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 \( \text{FSPACE}\)-complete. We also establish membership in \( \text{NP}\) for the multiplicative fragment, \( \text{NP}\)-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 used 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 \((A \text{ and } A) \implies A\) is derivable using contraction. Weakening allows deductions which do not use all of their hypotheses, e.g., the formula \((A \text{ and } B) \implies A\) 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 \(\Gamma\), definable from \(!\) using negation. The formula \(\Gamma B\) allows the unlimited consumption of \(B\).
Since the basic framework remains linear, unbounded reuse or consumption is only allowed “locally”, at formulas specifically marked with ! or $\Gamma$ (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 extensions extending pure multiplicative linear logic. We show that the first extension, with additive and multiplicative connectives but not $\Gamma$, is PSPACE-complete.

We note in passing that the second extension, with only multiplicative connectives and storage ($\Gamma$), 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 $\oplus$, the additive connectives, $\&$ and $\oplus$, the constants $0, 1, \top$, and $\bot$, but excludes the modal storage operators ! and $\Gamma$. 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 $Q_1 x_1 \ldots Q_n x_n M$, 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 $A \land B \vdash B$ is written as $\vdash A \bot, A \circ B \bot, B$.

### 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

$$\forall X_2 \exists X_1: \neg(\neg X_1 \land X_2) \land \neg(\neg X_2 \land X_1)$$

$G$ is a restatement of $\forall X_2 \exists X_1: (X_1 \iff X_2)$, so that with the quantifier reversed, $\exists X_1 \forall X_2: (X_1 \iff X_2)$ 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 $(I)$, 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 $\vdash (I), [M]_{a}, a$, whereas $I \not\models M$ is encoded by $\vdash (I), [M]_{\bot}, a \bot$.

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

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

$$\text{not}(x_1, y) = (x_1 \circ y) \circ (x_1 \circ y \circ) \quad (1)$$

Which is the negation of the formula $(x_1 \circ a) \circ (x_1 \circ a \circ)$. The sequent

$$\vdash x_1, \text{not}(x_1, a), a \quad (2)$$

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

$$\begin{align*}
\vdash x_1, x_1 /\bot &\quad \vdash a /\bot, a \\
\vdash x_1, (x_1 \circ a) \circ (x_1 \circ a \circ), a
\end{align*}$$

Similarly, the sequent (3) representing $\{X_1 \leftarrow T\} \neq \neg X_1$ is also provable.

$$\vdash x_1 /\bot, \text{not}(x_1, a), a \quad (3)$$

On the other hand, the sequent

$$\vdash x_1 /\bot, \text{not}(x_1, a), a$$

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 $(x_1 \circ x_1 /\bot)$, and $\forall X_2$ by $(x_2 \circ x_2 /\bot)$. The reason the formula $(x_1 \circ x_1 /\bot)$ behaves like existential quantification in proof search is that a nondeterministic choice can be made between

$$\vdash x_1 /\bot, \Gamma \quad \text{and} \quad \vdash x_1, \Gamma \quad (4)$$

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

$$\vdash x_2 /\bot, \Gamma \quad \vdash x_2, \Gamma$$

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

To guarantee that $\exists X_1 \forall X_2 M$ 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.

$$\begin{align*}
\vdash x_1, x_1 /\bot, \Gamma &\quad \vdash x_1, x_2, \Gamma \\
\vdash (x_1 \circ x_1 /\bot), x_2, \Gamma &\quad \vdash (x_1 \circ x_1 /\bot), x_2, \Gamma
\end{align*}$$

$$\begin{align*}
\vdash x_1, (x_1 \circ a) \circ (x_1 \circ a \circ), a \\
\vdash x_1, x_2, \Gamma
\end{align*}$$
Propositional Linear Logic is Undecidable

Given a closed QBF assignment for the variables in \( \varphi \), that is needed to unlock the quantifier encoding for \( \forall X \). One attempt to violate the quantifier ordering as before, is shown in Figure 1, where the subgoal \( \vdash q_i, q_1, x_1 \) 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.

![Figure 1: A failed proof attempt](image1)

![Figure 2: A correct deduction respecting quantifier order](image2)

The solution is to encode the quantifier order in a way that forces \( (x_1 \ominus x_1^i) \) to be introduced below \( (x_2 \ominus x_2^i) \) in any cut-free proof. For this we introduce new MALL atoms \( q_i, q_1, q_2 \), and define the encoding \([\exists X_1 \forall X_2 G]_s\) as

\[
\vdash q_2, q^2_i \odot ((q_1 \oslash x_{1^1}) \odot (q_1 \oslash x_{1^1}^i)), q^1_i \odot ((q_1 \oslash x_{1^2}) \odot (q_1 \oslash x_{1^2}^i)), q^0_i \odot [M]_s, g
\]

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 \( \vdash q_i, q_1, x_1 \) 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. \( I \models M \iff \vdash \langle I \rangle, [M]_s, g \)
2. \( I \nmodels M \iff \vdash \langle I \rangle, [M]_s, g \}

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 \( X_{m+1}, \ldots, X_n \), \( I = Q_m X_m \ldots Q_1 X_1 \) iff \( \vdash q_m, \langle I \rangle, [Q_m X_m \ldots Q_1 X_1 M]_s, g \) is provable.

Lemma 2.4 Given a closed QBF \( Q_0 X_0 \ldots Q_0 X_0 \) \( M \), \( \vdash Q_0 X_0 \ldots Q_0 X_0 M \) iff \( \vdash q_n, [Q_n X_n \ldots Q_0 X_0 M]_s, g \) 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 read-
ing, 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
tree. Essentially, a theory is a set of non-logical ax-
oms (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 pi
propositions, a negative literal as one of the pi
propositions, and an atomic formula as any positive or nega-
tive literal.

For the theories of interest here, an axiom may be
any linear logic sequent of the form \( \Gamma \vdash C, p^+_1, p^+_2, \ldots, p^+_n \),
where \( C \) is any linear logic formula, and the remain-
der of the sequent is made up of negative literals. For
example, the sequents \( \Gamma \vdash p_1, p_2, \Gamma(p_1 \otimes p_2), \Gamma(p_1 \oplus p_2), \Gamma(p_1 \wedge p_2) \), and \( \Gamma \vdash p^+_1, p^+_2 \) may all be axioms. However,
\( \Gamma \vdash p_1, p_2, \Gamma(p_1 \otimes p_2) \) may not. We use this restric-
tion 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 re-
sults 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 \( \Gamma \vdash \gamma \) is provable in \( T \) exactly
when we are able to derive \( \Gamma \vdash \gamma \) 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 there-
fore 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 \( \Gamma \vdash \gamma \) in theory \( T \), then there is a directed
proof of \( \Gamma \vdash \gamma \) in theory \( T \).

The proof of this lemma follows by induction on the
length of proofs. At each step of the induction we ap-
peal 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

\[
[t_1, t_2, \ldots, t_k] = [t_1], [t_2], \ldots, [t_k]
\]

where \([t_i]\) is defined for each axiom \( t_i \) as follows:

\[
[\Gamma \vdash p_1, p_2, \ldots, p_n] = (p_1 \otimes p_2) \otimes \cdots \otimes p_n
\]

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 follow-
ing result.

**Lemma 3.2** For any finite set of axioms \( T \), the se-
quent \( \Gamma \vdash \gamma \) is provable in theory \( T \) if and only if \( \Gamma \vdash [T], \gamma \)
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, \( Q_i \) For \( k \) \( Q_j, Q_k \), is an instruction which
allows a machine in state \( Q_i \) to continue computation
from both states \( Q_j \) and \( Q_k \), each computation continu-
ing with the current counter values. For brevity in the
following proofs, we emphasize two counter machines.
More formally, an And-Branching Two Counter Ma-
cine Without Zero-Test, or \( ACM \) for short, is given by
a finite set \( Q \) of states, a finite set \( \delta \) 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 \( (Q_i, A, B) \), 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 \( (Q_F, 0, 0) \). 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 \( \{Q_I, n, 0\} \)
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 \( \delta \) may contain transitions of the following
form:

\[(Q_i \text{ Increment } A \ Q_j) \text{ taking } \{Q_i, A, B\} \text{ to } \{Q_j, A + 1, B\}\]

\[(Q_i \text{ Increment } B \ Q_j) \text{ taking } \{Q_i, A, B\} \text{ to } \{Q_j, A + 1, B\}\]

\[(Q_i \text{ Decrement } A \ Q_j) \text{ taking } \{Q_i, A + 1, B\} \text{ to } \{Q_j, A, B\}\]

\[(Q_i \text{ Decrement } B \ Q_j) \text{ taking } \{Q_i, A, B + 1\} \text{ to } \{Q_j, A, B\}\]

\[(Q_i \text{ Fork } Q_j, Q_k) \text{ taking } \{Q_i, A, B\} \text{ to } \{\langle Q_j, A, B\rangle, \langle Q_k, A, B\rangle\}\]

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 \(Q_i \text{ Increment } Q_j\) takes an ACM from ID: \{\cdots, \{Q_i, A, B\}, \cdots\} to ID: \{\cdots, \{Q_i, A + 1, B\}, \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 \(M = (Q, \delta, Q_F, Q_F)\) we define a linear logic theory from the transition relation \(\delta\) as follows:

\[Q_i \text{ Increment } A \ Q_j \iff \vdash q_i^+, \ (q_j \odot a)\]

\[Q_i \text{ Increment } B \ Q_j \iff \vdash q_i^+, \ (q_j \odot b)\]

\[Q_i \text{ Decrement } A \ Q_j \iff \vdash q_i^+, \ a^+, q_j\]

\[Q_i \text{ Decrement } B \ Q_j \iff \vdash q_i^+, \ b^+, q_j\]

\[Q_i \text{ Fork } Q_j, Q_k \iff \vdash q_i^+, \ (q_j \odot q_k)\]

Using linear implication, the “\(Q_i \text{ Increment } B \ Q_j\)” transition may be viewed as \(\vdash q_i \odot (q_j \odot B)\), 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 \(\theta:\)

\[\theta(\langle Q_i, x, y \rangle) \triangleq \vdash q_i^+, \ (a^+)^\circ, \ (b^+)^\circ, q_F\]

Thus all sequents which correspond to elements of ACM ID’s have exactly one positive literal, \(q_F\), some number of \(a^+\)’s, and \(b^+\)’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 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 \(\delta\) of a standard two counter machine with zero-test consists of the following:

\[\delta_1 \iff Q_1 \text{ Increment } A \ Q_2\]

\[\delta_2 \iff Q_3 \text{ Decrement } A \ Q_F\]

\[\delta_3 \iff Q_2 \text{ ZeroTest } B \ Q_3\]

then the machine may perform the following transitions, where an instantaneous description of a two
counter machine is given by the triple consisting of \( Q_i \),
the current state, and the values of counters \( A \) and \( B \).

\[
\langle Q_I, 0, 0 \rangle \xrightarrow{\delta^1} \langle Q_2, 1, 0 \rangle \xrightarrow{\delta^2} \langle Q_3, 1, 0 \rangle \xrightarrow{\delta^3} \langle Q_F, 0, 0 \rangle
\]

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 \( \delta \) may be translated into a
transition relation \( \delta' \) for an and-branching two counter machine without zero-test. The modified relation \( \delta' \)
(shown on the left below), may then be encoded as a
linear logic theory (shown on the right):

\[
\begin{align*}
\text{Transitions} & : & \text{Theory Axioms} \\
\delta^1_i & ::= & Q_I \text{ Increment } A \ Q_2 & & \vdash q^i_F, (q_2 \& a) & & \mathbf{T}_1 \\
\delta^2_i & ::= & Q_3 \text{ Decrement } A \ Q_F & & \vdash q^i_3, a^+, q_F & & \mathbf{T}_2 \\
\delta^3_i & ::= & Z_B \text{ Fork } Z_B, Q_3 & & \vdash q^i_3, (z_B \& q_3) & & \mathbf{T}_3 \\
\delta^4_i & ::= & Z_B \text{ Decrement } A \ Z_B & & \vdash z_B, a^+, z_B & & \mathbf{T}_4 \\
\delta^5_i & ::= & Z_B \text{ Fork } Q_F, Q_F & & \vdash z_B, (q_F \& q_F) & & \mathbf{T}_5
\end{align*}
\]

Notice how the first two transitions (\( \delta^1_i \) and \( \delta^2_i \)) of the standard two counter machine are preserved in the
translation from \( \delta \) to \( \delta' \). Also, the ZeroTest instruction \( \delta_3 \) is encoded as three ACM transitions — \( \delta^3_3 \), \( \delta^4_3 \),
and \( \delta^5_3 \). The transition \( \delta^3_3 \) is a fork to a special state
\( Z_B \), and one other state, \( Q_3 \). The two extra transitions,
\( \delta^4_3 \) and \( \delta^5_3 \), 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:

\[
\begin{align*}
\langle Q_I, 0, 0 \rangle & \xrightarrow{\delta^1_3} \langle Q_2, 1, 0 \rangle \xrightarrow{\delta^2_3} \langle Q_3, 1, 0 \rangle \xrightarrow{\delta^3_3} \langle Q_F, 0, 0 \rangle \\
\langle Q_F, 0, 0 \rangle & \xrightarrow{\delta^4_3} \langle Q_3, 1, 0 \rangle \xrightarrow{\delta^5_3} \langle Q_F, 0, 0 \rangle
\end{align*}
\]

Figure 3: Zero-test proof

![Figure 3: Zero-test proof](image)

Figure 4: Proof corresponding to computation

![Figure 4: Proof corresponding to computation](image)

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
decrements 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 (T rule) corresponds to one step of ACM computation. We represent the values of the ACM counters in unary by copies of the formulas \( a^+ \) and \( b^- \). In this example the B counter
is always zero, so there are no occurrences of \( b^- \).

The proof shown in Figure 3 of \( \vdash z_B^+, a^+, q_F \) in the
above linear logic theory (T1 through T5) corresponds
to the ACM verifying that the B counter is zero. Reading the proof bottom up, it begins by cutting against the theory axiom $T_1$, leaving the sequent $\vdash z^+_B, q_F$ as an intermediate step. Correlating with the ACM computation, $T_4$ corresponds to the Decrement A instruction $\delta^+_1$, and $\vdash z^+_B, q_F$ has exactly one less $a^+$ than $\vdash z^+_B, a^+ q_F$. The next step is to cut against axiom $T_5$, and after application of the $\&$ rule, we have two sequents left to prove $\vdash q^+_F, q_F$ and $\vdash q^+_F, q_F$. Both of these correspond to the ACM triple $(Q_F, 0, 0)$ 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.

The proof shown in Figure 4 of $\vdash q^+_F, q_F$ in the same theory demonstrates the remainder of the ACM machinery. The lowermost introduction of a theory axiom, $T_1$, cut, and $\varnothing$ together correspond to the application of the increment instruction $\delta^-_1$. That is, the $q^+_F$ has been “traded in” for $q^+_F$ along with $a^+$. The application of $T_3$, cut, and $\&$ correspond to the fork instruction, $\delta^+_3$, 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 $\vdash z^+_B, a^+ q_F$ appears in Figure 3, and corresponds to the verification that the B counter is zero. The application of $T_2$, 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.

\[
\begin{array}{c}
\text{Weakening} \\
\hline
\vdash \Sigma \\
\vdash A, \Sigma
\end{array}
\]

**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 non-commutative linear logic, cyclic linear logic, but we call it circular logic [16, 34].

\[
\begin{array}{c}
\text{Exchange} \\
\hline
\vdash \Sigma, \Gamma, \Gamma A \\
\vdash \Sigma, \Gamma A, \Gamma
\end{array}
\]

\[
\begin{array}{c}
\text{Rotate} \\
\hline
\vdash \Sigma, A \\
\vdash A, \Sigma
\end{array}
\]

**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 → DE” is faithfully represented by the circular logic formula $!(A \odot B \odot C \odot D \odot E)$, since in this logic $\odot$ 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 non-commutative 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 non-commutative 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

A linear logic sequent is a \( \vdash \) followed by a multiset of linear 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
- \( \Sigma, \Gamma, \Delta \) 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 \( \otimes \) and \( \& \) associate to the right, and that \( \otimes \) has higher precedence than \( \& \). The notation \( \Gamma \Sigma \) is used to denote a multiset of formulas which all begin with \( \Gamma \). 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.

### A Propositional Linear Logic Proof Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premises</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>( \vdash p_i, p_i^\perp )</td>
<td>( \vdash \Sigma, A, A^\perp )</td>
</tr>
<tr>
<td>Cut</td>
<td>( \vdash \Sigma, A ) ( \vdash \Gamma, A^\perp )</td>
<td>( \vdash \Sigma, \Gamma )</td>
</tr>
<tr>
<td>( \otimes )</td>
<td>( \vdash \Sigma, A ) ( \vdash \Gamma, B )</td>
<td>( \vdash \Sigma, \Gamma, (A \otimes B) )</td>
</tr>
<tr>
<td>( &amp; )</td>
<td>( \vdash \Sigma, A, B )</td>
<td>( \vdash \Sigma, (A &amp; B) )</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>( \vdash \Sigma, A ) ( \vdash \Sigma, B )</td>
<td>( \vdash \Sigma, (A \oplus B) )</td>
</tr>
<tr>
<td>( \Gamma W )</td>
<td>( \vdash \Sigma )</td>
<td>( \vdash \Sigma, \Gamma A )</td>
</tr>
<tr>
<td>( \Gamma C )</td>
<td>( \vdash \Sigma, \Gamma A, \Gamma A )</td>
<td>( \vdash \Sigma, \Gamma A )</td>
</tr>
<tr>
<td>( \Gamma D )</td>
<td>( \vdash \Sigma, A )</td>
<td>( \vdash \Sigma, \Gamma A )</td>
</tr>
<tr>
<td>( \Gamma S )</td>
<td>( \vdash \Gamma \Sigma, A )</td>
<td>( \vdash \Gamma \Sigma, A )</td>
</tr>
<tr>
<td>( \bot )</td>
<td>( \vdash \Sigma )</td>
<td>( \vdash \Sigma, \bot )</td>
</tr>
<tr>
<td>( 1 )</td>
<td></td>
<td>( \vdash 1 )</td>
</tr>
<tr>
<td>( \top )</td>
<td></td>
<td>( \vdash \Sigma, \top )</td>
</tr>
</tbody>
</table>

Linear negation is defined as follows:

\[
(p_i)^\perp \triangleq p_i^\perp \\
(p_i^\perp)^\perp \triangleq p_i \\
(A \otimes B)^\perp \triangleq A^\perp \& B^\perp \\
(A \& B)^\perp \triangleq A^\perp \& B^\perp \\
(A \oplus B)^\perp \triangleq A^\perp \oplus B^\perp \\
(!A)^\perp \triangleq \Gamma A \perp \\
(\Gamma A)^\perp \triangleq !A \perp \\
(1)^\perp \triangleq \bot \\
(\bot)^\perp \triangleq 1 \\
(0)^\perp \triangleq \top \\
(T)^\perp \triangleq 0 
\]

Linear implication, \( \otimes \), is defined as follows:

\[
A \otimes B \triangleq A^\perp \& B 
\]