Appeared in FOCS 1990

Decision Problems for Propositional Linear Logic

Patrick Lincoln* John Mitchell† Andre Scedrov‡ Natarajan Shankar§

Abstract

Linear logic, introduced by Girard, is a refinement of classical logic with a natural, intrinsic accounting of resources. We show that unlike most other propositional (quantifier-free) logics, full propositional linear logic is undecidable. Further, we prove that without the modal *storage* operator, which indicates unboundedness of resources, the decision problem becomes PSPACE-complete. We also establish membership in NP for the multiplicative fragment, NP-completeness for the multiplicative fragment extended with unrestricted weakening, and undecidability for certain fragments of noncommutative propositional linear logic.

1 Introduction

Linear logic, introduced by Girard [14, 18, 17], is a refinement of classical logic which may be derived from a Gentzen-style sequent calculus axiomatization of classical logic in three steps. The resulting sequent system is given in Appendix A, where some standard notation is also defined.

The first step in deriving linear logic from classical logic is to eliminate two structural rules, *contraction* and *weakening*. We view hypothesis as resources, and conclusions as requirements to be met using the resources. Therefore the formula A *implies* A means that the resource A can be use to meet the requirement A. Contraction allows any property which follows from two assumptions of a formula to be derived from a single assumption of that formula. For example the formula (A *and* A) *implies* A is derivable using contraction. Weakening allows deductions which do not use all of their hypotheses, *e.g.*, the formula (A *and* B) *implies* A is derivable with weakening, but not without. Since contraction and weakening make it possible to use an assumption as little or as often as desired, these rules are responsible for what we see in hindsight as a loss of control over resources in both classical and intuitionistic logic. Excluding these rules produces a *linear* system in which each assumption must be used *exactly once*. In the resulting linear logic, formulas indicate bounded or finite resources which cannot necessarily be discarded or duplicated.

The second step in deriving linear logic involves the propositional connectives. Briefly, the change in structural rules leads naturally to two forms of conjunction, one called "multiplicative" and the other "additive", and similarly two forms of disjunction. The multiplicative forms disallow sharing of resources, while the additive forms require resource sharing.

Finally, in order to recover the full deductive power of classical logic, a *storage* or *reuse* operator, !, is added. Intuitively, the formula $!A$ provides unlimited use of the resource A. Using a computational metaphor, the formula $!A$ means 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 ?, definable from ! using negation. The formula $?B$ allows the unlimited consumption of B.

*Lincoln@CS.Stanford.EDU Department of Computer Science, Stanford University, Stanford, CA 94305, and the Computer Science Laboratory, SRI International, Menlo Park, CA 94025. Supported by AT&T Bell Laboratories Doctoral Scholarship, and SRI internal funding.

†JCM@CS.Stanford.EDU Department of Computer Science, Stanford University, Stanford, CA 94305. Supported in part by an NSF PYI Award, matching funds from Digital Equipment Corporation, the Powell Foundation, and Xerox Corporation; NSF grant CCR-8814921 and the Wallace F. and Lucille M. Davis Faculty Scholarship.

‡Andre@Saul.CIS.Upenn.EDU Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104. Partially supported by NSF Grant CCR-87-05596, by ONR Grant NOOO14-88-K-0635 and by the 1987 Young Scientist Award from the Natural Sciences Association of the University of Pennsylvania. Work completed while on sabbatical at the Computer Science Department and Center for the Study of Language and Information, Stanford University, Stanford, CA.

§Shankar@CSL.SRI.COM Computer Science Laboratory, SRI International, Menlo Park, CA 94025. Supported by SRI internal funding.

Since the basic framework remains linear, unbounded reuse or consumption is only allowed "locally", at formulas specifically marked with ! or ? (respectively). The resulting logic is natural from both proof-theoretic and computational standpoints. In particular, Gentzen-style *cut-elimination*, a central property in the proof-theoretic tradition (see [13, 18], for example), has been established for linear logic [14]. Cut elimination establishes consistency and provides a natural computational mechanism that resembles reduction in lambda calculus (*e.g.*, [22, 18]).

An early application of the resource-sensitive aspect of the logic was the implementation of a functional programming language in which garbage collection was replaced by explicit duplication operations based on linear logic [25]. Further studies have demonstrated connections with Petri nets [3, 20, 30, 4, 12, 10] and other models of concurrency [26, 1]. With regard to concurrency, there is a similarity between *proof nets,* the inherent model of computation associated with cut-elimination in multiplicative linear logic (*c.f.* [14, 15, 9, 26]), and *connection graphs,* which were designed to model connection machine computation [5]. Other applications include optimization of copying in lazy functional programming language implementation [21] and the control structure of logic programs [7, 2]. A natural characterization of polynomial time computations can be given in a bounded version of linear logic [19] obtained by limiting reuse to specified bounds, *i.e.*, by bounding the number of references to each datum in memory.

In this paper, we study the complexity of provability for several fragments of propositional (quantifier-free) linear logic. Perhaps our most notable result is that full propositional linear logic is undecidable. However, we begin the description of our results with the smallest fragment considered, the so-called multiplicative linear logic.

The multiplicative fragment contains only linear implication, negation, and forms of conjunction and disjunction which require the available resource to be partitioned among subformulas, rather than shared. We show that the decision problem for this fragment is in NP. Moreover, if unrestricted weakening is allowed, then the multiplicative fragment becomes NP-complete.

There are two natural fragments extending pure multiplicative linear logic. We show that the first extension, with additive and multiplicative connectives but not !, is PSPACE-complete.

We note in passing that the second extension, with only multiplicative connectives and storage (!), is at least as hard as the reachability problem for Petri nets (or, equivalently, commutative semi-Thue systems or vector addition systems). This follows from conservativity properties established in this paper and previous work relating linear logic and Petri nets. Although reachability is decidable [32, 24], the best known lower bound is EXPSPACE [29, 31]. A likely upper bound is primitive recursive in the Ackermann function [33, 8].

Finally, we show that provability in full propositional linear logic, with additive and multiplicative connectives and modal storage operator, is undecidable. (Provability is trivially *r.e.,* since the proof system is effective.) We also establish the undecidability of a noncommutative variant of linear logic (even without additive connectives), a system that extends the calculus in [27], see [16, 34].

In the remainder of this paper we state the requisite lemmas and give a brief overview of the proofs of our theorems. Complete detailed proofs may be found in our technical report [28].

2 Multiplicative Additive Linear Logic is PSPACE-complete

This section deals with the fragment of propositional linear logic, called MALL, which contains the multiplicative connectives, $\otimes$ and $\wp$, the additive connectives, & and $\oplus$, the constants 0, 1, $\top$, and $\perp$, but excludes the modal storage operators ! and ?. The proof rules of MALL are all of the rules in the Appendix A that are associated with these connectives and constants. MALL has been studied by Girard and Bellin, and used by Andreoli, Pareschi, and Cerrito [16, 6, 2, 7]. In contrast to classical propositional logic, which is co-NP-complete, we show below that provability for MALL is PSPACE-complete.

The cut-elimination theorem for the MALL fragment follows from the cut-elimination theorem and the subformula property for linear logic.

Lemma 2.1 *The provability in* MALL *of a given sequent can be decided by a polynomial space bounded Turing machine.*

Proof. There is a linear bound on the depth of cut-free MALL proofs so that an alternating Turing machine could generate and check all branches of a cut-free proof in parallel. ∎

2.1 Informal Outline of PSPACE-hardness of MALL

The PSPACE-hardness of MALL provability is demonstrated by a reduction from the validity problem for quantified Boolean formulas (QBF). In this brief abstract, only the key intuitions are emphasized.

A *quantified Boolean formula* has the (prenex) form $Q_m X_m \ldots Q_1 X_1 \colon M$, where M is a quantifier-free Boolean *matrix* and each Q_i is either $\forall$ or $\exists$ quantifying Boolean variables in M. The syntactic variable G ranges over QBFs, M ranges quantifier-free Boolean formulas, and X ranges over Boolean variables. An *assignment* I for G maps the free variables in G to truth values T or F. The validity of G under I is written as $I \models G$. The QBF validity problem is: *Given a closed QBF G, is $\models G$?*

We provide a succinct encoding of a closed QBF G as a MALL sequent that is provable *exactly* when G is valid. One part encodes the Boolean matrix, and a second part encodes the quantifier prefix. The encoding is carried out in a one-sided formulation of the MALL sequent calculus where, for example, the provable two-sided sequent $A, A \multimap B \vdash B$ is written as $\vdash A^{\perp}, A \otimes B^{\perp}, B$.

2.1.1 Encoding Boolean Evaluation.

The encoding of the Boolean connectives and quantifiers in MALL is described by means of an example. Let G be the *valid* QBF

$$\forall X_2 \exists X_1 \colon \neg(\neg X_1 \wedge X_2) \wedge \neg(\neg X_2 \wedge X_1)$$

G is a restatement of $\forall X_2 \exists X_1 \colon (X_1 \iff X_2)$, so that with the quantifier reversed, $\exists X_1 \forall X_2 \colon (X_1 \iff X_2)$ is falsifiable. The encoding of the Boolean matrix M of G describes the formula as a circuit with signals labeled by MALL literals. If the assignment I is encoded by a sequence of MALL formulas $\langle I \rangle$, and $[M]_a$ is the MALL formula encoding M with output labeled by the literal a, then $I \models M$ is encoded by the sequent $\vdash \langle I \rangle, [M]_a, a$, whereas $I \not\models M$ is encoded by $\vdash \langle I \rangle, [M]_a, a^{\perp}$.

The assignment $\langle \{X_1 \leftarrow T, X_2 \leftarrow F\} \rangle$ is encoded by the sequence of formulas $x_1^{\perp}, x_2$. These literals are the input signals to the encoding of the Boolean formula.

The encoding $[\neg X_1]_a$ of $\neg X_1$ with output labeled a is the formula $\text{NOT}(x_1, a)$ expressing the truth table for negation within MALL.

$$\text{NOT}(x_1, y) = (x_1 \otimes y) \oplus (x_1^{\perp} \otimes y^{\perp}) \tag{1}$$

Which is the negation of the formula $(x_1 \multimap a^{\perp}) \,\&\, (x_1^{\perp} \multimap a)$. The sequent

$$\vdash x_1, \text{NOT}(x_1, a), a \tag{2}$$

encodes the situation where the input X_1 is F, and asserts (correctly) that the output $\neg X_1$ is T. The sequent (2) is easily seen to have the proof

$$\cfrac{\cfrac{\cfrac{}{\vdash x_1, x_1^{\perp}} \qquad \cfrac{}{\vdash a^{\perp}, a}}{\vdash x_1, (x_1^{\perp} \otimes a^{\perp}), a}}{\vdash x_1, (x_1 \otimes a) \oplus (x_1^{\perp} \otimes a^{\perp}), a}$$

Similarly, the sequent (3) representing $\{X_1 \leftarrow T\} \not\models \neg X_1$ is also provable.

$$\vdash x_1^{\perp}, \text{NOT}(x_1, a), a^{\perp} \tag{3}$$

On the other hand, the sequent

$$\vdash x_1^{\perp}, \text{NOT}(x_1, a), a \tag{4}$$

asserts (falsely) that $\{X_1 \leftarrow T\} \models \neg X_1$. Sequent (4) is not provable because MALL is a refinement of classical logic in which no falsifiable sequents of ordinary propositional logic are provable.

The truth tables for the other Boolean connectives can similarly be expressed as MALL formulas and the entire matrix M encoded as above. The only subtlety is that when the fanout of a signal exceeds one, the encoding must provide an explicit means for duplicating the corresponding MALL literal, since MALL lacks a contraction rule.

2.1.2 Encoding Boolean Quantification

To encode Boolean quantification, we need to encode individual quantifiers as well as the dependencies between quantifiers. Given the above encoding for assignments and the Boolean connectives, an almost correct way to encode Boolean quantifiers would be to encode $\exists X_1$ by the formula $(x_1 \oplus x_1^{\perp})$, and $\forall X_2$ by $(x_2 \,\&\, x_2^{\perp})$. The reason the formula $(x_1 \oplus x_1^{\perp})$ behaves like existential quantification in proof search is that a nondeterministic choice can be made between

$$\cfrac{\vdash x_1^{\perp}, \Gamma}{\vdash (x_1 \oplus x_1^{\perp}), \Gamma} \quad \text{and} \quad \cfrac{\vdash x_1, \Gamma}{\vdash (x_1 \oplus x_1^{\perp}), \Gamma}$$

according to the assignment (T or F, respectively) to X_1 which makes $\exists X_1 \colon M$ valid. Similarly, the formula of $(x_2 \,\&\, x_2^{\perp})$ in a sequent behaves like universal quantification requiring proofs of both $\vdash x_2^{\perp}, \Gamma$ and $\vdash x_2, \Gamma$:

$$\cfrac{\vdash x_2^{\perp}, \Gamma \qquad \vdash x_2, \Gamma}{\vdash (x_2 \,\&\, x_2^{\perp}), \Gamma}$$

However, with this representation of quantifiers, the MALL encoding of $\exists X_1 \forall X_2 \colon M$ would be the same as that of $\forall X_2 \exists X_1 \colon M$, but only the latter formula is valid since M expresses $(X_1 \iff X_2)$.

To guarantee that $\exists X_1 \forall X_2 \colon M$ has been correctly encoded, we need to ensure that if the encoding is provable, there is a proof in which the choice of witness for X_1 does not depend on whether X_2 is T or F. Such a dependency of X_1 on X_2 is shown in the proof below.

$$\cfrac{\cfrac{\vdash x_1, x_2, \Gamma}{\vdash (x_1 \oplus x_1^{\perp}), x_2, \Gamma} \qquad \cfrac{\vdash x_1^{\perp}, x_2^{\perp}, \Gamma}{\vdash (x_1 \oplus x_1^{\perp}), x_2^{\perp}, \Gamma}}{\vdash (x_1 \oplus x_1^{\perp}), (x_2 \,\&\, x_2^{\perp}), \Gamma}$$

$$
\dfrac{
  \dfrac{}{\vdash q_2, q_2^{\perp}}\mathbf{I}
  \quad
  \dfrac{\dfrac{\dfrac{?}{\vdash q_1^{\perp}, q_1, x_1}}{\vdash q_1^{\perp}, q_1 \wp x_1}\wp}{\vdash q_1^{\perp}, ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp}))}\oplus
}{\vdash q_2, q_1^{\perp}, q_2^{\perp} \otimes ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp}))}\otimes
\quad
\dfrac{\vdots}{\vdash (q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp}), q_0^{\perp} \otimes [M]_g, g}
$$

$$
\overline{\vdash q_2, q_2^{\perp} \otimes ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp})), q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), q_0^{\perp} \otimes [M]_g, g}\;\otimes
$$

Figure 1: A failed proof attempt

$$
\dfrac{
  \dfrac{}{\vdash q_2, q_2^{\perp}}\mathbf{I}
  \quad
  \dfrac{\dfrac{\vdash q_1, x_1, q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), q_0^{\perp} \otimes [M]_g, g}{\vdash q_1 \wp x_1, q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), q_0^{\perp} \otimes [M]_g, g}\wp}{\vdash ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp})), q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), q_0^{\perp} \otimes [M]_g, g}\oplus
}{\vdash q_2, q_2^{\perp} \otimes ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp})), q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), q_0^{\perp} \otimes [M]_g, g}\otimes
$$

Figure 2: A correct deduction respecting quantifier order

The solution is to encode the quantifier order in a way that forces $(x_1 \oplus x_1^{\perp})$ to be introduced *below* $(x_2 \,\&\, x_2^{\perp})$ in any cut-free proof. For this we introduce new MALL atoms q_0, q_1, q_2, and define the encoding $[\![\exists X_1 \forall X_2 \colon G]\!]_g$ as

$$
\begin{aligned}
\vdash \quad & q_2, \\
& q_2^{\perp} \otimes ((q_1 \wp x_1) \oplus (q_1 \wp x_1^{\perp})), \\
& q_1^{\perp} \otimes ((q_0 \wp x_2) \,\&\, (q_0 \wp x_2^{\perp})), \\
& q_0^{\perp} \otimes [M]_g, g
\end{aligned}
$$

The quantifier encoding for $\exists X_1$ now hides the "key" q_1 that is needed to unlock the quantifier encoding for $\forall X_2$. One attempt to violate the quantifier ordering as before, is shown in Figure 1, where the subgoal $\vdash q_1^{\perp}, q_1, x_1$ is unprovable in MALL due to the absence of an applicable weakening rule.

All other attempts at violating the quantifier ordering also fail, but the deduction which does respect the quantifier order succeeds as shown in Figure 2.

2.2 Proof of PSPACE-hardness of MALL

Lemma 2.2 *Let M be a Boolean formula and I an assignment for the variables in M, then*

1. $I \models M$ *iff* $\vdash \langle I \rangle, [M]_g, g$
2. $I \not\models M$ *iff* $\vdash \langle I \rangle, [M]_g, g^{\perp}$

The next lemma establishes the correctness of the encoding of quantifiers.

Lemma 2.3 *Let M be a Boolean formula in the variables $X_1, \ldots, X_n$, then for any m, $0 \leq m \leq n$, and assignment I for $X_{m+1}, \ldots, X_n$, $I \models Q_m X_m \ldots Q_1 X_1 \colon M$ iff $\vdash q_m, \langle I \rangle, [\![Q_m X_m \ldots Q_1 X_1 \colon M]\!]_g, g$ is provable.*

Lemma 2.4 *Given a closed QBF $Q_n X_n \ldots Q_0 X_0 \colon M$, $\models Q_n X_n \ldots Q_0 X_0 \colon M$ iff $\vdash q_n, [\![Q_n X_n \ldots Q_0 X_0 \colon M]\!]_g, g$ is provable in* MALL.

The size of the sequent encoding a QBF G is polynomial in G and the encoding takes place in polynomial time. Along with Lemmas 2.4 and 2.1 yields the final result.

Theorem 2.5 MALL *provability is* PSPACE*-complete.*

With two-sided sequents, the intuitionistic fragment of MALL constrains the right-hand side of the sequent to contain at most one formula. A two-sided reformulation of the above proof could be carried out entirely within the intuitionistic fragment of MALL so that intuitionistic MALL is also PSPACE-complete.

3 Propositional Linear Logic is Undecidable

We show that propositional linear logic is undecidable by reduction from the halting problem for a form of counter machine. More specifically, we begin by extending linear logic with theories whose axioms may be used any number of times in a proof. We then describe a form of *and*-branching two-counter machines with undecidable halting problem and show how to encode these machines in propositional linear logic with theories. Since the axioms of our theories must have a special form, we are able to show the faithfulness of this encoding using a natural form of cut-elimination with non-logical axioms. To illustrate the encoding of two-counter machines, we present an example simulation of a simple computation in Section 3.5. On first reading, the reader may wish to jump ahead to that section since it demonstrates the basic mechanism used in the undecidability proof.

3.1 Linear Logic Augmented With Theories

We begin by augmenting linear logic with a notion of theory. Essentially, a theory is a set of non-logical axioms (sequents) that may occur as leaves of a proof tree. The theories described here are an extension of earlier work on multiplicative theories [20, 30].

We define a *positive* literal as one of the given p_i propositions, a *negative* literal as one of the $p_i^\perp$ propositions, and an *atomic* formula as any positive or negative literal.

For the theories of interest here, an *axiom* may be any linear logic sequent of the form $\vdash C, p_{i_1}^\perp, p_{i_2}^\perp, ..., p_{i_n}^\perp$, where C is any linear logic formula, and the remainder of the sequent is made up of negative literals. For example, the sequents $\vdash p_1, p_2^\perp$, $\vdash (p_1 \otimes p_2), p_2^\perp$, $\vdash ?(p_1 \wp p_1)$, and $\vdash p_1^\perp, p_2^\perp$ may all be axioms. However, $\vdash p_1, p_1$ and $\vdash (p_1 \otimes p_2), p_3$ may not. We use this restriction on axioms to achieve strict control over the shape of a proof as provided by Lemma 3.1. Some of this control is lost if the definition of theory is generalized, although for some applications the weaker available results would be sufficient.

Any finite set of axioms is a *theory*. We consider only finite theories so that theories may be encoded as formulas of linear logic, which would be impossible if we allowed arbitrary sets of axioms. For any theory T, we say that a sequent $\vdash \Gamma$ is provable in T exactly when we are able to derive $\vdash \Gamma$ using the standard set of linear logic proof rules and axioms from T. Thus each axiom of T is treated as a reusable sequent which may occur as a leaf of a proof tree.

A *directed* cut is one where at least one premise is an axiom and where the cut formula is not a negative literal of the axiom. A cut between two axioms is therefore always directed. A *directed* or *standardized* proof where all the cuts are directed. With these definitions, we may obtain the following result.

Lemma 3.1 (Cut Standardization) *If there is a proof of* $\vdash \Gamma$ *in theory* T*, then there is a directed proof of* $\vdash \Gamma$ *in theory* T*.*

The proof of this lemma follows by induction on the length of proofs. At each step of the induction we appeal to a modified version of the usual cut-elimination procedure.

3.2 Coding Theories in Formulas

We define the translation $[T]$ of a theory T with k axioms into a pure linear logic formula by

$$[\{t_1, t_2, \cdots, t_k\}] \quad = \quad ?[t_1], ?[t_2], \cdots, ?[t_k]$$

where $[t_i]$ is defined for each axiom t_i as follows:

$$[\vdash C, p_a^\perp, p_b^\perp, ..., p_z^\perp\,] \;\triangleq\; (C^\perp \otimes p_a \otimes p_b \otimes \cdots \otimes p_z)$$

Thus each axiom becomes a reusable formula, where the parity of the subformulas of the axiom have been inverted in the formula.

With this translation we are able to achieve the following result.

Lemma 3.2 *For any finite set of axioms* T*, the sequent* $\vdash \Gamma$ *is provable in theory* T *if and only if* $\vdash [T], \Gamma$ *is provable.*

3.3 And-Branching Two Counter Machines Without Zero-Test

We introduce a nondeterministic two counter machine with *and*-branching but without a zero-test instruction. Intuitively, $Q_i\ Fork\ Q_j, Q_k$ is an instruction which allows a machine in state Q_i to continue computation from both states Q_j and Q_k, each computation continuing with the current counter values. For brevity in the following proofs, we emphasize *two* counter machines. More formally, an *And-Branching Two Counter Machine Without Zero-Test*, or ACM for short, is given by a finite set Q of states, a finite set δ of transitions, and distinguished initial and final states, Q_I and Q_F, as follows.

An instantaneous description, or *ID*, of an ACM M is a list of ordered triples $\langle Q_i, A, B\rangle$, where $Q_i \in Q$, and A and B are natural numbers, each corresponding to a *counter* of the machine.

We define the *accepting* triple as $\langle Q_F, 0, 0\rangle$. We also define an *accepting* ID as any ID where every element of the ID is the accepting triple. That is, every and-branch of the computation has reached an accepting triple. We say that an ACM *accepts* input n if and only if there is some computation from initial ID $\{\langle Q_I, n, 0\rangle\}$ to an accepting ID. It is essential for our undecidability result that both counters be zero in all elements of an accepting ID.

The set δ may contain transitions of the following form:

(Q_i Increment A Q_j) taking
$\langle Q_i, A, B\rangle$ to $\langle Q_j, A+1, B\rangle$
(Q_i Increment B Q_j) taking
$\langle Q_i, A, B\rangle$ to $\langle Q_j, A, B+1\rangle$
(Q_i Decrement A Q_j) taking
$\langle Q_i, A+1, B\rangle$ to $\langle Q_j, A, B\rangle$
(Q_i Decrement B Q_j) taking
$\langle Q_i, A, B+1\rangle$ to $\langle Q_j, A, B\rangle$
(Q_i Fork Q_j, Q_k) taking
$\langle Q_i, A, B\rangle$ to $(\langle Q_j, A, B\rangle, \ \langle Q_k, A, B\rangle)$

where the Q_i, Q_j, and Q_k are states in Q. Note that the Decrement instructions only apply if the appropriate counter is not zero.

Thus, for example, the single transition Q_i Increment Q_j takes an ACM from ID: $\{\cdots, \langle Q_i, A, B\rangle, \cdots\}$ to ID: $\{\cdots, \langle Q_j, A+1, B\rangle, \cdots\}$

Since we may simulate a zero-test with and-branching, using the fact that all branches must terminate with counters set to zero, acceptance by an and-branching machines is undecidable.

Lemma 3.3 *It is undecidable whether an and-branching two counter machine without zero-test accepts input* 0*. This remains so if the transition relation of the machine is restricted so that there are no outgoing transitions from the final state.*

3.4 From Machines to Logic

We will write C^n to indicate a sequence of n C's, separated by commas.

We have already seen how the linear connective & may be used to achieve and-branching in the proof of PSPACE-completeness of MALL. We now make use of that, along with some other machinery, to simulate ACM computation.

Given an ACM $M = \langle Q, \delta, Q_I, Q_F\rangle$ we define a linear logic theory from the transition relation δ as follows:

$$\begin{array}{rcl}
Q_i \text{ Increment A } Q_j & \mapsto & \vdash q_i^\perp, (q_j \otimes a) \\
Q_i \text{ Increment B } Q_j & \mapsto & \vdash q_i^\perp, (q_j \otimes b) \\
Q_i \text{ Decrement A } Q_j & \mapsto & \vdash q_i^\perp, a^\perp, q_j \\
Q_i \text{ Decrement B } Q_j & \mapsto & \vdash q_i^\perp, b^\perp, q_j \\
Q_i \text{ Fork } Q_j, Q_k & \mapsto & \vdash q_i^\perp, (q_j \oplus q_k)
\end{array}$$

Using linear implication, the "Q_i Increment B Q_j" transition may be viewed as $\vdash q_i \multimap (q_j \otimes b)$, *i.e.*, from state Q_i, move to state Q_j and add one to B.

Given an element of an ID of an ACM $\langle Q_i, x, y\rangle$, we define a translation θ:

$$\theta(\langle Q_i, x, y\rangle) \stackrel{\Delta}{=} \vdash q_i^\perp, (a^\perp)^x, (b^\perp)^y, q_F$$

Thus all sequents which correspond to elements of ACM ID's have exactly one positive literal, q_F, some number of $a^\perp$s, and $b^\perp$s, the multiplicity of which correspond to the values of the two counters of the ACM, and exactly one other negative literal, which corresponds to the current state of the ACM.

The translation of an ACM ID is simply the set of translations of the elements of the ID. We claim that an ACM M halts from ID if and only if every element of $\theta(ID)$ is provable in the theory corresponding to the transition relation of the machine.

Lemma 3.4 (Machine $\Rightarrow$) *An and-branching counter machine M accepts from ID only if $\theta(ID)$ is provable in the theory derived from M.*

Given a halting computation of an ACM machine M from ID we must build a proof of $\theta(ID)$ in the theory derived from M. This may be accomplished by induction on the length of accepting ACM computation.

Lemma 3.5 (Machine $\Leftarrow$) *An and-branching counter machine M accepts from ID if $\theta(ID)$ is provable in the theory derived from M.*

Given a proof of $\theta(ID)$ in the theory derived from M, we must show that a halting computation of the ACM M from state ID can be extracted from that proof. We achieve this with the aid of cut standardization, Lemma 3.1, which in this case leaves cuts in the proof only where they correspond to applications of ACM instructions. We may thus simply read the description of the computation from the normalized proof. The formal proof of this lemma proceeds by induction on the length of the standardized proof, and depends on the particular encoding of the ACM state.

3.5 Example Computation

This section is intended to give an overview of the mechanisms we have defined above, and lend some insight into our undecidability result, stated below. We present a simple computation of an ordinary two counter machine *with* zero-test instruction, a corresponding ACM computation, and a corresponding linear logic proof. The overall structure is that searching for a proof of a sequent is analogous to searching for an accepting ACM computation.

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

$$\begin{array}{rcl}
\delta_1 & ::= & Q_I \text{ Increment A } Q_2 \\
\delta_2 & ::= & Q_3 \text{ Decrement A } Q_F \\
\delta_3 & ::= & Q_2 \text{ ZeroTest B } Q_3
\end{array}$$

then the machine may perform the following transitions, where an instantaneous description of a two

$$\dfrac{\dfrac{}{\vdash z_B^\perp, a^\perp, z_B}\mathbf{T_4} \qquad \dfrac{\dfrac{}{\vdash z_B^\perp, (q_F \oplus q_F)}\mathbf{T_5} \qquad \dfrac{\dfrac{}{\vdash q_F^\perp, q_F}\mathbf{I} \quad \dfrac{}{\vdash q_F^\perp, q_F}\mathbf{I}}{\vdash (q_F^\perp \,\&\, q_F^\perp), q_F}\&}{\vdash z_B^\perp, q_F}\mathbf{Cut}}{\vdash z_B^\perp, a^\perp, q_F}\mathbf{Cut}$$

Figure 3: Zero-test proof

$$\dfrac{\dfrac{}{\vdash q_I^\perp, (q_2 \otimes a)}\mathbf{T_1} \qquad \dfrac{\dfrac{\dfrac{}{\vdash q_2^\perp, (z_B \oplus q_3)}\mathbf{T_3} \qquad \dfrac{\begin{matrix}\vdots\\ \vdash z_B^\perp, a^\perp, q_F\end{matrix} \qquad \dfrac{\dfrac{}{\vdash q_3^\perp, a^\perp, q_F}\mathbf{T_2} \quad \dfrac{}{\vdash q_F^\perp, q_F}\mathbf{I}}{\vdash q_3^\perp, a^\perp, q_F}\mathbf{Cut}}{\vdash (z_B^\perp \,\&\, q_3^\perp), a^\perp, q_F}\&}{\vdash q_2^\perp, a^\perp, q_F}\mathbf{Cut}}{\vdash (q_2^\perp \wp a^\perp), q_F}\wp}{\vdash q_I^\perp, q_F}\mathbf{Cut}$$

Figure 4: Proof corresponding to computation

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 translated into a transition relation δ' for an and-branching two counter machine *without* zero-test. 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	$\vdash q_I^\perp, (q_2 \otimes a)$	$\mathbf{T}_1$
δ_2'	$::=$	Q_3 Decrement A Q_F	$\vdash q_3^\perp, a^\perp, q_F$	$\mathbf{T}_2$
δ_3'	$::=$	Q_2 Fork Z_B, Q_3	$\vdash q_2^\perp, (z_B \oplus q_3)$	$\mathbf{T}_3$
δ_4'	$::=$	Z_B Decrement A Z_B	$\vdash z_B^\perp, a^\perp, z_B$	$\mathbf{T}_4$
δ_5'	$::=$	Z_B Fork Q_F, Q_F	$\vdash z_B^\perp, (q_F \oplus q_F)$	$\mathbf{T}_5$

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 ZeroTest 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', embody the encoding of that special zero-testing *state*, Z_B, of the ACM. Given the above transitions, the and-branching machine without zero-test may then perform these moves:

$$\begin{aligned}&\{\langle Q_I, 0, 0\rangle\} \xrightarrow{\delta_1'} \{\langle Q_2, 1, 0\rangle\} \xrightarrow{\delta_3'} \{\langle Z_B, 1, 0\rangle, \langle Q_3, 1, 0\rangle\}\\ &\xrightarrow{\delta_4'} \{\langle Z_B, 0, 0\rangle, \langle Q_3, 1, 0\rangle\} \xrightarrow{\delta_5'}\\ &\{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_3, 1, 0\rangle\} \xrightarrow{\delta_2'}\\ &\{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle\}\end{aligned}$$

Note that the instantaneous descriptions 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 essentially 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 3 and 4. In these proofs, each application of a theory axiom ($\mathbf{T}$ rule) corresponds to one step of ACM computation. We represent the values of the ACM counters in unary by copies of the formulas $a^\perp$ and $b^\perp$. In this example the B counter is always zero, so there are no occurrences of $b^\perp$.

The proof shown in Figure 3 of $\vdash z_B^\perp, a^\perp, q_F$ in the above linear logic theory ($\mathbf{T}_1$ through $\mathbf{T}_5$) corresponds to the ACM verifying that the B counter is zero. Reading the proof bottom up, it begins by cutting against the theory axiom $\mathbf{T}_4$, leaving the sequent $\vdash z_B^\perp, q_F$ as an intermediate step. Correlating with the ACM computation, $\mathbf{T}_4$ corresponds to the Decrement A instruction δ'_4, and $\vdash z_B^\perp, q_F$ has exactly one less $a^\perp$ than $\vdash z_B^\perp, a^\perp, q_F$. The next step is to cut against axiom $\mathbf{T}_5$, and after application of the & rule, we have two sequents left to prove — $\vdash q_F^\perp, q_F$ and $\vdash q_F^\perp, q_F$. 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 $b^\perp$, we would be unable to complete the proof.

The proof shown in Figure 4 of $\vdash q_I^\perp, q_F$ in the same theory demonstrates the remainder of the ACM machinery. The lowermost introduction of a theory axiom, $\mathbf{T}_1$, cut, and $\wp$ together correspond to the application of the increment instruction δ'_1. That is, the $q_I^\perp$ has been "traded in" for $q_2^\perp$ along with $a^\perp$. The application of $\mathbf{T}_3$, cut, and & 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 elided proof of $\vdash z_B^\perp, a^\perp, q_F$ appears in Figure 3, and corresponds to the verification that the B counter is zero. The application of $\mathbf{T}_2$, cut, and identity correspond to the final decrement instruction of the computation, and complete the proof.

From the earlier lemmas, our main result is provable.

Theorem 3.6 *The provability problem for propositional linear logic is recursively unsolvable.*

As mentioned earlier, linear logic, like classical logic, has an intuitionistic fragment. Briefly, the intuitionistic fragment is restricted so that there is only one positive formula in any sequent. In fact, the entire construction above was carried out in intuitionistic linear logic, and thus the undecidability result also holds for this logic.

4 Additional Results

The pure multiplicative fragment (without additive connectives or storage operator) is the simplest fragment of linear logic that we have investigated.

Theorem 4.1 *!-free multiplicative linear logic is in* NP.

The proof is straightforward: Each connective in the conclusion sequent is the principle connective in exactly one proof step in any cut-free proof, thus giving a polynomial bound on the size of cut-free proofs. Thus the entire proof may be guessed in polynomial time.

We have been unable to prove this fragment NP-complete. We now believe that this may be very difficult, due to the lack of redundancy in this problem [11]. As part of our investigation of the need to discard arbitrary resources to achieve NP-completeness, we studied propositional logic with weakening, but without contraction, which is sometimes called Direct Logic [23]. This is equivalent to linear logic with the structural modification of adding the weakening rule.

$$\textbf{Weakening}\quad \frac{\vdash \Sigma}{\vdash A, \Sigma}$$

Theorem 4.2 *!-free multiplicative linear logic with weakening is* NP*-Complete.*

Provability in this logic is in NP by the same reasoning as the previous fragment: multiplicative sequent proofs are polynomially short, and may be guessed. Provability is NP-hard by reduction from Vertex Cover. In this reduction, weakening appears essential since an edge may be covered by selecting one endpoint or both.

Finally, we investigate linear logic where we consider sequents to be cyclic *lists* of formula, instead of multisets, and add an explicit exchange rule limited to reusable formulas. We also add a **Rotate** rule which allows formulas to circulate, so that the proof rules in Appendix A are applicable to more than just the end formulas of a sequent. This logic has been called noncommutative linear logic, cyclic linear logic, but we call it circular logic [16, 34].

$$\textbf{Exchange ?}\quad \frac{\vdash \Sigma, \Gamma, ?A}{\vdash \Sigma, ?A, \Gamma}$$

$$\textbf{Rotate}\quad \frac{\vdash \Sigma, A}{\vdash A, \Sigma}$$

Theorem 4.3 *Noncommutative propositional multiplicative linear logic is undecidable, even without additives.*

This result is obtained by reduction from semi-Thue systems. A rule such as "ABC $\rightarrow$ DE" is faithfully represented by the circular logic formula $!(A \otimes B \otimes C \multimap D \otimes E)$, since in this logic $\otimes$ is not commutative. Although the main idea behind this reduction is straightforward, substantial proof-theoretic machinery is required to demonstrate that the reduction is correct. An immediate corollary of this theorem is that full noncommutative propositional linear logic (with additives) is undecidable.

5 Conclusion

We have investigated the complexity of provability for several fragments of propositional linear logic. Our most significant results are that provability for full propositional linear logic is undecidable, but that provability becomes PSPACE complete when the modal storage operator is removed. This gives some interesting insight into the power of reuse when combined with linear propositional connectives. We have also shown that the decision problem for the multiplicative fragment is in NP, and becomes NP-complete in the presence of unrestricted weakening. Finally, we have shown that provability for circular logic (the noncommutative fragment of linear logic without additive connectives) is also undecidable.

A few open problems remain. We have been unable to establish tight bounds for the multiplicative fragment or settle the decidability of the multiplicatives with the storage operator. The latter seems particularly difficult, since a positive solution would involve an extension of the reachability algorithm for Petri nets. The final open problem, which may have logical interest, is the decidability of the !-free fragment, extended with propositional quantifiers.

References

[1] S. Abramsky and S. Vickers. Quantales, observational logic, and process semantics. Preprint, January 1990.

[2] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. In *Proc. 7-th International Conference on Logic Programming, Jerusalem*, May 1990.

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

[4] 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.

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

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

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

[8] P. Clote. On the finite containment problem for Petri nets. *Theoretical Computer Science*, 43:99–105, 1986.

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

[10] 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*, 1990.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

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

[13] G. Gentzen. *Collected Works. Edited by M.E. Szabo.* North-Holland, Amsterdam, 1969.

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

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

[16] J.-Y. Girard. Towards a geometry of interaction. In: Contemporary Math. 92, Amer. Math. Soc., 1989. 69-108.

[17] J.-Y. Girard. La logique linéaire. *Pour La Science, Édition Francaise de Scientific American*, 150:74–85, April 1990.

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

[19] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: A modular approach to polynomial time computability. In *Proc. Math. Sci. Institute Workshop on Feasible Mathematics, Cornell University, June, 1988.* Birkhauser, 1990.

[20] 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.

[21] 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.

[22] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and Lambda Calculus.* London Mathematical Society Student Texts, Cambridge University Press, 1986.

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

[24] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14-th ACM Symp. on Theory of Computing*, pages 267–281, 1982.

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

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

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

[28] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. Technical Report SRI-CSL-90-08, CSL, SRI International, 1990.

[29] R. Lipton. The reachability problem is exponential-space hard. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[30] 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.

[31] E. Mayr and A. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329, 1982.

[32] E.W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. 13-th ACM Symp. on Theory of Computing, Milwaukee*, pages 238–246, 1981.

[33] K. McAloon. Petri nets and large sets. *Theoretical Computer Science*, 32:173–183, 1984.

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

A Propositional Linear Logic Proof Rules

A linear logic sequent is a $\vdash$ followed by a multiset of linear logic formulas. We assume a set of propositions p_i given, along with their associated negations, $p_i^\perp$. Below we give the inference rules for the linear sequent calculus, along with the definition of negation and implication. The reader should note that negation is a defined concept, not an operator.

The following notational conventions are followed throughout this paper:

p_i	Positive propositional literal
$p_i^\perp$	Negative propositional literal
A, B, C	Arbitrary formulas
Σ, Γ, Δ	Arbitrary multisets of formulas

Thus the identity rule (**I** below) is restricted to atomic formulas, although in fact the identity rule for arbitrary formulas ($\vdash A, A^\perp$) is derivable in this system. For notational convenience, it is usually assumed that $\multimap$ and $\otimes$ associate to the right, and that $\otimes$ has higher precedence than $\multimap$. The notation $?\Sigma$ is used to denote a multiset of formulas which all begin with ?. The english names for the rules given below are identity, cut, tensor, par, plus, with, weakening, contraction, dereliction, storage, bottom, one, and top, respectively. Note that there is no rule for the 0 constant.

I $$\frac{}{\vdash p_i, p_i^\perp}$$

Cut $$\frac{\vdash \Sigma, A \qquad \vdash \Gamma, A^\perp}{\vdash \Sigma, \Gamma}$$

$\otimes$ $$\frac{\vdash \Sigma, A \qquad \vdash \Gamma, B}{\vdash \Sigma, \Gamma, (A \otimes B)}$$

$\wp$ $$\frac{\vdash \Sigma, A, B}{\vdash \Sigma, (A \wp B)}$$

$\oplus$ $$\frac{\vdash \Sigma, A}{\vdash \Sigma, (A \oplus B)} \qquad \frac{\vdash \Sigma, B}{\vdash \Sigma, (A \oplus B)}$$

$\&$ $$\frac{\vdash \Sigma, A \qquad \vdash \Sigma, B}{\vdash \Sigma, (A \,\&\, B)}$$

?W $$\frac{\vdash \Sigma}{\vdash \Sigma, ?A}$$

?C $$\frac{\vdash \Sigma, ?A, ?A}{\vdash \Sigma, ?A}$$

?D $$\frac{\vdash \Sigma, A}{\vdash \Sigma, ?A}$$

!S $$\frac{\vdash ?\Sigma, A}{\vdash ?\Sigma, !A}$$

$\perp$ $$\frac{\vdash \Sigma}{\vdash \Sigma, \perp}$$

1 $$\frac{}{\vdash 1}$$

$\top$ $$\frac{}{\vdash \Sigma, \top}$$

Linear negation is defined as follows:

$$\begin{aligned}
(p_i)^\perp &\triangleq p_i^\perp \\
(p_i^\perp)^\perp &\triangleq p_i \\
(A \otimes B)^\perp &\triangleq A^\perp \wp B^\perp \\
(A \wp B)^\perp &\triangleq A^\perp \otimes B^\perp \\
(A \oplus B)^\perp &\triangleq A^\perp \,\&\, B^\perp \\
(A \,\&\, B)^\perp &\triangleq A^\perp \oplus B^\perp \\
(!A)^\perp &\triangleq ?A^\perp \\
(?A)^\perp &\triangleq !A^\perp \\
(1)^\perp &\triangleq \perp \\
(\perp)^\perp &\triangleq 1 \\
(0)^\perp &\triangleq \top \\
(\top)^\perp &\triangleq 0
\end{aligned}$$

Linear implication, $\multimap$, is defined as follows:

$$A \multimap B \triangleq A^\perp \wp B$$