' \longrightarrow \Delta''}{B \multimap C, \Gamma', \Gamma'' \longrightarrow \Delta', \Delta''}$$

$$\frac{B, \Gamma \longrightarrow \Theta, C}{\Gamma \longrightarrow \Theta, B \multimap C}$$

$$\frac{B, \Gamma \longrightarrow \Theta}{\Gamma \longrightarrow \Theta, B^{\perp}} \qquad \frac{\Gamma \longrightarrow \Theta, B}{B^{\perp}, \Gamma \longrightarrow \Theta}$$

$$\frac{}{B \longrightarrow B} \qquad \frac{}{0, \Delta \longrightarrow \Gamma} \qquad \frac{}{\Delta \longrightarrow \top, \Gamma}$$

$$\frac{B, \Delta \longrightarrow \Gamma}{!B, \Delta \longrightarrow \Gamma} \qquad \frac{\Delta \longrightarrow \Gamma}{!B, \Delta \longrightarrow \Gamma} \qquad \frac{!B, !B, \Delta \longrightarrow \Gamma}{!B, \Delta \longrightarrow \Gamma}$$

$$\frac{\Delta \longrightarrow B, \Gamma}{\Delta \longrightarrow ?B, \Gamma} \qquad \frac{\Delta \longrightarrow \Gamma}{\Delta \longrightarrow ?B, \Gamma} \qquad \frac{\Delta \longrightarrow ?B, ?B, \Gamma}{\Delta \longrightarrow ?B, \Gamma}$$

$$\frac{!\Delta \longrightarrow B, ?\Gamma}{!\Delta \longrightarrow !B, ?\Gamma} \qquad \frac{B, !\Delta \longrightarrow ?\Gamma}{?B, !\Delta \longrightarrow ?\Gamma}$$

Intuitionistic implication $B \supset C$ is coded as $!B \multimap C$.

Horn Clauses

How should the syntax

$$A_0 :\!- A_1, \ldots, A_n. \qquad (n \geq 0)$$

be translated into linear logic? Here, A_0, $\ldots, A_n$ are atomic formulas.
The following is an immediate translation starting from the intuitionistic setting.

$$!\forall \bar{x}[(A_1 \,\&\, \cdots \,\&\, A_n) \supset A_0]$$

This could be simplified by dropping the modality in the intuitionistic implication.

$$!\forall \bar{x}[(A_1 \,\&\, \cdots \,\&\, A_n) \multimap A_0]$$

However, the following translation seems to be the "tightest".

$$!\forall \bar{x}[(A_1 \otimes \cdots \otimes A_n) \multimap A_0]$$

In particular, this translation makes it easy to explore linear extensions where not all program components are unbounded (prefixed with !).

Proofs Involving Horn Clauses

$$\dfrac{\dfrac{\dfrac{}{A_0 \longrightarrow A_0}}{!\Delta, A_0 \longrightarrow A_0} \quad \dfrac{!\Delta \longrightarrow A_1 \cdots !\Delta \longrightarrow A_n}{!\Delta \longrightarrow A_1 \otimes \cdots \otimes A_n}}{!\Delta, (A_1 \otimes \cdots \otimes A_n) \multimap A_0 \longrightarrow A_0}}{!\Delta \longrightarrow A_0}$$

The search for a proof of $!\Delta \longrightarrow A_0$ reduces to the search for proofs of

$$!\Delta \longrightarrow A_1 \quad \ldots \quad !\Delta \longrightarrow A_n.$$

The program (the left-hand side) stays the same; the goal (the right-hand side) has been reduced.

Another Presentation of Horn Clauses

Translate the syntax

$$A_0 \text{:-} A_1, \ldots, A_n. \qquad (n \geq 0)$$

as the formula $!\forall \bar{x}[(A_1 \parr \cdots \parr A_n) \multimap A_0]$. In particular, if $n = 0$ then the translated formula is $!\forall \bar{x}[\bot \multimap A_0]$.

$$\dfrac{\dfrac{A_0 \longrightarrow A_0}{!\Delta, A_0 \longrightarrow A_0} \quad \dfrac{!\Delta \longrightarrow A_1, \ldots, A_n, \Gamma}{!\Delta \longrightarrow A_1 \parr \cdots \parr A_n, \Gamma}}{\dfrac{!\Delta, (A_1 \parr \cdots \parr A_n) \multimap A_0 \longrightarrow A_0, \Gamma}{!\Delta \longrightarrow A_0, \Gamma}}$$

Such a reduction never ends in an initial sequent: at best it ends in the sequent $!\Delta \longrightarrow \bot$. For the propositional calculus, the following holds: the list of goal $\langle G_1, \ldots, G_n \rangle$ reduces via SLD-resolution to the list of goals $\langle H_1, \ldots, H_m \rangle$ if and only if

$$!\Delta, H_1 \parr \cdots \parr H_m \vdash G_1 \parr \cdots \parr G_n.$$

In other words, goal reduction corresponds to implied-by.

Multiple-Conclusion Horn Clauses

Notice that goals in the list $\langle G_1, \ldots, G_n \rangle$ do not interact with each other.

If we admit clauses of the the form

$$\forall \bar{x}[A_1 \parr \cdots \parr A_n \multimap B_1 \parr \cdots \parr B_m]$$

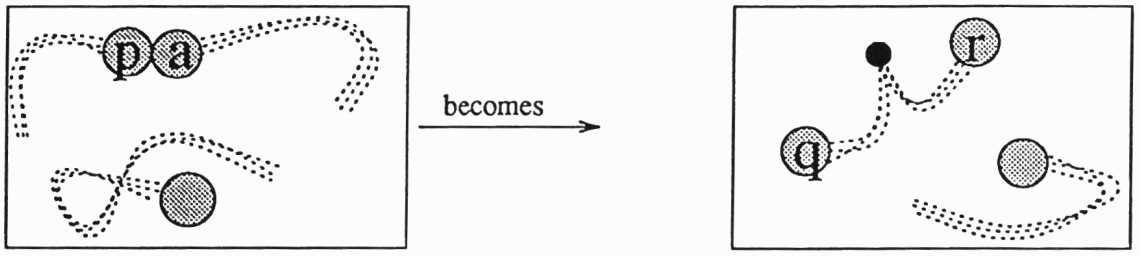
then we can describe some interactions.

When $A_1, \ldots, A_n, B_1, \ldots, B_m$ are atomic and $m > 0$, such clauses are *multi-conclusion* Horn clauses.

INTER-SPACE INTERACTION

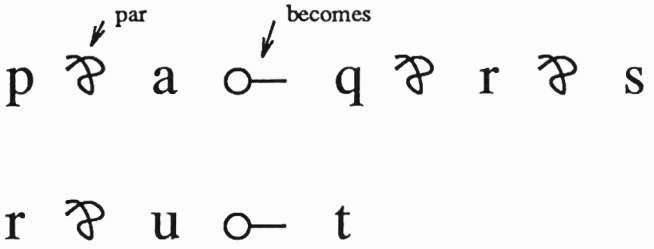
Each space can broadcast (typed) waves to all surrounding spaces.

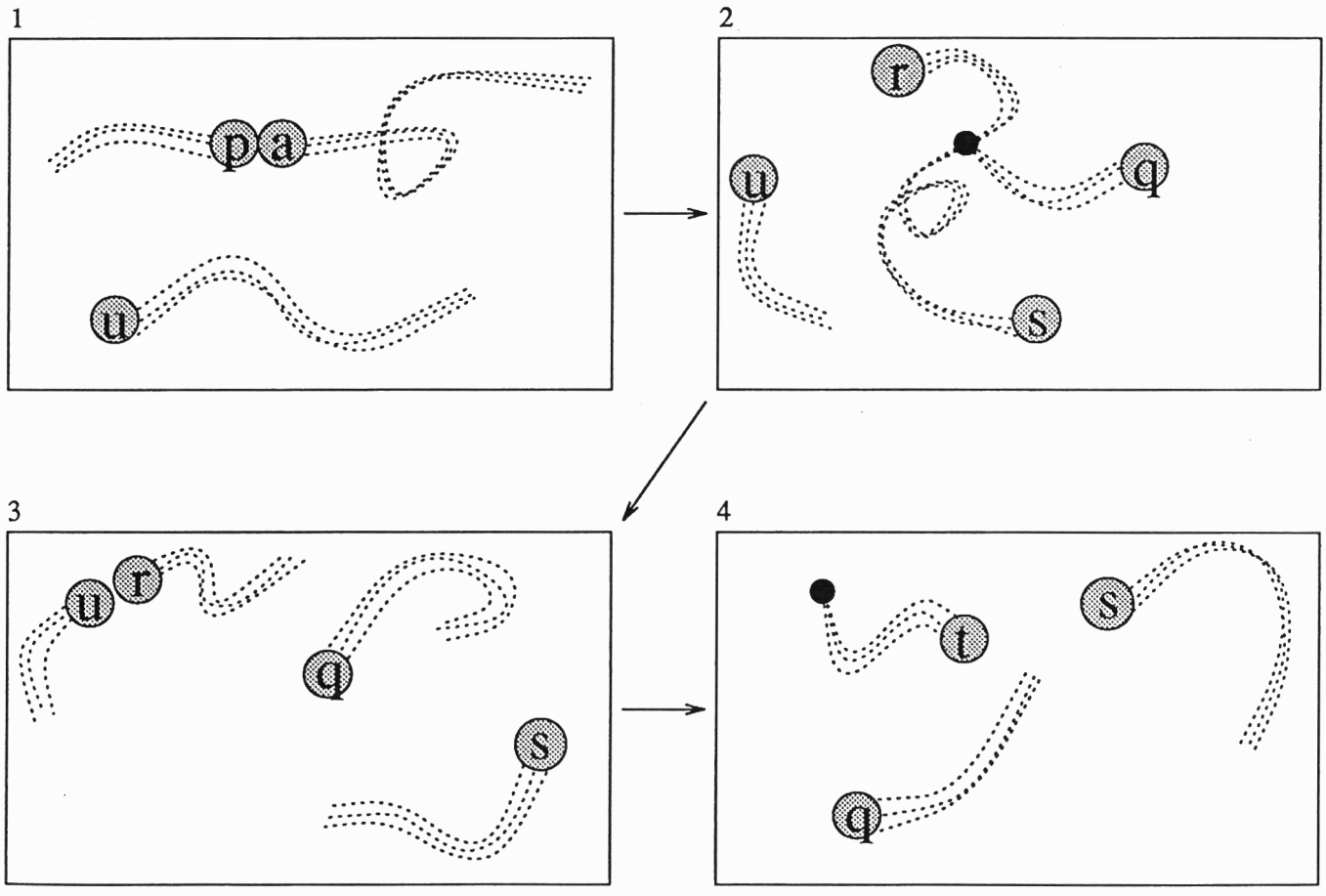
p ☤ a o— q ☤ r



"Thanks to J-M. Andreoli and R. Pareschi for this and the following three slides."

INTRA-SPACE INTERACTION





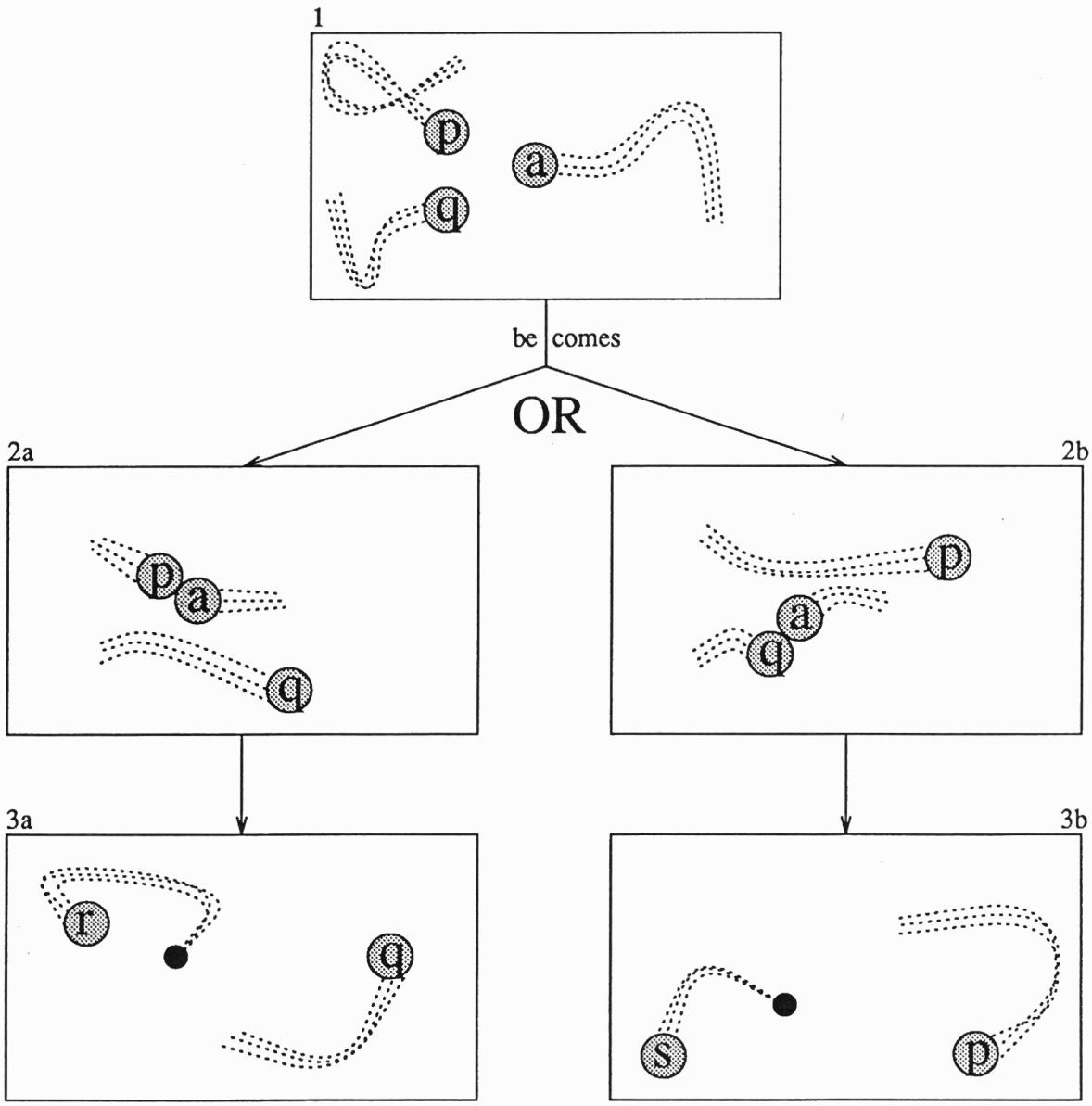
INTRA-SPACE INTERACTION

Characteristics

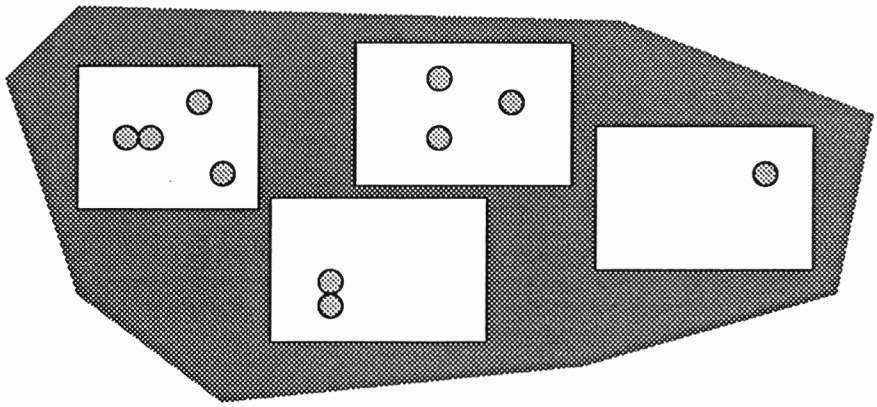
Concurrent and Competitive

p ∂ a o— r

q ∂ a o— s

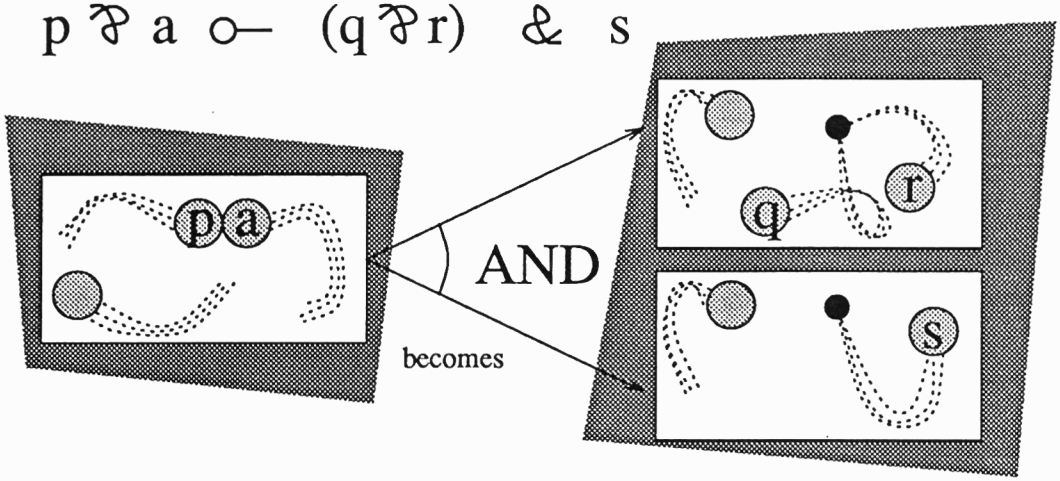


COMPUTING WITH MULTIPLE SPACES



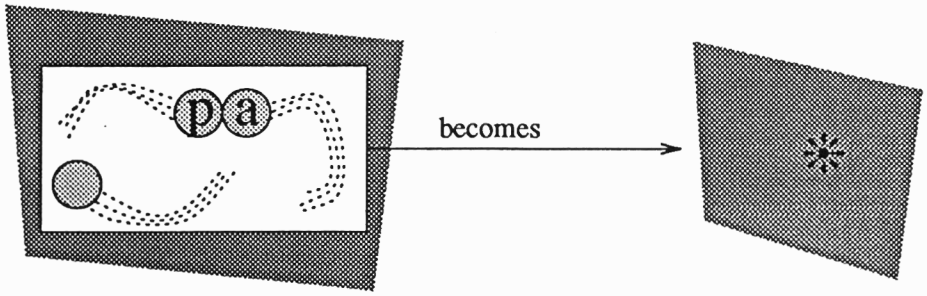
- Space Creation: Operator $\&$ (with) (cloning)

$p \parr a \multimap (q \parr r) \;\&\; s$



- Space Termination: Operator $\top$ (top)

$p \parr a \multimap \top$



CCS as an Example

Now allow quantification over propositions. If P is a formula and a is a constant then $a.P$ is a proposition and $\bar{a}.P$.

$$\forall a \forall P \forall Q \; [P \parr Q \multimap a.P \parr \bar{a}.Q]$$
$$\forall P \forall Q \; [P \multimap P + Q]$$
$$\forall P \forall Q \; [Q \multimap P + Q]$$
$$\forall P \; [\bot \multimap !P]$$
$$\forall P \; [!P \parr !P \multimap !P]$$
$$\forall P \; [P \multimap !P]$$

Notice that $\parr$ corresponds to parallel composition $|$ in CCS.

Work on Multiple-Conclusion Clauses

$$\forall \bar{x}[Body \multimap A_1 \parr \cdots \parr A_n]$$

1. Andreoli & Pareschi have developed LO (Linear Objects) on multiple-conlcusion clauses. *Body* may contain $\parr$, $\top$, &.
2. Miller has a presentation of CCS and the π-calculus using multiple-conclusion clauses. *Body* may contain $\parr$, $\bot$, $\forall$.
3. Saraswat & Lincoln have considered a dual translation (using $\otimes$ instead of $\parr$).
4. See other papers in these proceedings.

All of these systems propose a setting for addressing concurrency.

Single-Conclusion Clauses

The fragment of intuitionistic linear logic restricted to formulas freely generated by

$$\top, \&, \multimap, \supset, \forall$$

has been proposed by Hodas & Miller [Information and Computation, 1992] as a refinement to the logic underlying λProlog (that logic is based on $\top, \&, \supset, \forall$). Positive occurrences of $\oplus, \exists, \otimes, !, 1$ can be admitted and described using the clauses

$$\forall P \forall Q [P \multimap (P \oplus Q)]$$
$$\forall P \forall Q [Q \multimap (P \oplus Q)]$$
$$\forall B \forall T [(B\ T) \multimap (\exists B)]$$
$$\top \Rightarrow 1$$
$$\forall P \forall Q [P \multimap Q \multimap (P \otimes Q)]$$
$$\forall P [P \Rightarrow !\, P]$$

These rules describe the right-hand behavior of these connectives.

Proof Rules for $\top, \&, \multimap, \supset, \forall$

$$\frac{}{\Gamma; A \longrightarrow A}\ \text{identity} \qquad \frac{}{\Gamma; \Delta \longrightarrow \top}$$

$$\frac{\Gamma, B; \Delta, B \longrightarrow C}{\Gamma, B; \Delta \longrightarrow C}\ \textit{absorb}$$

$$\frac{\Gamma; \Delta, B_i \longrightarrow C}{\Gamma; \Delta, B_1 \,\&\, B_2 \longrightarrow C}$$

$$\frac{\Gamma; \Delta \longrightarrow B \quad \Gamma; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \,\&\, C}$$

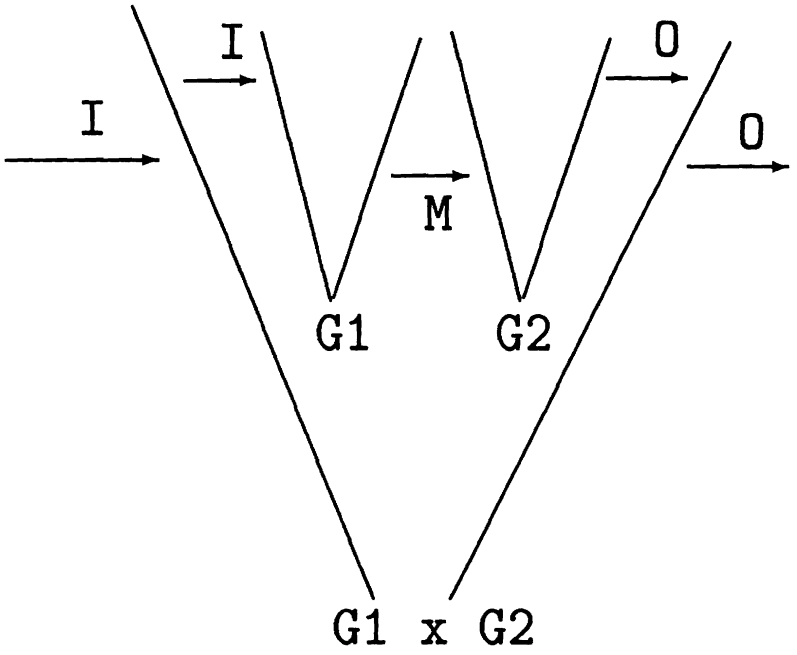
$$\frac{\Gamma; \Delta_1 \longrightarrow B \qquad \Gamma; \Delta_2, C \longrightarrow E}{\Gamma; \Delta_1, \Delta_2, B \multimap C \longrightarrow E}$$

$$\frac{\Gamma; \Delta, B \longrightarrow C}{\Gamma; \Delta \longrightarrow B \multimap C} \qquad \frac{\Gamma, B; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \supset C}$$

$$\frac{\Gamma; \emptyset \longrightarrow B \qquad \Gamma; \Delta, C \longrightarrow E}{\Gamma; \Delta, B \supset C \longrightarrow E}$$

$$\frac{\Gamma; \Delta, B[t/x] \longrightarrow C}{\Gamma; \Delta, \forall x.B \longrightarrow C} \qquad \frac{\Gamma; \Delta \longrightarrow B[y/x]}{\Gamma; \Delta \longrightarrow \forall x.B}$$

An Input-Output Interpretation of Proof Search



Formulas in the bounded part of context are treated as resources that can be consumed during the construction of proofs.

The proposition $I\{G\}O$ denotes the fact that it is possible to prove the goal G from the formulas in I leaving the formulas in O unused.

A Proof System for the I-O Interpretation

$$\frac{}{I\{1\}I} \qquad \frac{\mathsf{subcontext}(O,I)}{I\{\top\}O} \qquad \frac{I\{G\}I}{I\{!\,G\}I}$$

$$\frac{I\{G_1\}M \qquad M\{G_2\}O}{I\{G_1 \otimes G_2\}O}$$

$$\frac{I\{G_1\}O \qquad I\{G_2\}O}{I\{G_1 \,\&\, G_2\}O}$$

$$\frac{R{::}I\{G\}\mathsf{del}{::}O}{I\{R \multimap G\}O} \qquad \frac{!\,R{::}I\{G\}\,!\,R{::}O}{I\{R \Rightarrow G\}O}$$

$$\frac{\mathsf{pickR}(I,O,A)}{I\{A\}O}$$

$$\frac{\mathsf{pickR}(I,M,G \multimap A) \qquad M\{G\}O}{I\{A\}O}$$

$$\frac{\mathsf{pickR}(I,O,G \Rightarrow A) \qquad O\{G\}O}{I\{A\}O}$$

What is Logic Programming?

Programs are collections of logical formulas (theories).

Computation is a search for cut-free proofs.

Constract this to functional programming where programs are proofs and computation is proof reduction (cut-elimination).

Search for arbitary cut-free proofs, however, does not seem to capture the full spirit of logic programming.

In logic programming languages, the "search semantics" of a logical connective in a goal is independent from its surrounding context, which are only relevant for proving atomic formulas.

$$\Delta \longrightarrow \Gamma_1, B_1 \,\&\, B_2, B_3 \otimes B_4, \Gamma_2.$$

Formalizing Goal-Directed Search

A cut-free sequent proof Ξ is *uniform* if

- for every subproof Ψ of Ξ and
- for every non-atomic formula occurrence B in the right-hand side of the endsequent of Ψ,

there is a proof Ψ' that is

- equal to Ψ up to permutation of inference rules and
- is such that the last inference rule in Ψ' introduces the top-level logical connective occurring in B.

The given logic and proof system is called an *abstract logic programming language* if a sequent has a proof if and only if it has a uniform proof.

A Brief Guide to Linear Logic

ANDRE SCEDROV*

Department of Mathematics, University of Pennsylvania

Philadelphia, PA 19104-6395, USA

Abstract

An overview of linear logic is given, including an extensive bibliography and a simple example of the close relationship between linear logic and computation.

1 Overview

Linear logic, introduced by Girard [41], is a refinement of classical logic. Linear logic is sometimes described as *resource sensitive* because it provides an intrinsic and natural accounting of resources. This is indicated by the fact that in linear logic, two assumptions of a formula A are distinguished from a single assumption of A. Informally, on the level of basic intuition, one might say that classical logic is about truth, that intuitionistic logic is about construction of proofs, and that linear logic is about process states, events, or resources, which must be carefully accounted for.

A convenient way to present the syntax of linear logic is by modifying the traditional Gentzen-style sequent calculus axiomatization of classical logic,

*andre@cis.upenn.edu. Research partially supported by NSF Grant CCR-91-02753 and by ONR Grant N00014-92-J-1916. This is an updated version of the article that has originally appeared in the *Bulletin of the European Association for Theoretical Computer Science* vol. 41, June, 1990, pp. 154-165 in the column "Logic in Computer Science" edited by Yuri Gurevich. This version will appear in a collection of articles published by World Scientific and edited by G. Rozenberg.

which may be found in *e.g.*, Girard *et al.* [49] or in Gallier [38]. The modification may be briefly described in three steps, bearing in mind that assumptions are viewed as resources, and conclusions as requirements to be met by spending the given resources. In this reading the formula A *implies* A, for instance, means that the resource A is spent to meet the requirement A. This is an axiom in linear logic.

The first step in presenting linear logic as a modification of classical logic is to remove two "structural" rules, *contraction* and *weakening,* which manipulate the use of assumptions and conclusions in logical deductions. For expository purposes let us concentrate on the treatment of assumptions. The contraction rule states that if a property follows from two assumptions of a formula, then that property can be derived just from a single assumption of that formula. In effect, this means that any assumption, once stated, may then be reused as often as desired. For instance, the formula A *implies* (A *and* A) is derivable from contraction. The weakening rule makes it possible to form deductions that have dummy assumptions, *i.e.*, weakening allows us to carry out a deduction without using all of the assumptions. For instance, the formula (A *and* B) *implies* A is derivable from weakening. Because contraction and weakening together make it possible to use an assumption as often or as little as desired, these rules are responsible for what one may see in hindsight as a loss of control over resources in classical (and in intuitionistic) logic. This realization is the starting point of linear logic. Removing the rules of contraction and weakening produces a *linear* system in which each assumption must be used, nay, spent *exactly once*. In the resulting linear logic, formulas indicate finite resources that cannot necessarily be discarded or duplicated without effort.

The second step in deriving linear logic involves the propositional connectives. Briefly, the removal of structural rules just mentioned leads naturally to two forms of conjunction, one called *multiplicative* and the other *additive*, and similarly to two forms of disjunction. A proof of the multiplicative conjunction as a conclusion forbids any sharing between the resources used to establish each conjunct, whereas the additive conjunction requires the sharing of all of the resources. We should mention that unlike this distinction between the two forms of conjunction and disjunction, the quantifier rules remain the same as in classical logic.

The third step in the presentation of linear logic involves adding a kind of modality: a *storage* or *reuse* operator, ! . Intuitively, the assumption $!A$

provides unlimited use of the resource A. A computational metaphor that describes the meaning of $!A$ quite well is that "the datum A is stored in the memory and may be referenced an unlimited number of times". There is also a dual modal operator, $?$, which is definable from $!$ using negation. Intuitively, while $!A$ provides unlimited creation of A, the formula $?B$ allows the unlimited consumption of B.

Because the basic framework remains linear, unbounded consumption or reuse is allowed "locally", only at formulas specifically marked with $?$ or $!$, respectively. In effect, the structural rules of contraction and weakening are replaced by the explicit logical rules about the modalities. The resulting logical system is remarkably natural and well-structured, and it brings logical form and content closer together. Linear logic also indicates a relationship between classical logic and intuitionistic logic that is more subtle than the standard negative interpretation, see Girard [45, 46].

Detailed descriptions of linear logic rules may be found in, *e.g.*, [41, 68, 85]. Let us mention that the nonmodal fragment of linear logic was anticipated by a calculus proposed by Lambek [62, 63], motivated by linguistic considerations of syntax of natural languages.

A remarkable result of Lincoln and Winkler [71] shows that there is no simple minded truth table characterization of provability even for the multiplicative fragment of linear logic, unless P = NP (see below). In this sense semantics of linear logic is necessarily involved. Basic linear algebra constructions on finite dimensional vector spaces provide a first, naive interpretation of some of the linear logic connectives, much as the basic operations on sets provide an interpretation of the usual logical connectives. For instance, multiplicative conjunction may be interpreted as tensor product of vector spaces and linear negation may be interpreted as the dual vector space. More subtle aspects of linear logic, however, originate in so-called coherence domains [41], which maintain a notion of finite basis and isomorphism with the double dual without imposing isomorphism with the dual. Indeed, an important turning point in the discovery of linear logic was Girard's insight that in coherence domains, function type $A \Rightarrow B$ could be decomposed as $!A \multimap B$, where $!$ is the reuse operator mentioned above, and $\multimap$ is *linear implication*, which provides the type of functions that "use" their argument exactly once.

Similar phenomena had been previously observed by Blass in certain natural set-theoretic operations on infinite games [23] and by Barr in so-called *-autonomous categories [16]. Both settings, in addition to coherent domains,

are now understood to yield mathematical models for linear logic proofs, more precisely, for the relation "t is a linear logic proof of a formula A " [24, 84, 17]. Other versions of game semantics are given by Abramsky and Jagadeesan [2] and by Lafont and Streicher [61]. Event spaces, which come about from Pratt's work in semantics of concurrency, also provide models for certain linear logic proofs [78]. Models investigated by de Paiva [35] are motivated by important proof-theoretic transformations. A mathematical model for the linear logic provability relation is given by phase spaces, discussed by Girard [41] and by Avron [14, 15]. Kripke-style models are investigated by Allwein and Dunn [6].

A mathematical structure underlying linear logic proof reduction and normalization is provided by proof nets, introduced by Girard [41, 42, 47] and studied by Danos and Regnier [34], Blute [25, 26], and others. A significant insight into dynamic, operational semantics of linear logic proof reduction (cut elimination) is provided by geometry of interaction, proposed by Girard [43, 44]; an exposition is in Danos [32]. Geometry of interaction is further investigated in the work of Danos [33], Regnier [79], Malacaria and Regnier [74], Abramsky and Jagadeesan [3], and Gonthier *et al.* [51, 52] (see below).

Linear logic continues to be a very active field of research. Up-to-date developments are discussed on the electronic forum "Linear". One subscribes by sending an email message to linear-request@cs.stanford.edu. An introductory description of linear logic is in Lincoln [65]. The first book on linear logic, by Troelstra [85], was published in 1992. It provides a valuable first reference for anyone interested in studying the subject.

2 Computational Aspects

Computer science ramifications of linear logic may be generally divided into two kinds. In what may be broadly called functional programming ramifications (often involving certain aspects of concurrency), computation is seen as term reduction corresponding to proof reduction (cut elimination). On the other hand, in what may be broadly called logic programming ramifications (again, often involving concurrency), computation is expressed by cut free proof search in certain linear logic theories. From the latter perspective, the cut elimination property is used simply to allow one to concentrate on cut free proofs without loss of generality. Certain *permutability* properties,

which yield optimized presentations of cut free proofs play a central role in the proof search paradigm, see Lincoln [72], Andreoli [7].

2.1 Proof Reduction Paradigm

The earliest work in this direction was Lafont's investigation of a functional programming language implementation in which garbage collection was replaced by explicit duplication operations based on linear logic [59]. Another possible application in functional programming is in optimization of copying in lazy functional programming language implementation ("single-threadedness"), studied by Guzmán and Hudak [54]. Recent topics involve linear lambda calculus and memory allocation, investigated by Lincoln and Mitchell [66], Chirimar *et al.* [30], Wadler [86], Mackie [73], and Benton *et al.* [22].

A strong relationship of the multiplicative fragment of linear logic to Petri nets has been demonstrated by Gehlot and Gunter [53, 40, 39], Asperti *et al.* [11, 13], Engberg and Winskel [37], Marti-Oliet and Meseguer [75], and Brown and Gurr [27]. Interpretations of linear logic proofs in concurrent paradigms such as the chemical abstract machine or Milner's π-calculus are given by Abramsky [1] and by Bellin and P.J. Scott [21]. With regard to concurrency, we also note a remarkable similarity between a programming paradigm based on proof nets and developed by Lafont [60] and connection graphs, which were designed by Bawden to model the massively parallel connection machine computations [18].

A modular setting for polynomial time computations can be given in a *bounded linear logic*, studied by Girard *et al.* [50]. In this system unlimited reuse is not allowed. Instead, bounded linear logic contains bounded or limited reuse operators. A bounded reuse operator of order n indicates that a datum is stored in the memory and may be referenced at most n times, where variable n is the bound. Proof-rules of the sequent calculus then naturally generate polynomial bounds. Bounded linear logic has *polynomial time cut elimination*. This system might serve as a basis for a modular calculus of efficient algorithms.

Recent topics include the use of geometry of interaction by Gonthier *et al.* [51, 52] in a correctness proof for Lamping's graph reduction. (A companion reference is Asperti [12].) Lamping [64] discovered an optimal graph-

reduction implementation of the lambda calculus, independently of Girard's work on geometry of interaction (see above). Gonthier *et al.* show how the geometry of interaction provides a suitable semantic basis for explaining and improving Lamping's system. On the other hand, Gonthier *et al.* show that graphs similar to Lamping's and related to Lafont's nets (see above) provide a concrete representation of the geometry of interaction.

2.2 Proof Search Paradigm

Linear logic is also related to computation in another way, namely, computation may be expressed by cut free proof search in certain linear logic theories. The remarkable expressiveness of this computational paradigm is brought to light in a body of research accumulated over the past three years. In a number of papers Andreoli and Pareschi develop a declarative treatment of object communication and concurrent object-oriented computation [8, 9, 7, 10]. A treatment of linear logic programming is given by Hodas and Miller [55]. A role of linear logic in a declarative semantics of SLDNF-resolution is considered by Cerrito [29]. An approach that spans both the proof reduction and the proof search paradigms is proposed by Girard [48].

Recent topics include a treatment of concurrent constraint programming by Saraswat and Lincoln [82] and a closely related treatment of Milner's π-calculus of communicating processes by Miller [76]. In Miller's work the agent expressions of the π-calculus are translated into a theory of linear logic in such a way that the π-calculus reductions are identified, step by step, with cut free proof search in the linear logic theory. The nonlogical axioms of the theory resemble Horn clauses except that they may have multiple conclusions. In particular, their heads may be the multiplicative disjunction of atomic formulas. Such multiple conclusion clauses are used to axiomatize communications among agents.

The striking expressiveness of cut free linear logic proof search as a computational paradigm is also indicated by the complexity and undecidability of (provability in) the natural fragments of linear logic. This comes about as a consequence of direct, lockstep simulations of computations on generic machines by cut free proof search in fragments of linear logic, see Section 3 below. The simulations reveal a structural relationship between the natural fragments of linear logic on the one hand and the standard computa-

tional complexity classes on the other. Lincoln *et al.* [68] show that the full propositional (*i.e.*, quantifier free) linear logic is undecidable. However, even the fragment of propositional linear logic that does not allow modalities is unexpectedly expressive. Kanovich [57] shows that the multiplicative fragment is NP-complete. Furthermore, in a marked contrast to the standard NP-completeness of the satisfiability of propositional formulas in classical logic, in linear logic even the decision problem for *constant-only* multiplicative propositional formulas is NP-complete. This result, obtained by Lincoln and Winkler [71], shows that a simple minded, efficient, truth-table style characterization of provability in the multiplicative fragment of linear logic would imply that P = NP. Beyond the multiplicatives, Lincoln *et al.* [68] show PSPACE-completeness of the nonmodal propositional fragment (*i.e.*, the multiplicative-additive propositional fragment). Lincoln *et al.* [70] exhibit a structural embedding of a cut free proof system for implicational propositional intuitionistic logic in the nonmodal propositional linear logic. Lincoln and Scedrov [69] obtain NEXPTIME-hardness of nonmodal linear logic with first order quantifiers and function symbols.

3 Example

This section describes an example of the ability of linear logic to express computational features. A simple computation on an ordinary two counter machine with zero-test instruction is simulated, step by step, by cut free proof search in propositional linear logic. A key insight is that searching for a certain kind of proof of a linear logic formula from finitely many nonlogical axioms that involve only multiplicative conjuction $\otimes$ and additive disjunction $\oplus$ corresponds directly to searching for an accepting computation. The product of a successful search is an accepting computation. This example is taken from Lincoln *et al.* [68, 67].

Suppose the transition relation δ of a standard two counter machine with zero-test consists of the following:

$$\begin{aligned} \delta_1 &::= Q_I \textbf{ Increment } A\ Q_2 \\ \delta_2 &::= Q_3 \textbf{ Decrement } A\ Q_F \\ \delta_3 &::= Q_2 \textbf{ Zero-Test } B\ Q_3 \end{aligned}$$

The machine may perform the following transitions, where an instantaneous

description of a two counter machine is given by the triple consisting of Q_j, the current state, and the values of counters A and B.

$$\langle Q_I, 0, 0\rangle \xrightarrow{\delta_1} \langle Q_2, 1, 0\rangle \xrightarrow{\delta_3} \langle Q_3, 1, 0\rangle \xrightarrow{\delta_2} \langle Q_F, 0, 0\rangle$$

This computation starts in state Q_I, increments the A counter and steps to state Q_2. Then it tests the B counter for zero, and moves to Q_3, where it then decrements the A counter, moves to Q_F, and accepts.

The transition relation δ may be transformed into a transition relation δ' for an equivalent and-branching two counter machine *without* zero-test, or briefly ACM. The modified relation δ' (shown on the left below), may then be encoded as a linear logic theory (shown on the right):

		Transitions	*Theory Axioms*
δ'_1	$::=$	Q_I **Increment** A Q_2	$q_I \vdash (q_2 \otimes a)$
δ'_2	$::=$	Q_3 **Decrement** A Q_F	$q_3, a \vdash q_F$
δ'_3	$::=$	Q_2 **Fork** Z_B, Q_3	$q_2 \vdash (z_B \oplus q_3)$
δ'_4	$::=$	Z_B **Decrement** A Z_B	$z_B, a \vdash z_B$
δ'_5	$::=$	Z_B **Fork** Q_F, Q_F	$z_B \vdash (q_F \oplus q_F)$

Notice how the first two transitions (δ_1 and δ_2) of the standard two counter machine are preserved in the translation from δ to δ'. Also, the Zero-Test instruction δ_3 is encoded as three ACM transitions — δ'_3, δ'_4, and δ'_5. The transition δ'_3 is a fork to a special state Z_B, and one other state, Q_3. The two extra transitions, δ'_4 and δ'_5, force the computation branch starting in state Z_B to verify that counter B is zero. Given the above transitions, the and-branching machine without zero-test may then perform these moves:

$$\{\langle Q_I, 0, 0\rangle\} \xrightarrow{\delta'_1} \{\langle Q_2, 1, 0\rangle\} \xrightarrow{\delta'_3} \{\langle Z_B, 1, 0\rangle, \langle Q_3, 1, 0\rangle\} \xrightarrow{\delta'_4} \{\langle Z_B, 0, 0\rangle, \langle Q_3, 1, 0\rangle\}$$
$$\xrightarrow{\delta'_5} \{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_3, 1, 0\rangle\} \xrightarrow{\delta'_2} \{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle\}$$

Note that an instantaneous description of this and-branching machine is a list of triples, and the machine accepts if and only if it is able to reach $\langle Q_F, 0, 0\rangle$ in *all* branches of its computation. This particular computation starts in state Q_I, increments the A counter and steps to state Q_2. Then it forks into two separate computations; one which verifies that the B counter is zero, and the other which proceeds to state Q_3. The B counter is zero, so the proof of that branch proceeds by decrementing the A counter to zero, and jumping

to the final state Q_F. The other branch from state Q_3 simply decrements A and moves to Q_F. Thus all branches of the computation terminate in the final state with both counters at zero, resulting in an accepting computation.

The linear logic proof corresponding to this computation is displayed in Figures 1 and 2, and is explained in the following paragraphs. In these proofs, each application of a theory axiom corresponds to one step of ACM computation. We represent the values of the ACM counters in unary by copies of the formulas a and b. In this example the B counter is always zero, so there are no occurrences of b.

The proof shown in Figure 1 of $z_B, a \vdash q_F$ in the above linear logic theory corresponds to the ACM verifying that the B counter is zero. Reading the proof bottom up, it begins with a directed cut. The sequent $z_B \vdash q_F$ is left as an intermediate step. The next step is to use another directed cut, and after application of the $\oplus$ **L** rule, we have two sequents left to prove: $q_F \vdash q_F$ and $q_F \vdash q_F$. Both of these correspond to the ACM triple $\langle Q_F, 0, 0\rangle$ which is the accepting triple, and are provable by the identity rule. If we had attempted to prove this sequent with some occurrences of b, we would be unable to complete the proof.

$$\dfrac{\dfrac{}{z_B, a \vdash z_B}\,\delta'_4 \qquad \dfrac{\dfrac{}{z_B \vdash (q_F \oplus q_F)}\,\delta'_5 \qquad \dfrac{\dfrac{}{q_F \vdash q_F}\,\mathbf{I} \quad \dfrac{}{q_F \vdash q_F}\,\mathbf{I}}{(q_F \oplus q_F) \vdash q_F}\,\oplus\mathbf{L}}{z_B \vdash q_F}\,\mathbf{Cut}}{z_B, a \vdash q_F}\,\mathbf{Cut}$$

Figure 1: Zero-test proof

The proof shown in Figure 2 of $q_I \vdash q_F$ in the same theory demonstrates the remainder of the ACM machinery. The lowermost introduction of a theory axiom, **Cut**, and $\otimes$ **L** together correspond to the application of the increment instruction δ'_1. That is, the q_I has been "traded in" for q_2 along with a. The application of a directed cut and $\oplus$ **L** correspond to the fork instruction, δ'_3 which requires that both branches of the proof be successful in the same way that and-branching machines require all branches to reach an accepting configuration. The indicated proof of $z_B, a \vdash q_F$ appears in Figure 1, and corresponds to the verification that the B counter is zero.

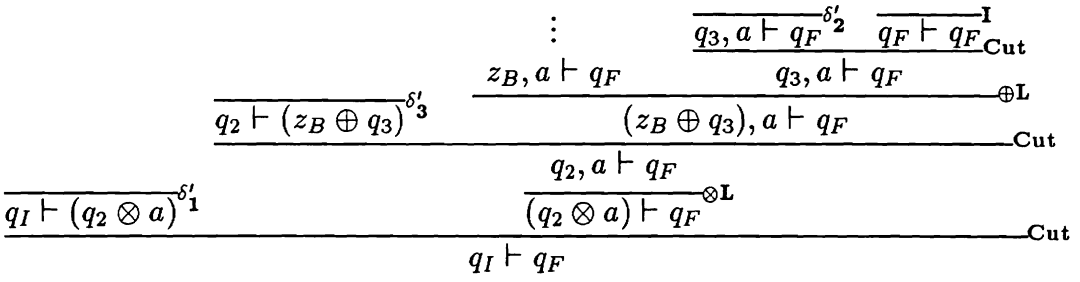


Figure 2: Proof corresponding to computation

The application of **Cut**, theory axiom, and identity correspond to the final decrement instruction of the computation, and complete the proof.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 1992. Special Issue on the 1990 Workshop on Math. Found. Prog. Semantics. To appear.

[2] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. Manuscript, September 1992. Available using anonymous ftp from the host theory.doc.ic.ac.uk and the file theory/papers/Abramsky/gfc.dvi.

[3] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proc. 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 211–222. IEEE Computer Society Press, Los Alamitos, California, June 1992.

[4] V.M. Abrusci. Noncommutative intuitionistic linear propositional logic. *Zeitschr. f. Math. Logik u. Grundlagen d. Math.*, 36:297–318, 1990.

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

[6] G. Allwein and J.M. Dunn. Kripke models for linear logic. Manuscript, January 1992.

[7] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 1992. To appear.

[8] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems: a linear logic approach. In *Proc. Workshop on Extensions of Logic Programming, Tuebingen*. Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 1990.

[9] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9, 1991.

[10] J.-M. Andreoli and R. Pareschi. Associative communication and its optimization via abstract interpretation. Manuscript, August 1992.

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

[12] A. Asperti. Linear logic, comonads and optimal reductions. Manuscript, December 1991.

[13] A. Asperti, G.-L. Ferrari, and R. Gorrieri. Implicative formulae in the 'proofs as computations' analogy. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 59–71, January 1990.

[14] A. Avron. The semantics and proof theory of linear logic. *Theoretical Computer Science*, 57:161–184, 1988.

[15] A. Avron. Some properties of linear logic proved by semantic methods. Manuscript, December 1991.

[16] M. Barr. **-Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1979.

[17] M. Barr. Accessible categories and models of linear logic. *Journal Pure Appl. Algebra*, 69:219–232, 1990.

[18] A. Bawden. Connection graphs. In *Proc. ACM Symp. on Lisp and Functional Programming*, pages 258–265, 1986.

[19] G. Bellin. *Mechanizing Proof Theory: Resource-Aware Logics and Proof-Transformations to Extract Implicit Information.* PhD thesis, Stanford University, 1990.

[20] G. Bellin and J. Ketonen. A decision procedure revisited: Notes on direct logic, linear logic, and its implementation. *Theoretical Computer Science*, 95:115–142, 1992.

[21] G. Bellin and P.J. Scott. Remarks on the π-Calculus and Linear Logic. Manuscript to be submitted to Proc. MFPS 8, Oxford, 1992.

[22] N. Benton, G. Bierman, V. de Paiva, and J.M.E. Hyland. Term assignment for intuitionistic linear logic. Manuscript, September 1992.

[23] A. Blass. Degrees of indeterminacy of games. *Fundamenta Mathematicae*, 77:151–166, 1972.

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

[25] R. Blute. Linear logic, coherence, and dinaturality. Dissertation, University of Pennsylvania, September 1991. Edited version to appear in *Theoretical Computer Science.*

[26] R. Blute. Proof nets and coherence theorems. In *Category Theory and Computer Science, Proceedings 1991.* Springer LNCS 530, 1991.

[27] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[28] C. Brown and D. Gurr. Relations and noncommutative linear logic. Manuscript, December 1991.

[29] S. Cerrito. A linear semantics for allowed logic programs. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[30] J. Chirimar, C. Gunter, and J. Riecke. Linear ML. In *Proc. ACM Symposium on Lisp and Functional Programming, San Francisco.* ACM Press, June 1992.

[31] J. Chirimar and J. Lipton. Provability in TBLL: A decision procedure. In *Computer Science Logic, Proceedings 1991*, pages 341–356. Springer LNCS 626, 1992.

[32] V Danos. Logique linéaire. une représentation algébrique du calcul. *Gaz. Math. (Soc. Math. de France)*, 41:55–64, 1989.

[33] V. Danos. La Logique Linéaire Appliquée à l'Étude de Divers Processus de Normalisation at Principalement du Lambda-Calcul. Thèse de Doctorat, Université de Paris VII, 1990.

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

[35] V.C.V. de Paiva. A dialectica-like model of linear logic. In *Category Theory and Computer Science*, pages 341–356. Springer LNCS 389, September 1989.

[36] K. Dosen. Nonmodal classical linear predicate logic is a fragment of intuitionistic linear logic. *Theoretical Computer Science*, 102:207–214, 1992.

[37] U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *Proceedings of CAAP '90. Lecture Notes in Computer Science vol. 431, Springer-Verlag*, 1990.

[38] J.H. Gallier. *Logic for computer science.* Harper & Row Publishers, New York, 1986.

[39] V. Gehlot. A proof-theoretic approach to semantics of concurrency. Dissertation, University of Pennsylvania, 1991.

[40] V. Gehlot and C.A. Gunter. Normal process representatives. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[41] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[42] J.-Y. Girard. Multiplicatives. *Rendiconti del Seminario Matematico dell' Universitá e Politecnico Torino, Special Issue on Logic and Computer Science*, pages 11–33, 1987.

[43] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Logic Colloquium '88*, Amsterdam, 1989. North-Holland.

[44] J.-Y. Girard. Geometry of interaction II: Deadlock-free algorithms. In: Springer LNCS 417, 1990.

[45] J.-Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.

[46] J.-Y. Girard. On the unity of logic. Manuscript, 1991.

[47] J.-Y. Girard. Quantifiers in linear logic II. Prépublications Paris 7 Logique, No. 19, January 1991.

[48] J.-Y. Girard. A fixpoint theorem in linear logic. Announcement on 'Linear' electronic forum , February 1992.

[49] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1989.

[50] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992.

[51] G. Gonthier, M. Abadi, and J.-J. Levy. The geometry of optimal lambda reduction. In *Proc. 19-th Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico.* ACM Press, New York, NY, January 1992.

[52] G. Gonthier, M. Abadi, and J.-J. Levy. Linear logic without boxes. In *Proc. 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 223–234. IEEE Computer Society Press, Los Alamitos, California, June 1992.

[53] C.A. Gunter and V. Gehlot. Nets as tensor theories. In G. De Michelis, editor, *Proc. 10-th International Conference on Application and Theory of Petri Nets, Bonn*, pages 174–191, 1989.

[54] J.C. Guzman and P. Hudak. Single-threaded polymorphic lambda calculus. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[55] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 32–42. IEEE Computer Society Press, Los Alamitos, California, July 1991. Full paper to appear in *Information and Computation.* Draft available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/miller/ic92.dvi.Z.

[56] B. Jacobs. Semantics of weakening and contraction. Manuscript, May 1992.

[57] M. Kanovich. Horn programming in linear logic is NP-complete. In *Proc. 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 200–210. IEEE Computer Society Press, Los Alamitos, California, June 1992.

[58] J. Ketonen and R. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32, 1984.

[59] Y. Lafont. The linear abstract machine. *Theoretical Computer Science*, 59:157–180, 1988.

[60] Y. Lafont. Interaction nets. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 95–108, January 1990.

[61] Y. Lafont and T. Streicher. Game semantics for linear logic. In *Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 43–50. IEEE Computer Society Press, Los Alamitos, California, July 1991.

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

[63] J. Lambek. Multicategories revisited. In *Categories in Computer Science and Logic*, pages 217–239. Contemporary Math., vol. 92, American Math. Soc., Providence, RI, 1989.

[64] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. 17-th Annual ACM Symposium on Principles of Programming Languages, San Francisco*, pages 16–30. ACM Press, New York, NY, January 1990.

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

[66] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *Proc. 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 235–246. IEEE Computer Society Press, Los Alamitos, California, June 1992.

[67] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pages 662–671, 1990.

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

[69] P. Lincoln and A. Scedrov. First order linear logic without modalities is NEXPTIME-hard. Manuscript, September 1992. Available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/scedrov/mall1.dvi.

[70] P. Lincoln, A. Scedrov, and N. Shankar. Linearizing intuitionistic implication. In *Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 51–62. IEEE Computer Society Press, Los Alamitos, California, July 1991. Full paper to appear in *Annals of Pure and Applied Logic*. Draft available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/scedrov/lss91.dvi.

[71] P. Lincoln and T. Winkler. Constant-Only Multiplicative Linear Logic is NP-Complete. Manuscript, September 1992. Available using anonymous ftp from host ftp.csl.sri.com and the file pub/lincoln/comult-npc.dvi.

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

[73] I. Mackie. Lilac - a functional programming language based on linear logic. Master's Thesis, Imperial College, London, 1991.

[74] P. Malacaria and L. Regnier. Some results on the interpretation of λ-calculus in operator algebras. In *Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 63–72. IEEE Computer Society Press, Los Alamitos, California, July 1991.

[75] N. Marti-Oliet and J. Meseguer. From Petri nets to linear logic. In: Springer LNCS 389, ed. by D.H. Pitt et al., 1989. 313-340.

[76] D. Miller. The π-calculus as a theory in linear logic: Preliminary results. Technical Report MS-CIS-92-48, Computer Science Department, University of Pennsylvania, June 1992. Submitted. Available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/miller/pic.dvi.Z.

[77] H. Ono. Phase structures and quantales — a semantical study of logics without structural rules. Manuscript, 1991.

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

[79] L. Regnier. λ-calcul et Réseaux. Thèse de Doctorat, Université de Paris VII, 1992.

[80] D. Roorda. Resource logics: proof-theoretical investigations. Dissertation, University of Amsterdam, September 1991.

[81] K.I. Rosenthal. Girard quantaloids. *Mathematical Structures in Computer Science*, 2:93–108, 1992.

[82] V. Saraswat and P. Lincoln. Higher-order, linear, concurrent constraint programming. Manuscript, August 1992.

[83] H. Schellinx. Some syntactical observations on linear logic. *Journal of Logic and Computation*, 1:537–559, 1991.

[84] R.A.G. Seely. Linear logic, *-autonomous categories, and cofree coalgebras. In *Categories in Computer Science and Logic*, pages 371–382. Contemporary Math., vol. 92, American Math. Soc., Providence, RI, 1989.

[85] A.S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes No. 29. Center for the Study of Language and Information, Stanford University, 1992.

[86] P. Wadler. There's no substitute for linear logic. Manuscript, December 1991.

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

A Synopsis on the Identification of Linear Logic Programming Languages (Extended Abstract)[1]

James Harland
University of Melbourne
Australia

David Pym
University of Edinburgh
Scotland, U.K.

We investigate the definition of logic programming languages within linear logic [Gir87]. We take as our point of conceptual departure the *uniform proofs* of Miller *et al.*, [MNPS91], [Mil89] and the class of hereditary Harrop formulae of intuitionistic logic, although our aims are more foundational. In particular, we assume that the characteristic feature of logic programming is goal-directed proof-search. More precisely, there is a search operation corresponding to each logical connective, and when searching for a proof of a given goal one applies the search operation that corresponds to the outermost connective of that goal, and then to the outermost connective of each subgoal so generated, *etc.*. Furthermore, we assume that it must be possible to rewrite the program so that just one left rule, an appropriate notion of *resolution*[2] rule is required.

In *linear logic*, the identification of a class of computationally appealing proofs, uniform proofs, is somewhat more intricate than in logics that have been considered previously. For example, in first-order *intuitionistic* or *minimal* hereditary Harrop formulae [MNPS91], [Mil89], the search operation corresponding to each connective that can occur in a goal is given by the right rule for that connective. It turns out that for the whole intuitionistic hereditary Harrop fragment, in which just $\wedge$, $\supset$ and $\forall$ may occur negatively (the *definite formulae*) and $\wedge$, $\vee$, $\supset$, $\forall$ and $\exists$ may occur positively (the *goal formulae*), the strategy of constructing proofs (from root to leaves) by applying a right rule wherever one is applicable is complete. This property arises from the permutability properties of the rules of the (classical and) intuitionistic sequent calculus first studied by (Curry and) Kleene [Kle52, MNPS91, HP91, HP92].

The main novelty in linear logic is that for the desired classes of definite formulae and goal formulae, it may be necessary, under certain circumstances, to apply a left rule in the middle of a sequence of right rules in order to maintain completeness. In particular, we need to allow the $\otimes$-L and C!-L (contraction on the left) rules to occur immediately after the $\otimes$-R rule, and various left rules after the !-R rule [HP91, HP92]. One solution to this problem would be to exclude $\otimes$ and ! from the class of definite formulae in linear logic. However, this is both undesirable and unnecessary. It turns out that these exceptions to the strategy of applying right rules wherever they are applicable can be eliminated in the

[1]This work was supported in part by ESPRIT BRA, "Logical Frameworks"; and U.K. SERC grant GR/G 58588, "Logical and Semantical Frameworks"; and by a grant of the Australian Research Council.

[2]We stress that our notion of resolution is an *analytic* one; *cf.* classical Horn clause resolution, in which resolution amounts to cut together with unification.

next step of our analysis, in which, just as in the setting of intuitionistic hereditary Harrop formulae, we rewrite the antecedents to a certain *clausal form* for which a single left rule, the appropriate notion of resolution rule, is complete.

Thus we show in [HP91, HP92] how an analysis of the permutability properties of the two-sided linear sequent calculus can be used to determine fragments of linear logic for which a suitable notion of resolution proof can be defined. A summary of this analysis consists in essentially two, mutually dependent, steps: (i) The identification of the class of formulae such that the strategy of constructing proofs (considering rules to be reduction operators, in the sense of Kleene) [Kle68] by applying right rules wherever they are applicable is complete, subject to certain exceptions to this strategy, which must be handled, *i.e.*, made goal-directed, by the second step. This class of proofs, constructed by the application of the right rules wherever possible, subject to the exceptions to be handled by step (ii), is the appropriate class of *uniform* proofs; (ii) The exploitation of the structural properties of the antecedents of linear sequents so that exceptions to the right rule-first strategy can be eliminated and so that we can invoke a single left rule, a *resolution* rule whilst retaining (soundness and) completeness with respect to linear sequent calculus. This is achieved by introducing a mapping which reduces antecedents to multisets of *clauses*.

This analysis leads us to identify the following classes of formulae as being available for use as a linear logic programming language:

$$\begin{array}{lllll} \textit{Definite formulae} & D & ::= & A \mid \mathbf{1} \mid \bot \mid D \,\&\, D \mid D \otimes D \mid D \mathbin{⧟} D \mid G \multimap A \mid \bigwedge x \,.\, D \mid \,!\, D \\ \textit{Goal formulae} & G & ::= & A \mid \mathbf{1} \mid \bot \mid \top \mid D^{\bot} \mid G \otimes G \mid G \oplus G \mid G \mathbin{⧟} G \mid G \,\&\, G \\ & & & \mid D \multimap G \mid \bigwedge x \,.\, G \mid \bigvee x \,.\, G \mid \,!\, G \mid \,?\, G, \end{array}$$

where A ranges over atomic formulae. Programs are linear antecedents that consist of just closed definite formulae and goals are linear succedents that consist of just closed goal formulae.

Of course, there are other, more pragmatically motivated, influences on the choices of definite formulae, and indeed on the choices of goal formulae. We remark here that if we were to restrict definite formulae to be just the clauses permitted by (ii), above, we should be forced to constrain goal formulae rather more than is essential.

It remains an open problem to determine precisely the maximal class of linear formulae for which suitable notions of goal-directed proof are complete. However, some upper and lower bounds may be given. The main omissions from the class of definite formulae are those of the form $D_1 \oplus D_2$, $\bigvee x.D$ and $?D$, as well as the constant $\mathbf{0}$. The first two cases should not be surprising; note that in the intuitionistic case, the formulae $D_1 \vee D_2$ and $\exists x.D$ are not allowed in definite formulae, as the relevant permutation properties do not hold otherwise. The same property holds in the linear case for $D_1 \oplus D_2$ and $\bigvee x.D$, and also for $?D$. The omission of $\mathbf{0}$ is also not peculiar to linear logic, but essentially a denial of the principle of *ex nihil quodlibet*; it seems difficult to reconcile the notion of goal-directed provability with the provability of $\Gamma, \mathbf{0} \vdash \Delta$ for an arbitrary Δ. Thus it would seem that the omission of these four cases is necessary to preserve the notion of goal-directed provability.

The situation is less clear cut for some of the other omissions. Whilst the permutation properties appear to disqualify $G^{\bot}$ as a definite formula on the same grounds as $D_1 \oplus D_2$,

$\bigvee x.D$ and $?D$, $G^{\perp}$ is linearly equivalent to $G \multimap \perp$, and as $\perp$ may appear in goals, it seems reasonable to suggest that formulae of the form $G \multimap \perp$ may be included in the class of definite formulae. This form of negated formulae may considered as a way of ensuring that definite formulae containing negations may only be "used" at particular places in a proof.

Another "grey area" is for formulae of the form $G \multimap D$, rather than merely $G \multimap A$ where A is an atom. In the intuitionistic case, it is known that a formula of the form $G \supset D$ may be converted into an equivalent set of definite formulae in which only formulae of the form $G \supset A$ appear. However, this property does not hold in linear logic — for example, $p \multimap (q \otimes r)$ is not linearly equivalent to $(p \multimap q) \otimes (p \multimap r)$. Hence in the linear case, the addition of formulae of the form $G \multimap D$ increases both expressibility and power. The main reason that formulae of the form $G \multimap A$ are desirable is that they allow *simple* proofs to be used, *i.e.,* that the $\supset$-L rule need only be used as a unary rule, which in turn leads to the familiar notion of unifying an atom with the head of a clause. Clearly such a notion of proof cannot be used in the presence of formulae of the form $G \multimap D$. However, uniform provability remains complete for such formulae, and so we may choose to maintain either simple proofs or formulae of the form $G \multimap D$, but not both. In this sense the notion of maximality will depend upon whether we consider goal-directed provability to include the notion of simple proof or not. Another possibility is to restrict D to a subclass of definite formulae with a correspondingly restricted class of subproofs appearing on the right of $\multimap$-L, such as allowing the subproof to contain only the rules $\Cross$-L, & -L and $\otimes$-L. Such a straightforward modification of our notion of uniform proof permits the presence of definite formulae of the form $G \multimap (A_1 \Cross \ldots \Cross A_m)$, for example. This seems to be an interesting compromise between simple proofs and allowing arbitrary (uniform) proofs on the right of $\multimap$-L; in particular, no right rules or occurrences of $\multimap$-L will appear in such subproofs, and so the subproof resembles a particularly simple form of clausal decomposition. Whatever choice is made, it is clear that the answer to the maximality question depends very much on the the way in which we allow the $\multimap$-L rule to be used.

Hence the above class of formulae may not be the "ultimate" linear logic programming language, but it would seem that it is not very far from it, and that it will be a subtle and/or intricate task to expand the language in significant ways. The more abstract problem of identifying, for a given class of proofs (such as uniform proofs), the maximal class of formulae for which that class of proofs is complete with respect to provability remains an intriguing open problem and the subject of active research.

Whilst resolution proofs determine the nature of the search primitives needed to mechanically implement this language as far as is possible *logically*, they are not, in *operational* terms, deterministic. In order describe *interpreters* we must have a suitably deterministic, operational characterization of proofs. For example, consider the $\otimes$-rule of [HP92]:

$$\frac{\mathcal{D}_0 \vdash_R G_1, \mathcal{G}_0 \quad \mathcal{D}_0' \vdash_R G_2, \mathcal{G}_0' \quad \mathcal{D}_1 \vdash_R \mathcal{G}_1 \ \ldots \ \mathcal{D}_n \vdash_R \mathcal{G}_n}{\mathcal{D} \vdash_R G_1 \otimes G_2, \mathcal{G}},$$

where $C \in \mathcal{D}$ and $\{\mathcal{D}_0, \mathcal{D}_0' \vdash_R G_1 \otimes G_2, \mathcal{G}_0, \mathcal{G}_0'\} \cup \bigcup_{i=1}^{n} \{\mathcal{D}_i \vdash_R \mathcal{G}_i\}$ is a multiset of *C-components* of the sequent $\mathcal{D} \vdash_R G_1 \otimes G_2, \mathcal{G}$. In this rule, read from conclusion to premisses,

the program $\mathcal{D}$ and goal $\mathcal{G}$ are, roughly speaking, split, via C-components, into the parts $\mathcal{D}_0, \mathcal{D}'_0, \mathcal{D}_1, \ldots, \mathcal{D}_n$ and $\mathcal{G}_0, \mathcal{G}'_0, \mathcal{G}_1, \ldots, \mathcal{G}_n$, respectively. The reader might easily verify that the following is an example of the $\otimes$-rule:

$$\frac{!(p \mathbin{⅋} q), p \vdash_R p \quad !(p \mathbin{⅋} q), p \vdash_R p \quad !(p \mathbin{⅋} q), q \vdash_R q \quad !(p \mathbin{⅋} q), q \vdash_R q}{!(p \mathbin{⅋} q) \vdash_R p, p, q \otimes q}. \tag{1}$$

But the rule provides no determination of how this splitting is to be calculated. A similar problem arises with resolution rule in linear logic programming [HP92]: a clause may need to be deleted after it has been used, *i.e.*, after a resolution step has been performed. Indeed, on consideration of the resolution rule (below), the reader will see that both of these issues arise:

$$\frac{\mathcal{D}_1 \vdash_R \mathcal{G}_1 \ \ldots \ \mathcal{D}_n \vdash_R \mathcal{G}_n}{\mathcal{D} \vdash_R A, \mathcal{G}},$$

where $\bigcup_{i=1}^{n} \{\mathcal{D}_i \vdash_R \mathcal{G}_i\}$ is a *resolvant* of $\mathcal{D} \vdash_R A, \mathcal{G}$: a resolvant[3], for a particular choice of clause $G \multimap A \in \mathcal{D}$ which does not appear in $\mathcal{D}_1, \ldots, \mathcal{D}_n$, determines how to split $\mathcal{D}$ $(\mathcal{G})$ into $\mathcal{D}_1, \ldots, \mathcal{D}_n$ $(\mathcal{G}_1, \ldots, \mathcal{G}_n)$.

The definition of the resolution rule is not as complicated as the reader might at first suppose. For example, the reader might easily check that the following, in which we write c for the formula $!((p \multimap r) ⅋ ((q_1 \otimes q_2) \multimap s))$, is an instance of the resolution rule (*2*) (above):

$$\frac{c, p, p \multimap r \vdash_R r \qquad c, q_1, q_2, (q_1 \otimes q_2) \multimap s \vdash_R q_1 \otimes q_2}{p, q_1, q_2, !((p \multimap r) ⅋ ((q_1 \otimes q_2) \multimap s)) \vdash_R r, s}, \tag{2}$$

Again, the rule provides no determination of how this splitting is to be calculated.

Our solution [HP91, HP92], at the abstract level, to the problem of how to calculate splittings is to adopt a *lazy* approach, and to this end we permit an interpreter to construct *proto-proofs* by modifying the rules of resolution proof to delay the calculation of splittings (so that, for example, all suitable formulae in the antecedent and succedent go each way at a $\otimes$-rule). In order to maintain the soundness, we introduce the notion of *path* in such proto-proofs. Paths can be considered to be *proto-proof-objects* — they are the appropriate notion of proof-object for our computational purposes and are related to *proof-nets* [Gir87].

We sketch the definition of the construction of a *path*[4] in such a proto-proof as follows: (I) The endsequent is in every path; (II) Traverse the proto-proof tree towards the leaves, starting at the endsequent: (i) Whenever a &-rule is reached, choose a branch and proceed; (ii) Whenever a $\otimes$- or resolution rule is reached, proceed along all branches; (III) Continue until all branches of the path have reached a leaf. The proto-proof determines a proof just in case for each possible path in it, there are expansions (at the appropriate rule applications) of the antecedent and succedent that are compatible with leaves in the path.

We illustrate the notion of path with an example of a proto-proof involving both the $\otimes$- and resolution rules. (As above, we denote the formula $!((p \multimap r) ⅋ ((q_1 \otimes q_2) \multimap s))$ as c where convenient.)

[3] In a simple propositional setting.

[4] Such a construction proceeds dynamically, during the construction of a proto-proof.

$$\dfrac{\dfrac{c, p, q_1, q_2 \vdash^{\sharp}_R p}{c, p, q_1, q_2, p \multimap r \vdash^{\sharp}_R r}\ \text{res.} \qquad \dfrac{c, p, q_1, q_2, (q_1 \otimes q_2) \multimap s \vdash^{\sharp}_R q_1 \quad c, p, q_1, q_2 \vdash^{\sharp}_R q_2}{c, p, q_1, q_2, (q_1 \otimes q_2) \multimap s \vdash^{\sharp}_R q_1 \otimes q_2}\ \otimes}{p, q_1, q_2, !((p \multimap r) \Yen ((q_1 \otimes q_2) \multimap s)) \vdash^{\sharp}_R r, s}\ \text{res.}$$

There is just one path (marked by $\sharp$) in this proto-proof and the reader might easily verify that suitable splittings and expansions[5] exist for a resolution proof to be determined, *i.e.*, that there is indeed a resolution proof of this endsequent. To see this, consider that the leaves on the path require exactly one occurrence of each of q_1, q_2 and p; that the implicational subformulae of c are each used exactly once; and that these conditions are compatible with the proto-proof. It should be noted that the input/output model of [HM91] can be considered to be a particular computational method of determining paths.

The full theory of paths, which has much in common with the work of Andrews, Bibel and Wallen on the use of matrices and matings for proof-search [And81, Bib82, Wal89], remains to be developed in detail. Such a theory provides the transition from the logical characterization of goal-directed proof-search given by resolution proofs to a truly operational account, whilst making no inessential commitment to a particular implementation strategy.

We remark that it will be important and interesting to study broader classes of relevance logics.

References

[And81] Andrews, P.B. *Theorem-proving via general matings.* J. Assoc. Comp. Mach. 28(2):193-214, 1981.

[Bib82] Bibel, W. *Computationally Improved Versions of Herbrand's Theorem.* Logic Colloquium '81, pp. 11-28. North-Holland, 1982.

[Gir87] Girard, J.-Y. *Linear logic.* Theoretical Computer Science, 50:1–102, 1987.

[HP91] Harland, J., Pym, D. *The Uniform Proof-theoretic Foundation of Linear Logic Programming (Extended Abstract).* Proceedings of the International Logic Programming Symposium (V. Saraswat and K. Ueda, editors), San Diego, October, 1991. MIT Press, 1991. pp. 304-318.

[HP92] Harland, J.A., Pym, D.J. *Resolution in Fragments of Classical Linear Logic (Extended Abstract).* Proceedings of the Russian Conference on Logic Programming and Automated Reasoning (A. Voronkov, editor), St. Petersburg, July, 1992. Lecture Notes in Artificial Intelligence, Vol. 624, Springer-Verlag, 1992. pp. 30-41.

[HM91] Hodas, J., Miller, D. *Logic Programming in a Fragment of Intuitionistic Linear Logic: Extended Abstract.* Proceedings of the 6th Annual IEEE Symposium on

[5] Multiplicities of formulae, such as c, with ! as their outermost connective.

Logic in Computer Science, Amsterdam, July, 1991. IEEE Computer Society Press, 1991. pp. 32-42.

[Kle68] Kleene, S.C. *Mathematical Logic.* Wiley and Sons, 1968.

[Kle52] Kleene, S.C. *Permutability of inferences in Gentzen's calculi LK and LJ.* Memoirs of the American Mathematical Society **10**, pp. 1-26, 1952.

[Mil89] Miller, D. *A Logical Analysis of Modules in Logic Programming.* J. Logic Prog. 6(1& 2), 1989, pp. 79-108.

[MNPS91] Miller, D., Nadathur, G., Pfenning, F., Ščedrov, A. *Uniform Proofs as a Foundation for Logic Programming.* Annals of Pure and Applied Logic 51, pp. 125-157, 1991.

[Wal89] Wallen, L.A. *Automated Deduction in Non-classical Logics.* MIT Press, 1989.

Rules of definitional reflection in logic programming

Peter Schroeder-Heister
Universität Tübingen, Wilhelm-Schickard-Institut
Sand 13, 7400 Tübingen, Germany
e-mail: schroeder-heister@mailserv.zdv.uni-tuebingen.de

Given a set $\mathbf{D}$ of clauses of the form

$$F \Rightarrow A,$$

where F is a formula of some logic and A is an atom, it is natural to extend the sequent calculus for that logic by a rule like

$$\frac{\Gamma \vdash F}{\Gamma \vdash A}\ (\vdash \mathbf{D}),$$

yielding a logic over $\mathbf{D}$. This idea has been used in proof-theoretic interpretations and extensions of definite Horn clause programming, notably λ-Prolog, by giving a computational reading to $(\vdash \mathbf{D})$, which corresponds to resolution if the clauses in $\mathbf{D}$ are of a particular form.

In systems like GCLA, a principle dual to $(\vdash \mathbf{D})$ is considered in addition, yielding a fully symmetric sequent calculus. It is called "definitional reflection" since it is based on reading the database $\mathbf{D}$ as a definition. There are two main options for formulating definitional reflection. The rule on which GCLA is based is the following:

$$\frac{\{\Gamma, F\sigma \vdash G : F \Rightarrow B \in \mathbf{D} \textit{ and } A = B\sigma\}}{\Gamma, A \vdash G}\ (\mathbf{D} \vdash).$$

An alternative rule which has been considered by Eriksson and which seems also to be the one Girard is favoring, has the following form:

$$\frac{\{\Gamma\sigma, F\sigma \vdash G\sigma : F \Rightarrow B \in \mathbf{D} \textit{ and } \sigma = mgu(A, B)\}}{\Gamma, A \vdash G}\ (\mathbf{D} \vdash)^*.$$

As they stand, $(\mathbf{D} \vdash)^*$ is stronger than $(\mathbf{D} \vdash)$ (in the non-propositional case) - a standard example being the derivations of the axioms of ordinary first-order equality theory. Computationally, however, they rest on different intuitions. The first rule considers free variables as *existentially* quantified from outside, for which an appropriate substitution has to be computed. The second rule considers them as *universally* quantified from outside rather than something for which an substitution has still to be found. By means of unification it takes into account all possible substitution instances of the atom A, which can be inferred according to the given definition $\mathbf{D}$, thus corresponding to some kind of ω-rule.

Therefore, the extension of logic programming systems by computational variants of $(\mathbf{D} \vdash)$ and $(\mathbf{D} \vdash)^*$ leads to conceptually different approaches. A combination of $(\mathbf{D} \vdash)$ and $(\mathbf{D} \vdash)^*$ with both existential and universal variables, as proposed by Eriksson, would be a most desirable feature of a logic programming system with definitional reflection. There are certain algorithmic problems involved in such a combination that have still to be solved.

In any case, whether one considers $(\mathbf{D} \vdash)$ or $(\mathbf{D} \vdash)^*$ or a combination of both, cut-elimination fails for the full system but holds if the definition $\mathbf{D}$ does not contain implications in clause bodies or if the underlying logic is contraction-free (e.g., linear). We argue that the failure of cut-elimination is a matter of the definition $\mathbf{D}$ considered rather than a defect of the underlying logic.

A Relevance Logic Characterization of Static Discontinuity Grammars[6] (Extended Abstract)

James Andrews, Verónica Dahl, Fred Popowich
School of Computing Science, Simon Fraser University
Burnaby, BC, Canada V5A 1S6

The "logic grammar" framework [PW80, Per81, DA89] has proven to be an effective tool in computational linguistics [Joh87, Sta88, Dah89, DP90]. Rather than offering specific linguistic theories themselves, logic grammars offer clear and declarative ways to implement linguistic theories.

Studies of the foundations of logic grammars have suggested that they have a relationship to relevance logic, the brand of logic which refuses to recognize an implication as valid unless the assumptions are relevant to the conclusion. Similar connections have been established to linear logic [Gir87], which with relevance logic falls under the banner of so-called "substructural logics".

In this talk and its associated paper ([ADP91], submitted to the Journal of Logic Programming), we will describe how a specific logic grammar formalism, Dahl's Static Discontinuity Grammars, can be given a precise characterisation by a logic which is very similar in spirit to traditional relevance logics [AB75], but which also incorporates elements of linear logic.

1 "Parsing as Deduction" and its Difficulties

The parsing-as-deduction approach to syntax views the problem of parsing a sentence as the problem of deducing the assertion "we have heard (or read, or input) a sentence" from assertions of the form "we have heard (read, input) a certain word." We encode such premisses and the conclusion in some formal logic, following whatever linguistic principles we have in mind; then we give axioms and/or rules of inference which allow all and only our desired sentences to be parsed.

For instance, one possible way to encode the parsing of the sentence "Evelyn loves books" might be

$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(books)} \vdash \text{heard(s)}$$

[6]Research supported by operating grants OGP0002436 and OGP0041910 from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the BC Advanced Systems Institute. We are grateful to the Logic and Functional Programming Group, in whose lab this work was developed, and to the Centre for Systems Science, Laboratory for Computer and Communications Research and School of Computing Science at Simon Fraser University for the use of their facilities.

That is, "if we have heard the three words 'evelyn', 'loves', and 'books', we have heard a sentence." The logical system associated with this encoding would give axioms and rules which allow us to parse the sentence, by asserting that if we have heard certain tokens we have heard corresponding parts of speech, and so on.

However, if we interpret $\vdash$ as the classical consequence relation, we have problems; if the above statement is provable, so should be

$$\text{heard(books)}, \text{heard(loves)}, \text{heard(evelyn)} \vdash \text{heard(s)}$$
$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(loves)}, \text{heard(books)} \vdash \text{heard(s)}$$
$$\text{heard(evelyn)}, \text{heard(loves)}, \text{heard(very)}, \text{heard(books)} \vdash \text{heard(s)}$$

because the *set* of assumptions which allowed us to conclude that we had heard a sentence originally is still a subset of the set of assumptions in all three cases.

We could solve this problem by putting some temporal information explicitly into the logical encoding, by doing such things as labelling the words with the "times" at which they were heard, or (as in the encoding of DCGs into Horn clauses [PW80]) building such information into data structures. However, this solution may burden the axioms, rules of inference, and grammar rules with clumsy notational trivia. It would be more in the spirit of the parsing-as-deduction approach to encode this information at a basic level in the logic.

2 Relevance Logic

Relevance logic [AB75, Rea88] seems to be useful in solving the problems associated with parsing as deduction, in part because it can express the needed occurrence and ordering information in a way that is less notationally burdened than the ways given above.

Relevance logic arises from the view that one should be able to deduce a conclusion from assumptions only if the assumptions are relevant to the conclusion and made in a relevant order. Relevance logic therefore treats a list of assumptions as a sequence rather than a set, and rejects the "structural rules" of permutation (which says that the order of assumptions is irrelevant), contraction (which says that we can duplicate any assumption), and weakening (which says that if we add assumptions we can still prove the same things).

"Substructural logics" in general (those logics which reject some structural rules) seem to be connected to parsing as deduction. Evidence of this is given by work to do with linear logic [Gir87], another substructural logic. The sequent calculus version of the Lambek calculus for categorial grammars [Lam88] has been proven equivalent to a fragment of intuitionistic linear logic by Abrusci [Abr90] and independently by de Paiva [DP91]; and Hodas and Miller [HM91] have used linear logic to extend Definite Clause Grammars via a linear logic programming language.

3 The Approach of This Paper

In this paper, we make explicit the connection between substructural logic and logic grammars by a concrete example.

The logic grammar framework which we study is Dahl's Static Discontinuity Grammars (SDGs) [DP90], which grew out of Definite Clause Grammars [PW80] and Extraposition Grammars [Per81] in a desire to handle discontinuous constituents and "movement" at a more fundamental level. We show that both of the interpretations of SDGs can be characterized in a sound and complete manner by a relevance-logic sequent calculus. Since DCGs, Scattered Context Grammars [GH69], and so on are encompassed by the SDG formalism, this characterization also applies to them.

3.1 Static Discontinuity Grammars

A static discontinuity grammar (SDG) is a collection of *rules*, each of which is a tuple of *clauses*. A clause is an expression of the form $(\mathbf{t} \rightarrow \mathbf{A}_1, \ldots, \mathbf{A}_m)$, for $m \geq 1$, where (essentially) $\mathbf{t}$ is a non-terminal, and each of the $\mathbf{A}_i$'s is a terminal, non-terminal, or empty sequence marker ("[]").

An SDG in which each rule contains only one clause is interpreted exactly as if it were a DCG. Rules containing more than one clause are interpreted as stating that whenever an instance of the rule is used, each clause instance must be applied exactly once, in a left-to-right manner across the parse tree.

There are two interpretations of SDGs, one broader than the other. They differ only in their restrictions on the positions in the parse tree at which clauses of a multi-clause rule can be applied. In the "rewriting" interpretation, the "cut" in the parse tree corresponding to a rule application (informally, a line drawn across the parse tree going through each head node involved) must not cross any other rule's cut; in the "tree admissibility" interpretation, crossing cuts are allowed. Each interpretation has some advantages and disadvantages. Both interpretations are characterized by the logic given in the paper.

3.2 The Sequent Calculus Characterization

The sequent calculus which characterizes SDGs makes use of several features of relevance and/or linear logic:

1. The "splitting" conjunction $\otimes$, found in both relevance and linear logic;
2. The relevance-logic "bunch" construct $\langle \mathbf{A}_1, \ldots, \mathbf{A}_n \rangle$, which groups together formulae to preserve their order within a sequent; and
3. The unary linear-logic "of course" connective "!", which represents 0 or more copies of a formula.

However, the sequent calculus interprets bunches in a manner which is connected to, but not the same as, the standard interpretation in relevance logic.

The first step in the characterization of SDGs is to give a translation of an SDG and a parsing problem into formulae. A clause $(\mathbf{t} \rightarrow \mathbf{A}_1, \ldots, \mathbf{A}_m)$ is translated into the formula $(\mathbf{t} \leftarrow \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_m)$. A rule $\langle C_1, \ldots, C_n \rangle$ is translated into the formula $!(t(C_1) \otimes \cdots \otimes t(C_n))$,

Cont/l: $\dfrac{!\Gamma, !\mathbf{R}, !\Gamma', \Delta, \langle \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, !\mathbf{R}, !\Gamma', \Delta, \Delta' \vdash \mathbf{B}}$

("copy SDG rule")

$\forall$/l: $\dfrac{!\Gamma, \Delta, \langle \mathbf{R}[\mathbf{x} := \mathbf{t}] \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \forall \mathbf{x} \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}$

("choose substitutions")

$\otimes$/l: $\dfrac{!\Gamma, \Delta, \langle \Lambda, \mathbf{C}, \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \Lambda, \mathbf{C} \otimes \mathbf{R} \rangle, \Delta' \vdash \mathbf{B}}$

("separate rule clauses")

$\otimes$/r: $\dfrac{!\Gamma, \langle \Lambda_1 \rangle, \ldots, \langle \Lambda_k \rangle \vdash \mathbf{B} \quad !\Gamma, \langle \Lambda'_1 \rangle, \ldots, \langle \Lambda'_k \rangle \vdash \mathbf{B}'}{!\Gamma, \langle \Lambda_1, \Lambda'_1 \rangle, \ldots, \langle \Lambda_k, \Lambda'_k \rangle \vdash \mathbf{B} \otimes \mathbf{B}'}$

("allocate clauses across tree")

$\leftarrow$/l: $\dfrac{!\Gamma, \Delta, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle \mathbf{t} \leftarrow \mathbf{B} \rangle, \Delta' \vdash \mathbf{t}}$

("call clause")

Refl: $\dfrac{}{!\Gamma, \langle \mathbf{t} \rangle \vdash [\mathbf{t}]}$

("match terminal")

$\langle\rangle$/l: $\dfrac{!\Gamma, \Delta, \Delta' \vdash \mathbf{B}}{!\Gamma, \Delta, \langle\rangle, \Delta' \vdash \mathbf{B}}$

("ignore empty rule")

[ ]/r: $\dfrac{}{!\Gamma \vdash [\,]}$

("match empty sequence")

Figure 1: Rules for sequent calculus Rsa.

where $t(C)$ is the translation of clause C. Finally, the problem of parsing a sequence of terminals $\mathbf{t}_1, \ldots, \mathbf{t}_k$ as a non-terminal $\mathbf{s}$ in a grammar G is translated into the sequent

$$t(G), \langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle \vdash \mathbf{s}$$

where $t(G)$ is the translation of the rules of G.

With this translation given, we have only to give the rules for the sequent calculus, called Rsa (Figure 1). We have the result that a sentence is accepted by an SDG under the "admissibility" interpretation iff the translation of the parsing problem is a theorem of Rsa. Furthermore, there is a variant of Rsa, called Rsr, which characterizes the "rewriting" interpretation in the same way: it is the restriction of Rsa in which the indicated bunch of the $\leftarrow$/l rule must be the *leftmost* one.

The formal definitions, theorems, and proofs are given in [ADP91].

References

[AB75] Alan R. Anderson and Nuel D. Belnap. *Entailment: The Logic of Relevance and*

Necessity. Princeton University Press, Princeton, N.J., 1975.

[Abr90] M. Abrusci. A comparison between Lambek syntactic calculi and intuitionistic linear logic. *Zeitschrift für Mathematische Logik*, 36:11–15, 1990.

[ADP91] James Andrews, Veronica Dahl, and Fred Popowich. A relevance logic characterization of static discontinuity grammars. Technical Report CSS/LCCR TR91-12, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, December 1991. Expanded version submitted to the Journal of Logic Programming.

[DA89] V. Dahl and H. Abramson. *Logic Grammars.* Monograph, Symbolic Computation AI Series. Springer-Verlag, 1989.

[Dah89] Veronica Dahl. Discontinuous grammars. *Computational Intelligence*, 5:161–179, 1989.

[DP90] Veronica Dahl and Fred Popowich. Parsing and generation with static discontinuity grammars. *New Generation Computing*, 8:245–274, 1990.

[DP91] Valeria De Paiva. A Dialectica model of the Lambek calculus. In *8th Amsterdam Logic Colloquium*, Amsterdam, December 1991.

[GH69] S. Greibach and J. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3:233–247, 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[HM91] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. In *Logic in Computer Science*, Amsterdam, July 1991.

[Joh87] M. Johnson. The use of knowledge of language. Technical report, Brain and Cognitive Sciences, M.I.T., 1987.

[Lam88] J. Lambek. Categorial and categorical grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, volume 32 of *Studies in Linguistics and Philosophy*, Dordrecht, 1988. D. Reidel.

[Per81] Fernando Pereira. Extraposition grammars. *American Journal for Computational Linguistics*, 7:243–256, 1981.

[PW80] Fernando Pereira and David H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[Rea88] Stephen Read. *Relevant Logic.* Blackwell, Oxford, 1988.

[Sta88] E. P. Stabler, Jr. The logical approach to syntax: Foundations, specifications and implementations of theories of government and binding. Technical report, Department of Linguistics, UCLA, November 1988.

Disjunction in Linear Deductive Planning[7]

(Extended Abstract)

S. Brüning, G. Große, S. Hölldobler, J. Schneeberger, U. Sigmund, M. Thielscher
Intellektik, Informatik, TH Darmstadt, Alexanderstraße 10, D–6100 Darmstadt (Germany)

It has always been a major research goal in Intellectics[8] to model in a formal system the remarkable ability of human agents to reason about situations and actions [7]. But despite all efforts the goal is far from being reached. Recently three new deductive approaches were proposed [1, 5, 6], where the facts describing situations are treated as resources. In fact, a situation is a multiset of facts. The facts are consumed when the conditions of action are to be satisfied and are produced as the effects of an action. None of these approaches requires to state frame axioms explicitely and even though they are based on very different paradigms, they are essentially equivalent [4].

In more realistic examples situations are not just multisets of facts but there are usually alternatives and, even worse, an agent may not even be aware of all alternatives. In this paper we consider alternatives but restrict ourselves to the case where the agent has complete knowledge about the various possibilities. We also assume that the agent is quite cautious in the sense that she is only interested in plans — ie. sequences of actions — which solve her goals no matter which one of the alternatives holds in the real world. Technically, this amounts to an extension of the previously mentioned approaches. In this paper we extend two approaches — the equational logic approach of [5] and the linear connection method of [1] — and allow situations to be described by formulae containing (non-idempotent) conjunctions and (idempotent) disjunctions. The conjunctions model the multisets of resources as before whereas the disjunctions model the alternatives. We give a semantics for these extensions and show that they are equivalent. Thereby it will turn out that Bibel's linearity constraint — ie. the requirement that in a connection proof of a planning problem each literal is connected at most once — is insufficient for handling disjunction. We introduce a new linearity constraint and demonstrate that this new constraint guarantees the proper treatment of disjunction.

Disjunctive Planning Problems

The use of disjunction in planning is best explained by formalizing a short story which we want to call the tragic king's problem. *Once upon a time the king of an old country suffered from the kidnapping of his only son by the wizzard. There was no hope until a young and brave lady crossed the country. As she heard about the king's suffering and saw a beautiful*

[7]This work was funded in part by the (German) Ministry for Research and Technology (BMFT) within project TASSO under grant no. ITW 8900 C2, by ESPRIT within basic research action MEDLAR-II under grant no. 6471, by the Deutsche Forschungsgemeinschaft (DFG) within projects MUSE under grant no. HE 1170/5-1 and KONNEKTIONSBEWEISER under grant no. Bi 228/6-1.

[8]Ie. Artificial Intelligence and Cognition [3].

painting of his lovely son she knew what to do. There were two ways to release him. The first one was to fight an enormous dragon. The second one was to kill the wizzard. For both tasks she needed the stiletto she got from the mother of her grandmother. But to kill the dragon she needed also a lance and to kill the wizzard she needed also a magic wand. Fortunately, the lady had a blue fairy, who has always fulfilled her wishes, and she asked her for help. Did the king see his lovely son ever again?

In our formal treatment we use the strings s, w, d, dw, dd, l, m to represent the stiletto, the wizzard, the dragon, the dead wizzard, the dead dragon, the lance, and the magic wand, respectively. A situation is described by a multiset of facts, eg. the multiset $\{s, w, d\}$ [9] models a situation in which the stiletto, the wizzard, and the dragon are present. A *disjunctive planning problem* consists of an initial set of situations $\mathcal{I}$ — here $\{\{s, w, d\}\}$, a set of actions $\mathcal{A}$ — here

$$\begin{array}{lll} bf: & \{\} \mapsto \{\{l\}, \{m\}\} & \text{for asking the blue fairy,} \\ kd: & \{s, l, d\} \mapsto \{\{dd\}\} & \text{for killing the dragon,} \\ kw: & \{s, m, w\} \mapsto \{\{dw\}\} & \text{for killing the wizzard,} \end{array}$$

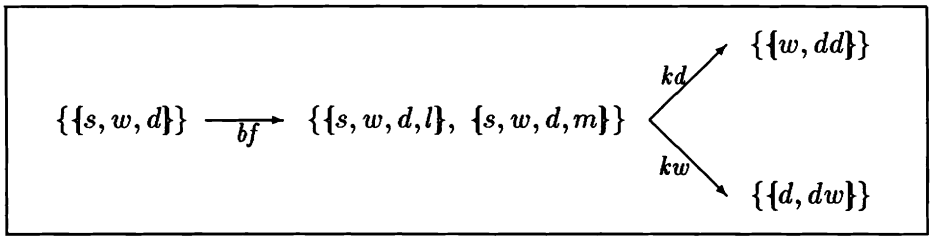
and a set of goal situations $\mathcal{G}$ — here $\{\{dw\}, \{dd\}\}$. One should observe that the goal as well as the effects of the first action contain alternatives. Receiving either a lance or a magic wand is expressed by a set consisting of two multisets. Throughout the paper we assume the condition of the actions to consist of only one situation. In this case, a rule containing alternatives as condition can easily be expanded to an equivalent set of actions where the condition of each rule corresponds to one alternative.

An action $A\colon\ C \mapsto \{E_1, \ldots, E_n\}$ is *applicable* to a set of situation $\mathcal{S} = \{S_1, \ldots, S_m\}$ iff $C \dot{\subseteq} S_i$ for all $S_i \in \mathcal{S}$. If A is applicable to $\mathcal{S}$ then the application of A to $\mathcal{S}$ yields the set of situations $\{(S_i \dot{-} C) \dot{\cup} E_j \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$. A *plan* is a tree whose nodes represent sets of situations and whose edges represent actions. A plan p is called *permitted* with respect to a set $\mathcal{A}$ of actions iff for all inner nodes $\mathcal{S}$ of p the following condition holds. If $\mathcal{S}$ has k successors $\mathcal{S}'_1, \ldots, \mathcal{S}'_k$, $k \geq 1$, then there exist disjoint sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that $\mathcal{S} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_k$ and for all $i = 1, \ldots, k$, if the edge from $\mathcal{S}$ to $\mathcal{S}'_i$ is labelled with action $A_i \in \mathcal{A}$ then A_i is applicable to $\mathcal{S}_i$ and yields $\mathcal{S}'_i$. A plan p solves a given disjunctive planning problem $\langle \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ iff p is permitted with respect to the set $\mathcal{A}$ of actions, the root of p represents the set $\mathcal{I}$ of initial situations, and each element of each set of situations represented by a leaf of p contains one element of the set $\mathcal{G}$ of goal situations.

The following figure shows a plan which solves the tragic king's problem. One should observe, that after asking the blue fairy the lady must observe the answer before she can decide on which action to take next. This can be expressed by introducing a conditional in a linearized version of the plan. For the tragic king's problem we obtain the linearized plan $[bf, cond(l, kd, m, kw)]$, which should be read as ask the blue fairy first, and then, if

[9] Multisets are denoted by the brackets $\{$ and $\}$ whereas sets are denoted as usual by $\{$ and $\}$. The operations $\dot{\subseteq}$, $\dot{\cup}$ and $\dot{-}$ denote the multiset extensions of the usual operations $\subseteq$, $\cup$, and $-$ defined on sets.

she hands you a lance, kill the dragon, otherwise, if she hands you a magic wand, kill the wizzard.



Disjunction in Equational Logic Programming

In [5], situations are represented as terms built from the facts and the constant $\emptyset$ using the binary function symbol $\circ$. For example, the empty situation is denoted by $\emptyset$ and the initial situation in the tragic king's problem can be represented by the term $s \circ w \circ d$. Such a term models a multiset if $\circ$ is commutative, associative, and has $\emptyset$ as a unit element. In other words, $\circ$ represents an AC1-function. Planning problems are specified using a predicate $plan(s,p,t)$ which is interpreted declaratively as *the execution of plan* p *transforms situation* s *into situation* t. Actions are defined by rules of the form

$$plan(conditions \circ V,\ [action, P],\ W) :- plan(effects \circ V,\ P,\ W).$$

Such an action is applicable if the term $conditions \circ V$ unifies with the term representing the current situation under the equational theory for $\circ$. The planning process terminates if the goal situation is contained in the current situation, ie.

$$plan(X \circ Y, [], X). \tag{3}$$

Extending this approach to handle disjunctive planning problems, we first have to represent sets of alternative situations. This is done with the help of a binary function symbol $|$, which is associative, commutative, idempotent, and admits a unit element $\perp$. In other words, $|$ represents an ACI1-function whose intended meaning is to denote alternatives. For instance, the term $l \mid m$ denotes the set $\{\{l\}, \{m\}\}$ obtained when receiving a gift from the fairy, whereas the term $l \mid l$ denotes the set $\{\{l\}\}$, which contains the only alternative if the fairy only hase two lances to choose from. If we regard facts as resources and a term of the form $X \mid Y$ as having either the resources contained in X or the resources contained in Y but not both, then it is natural to require that $\circ$ distributes over $|$, ie. that the law of distibutivity (D)

$$\forall X, Y, Z:\ X \circ (Y \mid Z) \ = \ (X \circ Y) \mid (X \circ Z)$$

holds. Using this equation every term can be transformed into disjunctive normal form. One should observe that $\mid$ does not distribute over $\circ$ as this contradicts the intended interpretation.

The rules to express disjunctive planning problems have the same form as the rules in [5] except that now the function symbol $\mid$ may also be used to specify situations. Hence, the rules

$$plan(V_1, [bf, P_1], W_1) \ :- \ plan((l \mid m) \circ V_1, P_1, W_1), \tag{4}$$

$$plan(s \circ l \circ d \circ V_3, [kd, P_3], W_3) \ :- \ plan(dd \circ V_3, P_3, W_3), \tag{5}$$

$$plan(s \circ m \circ w \circ V_4, [kw, P_4], W_4) \ :- \ plan(dw \circ V_4, P_4, W_4). \tag{6}$$

encode the actions in the tragic king's problem. The planning process is triggered by formulating the planning problem as a query to the logic program.

$$?- \ plan(s \circ w \circ d, P, dw \mid dd). \tag{7}$$

As in [5], queries are answered using SLDE-resolution, where the union of the equational theory AC1 of the operator $\circ$, the theory ACI1 of the operator $\mid$, and the law of distributivity (D) are built into a special unification procedure. Resolving the goal clause (7) with rule (4) yields the binding $\{P \mapsto [bf, P_1]\}$ for P and the resolvent

$$?- plan((s \circ w \circ d \circ l) \mid (s \circ w \circ d \circ m), P_1, dw \mid dd). \tag{8}$$

As our lady may have received either the lance or the magic wand, there are now two alternative situations. Since we intend to model cautious agents, which are looking for plans such that their goals are achieved no matter which situation they are in, we have to solve the problem in both alternatives. This can be done by splitting goal (8) using the following rule.

$$\begin{aligned} plan((X_2 \mid X_2') \circ V_2, cond(X_2, P_2, X_2', P_2'), W_2) \ :- \ & plan(X_2 \circ V_2, P_2, W_2), \\ & plan(X_2' \circ V_2, P_2', W_2). \end{aligned} \tag{9}$$

The term $cond(X_2, P_2, X_2', P_2')$ is a conditional and should be read as *if the agent observes* X_2 *then plan* P_2 *solves the problem, otherwise, if the agent observes* X_2' *then plan* P_2' *solves the problem.* Resolving (8) and (9) yields the binding $\{P_1 \mapsto cond(l, P_2, m, P_2')\}$ for P_1 and the resolvent

$$?- plan(s \circ w \circ d \circ l, P_2, dw \mid dd), \ plan(s \circ w \circ d \circ m, P_2', dw \mid dd). \tag{10}$$

The subgoals of this query can be resolved with the rules (5) and (6), respectively, leading to the bindings $\{P_2 \mapsto [kd, P_3], \ P_2' \mapsto [kw, P_4]\}$ for P_2 and P_2' and the resolvent

$$?- plan(w \circ dd, P_3, dw \mid dd), \ plan(d \circ dw, \ P_4, dw \mid dd). \tag{11}$$

The tragic king's problem is almost solved. The agent is in a situation where either the dragon or the wizzard is dead. However, the fact (3) cannot immediately be used to terminate the refutation because the goal situation contains alternatives. However, if we use the fact $plan(X \circ Y, [], X \mid X')$ instead, then P_3 and P_4 are bound to the empty plan $[]$. Composing the substitutions obtained in the refutation of (7), we obtain the computed answer substitution $\{P \mapsto [bf, cond(l, [kd, []], m, [kw, []])]\}$ which is precisely the desired answer.

Disjunction in the Linear Connection Method

We know from [8] or [2] that a proof for a first order formula consists of a set of connections such that all connected literals are simultaneously unifiable. In [1] it was shown that such proofs solve planning problems if each literal is engaged in at most one connection. Although in all examples given in [1] the conditions and effects of actions were conjunctions of atoms, the so-called linear connection method is not restricted to such conjunctions.

In this section we want to illustrate with the help of the tragic king's problem how Bibel's linear connection method handles disjunctions. The initial set $\mathcal{I}$ of situations is encoded as the formula

$$I \equiv \exists P : s \wedge w \wedge d \wedge st(P),$$

where $st(P)$ is the so-called state literal, whose only purpose is to record the actions taken in order to achieve a goal. The set $\mathcal{G}$ of goal situations is encoded as

$$G \equiv [dw \vee dd] \wedge st([]),$$

where $[]$ is a constant denoting the empty plan. The set $\mathcal{A}$ of actions is represented by $A \equiv BF \wedge KD \wedge KW$, where

$$\begin{array}{lcl} BF & \equiv & \forall P_1, P_2 : st([t, cond(n, P_1, l, P_2)]) \Rightarrow [l \wedge st(P_1)] \vee [m \wedge st(P_2)], \\ KW & \equiv & \forall Y : st([kw, Y]) \wedge s \wedge w \wedge m \Rightarrow dw \wedge st(Y), \text{ and} \\ KD & \equiv & \forall Z : st([kd, Z]) \wedge s \wedge d \wedge l \Rightarrow dd \wedge st(Z). \end{array}$$

A linear connection proof of the formula $I \wedge A \Rightarrow G$ should yield the desired plan. The following figure shows a connection proof. This proof generates this plan as the binding $\{P \mapsto [bf, cond(l, [kw, []], m, [kd, []])]\}$ for P.
Unfortunately, the proof is not linear. The literals $\neg s$ and $st([])$ are connected twice. The literal $st([])$ can be duplicated with the help of a distributivity law similar to (D), whereas the literal $\neg s$ cannot be duplicated as this would contradict the concept of resources. However, the proof shows an interesting feature. For every alternative plan — viz. the plans $[bf, cond(l, [kw, []])]$ and $[bf, cond(m, [kd, []])]$ — a linear connection proof can be found if we split the alternatives in the set of goal situations as well as in the rule BF into two submatrices as shown in the following figure, and proof each goal situation separately. In this figure the proof of $[bf, cond(l, [kw, []])]$ is shown in the usual way, whereas the connections of the proof of $[bf, cond(m, [kd, []])]$ are drawn with dotted lines.

This observations suggests a modified linearity constraint under which disjunctive planning problems can be solved by the linear connection method. A connection proof of a matrix M is said to be *globally linear* iff each subproof of M is linear, where the subproofs are obtained by splitting the alternatives in the effects of actions and in the initial situation and proofing alternative goal situations separately.

Results

In the talk we discuss the soundness and completeness of the equational logic approach as well as its equivalence to the modified linear connection method. Whether it is also equivalent to a corresponding extension of the linear logic approach in [6] remains to be seen.

References

[1] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[2] W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, second edition, 1987.

[3] W. Bibel. Intellectics. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 705–706. John Wiley, New York, 1992.

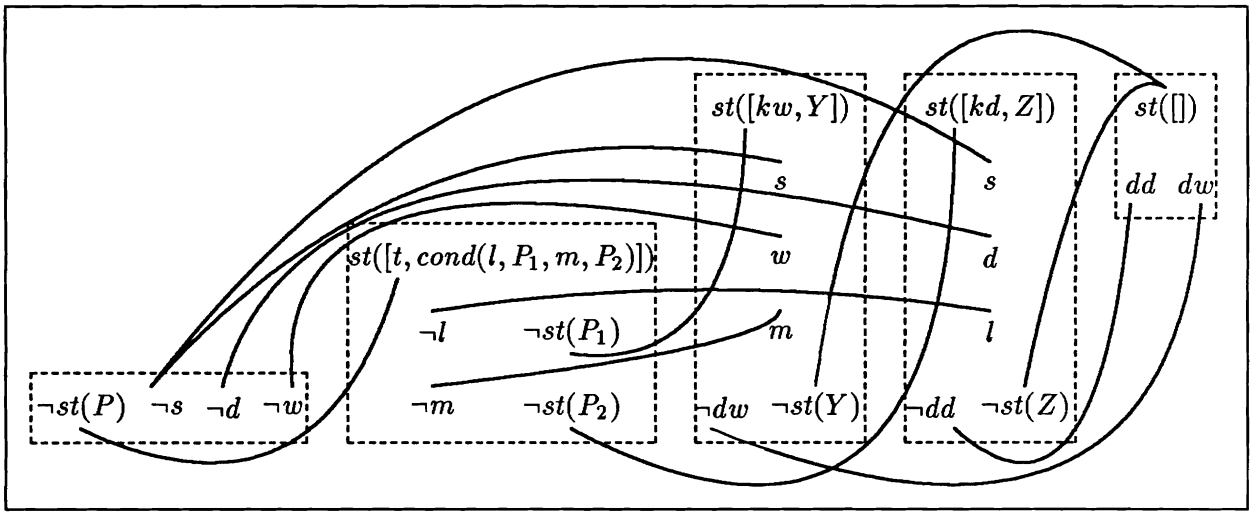
[4] G. Grosse, S. Hölldobler, and J. Schneeberger. On linear deductive planning. Internal Report, Technische Hochschule Darmstadt, Fachbereich Informatik, 1992.

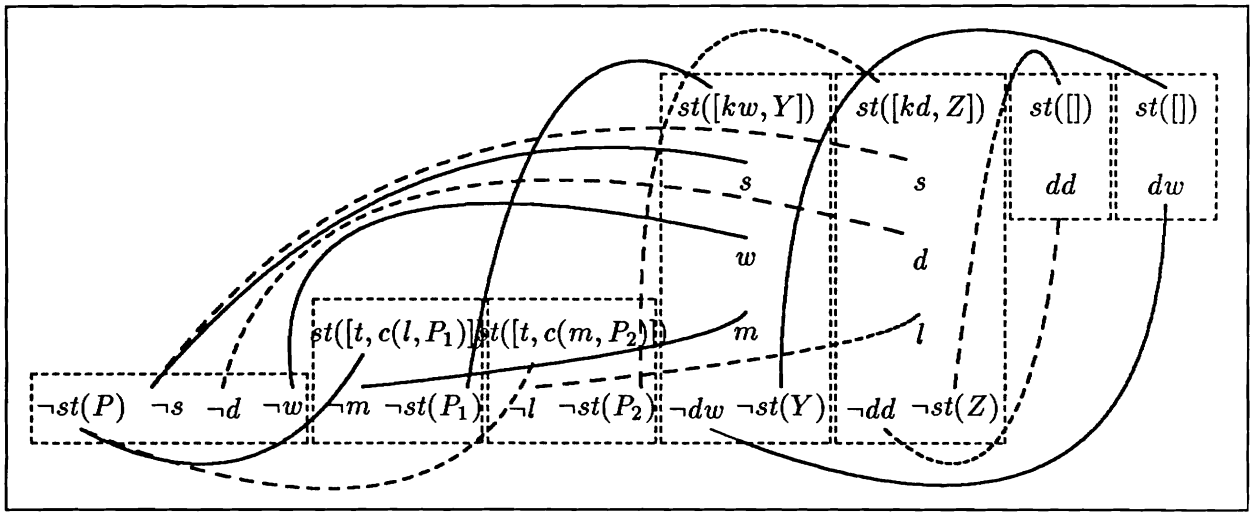
[5] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989.

[6] M. Masseron, C. Tollu, and J. Vauzielles. Generating plans in linear logic. In *Foundations of Software Technology and Theoretical Computer Science*, pages 63–75. Springer, LNCS *472*, 1990.

[7] J. McCarthy. Situations and actions and causal laws. Stanford Artificial Intelligence Project: Memo 2, 1963.

[8] M. E. Stickel. An introduction to automated deduction. In W. Bibel and P. Jorrand, editors, *Fundamentals of Artificial Intelligence*, pages 75 – 132. Springer, 1987.


st([kw, Y])
st([kd, Z])
st([])
s
s
dd
dw
w
d
st([t, cond(l, P1, m, P2)])
¬l
¬st(P1)
m
l
¬st(P)
¬s
¬d
¬w
¬m
¬st(P2)
¬dw
¬st(Y)
¬dd
¬st(Z)


st([kw, Y])
st([kd, Z])
st([])
st([])
s
s
dd
dw
w
d
st([t, c(l, P1)])
st([t, c(m, P2)])
m
l
¬st(P)
¬s
¬d
¬w
¬m
¬st(P1)
¬l
¬st(P2)
¬dw
¬st(Y)
¬dd
¬st(Z)

An Observational Semantics of Linear Logic

Yasushi Fujiwara
Department of Computer Science
Stanford University[10]
fujiwara@cs.stanford.edu

Abstract

We present a simple semantics of *noncommutative* linear logic based on the observation of labeled transition systems, which is familiar notion in the study of concurrency. In this interpretation a little similar to Kripke semantics of modal logics, $\otimes$ will be viewed as sequential composition. We also show that many process equivalences discussed so far in concurrency theory can be characterized by using linear logic formulas, which is an analogue of modal characterization theorems. This exhibits a relationship between linear logic and process observation of calculi like CCS or CSP.

4 Transition System Semantics

Linear logic was proposed by Girard as a logic of action [Girard]. The departure of linear logic from (ordinary) classical logic is the abondonment of the structural rules. It is Girard's crucial observation that structural rules may lose the control of cut-elimination of the logic and injure the constructiveness.

From its birth, linear logic has been expected to be significant in the study of parallelism or concurrency. In this note, we will consider an aspect of linear logic as observational logic. The semantics we discuss is based on the observation of labeled transition systems, which is a common notion in the study of concurrency.

A labeled transition system over a given set A is a pair (X, δ), where

- X is a set, called the set of processes;
- $\delta : A \times X \to 2^X$ is any function, called the transition function (2^X denotes the power set of X.);

We denote by $x \xrightarrow{a} y$ the condition that $y \in \delta(a, x)$. In this note, we tacitly impose image-finiteness condition, i.e. $\delta(a, x)$ is finite for all $(a, x) \in A \times X$.

For the definition of noncommutative linear logic, the reader is referred to [BrownGurr] or [Yetter]. We *identify* A with the set of atomic propositions of linear formulas. Given an equivalence relation $\sim$ on X, we will interpret linear formulas as relations that are right and left-invariant with respect to $\sim$. (On the analogy with Bainbridge et al [BFSS], we call such relations *saturated.*) For example, an atomic proposition a will be interpreted by the

[10]On leave from Toshiba Corporation.

saturated relation $\{(x,y)|\exists x'\exists y'(x \sim x', x' \stackrel{a}{\rightarrow} y', y' \sim y)\}$, which is the smallest among saturated relations that contain $\{(x,y)|x \stackrel{a}{\rightarrow} y\}$. The satisfaction relation of noncommutative linear logic (without modalities ! and ?) is defined as follows:

$$\begin{aligned}
(x,y) \models \mathbf{1} &\equiv x \sim y \\
(x,y) \models \top &\equiv \textbf{true} \\
(x,y) \models \mathbf{0} &\equiv \textbf{false} \\
(x,y) \models \bot &\equiv x \not\sim y \\
(x,y) \models a &\equiv \exists x'\exists y'(x \sim x', x' \stackrel{a}{\rightarrow} y', y' \sim y) \\
(x,y) \models a^{\bot} &\equiv (y,x) \not\models a \\
(x,y) \models \phi \otimes \psi &\equiv \exists z((x,z) \models \phi \text{ and } (z,y) \models \psi) \\
(x,y) \models \phi \wp \psi &\equiv \forall z((x,z) \models \phi \text{ or } (z,y) \models \psi) \\
(x,y) \models \phi \& \psi &\equiv (x,y) \models \phi \text{ and } (x,y) \models \psi \\
(x,y) \models \phi \oplus \psi &\equiv (x,y) \models \phi \text{ or } (x,y) \models \psi
\end{aligned}$$

(We here use $\wp$ for "multiplicative or.") It is easy to see that all the relations that interpret linear formulas are saturated. The interpretation of additive connectives & and $\oplus$ follows Tarskian semantics, but the interpretation of multiplicative connectives $\otimes$ and $\wp$ is similar to that in Kripke semantics of modal logics. And linear negation is interpreted as "reaction."

A linear logic formula ϕ is called *valid* if $(x,x) \models \phi$ for any $x \in X$ and will be denoted by $(X,\delta,\sim) \models \phi$. It is easy to check the following claim.

Theorem 1 *Let (X,δ) be a transition system over A and $\sim$ be an equivalence relation on X. If ϕ is provable in noncommutative linear logic, then $(X,\delta,\sim) \models \phi$.*

(The logic is *not* complete with respect to this interpretation. For example, & is distributive over $\oplus$ because of the set-theoretic definiton.)

5 Linear Characterization Theorem

In the study of concurrency, many notions of process classification have been proposed. Probably, the most famous among them are bisimilarity and trace equivalence. Different process equivalences represent behavioral equivalence under different notions of processe observation.

Hennessy and Milner [HennMil] introduced Hennessy-Milner logic (HML), a kind of modal logic, and showed that bisimilarity is characterized by the satisfaction relation of HML formuals. Two processes are bisimilar if and only if the set of HML formulas they satisfy coincide. It has been shown that many other process preorders are characterized by the satisfaction relation of fragments of HML ("modal characterization theorems" [AbrVick]). Modal characterization theorems are one of central topics in concurrency theory.

As the labeled transition systems can be seen as models of linear logic, it is natural to expect that linear logic model may be useful for the understanding of such process preorders. By the construction, the following proposition is obvious.

Proposition 1 *Let (X, δ) be a labeled transition system over A and $\sim$ be an equivalence relation on X. If $x \sim y$ and $x' \sim y'$, then, for any linear formula ψ, we have*

$$(x, x') \models \psi \Leftrightarrow (y, y') \models \psi.$$

In particular, if $x \sim y$, for any linear formula ψ, we have

$$(x, x) \models \psi \Leftrightarrow (y, y) \models \psi.$$

We will exploit the possibility of the opposite implication. That is, we will consider whether the equivalence relation $\sim$ is characterized by satisfaction relation of linear formulas.

We give a brief review of the definition of HML. The syntax is given as follows:

$$\psi ::= \mathbf{t}|\mathbf{f}|\psi \wedge \psi|\psi \vee \psi|\langle a\rangle\psi|[a]\psi$$

The satisfaction relation $x \models \psi$ is defined following the line of Kripke semantics. It is sufficient to display the definiton of modalities.

$$\begin{aligned} x \models \langle a\rangle\psi &\equiv \exists y(x \xrightarrow{a} y \text{ and } y \models \psi) \\ x \models [a]\psi &\equiv \forall y(x \xrightarrow{a} y \Rightarrow y \models \psi) \end{aligned}$$

An equivalence relation $\sim$ on X is said to be characterized by a set S of HML formulas if the following holds:

$$x \sim y \Leftrightarrow \forall \psi \in S.(x \models \psi \text{ iff } y \models \psi)$$

For example, it is known that bisimilarity is characterized by the set of all HML formulas and that trace equivalence is characterized by the set of all HML formulas of the form $\langle a_1\rangle \ldots \langle a_n\rangle\mathbf{t}$.

We now give a translation of HML into noncommutative linear logic. The translation is based on the induction of the construction of HML formulas. The translation of logical connectives in HML are rather direct. We will translate HML formulas $\mathbf{t}$ and $\mathbf{f}$ into $\top$ and $\mathbf{0}$, respectively. If an HML formula ψ (resp. ϕ) is translated into $\overline{\psi}$ (resp. $\overline{\phi}$), then $\psi \wedge \phi$ (resp. $\psi \vee \phi$) will be trasnlated into $\overline{\psi}\&\overline{\phi}$ (resp. $\overline{\psi} \oplus \overline{\phi}$). The modalities of HML will be encoded by multiplicatives $\otimes$ or $\wp$. If an HML formula ψ is translated into $\overline{\psi}$, then $\langle a\rangle\psi$ (resp. $[a]\psi$) will be translated into $a \otimes (\overline{\psi}\&\mathbf{1}) \otimes \top$ (resp.$\mathbf{0}\wp(\overline{\psi} \oplus \bot)\wp a^{\bot}$.) By definition, we can check that

$$\begin{aligned} (x, x) \models a \otimes (\overline{\psi}\&\mathbf{1}) \otimes \top &\equiv \exists y((x, y) \models a \text{ and } (y, y) \models \overline{\psi}) \\ (x, x) \models \mathbf{0}\wp(\overline{\psi} \oplus \bot)\wp a^{\bot} &\equiv \forall y((x, y) \models a \Rightarrow (y, y) \models \overline{\psi}) \end{aligned}$$

Notice that $[a]$-free fragment of HML will be translated into the intuitionistic fragment of linear logic.

When the equivalence relation $\sim$ is the equality on X, we easily see that, for any HML formula ψ, $x \models \psi \Leftrightarrow (x,x) \models \overline{\psi}$. But, in general, the situation can be more complicated. The following lemma shows how the interpretation of HML formulas is related to that of translated linear formulas. Notice that we here pose a special restriction on the occurrence of $[a]$. Interestingly, the same restriction appears in the modal charcterization theorem of ready simulation preorder.

Lemma 1 *Let $\sim$ be an equivalence relation characterized by an HML fragment S in which $[a]$ occurs only in subformulas of the form $[a]\mathbf{f}$. Suppose that S satisfies the following conditions:*

- $\phi \wedge \psi \in S \Rightarrow \phi, \psi \in S$
- $\phi \vee \psi \in S \Rightarrow \phi, \psi \in S$
- $\langle a \rangle \psi \in S \Rightarrow \psi \in S$

Then, for $\psi \in S$, $x \models \psi$ if and only if $(x,x) \models \overline{\psi}$.

The proof will be done by induction on the structure of HML formulas.

As bisimilarity is characterized by the set of all HML formulas, the above lemma is not applicable to bisimilarity. But, we can establish the same conclusion in this case.

Lemma 2 *Let $\sim$ denote bisimilarity. Then we have $x \models \psi \Leftrightarrow (x,x) \models \overline{\psi}$ for any HML formula ψ.*

Now we state our main result, which is a linear logic analogue of modal characterization theorems. This theorem applies to all process preorders listed on page 15 of [AbrVick].

Theorem 2 *Let an equivalence relation $\sim$ on X be bisimilarity or satisfy the conditions listed in Lemma 1. Then,*

$$x \sim y \Leftrightarrow \forall \psi((x,x) \models \psi \text{ if and only if } (y,y) \models \psi)$$

Proof We already observed the implication $\Rightarrow$. Suppose that x and y satisfy the condition of the RHS. In the case that $\sim$ is bisimilarity, let S be the set of all HML formulas. In the case that $\sim$ satisfies the conditions in Lemma 1, let S be as in the lemma. By the assumption, for any HML formula $\psi \in S$, $(x,x) \models \overline{\psi} \Leftrightarrow (y,y) \models \overline{\psi}$. By the above lemmas, this is equivalent to $\forall \psi \in S.(x \models \psi \Leftrightarrow y \models \psi)$, which means $x \sim y$ by the definition of S. □

6 Conclusion and Related Work

The applications of linear logic to concurrency theory obtained so far are mainly concentrated in Petri-net theory (e.g. [EngWin]). In this note, we proposed a simple semantics of noncommutative linear logic and showed that the model can be related to observations in process calculi like CCS or CSP. We expect that our result may shed some light on further relationship between linear logic and concurrency theory.

Technically speaking, the model we discused here can be viewed as a kind of relational quantale model according to the terminology of [BrownGurr]. Brown and Gurr discussed the relational quantale model and showed that noncommutative linear logic was complete with respect to their semantics. But their technique heavily relies on representation theorem of quantales and their result is little suggestive for the relationship between linear logic and concurrency theory.

Abramsky and Vickers [AbrVick] discussed an algebraic formulation of concurrency theory. They constructed quantales corresponding to various process preorders and viewed processes as modules over quantales. Compared to their work, ours is more logical rather than algebraic. We hope our formulation is easier to understand and gives intuitive picture. And our formulation is also applicable to bisimilarity, but they were not able to construct the quantale for bisimilarity. On the other hand, they established characterization theorem for process preorders, but our characterization theorem is for the corresponding equivalences.

References

[AbrVick] S. Abramsky and S. Vickers, *Quantales, Observational Logic, and Process Semantics*, Imperial College Research Report DOC 90/1, 1990.

[BFSS] E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott, Functorial polymorphism, *Theoretical Computer Science*, 70:35–64, 1990.

[BrownGurr] C. Brown and D. Gurr, *Relations and Non-commutative Linear Logic*, University of Århus Technical Report DAIMI PB-372, 1991.

[EngWin] U. Engberg and G. Winskel, Petri nets and models of linear logic, in A. Arnold (ed) *CAAP'90*, 147–161, LNCS 431, Springer Verlag, 1990.

[Girard] J-Y. Girard, Linear logic, *Theoretical Computer Science*, 50:1–102, 1987.

[HennMil] M. C. B. Hennessy and R. Milner, Algebraic laws for nondeterminism and concurrency, *J. of ACM*, 32(1):137–161, 1985.

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

Asynchronous Communucation Model Based on Linear Logic (Extended Abstract)

Naoki Kobayashi Akinori Yonezawa
Department of Information Science
The University of Tokyo
7-3-1 Hongo Bunkyo-ku Tokyo, 113 Japan
{koba, yonezawa}@is.s.u-tokyo.ac.jp

7 Introduction

Recently, several applications of Girard's linear logic[Gir87] to logic programming were proposed and shown that they correspond to reactive paradigms[AP91a][AP91b][Mil92]. We propose a new framework called ACL[KY92] for concurrent computation along this line.

Computation in ACL is described in terms of *proof construction* in linear logic. We restrict inference rules and formulas in linear sequent calculus so that restricted rules have a proof power equivalent to the original rules for the restricted formulas(given in the definition 1). The resulting computational framework contains rich mechanisms for concurrent computation, such as message-passing style asynchronous communication, identifier creation, and hiding operator. They are all described in a *pure logical* form. We also give a model-theoretic semantics as a natural extension of *phase semantics*, a model of linear logic. ACL inference rules can be proven to be sound and complete w.r.t. this model-theoretic semantics. Our framework well captures concurrent computation based on asynchronous communication. It will, therefore, provide us a new insight into other models of concurrent computation from a *logical* point of view. In fact, the actor model[Agh86] and asynchronous CCS[Mil83] can be directly translated into our ACL framework. We also expect ACL to become a formal framework for verification, reasoning, and transformation of concurrent programs with techniques used in traditional logic programming. ACL also exhibits attractive features as a programming language.

8 ACL Framework

In this section, we introduce the basic (propositional) fragment of ACL. We give transition rules in a form of restricted inference rules of linear sequent calculus.

8.1 Program Syntax

First, we define the *ACL program clause.*

Definition 1 *A program is a set of clauses, which are defined as follows:*

Clause ::= Head $\circ\!\!-$ Body
Head ::= A_P
Body ::= Statement | Choice
Choice ::= Guarded_Statement| Choice $\oplus$ Guarded_Statement
Guarded_Statement ::= Guard $\otimes$ Statement
Guard ::= $A_m^{\perp}$ | Guard $\otimes$ $A_m^{\perp}$
Statement ::= $\top$ | $\perp$ | A_P | A_m | Body @ Body | Body & Body | $?A_m$
A_P ::= $P, Q, R, \ldots$ (process predicates)
A_m ::= $m, n, \ldots$ (message predicates)

Example. A buffer process with one capacity can be defined in ACL as follows:

$$EmptyBuffer \circ\!\!- put^{\perp} \otimes FullBuffer$$

$$FullBuffer \circ\!\!- get^{\perp} \otimes (reply \mathbin{@} EmptyBuffer)$$

This definition is quite similar to the following description in CCS[Mil89a],

$$EmptyBuffer = put.FullBuffer$$

$$FullBuffer = get.\overline{reply}.EmptyBuffer)$$

though there is a significant difference that communication in ACL is *asynchronous* as is described below, whereas it is synchronous in CCS.

8.2 Operational Semantics

Transition rules are given as a restricted form of inference rules in linear sequent calculus. Please note that the rules should be read that the *conclusion* of an inference rule transits to its *premise formula.* For instance, rule (C2) should be read as $\oplus_j(m_j^{\perp} \otimes A_j), m_i, \Gamma \longrightarrow A_i, \Gamma$.

ACL Inference rules are given as follows:

- Structural Rules

  (S1) $\dfrac{\vdash \Delta}{\vdash \Gamma}$($\Delta$ is a permutation of Γ) $\cdots$ (Exchange)

- Parallel

  (P1) $\dfrac{\vdash A, B, \Gamma}{\vdash A \mathbin{@} B, \Gamma} \cdots$ (parallel)

  (P2) $\dfrac{\vdash A, \Gamma \qquad \vdash B, \Gamma}{\vdash A \mathbin{\&} B, \Gamma} \cdots$ (fork)

  $A \mathbin{@} B$ is an ordinary parallel composition of A and B, whereas $A \mathbin{\&} B$ is a process which copies the entire environment and executes A and B independently.

- Communication Rules

  (C1) $\frac{\vdash m, B, \Gamma}{\vdash m \mathbin{@} B, \Gamma} \cdots$(message send)

  (C2) $\frac{\vdash m_i, m_i^{\perp} \vdash A_i, \Gamma}{\vdash \oplus_j (m_j^{\perp} \otimes A_j), m_i, \Gamma}$(normal message reception)

  (C3) $\frac{\vdash m_i, m_i^{\perp} \vdash A_i, ?m_i, \Gamma}{\vdash \oplus_j (m_j^{\perp} \otimes A_j), ?m_i, \Gamma}$(modal message reception)

- Termination Rules

  (T1) $\frac{}{\vdash \top, A} \cdots$(program termination)

  (T2) $\frac{\vdash A}{\vdash \perp, A} \cdots$(suicide)

- Clause Rule

  (Cl1) $\frac{\vdash B, \Gamma}{\vdash A, \Gamma}$(if $A \circ\!\!- B \in P$)

- Context Rule

  (Co1) $\frac{\vdash C[B], \Gamma}{\vdash C[A], \Gamma}$(if $\frac{\vdash B}{\vdash A}$ is derived from the other rules)

  C[ ], called *positive context*, is defined as follows:

  $$C[\,] ::= [\,] \mid C[\,] \mathbin{@} F \mid F \mathbin{@} C[\,] \mid C[\,] \,\&\, F \mid F \,\&\, C[\,] \mid C[\,] \otimes F \mid F \otimes C[\,] \mid C[\,] \oplus F \mid F \oplus C[\,]$$

  where F is any formula of linear logic.

Rules (C1)-(C3) are rules for communication. $m \mathbin{@} A$ in rule (C1) represents a *sender process* which sends message m. This operation is asynchronous, because $\vdash m, A$ and $\vdash m@A$ are logically equivalent in linear logic. $\oplus_j(m_j^{\perp} \otimes A_j)$ in rule (C2) represents a *receiver process* which waits any one of messages $m_1, \ldots, m_k$ and becomes A_i when receiving m_i. $?m$, which we call a *modal message*, is a message which can be copied unboundly, hence may be used several times by several processes.

The following proposition states that the above inference rules have an equivalent proof power to the original inference rules in linear sequent calculus for the restricted formula.

Proposition 3 *Let P be a program and A be a body formula. $\vdash A$ is provable by the above inference rules if and only if $\vdash ?P^{\perp}, A$ is provable in linear sequent calculus.*

9 Model based on Phase Semantics

In this section, we give a model for the body formulas defined in the previous section by extending the *phase semantics*[Gir87] of linear logic.

9.1 Model for ACL

A set of program clauses is written in the form of

$$< P, Q, R, \ldots > \circ\!\!- \vec{F}(< P, Q, R, \ldots >),$$

where F is a monotonic function on phase space, which is composed of projection, product, and connectives of linear logic $(@, \&, \oplus, \otimes, ?)$.

Given a phase model $(M, \top, m^*)$ where m^* is an assignment of facts to message predicates, we define the model $P^*, Q^*, R^*, \ldots$ of process predicates $P, Q, R, \ldots$ by the following equation:

$$< P^*, Q^*, R^*, \ldots >= \bigoplus_{n \in \omega} (\vec{F^*})^n(\vec{\mathbf{0}^*})$$

We can prove that the ACL inference rules are sound and complete w.r.t. this model. Proofs are given in [KY92].

Proposition 4 (Soundness) *The ACL inference rules are sound w.r.t. extended phase model in the following sense: Let G be a body formula. If G is provable, then G is valid (i.e., $1 \in G^*$) in all the extended phase models.*

Proposition 5 (Completeness) *ACL inference rules are complete w.r.t. the extended phase model in the following sense: Let G be a body formula. If G is valid (i.e., $1 \in G^*$) in all the extended phase models, G is provable by ACL inference rules.*

10 Extensions of ACL

10.1 First Order Extension

First order existential quantification and universal quantification provide mechanisms for value passing and identifier creation respectively.

10.1.1 First-Order Existential Quantification for Value Passing

We introduce first-order existential quantification to the receiver part of a message. Then, communication rules (C2)-(C3) are modified as follows:

$$\textbf{(C2')}\ \frac{(\vdash m_i(a), m_i(a)^\perp) \qquad \vdash A_i(a), \Gamma}{\vdash \oplus_j \exists X(m_j(X)^\perp \otimes A_j(X)), m_i(a), \Gamma}$$

$$\textbf{(C3')}\ \frac{(\vdash m_i(a), m_i(a)^\perp) \qquad \vdash A_i(a), ?m_i(a), \Gamma}{\vdash \oplus_j \exists X(m_j(X)^\perp \otimes A_j(X)), ?m_i(a), \Gamma}$$

The formula $\exists X(m(X)^\perp \otimes P(X))$ represents a process which waits for the values of X via m, and becomes P(a) after receiving message m(a). This extension allows processes to send values in messages.

10.1.2 First-Order Universal Quantification for Identifier Creation

First-order universal quantification works as a mechanism for *identifier creation.* Identifier creation is often very important in concurrent computing environment[Agh86][Mil89b]. Identifiers work as pointers to access resources including processes such that resources can be accessed only by processes which know their pointers. By passing identifiers in messages, acquaintences can be dynamically changed.

Let us look at $\forall$-rule in linear sequent calculus:

$$\frac{\vdash A(X), \Gamma}{\vdash \forall X.A(X), \Gamma} \text{ X not free in } \Gamma$$

We modify this rule as

(ID1) $\dfrac{\vdash A(id), \Gamma}{\vdash \forall X.A(X), \Gamma}$

where id is a unique identifier which does not appear in $A(X)$ and Γ.

10.2 Second-Order Universal Quantification as Hiding Operator

In this section, we introduce second-order universal quantification for message formulas. It works as a *hiding operator* as in CCS. Here is an original rule in linear sequent calculus:

$$\frac{\vdash A, \Gamma}{\vdash \wedge X.A, \Gamma} \text{ X not free in } \Gamma$$

We use the symbol $\wedge$, instead of $\forall$ to distinguish from the first-order universal quantification. Notice the side condition. Quantified variable cannot be free outside the scope of $\wedge$. It is, therefore, invisible from outside. We introduce the following ACL rules instead of the above original rules in linear sequent calculus.

- Hiding Rules

  (H1) $\dfrac{\vdash \wedge m.(A, n), \Gamma}{\vdash \wedge m.(A), n, \Gamma}$ where n contains neither m nor process predicates.

  (H2) $\dfrac{\vdash \wedge m.(A), n, \Gamma}{\vdash \wedge m.(A, n), \Gamma}$

  (H3) $\dfrac{\vdash \Gamma}{\vdash \wedge m.(), \Gamma}$

$C[\,]$ in context rules are also extended to include the form $\wedge m.(C[\,])$. Then, again this extension can be proven to be equivalent to linear sequent calculus.

11 ACL as a Programming Paradigm

ACL has the following attractive features as a progarmming paradigm.

1. Waiting multiple messages
2. Encapsulation mechansim by hiding operator
3. Modal messages for sharing information
4. Dynamic restructuring of processes

Details are given in [KY92].

12 Conclusion

We have proposed a logical framework ACL for concurernt programming languages based on linear logic. We gave the operational semantics of ACL by restricting inference rules in linear sequent calculus and model theoretic semantics by extending phase semantics. In ACL, message passing style communication, identifier creation, and hiding operator are formulated pure logically, hence these mechanisms are uniformly treated by the logical semantics. Future work includes the application of techniques for traditional logic programming to transformation, reasoning and verification of concurrent programs written in ACL. More detailed accounts of our ACL are given in [KY92].

References

[Agh86] Agha, G., *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[AP91a] Andreoli, J.-M. and R. Pareschi, "Linear Objects: Logical processes with built-in inheritance," *New Generation Computing*, 1991.

[AP91b] Andreoli, J.-M. and R. Pareschi, "Communication as Fair Distribution of Knowledge," in *Proceedings of OOPSLA '91*, pp. 212–229, 1991.

[Gir87] Girard, J.-Y., "Linear Logic," *Theoretical Computer Science*, vol. 50, pp. 1–102, 1987.

[KY92] Kobayashi, N. and A. Yonezawa, "Asynchronous Communication Model Based on Linear Logic," tech. rep., Department of Information Science, University of Tokyo, 1992.

[Mil92] Miller, D., "The π-calculus as a theory in linear logic: Preliminary results," Tech. Rep. MS-CIS-92-48, Computer Science Department, University of Pennsylvania, 1992. To appear in the 1992 Workshop on Extensions to Logic Programming, LNAI Series.

[Mil83] Milner, R., "Calculi for Synchrony and Asynchrony," *Theoretical Computer Science*, vol. 25, pp. 267–310, 1983.

[Mil89a] Milner, R., *Communication and Concurrency.* Prentice Hall, 1989.

[Mil89b] Milner, R., J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Part I," Tech. Rep. ECS-LFCS-89-85, University of Edinburgh, 1989.